Missouri S&T

Academic Assignments

Illya Starikov

Homework, Projects, and Laboratory Reports

Bachelor of Science, Computer Science

Class of 2018

Preface

This collection of academic assignments showcases the practical application of theoretical knowledge gained during my studies at Missouri University of Science and Technology. Through homework assignments, programming projects, and laboratory reports, these materials demonstrate my ability to translate complex concepts into working solutions across diverse areas of computer science and engineering.

Each assignment represents a stepping stone in my technical development, from early programming exercises in introductory courses to sophisticated implementations in advanced algorithms and software engineering. The progression visible in these works illustrates not only mastery of individual topics but also the integration of knowledge across disciplines—a hallmark of the comprehensive education provided at Missouri S&T.

Illya Starikov

Bachelor of Science, Computer Science
Missouri University of Science and Technology

Academic Advisors: Dr. Jennifer Leopold, Dr. A. Ricardo Morales,
Dr. Simone Silvestri, Professor Clayton Price

Class of 2018
starikov@mst.edu

"Miners Dig Deeper"

Table of Contents

Contents

CPE2210

Introduction To Computer Engineering

Lab #1: Laboratory Equipment

Illya Starikov

January 25, 2016

1 Objective

To familiarize self with lab equipment, namely the DC Power Supply, Function Generator, Frequency Counter/Timer, Digital Multimeter, and Mixed Signal Oscilloscope.

2 Equipment Used

- DC Power Supply
- Function Generator
- Frequency Counter/Timer
- Digital Multimeter
- Mixed Signal Oscilloscope

3 Procedures

1. Generate a square wave with an amplitude of 3 V and frequency of 900 Hz.
    - Offset DC by 2 V.
2. Generate 3 signals:
    - 1 kHz
    - 10 kHz
    - 50 kHz

3. Generate 7 V DC, measuring with the Digital Multimeter.
4. Follow the steps of category 5.

4 Conclusion and Results

4.1 Function Generator

4.2 Frequency Counter/Timer

Function Generator	Function Counter
1.017 kHz	1.1 kHz
10.02 kHz	10.1 kHz
49.99 kHz	50.1 kHz

4.3 Digital Multimeter (DMM)

Multimeter:	7.015 V
Power Supply:	7.1 V

4.4 Mixed Signal Oscilloscope

At approximately 2.430 MHz the square wave starts becoming indistinguishable, taking the shape of a sinusoidal wave.

5 Notes And Comments

- $v_{rms} = \frac{VP}{\sqrt{2}}$
- $p - p$ means peak to peak.

Lab # 2

Illya Starikov

February 01, 2016

1 Objective

To familiarize self with Quartus and ModelSim software.

2 Equipment Used

- Computer
  - Quartus Software
  - ModelSim Software

3 Procedures

1. Launch Quartus and go through initial project startup
2. Create a new Block Diagram / Schematic File
3. Add logic gates (as specified)
4. Compile the logic gates.
   - If any compiler errors, fix.
   - Rinse and repeat.
5. Create VHDL from schematic.
6. Launch ModelSim, go through initial setup.
7. Add VHDL to the project.
8. Simulate the schematic built and print the results.

4 Conclusion and Results

ModelSim and Quartus is quite useful software that makes the simulation of circuits a trivial task.

5 Notes And Comments

- Every project must have a file named the same as the project.
- Do **not** save into the C directory.

CPE3150

Micro Embedded Design

Project #2

Michael Schoen, Abdirahman Osman, Illya Starikov

Due Date: November 8th, 2016

For our project, we have decided to implement a memory matching game. The game will start off by enabling a single light. If the user correctly presses said light, the user will move onto the next level and enable a light sequence. For every successive level, an additional light have to be kept in mind (i.e. for the n^{th} level, there will be n lights that will have to be pressed).

This will continue for 15 games, where upon a successful finish, a special light sequence and a song will asynchronously play. From here, the user can choose to play another game if they so choose.

1 Explanation

Below we will take a more in-depth look into the implementation of our project.

1.1 Random Number Generation

Because we want our game to be different every round, we have to have some form of random number generation. Initially, we had tried to implement a *linear congruential generator*; unfortunately, we were unsuccessful. We instead went with a different implementation.

Both implementations require a seed value (and to have different results, we need a different seed value) — we "cheat" to get this. During our input, we not only check to see if the user initiated a game, but we also increment the register the seed is stored in. This way, there is only a $1/255$ chance the user will get the same game[1].

[1] Although this is a significant value (0.39%), it serves well for demo purposes.

Failed Attempt

The basic summary is as follows: beginning with a seed value (named X_n), a new linear, psuedo-random number can be calculated via:

$$X_{n+1} = (aX_n + c) \mod m \tag{1}$$

where the follow conditions hold

1. $a, X_n, c, m \in \mathbb{Z}^+$
2. $0 \leq X_n, c < m$
3. $0 < a < m$
4. $m \neq 0$

The issue we face is it is *highly* recommended that m be quite large (most popular implementations range from 2^{31} to 2^{48}); unfortunately, we only have 2^8 available to us. Because of this, we either get a cycle roughly every 10 iterations or the same number produced. As one might imagine, this is a game-breaking bug; we decided to go with something different.

Our failed attempt for a linear congruential generator is as follows.

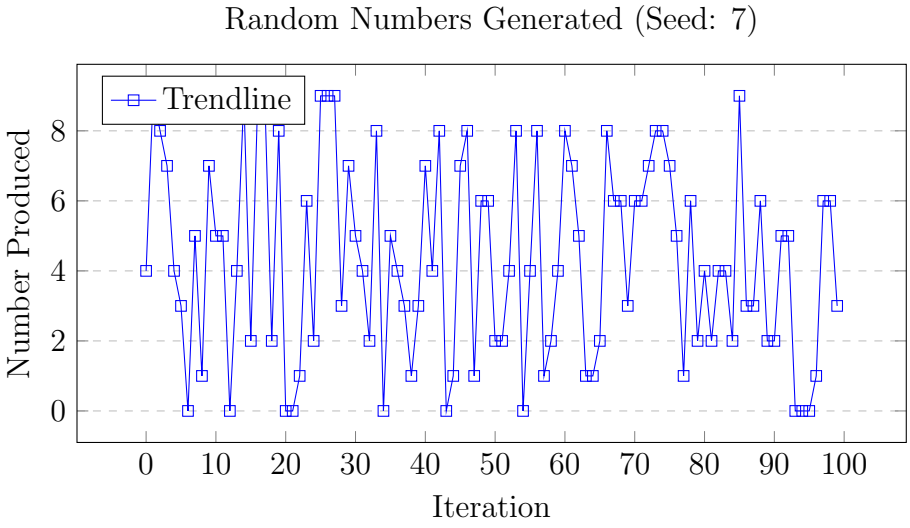
```
; Check writeup for how this works
; For now, the formula is

; X_n+1 = aX_n + c mod m

; For our purposes, a = 7, c = incrementor, m = 72
; for our purposes, incrementor can't be a multiple of 7
; X_n is stored in R7, incrementor in R6

; Result will be stored in A
RNG:
                MOV A, R7
                MOV B, #7D
                DIV AB
                MOV A, B
```

```
                JNZ SKIP1
                INC R6
SKIP1:          MOV A,  #7D
                MOV B, R7
                MUL AB                          ; A = aX_n

                ADD A, R6
                MOV B, #72D
                DIV AB

                MOV R7, B

                MOV A, B
                MOV B, #10D
                DIV AB

                MOV A, B
                RET
```

Successful Attempt

For our successful attempt, we ported a random number generating from an open source code base[2]. It similar to the linear congruential generator in the sense that it linearly produces a psuedo-random number; however, it is a different formula (one that can't be eloquently described in an equation). Below is a graph that shows the distribution of numbers up to 100 iterations.

[2] `https://www.pjrc.com/tech/8051/`



1.2 Light Show

When programming the lights, we notice there to be 9 lights; which just so happens to be the exact size of register A and the carry bit C in the `sfr`. So we decided to map every bit in A and C to a particular light. For simplicity, we used a 1_2 to represent an 'on' state and a 0_2 for the 'off' state.

Our mapping is as follows: we start the MSB of A, which represents the top-left light. The proceeding two bits make up the top-most row of the lighting board. The fourth bit represents the middle row, *first* light. We continue with this pattern until we come to the last light. The last light is represented by the carry bit C. Fig. 1 summarizes this lighting schematic.

As an example, $\boxed{1}\boxed{1}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{1}$ $\boxed{1}$ would enable only the top and bottom lights, leaving the middle row blank. $\boxed{1}\boxed{0}\boxed{1}\boxed{1}\boxed{0}\boxed{1}\boxed{1}\boxed{0}$ $\boxed{1}$ enables the left and right columns. $\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{0}$ $\boxed{1}$ makes an X.

As mentioned, to represented an 'on' state, we used a 1_2; however, the lights are active low. So, the first thing we do is `CPL A` and `CPL C` to ensure we don't have opposite affect. From there, we rotate through C to enable individual light. Finally, we `CPL` A and C again to bring the data back to its original state.

The individual lighting effects were done through trial and error of what looked aesthetically pleasing. There are three light patterns, one for losing, advancing to the next level, and the winning light sequence. The winning light sequence has five parts, eloquently named:

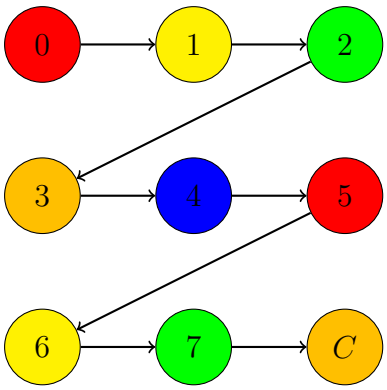


Figure 1 – The lighting schematic for "lights" subroutine.

1. Light Spiral Part I
2. Light Spiral Part II
3. Full Light Spiral
4. Os
5. Bring the Thundaaa

1.3 Music

We know the frequency of the Philips P89LPC932A1 to be 7.373 MHz, with 2 cycles per machine cycle. Therefore,

$$\frac{2 \text{ cycles}}{\text{machine cycle}} \cdot \frac{1 \text{ Period}}{7.373\,\text{MHz}} = 0.271\,26\,{}^{\mu s}/_{\text{mc}} \tag{2}$$

We use this calculation as the base of our music.

E5	$f = 659.255\,\text{Hz} \implies T = 1516\,\mu\text{s}$ $1516\,\mu\text{s} \div 0.271\,26\,{}^{\mu\text{s}}/_{\text{mc}} = 5589\,\text{mc}$ $5589\,\text{mc} \div 4 = 1398\,\text{mc}$ $1398\,\text{mc} \implies 699$ iterations (with `DJNZ`)
F5	$f = 698.456\,\text{Hz} \implies T = 1431\,\mu\text{s}$ $1431\,\mu\text{s} \div 0.271\,26\,{}^{\mu\text{s}}/_{\text{mc}} = 5275\,\text{mc}$ $5589\,\text{mc} \div 4 = 1318\,\text{mc}$ $1318\,\text{mc} \implies 569$ iterations (with `DJNZ`)
G5	$f = 783.991\,\text{Hz} \implies T = 1275.5\,\mu\text{s}$ $1275.5\,\mu\text{s} \div 0.271\,26\,{}^{\mu\text{s}}/_{\text{mc}} = 4702\,\text{mc}$ $4702\,\text{mc} \div 4 = 1176\,\text{mc}$ $1176\,\text{mc} \implies 588$ iterations (with `DJNZ`)
D5	$f = 587.330\,\text{Hz} \implies T = 1702.6\,\mu\text{s}$ $1702.6\,\mu\text{s} \div 0.271\,26\,{}^{\mu\text{s}}/_{\text{mc}} = 6277\,\text{mc}$ $7045\,\text{mc} \div 4 = 1570\,\text{mc}$ $1570\,\text{mc} \implies 785$ iterations (with `DJNZ`)
C5	$f = 523.251\,\text{Hz} \implies T = 1911.1\,\mu\text{s}$ $1911.1\,\mu\text{s} \div 0.271\,26\,{}^{\mu\text{s}}/_{\text{mc}} = 7045\,\text{mc}$ $7045\,\text{mc} \div 4 = 1760\,\text{mc}$ $1760\,\text{mc} \implies 880$ iterations (with `DJNZ`)
Flat D5	$f = 554.365\,\text{Hz} \implies T = 1803.8\,\mu\text{s}$ $1803.8\,\mu\text{s} \div 0.271\,26\,{}^{\mu\text{s}}/_{\text{mc}} = 6650\,\text{mc}$ $6650\,\text{mc} \div 4 = 1662\,\text{mc}$ $1662\,\text{mc} \implies 831$ iterations (with `DJNZ`)

2 Future Work

3 Work Effort

- Michael Schoen
  - Programmed binary counter.

  - Programmed game logic.

- Osman Abdirahman
  - Programmed initial beep.
  - Programmed song implementation.

- Illya Starikov
  - Programmed random number renerator.
  - Programmed light sequence.

Project #3

Michael Schoen, Abdirahman Osman, Illya Starikov

Due Date: Tuesday, December 6th, 2016

For the second project, we were delighted to be able to use a higher level programming language; we[1] decided to apply this new-found excitement to implement a retro game from the 70s: Space Invaders.

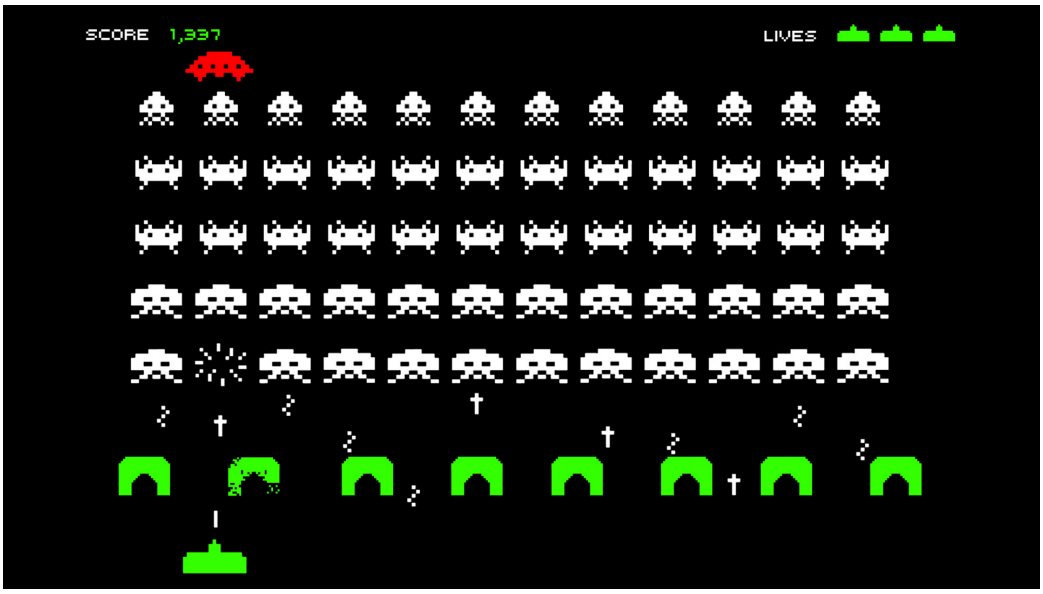


Figure 1 – The original space invaders.

When this idea came crashing and burning down to the ground, we decided to combat this with a additional music, an elegant user interface, and a keyboard (with some other miscellanies).

[1] Illya.

1 Project Description

Below we will go into more detail about each individual parts of our project.

1.1 The Game

The game we initially decided to go with was space invaders. We had intention of doing the game logic on the microcontroller through serial communication; however, we learned early that this was likely not be possible[2]. We describe in *Problems Encountered* how we absolved this. From here, we decided our game would exclusively be in the terminal.

Our game essentially creates a two dimensional array (in three segments — the header to show score and level, the aliens, and the shooter). Then we loop through depending on the input:

- ← Move the shooter left.
- → Move the shooter left.
- q Quits the game
- Space Shoots with the gunner.
- No input meant refresh game.

We achieved the drawing through `curses`[3]. Ultimately, we did not get our game fully done, but a good majority of it. See Figure. 2 for a look at the UI of the game.

1.2 Music

The formula for each note is

$$\frac{1/4 \,\text{Oscillator Frequency}}{\text{Note Frequency}}$$

So for supposing we want to produce the note $C4$,

[2]Our hex file with very basic functionality was 25 kB.

[3]Can be read about here `https://en.wikipedia.org/wiki/Curses_(programming_library)`

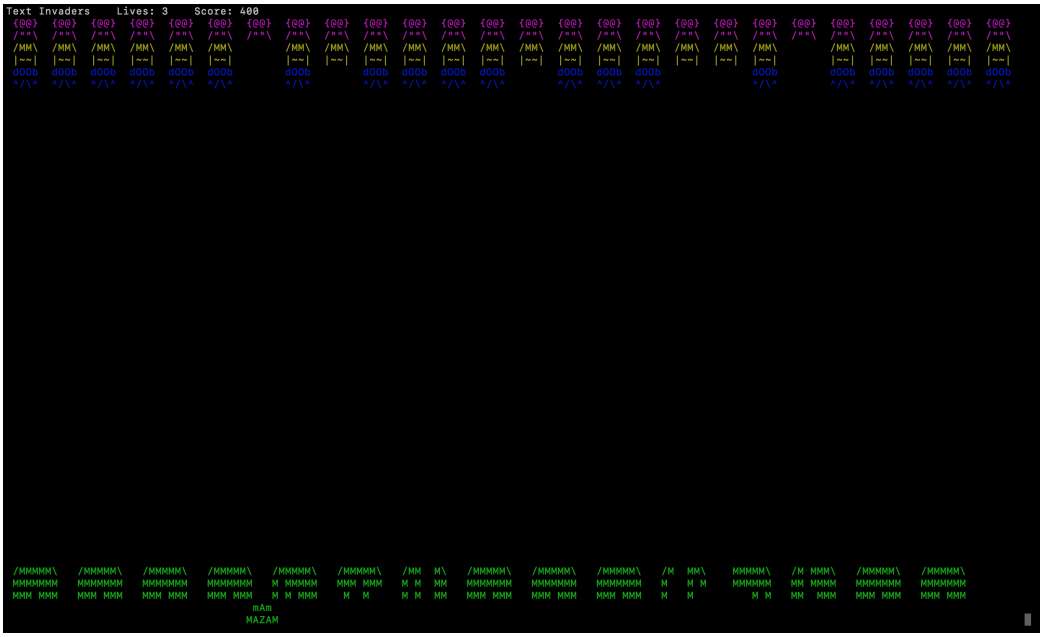


Figure 2 – Text Invaders, our version of space invaders..

$$\frac{1/4 \,\text{Oscillator Frequency}}{\text{Note Frequency}} = \frac{1/4 \cdot 7.373\,\text{Hz}}{261.63\,\text{Hz}} = 7.045\,\text{ms} \tag{1}$$

To produce the music, we had a function that took in which note and the type of note (16^{th}, 8^{th}, 4^{th}, half or whole note). The function multiplied the base length of a note by a constant depending on the type of note. A 16^{th} note would be multiplied by 1, an 8^{th} by 2, a quarter note by 4, a half-note by 8, and a whole note by 16. We used timers and interrupts for the period of the note and the length of the note.

We also used sheet music to get the notes. For speeding and slowing down the music, the type of note was just multiplied by 2 or divided by 2 respectively.

1.3 Interactive User Interface

The menu goes through three stages; printing the pen-rose triangle background, printing options, then allowing the selection of items. The following are printed

1. Counter

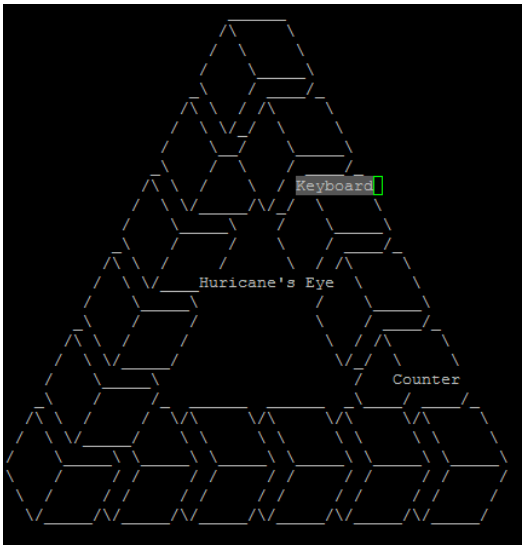


Figure 3 – The pen-rose style menu.

2. Hurricane's Eye
3. Keyboard

respectively. From there the user has the option to to cycle upwards using P_2 (see Figure. 4), cycle down using P_8, or select using P_5. The cycle starts with 1, highlighting Counter if P_2 is pressed, then Hurricane's eye will be highlighted, and if printed once again Counter will be selected. If the menu user presses P_1 once at the 3rd option (Keyboard) then the menu will cycle back to option 1 and vice versa if P_8 were clicked at the 1st option. Selecting an menu option with P_5 while highlighted will start its respective function. Also each menu option has a corresponding number displayed on a seven segment display.

1.4 Seven Segment Display

For the seven segment display, we had to set P_2 to be push/pull so that the display would receive enough power. We then looked up the correct values to output to the pins to make each number show up on the display and used those values in the increment/decremented function as well as the simple function that lets you display any number. We used the data sheet for the

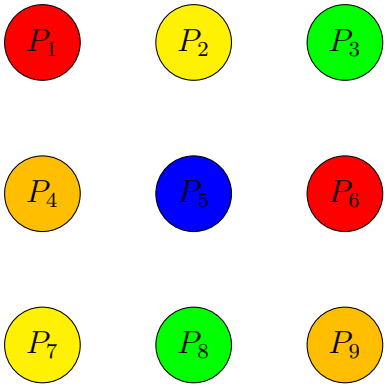


Figure 4 – The Main Menu.

seven segment display to see how to connect it on the breadboard. See Figure. 5 for additional information.

1.5 Keyboard

Our keyboard has a C scale set to switches $P_1 \cdots P_8$ with P_9 escaping to the menu. Each switch uses the corresponding light associated with its position save P_9 which is the escape button.

1.6 Curses

In order to print to any part of the screen we needed to implement a structure to send ANSI control sequences and characters to the terminal using `uart_get()` as a base. To do this we implemented an 8051 port of curses that takes in (y, x) coordinates as arguments for printing `chars` and `strings` at any defined (x, y) location. This allows for the implementation of the menu since it requires printing of options in sometimes nonlinear fashion.

We found a whole host of others issues associated with printing characters to a terminal, printing to an arbitrary location, control sequences, fluctuation in crystal timings, and requesting information from a terminal. We accounted for these individually.

PIN NO.			
1	Cathode E	6	Cathode B
2	Cathode D	7	Cathode A
3	Common Anode DIG. 1	8	Common Anode DIG. 2
4	Cathode C	9	Cathode F
5	Cathode DP	10	Cathode G

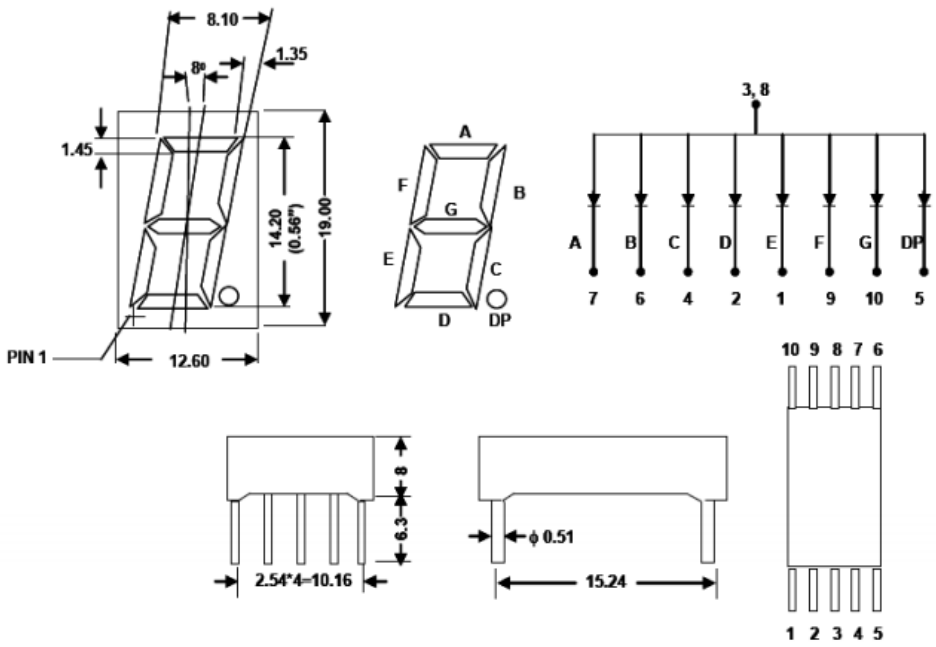


Figure 5 – The schematic for the seven segment display.

2 Problems Encountered

Below we will list some of the "several" problems we ran into.

2.1 Memory Constraints

By far, the biggest problem we encountered was the memory constraint. The 8051 has 8 kB of memory; our code base, with the exclusion of all the `malloc`s[4], it is roughly 25 kB — a size a bit larger than the 8051 allows. Our workaround was to fight fire with fire.

Instead of "dumbing" down our game[5] to get it to fit, we decided to have an external interface; specifically, a port sniffer. It would listen for input from the 8051, and if there is a signal on the serial port, use that as input. If not, default to the keyboard input.

Ultimately, we were unsuccessful with the port sniffer. Originally, we had tried to use a Linux port sniffer so we can just embed it into our program, `slsnif`[6]. Unfortunately this port sniffer does not support "legacy" ports, and the 8051 falls in this category. So we moved onto a Windows port sniffer, `Serial Input For Windows`[7]. This too did not work, because we could only have one interface use the serial port, so we would need a dedicated socket to intercept the `COM1` port's input — something we were not familiar with.

Ultimately, we were simply unsuccessful and had to scrap this.

2.2 Printing Lines and Columns.

We had an issue with the timing crystals of the Simon board where the board heating up would cause the serial port to start printing unrecognizable characters. This lead us to think we were causing a segfault when we were printing string literals. This caused hours of wasted time since nothing we could change would cause the board to start printing what we wanted, but one afternoon after hours of exhaustive research and help at the lead sessions the conclusion was reached that heat changes were causing the board to lose its timing and print out nonsense. The fix for this was to let the board cool off while writing code. After this change garbage outputs dropped significantly.

[4]Since `unsigned char` = 1 B, the terminal window will roughly be 20 height × 80 width, we can roughly expect 1.6 kB to be allocated on the heap; a non-insignificant amount compared to 8 kB.

[5]According to back of the hand calculations, a space-optimized version of the game would still be 7 kB. This was likely to be impossible.

[6]Can be found at `https://sourceforge.net/projects/slsnif/`.

[7]Can be found at `http://www.randomnoun.com/wp/2013/02/03/serial-input-for-windows/`.

2.3 Seven Segment Display Buttons

One problem we ran into was some of the buttons use the same pins as we used for the seven segment display. The display used push/pull which makes the buttons not work, so we had to carefully chose buttons that didn't use the same pins.

3 8051 Architecture

For this project, we made sure to utilize the additional functionality we learned in the later parts of the semester — specifically, timers, serialization, and interrupts.

We spent a great deal of time with the serial communication aspect of the board; we took advantage of this the most (with respect to the 8051 architecture). We have already discussed the things that made our board unique; however, we will show an example of how our implimination of `curses`.

```
1  void move(unsigned char y, unsigned char x) {
2      unsigned char i; // counter
3      zero_cursor(); // moves the code to (0, 0)
4
5      for(i = 0; i < x; i++) {
6          cursor_jump(1, 'R'); // move cursor to the x position
7      }
8      for(i=0; i < y; i++) {
9          cursor_jump(1, 'D'); // now to the y
10     }
11 }
```

Now combining that with `addch(char)`,

```
1  void addch(unsigned char value) {
2      uart_isr();
3      uart_transmit(value);
4      while(TI==0);
5  }
```

we now have a fundamental idea to `curses`, printing anywhere!

Next, we implemented all delays with timer interrupts. For example,

```
1
2  void delay1ms() {
3      TH0=-0x0E;
```

```
4       TL0=-0x66;
5       TR0 = 1; // start
6       while(TF0 == 0); // poll to finish
7       TR0 = 0; // stop
8       TF0 = 0; // clear the finish
9   }
```

delays for 1 ms, because $(\text{FFFF}_{16} - 0\text{E66}_{16} + 1) \times 1.085\,\mu\text{s} \approx 1\,\text{ms}$. As it turns out, we mostly needed millisecond precision, so we used this as the foundation for the rest of the program.

4 Individual Features

- Michael Schoen — 33% Contribution
  - Song
  - Seven Segment Display
- Abdirahman Osman — 33% Contribution
  - Port Serialization
  - Text User Interface
  - Menu
- Illya Starikov — 33% Contribution
  - Space Invaders Game
  - Keyboard

Happy Holidays
From Michael, Abdirahman, and Illya!

CS1001

Data Structures Lab

Lab #1

Illya Starikov

July 27, 2025

1 Output Redirection

1. foo
2. bar. I conclude that it is echoing the variable (foo and bar) into the file.txt.
3. `bar`
   `baz`
4. »is prepending a newline.

2 ls and Friends

1. Outputs all directories (hidden and public) as with the subdirectories. * matches all directories and subdirectories.
2. It outputs all super directories, .* matches anything with that in the name.
3. It will annihilate all super directories, including your home directory.

3 Intermediate man usage

1. Shows the manual for the passwd file.
2. *man 5 passwd*, a lot of tinkering.
3. $Home, $SHELL, $PATH, $LOGNAME, $Mail

4 A series of tubes

1. Done.
2. cat My Cow Story | foo (The Build File)
3. Bash error.

5 Submitting your work

1. Done.
2. `zip -r Lab\ #2 Lab\ #2/`
3. Done.

1 That filter problem, AGAIN

1. `grep \# -v story-plain.txt`
2. `grep -v "[\#]" story-space.txt`

2 Lk m, n vwls!

1. `sed s/[aeuio]//gi story-plain.txt`

3 Counting Lines

1. `grep -P "\d{4}-01" numbers.log | wc -l`

4 Phone numbers

1. `grep -Po "[(]?\d{3}[)\s-]?\d{3}[\s-]?\d{4}" phonebook.txt`

Debuggers

Illya Starikov

March 14, 2016

1 Segfaults

1. `List::append`
2. Passed a `nullptr`. It's fixed!

2 Loopy

1. `Print`
2. They repeat. It's a looped List!
3. Fixed.

3 Math is hard

1. Fixed!

Lab #9

Illya Starikov

March 21, 2016

1 Drip, Drip, Drip, Drip

1. Memory leak, `indirectly lost: 80 bytes in 5 blocks` and `definitely lost: 16 bytes in 1 blocks`
2. Because a `pointer` to a list is being created, but not freed.
3. Done!

2 Imagine the possibilities!

1. `possibly lost, still reachable, suppressed`.
2. I was.. I knew it.
3. There is a destructor in `dtor` and `plain` only has the default (which does not take care of manual memory allocation).

3 Use wisely

1. Memory Leak.
2. You have a memory address assigned to an memory address.
3. Done.

Project #1

Illya Starikov

Due Date: March 25, 2016

0 Introduction

- Install 2 editor plugins
- Document 5 editor features (with vim, this'll be a cinch)
- Do the job control section of project B (I use this all the time)
- What does the -exec option on find do?
- What does xargs do?
- Look up and use tmux.

1 Two Editor Plugins

For two text editors plugins, I chose (facetiously) pathogen and powerline.

1.1 pathogen.vim

Naturally when I knew I was going to install packages, my first thought was to use a package manager. Easily manipulating and deleting new packages was going to be critical, so Pathogen looked to be the proper choice.

1.1.1 What It Does

In its simplest terms, pathogen makes package management as easy as `rm -r`. My main use has been mostly to easily locate and delete new plugins that I do not use in my `bundle` directory.

1.1.2 How To Use It

After initial setup, pathogen actually does the rest.

1.2 Powerline

While on my quest to find text editor plugins, I stumbled upon and installed powerline. While trying to think of my second plugin to write in this report, I couldn't help but notice how much I was using Powerline.

1.2.1 What It Does

Powerline is an alternative to the standard Vim statusline.

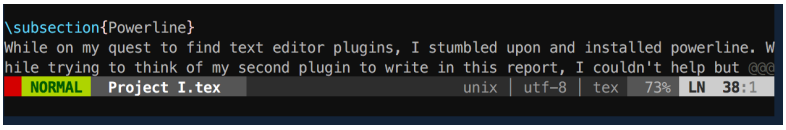


Figure 1: The Powerline In Action

It changes the configuration based on the mode that you are.

1.2.2 How To Use It

Use of powerline is automatic.

2 Five Editor Features

For my five editor features, I decided to do:

External commands (r ! `Command`) To run an external command outside of vim, and pipe the input back into the file. Makes inserting the date into LaTeXdocuments much, much faster.

Searching (/ ?) Command F for Vim.

Go to line (`:number`) Jumping to a specified line number.

Autocomplete (control+n) Attempts to finish the work being typed (whether it be a command or variable)

Text folding (z+f)a

3 Job Control

1. It terminates.
2. It terminates the command from the background.
3. Waits until all previous processes have finished until continueing.

4 Find `exec`ution

According to `Man`,

```
-exec utility [argument ...] ;
      True if the program named utility returns a zero value as its exit
      status.  Optional arguments may be passed to the utility.  The
      expression must be terminated by a semicolon (``;'').  If you
      invoke find from a shell you may need to quote the semicolon if the
      shell would otherwise treat it as a control operator.  If the
      string ``{}'' appears anywhere in the utility name or the arguments
      it is replaced by the pathname of the current file.  Utility will
      be executed from the directory from which find was executed.
      Utility and arguments are not subject to the further expansion of
      shell patterns and constructs.
```

Essentially `exec` allows the `exec`ution of certain programs during a command. For example, to make the use relevant to `find`, searching for an instance of a keyword in open files would `find . -exec grep KEYWORD +`.

5 *T*erminal *Mu*ltiple*x*or

Finished, used for the job controls section (SSH to tmux to logout).

6 Xargs

Back to `Man`,

```
The xargs utility reads space, tab, newline and end-of-file delimited
strings from the standard input and executes utility with the strings as
arguments.

Any arguments specified on the command line are given to utility upon each
invocation, followed by some number of the arguments read from the standard
input of xargs.  The utility is repeatedly executed until standard input is
exhausted.

Spaces, tabs and newlines may be embedded in arguments using single
(`` ' '') or double (``"'') quotes or backslashes (``\'').  Single quotes
escape all non-single quote characters, excluding newlines, up to the
matching single quote.  Double quotes escape all non-double quote charac-
ters, excluding newlines, up to the matching double quote.  Any single
character, including newlines, may be escaped by a backslash.
```

`xargs` allow for the manipulation of standard input, becoming quite useful with commands such as `find` or `ls`.

7 Conclusion

In conclusion, I have gained more familiarity with the command line. To take the assignment another step up, the entirety of this assignment has been created using the command line — Vim, `pdflatex`, `rm` for removing of LaTeXfiles, etc.

CS1200

Discrete Math

COMP SCI 1200 Section B FS2015 Assignment 4

Illya Starikov

November 8, 2015

To compile: Run a standard makefile:

```
fg++ *.cpp -o foo
```

Notes:

- Any two int are acceptable (within the scope of the primitive int).
- Negative integers are also accepted, but they will be converted to positive.

COMP SCI 1200 Section B FS2015 Assignment 5

Illya Starikov

November 17, 2015

To compile: Run a standard makefile:

```
fg++ *.cpp -o foo
```

Notes:

- Any two non-negative int are acceptable (within the size of the primitive int).

CS2300

Databases

Homework #1

Claire Trebing, Jason Young, Illya Starikov

Due Date: February 04, 2016

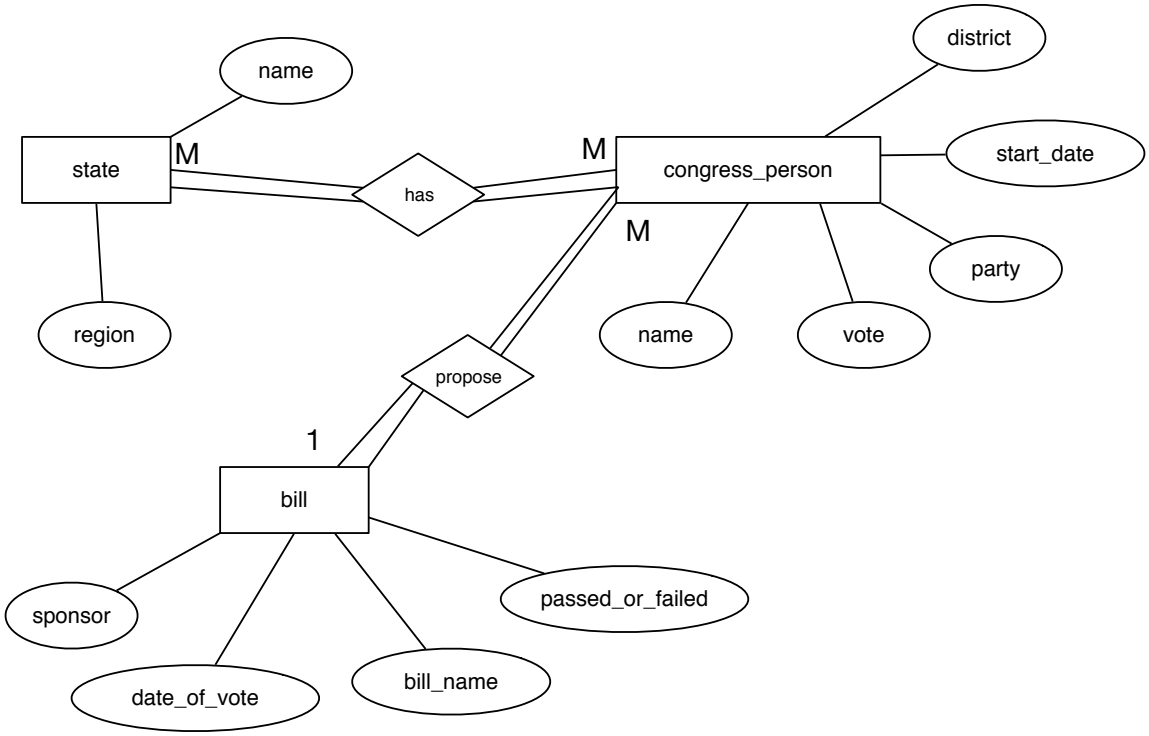
Question 1

a Section's CRoom (Building and Room Number).

b Section's DayTime and Year.

c Section's SecID for every Year.

Question 2


district
name
state
M
has
M
congress_person
start_date
M
party
region
name
vote
propose
1
bill
sponsor
passed_or_failed
date_of_vote
bill_name

Question 3

a Strong Entity Types

- Bank
- Accounts
- Loans
- Customer

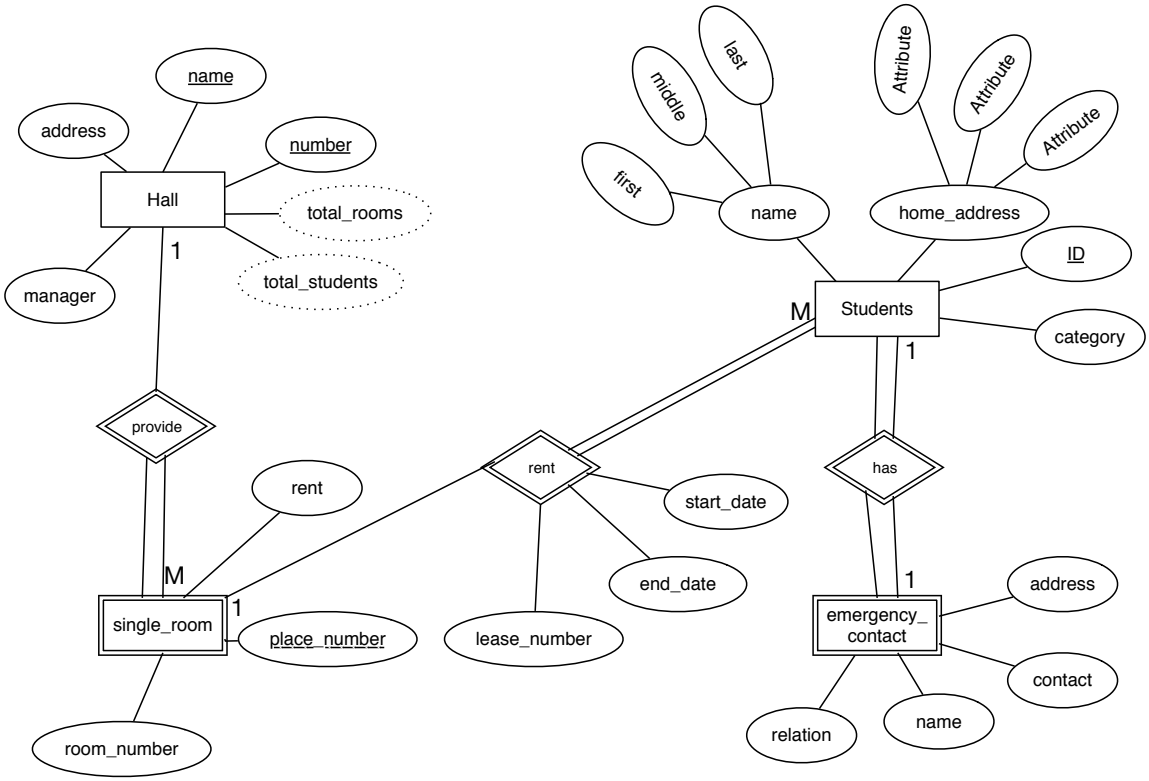
b Weak Entity Type

- Yes, the **Bank Branch** is a weak entity type. The partial key is the **Branch Number** and the identifying relationship is the **Branches** relationship.

c Constraints

- This information specifies that the branch number of a bank branch is only unique within a bank. For example, a bank branch in Bank A may have the same branch number as a bank branch in Bank B, but a bank branch in Bank A would not have the same branch number as any other bank branch in Bank A.

Question 4


name
address
number
Hall
total_rooms
manager
1
total_students
provide
rent
M
1
single_room
place_number
room_number
middle
last
first
name
Attribute
Attribute
Attribute
home_address
ID
M
Students
category
1
rent
start_date
end_date
lease_number
has
1
address
emergency_
contact
contact
relation
name

Homework #2

Claire Trebing, Jason Young, Illya Starikov

February 25, 2016

Question #1

a. Valid Operation

b. Valid Operation

c. Key Constraint (Key #4 is already taken)

d. Key Constraint, Not Null Constraint (Key cannot be null)

e. Valid Operation

f. Valid Operation

g. Integrity Constraint (Did not update `WORKS_ON` or `DEPENDANT`)

h. Integrity Constraint (Did not update `WORK_ON`)

i. Valid Operation

j. Integrity Constraint (Did not update `WORKS_ON`)

k. Valid Operation

Question #2

Grader note: the rho (ρ) is too small in LaTeXto be distinguished from a subscript, so instead P is used in it's place.

a. $\Pi_{\text{Fname, Minit, Lname}}(\sigma_{\text{Dno = '5'} \wedge \text{hours = >10}}$(`roject \end{verb} \bowtie _{\text{Dname = ‘`$roject X'$`or`

b. $\Pi_{\text{Fname = Dependent\_Name}}$(`mployee \end{verb} \bowtie \begin{verb} Dependent \end{verb})$`

c. $\Pi_{\text{Fname, LName}}$(`mployee \end{verb} \div \Pi_{\text{Super\_SSN}}(\sigma _{\text{Fname = ‘Frankli`

d. $\Pi_{\text{Lname, Sum\_Hours}}(s_{\text{name}}\mathcal{F}_{\text{sum hours}}$(`ROJECT \end{verb} \bowtie \rho_{\text{ESSN = SSN}} \begin{verb}`

e. `mployee1 \end{verb} \leftarrow \begin{verb}` $mployee \bowtie \rho_{\text{ESSN = SSN}}$`ork_On \end{verb}$ \\`$\Pi_{\text{Lname,}}$

f. $\Pi_{Lname}(\Pi_{\text{Lname, SSN}}$mployee \end{verb} - \Pi _{\text{Lname, SSN}} (\begin{verb} $mployee \bowtie \rho_{\text{ESSN = SSN}}$orks_On \end{verb}))$

g. $\Pi_{\text{Fname, Average Salary}}(_{\text{DName}}\mathcal{F}_{\text{average salary}}\text{P}_{\text{Dno = Dnumber}}$mployee \end{verb} \bowtie (\begin{verb} Depa

h. $_{\text{Sex}}\mathcal{F}_{\text{average salary}}(\sigma_{\text{sex = 'female'}}$mployee \end{verb})$

i. epartment1 \end{verb} \leftarrow \Rho _{\text{$Number = DNo$roject \end{verb} \bowtie \Rho _

j. $\Pi_{\text{Lname}}($mployee \end{verb} \bowtie _{\text{SSN = MGR_SSN}} \begin{verb} Department \end{verb $SSN) - \Pi_{\text{Lname}}$mployee \end{verb}$

Question #3

- ust# \end{verb} (\begin{verb} ORDER \end{verb})
- rder \end{verb} (\begin{verb} $\text{RDER}_I TEM$, HIPMENT \end{verb})
- tem# \end{verb} (\begin{verb} ORDER \end{verb}, \begin{verb} TEM)
- arehouse# \end{verb} (\begin{verb} SHIPMENT \end{verb})

Assumptions

  - Table headers with the same name are not unique.

Homework #4

Databases and File Structures

Illya Starikov, Jason Young, Claire Trebing

Due Date: April 21, 2016

1 Closures, Candidate Keys

- $A^+ = \{A\}$
- $AB^+ = \{A, B, C, D, E\}$ **Candidate Key**
- $AC^+ = \{A, B, D, C, E\}$ **Candidate Key**
- $ACD^+ = \{A, B, C, D, E\}$ **Candidate Key**
- $B^+ = \{B, D\}$
- $BC^+ = \{A, B, C, D, E\}$ **Candidate Key**
- $C^+ = \{C\}$
- $CD^+ = \{A, B, C, D, E\}$ **Candidate Key**
- $E^+ = \{A, B, C, D, E\}$ **Candidate Key**
- $DE^+ = \{A, B, C, D, E\}$ **Candidate Key**

2 Functional Dependence Equivalence

No, because there is no occurrence of $C \to D$ in G, but there is one in F.

3 Minimal Cover

Step 1

- $A \to B, A \to C, A \to D$
- $E \to G, E \to H$
- $C \to D$
- $AE \to J$

Step 2

- $A \to B, A \to C, A \to D$
- $E \to G, E \to H$
- $C \to D$
- $AE \to J$

Step 3

- $A \to B, A \to C$
- $E \to G, E \to H$
- $C \to D$
- $AE \to J$

4 3NF and BCNF

1. $F = \{AB \to C, C \to D, C \to A\}$, *Candidate Keys*: $(AB), (BC)$
   - No, because the in the functional dependency $C \to D$, D is not a part of any candidate key and A is not a super key.
   - No, because $\{C \to D\}$ is neither trivial nor a superkey.
2. $F = ACE \to BD, B \to C$, *Candidate Key*: (ACE)
   - Yes, because ACE is a super key and C is part of the candidate key.
   - No, because $\{B \to C\}$ is neither trivial nor a superkey.

ForFit

Never Quit Again

Illya Starikov, Jason Young, Claire Trebing

July 27, 2025

Contents

Part I.

The Database Design Manual

1. Problem Statement

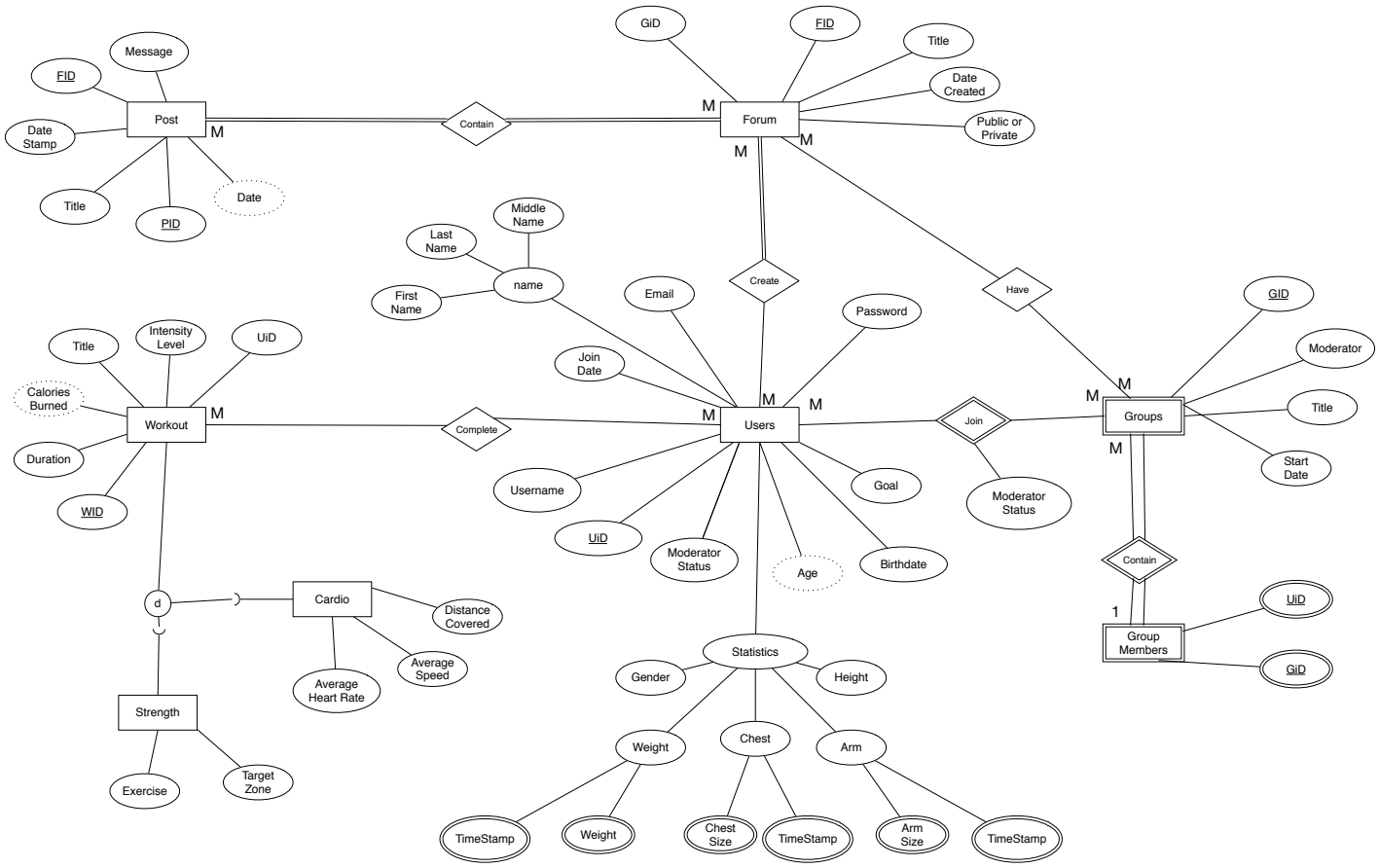
2015 marked a record year for Americans regularly exercising, hitting just over 55%. Keeping that momentum is difficult. As blue-collar jobs continue to decline, it has been more important than ever to keep a weekly regimen of healthy eating and exercising.

Living in the most interconnected generation poses quite a viable idea for social networking: healthy living. We can build a social network that connects users to friends, peers and family to make an online community of healthy living.

Our database will be essential because it will unite something so mundane and uninteresting with familiar faces. This will make exercise more enjoyable and offer a group to hold you accountable. We can shape an entire generation by having motivation a click away.

This database will consist of premade workouts, groups, and reminder emails to help you stay on top of your fitness plan.

2. Conceptual Database Design


GiD
FID
Title
Date Created
Public or Private
Forum
Contain
Post
Message
FID
Date Stamp
Title
PID
Date
M
M
M
M
Middle Name
Last Name
First Name
name
Email
Create
Password
Join Date
Have
GID
Moderator
Title
Start Date
Groups
Join
Moderator Status
Users
Complete
Workout
Title
Intensity Level
UiD
Calories Burned
Duration
WID
Username
UiD
Moderator Status
Age
Birthdate
Goal
Contain
1
Group Members
UiD
GiD
d
Cardio
Distance Covered
Average Speed
Average Heart Rate
Strength
Exercise
Target Zone
Statistics
Gender
Height
Weight
Chest
Arm
TimeStamp
Weight
Chest Size
TimeStamp
Arm Size
TimeStamp

3. Logical Database Design

3.1. Relational Set

Please note primary keys are signified by $^{\mathbf{PK}}$ and foreign keys are signified by ^{FK}.

Post

PiD $^{\mathbf{PK}}$	FiD ^{FK}	Message	DateStamp	Title	Date

Forum

FiD $^{\mathbf{PK}}$	Title	DateCreated	PublicOrPrivate	GiD ^{FK}

Groups

GiD $^{\mathbf{PK}}$	Moderator ^{FK}	Title	StartDate

Workouts

WiD $^{\mathbf{PK}}$	Duration	Title	IntensityLevel	CaloriesBurned	UiD ^{FK}

Strength

WiD $^{\mathbf{PK}}$	Duration	Title	IntensityLevel	CaloriesBurned	UiD ^{FK}	Target Zone

Cardio

WiD $^{\mathbf{PK}}$	Duration	Title	IntensityLevel	CaloriesBurned	UiD ^{FK}	AverageHeartRate
AverageSpeed	DistanceCovered					

Users

UiD $^{\mathbf{PK}}$	Username	Height	Birthdate	Goal	Password	JoinDate	Gender
FirstName	MiddleName	LastName					

Weight

UiD $^{\mathbf{PK, FK}}$	TimeStamp $^{\mathbf{PK}}$	Weight

Arm Size

UiD $^{\mathbf{PK, FK}}$	TimeStamp $^{\mathbf{PK}}$	Arm Size

Chest Size

UiD $^{\mathbf{PK, FK}}$	TimeStamp $^{\mathbf{PK}}$	ChestSize

Group Members

UiD **PK**	GiD *FK*	ModeratorStatus

3.2. Summary Table

Attribute	Data Type	Constraints	Meaning
Message	varchar	500 characters	Contents of a post.
FID	int	unique	Forum ID.
DateStamp	timestamp	timestamp of creation	Timestamp of post was creation.
Title	char	35 characters	Title of a post.
PID	int	unique	Post ID.
Date	timestamp	date of creation	Timestamp of post creation.
Title	char	35 characters	Title of a forum.
DateCreated	timestamp	timestamp of creation	Timestamp of forum creation.
PublicOrPrivate	boolean	-	Is forum public.
GID	int	unique	Group ID.
Moderator	char	must match a username	Group owner/moderator.
Title	char	35 characters	Title of the group.
StartDate	timestamp	timestamp of creation	Timestamp of group creation.
IntensityLevel	int	-	Enumerated value, code for different options.
Title	char	35 characters	Title of the workout.
CaloriesBurned	int	num > 0	How many calories burned during exercise.
Duration	timestamp	num > 0	Length of exercise.
WID	int	unique	Workout ID.
Exercise	char	35 characters	Exercise name.
TargetZone	char	35 characters	Area exercise is intended to workout.
AverageHeartRate	int	num > 0	Average heart rate recorded during exercise.
AverageSpeed	int	num > 0	Average speed of cardio exercise.
DistanceCovered	int	num > 0	Distance covered during cardio exercise.
Password	char	-	Password of user.
JoinDate	timestamp	timestamp of creation	Timestamp of users sign up.
Username	char	unique	Users username, used for sign in.
ModeratorStatus	boolean	-	Is a moderator.
Birthdate	date	-	Date of users birth.
Goal	varchar	500 characters	Users intended workout goals.
Gender	char	1 character, M or F	Users gender, male(M) or female(F).
Weight	int	num > 0	Users weight at a certain date.
ChestSize	int	num > 0	Users chest size at a certain date.
ArmSize	int	num > 0	Users arm size at a certain date.
Height	int	num > 0	Users height at a certain date.
UID	int	unique	Users ID.
TimeStamp	date	unique	Date of users weight mesurement.
TimeStamp	date	unique	Date of users chest measurement.
TimeStamp	date	unique	Date of users arm measurement

4. Application Program Design

4.1. Create a New User

This function creates a new user, accessing only the `user` table.

Input	`Username, Height, Birthdate, Goal, Password, Gender`.
Steps	1. Check to see if the `username` is available. If available, proceed. If not, display appropriate message to notify the user. 2. Insert appropriate information to the User table. (Username to `username`, height to `height`) 3. Generate a user ID, assign it to the `UiD` attribute. 4. Take a time stamp, assign it to the `JoinDate` attribute. 5. Calculate the age based on the `BirthDate` attribute.
Output	A new `User` entity will be inserted into the table, with appropriate data into proper columns (along with computed properties and derived properties).
Assumptions	`Username, Height, Birthdate, Goal, Password, Gender` are all correct (this will be validated in the sign up form).

4.2. Delete A Group

This function deletes a group by updating information in `Forum`, and removing the `Group` and `Group Members` tables.

Input	The Group ID (`GiD`) that is to be deleted.
Steps	1. Check to see if the request is made by the moderator via the `Moderator` column in the `Groups` table. If true, approve the request. If not, cancel the request and notify the user. 2. Remove the row that has a matching `GiD` that was provided for deletion in the `Groups` table. 3. Query the `Group` Members table, removing any row that match the GiD provided for deletion. 4. Query the `Forum` table for any matching Group Ids (`GiD`), setting the `GiD` to `null` if matching.
Output	The groups are deleted, the group members within that group are deleted, and any reference to the group is deallocated.
Assumptions	None.

4.3. Modifying User Statistics

Our user statistics have the ability to fluctuate. We would like to accommodate for this fluctuation by allowing users to update their respective statics; specifically, we would like to let users update `Username, Height, Goal, Password, Gender` in the `Users` table.

Input	The specific attribute(s) of the set `Username, Height, Goal, Password, Gender` that would like to be updated with the new value.
Steps	1. Ensure the data is valid (e.g. is not `null` when applicable, in the proper domain). If it is valid, continue. If not, prompt the user with an error message and try again. 2. Modify the attribute to reflect the new value. 3. Repeat for any additional attributes provided.
Output	The attribute(s) should now reflect the new value provided.
Assumptions	The data is within a proper range (will not overflow).

4.4. Query Other Users

This function allows for users to query other users; this can be done via `Username` or `FirstName, MiddleName`, and `LastName` from the `Users` table.

Input	Either a `Username` xor any subset of `FirstName, MiddleName`, or `LastName`.
Steps	1. Check to see if input is valid. If is, proceed. If not, display error message to the user. 2. Query the `Users` table to see if the user exists. If the user exists, proceed. If not, display appropriate message to the user. 3. Project the profile.
Output	Either the search user will be projected or an error message is the user does not exist.
Assumptions	The first, middle and last name are all provided. The names are unique (solely for the testing purposes).

4.5. User Leaderboards

Generate the leaderboard based on the workouts accomplished; specifically aggregating data from the `Strength` and `Cardio` table. *Note this is the function that requires multiple tables.*

Input	None.
Steps	1. Merge the `User` and `Workouts` table, call the new table `Merged`. 2. Add up the total duration (call the new property `TotalDuration`) in the `Merged` table based on the `UiD` attribute, making a new table named `Sums`. 3. Sort the `Sums` by the `TotalDuration` attribute. 4. Display the top 10 on the sorted `Sums` table to the user. 5. Display the user their current rank.
Output	An eleven-row table displaying the top 10 leaderboards and the user's current rank.
Assumptions	There is a bare minimum of eleven users.

Part II.

User Manual

5. A Brief Introduction To ForFit

ForFit is a new way to workout. Over the past year, just about 55% of Americans exercised regularly. ForFit is a way to connect you to your friend, family, and other exercise enthusiasts in your area. These exercises can range from traditional bike riding and swimming to newer forms of exercise such as parkour and yoga. ForFit encourages group activity and connection so that workouts are never dull and so that users will be continually motivated and pushed toward their workout goals.

6. How It Works

ForFit has two main components. The first part is the users themselves. Users sign up for the service and then begin recording their workouts. Workouts can be logged as many times as liked, from once a month to several times a day. The more workouts logged, the better data and the more progress toward your exercise goals.

Users can view other users and the workouts they've done. This leads into the second component of ForFit, which is the encouragement of community. Users can create groups and join existing groups. These groups can help users focus on specific areas. For example, a user wanting to run a marathon might join a group of other runners training for a marathon. This way the user can keep in touch and stay motivated through the group and also have a resource to ask questions or get other workouts.

Groups allow users to make forums, for in-depth discussion, and posts, for quick updates. These groups are the main area of community creation that ForFit focuses on in the design. Users can find other users and join groups with users they have been exercising with before. This creates a network of community interaction.

7. Test Procedure

This database was well tested before it reached you, the user. This section will explain how the test database was created and what tests the database went through before completion. The mock data was created using C++ code to randomly generate data. Data from online was used to create First Names, Last Names, Usernames, and Group Names as well as to populate the workout subclasses in strength and cardio. Using C++ code to randomly select and combine information, creating unique data. This mock data was then put through a series of tests to make sure the functions would work properly. Each function was tested over 500 times to error or improper analysis of the data. After confirming that these functions were working properly, they were implemented into the database.

8. Functions

8.1. Delete Group

Delete Group allows you to remove a group from the database. Once a group has been deleted, you can no longer make forums or posts in this group.

Parameters

groupID The `GiD` of the group that is to be deleted.

Results

The group who's ID matches the given `groupID` will be deleted. If the group does not exist, nothing is displayed.

```
rMBP:~ postgres$ '/Applications/Postgres.app/Contents/Versions/9.5/bin'/psql -p5432
psql (9.5.2)
Type "help" for help.

postgres=# select delete_group(42);
 delete_group
--------------
(0 rows)

postgres=#
```

8.2. Modify User

Modify User allows users to change certain values in the user's data by providing the user name. Users can change their username, height, goal, password, and gender.

Parameters

userID The `UiD` of the user to who's statistics that will be modified. If this is not provided, there is no way of changing the user.

newUsername The new username property of the user. If this statistic will not change, pass a `'0'` to the function. Otherwise, proceed with changing.

newHeightFeet See above.

newHeightInches See above.

newGoal See above.

newPassword See above.

newGender See above.

Results

For all attributes that do not have the value of `'0'`, the value of said attribute will be updated (provided it upholds all constraint).

```
rMBP:~ postgres$ '/Applications/Postgres.app/Contents/Versions/9.5/bin'/psql -p5432
psql (9.5.2)
Type "help" for help.

postgres=# select modify_user(1, 'IllyaStarikov', 0, 0, 42, '0', 'F');
 modify_user
-------------
(0 rows)

postgres=# select * from USERS where UiD = 1;
 uid |   username    |          email          | heightfeet | heightinches | birthdate  | goal |   password    |  joindate  | gender | fname | minit |  lname
-----+---------------+-------------------------+------------+--------------+------------+------+---------------+------------+--------+-------+-------+----------
   1 | IllyaStarikov | CasieScarlett1@gmail.com |          6 |            2 | 1967-03-31 |   42 | EtZEdlnfNiBAI | 1999-12-31 | F      | Casie | M     | Scarlett
(1 row)
```

8.3. New User

New User creates new users in the database. The user must input a username, email, height, birthdate, goal, and password. If any of those are not provided, an error message will be generated.

Parameters

id, username, email, heightFeet, heightInches, birthdate, goal, password All the attributes that can be inserted into user table.

joinDate, gender, firstName, mInitial, lastName See above.

Results

Provided all the attributes meet database constraints, a new user will be inserted into the database. If a constraint is not meant, there will be a message outputting the error and the insert will be rejected.

```
postgres=# select new_user(404, 'IllyaStarikov', 'IllyaStarikov@iCloud.com', 5, 10, '07-20-1994', 165, '42', '07-21-1994', 'M', 'Illya', 'A', 'Starikov');
 new_user
----------
(0 rows)

postgres=# select * from USERS where UiD = 404;
 uid |   username    |          email          | heightfeet | heightinches | birthdate  | goal | password |  joindate  | gender | fname | minit |  lname
-----+---------------+-------------------------+------------+--------------+------------+------+----------+------------+--------+-------+-------+----------
 404 | IllyaStarikov | IllyaStarikov@iCloud.com |          5 |           10 | 1994-07-20 |  165 | 42       | 1994-07-21 | M      | Illya | A     | Starikov
(1 row)
```

8.4. Query User

Query User allows users to find others by inputting a first and last name. If a user does not exist than the results are empty.

Parameters

firstName, lastName The first and last name of the user to be queried.

Results

If the user exists, the column of the user will be returned. If the user does not exist, not such column will be returned.

```
[rMBP:~ postgres$ '/Applications/Postgres.app/Contents/Versions/9.5/bin'/psql -p5432
psql (9.5.2)
Type "help" for help.

[postgres=# select query_user('Illya', 'Starikov');
                                   query_user
-------------------------------------------------------------------------------------------
 (117,IllyaStarikov,IllyaStarikov@iCloud.com,5,10,1994-07-20,165,42,1994-07-21,M,Illya,A,Starikov)
(1 row)
```

8.5. Total Users

Total Users returns how many users are in the whole database. In the future, this function will give a user count for an individual group.

Parameters

None.

Results

Returns the total count of users, a maintenance function used to compare to the total count in the database.

```
[rMBP:~ postgres$ '/Applications/Postgres.app/Contents/Versions/9.5/bin'/psql -p5432
psql (9.5.2)
Type "help" for help.

[postgres=# select total_users();
 total_users
-------------
          52
(1 row)
```

8.6. User Leaderboards

User Leaderboards gives you a top 10 user count. This is calculated by a count of workouts done by a user. This count includes all the users in a database and is not specific to a group.

Parameters

None.

Results

Returns the top 10 users (calculated via count of workouts done) in the database.

```
rMBP:~ postgres$ '/Applications/Postgres.app/Contents/Versions/9.5/bin'/psql -p5432
psql (9.5.2)
Type "help" for help.

postgres=# select user_leaderboards();
   user_leaderboards
----------------------
 (Gregory,Patty,6)
 (Agueda,Heather,4)
 (Bertram,Gerardoy,4)
 (Gay,Arthury,4)
 (Garry,Shanony,4)
 (Brad,Lana,4)
 (Francisco,Bruce,4)
 (Edgar,Dennisy,4)
 (Alvina,Carter,4)
 (Darlene,Salvatore,3)
(10 rows)
```

9. Shell

We know doing tasks such as creating the database, generating a new set of random inputs, or even running all the function files to makes sure all of the function are created can tedious. Forunatly, we have a solution — that solution is Shell scripting.

We quickly realized that repetitively running the same commands[1] could get tedious, so we have provided multiple shell scripts that automate most of the monotonous tasks.

To run a shell script, first you must give it privileges to be able to run `Bash` commands. This is accomplished by `chmod -x /path/to/file.sh`. This give read/write access to the shell script, which then allows the shell script to be run via `./path/to/file.sh`. This then allows the user to do tasks like setting up the entirety of the database (which spans multiple files!) one file path away.

Below are the shell scripts we provide.

9.1. Create Database

A lot of testing requires the use of setting up the database from scratch. Unfortunately, this usually means creating the database scheme, followed by inserting the data, followed by inserting the functions, and so forth. Although an easy fix would be to have all the code in a single file, this would combine concerns. We would rather have a lot of function specific files apposed to one giant file. Our solution: `./create_database.sh`

The create database script works like so:

1. Create the database schema by `create`ing all the tables and `modify`ing all for foreign key constraints.
2. Insert data, which is artifically generated (as will be mentioned below).
3. Go through the *entirety* of the functions directory, and running each function `sql` script to ensure all functions are inserted into the database.

This allows for easily bringing a database back up after it has been reset.

9.2. Random Input

Let face it, every social network occasionally needs some help. That is why we have a shell script that executes our C++ codebase to auto-populate the table. The shell script runs like so:

[1] Like running `` `/Applications/Postgres.app/Contents/Versions/9.5/bin'/psql -p5432 -f `` to create every file!

1. Change to the directory of the C++ codebase.
2. Run the makefile
3. Run the executable, pipe that to an output file.
4. Move the shell script to the directory of all the other `sql` files.

This makes for easily produced, new data.

9.3. Drop Database

Drop database simply runs the `sql` script to drop all tables in a cascade fashion. It's a simple script but is much quicker than manually running the command to drop the database.

10. Website

To setup the website you only need to place the website files inside of an apache server and then rewrite the config.php file.

For the config.php file you need to modify the variables;

```
"
   date_default_timezone_set('America/Chicago');
   define('DB_SERVER', 'localhost:3306');
   define('DB_USERNAME', 'JasonY');
   define('DB_PASSWORD', 'Servant83');
   define('DB_DATABASE', 'forfit');
"
```

to fit the website's database.

The website is a blend of PHP and HTML. It allows you to sign up, logout, view groups and forums.

The main page contains the link to signup. After signing up or logging in you can open the menu and view the groups which contain forums. You may also enter your personal information in the profile which tracks your exercises and gains.

CS2500

Algorithms

Homework #1

Illya Starikov

Due Date: February 3, 2016

1 Insertion Sort

1.1 Operations of Insertion Sort

3 $\underline{41}$ 52 26 38 57 9 49	1 Comparison(s).
3 41 $\underline{52}$ 26 38 57 9 49	1 Comparison(s).
3 41 52 $\underline{26}$ 38 57 9 49	3 Comparison(s).
3 26 41 52 $\underline{38}$ 57 9 49	3 Comparison(s).
3 26 38 41 52 $\underline{57}$ 9 49	1 Comparison(s).
3 26 38 41 52 57 $\underline{9}$ 49	6 Comparison(s).
3 9 26 38 41 52 57 $\underline{49}$	3 Comparison(s).
3 9 26 38 41 49 52 57	0 Comparison(s).

1.2 Number of Comparisons

18 total operations.

2 Question 2

B	A	O	o	Ω	ω	Θ
2^n	$2^{\frac{n}{2}}$	no	no	yes	yes	no
$n^{\lg c}$	$c^{\lg n}$	yes	no	yes	no	yes

2.1 Justification

To determine larger asymptotic growth, take the limit of one function over the other. Arbitrarily choosing 2^n for the numerator, we see that:

$$\lim_{n\to\infty} \frac{2^n}{2^{\frac{n}{2}}} = \lim_{n\to\infty} 2^{\frac{n}{2}} = \infty \tag{1}$$

Alternatively, if we chose 2^n as the denominator, we notice that $\lim_{n\to\infty} \frac{2^{\frac{n}{2}}}{2^n} = 0$, so we know that the condition $2^n > 2^{\frac{n}{2}}$ holds, and for certain conditions the functions are equal (take $n = 0, 2^0 = 2^{\frac{0}{2}}$).

$\therefore 2^n$ is $\omega(2^{\frac{n}{2}})$ and $\Omega(2^{\frac{n}{2}})$. QED.

2.2 Justification II

By the properties of logarithms, we will show that $n^{\lg c} = c^{\lg n}$.

$$\begin{aligned} n^{\lg c} &= c^{\lg n} && (2)\\ &= c^{\log_2 n} && (3)\\ &= c^{\frac{\ln n}{\ln 2}} && (4)\\ &= e^{\ln c \times \frac{\ln n}{\ln 2}} && (5)\\ &= e^{\frac{\ln c}{\ln 2} \times \ln n} && (6)\\ &= e^{\lg c \ln n} && (7)\\ &= n^{\lg c} && (8) \end{aligned}$$

$\therefore n^{\lg c}$ is $O(c^{\lg n})$, $\Omega(c^{\lg n})$ and $\Theta(c^{\lg n})$. QED.

3 Big-O Implies Big-Ω

Theorem: Let f(n) and g(n) be asymptotically positive functions, $O(g(n))$ be the set $\{f(n) : \exists c, n_0 \in \mathbb{R}^+, \forall n, n_0 \in \mathbb{R}, 0 \leq f(n) \leq c\ g(n) \wedge n > n_0\}$, and $\Omega(g(n))$ be the set $\{f(n) : \exists c, n_0 \in \mathbb{R}^+, \forall n, n_0 \in \mathbb{R}^+, 0 \leq cg(n) \leq f(n) \wedge n > n_0\}$ Then,

$$f(n) = O(g(n)) \Rightarrow g(n) = \Omega(f(n)) \quad (9)$$

[*We will prove so by contradiction.*]

Proof: Suppose not. That is, suppose

$$\begin{aligned} \sim [f(n) = O(g(n)) \quad &\Rightarrow \quad g(n) = \Omega(f(n))] && (10)\\ f(n) = O(g(n)) \quad &\wedge \quad \sim [g(n) = \Omega(f(n))] && (11) \end{aligned}$$

Using the formal definition of $\Omega(g(n))$, we can see the negation is as follows:

$$\sim \Omega(g(n)) = \{f(n) : \forall c, n_0 \in \mathbb{R}^+, \exists n, n_0 \in \mathbb{R}^+ 0, > cg(n) > f(n) \wedge n \leq n_0\} \quad (12)$$

This is a contradiction, for there are no c, n_0 such that $0 \leq cg(n) \leq f(n) \wedge 0 > cf(n) > g(n)$ because c, n_0 are defined as *positive* constants, and $f(n), g(n)$ are defined as *positive* functions. Because no such positive function and positive constants exists to satisfy

$$0 > cf(n) > g(n) \tag{13}$$

This has led us to a contradiction. QED.

4 Asymptotic Proofs

4.1 Proof I

Assuming $n > 1$,

$$\begin{aligned} n^2 &\leq 20n^2 + 2n + 5 && (14)\\ &\leq 20n^2 + 2n^2 + 5 && (15)\\ &\leq 27n^2 && (16) \end{aligned}$$

$\therefore$ For $C = 27, n_0 = 1$, $20n^2 + 2n + 5 = O(n^2)$. QED.

4.2 Proof II

Assume $C = 1, n_0 = 1$. $\therefore$This satisfies the condition that c and n_0 are positive constants such that $0 \leq Cn^2 \leq 5n^2 - 15n + 100 \ \forall n \geq n_0$. QED.

4.3 Proof III

4.3.1 Lower Bound

Assume $C = 1, n_0 = 1$. $\therefore$This satisfies the condition that c and n_0 are positive constants such that $0 \leq Cn^2 \leq 5n^2 + 2n \ \forall n \geq n_0$. QED.

4.3.2 Upper Bound

Assuming $n > 1$,

$$\begin{aligned} n^2 &\leq 5n^2 + 2 && (17)\\ &\leq 5n^2 + 2n^2 && (18)\\ &\leq 7n^2 && (19) \end{aligned}$$

$\therefore C = 7, n_0 = 1, 5n^2 + 2n = \Theta(n^2)$. QED.

4.4 Proof IV

To prove that $5n + 7 = o(n^2)$ we must show that $\exists c, n_0 \in \mathbb{R}^+, 0 \leq f(n) < Cg(n) \wedge n > n_0$. We do so by showing that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. [*with the help of L'Hôpital's Rule*]

$$\lim_{n\to\infty} \frac{5n+1}{n^2} \tag{20}$$

$$= \lim_{n\to\infty} \frac{5}{2n} \tag{21}$$

$$= 0 \tag{22}$$

QED.

Homework #2

Illya Starikov

Date Due: February 15, 2016

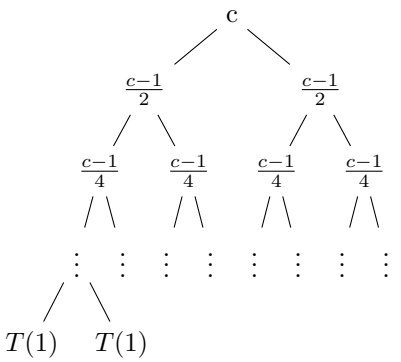
1 Programming Assignment

Code is attached. Binary search implementation is as follows.

```
extension Array where Element: Comparable {
    func binarySearch(key: Element) -> Int? {
        return binarySearch(key, low: 0, high: self.count - 1)
    }

    private func binarySearch(key: Element, low: Int, high: Int) -> Int? {
        if low > high { return nil }

        let middle = Int((low + high) / 2)

        if key == self[middle] {
            return middle
        } else if key > self[middle] {
            return binarySearch(key, low: middle + 1, high: high)
        } else {
            return binarySearch(key, low: low, high: middle - 1)
        }
    }
}
```

2 Recurrence Equation

2.1 Tower of Hanoi



The complexity of this tree is $O(\lg n)$, or alternatively, $O(2^n)$

2.2 Merge Sort

$$T(n) = 2\ T(\frac{n}{2}) + n \tag{1}$$

From the equation, we can see that

$$a = 2, b = 2, c = 1, f(n) = n \tag{2}$$

We observe that

$$f(n) \in \Theta(n^{\log_b a}) = \Theta(n^{log_2 2}) = \Theta(n) \tag{3}$$

It follows from Case #2 of the Master Theorem

$$\begin{aligned} T(n) &\in \Theta(n^{\log_b a} \lg n) && (4)\\ &= \Theta(n^{\log_2 2} \lg n) && (5)\\ &= \Theta(n^1 \lg n) && (6)\\ &= \Theta(n \lg n) && (7) \end{aligned}$$

Thus the recurrence relation $T(n)$ is in $\Theta(n \log n)$. QED.

3 Loop Invariant

```
1  Merge(A, p, q, r)
2      leftIndex = q - p + 1
3      rightIndex = r - q
4      let L[1..left + 1] and R[1..right + 1] be new arrays
5      for i = 1 to leftIndex
```

```
6           L[i] = A[p + i - 1]
7       for j = 1 to rightIndex
8           R[j] = A[q + j]
9       L[leftIndex + 1] = sentinel
10      R[rightIndex + 1] = sentinel
11      i = 1
12      j = 1
13      for k = p to r
14          if L[i] <= R[j]
15              A[k] = L[i]
16              i = i + 1
17          else A[k] = R[j]
18              j = j + 1
```

At the end of the for loops on 4-5, 6-7, new arrays will be created, holding the left and right half of the data in the passed array.

Initialization Initially we have two arrays, the left and right side of the original passed array. This holds trivially.

Maintenance During each iteration, we copy over the array's data.

Termination Upon termination, the Left array has the data from $[p + i - 1]$ and Right array has the data of $[q + j]$.

During lines 12 - 17, we merge the arrays to get a properly sorted array.

Initialization Initially we have the left and right arrays, unsorted, and the original array. This holds trivially.

Maintenance During the iterations of the array, we replace the data of the originally passed array with the smaller of the Left and Right array.

Termination Upon termination, we have a fully sorted array from $q...r$.

4 Quicksort

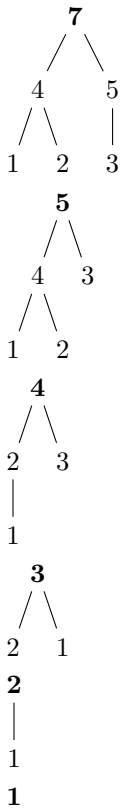
Let _ signify the pivot, || signify the wall.

1. || 4 2 7 6 3 5 $\underline{1}$
2. 1 || || 4 2 7 6 3 $\underline{5}$
3. 1 || 4 || 2 7 6 3 $\underline{5}$
4. 1 || 4 2 || 7 6 3 $\underline{5}$
5. 1 || 4 2 3 || 7 6 $\underline{5}$
6. 1 || 4 2 $\underline{3}$ || 5 || 7 6
7. 1 || || 4 $\underline{3}$ || 5 || 7 6

8. 1 || 2 || 4 3 || 5 || 7 6
9. 1 || 2 || 3 || 4 || 5 || 7 6
10. 1 || 2 || 3 || 4 || 5 || || 7 6
11. 1 || 2 || 3 || 4 || 5 || 6 || 7
12. 1 || 2 || 3 || 4 || 5 || 6 || 7

5 Heapsort

5.1 Heap Representation



5.2 Enumerated Steps

1. 7
2. 7 5
3. 7 5 4

4. 7 5 4 3

5. 7 5 4 3 2

6. 7 5 4 3 2 1

Homework #3

Illya Starikov

Due Date: February 28, 2016

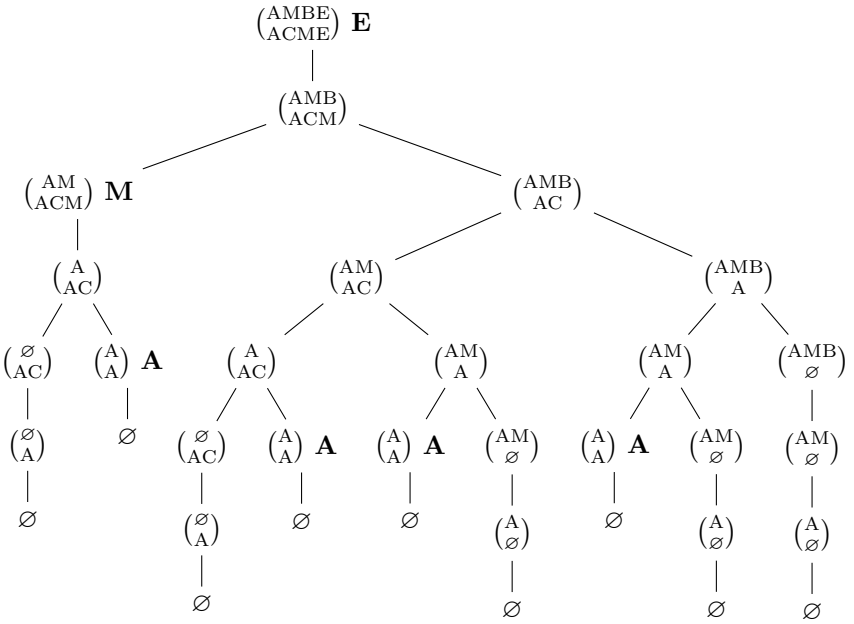
1 Matrix Multiplication

	1	2	3	4
1	0	$20^{[1]}$	$35^{[2]}$	$65^{[3]}$
2	X	0	$12^{[2]}$	$28^{[3]}$
3	X	X	0	$6^{[3]}$
4	X	X	X	0

Therefore, the minimum scalar operations can be achieved via $(((A_1 \times A_2) \times A_3) \times A_4)$

2 Longest Common Subsequence

2.1 Recursion Tree



Therefore, the longest common substring is AME.

2.1.1 Trace Table

	∅	A	M	B	E
∅	0	**0**	0	0	0
A	0	1	**1**	1	1
C	0	1	**2**	1	1
M	0	1	**2**	**2**	2
E	0	1	2	2	**3**

Therefore, the longest common substring is AME.

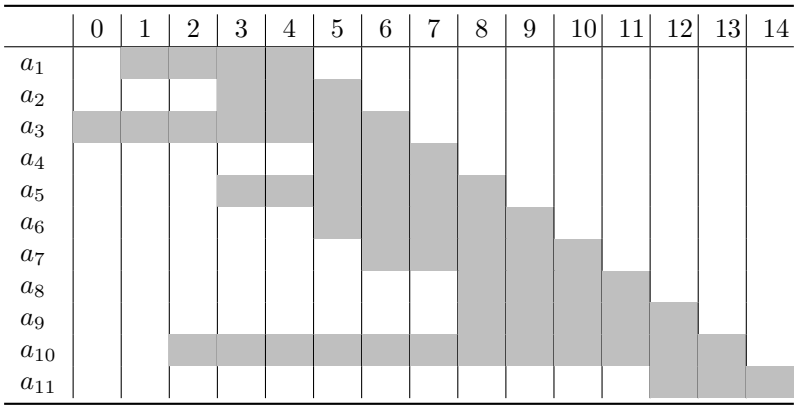
Homework #4

Illya Starikov

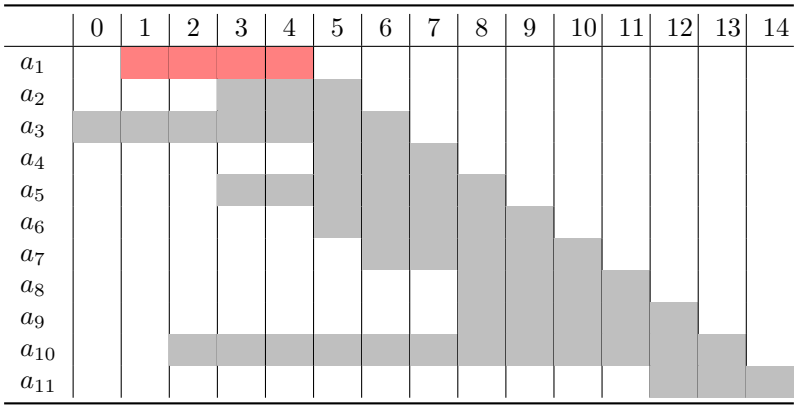
Due Date: March 23, 2016

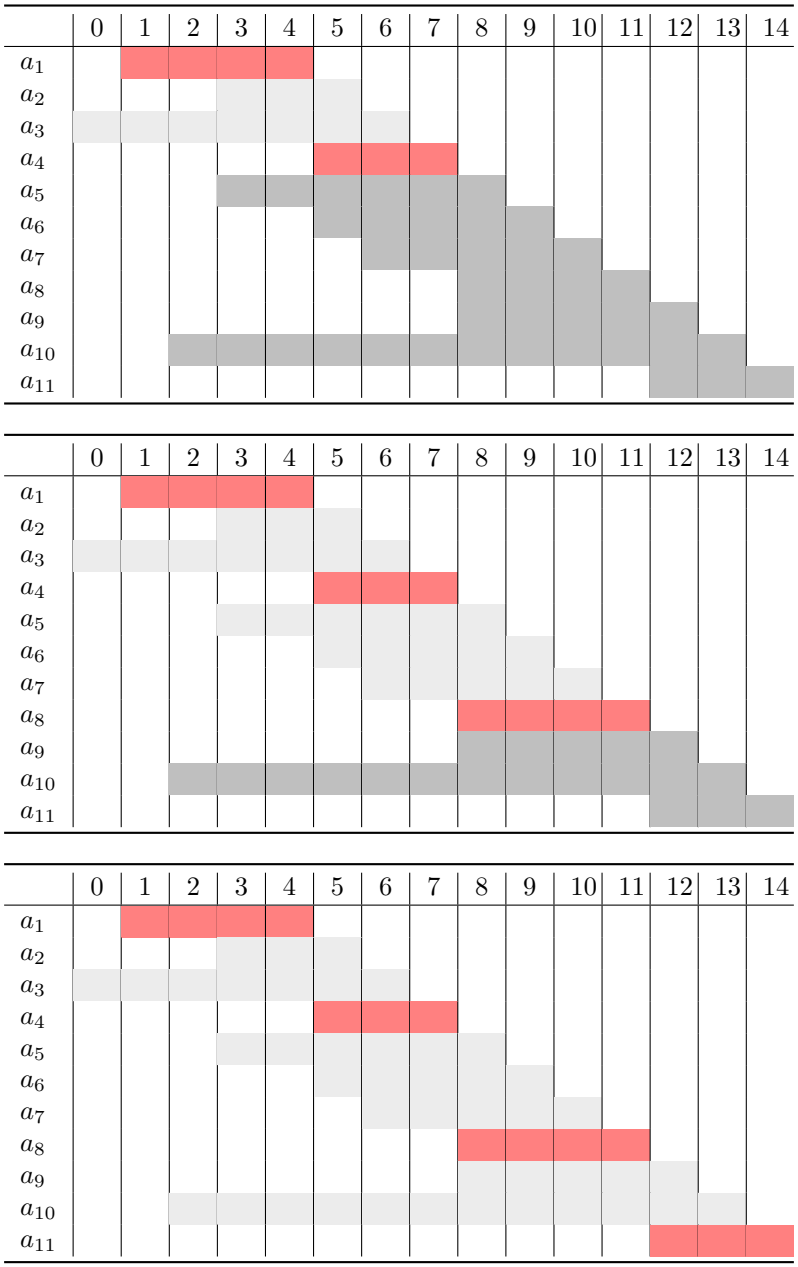
1 Activity Selection Problem

Plotting this out on a graph,

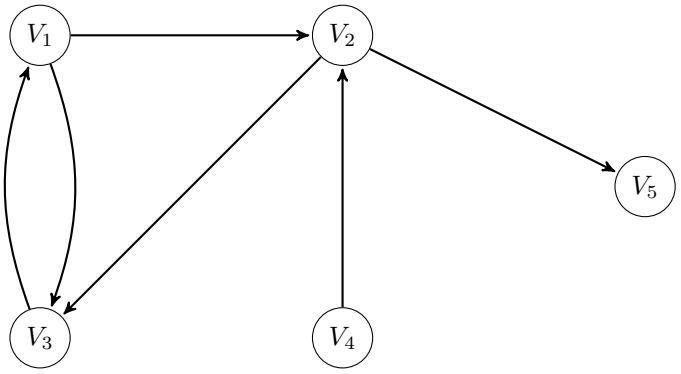


We can plainly see the optimal solution to be $\{a_1, a_4, a_8, a_{11}\}$. We get this solution by using the ending first greedy algorithm. Here are the steps to obtain the solution.




0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
a_1
a_2
a_3
a_4
a_5
a_6
a_7
a_8
a_9
a_{10}
a_{11}

2 Graph And Depth First Search



2.1 Depth First Search

Ouput:	V_1	V_2	V_3	V_5	V_4
Distance:	0	1	1	2	∞

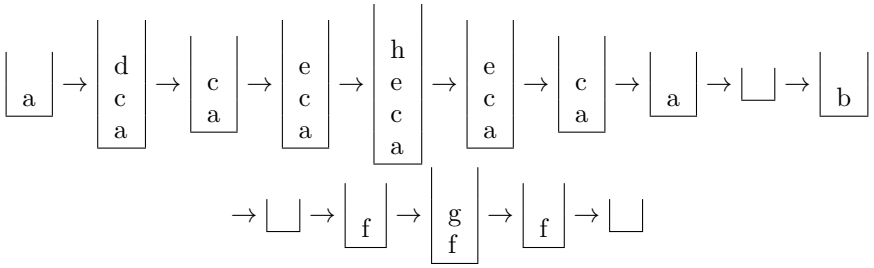
2.1.1 Queue Status

V_1
V_2 V_3
V_3 V_5
V_5
V_4

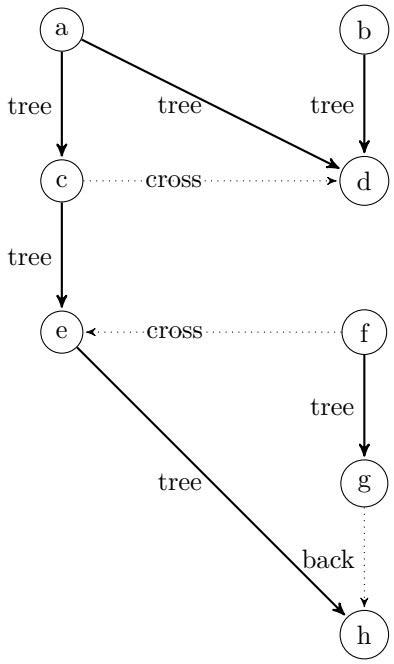
3 Depth First Search and Topological Sorting

Ouput	**Discovery**	**Finish**
a	1	10
c	2	9
d	3	4
e	5	8
h	6	7
b	11	12
f	13	16
g	14	15

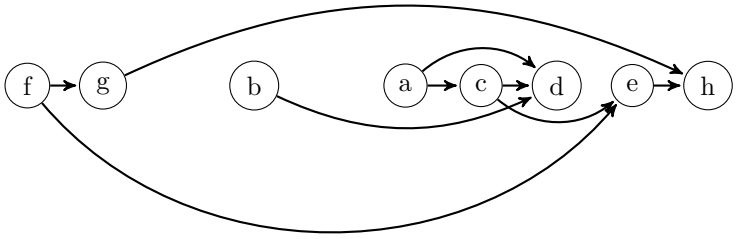
3.0.1 Queue Status



3.0.2 Edge Classification



3.0.3 Topological Sorting



4 Kruskal's and Prim's Algorithm

Please note that my tables are different, however the algorithm is the same is the same.

4.1 Kruskal

Kruskal's works the same way, except my column headers are more verbose and there an Iteration Number column, making it easy to keep track of large algorithms.

Iteration Number	Edge	Weight	Action Taken
1	$\{u,\ y\}$	1	Added
1	$\{u,\ x\}$	1	Added
2	$\{s,\ t\}$	2	Added
3	$\{t,\ y\}$	2	Added
4	$\{s,\ u\}$	3	Not Added
5	$\{s,\ x\}$	3	Not Added
6	$\{t,\ x\}$	3	Not Added

4.2 Prim

Prim's is also the same, with the exception that my headers are also verbose and the table is sorted linearly by the vertexes and edges that are added. The process is still the same of starting a vertex and branching out based on the best possible choice at the moment.

Iteration Number	Vertex Added	Edge Added	Weight
0	s		
1	t	$\{s,\ t\}$	2
2	x	$\{t,\ y\}$	2
3	u	$\{y,\ u\}$	1
4	y	$\{u,\ x\}$	1

Both algorithms produce the same result.

Programming Project I, First Report

Illya Starikov, Claire Trebing, & Timothy Ott

Due Date: March 07, 2016

1 Abstract

Smartphone users launch many apps everyday, however one of the most fundamental things a smartphone does is abstracted away: memory management.

Although smartphones have advanced significantly in many ways compared to their first predecessors (in terms of RAM, architecture, processors), deactivation, (the process of "the operating system needing to choose and remove some apps from the memory", a subproblem of memory management) is a solution that is often less-than-perfect. Although Java's Garbage Collection and Swift's Automatic Reference Counting (ARC) have sufficed, there are other methods.

In this project I propose to solve this problem using three techniques:

- Brute Force
- Dynamic Programming
- Greedy Solution

2 Introduction and Motivation

As stated previously, memory management is solved in a less-than-perfect manner. Although current technology suffices, we would like to compare algorithms to show the significant gains via three different approaches (Brute Force, Dynamics Programming, and Greedy).

3 Proposed Solution

For our project we decided to take a more skeuomorphic and object oriented approach, modeling objects after their real world counterparts, such as Application or Smartphone. As for the approaches, we have the following solutions:

3.1 Brute Force

For the brute force method, we knew that we had to check every possible subset (and for a set of size n, we know there to be 2^n subsets). We used this to our advantage, creating a lookup table modeled after a truth table. As an example, suppose we have a three items in our knapsack:

Items A	Item B	Item C	Item Number
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

We notice that the knapsack combination can be directly summarized by the n-bit binary representation. Using the previous example of three item knapsack:

$$\text{Subset } \#0 : 0_{10} = 000_2 = \sim (ABC) \quad = \quad \text{No Items} \tag{1}$$

$$\text{Subset } \#5 : 5_{10} = 101_2 = A \sim (B)C \quad = \quad \text{Items A, B} \tag{2}$$

$$\text{Subset } \#7 : 7_{10} = 111_2 = ABC \quad = \quad \text{Items A, B, C} \tag{3}$$

Now that we have have a represenation of every subset, we can multiply the columns value by the knapsack value to get the item's value into the lookup table. Extending the example, suppose we have the following table:

App	Memory	Cost
A	2	4
B	3	6
C	7	9

Multiplying the columns by the values produces the following results:

App A	App B	App C
0×4	0×6	0×9
0×4	0×6	1×9
0×4	1×6	0×9
0×4	1×6	1×9
1×4	0×6	0×9
1×4	0×6	1×9
1×4	1×6	0×9
1×4	1×6	1×9

$=$

App A	App B	App C
0	0	0
0	0	9
0	6	0
0	6	9
4	0	0
4	0	9
4	6	0
4	6	9

Adding the columns across we produce the following, and factoring the weight:

Items A	Item B	Item C	Total Cost	Total Memory
0	0	0	0	0
0	0	9	9	7
0	6	0	6	3
0	6	9	15	10
4	0	0	4	2
4	0	9	12	9
4	6	0	10	5
4	6	9	19	12

Now finding the minimal value is straightforward, find the minimal value is traversing down the additive value while looking at the size it frees up. Suppose we have to free up 10 blocks of memory, items A, B would be the most efficient choice.

The pseudocode is as follows, with a few notes:

- Comparison is done while "generating" a table.
- A table is not generate, but the binary representation is used instead.

3.1.1 Pseudocode

```
knapsackBrute(items, knapsackSize)
    minumum = 0
    for i = 0 to 2^n - 1
        subset = binaryToInteger(i)
        sum = 0

        for i to subset.length
            sum = sum + item[i].benefit * subset[i]
            size = size + item[i].weight * subset[i]

        if size <= knapsackSize && sum < minumum
            minumum = sum
            greatestSubset = i

    subset = binaryToInteger(greatestSubset)
    for i = 0 to subset.size
        if subset[i] == 1
            optimalSolution.append(item[i])

    return optimalSolution
```

3.1.2 Time Complexity

Analyzing this algorithm, we can see the complexity:

$$\sum_{k=1}^{n} 2^k \times \sum_{k=1}^{n} c = \mathcal{O}(n2^n) \tag{4}$$

We run in $\mathcal{O}(n2^n)$ time.

3.2 Dynamic Programming

The approach to dynamic programming came from creating a table of the possible memory and cost combinations. The table would contain rows for the values from 0 to M, the largest possible memory we would have to free up. The program starts by finding the lowest amount of memory an app can take up and the lowest cost. This will act as a base case. For example, if the lowest memory amount is 3 and the cost is 1, then if the user needed to clear out 1, 2, or 3 memory spaces then the phone would close that app. Amounts greater than 4 would be calculated when the program started.

3.2.1 Pseudocode

```
Create_Delete_Table(numberOfApps, numberToDelete, Memory[], Cost[]){
    Let d[1 ... M, 3] be new table as shown below
    Memory To Delete | Nr. of Apps | Cost    |  List of Apps Used
            0
            1
            2
            3
            ...
            M

    int low = findMin(d[],Memory[], Cost[]);
  //This function will find a base case for an app with
  the lowest memory and lowest cost
    for (j = 1 to low)
    {
            d[j, 0] = d[low, 0];
            d[j, 1] = d[low, 1];
            d[j, 2] = d[low, 2];
            d[j, 3] = d[low, 3];
    }

    for(j = low to M)
    {
            for( i = 1 to N){
                    if(Mi == j && Ci < d[j-low,2] + d[low,2]){
                            d[j, 1] = 1;
                            d[j, 2] = Ci;
```

```
                    d[j, 3] = i;
                }

                else{
                    d[j,1] = d[j-low,1]++;
                    d[j,2] = d[j-low,2] + d[low,2];
                    //add app to the list
                }
            }
        }
        return;
}
```

3.2.2 Time Complexity

We can see that this algorithm is $\mathcal{O}(n^2)$.

3.3 Greedy Solution

After considering several greedy algorithm solutions we settled on selecting based on a ratio of cost and memory freed by deactivating a particular app. Once each of the apps' ratio has been evaluated (and the ratio has been stored as a member variable) we sort the array of apps in an ascending order according to this new value. Because the apps are now in order in ascending cost per unit of memory we can now simply add apps to the solution set in the order they appear in the array until our memory requirements are met. For example, if we take our example set of three items:

App	Memory	Cost
A	2	4
B	3	6
C	7	9

We first take the ratio of each item's cost over the capacity of memory it takes up and sort the elements accordingly. Doing this gives us the following updated table:

App	Memory	Cost	Ratio
C	7	9	1.29
A	2	4	2
B	3	6	2

And now we simply add apps until our memory requirement has been met or exceeded. Suppose we need to free up 9 units of memory, according to this approach the optimal solution would be items C and A.

It is worth noting, however, that this solution doesn't always produce the absolute optimal solution. If we use the previous example of freeing up 10 units

of memory this algorithm produces a solution of all the apps, A, B and C as opposed to the actual optimal solution of app C and B. This inaccuracy is a sacrifice in order to benefit from a much lower runtime than that of the other solutions.

We could perhaps introduce a second greedy choice or to further refine our algorithm to improve the accuracy but we would do so at the cost of performance and the solutions chosen by our initial solution are accurate enough for our purposes.

3.3.1 Pseudocode

```
knapsackGreedy()
   A[] = Array of Apps
   S[] = Array of solution set
   M = Desired amount of memory to be freed
   sm = Amount of memory freed by solution set

   n = A.length

   for 0 to n
     A[i].ratio = A[i].cost/A[i].memory

   mergesort(A)

   GreedKnap(A, S)
     S[0] = A[0]
     sm = A[0].memory
     i = 1
     while (sm < M AND i < n)
       S[i] = A[i]
       sm += A[i].memory
```

3.3.2 Time Complexity

Analyzing this algorithm, including both going over the array to produce the ratio that will guide our greedy choices and choosing the actual solution set, we come up with the complexity of:

$$\sum_{k=1}^{n} k + \sum_{k=1}^{n} k = \mathcal{O}(2n) \tag{5}$$

This algorithm therefore runs in $\mathcal{O}(2n)$ time.

4 Plan of Experiments

The main objective of our experiments is to extrapolate time and efficiency from the algorithms for sufficiently large input, so our plan of experiments is thus:

1. Generate a random array of apps, and a smartphone with n memory.
2. Take a date stamp.
3. Run the Brute Force algorithm for the random array of apps and the smartphone.
4. Take another date stamp, record the difference.
5. Return to step 2 and run the Dynamic Programming algorithm.
6. Likewise, return to step 2 and run the Greedy algorithm.
7. Record the results from the experiment.
8. Return to step 1, repeat 9 additional times, take the average.
9. Run the algorithm again for a scale factor of n (the input size).

5 Team Roles

Illya Starikov Project Management, Development (Brute Force)

Timothy Ott Development (Greedy), Architecture

Claire Trebing Development (Dynamic), Quality Assurance, Documentation

Programming Project I, Second Report

Illya Starikov, Claire Trebing, & Timothy Ott

Due Date: March 17, 2016

1 Abstract

As proposed in the first report, we would like to the compare the effeciency of different memory management methods via a 0-1 Knapsack. We discussed the current memory management methods, the different approaches we would like to use (Brute Force, Dynamic Programming, Greedy Solution) and our plan of implementations.

After implementing the pseudocode provided in Report #1, rigerous testing and extensive running of sample of data, we have completed conclusive evidence to our proposed problem.

In this report we would like to discuss our findings.

2 Implementation

As we discussed in our earlier report we decided to go with a more object-oriented approach to this problem. We created a class called App to represent each application running on our platform. Each instance of the App class contains (as properties[1]) an identifying number, its memory usage, and its cost to recover from storage. These properties are largely randomized in the construction of each instance. For example, every app has a random memory usage in the range of 32 to 1028; the cost is a derived value, taking anywhere from 20 to 50 percent of the memory usage of the App in question; we also have a property called ratio which consists of the cost property divided by the memory property and is calculated at creation of the instance by the constructor, this will be used in our Greedy Algorithm; and lastly each App is given an identifying number based on the index of the cell in the array in which they are initially placed. This ensures that each App will have a unique identifier to help differentiate them whilst checking our solutions against one another.

Initially we considered creating a separate class for our "Smartphone" or the platform on which our Apps would be running but later decided against this as this functionality could easily be represented simply by a list (or array) of all the Apps, cutting down on the amount of code we would need.

[1] Or member variables, depending on your preferred notation.

We implemented our three solutions as one overarching test program that produces arrays of apps in ever increasing input sizes and feeding those arrays to our algorithms one at a time, testing the average runtime over 10 iterations. The general flow of our program is as follows:

First we specify a list of input sizes that our program will act on, we will discuss this is in more detail further along but for now it should be noted that our first input size is 5. Our program then creates an array of newly created Apps of the specified input size[2]. These newly created apps have each of their properties randomly assigned by its constructor. Next, because our Greedy Solution requires the inputs be sorted based on its ratio variable and the other solutions do not, we sort the list using Python's built in sort function.

Now we simply feed the newly sorted list to each of the solution functions in turn, timing the run time of each by taking a timestamp before running the algorithm, again after, then subtracting the two times. We run each program ten times per input and take the average of these times for greater accuracy. Once each solution algorithm has run ten times the program outputs the data (which includes the solution set, the amount of memory freed by the solution, the combined cost of the solution and the average time it took) to two text files in the directory of the program.

It is at this point that our program returns to the top of the loop for the next input. To ensure a adequate spread of input sizes we defined a list of "base inputs" which we initialized as $5, 10, 15$, and 25. Once the program finishes iterating over this list, testing each solution as mentioned above, the program returns to the beginning of the input list and multiplies each by 10 to the power of n, where n is incremented with each new round of the loop back to the beginning of the base inputs. This entire process is placed in a while loop that is set to infinitely repeat so that we are able to get as many data sets as the program can output in the time we have allotted for testing.

3 Experiment

As described above, the tests were run for a twelve hour period getting our input size to $1,000$. Noting that the for input size of $1,000$ it took the Dynamic Programming approach 1334.81 seconds, the total time to get accurate results was *3 hours, 42 minutes, and 28 seconds*[3]. We thought this was a good cut off point.

To ensure accuracy, we decided to run the tests on two independent machines. These tests can be seen on the graphs, but not the data sets. The tests were fairly comparable, so we decided that displaying data from one data set was sufficient.

We noticed that the Brute Force solution was taking a drastically longer time to run (the complexity being $n \times 2^n$). After a sample run, the Brute Force algorithm returned a Memory Error for an input size of 50. Seeing as we were

[2] Again, in our first case this is 5.

[3] 1334.81 seconds $\times 10$ iterations = 13348.1 seconds = 222.47 minutes.

not likely to be able to have sufficient enough time, we decided to limit the Brute Force to a maximum of 25 inputs. Below are the results.

3.1 Results Data Set

Below you can find the results from *one of our test*. As stated above, the Brute Force only goes to 25 inputs. Note that the input column is the Input size of applications, the Algorithm is the type of algorithm we used, the Memory was the goal memory we wanted to produce, the Cost was what we wanted to minimize, the Freed column is the memory that was actually freed (because we could go over the goal if it was an optimal solution) and finally the Time is the time taken (strictly of the algorithm in question) measured in seconds.

Input	**Algorithm**	**Memory**	**Cost**	**Freed**	**Time**
5	Brute	609	206.960013686	769	0.0000652
	Dynamic	609	413.920027372	1538	0.008588
	Greedy	609	206.960013686	769	0.00572586666667
10	Brute	1254.0	306.007083914	1307	0.0023169
	Dynamic	1254.0	913.397426561	3126	0.033041
	Greedy	1254.0	418.629747	1961	0.0220281
15	Brute	1896.2	462.268333337	1980	0.0933534
	Dynamic	1896.2	706.822856172	3129	0.10722175
	Greedy	1896.2	506.866912451	2383	0.0714820333333
25	Brute	2469.0	619.523226819	2479	113.6533167
	Dynamic	2469.0	1204.68191427	4035	56.92951935
	Greedy	2469.0	620.454984919	2562	37.9530137333
50	Dynamic	5421.2	1664.89861527	7120	0.870269
	Greedy	5421.2	1177.03320938	5519	0.43513695
100	Dynamic	9783.2	2809.24740865	11268	3.8071061
	Greedy	9783.2	2392.11409139	9849	1.90355725
150	Dynamic	14995.8	4027.50565053	16594	9.4551353
	Greedy	14995.8	3696.37456362	15462	4.7275798
250	Dynamic	26793.4	6591.81933106	28200	32.8375575
	Greedy	26793.4	6272.26215744	27018	16.41878765
500	Dynamic	52354.6	12547.9234913	53429	192.5988691
	Greedy	52354.6	12365.0250774	52842	96.2994775
1000	Dynamic	107148.2	25671.8344823	109000	1334.8144697
	Greedy	107148.2	25256.6939401	107895	667.4074229

3.2 Results

While looking at the data we got back, we notice a very noticeable trend immediately: the exponential nature of the quasi-exponential nature of the algorithms. When plotting on an input size vs. time graph, we could see even the algorithm with $\mathcal{O}(n)$ (the Greedy Solution) complexity showed some exponential growth. Taking into consideration the RAM Model of Computing is not completely applicable in this situation, the results are more than staggering.

As can be seen in the above figure, although the input follows the pattern of $5n, 10n, 15n, 25n | n \in \mathbb{Z}^+$, the results typically follow a pattern of $2\times$ their predecessor.

When comparing the Time vs. Costs, the trend is also quite exponential. For, as our algorithms get a larger cost, the time is unquestionably larger. When actually looking at the Time vs. Accuracy, the clear cut winner is Greedy. The time saving are immense compared to Dynamic and Brute Force. On certain cases, Greedy can finish before Dynamic and Brute force combined.

We also see that Greedy Solution outperformed the Dynamic in not only time but accuracy. We are unsure of the reasoning behind this, for our algorithm certainly seemed correct on paper.

Tying this back into the real world example, none of these solution would be able to work. Assuming a smartphone of 2 GB RAM[4], figuring an optimal solution would take a bare minimum of 38 seconds (not to mention 2 minutes if we are using the Brute Force.)

While this may not be applicable to the real world, it was an interesting thought experiment.

4 Roles

- Illya Starikov
  - Project Manager
  - Brute Force Development
  - Abstract, Experiment Writement
- Timothy Ott
  - Greedy Algorithm Development
  - Development, Conclusion, Interesting Results Writeup
  - Testing Documents

[4]We are assuming our memory is equivalent to 1 MB. Therefore, 2 GB = 2000 MB

- Claire Trebing
  - Dynamic Programming Development
  - Experiment Writing Up
  - Quality Assurance

5 Interesting Results

In the process of going over the data generated in our experiments we noted several interesting results that we would like to highlight here.

- We noticed that for input sizes under roughly 15 the Brute Force method was both more accurate, which we expected, but also significantly faster than the other two methods. When we take into consideration that the Brute Force's complexity is $\mathcal{O}(n2^n)$ and the Greedy Solution is $\mathcal{O}(2n)$, this is an excellent highlight to the fact that the growth of the function is far more important than its initial run speeds.
- There is an interesting anomaly in the data set for the input size of 25. At this point the run times for each algorithm spike noticeably only to then immediately dip once again. What's more interesting is that as we mentioned earlier we ran these experiments twice, independently of one another on two different machines and this anomaly occurred in both rounds of experimentation. So, what is so special about 25 inputs? The short answer is we are not sure. Our best guess is that the increase and subsequent drop are somehow related to the Brute Force algorithm since that is where we stopped testing that particular algorithm. Particularly we think this may have been the fault of the language's garbage collection functionality but without further testing we cannot be sure.
- Also of note is that while initially our dynamic approach was a good deal less accurate than either the Brute Force or even the Greedy approach, as the input sizes increased the accuracy of the dynamic algorithm also increased.

6 Conclusion

Memory management is a complex issue with many interesting solutions of varying effectiveness. With this project we explored three approaches to solve this issue using a smartphone's memory management system as our testing ground. Our solutions took the forms of:

- A brute force approach, comparing every possible subset of solutions to find an optimal choice at the expense of a high complexity and eventual run-time;

- A dynamic programming solution which built a table of solutions to effectively try every solution but with a lower complexity due to being able to look up values that were already computed; and
- A greedy algorithm approach which took a ratio of the cost and memory and produced a solution that was simply taking the lowest ratio apps until the memory requirements were met, in effect trading the accuracy of the other two approaches for a significant increase in speed.

Through our data gathered during experimentation we see that these early predictions hold fairly consistently, despite some unaccounted for inaccuracy on the part of our dynamic solution. While it is tempting to label one solution as the "best" or "optimal" solution, this would be misleading as each approach shines under different circumstances. For instance, for small input sizes the brute force algorithm seemed to exceed both other solutions in terms of both accuracy and speed. For higher input sets the greedy algorithm was the clear winner in terms of speed but theoretically would lose out on accuracy to the dynamic solution every time. Thus, we see that is important to tailor your solutions to the problem and not simply favor one approach over another arbitrarily.

Programming Project II, First Report

Illya Starikov, Claire Trebing, Timothy Ott

Due Date: April 22, 2016

1 Abstract

Social Networks have revolutionized the way we communicate, meet others, consume information, and essentially influence our day-to-day lives. To show Social Network's prominence, here are the percentages of online adults who use social media:

Facebook : 71% Adults

Twitter : 23% Adults

Instagram : 26% Adults

Pinterest : 28% Adults

LinkedIn : 28% Adults

This is unprecedented. Seeing as an overwhelming majority of adults have some sort of social media account, this can be used to model real world relationships — through social graphs.

In this experiment we would like to examine the social graph through the follower-followee relationship.

2 Introduction and Motivation

As stated above, social networks play a dominant role in our lives. Through social graphs, we can examine real world relationships.

There are many reasons for this to be valuable, such as common interest, community detection, influence amongst followers, etc. For this experiment, we would just like to test on the following measures:

Degree Distribution Test the direct amount of followers compared to followees a person has.

Shortest Path Distribution Test the degree of separation between a follower-followee.

Graph Diameter Test the breadth of a community.

Closeness Centrality Test the dependence of a degree of separation on a singular person.

Betweenness Centrality Distribution Same as previous.

Community Detection Based on Closeness Centrality, find the diameter of a community.

This will give us a reasonable dataset for the relationship of a community in a social graph.

3 Proposed Solutions

3.1 Uweighted In Degree Distribution

Unweighted In Degree Distribution goes through each entry to determine what the origin of each edge is. An array with cells for each possible vertex (total) is created and once the origin is determined the respective cell in total is increased by one. After all of the entries have been checked, the function goes through the totals.

The functions begins by assuming `total[0]` is the maximum number of In Degree edges and 0 is added to `arrayMax`. `arrayMax` is the array which holds all of the vertexes with the maximum number of in degree edges. Each entry in total is compared to max. If the total entry is larger than the current max, then the current max is replaced with current total. The `arrayMax` list is cleared and the current total origin is added to the `arrayMax`. If the current total is equal to the max, then the origin of current total is added to `arrayMax`.

```
void Unweighted_InDegree_Distribution(&int arrayMax[])
{
    N = the number of vertices
    int * total;
    total = new int [maxVertice];

    for(i = 0 to N){
        total[origin of i]++;
    }
    int degreeMax = total[0];
    add 0 to arrayMax

    for(i = 1 to maxVertice){
        if(total[i] > degreeMax){
            degreeMax = total[i];
            clear arrayMax
            add i to arrayMax
```

```
        }
        if(total[i] == degreeMax){
            add i to arrayMax
        }
    }
    delete[] total;
    return;
}
```

The time complexity of this function is $2n = \mathcal{O}(n)$. The function must traverse through the array twice before finishing.

3.2 Unweighted Out Degree Distribution

Unweighted Out Degree Distribution goes through each entry to determine what the destination of each edge is. An array with cells for each possible vertex (total) is created and once the destination is determined the respective cell in total is increased by one. After all of the entries have been checked, the function goes through the totals. The functions begins by assuming `total[0]` is the maximum number of Out Degree edges and 0 is added to `arrayMax`. `arrayMax` is the array which holds all of the vertexes with the maximum number of out degree edges. Each entry in total is compared to max. If the total entry is larger than the current max, then the current max is replaced with current total. The `arrayMax` list is cleared and the current total destination is added to the `arrayMax`. If the current total is equal to the max, then the destination of current total is added to the `arrayMax`.

```
void Unweighted_OutDegree_Distribution(&int arrayMax[])
{
    N = the number of vertices
    int * total;
    total = new int [maxDestination];

    for(i = 0 to N){
        total[Destination of i]++;
    }
    int degreeMax = total[0];
    add 0 to arrayMax

    for(i = 1 to maxOrigin){
        if(total[i] > degreeMax){
            degreeMax = total[i];
            clear arrayMax
            add i to arrayMax
        }
        if(total[i] == degreeMax){
```

```
            add i to arrayMax
        }
    }
    delete[] total;
    return;
}
```

3.2.1 Complexity Analysis

The time complexity of this function is $2n = \mathcal{O}(n)$. The function must traverse through the array twice before finishing.

3.3 Weighted In Degree Distribution

Weighted In Degree Distribution goes through each entry to determine what the origin of each edge is and the sum of the weighted edges. An array with cells for each possible vertex (total) is created and once the origin is determined the respective cell in total is increased by weight of that edge. After all of the entries have been checked, the function goes through the totals.

The functions begins by assuming `total[0]` is the maximum number of Weighted In Degree edges and 0 is added to `arrayMax`. `arrayMax` is the array which holds all of the vertexes with the maximum number of weighted in degree edges. Each entry in total is compared to max. If the total entry is larger than the current max, then the current max is replaced with current total. The `arrayMax` list is cleared and the current total origin is added to the arrayMax. If the current total is equal to the max, then the origin of current total is added to the `arrayMax`.

```
void Weighted_InDegree_Diribution(&int arrayMax[])
{
    N = the number of vertices
    int * total;
    total = new int [maxOrigin];

    for(i = 0 to N){
        total[Origin of i]+= (Weight of i);
    }
    int degreeMax = total[0];
    add 0 to arrayMax

    for(i = 1 to maxOrigin){
        if(total[i] > degreeMax){
            degreeMax = total[i];
            clear arrayMax
            add i to arrayMax
        }
```

```
        if(total[i] == degreeMax){
            add i to arrayMax
        }
    }
    delete[] total;
    return;
}
```

3.3.1 Complexity Analysis

The time complexity of this function is $2n = \mathcal{O}(n)$. The function must traverse through the array twice before finishing.

3.4 Weighted Out Degree Distribution

Weighted Out Degree Distribution goes through each entry to determine what the destination of each edge is and the weight of those edges. An array with cells for each possible vertex (total) is created and once the destination is determined the respective cell in total is increased by the value of that edge. After all of the entries have been checked, the function goes through the totals. The functions begins by assuming `total[0]` is the maximum number of Weighted Out Degree edges and 0 is added to `arrayMax`. `arrayMax` is the array which holds all of the vertexes with the maximum number of weighted out degree edges. Each entry in total is compared to max. If the total entry is larger than the current max, then the current max is replaced with current total. The `arrayMax` list is cleared and the current total destination is added to the arrayMax. If the current total is equal to the max, then the destination of current total is added to the `arrayMax`.

```
void Weighted_OutDegree_Diribution(&int arrayMax[])
{
    N = the number of vertices
    int * total;
    total = new int [maxDestionation];

    for(i = 0 to N){
        total[Destination of i]+= (Weight of i);
    }
    int degreeMax = total[0];
    add 0 to arrayMax

    for(i = 1 to maxDestination){
        if(total[i] > degreeMax){
            degreeMax = total[i];
            clear arrayMax
            add i to arrayMax
```

```
        }
        if(total[i] == degreeMax){
            add i to arrayMax
        }
    }
    delete[] total;
    return;
}
```

The time complexity of this function is $2n = O(n)$. The function must traverse through the array twice before finishing.

3.5 Shortest Path

For our Shortest Path algorithm we decided to implement the Floyd-Warshall algorithm. We chose this algorithm because of the need to examine the shortest path for all pairs of vertices as well as the lower complexity of the dynamic programming solution. The Floyd-Warshall algorithm creates an array of VxV size keeping a running tally of the shortest path while adding verticies to the list of possible intermediate points on those paths.

```
V = the number of vertices
distance = new array[V][V]
Initialize distance to -1   //since there are no negative weights this will
                            //represent infinity
Floyd(V):
    for each vertex v:
        distance[v][v] = 0
    for each edge (u,v)
        distance[u][v] = w(u,v) //the weight of the edge (u,v)
    for k from 1 to V
        for i from 1 to V
            for j from 1 to V
              if distance[i][j] > distance[i][k] + distance[k][j]
                distance[i][j] = distance[i][k] + distance[k][j]

shortest = new array[100]
initialize shortest to 0
for i from 1 to V
    for j from 1 to V
        if distance[i][j] > 0
          if distance[i][j] > shortest.length
              temp = new array[distance[i][j]+10]
              for i from 0 to shortest.length
                  temp[i] = distance[i]
              delete distance
```

```
                distance = temp
            shortest[distance[i][j]]++

for i from 1 to shortest.length
    if shortest[i] > 0
        output shortest[i]
```

3.5.1 Complexity Analysis

As we know, the Floyd-Warshall algorithm operates on n^3 time. Combining this with the algorithm to iterate over the resulting V x V matrix results in a $n^2 + n^3$ complexity — or, $\mathcal{O}(n^3)$ time.

3.6 Unweighted Graph Diameter

Our solution first creats an $N \times N$ array to hold all of the shortest paths from each vertex to every other vertex. For an unweighted graph the shortest path from $i \rightarrow j$ is equal to the shortest path from $j \rightarrow i$. After creating and filling the matrix, the function assumes that the value at `distance[0][1]`. The function then tests every value in the matrix and compares it to the max. If the value is greater than the max then the max is replaced with `distance[i][j]`. Clear the `GraphDiameterList` and then add (i, j) to the graph diameter list. If `distance[i][j]` is equal to max then add (i, j) to the `GraphDiameterList`.

The pseudocode is as follows.

```
Unweighted Graph Diameter {
    N = the number of vertices
    distance = new array[N][N]
    all distances start at -1

    for(i = 0 to N){
        for(j = i+1 to N){
            distance[i][j] = shortest path(i, j)
        }
    }

    max = Diameter[0][1]
    Add (0,1) to GraphDiameterList

    for(i = 0 to N){
        for(j = 1 to N){
            if(distance[i][j] > max){
                clear GraphDiameterList
                max = distance[i][j]
                add (i,j) to GraphDiameterList
            }
```

```
            if(distance[i][j] = max){
                add (i,j) to GraphDiameterList
            }
    }
    return GraphDiameterList
}
```

3.6.1 Complexity Analysis

The time complexity of this code is $\mathcal{O}(n^2)$. You must go through the matrix twice to complete this function.

3.7 Weighted Graph Diameter

This functions first created an $N \times N$ array to hold all of the shortest paths from each vertex to every other vertex. After creating and filling the matrix, the solution assumes that the value at `distance[0][1]`. The function then tests every value in the matrix and compares it to the max. If the value is greater than the max then the max is replaced with `distance[i][j]`. Clear the `GraphDiameterList` and then add (i, j) to the graph diameter list. If `distance[i][j]` is equal to max then add (i, j) to the `GraphDiameterList`.

The pseudocode is as follows:

```
Weighted Graph Diameter{
    N = the number of vertices
    distance = new array[N][N]
    all distances start at -1

    for(i = 0 to N){
        for(j = 0 to N){
            if(i != j){
                distance[i][j] = shortest path(i, j)
            }
        }
    }

    max = Diameter[0][1]
    Add (0,1) to GraphDiameterList

    for(i = 0 to N){
        for(j = 0 to N){
            if(i != j)
            if(distance[i][j] > max){
                clear GraphDiameterList
                max = distance[i][j]
                add (i,j) to GraphDiameterList
```

```
            }
            if(distance[i][j] = max){
                add (i,j) to GraphDiameterList
            }
    }
    return GraphDiameterList
```

3.7.1 Complexity Analysis

The time complexity of this code is $\mathcal{O}(n^2)$. You must go through the matrix twice to complete this function.

3.8 Closeness Centrality

The Closeness Centrality is defined as $C_c(i) = [\sum_{j=1}^{N} d(i,j)]^{-1}$. So in order to find the Closeness Centrality we first must find the sum of all the shortest paths for each vertices. Once we find this sum we simply record it as a float by dividing one by the sum and storing it in an array of Closeness values.

```
sum = 0
closeness = new array[100]
initialize closeness to 0
for i from 0 to V
    for j from 1 to V
        sum += distance[i][j]
    closeness[i] = 1/sum
    output closeness[i]
    sum = 0
```

3.8.1 Complexity Analysis

Because we are iterating over the entire matrix resulting from the Floyd-Warshall algorithm of $V \times V$ size. We arrive at a complexity of $\mathcal{O}(n^2)$.

3.9 Undirected Betweenness Centrality Distribution

This algorithms takes a vertex i from the main function. The function then looks at every shortest path possible. Starting at vertex 0 the function and going to the largest vertex. If i is the starting point (j) or the ending point (k) it is ignored. K is defined as $j+1$ If the shortest path because in undirected graphs, the shortest distance from $j \to k$ == the shortest distance from $k \to j$. If the shortest distance from j to k includes the vertex i, the value of the betweenness is increased by 1. The total betweens is then returned to the main function.

```
Undirected Betweenness Centrality Distribution of Vetex i(vertex i){
    N = max vertex
    B = betweenses = 0
```

```
    for(j = 0 to N){
        if(i != j){
            for(k = j+1 to N){
                if(i != k){
                    shortestpath between j and k
                    if i is included, increase betweeneses by 1
                }
            }
        }
    }

    return B;
}
```

This function has a complexity of n. Each vertex check every vertex larger than itself to determine all of the shortest paths. The first vertex must check nearly other ever vertex, except i. This gives it a complexity of $\mathcal{O}(n)$.

3.10 Directed Betweenness Centrality Distribution

This algorithm takes a vertex i from the main function. The function then looks at every shortest path possible. Starting at vertex 0 the function and going to the largest vertex. If i is the starting point (j) or the ending point (k) it is ignored. The path is also ignored if j and k are equal. If the shortest distance from j to k includes i, then the betweenness is increased by 1. The total betweenness is then returned to the main function.

```
Directed Betweenness Centrality Distribution of Vertex i(vertex i){
    N = max vertex
    B = betweenses = 0
    for(j = 0 to N){
        if(i != j){
            for(k = 0 to N){
                if(i != k && j != k){
                    shortest path between j and k
                    if i is included, increase betweenses by 1
                }
            }
        }
    }
}
```

3.10.1 Complexity Analysis

This function has a complexity of n^2. Each vertex must check every other vertex to determine all of the shortest paths. This gives it a complexity of $\mathcal{O}(n^2)$.

3.11 Undirected Unweighted Betweenness Centrality Distribution

This algorithm takes an edge e from the main function. The function then looks at every shortest path possible. Starting at vertex 0 the function and going to the largest vertex. K is defined as j + 1 because the shortest path in undirected graphs, the shortest distance from j -¿ k == the shortest distance from k -¿ j. If the shortest distance from j to k includes the edge e, the value of the betweenness is increased by 1. The total betweenness is then returned to the main function.

```
Undirected Unweighted Betweenness Centrality Distribution of Edge e(edge e){
    N = max vertex
    B = betweenses = 0
    for(j = 0 to N){
        for(k = j+1 to N){
            shortest path between j and k
            if e is included, increase betweeneses by 1
            }
        }
    }
    return B;
}
```

3.11.1 Complexity Analysis

This function has a complexity of n. Each vertex check every vertex larger than itself to determine all of the shortest paths. The first vertex must check nearly other ever vertex. This gives it a complexity of n.

3.12 Directed Unweighted Betweenness Centrality Distribution of Edge

This algorithm takes an edge e from the main function. The function then looks at every shortest path possible. Starting at vertex 0 the function and going to the largest vertex. If the shortest distance from j to k includes the edge e, the value of the betweenness is increased by 1. The total betweenness is then returned to the main function.

```
Directed Unweighted Betweenness Centrality Distribution of Edge e(edge e){
    N = max vertex
    B = betweenses = 0
    for(j = 0 to N){
        for(k = 0 to N){
            if(j != k){
                shortest path between j and k
                if e is included, increase betweeneses by 1
```

```
            }
        }
    }
    return B;
}
```

3.12.1 Complexity Analysis

This function has a complexity of n^2. Each vertex must check every other vertex to determine all of the shortest paths. This gives it a complexity of $\mathcal{O}(n^2)$.

3.13 Undirected Weighted Betweenness Centrality Distribution of Edge e

This algorithm takes an edge e from the main function. The function then looks at every shortest path possible. Starting at vertex 0 the function and going to the largest vertex. K is defined as j + 1 because the shortest path in undirected graphs, the shortest distance from $j \to k$ == the shortest distance from $k \to j$. If the shortest distance from j to k includes the edge e, the value of the betweenness is increased by 1. The total betweenness is then returned to the main function divided by the weight of e.

```
Undirected Weighted Betweenness Centrality Distribution of Edge e(edge e){
    N = max vertex
    B = betweenness = 0
    for(j = 0 to N){
        for(k = j+1 to N){
            shortest path between j and k
            if e is included, increase betweenness by 1
            }
        }
    }
    return B/(weight of e);
}
```

3.13.1 Complexity Analysis

This function has a complexity of n. Each vertex check every vertex larger than itself to determine all of the shortest paths. The first vertex must check nearly other ever vertex. This gives it a complexity of $\mathcal{O}(n)$.

```
Directed Weighted Betweenness Centrality Distribution of Edge e(edge e){
    N = max vertex
    B = betweenness = 0
    for(j = 0 to N){
        for(k = 0 to N){
            if(j != k){
```

```
                shortest path between j and k
                if e is included, increase betweeneses by 1
            }
        }
    }
    return B/(weight of edge e);
}
```

This algorithms takes an edge e from the main function. The function then looks at every shortest path possible. Starting at vertex 0 the function and going to the largest vertex. If the shortest distance from j to k includes the edge e, the value of the betweenness is increased by 1. The total betweenness is then returned to the main function divided by the edge e.

3.13.2 Complexity Analysis

This function has a complexity of n^2. Each vertex must check every other vertex to determine all of the shortest paths. This gives it a complexity of $\mathcal{O}(n^2)$.

3.14 Community Detection

To detect communities of vertices within the data set we set out to remove those edges that are most likely between those communities to make those communities more apparent. To do this we take the betweenness centrality of edges and sort them in decending order. Then we remove the edge with the highest value (or edges if there are more than one of the same value). Once this is complete we simply recalculate the matrix of shortest paths using the Floyd-Warshall algorithm and the graph diameter using the algorithm detailed above. We repeat this for a total of five times and the resulting diameters are our likely communities.

```
UWBetween[] = Unweighted Betweenness Centrality Edges
V = Original Network

For k from 0 to 4:
    DescendSort(UWBetween)
    UWBetween[0] = x
  while (UWBetween[0] == x)
      V.remove(UWBetween[i])
      UWBetween.remove(UWBetween[0])
  Floyd(V)            //Run Floyd-Warshall Algorithm on revised data set
  Diameter(V)         // Calculate and output max diameter based on new matrix from above
  Betweenness(V)      // Calculate Betweenness centrality based on
                      // new matrix giving us a new UWBetween array
```

3.14.1 Complexity Analysis

Because the Community detection algorithm calls the algorithms to find all shortest paths, the diameter and the unweighted betweenness centrality edges within it and since we are executing this algorithm a total of five times, the complexity of this algorithm is five times the sum of the complexities of these algorithms.

4 Plan of Experiments

The major purpose of our experiment is to dissect and examine networks, so we propose the following plan of experiments:

1. Extrapolate data and store in a relevant data structure — in our case, a sorted `map<int, vector<pair<int, double>>>`
    - The `int` is the key to be used, signifying the origin.
    - The `vector<pair>` holds all the adjacent edges.
2. Iterate over the entirety of the data structure to determine degree distribution.
    - For *weighted and unweighted out degree*, it is simply counting the the `vector` size with respect to each key.
    - For *weighted and unweighted in degree*, making an efficient algorithm is still difficult — not only is by default $\mathcal{O}(n)$ but there is an efficient way of finding where the edges lead to. We accomplish this by a Binary Search Tree. By iterating over every vertex and storing their destination in a binary search tree, we have achieved a sufficient algorithm.
3. Detect the shortest path via the Floyd-Warshall algorithm for directed graphs.
    - Make the graph undirected by making it symmetric about the $a_{i,i}$ elements $\forall i \in$ edges.
    - Test again.
4. Detect closeness centrality and betweenness.
5. Implement community detection.
6. Output results to user.

5 Team Roles

- Illya Starikov
  - Project Manager
  - Official Write-up
- Timothy Ott
  - Pseudocode Write-up
  - Algorithm Analysis
- Claire Trebing
  - Pseudocode Write-up
  - Algorithm Analysis

Programming Project II, Second Report

Illya Starikov, Claire Trebing, Timothy Ott

Due Date: May 1st, 2016

1 Abstract

As stated in the prior report, social networks have revolutionized the way we communicate. Seeing as social *networks* are a real-life representation of a graph, we would like to more closely examine them. Particularly, we would like to test

- Degree Distribution
- Shortest Path Distribution
- Graph Diameter
- Closeness Centrality
- Betweenness Centrality Distribution
- Community Detection

This report will showcase our results — more specifically, we would like to discuss, in detail, our implementation and experiments. Also, we will list any relevant, interesting results we obtain.

2 Implementation

Our implementation is as follows, from a higher level:

1. Get and parse graph input.
   (a) Because our data was given in the form of a `csv`, we decided to just pipe that input directly to a `vector<string>`.
   (b) We then pass said `vector<string>` to a function that parses using C++ string functions.
   (c) We pipe the parsed data to a data structure of `map<int, vector<pair<int, double>>>`
      - The `int` is the key for retrieval of the `vector`

- The `vector<pair<int, double>>` stores a `vector` of the edges, in pairs — where the pair `<int, double>` are proportional to the target vertex and weight.

2. Move on to calculating the out degree of the vertexes.
   (a) Initialize a `map` of `<int, int>` for unweighted and `<int, double>` for unweighted.
   (b) For both weighted and unweighted simply use the source as the `key`.
   (c) For unweighted, use the `size()` method of the vector class to determine the out degree[1].
   (d) For weighted, sum the `second` property of the `pair` s in the `vector` — note that the second property of `pair` is the weight. This gives a total weight.
3. Move to calculating the in degree of the vertexes.
   (a) This is done almost the same way, except there is a weight `map`.
   (b) Iterate over the entirety of our data structure[2], and store where the target vertex points to in all the edges in the weight `map`.
4. Calculate shortest path via the Floyd-Warshall algorithm.
   (a) Initialize an adjacency matrix — a `vector<vector<int>>`
      - Default all values to infinity — in our case, 999999.
   (b) Copy over data from our `map` to said adjacency matrix.
      - If unweighted and an edge exists, default to 1.
   (c) If requested an undirected shortest path, make the graph indirected.
      - This is done by making a mirror image of the adjacency matrix A, by setting $\forall i, j \in A, A_{i,j} = A_{j,i}$. Just copying over the diagonal.
   (d) Run the Floyd-Warshall algorithm, with triple C-Style `for` loops.
5. Implemented Graph Diameter
   (a) Took the map and using the Floyd Walsh algorithm, found the shortest paths between each vertex
   (b) Begin by assuming that the path from 0 to 1 is the longest path
   (c) Compare each other path to the longest path. If a longer path is found, replace the longest path with that path. If a path of equal length is found, add the path to the list of paths.

[1] Remember, the key return the a `vector` of pairs. The number of pairs are directly proportional the out degree.

[2] An adjacency map of sorts, `map<int, vector<pair<int, double>>>`.

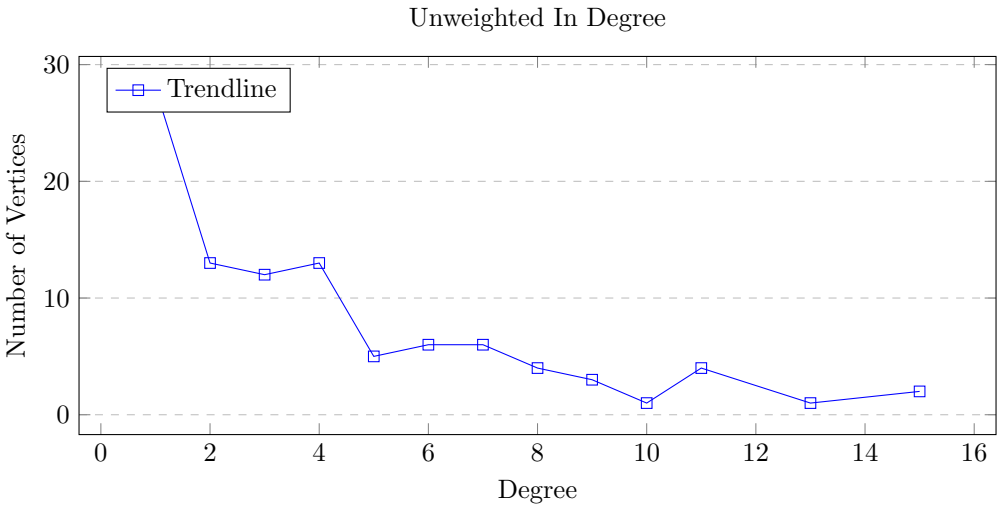
(d) Output the list of all paths of the maximum length.

6. Next we endeavor to find the various closeness centrality of the vertices.
   (a) Initialize one vector to hold the solutions set and four separate vectors to contain the vertices that correspond to the four vertices with the highest closeness centrality values.
   (b) Add each edge weight for each vertex together (iterating over each vertex and each edge) and pushing the result into the solution set vector.
   (c) After this has been pushed we check the value against the recorded highest four values to see if this vertex either equals or exceeds those four values and change our recorded values/vertices accordingly.
   (d) Finally, we output both our results for highest closeness centrality values and the data needed to construct the graph.
7. Calculate Betweenness Centrality
   - Betweenness Vertex
     (a) Take the map and use the Floyd Walsh algorithm to determine the shortest paths between each vertex
     (b) While constructing the paths, if a vertex is used in a path, increase the betweenness value for that vertex.
     (c) Go through the array of betweenness values and find the one with the maximum value. If two are equal betweenness, record them both.
     (d) Output the results.
   - Betweenness Edge
     (a) Take the map and use the Floyd Walsh algorithm to determine the shortest paths between each vertex
     (b) While constructing the paths, if an edge is used in a path, increase the betweenness value for that edge.
     (c) Go through the array of betweenness values and find the one with the maximum value. If two are equal betweenness, record them both.
8. Lastly we must run an algorithm to detect the communities present within the graph.
   (a) Take the results from the Unweighted Edge Betweeness algorithm and take the maximum value.
   (b) Remove the edge that corresponds to this result from the original network.
   (c) Re-run the Shortest Path and Graph Diameter Algorithms on the new network.
   (d) Repeat these steps until a total of five deletions from the original network have been completed.

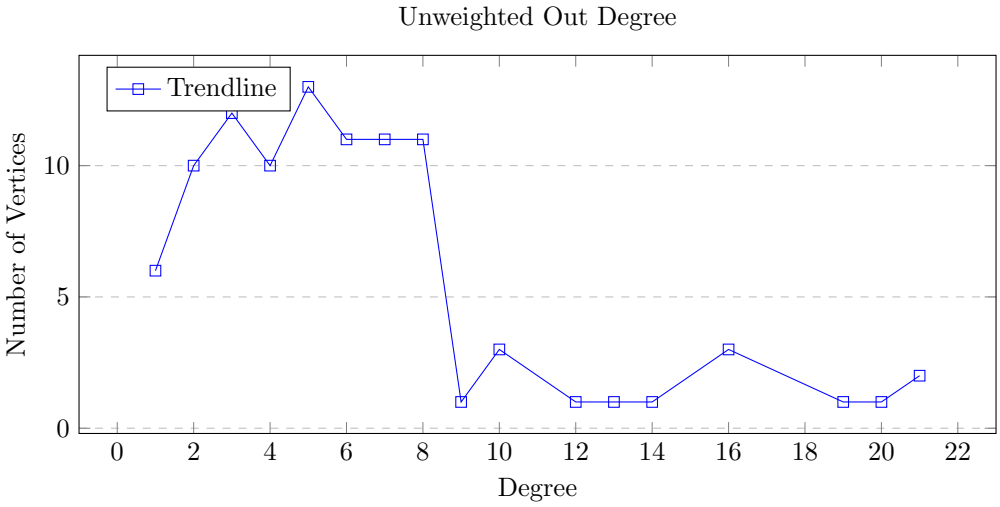
3 Experiments

3.1 Degree Distribution

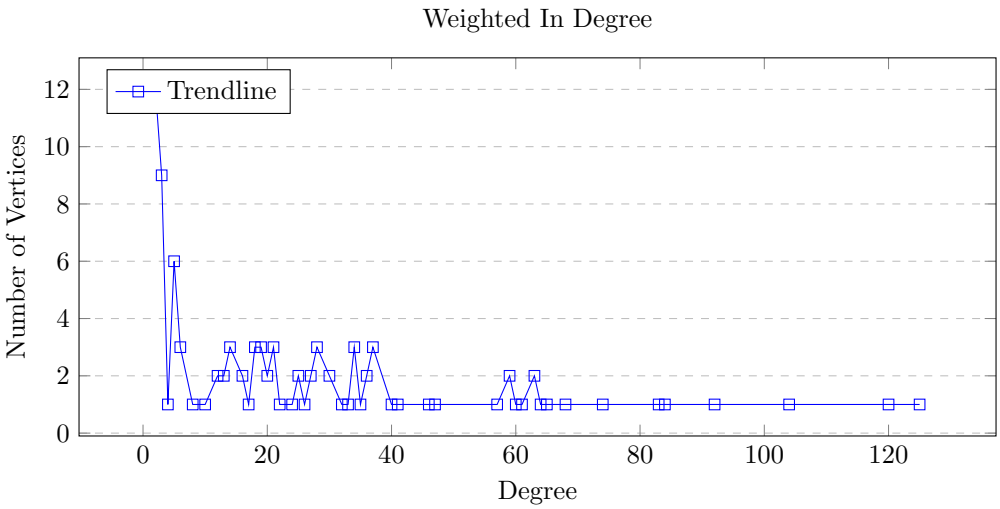
3.1.1 Unweighted In Degree



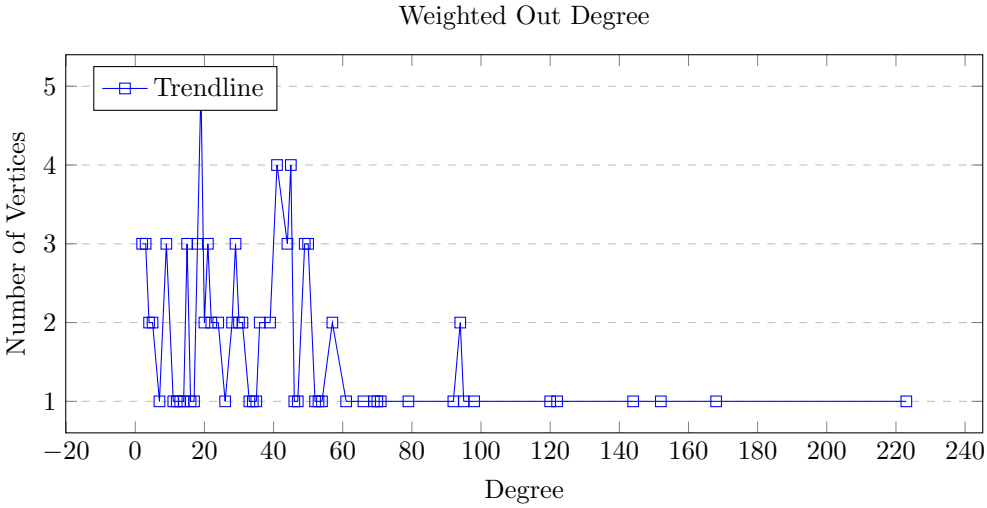
3.1.2 Unweighted Out Degree



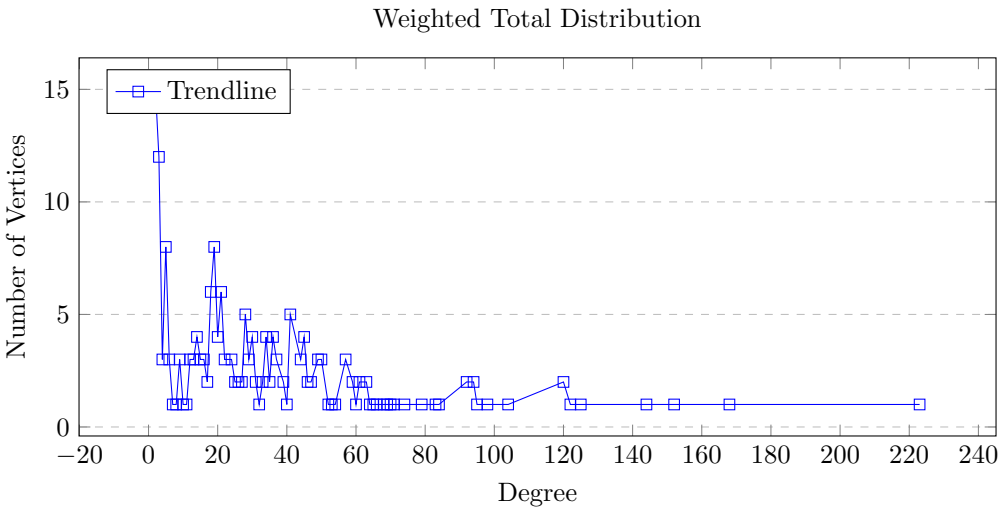
3.1.3 Weighted In Degree



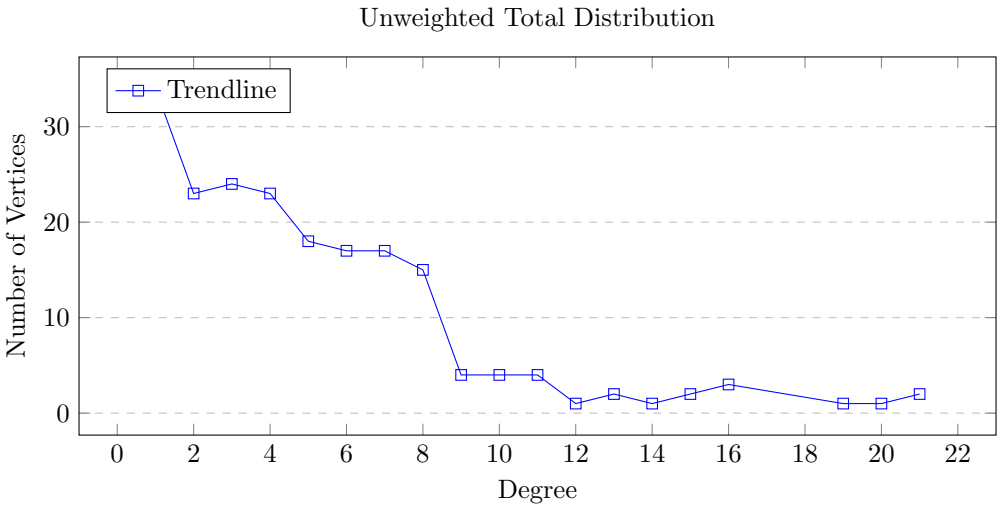
3.1.4 Weighted Out Degree



3.1.5 Weighted Total Distribution



3.1.6 Unweighted Total Distribution



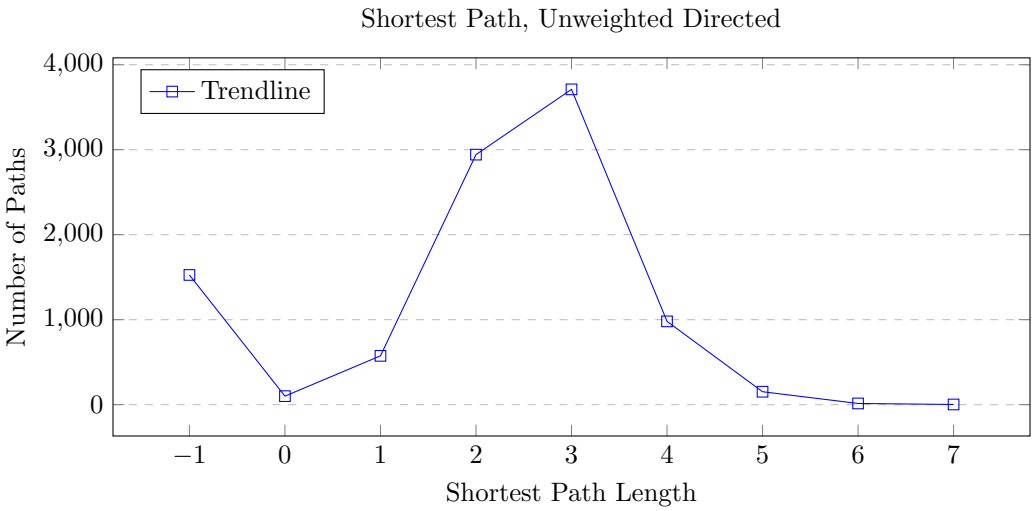
3.1.7 Highest In Degree

Out Degree, Unweighted: **21**	• 5 • 7
Out Degree, Weighted: **223**	• 3
In Degree, Unweighted: **15**	• 70 • 78
In Degree, Weighted: **125**	• 70
Total Degree, Unweighted: **22**	• 5 • 7
Total Degree, Weighted: **226**	• 3

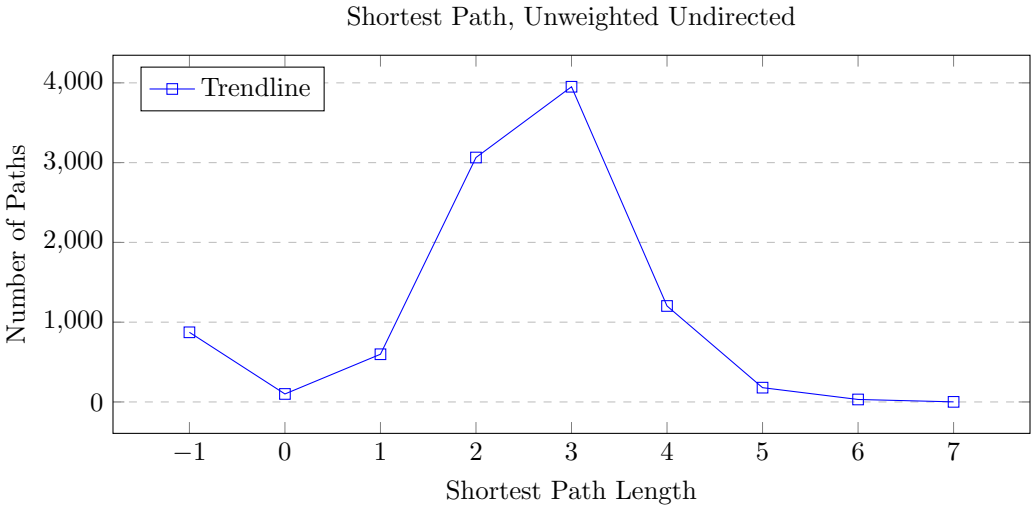
3.2 Shortest Path

Please not that -1 corresponds to a path between two vertices not existing.

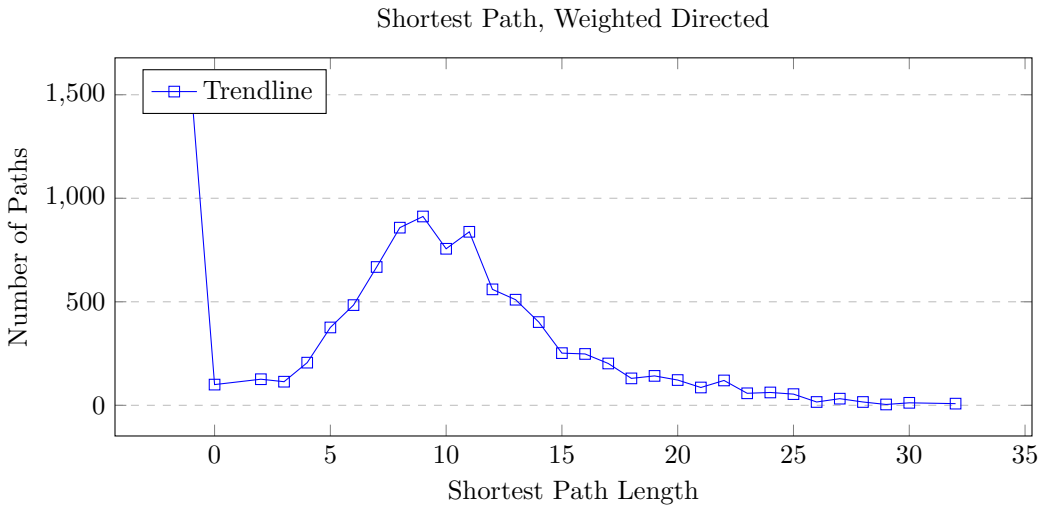
3.2.1 Shortest Path, Unweighted Directed



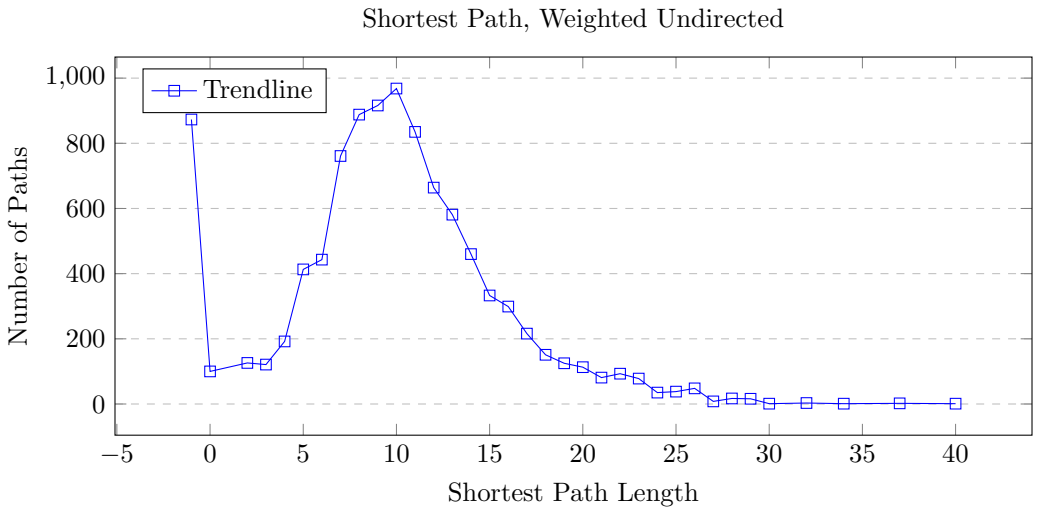
3.2.2 Shortest Path, Unweighted Undirected



3.2.3 Shortest Path, Weighted Directed

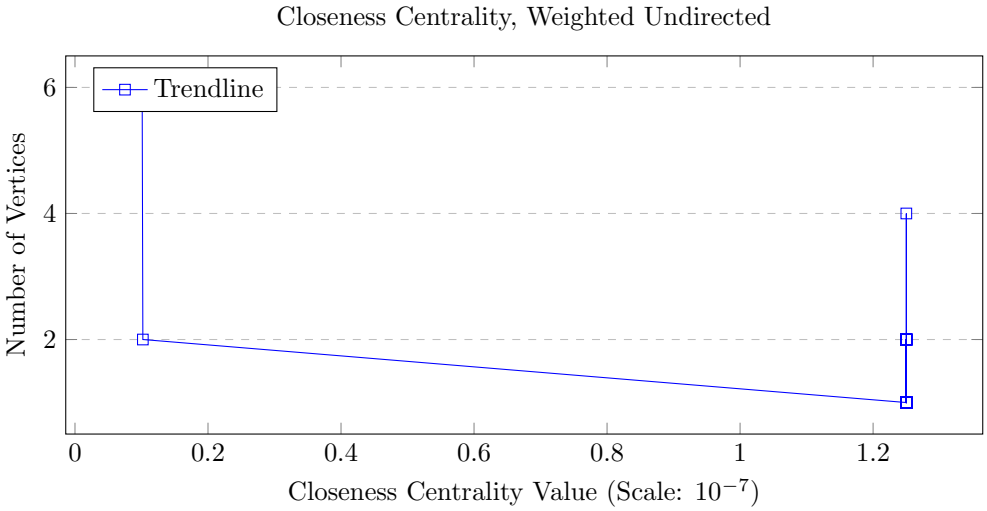


3.2.4 Shortest Path, Weighted Undirected

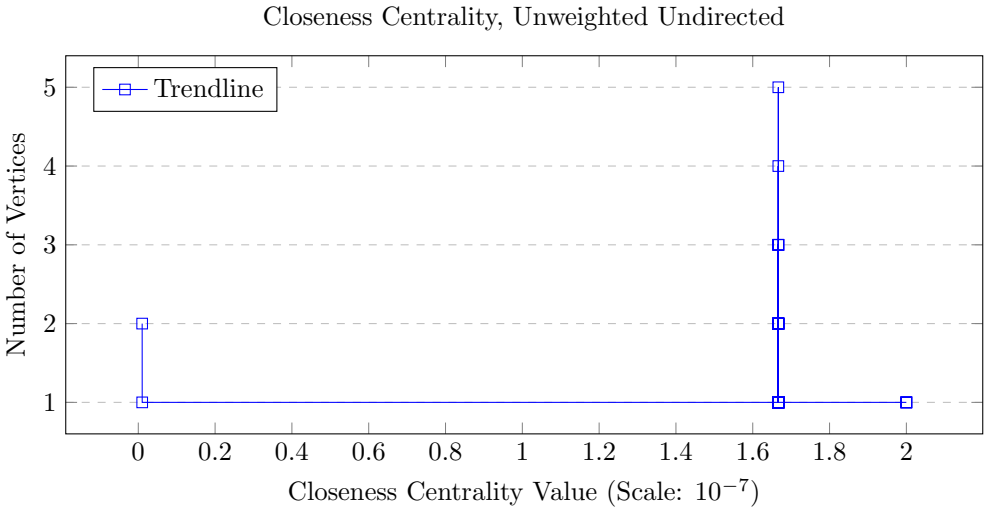


3.3 Closeness Centrality

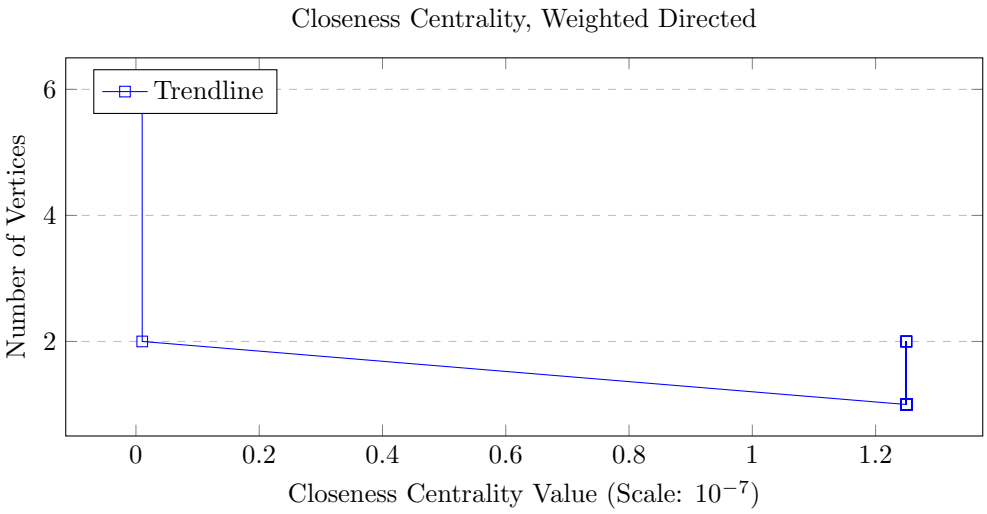
3.3.1 Closeness Centrality, Unweighted Directed



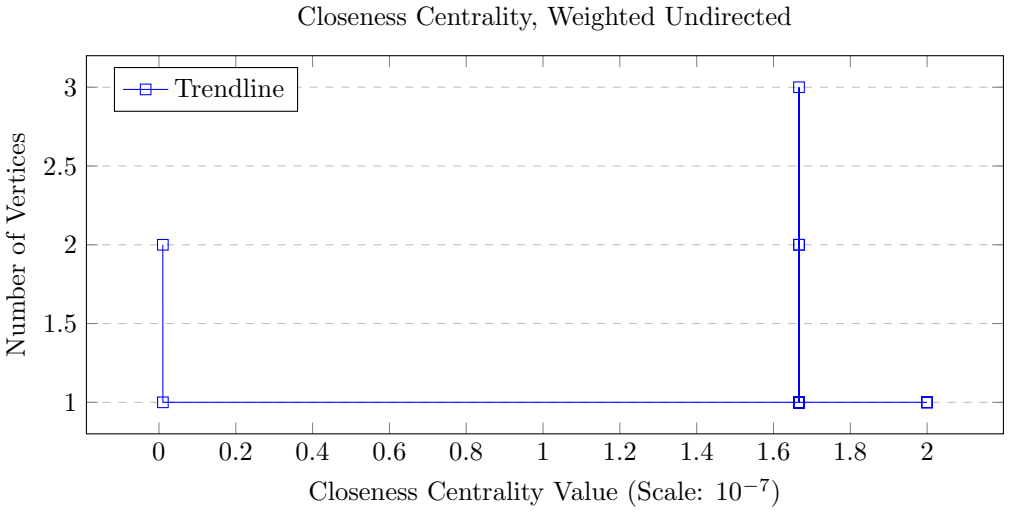
3.3.2 Closeness Centrality, Unweighted Undirected



3.3.3 Closeness Centrality, Weighted Directed



3.3.4 Closeness Centrality, Weighted Undirected

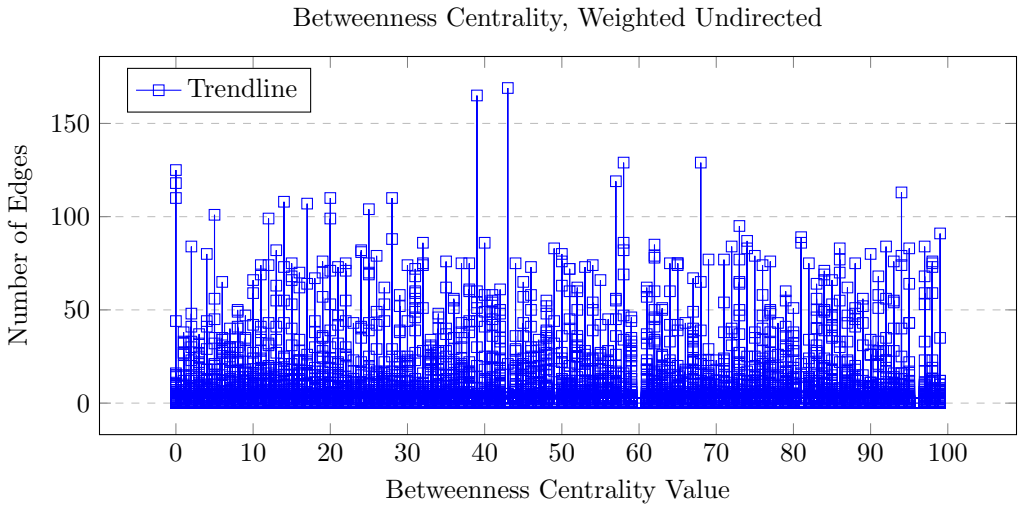


3.3.5 Highest Closeness Centrality

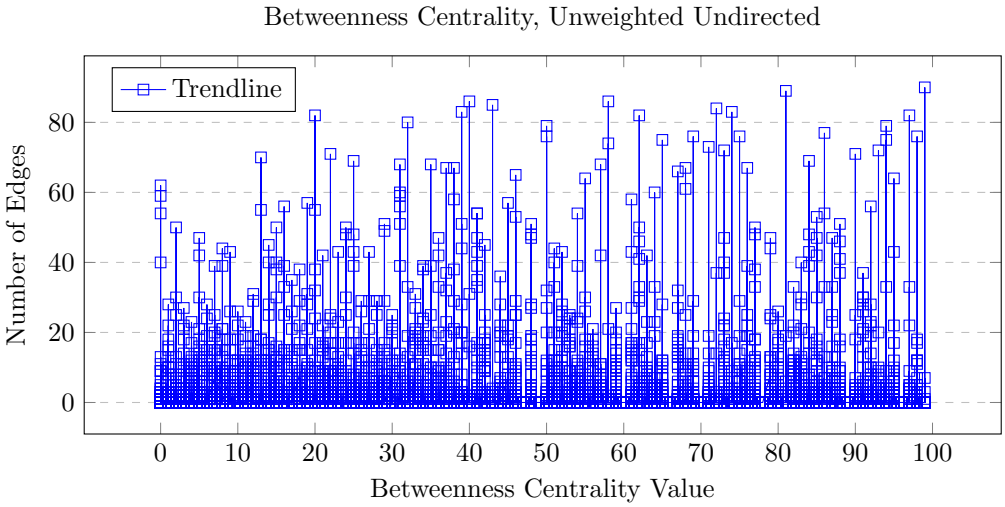
Type	Value	Vertex
Unweighted, Directed	0.000000124997e	5
	0.000000124997	7
	0.000000124997	1
	0.000000124997	3
Weighted, Directed	0.00000012498937590	5
	0.00000012498914157	64
	0.00000012498907908	52
	0.00000012498875101	11
Weighted Undirected	0.00000019996612574	47
	0.00000019996412644	0
	0.00000019996168734	70
	0.00000019995872852	89
Unweighted Undirected	0.00000019999132038	0
	0.00000019999036046	89
	0.00000019998952055	70
	0.00000019998928057	47

3.4 Betweenness Centrality

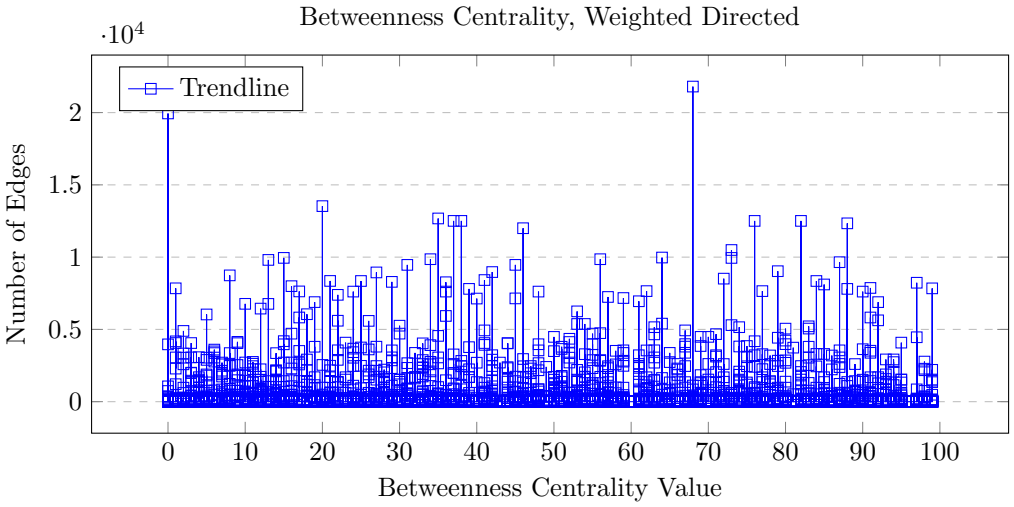
3.4.1 Betweenness Edge, Unweighted Directed



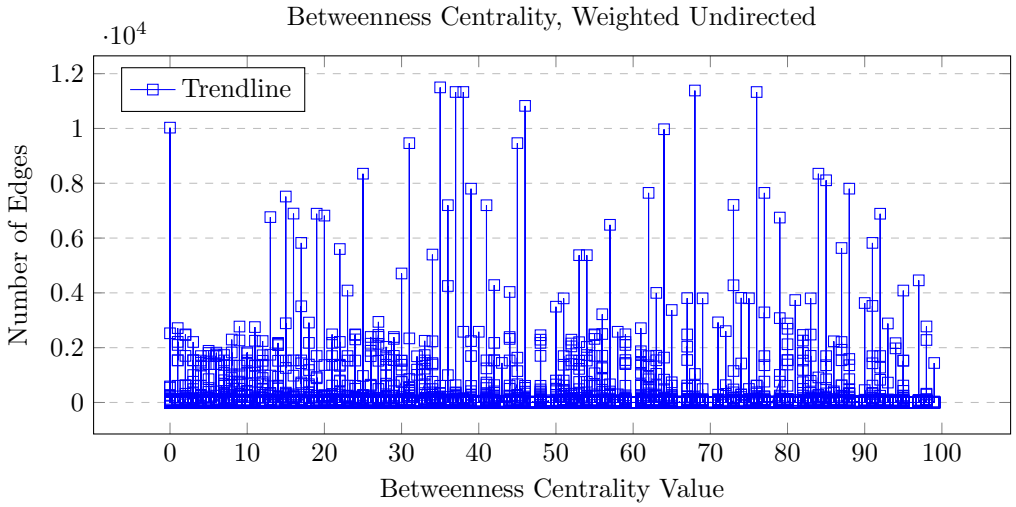
3.4.2 Betweenness Edge, Unweighted Undirected



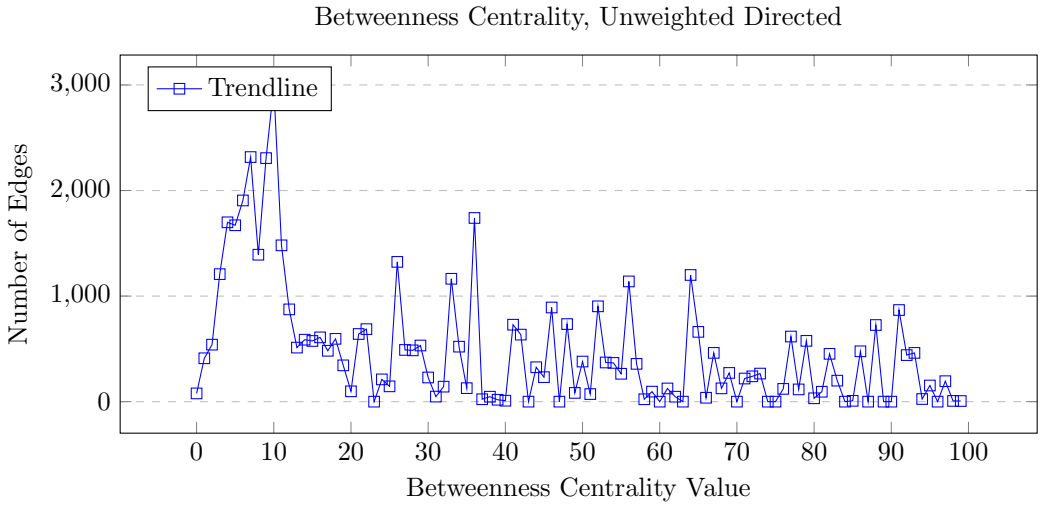
3.4.3 Betweenness Edge, Weighted Directed



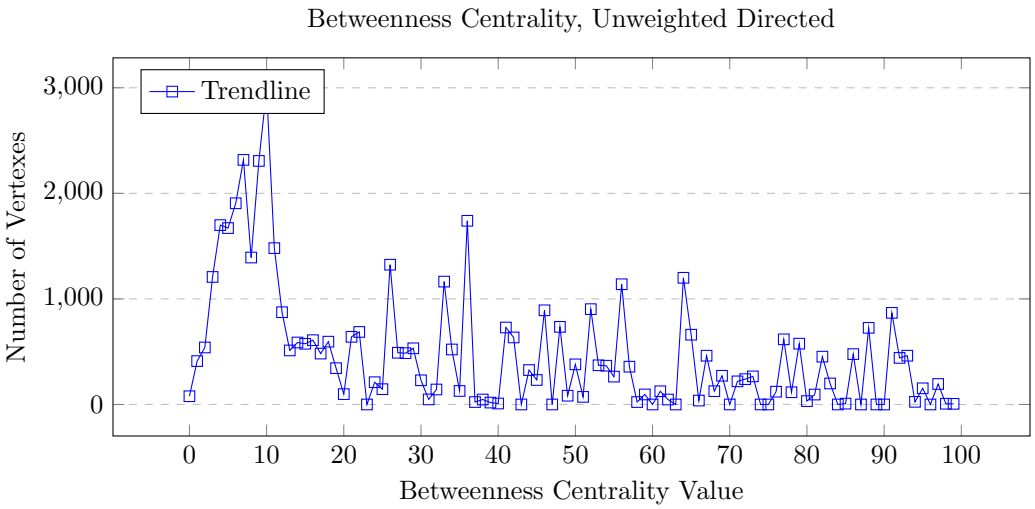
3.4.4 Betweenness Edge, Weighted Undirected



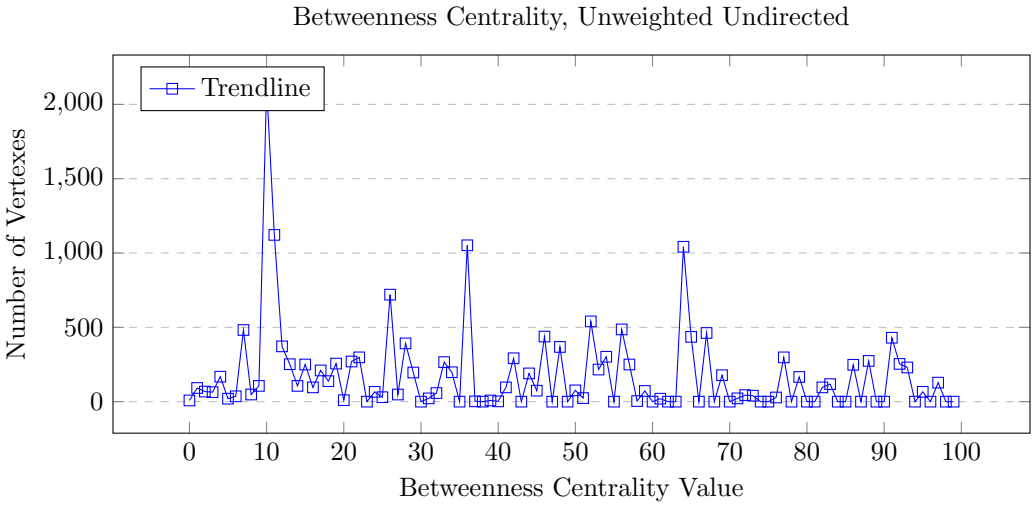
3.4.5 Betweenness Vertex, Unweighted Directed



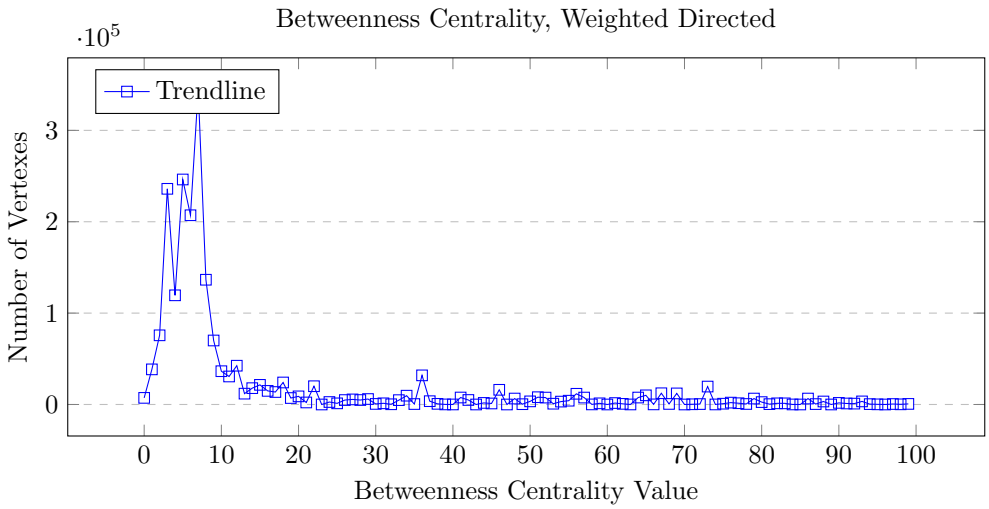
3.4.6 Betweenness Vertex, Weighted Undirected



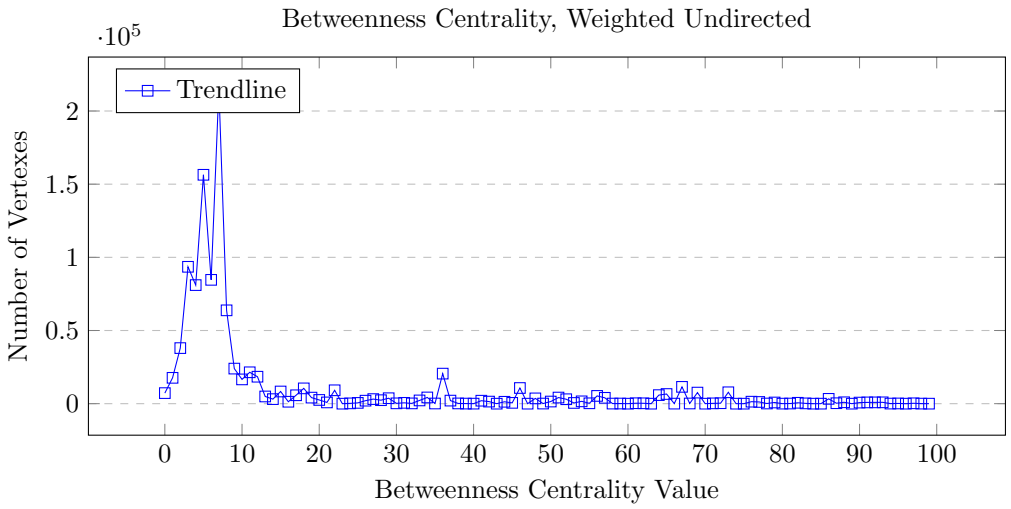
3.4.7 Betweenness Vertex, Unweighted Undirected



3.4.8 Betweenness Vertex, Weighted Directed



3.4.9 Betweenness Vertex, Weighted Undirected



4 Team Roles

- Illya Starikov
  - Project Manager

  - Implementation
    * Weight Distribution
    * Shortest Path

- Timothy Ott
  - Report Writeup
  - Implementation
    * Closeness Centrality
    * Community Detection

- Claire Trebing
  - Report Writeup
  - Implementation
    * Unweighted/Weighted Graph Diameter
    * Betweenness Centrality Distribution

5 Interesting Results

Below are interesting results we have found in regards to the project so far.

- Our pick of data structure (essential a Binary Search Tree) made the project easily testable. Because of Binary Search Tree's $\mathcal{O}(\lg n)$ retrieval complexity, this made our project run under 100 macro-seconds every compilation.

- For degree distribution, there seemed to be roughly 10 vertexes that were very dense. Although the graph seems fairly homogeneous many vertexes had a total degree of 15 or more.
  - When observing total weighted distribution (which appears almost to be $f(x) = \frac{1}{x}$), there is a tail caused by three nodes.

- There are always more paths between an arbitrary two vertexes than there are not. Meaning from any two nodes, it is more likely that there is a path.
  - This furthers our claim that this graph is dense.

- Closeness Centraility is so miniscule it cannot be computed by a standard `double` — the accuracy was not good enough. `long double` was used instead.

6 Conclusions

The experiments detailed above seek to find meaning in what would ordinarily be only raw data, the results of which, while being open to some interpretation, describe the small subsection of a social community that is given by the data set we were provided. For instance, the distribution of degrees is a measure of how interconnected individual nodes or profiles are to the rest of the graph. Because this graph is directed, it closely resembles a social network such as Tumblr where individuals can follow another profile but that profile is not required to follow back. From our results, particularly those of the unweighted distribution, we can tell that most of our nodes are followed by under 5 profiles and in turn follow between 2 and 8 profiles. The algorithms for Shortest Path are also a test of interconnectedness, though this time measuring the degrees of separation between nodes. From our results we can see that on average the nodes of our network are no more than two or three connections away from each other. The Graph Diameter measurements are meant to give an idea of the outliers in this community or to define the size of the outward edges of this network. Centrality metrics seek to identify the most important nodes within the network, using a variety of qualifications to define what is most important. A measure of Closeness centrality is effectively the inverse of a nodes farness or distance from other nodes. The next centrality metric that we tested for was the Betweenness Centrality which attempts to determine those nodes that most often act as a bridge between two other nodes. This measurement has applications within social networks to determine which individual (or in our social network examples, profile) has the most influence on communication between other individuals. Lastly we attempted to locate and determine the smaller communities within our social network. One way to accomplish this is to locate those edges that have the highest betweenness centrality and eliminate them from the network. These edges are most likely to be used as bridges between communities so by eliminating them we are able to separate and identify those communities. Once we have eliminated those edges we simply recalculate the shortest paths and diameters of the network to see how the network has changed. As we can see, these experiments and algorithms are of great use to us to sift through a large quantity of raw data and attribute meaning to them where there originally was little to none. After all, after applying these algorithms to a simple adjacency list we are now able to infer a large amount of information about the underlying situation that is being represented by this graph.

CS3001

Skills Development

Illya Starikov (isgx2)

CS5400-SP2018 Puzzle3 Grading Rubric

Datastructures: 100.00%

Looks good.

Action & state generation: 100.00%

Looks good.

Search Algorithm: 100.00%

Looks good.

Programming Practice: 100.00%

Good documentation!

Code reliability: 100.00%

Good.

Bonus: 100.00%

Good analysis!

Total = 3.00%(Datastructures) + 2.00%(Action & state generation) + 75.00%(Search Algorithm) +

10.00%(Programming Practice) + 10.00%(Code reliability) + 15.00%(Bonus)

Total: 115.00%

CS3200

Numerical Methods

Homework _num9

Illya Starikov

Due Date: December 1st, 2016

1 Simpson 1/3 Rule For $1 + 2x + 3x^2$

$$\begin{aligned}\int_0^4 1 + 2x + 3x^2\,dx &\approx {}^{h}\!/_{6}(f(a) + 4\,f(a+h) + f(b)) \\ &= {}^{4}\!/_{6}(1 + 4(1 + 4 + 12) + (1 + 8 + 48)) \\ &= {}^{2}\!/_{3}(1 + 68 + 57) \\ &= 84\end{aligned}$$

2 Simpson's 3/8 Rule For x^3

$$\begin{aligned}\int_0^3 x^3\,dx &\approx {}^{3}\!/_{8}h(f(a) + 3(a+h) + 3(a+2h) + f(b)) \\ &= {}^{3}\!/_{8}(0 + 1 + 8 + 27) \\ &= {}^{3}\!/_{8} \cdot 36 \\ &= 13.5\end{aligned}$$

3 Trapezoidal, Richardson and Romberg Method of x^4

We know the equation for the trapezoid rule to be

$$I = \frac{h}{2}\left[f(x_0) + 2\sum_{i=1}^{n} + f(x_n)\right]$$

From this, we obtain

$$\begin{aligned} I_4 &= \frac{4}{2}\left(f(0) + f(4)\right) = 512 \\ I_2 &= \frac{2}{2}\left(f(0) + 2\,f(2) + f(4)\right) = 288 \\ I_1 &= \frac{1}{2}\left(f(0) + 2\,f(2) + 2\,f(3) + f(4)\right) = 226 \end{aligned}$$

For Richardson, we calculate the terms as follows:

$$\begin{aligned} I &\approx {}^{4}\!/_{3} I(h_2) - {}^{1}\!/_{3} I(h_1) \\ I_2 &= {}^{4}\!/_{3}(288) - {}^{1}\!/_{3}(512) = {}^{640}\!/_{3} = 213.\bar{3} \\ I_1 &= {}^{4}\!/_{3}(226) - {}^{1}\!/_{3}(288) = {}^{616}\!/_{3} = 205.\bar{3} \end{aligned}$$

And for Romberg,

$$\begin{aligned} I_{j,k} &= \frac{4^{k-1} I_{j+1,k-1} - I_{j,k-1}}{4^{k-1}} \\ I_1 &= \frac{4^2({}^{616}\!/_{3}) - {}^{640}\!/_{3}}{15} = \frac{1024}{5} = 204.8 \end{aligned}$$

We observe that the table looks like

h	**trapazoidal**	**Richardson**	**Romberg**
4	512		
2	288	$213.\bar{3}$	
1	226	$205.\bar{3}$	204.8

4 Derivation of Richardson Extrapolation

We recall that the value of integration I is equal to approximation + error, which we can eloquently write

$$I = I(h) + \mathcal{E}(h) \tag{1}$$

Where $\mathcal{E}$ represents our error and $I(h)$ represents our approximation. Supposing we have two different steps size (h_1 and h_2), we can rewrite equation (**??**) in the form

$$I = I(h_1) + \mathcal{E}(h_1) = I(h_2) + \mathcal{E}(h_2) \tag{2}$$

Now, let us expand $\mathcal{E}(h)$ from equation (**??**) and equation (**??**).

$$\mathcal{E} \approx -\frac{b-a}{2} h^2 \, \bar{f}'' \tag{3}$$

Where b and a are the upper and lower bounds, respectively, and $\bar{f}''$ is the average value of $\frac{d}{dx} f(x)$ (where $f(x)$ is the function we are integrating). Therefore, we can a ratio of the two errors

$$\frac{\mathcal{E}(h_1)}{\mathcal{E}(h_2)} \approx -\frac{\frac{b-a}{2} h_1^2 \, \bar{f}''}{\frac{b-a}{2} h_2^2 \, \bar{f}''} \approx \frac{h_1^2}{h_2^2}$$

Which can be algebraically manipulated to

$$\mathcal{E}(h_1) \approx \mathcal{E}(h_2) \left(\frac{h_1}{h_2} \right)^2 \tag{4}$$

Now we can substitute equation (**??**) back into equation (**??**) to obtain

$$I \approx I(h_1) + \mathcal{E}(h_2) \left(\frac{h_1}{h_2} \right)^2 \approx I(h_2) + \mathcal{E}(h_2)$$

Which can be solve for

$$\mathcal{E}(h_2) \approx \frac{I(h_2) - I(h_1)}{(h_1/h_2)^2 - 1}$$

Now that we have an estimate (hence the $\approx$) of the truncation error, We once plug this back into equation (**??**) to get

$$I \approx I(h_2) + \approx \frac{I(h_2) - I(h_1)}{(h_1/h_2)^2 - 1}.$$

Because we know the interval to be halved, we can safely assume $h_2 = h_1/2$. Thus, our equation reduces to

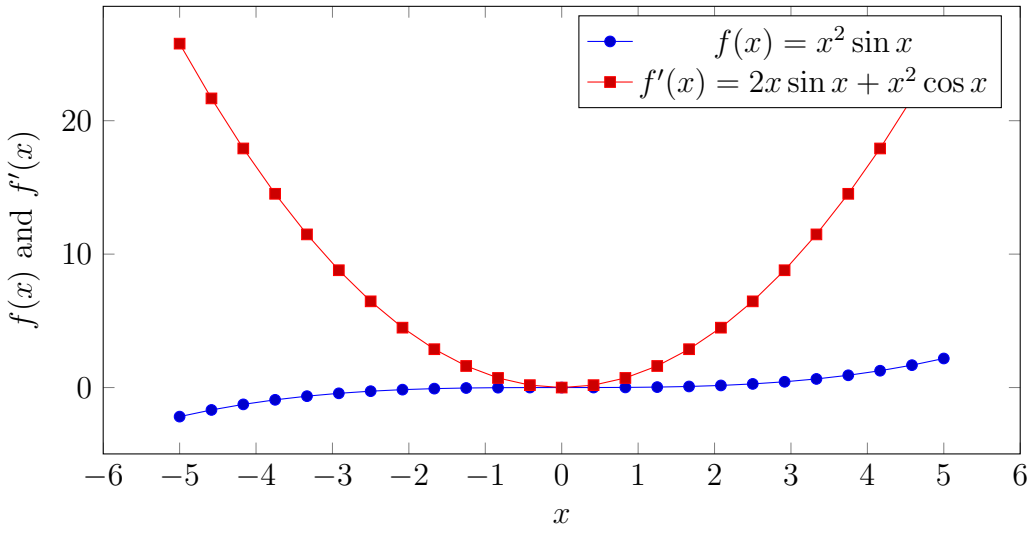
$$I \approx {}^4\!/_3\, I(h_2) - {}^1\!/_3\, I(h_1) \tag{5}$$

5 Derivative of $x^2 \sin x$

From the chain rule, we get

$$\frac{d}{dx} x^2 \sin x = 2x \sin x + x^2 \cos x$$

It can be plotted as follows



To compute the two derivatives for $h_1 = 0.2$ and $h_2 = 0.1$, we use the centered difference formulas. For h_1

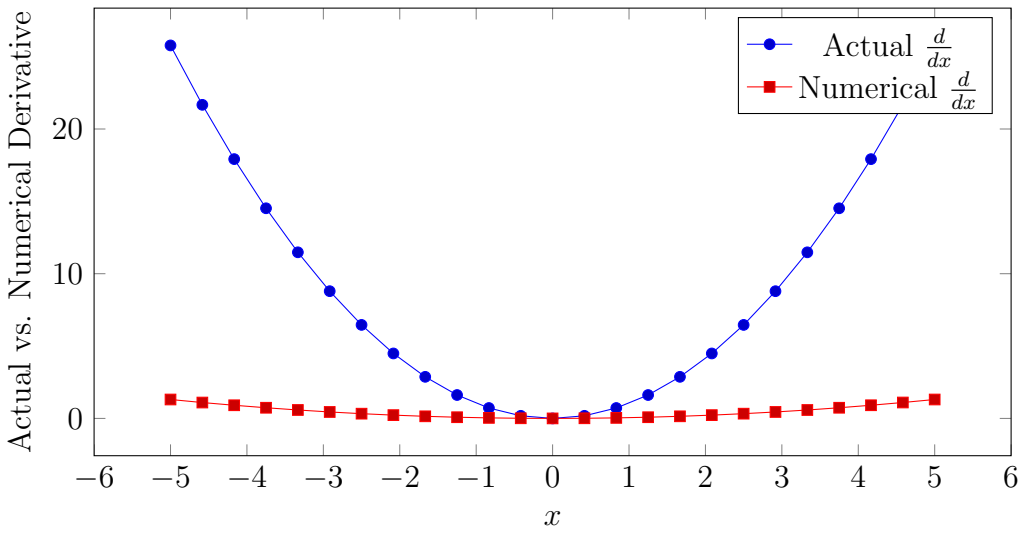
$$\begin{aligned}
\frac{d}{dx} &= \frac{f(x_{i+1}) - f(x_{i-1})}{2h} \\
&= \frac{f(x+h) - f(x-h)}{2h} \\
&= \frac{f(x + {}^1\!/_5) - f(x - {}^1\!/_5)}{{}^2\!/_5} \\
&= \frac{(x + {}^1\!/_5)^2 \sin(x + {}^1\!/_5) - ((x - {}^1\!/_5)^2 \sin(x - {}^1\!/_5))}{{}^2\!/_5}
\end{aligned}$$

Similarly, for h_2

$$\frac{(x+1/10)^2\sin(x+1/10)-((x-1/10)^2\sin(x-1/10))}{1/5}$$

We combine these for obtain the Richardson extrapolation formula,

$$\begin{aligned} D &\approx 4/3 D(h_2) - 1/3 D(h_1) && (6)\\ &\approx 4/3 \frac{(x+1/10)^2\sin(x+1/10)-((x-1/10)^2\sin(x-1/10))}{1/5} && (7)\\ &\quad - 1/3 \frac{(x+1/5)^2\sin(x+1/5)-((x-1/5)^2\sin(x-1/5))}{2/5} && (8)\end{aligned}$$



6 Central Difference of x^4

6.1 First Order

From the power rule, we notice that

$$\frac{d}{dx}x^4 = 4x^3 \implies f'(1) = 4$$

Using centered difference formula,

$$\begin{aligned}\frac{d}{dx} &= \frac{f(x_{i+1}) - f(x_{i-1})}{2h} \\ &= \frac{(1 + 1/2)^4 - (1 - 1/2)^4)}{(2 1/2)} \\ &= 5\end{aligned}$$

We notice the error to be $|\frac{4-5}{4}| = |-.25| \implies \mathbf{25}\%$

6.2 Second Order

From the power rule, we notice that

$$\frac{d^2}{dx^2}x^4 = 12x^2 \implies f'(1) = 12.$$

Using centered difference formula,

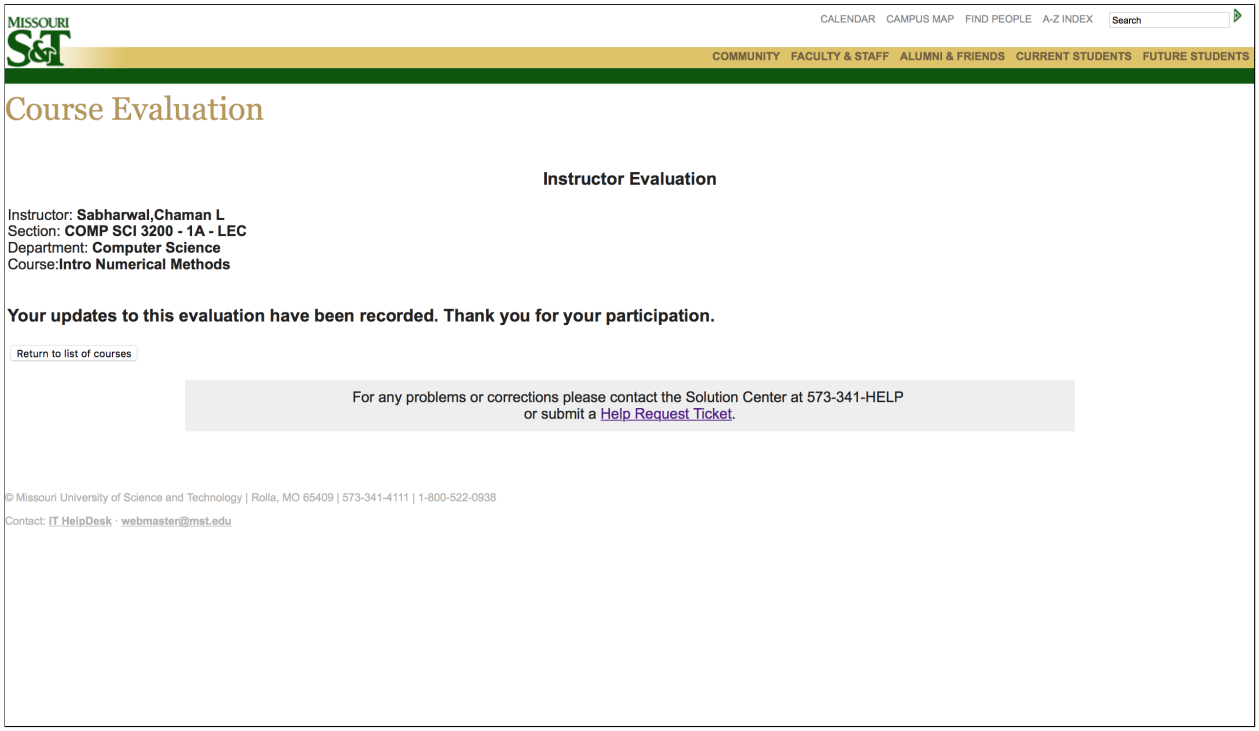
$$\begin{aligned}\frac{d^2}{dx^2} &= \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2} \\ &= \frac{(1 + 1/2)^4 - 2(1^4) + (1 - 1/2)^4)}{(1/2)^2} \\ &= {}^{25}/_2\end{aligned}$$

We notice the error to be $|\frac{12-25/2}{12}| = |-.041\bar{6}| \implies \mathbf{4.16}\%$.

Homework _num$_1$0

Illya Starikov

Due Date: December 6$^{\text{th}}$, 2016



CS3800

Operating Systems

Homework #2

Illya Starikov

Due Date: April 6th, 2017

For this assignment, the tasks were to simulate a memory manager with three page replacement algorithms

1. Clock Page Replacement
2. Least Recently Used Page Replacement
3. First In First Out Page Replacement

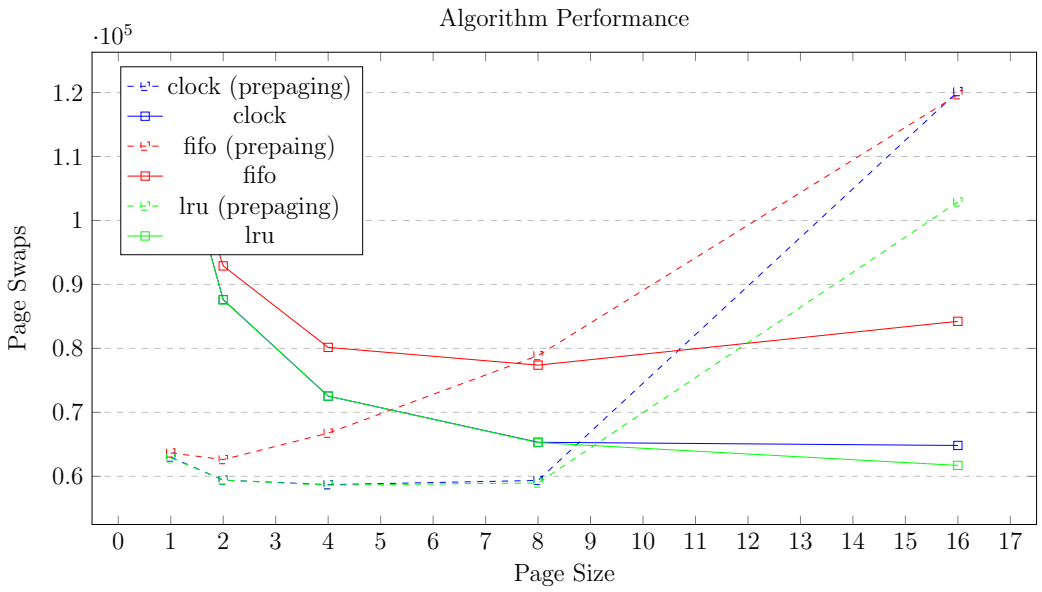


Figure 1 – Page Size vs. Page Swaps

This was accomplished by creating objects for the individual pages (eloquently named `Page`), programs (`Program`), and the memory simulator. The individual algorithms were implemented as methods of the `MemorySimulation` class.

The results can be summarized by Figure 1. For the most part, the page swaps ranged from 60 000 to 120 000. The different page sizing affected the algorithms the same, *depending on if there was prepaging*. For prepaging, higher page size increased the page swaps. The inverse also holds, for on demand paging, the algorithms faired better for higher page size. Overall, *Least Recently Used* and *Clock Based Paging* with prepaging worked the best for page sizes of 2, 4, and 8. The worst was *Clock Based Paging* and *First In First Out* with prepaging. The median solution was *Least Recently Used* with prepaging.

In retrospect, one algorithm was no significantly more difficult to implement than another. Conceptually, the *Least Recently Used* algorithm was the most difficult, and even that one was not hard. *First In First Out* was the easiest to implement, but was the worst performance wise. *Clock Based Paging* was the hardest, but had some of the best performance.

Overall, the complexity of the algorithms wasn't the deciding factor on which one to use in day-to-day situations. With very high page swapping happening in an operating system, the deciding factor is the performance.

CS5200

Analysis Of Algorithms

Homework #1

Analysis of Algorithms

Illya Starikov

Due Date: September 1st, 2017

Question #1

(a) 4.54 ± 0.05 billion years (`https://en.wikipedia.org/wiki/Age_of_the_Earth`)

(b) 5 billion years (`http://earthguide.ucsd.edu/virtualmuseum/ita/05_3.shtml`)

(c) 13.6 billion years $\pm$ 800 million years (`https://www.space.com/263-milky-age-narrowed.html`)

(d) 13.82 billion years (`http://www.slate.com/blogs/bad_astronomy/2013/03/21/age_of_the_universe_planck_results_show_universe_is_13_82_billion_years.html`)

(e) 7.79 billion years (`https://www.livescience.com/39775-how-long-can-earth-support-life`
`html`)

(f) 3.25 billion years. Guaranteed existential risks, human prevention not possible.

- Asteroid impact
- Extraterrestrial invasion (i.e. , aliens taking over the world)
- Geomagnetic reversal of the North and South pole

Prevention possible.

- Artificial super intelligence enslavement of the man race. Steps are unclear, besides keeping all experiments in a Faraday cage.
- Biotechnology engineering. A bio-engineered virus or disease that gets loose can cause a global pandemic. Steps to prevent are clear.
- Nuclear holocaust. Steps to prevent are clear.

Extinction prevention matters are to become a space-ferrying species, attempt some sort of peace treaty amongst all nations, and to try to prevent global pandemics. (`https://en.wikipedia.org/wiki/Global_catastrophic_risk#Asteroid_impact`)

(g) 10-12 billion years. The sun will become a Red Giant, no longer being able to provide fuel. Will consume several planets along the way. (https://www.forbes.com/sites/startswithabang/2017/01/27/how-our-solar-system-will-end-in-the-far-f#20ea714f4e5d)

> Eventually, about 5–7 billion years down the line, we'll run out of nuclear fuel in the Sun's core, which will cause our parent star to become a Red Giant, engulfing Mercury and Venus in the process. Due to the particulars of stellar evolution, the Earth/Moon system will probably be pushed outwards, and be spared the fiery fate of our inner neighbors.

(h) Either the universe will stop contracting, reach an influx in the gravitational pull on all cosmic bodies, and come back to a singularity; or, heat death, where entropy reaches its max, and all energy is evenly dissipated throughout the universe, freezing everything. The first theory has 1-100 trillion years, the second has no estimate. (http://www.bbc.com/earth/story/20150602-how-will-the-universe-end)

(i) 584.6 billion years. Between 0.585%–58.5%.

Question #2

(a) A standard, 12pt LaTeX document can fit roughly 39 lines of text.

$$\frac{9 \times 10^8}{39} \approx 2.31 \times 10^8 \,\text{sheets}$$

(b) 1500 sheets is approximately $14.99, before tax. (https://www.amazon.com/Georgia-Pacific-S dp/B00BB5DJU6/ref=sr_1_2?s=office-products&ie=UTF8&qid=1503934559&sr=1-2&keywords=printer+paper)

$$\left(2.31 \times 10^8 \,\text{sheets}\right) \times \frac{\$14.99}{1500 \,\text{sheets}} \approx \$2.31 \times 10^6$$

Assuming a standard filing cabinet can store roughly 15 reams, with 100 sheets in a ream,

$$2.31 \times 10^8 \,\text{sheets} \times \frac{1 \,\text{reams}}{100 \,\text{sheets}} \times \frac{1 \,\text{cabinet}}{15 \,\text{reams}} = 154\,000$$

(c) The monks expected the project to take *15,000 years* by hand, and *3 months* by modern technology. Assuming an average human lifespan of 79 years, approximately 50 years could be used writing.

$$\frac{15\,000 \,\text{years}}{50 \,\text{years}} = 300 \,\text{people}$$

(d) The time allocated to the computer was *3 months*. To print all 3 billion, it would have to print approximately 1120.07 names per second.

(e) I enjoyed the story, and the first thing I did was a mental check to see if the numbers added up. Naturally, they did not, but that did not detract from the story.

Question #3

```
1  def alternating_difference(numbers):
2      if not numbers:
3          return 0
4
5      return numbers[0] - alternating_difference(numbers[1:])
```

Question #4

```
1  def f53q(n):
2      if  n < 8:
3          return ValueError('Value less than 8')
4      elif n % 5 == 0:
5          return (0, n // 5)
6      elif n % 5 == 1:
7          return (2, (n - 5) // 5)
8      elif n % 5 == 2:
9          return (4, (n - 12) // 5)
10     elif n % 5 == 3:
11         return (1, n // 5)
12     else:
13         return (3, (n - 9) // 5)
```

Question #5

$$f(1) = 91$$
$$f(-6) = 91$$
$$f(200) = 190$$
$$f(27) = 91$$

A more appropriate, non-recursive function would be as follows:

```
1  def f(n):
2      if n < 101:
3          return 91
4      else:
5          return n - 10
```

This will always produce the same output. Let us write $f(n)$ as a piecewise-defined function.

$$f(x) = \begin{cases} n - 10 & 100 < n < \infty \\ f(f(n + 11)) & -\infty < n \leq 100 \end{cases}$$

We see that, for values less than 100, n will grow until it hits the first case. For values $90 \ldots 200$, we see that n will oscillate until it reaches 101, upon which one final $f(x)$ will be applied.

A more simple explanation would be to insert $n - 10$ into $f(f(n + 11))$ to see that $f(x) = f(f(n + 1))$ until it reaches 101, 1 past the bounds, and gets $n - 10$ applied to it one more time.

Question #6

The function is copy and pasted from implementation.

```
1 def A(x, y):
2     if x == 0:
3         return y + 1
4     elif y == 0:
5         return A(x - 1, 1)
6     else:
7         return A(x - 1, A(x, y - 1))
```

The maximum value of my system (Late-2013 MacBook Pro) is roughly `A(3, 5)`. For `A(2, 2)`, there should be 42 438 function calls.

Question #7

```
1 def gcd2(a, b):
2     if a == 0:
3         return (b, 0, 1)
4     else:
5         g, s, t = gcd2(b % a, a)
6         return [g, t - (b // a) * s, s]
```

Question #8

```
1 # defined as list_ to not stomp out the typename _list_
2 def super_reverse(list_):
3     if len(list_) < 2:
4         return list_
5     else:
6         last_element = list_[-1]
7         first_element = list_[0]
8
```

```
        # if is a list or a tuple, sort that as well
        if isinstance(first_element, list):
            first_element = super_reverse(first_element)
        if isinstance(last_element, list):
            last_element = super_reverse(last_element)

        return [last_element] + super_reverse(list_[1:-1]) + [
    first_element]
```

Question #9

```
def anagram(string_):
    if string_ == "":
        return [""]

    result = []

    for partial_anagram in anagram(string_[1:]):
        for position in range(len(partial_anagram) + 1) :
            right_side = partial_anagram[position:]
            left_side = partial_anagram[:position]
            original_element = string_[0]

            result += [left_side + original_element + right_side]

    return result
```

Table 1: Results (units in milliseconds)

String Size	One Recursive Call	n Recursive Calls	Absolute Difference	Percent Difference
1	0.006	0.004	-0.002	-56.250%
2	0.009	0.009	-0.000	-2.703%
3	0.015	0.026	0.011	42.202%
4	0.036	0.104	0.068	65.367%
5	0.139	0.528	0.389	73.668%
6	0.672	3.124	2.452	78.486%
7	5.312	25.050	19.738	78.794%
8	39.671	204.850	165.179	80.634%
9	332.021	1936.445	1604.424	82.854%
10	3471.336	21813.828	18342.492	84.087%
11	43223.737	267635.574	224411.837	83.850%

Homework #2

Analysis of Algorithms

Illya Starikov

Due Date: September 11th, 2017

Question #1

We define $Q = \{q | q + n = n + k\}$, and $n \in Z^+$. Given that $n + 0 = n = 0 + n$

$$\begin{aligned} q + (k + 1) &= (q + k) + 1 \\ &= (k + q) + 1 \\ &= k + (1 + k) \\ &= (k + 1) + m \\ &\therefore q \in Q \\ &\therefore n + q = q \end{aligned}$$

Question #2

Theorem 1. $\forall i \in \mathbb{Z}^+$,

$$\sum_{i=1}^{n} \frac{i^2}{(2i - 1)(2i + 1)} = \frac{n(n + 1)}{2(2n + 1)} \tag{1}$$

Proof. We will use the principle of recursion to solve this problem.

Step #1 Our problem states for an arbitrary integer i, summed to an arbitrary integer n, we have the summation (we denote by $f(x)$) and explicit forms ($g(x)$) described by Equation 1. We take the stopping value to be 0.

Step #2 Checking two values is trivial; take 1 and 2.

$$\sum_{i=1}^{1} \frac{i^2}{(2i-1)(2i+1)} = \frac{1(1+1)}{2(2*1+1)} = \frac{1}{3}$$
$$\sum_{i=1}^{n} \frac{i^2}{(2i-1)(2i+1)} = \frac{2*(2+1)}{2(2*2+1)} = \frac{3}{5}$$

To check the stopping value, we check the 0th value. We clearly see that $f(x) = g(x) = 0$.

Step #3 If n in $\mathbb{Z}^+$ triggers a recursive call, then $n > 0$. The only value used in the call is $n-1$, which is in $\mathbb{Z}$ and greater than or equal to 0, because it is an integer and $n1 \geq 0$ since $n > 0$.

Step #4 We use the integer n as the counter. When recursion is called the function is called with the value $n-1$. The counter strictly decreases and the recursion halts.

Step #5 To prove recursion stops, we take the following:

$$\begin{aligned}
f(n) &= \sum_{i=1}^{n} \frac{i^2}{(2i-1)(2i+1)} && \text{definition} \\
&= f(n-1) + \frac{n^2}{(2n-1)(2n+1)} && \text{recursive structure} \\
&= g(n-1) + \frac{n^2}{(2n-1)(2n+1)} && \text{by our assumption} \\
&= \frac{(n-1)((n-1)+1)}{2(2(n-1)+1)} + \frac{n^2}{(2n-1)(2n+1)} && \text{replacing } g(n) \text{ with definition} \\
&= \frac{n(n+1)}{2(2(2n+1))} && \text{algebraic simplification} \\
&= g(n)
\end{aligned}$$

Because $f(n) = g(n)$, the property is inherited recursively.

Step #6 Since Steps 1–5 have been verified, it follows from the Principle of Recursion that P holds for all values in $\mathbb{Z}^+$, i.e., $f(n) = g(n), \forall n \in \mathbb{Z}, n \geq 0$

□

Table 1: The results from the explicit form, the series form, and the difference for the first 40 values.

Sum Value	**Explicit Values**	**Difference**
0.333 333 333 333	0.333 333 333 333	0.000 000 000 000 000
0.600 000 000 000	0.600 000 000 000	0.000 000 000 000 000
0.857 142 857 143	0.857 142 857 143	0.000 000 000 000 000
1.111 111 111 111	1.111 111 111 111	0.000 000 000 000 000
1.363 636 363 636	1.363 636 363 636	0.000 000 000 000 000
1.615 384 615 385	1.615 384 615 385	0.000 000 000 000 000
1.866 666 666 667	1.866 666 666 667	0.000 000 000 000 000
2.117 647 058 824	2.117 647 058 824	0.000 000 000 000 000
2.368 421 052 632	2.368 421 052 632	0.000 000 000 000 000
2.619 047 619 048	2.619 047 619 048	0.000 000 000 000 000
2.869 565 217 391	2.869 565 217 391	0.000 000 000 000 000
3.120 000 000 000	3.120 000 000 000	0.000 000 000 000 001
3.370 370 370 370	3.370 370 370 370	0.000 000 000 000 001
3.620 689 655 172	3.620 689 655 172	0.000 000 000 000 001
3.870 967 741 935	3.870 967 741 935	0.000 000 000 000 001
4.121 212 121 212	4.121 212 121 212	0.000 000 000 000 001
4.371 428 571 429	4.371 428 571 429	0.000 000 000 000 001
4.621 621 621 622	4.621 621 621 622	0.000 000 000 000 002
4.871 794 871 795	4.871 794 871 795	0.000 000 000 000 001
5.121 951 219 512	5.121 951 219 512	0.000 000 000 000 002
5.372 093 023 256	5.372 093 023 256	0.000 000 000 000 002
5.622 222 222 222	5.622 222 222 222	0.000 000 000 000 002
5.872 340 425 532	5.872 340 425 532	0.000 000 000 000 003
6.122 448 979 592	6.122 448 979 592	0.000 000 000 000 003
6.372 549 019 608	6.372 549 019 608	0.000 000 000 000 003
6.622 641 509 434	6.622 641 509 434	0.000 000 000 000 004
6.872 727 272 727	6.872 727 272 727	0.000 000 000 000 004
7.122 807 017 544	7.122 807 017 544	0.000 000 000 000 004
7.372 881 355 932	7.372 881 355 932	0.000 000 000 000 004
7.622 950 819 672	7.622 950 819 672	0.000 000 000 000 004
7.873 015 873 016	7.873 015 873 016	0.000 000 000 000 003
8.123 076 923 077	8.123 076 923 077	0.000 000 000 000 004
8.373 134 328 358	8.373 134 328 358	0.000 000 000 000 004
8.623 188 405 797	8.623 188 405 797	0.000 000 000 000 004
8.873 239 436 620	8.873 239 436 620	0.000 000 000 000 004
9.123 287 671 233	9.123 287 671 233	0.000 000 000 000 004
9.373 333 333 333	9.373 333 333 333	0.000 000 000 000 004
9.623 376 623 377	9.623 376 623 377	0.000 000 000 000 004
9.873 417 721 519	9.873 417 721 519	0.000 000 000 000 004

Question #3

Theorem 2. *For an arbitrary $n \in Z$, such that $n \geq 0$,*

$$2^n \leq fib(n) \leq 2^{\frac{n}{2}}$$

Proof. *Step #1* Prove by induction that for $n > 0$

$$2^n \leq fib(n) \leq 2^{\frac{n}{2}}$$

Step #2 Check the base case and two other values,

$$2^0 \leq fib(0) \leq\leq 2^0 2^1 \leq fib(1) \leq\leq 2^{\frac{1}{2}} 2^2 \leq fib(2) \leq\leq 2^1$$

Step #3 To prove the recursive case, we take the following.

$$\begin{aligned} 2^n &\leq fib(n) \leq 2^{\frac{n}{2}} \\ 2^{n-1} &\leq fib(n) + fib(n-1) \leq 2^{\frac{n-1}{2}} \\ \frac{2^n}{2} &\leq fib(n) + fib(n-1) \leq \frac{2^{\frac{n}{2}}}{\sqrt{2}} \end{aligned}$$

Because we know $\frac{2^n}{2} \leq 2^n$ and $\frac{2^{\frac{n}{2}}}{\sqrt{2}} \leq 2^{\frac{n}{2}}$, we know that $f(n)$ must be bounded by those function.

Step #4 Since the hypotheses of Simple Induction are true, the conclusion follows, namely, for all natural numbers, $f(n) \mod 133 = 0$.

□

Question #4

Upon inspection, it's quite apparent that the series is the sum of all natural numbers, with every other number being negative. More formally,

$$\sum_{q=1}^{n} (-1)^{q-1} q^2 = (-1)^{n-1} \frac{n * (n+1)}{2}$$

The results can be summarized below.

Table 2: The results from the explicit form, the series form, and the difference for the first 40 values.

Sum Value	**Explicit Values**	**Sum of Numbers**
1.0000000000	1.0000000000	1.000000000000000
-3.0000000000	-3.0000000000	3.000000000000000
6.0000000000	6.0000000000	6.000000000000000
-10.0000000000	-10.0000000000	10.000000000000000
15.0000000000	15.0000000000	15.000000000000000
-21.0000000000	-21.0000000000	21.000000000000000
28.0000000000	28.0000000000	28.000000000000000
-36.0000000000	-36.0000000000	36.000000000000000
45.0000000000	45.0000000000	45.000000000000000
-55.0000000000	-55.0000000000	55.000000000000000
66.0000000000	66.0000000000	66.000000000000000
-78.0000000000	-78.0000000000	78.000000000000000
91.0000000000	91.0000000000	91.000000000000000
-105.0000000000	-105.0000000000	105.000000000000000
120.0000000000	120.0000000000	120.000000000000000
-136.0000000000	-136.0000000000	136.000000000000000
153.0000000000	153.0000000000	153.000000000000000
-171.0000000000	-171.0000000000	171.000000000000000
190.0000000000	190.0000000000	190.000000000000000
-210.0000000000	-210.0000000000	210.000000000000000
231.0000000000	231.0000000000	231.000000000000000
-253.0000000000	-253.0000000000	253.000000000000000
276.0000000000	276.0000000000	276.000000000000000
-300.0000000000	-300.0000000000	300.000000000000000
325.0000000000	325.0000000000	325.000000000000000
-351.0000000000	-351.0000000000	351.000000000000000
378.0000000000	378.0000000000	378.000000000000000
-406.0000000000	-406.0000000000	406.000000000000000
435.0000000000	435.0000000000	435.000000000000000
-465.0000000000	-465.0000000000	465.000000000000000
496.0000000000	496.0000000000	496.000000000000000
-528.0000000000	-528.0000000000	528.000000000000000
561.0000000000	561.0000000000	561.000000000000000
-595.0000000000	-595.0000000000	595.000000000000000
630.0000000000	630.0000000000	630.000000000000000
-666.0000000000	-666.0000000000	666.000000000000000
703.0000000000	703.0000000000	703.000000000000000
-741.0000000000	-741.0000000000	741.000000000000000
780.0000000000	780.0000000000	780.000000000000000

Question #5

Question #6

```
def depth(n):
    if n < 2:
        return 1
    if n % 2 == 1:
        return 1 + depth(3 * n + 1)
    else:
        return 1 + depth(n // 2)


def main():
    for i in range(101):
        print("%3.0f & %5.0f\\\\" % (i, depth(i)))


if __name__ == "__main__":
    main()
```

Iteration	Depth
0	1
1	1
2	2
3	8
4	3
5	6
6	9
7	17
8	4
9	20
10	7
11	15
12	10
13	10
14	18
15	18
16	5
17	13
18	21
19	21
20	8
21	8
22	16
23	16
24	11
25	24
26	11
27	112
28	19
29	19
30	19
31	107
32	6
33	27
34	14
35	14
36	22
37	22
38	22
39	35
40	9
41	110
42	9
43	30
44	17
45	17
46	17
47	105
48	12
49	25
50	25
51	25
52	12
53	12
54	113
55	113
56	20
57	33
58	20
59	33
60	20
61	20
62	108
63	108
64	7
65	28
66	28
67	28
68	15
69	15
70	15
71	103
72	23
73	116
74	23
75	15
76	23
77	23
78	36
79	36
80	10
81	23
82	111
83	111
84	10
85	10
86	31
87	31
88	18
89	31
90	18
91	93
92	18
93	18
94	106
95	106
96	13
97	119
98	26
99	26
100	26

Question #7

Theorem 3. *For an arbitrary $n \in Z$, such that $n \geq 0$,*

$$\sum_{i=1}^{n} = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Proof. *Step #1* Prove by induction that for $n = 0$

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

We choose the summation form as $f(x)$ and the explicit form $g(x)$.

Step #2 Check the base case and two other values,

$$\sum_{i=1}^{1} n^2 = 1 = \frac{1(1+1)(2*1+1)}{6}$$
$$\sum_{i=1}^{2} n^2 = 5 = \frac{1(1+1)(2*1+1)}{6}$$
$$\sum_{i=1}^{3} n^2 = 14 = \frac{2*(2+1)(2*2+1)}{6}$$

Step #3 $\forall n > 0$, prove that $P(n)$ is true if $P(n-1)$ is true,

$$\begin{aligned}
f(x) &= \sum_{i=1}^{n} i^2 && \text{by definition} \\
&= f(x-1) + n^2 && \text{by recursion} \\
&= g(x-1) + n^2 && \text{by hypothesis} \\
&= \frac{(n-1)(n-1+1)(2(n+1)+1)}{6} + n^2 \\
&= \frac{n(n+1)(2n+1)}{6} && \text{algebraic simplification}
\end{aligned}$$

Step #4 Since the hypotheses of Simple Induction are true, the conclusion follows,namely, for all natural numbers, $f(n) = g(n)$.

□

Question #8

Theorem 4. *For an arbitrary $n \in Z$, such that $n \geq 0$,*

$$11^{n+2} + 12^{2n+1} \mod 133 = 0$$

Proof. Step #1 Prove by induction that for $n > 0$

$$11^{n+2} + 12^{2n+1} \mod 133 = 0$$

Step #2 Check the base case and two other values,

$$\begin{aligned} 11^3 + 12^3 \mod 133 &= 0 \\ 11^4 + 12^5 \mod 133 &= 0 \end{aligned}$$

Step #3 $\forall n > 0$, prove that $P(n)$ is true if $P(n-1)$ is true,

$$\begin{aligned} f(x) &= 11^{n+2} + 12^{2n+1} \\ &= 11^{n+1} + 12^{2n-1} + 11^{n+2} + 12^{2n+1} \end{aligned}$$

Because we assume $11^{n+2} + 12^{2n+1}$ to be divisible, we can rewrite it to be $133a_1$, signifying it is evenly divisible by 133.

$$\begin{aligned} f(x) &= 11 * 11^n + 12^{-1} * 144^n + 133a_1 \\ &= a_2 11^n + a_3 144^n + 133a_1 \end{aligned}$$

We know the superposition of all of these numbers to be divisible by 133.

Step #4 Since the hypotheses of Simple Induction are true, the conclusion follows, namely, for all natural numbers, $f(n) \mod 133 = 0$.

□

Question #9

Let $F(0)$ denote the root of the tree T. If we start at $F(0)$, there must be an adjacent vertex to it, namely, v_1. There are infinitely many vertexes by going through the tree (this must be the case, or the tree T would be finite).

We can repeat this process, for at any vertex v_n, there exists a vertex v_{n+1}. Because the tree is infinite, there are infinitely many v_{n+1}.

By induction, for any n, there exists a path of length L in T.

Question #10

```
1 class Binary_Search_Tree:
2     def __init__(self, data):
3         if data is None:
4             self.data = [0, [], []]
5         else:
6             self.data = data
```

```
7
8       def add_node(self, node):
9           if node.data[0] < self.data[0]:
10              self.__add_node_left(node)
11          else:
12              self.__add_node_right(node)
13
14      def __add_node_left(self, node):
15          if self.data[1] == []:
16              self.data[1] = node
17          else:
18              self.data[1].add_node(node)
19
20      def __add_node_right(self, node):
21          if self.data[2] == []:
22              self.data[2] = node
23          else:
24              self.data[2].add_node(node)
25
26      def leaf_count(self):
27          if self.data[1] == []:
28              if self.data[2] == []:
29                  return 1
30              else:
31                  return self.data[2].leaf_count()
32          elif self.data[2] == []:
33              return self.data[1].leaf_count()
34          else:
35              return self.data[1].leaf_count() + self.data[2].leaf_count()
36
37      def internal_count(self):
38          if self.data[1] == []:
39              if self.data[2] == []:
40                  return 0
41          elif self.data[2] == []:
42              return 1 + self.data[1].internal_count()
43          else:
44              return 1 + self.data[1].internal_count() + self.data[2].
        internal_count()
45
46
47  def main():
48      T = Binary_Search_Tree([
49          42,
50          Binary_Search_Tree([29, [], []]),
51          Binary_Search_Tree([51, [], []])
52      ])
53
54      print("Leaf: {0}, Internal: {1}".format(T.internal_count(), T.
        leaf_count()))
55
56      T.add_node(Binary_Search_Tree([25, [], []]))
```

```
57      T.add_node(Binary_Search_Tree([22, [], []]))
58      T.add_node(Binary_Search_Tree([45, [], []]))
59      T.add_node(Binary_Search_Tree([8, [], []]))
60      T.add_node(Binary_Search_Tree([1000, [], []]))
61      T.add_node(Binary_Search_Tree([3, [], []]))
62      T.add_node(Binary_Search_Tree([1, [], []]))
63
64      print("Leaf: {0}, Internal: {1}".format(T.internal_count(), T.
        leaf_count()))
65
66
67  if __name__ == "__main__":
68      main()
```

From the results, we can very much see that Internal Nodes = Leaf Nodes − 1.

10.1 Proof

Proof. Take n to be the number of nodes, $I(T)$ the number of internal nodes and $L(T)$ to be the number of leaf nodes. We know the internal nodes to be the sum of previous internal nodes (i.e., $I(t) = I(t_1) + I(t_2) + 1$).

From this, we get

$$I(T) = I(t_1) + I(t_2) + 1 \tag{2}$$
$$I(t_1) = L(t_1) - 1 \tag{3}$$
$$I(t_2) = L(t_2) - 1 I(T) \qquad = L(t_1) - 1 + L(t_2) - 1 \tag{4}$$
$$= L(T) - 1 \tag{5}$$

□

Homework #3

Analysis of Algorithms

Illya Starikov

Due Date: September 19th, 2017

Question #1

```
1  def fn(n):
2      if n > 100:
3          return n - 10
4      return ((n - 101) % (c - 10)) + 91
```

Question #2

The following function tests random values up to in $\{-1\,000\,000, \ldots, 1\,000\,000\}$. `is_gcd_fast` is a faster alternative because $\gcd \in \mathcal{O}\left(\log_{\frac{a}{a \mod b}}(a \times b)\right)$ while $\gcd_{\text{fast}} \in \mathcal{O}(c)$.

```
1  from fractions import gcd
2  import random
3
4
5  def is_gcd(gcd_value, a, b):
6      return gcd_value == gcd(a, b)
7
8
9  def is_gcd_fast(gcd_value, a, b):
10     return a % gcd_value == b % gcd_value == 0
11
12 for i in range(1000):
13     a = random.randint(-1000000, 1000000)
14     b = random.randint(-1000000, 1000000)
15     g = gcd(a, b) if random.choice([True, False]) else random.randint
       (-1000000, 1000000)
16
17     print(is_gcd_fast(g, a, b) == is_gcd(g, a, b))
```

Question #3

3.1 All `Sums`

```
1 # Warning, for all lists (with the exclusion of [1], [0] and []) this
    function does not stop
2 def generate_all_sums(l):
3     for element in l:
4         if element == 0:
5             return [0] + generate_all_sums(element[1:])
6         else:
7             return [element] + [generate_all_sums([element*element])]
8
9     return l
```

3.2 $Sum(T) \notin \{$***all odds***$\}$

Take Sum to be the function that maps all elements to all of their multiples.

Theorem 1. *There is not set T such that $Sum(T) = \{$all odd integers$\}$.*

We prove so by proof of contradiction.

Proof. Suppose not. That is, suppose that is a set T such that $Sum(T) = \{$all odd integers$\}$. Set T, at a minimum, must have a single element (for the empty set $\varnothing$ cannot produce any integers). We take an arbitrary element from this set T and name it q. There are two forms for q ($\exists m \in \mathbb{Z}$),

$$q = \begin{cases} 2m & \Leftrightarrow q \text{ is even} \\ 2m+1 & \Leftrightarrow q \text{ is odd} \end{cases}$$

If q is even, there is already a contradiction. If q is odd, then we have the following contradiction. Supposing we sum q twice, we have the following form:

$$q = a(2m+1) + b(2m+1)$$

Because a and b are arbitrary, we take them to be 1.

$$\begin{aligned} q &= (2m+1) + (2m+1) \\ &= 4m + 2 \\ &= 2(2m+1) \end{aligned}$$

Because we know the sum of integers and multiplication of integers to be integers, $2(2m+1) \equiv 2n, \exists n \in \mathbb{Z}$, which has the form of an even integer.

This has lead us to a contradiction. Therefore, our hypothesis is false, concluding there is not set T such that $Sum(T) = \{$all odd integers$\}$.

□

3.3 Fast Algorithm For S

To get d from an arbitrary set S, we simply take the greatest common divisor from all the sets.

```
1 from functools import reduce
2
3 def gcd(numbers):
4     def gcd_single(a, b):
5         while b:
6             a, b = b, a % b
7         return a
8
9     return reduce(gcd_single, numbers)
```

3.4 $d \in \{540051690381, 5404079462298, 3485942644184\}$

547.

Question #4

Theorem 2. $\forall i \in \mathbb{Z}^+$,

$$\sum_{i=1}^{n} (-1)^{i-1} i^3 = -\frac{1}{8}(-1)^n \left(4n^3 + 6n^2 - 1\right) - \frac{1}{8}$$

Proof. *Step #1* We wish to prove that for all natural numbers $f(n) = g(n)$, where f and g are as defined as follows:

$$f(x) = \sum_{i=1}^{n} (-1)^{i-1} i^3$$
$$g(x) = -\frac{1}{8}(-1)^n \left(4n^3 + 6n^2 - 1\right) - \frac{1}{8}$$

Let D be the set of natural numbers (i.e., $D = \mathbb{Z}^+$). D includes the stopping value 1.

Step #2 Checking two values is trivial; take 2 and 3.

$$\sum_{i=1}^{2} (-1)^{i-1} i^3 = -\frac{1}{8}(-1)^2 \left(4 * 2^3 + 6 *^2 -1\right) - \frac{1}{8} = -7$$
$$\sum_{i=1}^{3} (-1)^{i-1} i^3 = -\frac{1}{8}(-1)^3 \left(4 * 3^3 + 6 * 3^2 - 1\right) - \frac{1}{8} = 20$$

To check the stopping value, we check the first value. We clearly see that $f(x) = g(x) = 1$.

Step #3 If n in $\mathbb{Z}^+$ triggers a recursive call, then $n > 0$. The only value used in the call is $n - 1$, which is in $\mathbb{Z}$ and greater than or equal to 0, because it is an integer and $n1 \geq 0$ since $n > 0$.

Step #4 We use the integer n as the counter. When recursion is called the function is called with the value $n - 1$. The counter strictly decreases and the recursion halts.

Step #5 To prove recursion stops, we take the following:

$$
\begin{aligned}
f(x) &= f(n-1) + f(n) && (1)\\
&= f(n-1) + (-1)^{n-1} n^3 && (2)\\
&= -\frac{1}{8}(-1)^{(n-1)} \left(4(n-1)^3 + 6(n-1)^2 - 1\right) - \frac{1}{8} + (-1)^{n-1} n^3 && (3)\\
&= -\frac{1}{8}(-1)^{n} \left(4n^3 + 6n^2 - 1\right) - \frac{1}{8} && (4)\\
&= g(x) && (5)
\end{aligned}
$$

Because $f(n) = g(n)$, the property is inherited recursively.

Step #6 Since Steps 1–5 have been verified, it follows from the Principle of Recursion that P holds for all values in $\mathbb{Z}^+$, i.e., $f(n) = g(n), \forall n \in \mathbb{Z}^+$

□

Table 1: The results from the explicit form, the series form, and the difference for the first 40 values.

Sum Value	**Explicit Values**	**Difference**
1	1	0
-7	-7	0
20	20	0
-44	-44	0
81	81	0
-135	-135	0
208	208	0
-304	-304	0
425	425	0
-575	-575	0
756	756	0
-972	-972	0
1225	1225	0
-1519	-1519	0
1856	1856	0
-2240	-2240	0
2673	2673	0
-3159	-3159	0
3700	3700	0
-4300	-4300	0
4961	4961	0
-5687	-5687	0
6480	6480	0
-7344	-7344	0
8281	8281	0
-9295	-9295	0
10388	10388	0
-11564	-11564	0
12825	12825	0
-14175	-14175	0
15616	15616	0
-17152	-17152	0
18785	18785	0
-20519	-20519	0
22356	22356	0
-24300	-24300	0
26353	26353	0
-28519	-28519	0
30800	30800	0

```python
def sum_solution(n):
    if n <= 1:
        return 1
    else:
        return (-1)**(n - 1) * n**3 + sum_solution(n - 1)


def explicit_solution(n):
    return -int(
        (1.0 / 8.0) *
        ((-1)**n) *
        (4 * n**3 + 6 * n**2 - 1) +
        1.0 / 8.0)


def main():
    for n in range(1, 40):
        print(sum_solution(n), explicit_solution(n), sum_solution(n) -
    explicit_solution(n))


if __name__ == "__main__":
    main()
```

Homework #4

Analysis of Algorithms

Illya Starikov

Due Date: September 26th, 2017

Question #1

```
1  def sum_of_two_elements(S, x):
2      S = sorted(S)
3      i, j = 0, len(S) - 1
4      while i <= j:
5          if S[i] + S[j] == x:
6              return True
7          elif S[i] + S[j] < x:
8              i += 1
9          else:
10             j -= 1
11
12     return False
```

Table 1: Output for `sum_of_two_elements`

List	x	$n \lg n$ solution	n^2 solution
[219, 526, 87, 829, 59, 384, 849, 65, 463, 934]	449	True	True
[393, 983, 584, 160, 421, 652, 41, 719, 686, 181]	1338	True	True
[491, 154, 345, 508, 208, 50, 11, 183, 723, 994]	931	True	True
[798, 810, 263, 786, 177, 314, 211, 708, 300, 286]	1061	True	True
[262, 494, 533, 38, 105, 937, 383, 625, 733, 428]	645	True	True
[730, 371, 356, 591, 126, 416, 732, 84, 505, 165]	717	True	True
[469, 821, 559, 492, 479, 91, 621, 55, 550, 507]	605	True	True
[822, 407, 527, 858, 498, 360, 551, 532, 503, 274]	1054	True	True
[278, 265, 997, 628, 563, 536, 783, 817, 725, 124]	1380	True	True
[455, 745, 422, 274, 335, 781, 909, 867, 669, 681]	943	True	True
[209, 836, 717, 858, 446, 773, 507, 693, 907, 59]	268	True	True
[18, 639, 711, 738, 583, 69, 714, 503, 597, 280]	1452	True	True
[690, 867, 901, 558, 367, 927, 439, 590, 651, 447]	1459	True	True
[156, 129, 61, 363, 948, 347, 874, 914, 775, 73]	503	True	True
[781, 985, 385, 523, 753, 804, 740, 7, 155, 441]	596	True	True
[897, 799, 83, 402, 144, 820, 621, 22, 640, 660]	1420	True	True
[478, 364, 216, 907, 638, 576, 835, 487, 571, 883]	1051	True	True
[193, 116, 483, 17, 363, 276, 14, 534, 145, 636]	162	True	True
[626, 446, 185, 716, 514, 225, 953, 826, 758, 809]	1072	True	True
[268, 778, 934, 880, 347, 306, 90, 767, 626, 230]	1712	True	True
[170, 591, 380, 744, 868, 242, 736, 756, 45, 798]	624	False	False
[85, 354, 59, 185, 916, 42, 567, 532, 106, 285]	704	False	False
[938, 218, 584, 440, 574, 748, 450, 931, 955, 588]	909	False	False
[930, 337, 292, 491, 668, 486, 630, 320, 91, 797]	17	False	False
[277, 754, 116, 486, 75, 868, 788, 346, 326, 188]	847	False	False
[951, 37, 80, 72, 515, 45, 925, 533, 626, 767]	346	False	False
[529, 286, 483, 105, 283, 224, 461, 245, 447, 861]	482	False	False
[563, 577, 795, 911, 692, 462, 755, 311, 898, 268]	706	False	False
[798, 617, 27, 746, 149, 763, 546, 752, 692, 279]	816	False	False
[334, 339, 184, 870, 776, 716, 375, 752, 414, 453]	522	False	False
[683, 969, 687, 274, 870, 566, 139, 664, 699, 325]	17	False	False
[641, 411, 4, 735, 742, 205, 264, 548, 331, 617]	604	False	False
[876, 939, 653, 487, 148, 433, 807, 238, 848, 556]	872	False	False
[385, 21, 172, 598, 463, 721, 187, 670, 328, 917]	510	False	False
[3, 246, 181, 435, 937, 974, 817, 109, 841, 383]	518	False	False
[889, 542, 73, 455, 946, 307, 189, 988, 440, 349]	58	False	False
[383, 691, 490, 938, 233, 139, 818, 231, 825, 720]	170	False	False
[772, 543, 127, 538, 273, 508, 924, 793, 495, 150]	229	False	False
[311, 843, 307, 863, 305, 602, 696, 351, 910, 776]	42	False	False

Question #2

1. As follows, the inversions (i, j) are:
   a) (1, 5)
   b) (2, 5)

c) (3, 4)

d) (3, 5)

e) (4, 5)

$(A[i], A[j])$ are:

a) (2, 1)

b) (3, 5)

c) (8, 6)

d) (8, 1)

e) (6, 1)

2. For a reverse sorted array, there would $\binom{n}{2} = \frac{n(n-1)}{2}$ total inversions.

3. Divide the array recursively into half and count number of inversions in sub-arrays (an $\lg n$ algorithm). To count all inversions, it takes n steps. $n -$ operations $\times$ $\lg n -$ recursive steps. Combined, $\Theta(n \lg n)$ algorithm.

Question #3

We recall that the geometric series, and its derivatives, state that:

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \quad \text{and} \quad \sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2} \tag{1}$$

Next, we know that our sum is as follows:

$$\sum_{k=0}^{\infty} \frac{(k-1)}{2^k}$$

Breaking these up we have the following:

$$\sum_{k=0}^{\infty} k\left(\frac{1}{2}\right)^k - \sum_{k=0}^{\infty} \left(\frac{1}{2}\right)^k \tag{2}$$

Replacing Equation 2 with Equation 1, we get the following:

$$\begin{aligned} \sum_{k=0}^{\infty} k\left(\frac{1}{2}\right)^k - \sum_{k=0}^{\infty} \left(\frac{1}{2}\right)^k &= \frac{1/2}{(1-1/2)^2} - \frac{1}{1-1/2} \\ &= 2 - 2 \\ &= 0 \end{aligned}$$

Question #4

Because k^3 is monotonically increasing, we have the following bounds upper bound:

$$\begin{aligned}\sum_{k=1}^{n} k^3 &\leq \int_1^{n+1} x^3\,dx \\ &= \frac{x^4}{4}\Big|_1^{n+1} \\ &= \frac{(n+1)^4 - 1}{4}\end{aligned}$$

For the lower bound,

$$\begin{aligned}\sum_{k=1}^{n} k^3 &\geq \int_0^{n} x^3\,dx \\ &= \frac{x^4}{4}\Big|_0^{n} \\ &= \frac{(n)^4}{4}\end{aligned}$$

So, the bounds are as follows:

$$\frac{(n)^4}{4} \leq \sum_{k=1}^{n} k^3 \leq \frac{(n+1)^4 - 1}{4}$$

Question #5

We use integral approximations for all of the following.

Problem #5.1

$$\begin{aligned}\int_0^{n} x^r\,dx \quad &\leq \sum_{k=1}^{n} k^r \quad &\leq \int_1^{n+1} x^r\,dx \\ \frac{n^{r+1}}{r+1} \quad &\leq \sum_{k=1}^{n} k^r \quad &\leq \frac{(n+1)^{r+1} - 1}{r+1}\end{aligned}$$

Therefore, the bound is $\Theta\left(n^{r+1}\right)$.

Problem #5.2

For the following, we take s for 1,

$$\begin{aligned} \int_0^n \lg^1 x\,dx \quad &\leq \sum_{k=1}^{n} \lg^1 k \quad \leq \int_1^{n+1} \lg^1 x\,dx \\ n\left(\lg\left(n\right)-1\right) \quad &\leq \sum_{k=1}^{n} \lg^1 k \quad \leq \left(n+1\right)\left(\lg\left(n+1\right)-1\right)+1 \end{aligned}$$

Therefore, the bound is $\Theta\left(n \lg^1 n\right)$.

For the following, we take s for 2,

$$\begin{aligned} \int_0^n \lg^2 x\,dx \quad &\leq \sum_{k=1}^{n} \lg^2 k \quad \leq \int_1^{n+1} \lg^2 x\,dx \\ n\left(\left(\lg\left(n\right)-2\right)\lg\left(n\right)+2\right) \quad &\leq \sum_{k=1}^{n} \lg^2 k \quad \leq \left(n+1\right)\left(\left(\lg\left(n+1\right)-2\right)\lg\left(n+1\right)+2\right)-2 \end{aligned}$$

Therefore, the bound is $\Theta\left(n \lg^2 n\right)$.

For the following, we take s for 3,

$$\int_0^n \lg^3 x\,dx \quad \leq \sum_{k=1}^{n} \lg^3 k \quad \leq \int_1^{n+1} \lg^3 x\,dx$$

From this, we get,

$$\begin{aligned} &n\left(\lg\left(n\right)\left(\left(\lg\left(n\right)-3\right)\lg\left(n\right)+6\right)-6\right) \\ &\leq \sum_{k=1}^{n} \lg^3 k \\ &\leq \left(n+1\right)\left(\lg\left(n+1\right)\left(\left(\lg\left(n+1\right)-3\right)\lg\left(n+1\right)+6\right)-6\right)+6 \end{aligned}$$

Therefore, the bound is $\Theta\left(n \lg^3 n\right)$.

In general, the integral requires s integration by parts, multiplying $\lg n$ s times. So, for $\sum_{k=1}^{n} \lg^s k$, the bound would be $\Theta\left(n \lg^s n\right)$.

Problem #5.3

In general, we know that the lower-bound integral we have will have the form

$$\int_0^n x^r \lg^s x\,dx$$

And likewise for the upper bound. Using the same process as before (omitted due to tedious mathematics), we get the tight bound $\Theta\left(n^{r+1} \lg^s n\right)$.

Homework #5

Analysis of Algorithms

Illya Starikov

Due Date: October 4th, 2017

Question #1

Theorem 1. *If $L \geq 2$, then every binary tree with L leaves contains a subtree having between $\frac{L}{3}$ and $\frac{2L}{3}$ leaves, inclusive.*

Proof. Suppose not. That is, suppose that there exists a binary tree with more than 1 leaves with all subtrees containing leaves in the range $\frac{L}{3} > n > \frac{2L}{3}$. The root contains L leaves, and all bottommost leaves contain 1 leaves.

This implies that two subtrees with size less than $\frac{L}{3}$ (because our hypothesis states all leaves must in the range $\frac{L}{3} > n > \frac{2L}{3}$) combined to make a tree of size greater than $\frac{2L}{3}$. This is a contradiction

$$\frac{L}{3} + \frac{L}{3} \not> \frac{2L}{3}$$

Because we have reached a contradiction, our hypothesis does not hold. Therefore, binary tree with L leaves contains a subtree having between $\frac{L}{3}$ and $\frac{2L}{3}$ leaves, inclusive. □

Question #2

Theorem 2. *For every binary tree T, let us associate a "weight" $w(q) = 2^{-depth(q)}$. For all leaves $q \in T$,*

$$\sum_{q} w(q) \leq 1$$

Proof. We prove so by induction.

(Base Case) For the root,

$$\sum_{x \in T} w(x) = 2^{-0} = 1$$

(Inductive Hypothesis) Suppose for any tree of n nodes, $\sum_{x \in T}^{n} w(q) \leq 1$.

(Inductive Step) Suppose the binary tree T_0 to have $n-1$ nodes. We must show that $\sum_{x \in T_0}^{n-1} w(x) \leq 1$. Suppose we take the left subtree T_L or the right subtree T_R. By the inductive hypothesis, the sum of the weights are less than or equal to 1; however, since the depth of a node in T_0 is one greater than the depth of T_L or T_R, the respective weights of ever leaf in T_0 is halved. Thus,

$$\sum_{x \in T_L} w(x) \leq 1/2 \qquad \sum_{x \in T_R} w(x) \leq 1/2$$

Because both of these are bounded by $1/2$, their sum forms T_0, which is bounded by 1.

By the principle of mathematical induction, $\sum_q w(q) \leq 1$ for any tree. □

Question #3

Theorem 3. *Any planar graph can be colored with six or fewer colors.*

Proof. We prove so by induction. We take n to be an arbitrary number of nodes in an arbitrary graph G.

(Base Case) For n vertices, where $1 \leq n \leq 6$, we can color the graph G with 6 or less colors (for every vertices maps to a single, unique color).

(Inductive Hypothesis) Suppose for $n-1$ vertices, where $n > 1$, the graph G can be colored with 6 or less colors.

(Inductive Step) We now prove that G can be colored with n vertices can be colored with 6 or fewer colors. Recall all connected, simple planar graph contains a maximum of 5 degrees. Suppose that the any vertex $v \in G$ has this max degree 5.

We remove this vertex, name it v_0, with all incident edges. We now have less than n vertices, and by our inductive hypothesis, this graph now can be colored with 6 or less colors. Adding this vertex back, we see that there are 5 incident vertexes, hence 5 colors. We use the 6^{th} color for v_0.

Hence, the inductive step holds. By the principle of mathematical induction, any planar graph can be colored with six or fewer colors. □

Question #4

Theorem 4. *Any point x in a graph is a cut-point iff there exists two vertices in the graph, a and b, such that every path between a and b has to pass through x.*

Proof. Assume G to be a connected graph, and $\exists$ vertices $a, b \in G$, where every path $a \longleftrightarrow b$ path passes through x. Also assume there exists no path $a \longleftrightarrow b \in G - \{x\}$. Therefore, $G - \{x\}$ is disconnected, for a and b are in two different components of $G - \{x\}$. This proves that x is a cut-point. □

Question #5

Euler Path A path that uses every edge of a graph exactly once.

Hamiltonian Cycle A path through a graph that starts and ends at the same vertex and includes every other vertex exactly once.

Yes, all **cyclic graphs** meet both these conditions.

Theorem 1. *Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the non-negative numbers by the recurrence*

$$T(n) = a\,T(n/b) + f(n) \tag{1}$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. *If $f(n) = \mathcal{O}\left(n^{\log_b a - \epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta\left(n^{\log_b a}\right)$.*
2. *If $f(n) = \Theta\left(n^{\log_b a}\right)$, then $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$.*
3. *If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some constant $\epsilon > 0$, and if*

$$a\,f(n/b) < c\,f(n)$$

for some constant $c < 1$ and sufficiently large n, then $T(n) = \Theta\left(f(n)\right)$.

Question #1

Problem #1.1

From Equation 1, we have the following:

$$a = 2 \qquad b = 4 \qquad f(n) = 1$$

From this,

$$\begin{aligned} f(n) &= \mathcal{O}\left(n^{\log_b a - \epsilon}\right) \\ 1 &= \mathcal{O}\left(n^{\log_4 2 - \epsilon}\right) \\ &= \mathcal{O}\left(n^{1/2 - \epsilon}\right) \\ &= \mathcal{O}\left(n^{1/2 - 1/2}\right) \\ &= \mathcal{O}\left(1\right) \end{aligned}$$

Therefore, the tight asymptotic bound for the recurrence is

$$\begin{aligned} T(n) &\in \Theta\left(n^{\log_b a}\right) \\ &\in \Theta\left(n^{\log_4 2}\right) \\ &\in \Theta\left(n^{1/2}\right) \\ &\in \Theta\left(\sqrt{n}\right) \end{aligned}$$

Therefore, the bound is $\Theta\left(\sqrt{n}\right)$.

Problem #1.2

From Equation 1, we have the following:

$$a = 2 \qquad b = 4 \qquad f(n) = \sqrt{n}$$

From this,

$$\sqrt{n} = \Theta\left(n^{\log_4 2}\right)$$

Therefore, the tight asymptotic bound for the the recurrence is

$$T(n) \in \Theta\left(\sqrt{n}\lg n\right)$$

Problem #1.3

From Equation 1, we have the following:

$$a = 2 \qquad b = 4 \qquad f(n) = n$$

From this,

$$n = \Omega\left(n^{\log_4 2+1/2}\right)$$

Therefore, the tight asymptotic bound for the the recurrence is

$$T(n) \in \Theta\left(n\right)$$

Problem #1.4

From Equation 1, we have the following:

$$a = 2 \qquad b = 4 \qquad f(n) = n^2$$

From this,

$$n^2 = \Omega\left(n^{\log_4 2+3/2}\right)$$

Therefore, the tight asymptotic bound for the recurrence is

$$T(n) \in \Theta\left(n^2\right)$$

Question #2

Recall Strassen's algorithm belongs to the complexity class $\Theta\left(n^{\lg 7}\right)$. By using the Case 1 of the Master Theorem, we must find a that solves the inequality $\log_4 a < \lg 7$. The integer that solves the equation $\log_4 a = \lg 7$ is 49; therefore,

$$a = 48$$

Question #3

Because the process is identical to Problem , the work is omitted; only answers are provided.

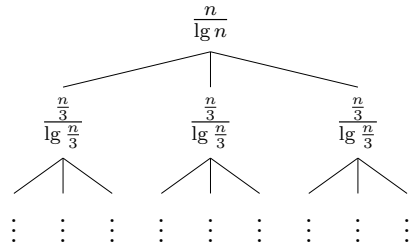
1. Master Theorem (*Case #3*), $T(n) \in \Theta\left(n^4\right)$
2. Master Theorem (*Case #3*), $T(n) \in \Theta\left(n\right)$
3. Master Theorem (*Case #2*), $T(n) \in \Theta\left(n^2 \lg n\right)$
4. Master Theorem (*Case #3*), $T(n) \in \Theta\left(n^2\right)$
5. Master Theorem (*Case #1*), $T(n) \in \Theta\left(n^{\log_2 7}\right)$
6. Master Theorem (*Case #2*), $T(n) \in \Theta\left(\sqrt{n} \lg n\right)$
7. Because $(n-2) \in \mathcal{O}\left(n\right)$, we can write an equivalent recurrence relation,

   $$T(n) = T(n/1) + n^2$$

   Because $\log_1 1$ is undefined, we cannot use the Master Theorem; but we do not need it. We have n^2 work to do n times. This implies that $T(n) \in \Theta\left(n^3\right)$.

Question #4

1. Master Theorem (*Case #1*), $T(n) \in \Theta\left(n^{\log_3 4}\right)$
2. Using a recursion tree,



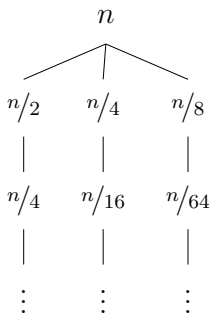
We see that the $T(n) \in \Theta(n \lg \lg n)$.

3. Master Theorem (*Case #3*), $T(n) \in \Theta(n^{2.5})$.

4. We augment the equation to be $3T(n/3) + n/2$, where we can use Master Theorem (*Case #2*), $T(n) \in \Theta(n \lg n)$.

5. With similar reasoning as in Step #2, we use a recursion tree to get the results $T(n) \in \Theta(n \lg \lg n)$

6. We solve so via the substitution method. We use a recursion tree to get our initial guess.

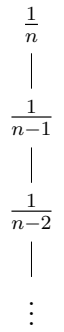


Guessing $T(n) \in \Theta(n)$, we get the following:

$$\begin{aligned} T(n) &= T(n/2) + T(n/4)T(n/8) + n \\ &= c\,n/2 + c\,n/4 + c\,n/8 + n \\ &= \frac{7}{8}\,n + n \\ &\leq c\,n \end{aligned}$$

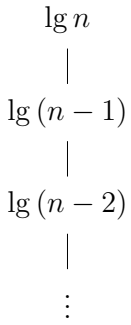
We see this inequality holds for $c \geq 8$. Therefore, $T(n) \in \Theta(n)$.

7. Using a recursion tree,



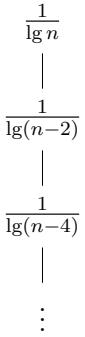
We see that $T(n) \in \Theta(n \lg \lg n)$.

8. Using a recursion tree,



We see that $T(n) \in \Theta(n \lg n)$.

9. Using a recursion tree,



We see that $T(n) \in \Theta(\lg \lg n)$.

10. We guess

$$T(n) \in \Theta(n \lg \lg n)$$

$$\begin{aligned} T(n) &= \sqrt{n}\, T\left(\sqrt{n}\right) + n \\ &= \sqrt{n}\, T\left(c\sqrt{n} \lg \lg \sqrt{n}\right) + n \\ &= c\, n \lg \lg n - c\, n + n \\ &\leq c\, n \lg \lg n \end{aligned}$$

We see this inequality for $c \in \mathbb{R}^+$. Therefore $T(n) \in \Theta(n \lg \lg n)$.

Question #5

Options c and e.

Question #6

1. We choose a root among the vertices, call it v_0. If we choose the kth smallest element, the smaller subtree will have $i-1$ vertices and the larger will have $n-i$ vertices. Summing over all possibilities, we get the following form for b_n:

$$\begin{aligned} b_n &= \sum_{k=1}^{n} b_{k-1} b_{n-k} \\ &= \sum_{k=0}^{n-1} b_k b_k b_{n-k-1} \end{aligned}$$

2.

$$\begin{aligned} B(x) &= \sum_{n=0}^{\infty} b_n x^n \\ &= 1 + \sum_{n=1}^{\infty} b_n x^n \\ &= 1 + \sum_{n=1}^{\infty} \sum_{k=0}^{n-1} b_k b_{n-k-1} x^{n-k-1} \\ &= 1 + x \sum_{n=1}^{\infty} \sum_{k=0}^{n-1} b_k x^k b_{n-k-1} x^{n-k-1} \\ &= 1 + x \sum_{n=0}^{\infty} \sum_{k=0}^{n} b_k x^k b_{n-k} x^{n-k} \\ &= 1 + x\, B(x)^2 \end{aligned}$$

3. We use the Taylor expansion of $\sqrt{1-4x}$, we have the following

$$\begin{aligned}
B(x) &= \frac{1}{2x}\left(1 - \sum_{n=0}^{\infty} \frac{1}{1-2n}\binom{2n}{n}x^n\right) \\
&= -\frac{1}{2x}\sum_{n=1}^{\infty}\sum_{n=1}^{\infty} \frac{1}{1-2n}\binom{2n}{n}x^n \\
&= \frac{1}{2}\sum_{n=1}^{\infty} \frac{1}{2n-1}\binom{2n}{n}x^{n-1} \\
&= \frac{1}{2}\sum_{n=0}^{\infty} \frac{1}{2n+1}\binom{2n+2}{n+1}x^n
\end{aligned}$$

Therefore $b_n = \frac{1}{n+1}\binom{2n}{n}$.

4.

$$\begin{aligned}
b_n &= \frac{1}{n+1}\frac{(2n)!}{n!n!} \\
&\approx \frac{1}{n+1}\frac{\sqrt{4\,\pi n}(2n/e)^{2n}}{2\,\pi n(n/e)^{2n}} \\
&= \frac{1}{n+1}\frac{4^n}{\sqrt{\pi n}} \\
&= \left(\frac{1}{n} + \left(\frac{1}{n+1} - \frac{1}{n}\right)\right)\frac{4^n}{\sqrt{\pi n}} \\
&= \left(\frac{1}{n} - \frac{1}{n^2+n}\right)\frac{4^n}{\sqrt{\pi n}} \\
&= \frac{1}{n}\left(1 - \frac{1}{n+1}\right)\frac{4^n}{\sqrt{\pi n}} \\
&= \frac{4^n}{\sqrt{\pi}n^{3/2}}(1 + O(1/n))
\end{aligned}$$

Homework #7

Analysis of Algorithms

Illya Starikov

Due Date: October 20th 2017

```
1  class Node:
2      value = -1
3      left_child, right_right = None, None
4
5      def __init__(self, value):
6          self.value = value
7          self.left_child, self.right_child = None, None
8
9      @property
10     def number_of_children(self):
11         if self.left_child is not None and self.right_child is not None:
12             return 2
13         elif self.left_child is not None or self.right_child is not None:
14             return 1
15         else:
16             return 0
17
18     @property
19     def valid_children(self):
20         number_of_children = self.number_of_children
21
22         if number_of_children == 0:
23             return None
24         elif number_of_children == 1:
25             if self.left_child is not None:
26                 return self.left_child
27             else:
28                 return self.right_child
29         else:
30             return (self.left_child, self.right_child)
31
32     def __str__(self):
33         return str(self.value)
34
35
36 class BinarySearchTree:
```

```
37      root = -1
38
39      def __init__(self):
40          self.root = None
41
42      def insert(self, value):
43          if self.root is None:
44              self.root = Node(value)
45          else:
46              self.__insert_node(self.root, value)
47
48      def exists(self, value):
49          return False if self.__find_node(self.root, value) is None else
        True
50
51      def delete(self, value):
52          self.root = self.__delete_node(self.root, value)
53
54      def print_tree(self):
55          if not self.root:
56              return
57
58          current_level = [self.root]
59
60          while current_level:
61
62              print(' '.join(str(node) for node in current_level))
63
64              next_level = list()
65              for n in current_level:
66                  if n.left_child:
67                      next_level.append(n.left_child)
68                  if n.right_child:
69                      next_level.append(n.right_child)
70                  current_level = next_level
71
72      # MARK: Private Methods
73      def __insert_node(self, current_node, value):
74          # for the values <= `value', we put on the left side of the tree
75          if value <= current_node.value:
76              if current_node.left_child is None:
77                  current_node.left_child = Node(value)
78              else:
79                  self.__insert_node(current_node.left_child, value)
80
81          # for values > `value', we put on the right side of the tree
82          else:
83              if current_node.right_child is None:
84                  current_node.right_child = Node(value)
85              else:
86                  self.__insert_node(current_node.right_child, value)
87
```

```
88      def __find_node(self, current_node, value):
89          if current_node is None:
90              return None
91          elif value < current_node.value:
92              return self.__find_node(current_node.left_child, value)
93          elif value > current_node.value:
94              return self.__find_node(current_node.right_child, value)
95          else:
96              return current_node
97
98      def __delete_node(self, current_node, key):
99          if not current_node:
100             return current_node
101
102         if current_node.value > key:
103             current_node.left_child = self.__delete_node(current_node.
    left_child, key)
104         elif current_node.value < key:
105             current_node.right_child = self.__delete_node(current_node.
    right_child, key)
106         else:
107             if not current_node.left_child:
108                 right_child = current_node.right_child
109                 del current_node
110                 return right_child
111
112             elif not current_node.right_child:
113                 left_child = current_node.left_child
114                 del current_node
115                 return left_child
116
117             else:
118                 successor = current_node.right_child
119                 while successor.left_child:
120                     successor = successor.left_child
121
122                 current_node.value = successor.value
123                 current_node.right_child = self.__delete_node(current_node
    .right_child, successor.value)
124
125         return current_node
```

Question #1

With the following elements,

42	2555	4830	7516
104	2718	4883	7604
321	2849	4961	7632
578	2954	5016	7706
600	2974	5119	7754
734	3063	5136	7989
929	3159	5246	8106
1004	3262	5253	8126
1120	3397	5502	8135
1128	3405	5520	8204
1149	3476	5550	8223
1213	3694	5644	8338
1323	3739	5675	8359
1347	3902	6082	8565
1512	4310	6306	8611
1522	4466	6425	8740
1730	4475	6562	8890
1831	4494	6570	8961
1853	4584	6653	9300
1886	4644	6712	9304
1899	4699	6940	9509
2040	4750	6966	9614
2082	4769	7164	9762
2164	4783	7324	9971
2253	4786	7499	9985

Problem #1.1

Determining if elements in the binary search tree are there. All these values should be true.

```
1  Element  3739 Found: True
2  Element  5550 Found: True
3  Element   600 Found: True
4  Element  3262 Found: True
5  Element  8611 Found: True
6  Element  4750 Found: True
7  Element  5246 Found: True
8  Element  6712 Found: True
9  Element  1323 Found: True
10 Element  1831 Found: True
11 Element  5520 Found: True
12 Element  3063 Found: True
13 Element  9300 Found: True
14 Element  1004 Found: True
15 Element  7164 Found: True
16 Element  4466 Found: True
17 Element  8890 Found: True
```

```
Element  8338 Found: True
Element  1853 Found: True
Element  8223 Found: True
Element  3159 Found: True
Element  4644 Found: True
Element  7706 Found: True
Element  8740 Found: True
Element  1120 Found: True
Element  2040 Found: True
Element  6653 Found: True
Element   578 Found: True
Element  9762 Found: True
Element  8565 Found: True
Element  7604 Found: True
Element  9985 Found: True
Element  4961 Found: True
Element   929 Found: True
Element  2082 Found: True
Element  5253 Found: True
Element  3405 Found: True
Element  7516 Found: True
Element  6306 Found: True
Element  8106 Found: True
Element  1347 Found: True
Element  4310 Found: True
Element  4584 Found: True
Element  2954 Found: True
Element  9614 Found: True
Element  2253 Found: True
Element  1522 Found: True
Element  8135 Found: True
Element  4475 Found: True
Element  1512 Found: True
Element  9304 Found: True
Element  6966 Found: True
Element   321 Found: True
Element  2974 Found: True
Element  6940 Found: True
Element  7754 Found: True
Element  1213 Found: True
Element  1149 Found: True
Element  8961 Found: True
Element  8359 Found: True
Element  4883 Found: True
Element  7499 Found: True
Element  4786 Found: True
Element  8126 Found: True
Element    42 Found: True
Element  4699 Found: True
Element  3902 Found: True
Element  1730 Found: True
Element  4494 Found: True
```

```
Element  5502 Found: True
Element   104 Found: True
Element  4830 Found: True
Element  6082 Found: True
Element  1899 Found: True
Element  5136 Found: True
Element  5644 Found: True
Element  2849 Found: True
Element  5675 Found: True
Element  8204 Found: True
Element  7632 Found: True
Element  1886 Found: True
Element  5119 Found: True
Element  5016 Found: True
Element  6562 Found: True
Element  3397 Found: True
Element  9971 Found: True
Element  3694 Found: True
Element  4783 Found: True
Element  2718 Found: True
Element   734 Found: True
Element  7989 Found: True
Element  7324 Found: True
Element  1128 Found: True
Element  6425 Found: True
Element  4769 Found: True
Element  2555 Found: True
Element  3476 Found: True
Element  9509 Found: True
Element  2164 Found: True
Element  6570 Found: True
```

Problem #1.2

Determining if elements **not** in binary search tree are found. These values should be false.

```
Element  9170 Found: False
Element  1059 Found: False
Element  6807 Found: False
Element  7990 Found: False
Element  9177 Found: False
Element  9933 Found: False
Element  9290 Found: False
Element  5479 Found: False
Element   611 Found: False
Element  5458 Found: False
Element  1563 Found: False
Element  5870 Found: False
Element  3134 Found: False
Element   178 Found: False
Element  3890 Found: False
```

```
Element  7225 Found: False
Element  9820 Found: False
Element  2292 Found: False
Element  5313 Found: False
Element  4489 Found: False
Element  3835 Found: False
Element  2543 Found: False
Element  5679 Found: False
Element    46 Found: False
Element  1812 Found: False
Element  4628 Found: False
Element  7310 Found: False
Element  6839 Found: False
Element  5411 Found: False
Element  2929 Found: False
Element  2351 Found: False
Element  9107 Found: False
Element  7498 Found: False
Element  5810 Found: False
Element  6692 Found: False
Element  3986 Found: False
Element  2941 Found: False
Element  9506 Found: False
Element  3485 Found: False
Element  1113 Found: False
Element  4268 Found: False
Element  9848 Found: False
Element  1525 Found: False
Element  7304 Found: False
Element  1792 Found: False
Element  2707 Found: False
Element  4111 Found: False
Element  7696 Found: False
Element  4897 Found: False
Element   120 Found: False
Element  9588 Found: False
Element   819 Found: False
Element  5893 Found: False
Element  9990 Found: False
Element  6444 Found: False
Element  2619 Found: False
Element  3091 Found: False
Element  8431 Found: False
Element  7965 Found: False
Element  4863 Found: False
Element  9778 Found: False
Element  9601 Found: False
Element  8084 Found: False
Element  7459 Found: False
Element  7299 Found: False
Element     1 Found: False
Element  6819 Found: False
```

```
Element  9085 Found: False
Element  5574 Found: False
Element  8748 Found: False
Element  8790 Found: False
Element  6042 Found: False
Element  5852 Found: False
Element  5341 Found: False
Element  7671 Found: False
Element  8456 Found: False
Element  1818 Found: False
Element  6918 Found: False
Element  9723 Found: False
Element  3980 Found: False
Element  6496 Found: False
Element  5121 Found: False
Element  2553 Found: False
Element  7316 Found: False
Element  1461 Found: False
Element  3896 Found: False
Element  9382 Found: False
Element  8610 Found: False
Element  5181 Found: False
Element  3305 Found: False
Element   549 Found: False
Element  7306 Found: False
Element  4612 Found: False
Element  9525 Found: False
Element  4847 Found: False
Element  6420 Found: False
Element  2643 Found: False
Element  4538 Found: False
Element  5934 Found: False
Element  7969 Found: False
```

Problem #1.3

Deleting entire tree. This output is ≈ 2000, so please refer to output for entire tree.

```
Current Tree Before Deleting Element 4769
3739
2974 7632
1149 3405 4783 9509
734 1899 3063 3694 4475 6562 8890 9985
42 1120 1512 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
104 1004 1128 1347 1522 2849 3159 3397 4310 4494 4750 5016 5675 7324 7604
    8565 8961 9614
321 929 1213 1853 2040 4466 4644 4769 4830 5520 6082 6653 7516 8126 8611
    9300 9762
578 1323 1831 1886 2718 4699 4786 4883 5502 5550 6306 6570 7164 7989 8223
600 1730 2253 4961 5253 6425 6940 7706 8106 8135 8338
2164 2555 5136 6712 6966 7754 8204 8359
2082 5119 5246
```

```
13
14 Current Tree After Deleting Element 4769
15 3739
16 2974 7632
17 1149 3405 4783 9509
18 734 1899 3063 3694 4475 6562 8890 9985
19 42 1120 1512 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
20 104 1004 1128 1347 1522 2849 3159 3397 4310 4494 4750 5016 5675 7324 7604
       8565 8961 9614
21 321 929 1213 1853 2040 4466 4644 4830 5520 6082 6653 7516 8126 8611 9300
       9762
22 578 1323 1831 1886 2718 4699 4786 4883 5502 5550 6306 6570 7164 7989 8223
23 600 1730 2253 4961 5253 6425 6940 7706 8106 8135 8338
24 2164 2555 5136 6712 6966 7754 8204 8359
25 2082 5119 5246
26 --------------------------------------------------------------------------------

27 Current Tree Before Deleting Element 1853
28 3739
29 2974 7632
30 1149 3405 4783 9509
31 734 1899 3063 3694 4475 6562 8890 9985
32 42 1120 1512 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
33 104 1004 1128 1347 1522 2849 3159 3397 4310 4494 4750 5016 5675 7324 7604
       8565 8961 9614
34 321 929 1213 1853 2040 4466 4644 4830 5520 6082 6653 7516 8126 8611 9300
       9762
35 578 1323 1831 1886 2718 4699 4786 4883 5502 5550 6306 6570 7164 7989 8223
36 600 1730 2253 4961 5253 6425 6940 7706 8106 8135 8338
37 2164 2555 5136 6712 6966 7754 8204 8359
38 2082 5119 5246
39
40 Current Tree After Deleting Element 1853
41 3739
42 2974 7632
43 1149 3405 4783 9509
44 734 1899 3063 3694 4475 6562 8890 9985
45 42 1120 1512 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
46 104 1004 1128 1347 1522 2849 3159 3397 4310 4494 4750 5016 5675 7324 7604
       8565 8961 9614
47 321 929 1213 1886 2040 4466 4644 4830 5520 6082 6653 7516 8126 8611 9300
       9762
48 578 1323 1831 2718 4699 4786 4883 5502 5550 6306 6570 7164 7989 8223
49 600 1730 2253 4961 5253 6425 6940 7706 8106 8135 8338
50 2164 2555 5136 6712 6966 7754 8204 8359
51 2082 5119 5246
52 --------------------------------------------------------------------------------

53 Current Tree Before Deleting Element 4699
54 3739
55 2974 7632
56 1149 3405 4783 9509
```

```
734 1899 3063 3694 4475 6562 8890 9985
42 1120 1512 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
104 1004 1128 1347 1522 2849 3159 3397 4310 4494 4750 5016 5675 7324 7604
    8565 8961 9614
321 929 1213 1886 2040 4466 4644 4830 5520 6082 6653 7516 8126 8611 9300
    9762
578 1323 1831 2718 4699 4786 4883 5502 5550 6306 6570 7164 7989 8223
600 1730 2253 4961 5253 6425 6940 7706 8106 8135 8338
2164 2555 5136 6712 6966 7754 8204 8359
2082 5119 5246

Current Tree After Deleting Element 4699
3739
2974 7632
1149 3405 4783 9509
734 1899 3063 3694 4475 6562 8890 9985
42 1120 1512 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
104 1004 1128 1347 1522 2849 3159 3397 4310 4494 4750 5016 5675 7324 7604
    8565 8961 9614
321 929 1213 1886 2040 4466 4644 4830 5520 6082 6653 7516 8126 8611 9300
    9762
578 1323 1831 2718 4786 4883 5502 5550 6306 6570 7164 7989 8223
600 1730 2253 4961 5253 6425 6940 7706 8106 8135 8338
2164 2555 5136 6712 6966 7754 8204 8359
2082 5119 5246
--------------------------------------------------------------------------------

Current Tree Before Deleting Element 2082
3739
2974 7632
1149 3405 4783 9509
734 1899 3063 3694 4475 6562 8890 9985
42 1120 1512 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
104 1004 1128 1347 1522 2849 3159 3397 4310 4494 4750 5016 5675 7324 7604
    8565 8961 9614
321 929 1213 1886 2040 4466 4644 4830 5520 6082 6653 7516 8126 8611 9300
    9762
578 1323 1831 2718 4786 4883 5502 5550 6306 6570 7164 7989 8223
600 1730 2253 4961 5253 6425 6940 7706 8106 8135 8338
2164 2555 5136 6712 6966 7754 8204 8359
2082 5119 5246

Current Tree After Deleting Element 2082
3739
2974 7632
1149 3405 4783 9509
734 1899 3063 3694 4475 6562 8890 9985
42 1120 1512 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
104 1004 1128 1347 1522 2849 3159 3397 4310 4494 4750 5016 5675 7324 7604
    8565 8961 9614
321 929 1213 1886 2040 4466 4644 4830 5520 6082 6653 7516 8126 8611 9300
    9762
```

```
578 1323 1831 2718 4786 4883 5502 5550 6306 6570 7164 7989 8223
600 1730 2253 4961 5253 6425 6940 7706 8106 8135 8338
2164 2555 5136 6712 6966 7754 8204 8359
5119 5246
--------------------------------------------------------------------------------

Current Tree Before Deleting Element 4961
3739
2974 7632
1149 3405 4783 9509
734 1899 3063 3694 4475 6562 8890 9985
42 1120 1512 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
104 1004 1128 1347 1522 2849 3159 3397 4310 4494 4750 5016 5675 7324 7604
    8565 8961 9614
321 929 1213 1886 2040 4466 4644 4830 5520 6082 6653 7516 8126 8611 9300
    9762
578 1323 1831 2718 4786 4883 5502 5550 6306 6570 7164 7989 8223
600 1730 2253 4961 5253 6425 6940 7706 8106 8135 8338
2164 2555 5136 6712 6966 7754 8204 8359
5119 5246

Current Tree After Deleting Element 4961
3739
2974 7632
1149 3405 4783 9509
734 1899 3063 3694 4475 6562 8890 9985
42 1120 1512 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
104 1004 1128 1347 1522 2849 3159 3397 4310 4494 4750 5016 5675 7324 7604
    8565 8961 9614
321 929 1213 1886 2040 4466 4644 4830 5520 6082 6653 7516 8126 8611 9300
    9762
578 1323 1831 2718 4786 4883 5502 5550 6306 6570 7164 7989 8223
600 1730 2253 5253 6425 6940 7706 8106 8135 8338
2164 2555 5136 6712 6966 7754 8204 8359
5119 5246
--------------------------------------------------------------------------------

Current Tree Before Deleting Element 1512
3739
2974 7632
1149 3405 4783 9509
734 1899 3063 3694 4475 6562 8890 9985
42 1120 1512 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
104 1004 1128 1347 1522 2849 3159 3397 4310 4494 4750 5016 5675 7324 7604
    8565 8961 9614
321 929 1213 1886 2040 4466 4644 4830 5520 6082 6653 7516 8126 8611 9300
    9762
578 1323 1831 2718 4786 4883 5502 5550 6306 6570 7164 7989 8223
600 1730 2253 5253 6425 6940 7706 8106 8135 8338
2164 2555 5136 6712 6966 7754 8204 8359
5119 5246

```

```
Current Tree After Deleting Element 1512
3739
2974 7632
1149 3405 4783 9509
734 1899 3063 3694 4475 6562 8890 9985
42 1120 1522 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
104 1004 1128 1347 1886 2849 3159 3397 4310 4494 4750 5016 5675 7324 7604
    8565 8961 9614
321 929 1213 1831 2040 4466 4644 4830 5520 6082 6653 7516 8126 8611 9300
    9762
578 1323 1730 2718 4786 4883 5502 5550 6306 6570 7164 7989 8223
600 2253 5253 6425 6940 7706 8106 8135 8338
2164 2555 5136 6712 6966 7754 8204 8359
5119 5246
--------------------------------------------------------------------------------

Current Tree Before Deleting Element 1730
3739
2974 7632
1149 3405 4783 9509
734 1899 3063 3694 4475 6562 8890 9985
42 1120 1522 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
104 1004 1128 1347 1886 2849 3159 3397 4310 4494 4750 5016 5675 7324 7604
    8565 8961 9614
321 929 1213 1831 2040 4466 4644 4830 5520 6082 6653 7516 8126 8611 9300
    9762
578 1323 1730 2718 4786 4883 5502 5550 6306 6570 7164 7989 8223
600 2253 5253 6425 6940 7706 8106 8135 8338
2164 2555 5136 6712 6966 7754 8204 8359
5119 5246

Current Tree After Deleting Element 1730
3739
2974 7632
1149 3405 4783 9509
734 1899 3063 3694 4475 6562 8890 9985
42 1120 1522 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
104 1004 1128 1347 1886 2849 3159 3397 4310 4494 4750 5016 5675 7324 7604
    8565 8961 9614
321 929 1213 1831 2040 4466 4644 4830 5520 6082 6653 7516 8126 8611 9300
    9762
578 1323 2718 4786 4883 5502 5550 6306 6570 7164 7989 8223
600 2253 5253 6425 6940 7706 8106 8135 8338
2164 2555 5136 6712 6966 7754 8204 8359
5119 5246
--------------------------------------------------------------------------------

Current Tree Before Deleting Element 2849
3739
2974 7632
1149 3405 4783 9509
734 1899 3063 3694 4475 6562 8890 9985
```

```
42 1120 1522 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
104 1004 1128 1347 1886 2849 3159 3397 4310 4494 4750 5016 5675 7324 7604
    8565 8961 9614
321 929 1213 1831 2040 4466 4644 4830 5520 6082 6653 7516 8126 8611 9300
    9762
578 1323 2718 4786 4883 5502 5550 6306 6570 7164 7989 8223
600 2253 5253 6425 6940 7706 8106 8135 8338
2164 2555 5136 6712 6966 7754 8204 8359
5119 5246

Current Tree After Deleting Element 2849
3739
2974 7632
1149 3405 4783 9509
734 1899 3063 3694 4475 6562 8890 9985
42 1120 1522 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
104 1004 1128 1347 1886 2040 3159 3397 4310 4494 4750 5016 5675 7324 7604
    8565 8961 9614
321 929 1213 1831 2718 4466 4644 4830 5520 6082 6653 7516 8126 8611 9300
    9762
578 1323 2253 4786 4883 5502 5550 6306 6570 7164 7989 8223
600 2164 2555 5253 6425 6940 7706 8106 8135 8338
5136 6712 6966 7754 8204 8359
5119 5246
--------------------------------------------------------------------------------

Current Tree Before Deleting Element 5520
3739
2974 7632
1149 3405 4783 9509
734 1899 3063 3694 4475 6562 8890 9985
42 1120 1522 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
104 1004 1128 1347 1886 2040 3159 3397 4310 4494 4750 5016 5675 7324 7604
    8565 8961 9614
321 929 1213 1831 2718 4466 4644 4830 5520 6082 6653 7516 8126 8611 9300
    9762
578 1323 2253 4786 4883 5502 5550 6306 6570 7164 7989 8223
600 2164 2555 5253 6425 6940 7706 8106 8135 8338
5136 6712 6966 7754 8204 8359
5119 5246

Current Tree After Deleting Element 5520
3739
2974 7632
1149 3405 4783 9509
734 1899 3063 3694 4475 6562 8890 9985
42 1120 1522 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
104 1004 1128 1347 1886 2040 3159 3397 4310 4494 4750 5016 5675 7324 7604
    8565 8961 9614
321 929 1213 1831 2718 4466 4644 4830 5550 6082 6653 7516 8126 8611 9300
    9762
578 1323 2253 4786 4883 5502 6306 6570 7164 7989 8223
```

```
600 2164 2555 5253 6425 6940 7706 8106 8135 8338
5136 6712 6966 7754 8204 8359
5119 5246
--------------------------------------------------------------------------------

Current Tree Before Deleting Element 8890
3739
2974 7632
1149 3405 4783 9509
734 1899 3063 3694 4475 6562 8890 9985
42 1120 1522 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
104 1004 1128 1347 1886 2040 3159 3397 4310 4494 4750 5016 5675 7324 7604
    8565 8961 9614
321 929 1213 1831 2718 4466 4644 4830 5550 6082 6653 7516 8126 8611 9300
    9762
578 1323 2253 4786 4883 5502 6306 6570 7164 7989 8223
600 2164 2555 5253 6425 6940 7706 8106 8135 8338
5136 6712 6966 7754 8204 8359
5119 5246

Current Tree After Deleting Element 8890
3739
2974 7632
1149 3405 4783 9509
734 1899 3063 3694 4475 6562 8961 9985
42 1120 1522 2954 3262 3476 3902 4584 5644 7499 8740 9304 9971
104 1004 1128 1347 1886 2040 3159 3397 4310 4494 4750 5016 5675 7324 7604
    8565 9300 9614
321 929 1213 1831 2718 4466 4644 4830 5550 6082 6653 7516 8126 8611 9762
578 1323 2253 4786 4883 5502 6306 6570 7164 7989 8223
600 2164 2555 5253 6425 6940 7706 8106 8135 8338
5136 6712 6966 7754 8204 8359
5119 5246
```

Question #2

```python
class Graph:
    adjacency_list = {}
    is_directed = False

    def __init__(self, is_directed=False):
        self.adjacency_list = {}
        self.is_directed = is_directed

    def add_edge(self, start, destination):
        self.adjacency_list[start].add(destination)

        if self.is_directed:
            self.adjacency_list[destination].add(start)

```

```
15      def add_node(self, node):
16          if node not in self.adjacency_list:
17              self.adjacency_list[node] = set()
18
19      def print_graph(self):
20          max_row, max_column = self.max_dimensions
21          list_version = [[0 for i in range(max_column)] for j in range(
        max_row)]
22
23          for source, destination_set in self.adjacency_list.items():
24              for destination in destination_set:
25                  list_version[source][destination] = 1
26
27          print(matrix(list_version))
28
29      @property
30      def max_dimensions(self):
31          max_element = max(self.adjacency_list.keys())
32
33          for _, value in self.adjacency_list.items():
34              value = list(value)
35
36              if value:
37                  if max(value) > max_element:
38                      max_element = max(value)
39
40          return max_element + 1, max_element + 1
41
42      def depth_first_search(self, start_node, preserve_order=True):
43          if preserve_order:
44              colors = {}
45              for vertex in self.adjacency_list:
46                  colors[vertex] = "white"
47
48              return self.__depth_first_search_preserve_order(start_node,
        colors, None)
49          else:
50              return self.__depth_first_search(start_node, None)
51
52      def breadth_first_search(self, start_node, preserve_order=True):
53          if preserve_order:
54              return self.__breadth_first_search_preserve_order(start_node)
55          else:
56              return self.__breadth_first_search(start_node)
57
58      def __depth_first_search(self, start_node, visited_nodes):
59          if visited_nodes is None:
60              visited_nodes = set()
61
62          visited_nodes.add(start_node)
63          for unvisited_node in self.adjacency_list[start_node] -
        visited_nodes:
```

```
64              self.__depth_first_search(unvisited_node, visited_nodes)
65
66          return visited_nodes
67
68      def __depth_first_search_preserve_order(self, start_node, colors,
    visited_nodes):
69          if visited_nodes is None:
70              visited_nodes = []
71
72          if colors[start_node] is "white":
73              colors[start_node] = "grey"
74              visited_nodes += [start_node]
75
76          for unvisited_node in [x for x in self.adjacency_list[start_node]
    if x not in visited_nodes]:
77              if colors[unvisited_node] is "white":
78                  self.__depth_first_search_preserve_order(unvisited_node,
    colors, visited_nodes)
79
80          colors[start_node] = "black"
81
82          return visited_nodes
83
84      def __breadth_first_search(self, start):
85          visited, queue = set(), [start]
86
87          while queue:
88              vertex = queue.pop(0)
89
90              if vertex not in visited:
91                  visited.add(vertex)
92                  queue.extend(self.adjacency_list[vertex] - visited)
93
94          return visited
95
96      def __breadth_first_search_preserve_order(self, start):
97          colors = {}
98
99          for vertex in self.adjacency_list:
100             colors[vertex] = "white"
101
102         queue = [start]
103         visited_nodes = [start]
104
105         while queue:
106             node = queue.pop(0)
107
108             for unvisited_node in list(self.adjacency_list[node]):
109                 if colors[unvisited_node] is "white":
110                     colors[unvisited_node] = "grey"
111                     queue.insert(0, unvisited_node)
112                     visited_nodes += [unvisited_node]
```

```
            colors[node] = "black"

        return visited_nodes
```

We test examples with the following adjacency list.

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Problem #2.1

For Depth-First Search, we get the following ordering (with the root taken at random).

$$3 \to 4 \to 2 \to 8 \to 0 \to 9 \to 1 \to 6 \to 5 \to 7$$
$$0 \to 9 \to 1 \to 2 \to 8 \to 3 \to 4 \to 6 \to 5 \to 7$$
$$8 \to 0 \to 9 \to 1 \to 2 \to 3 \to 4 \to 6 \to 5 \to 7$$
$$5 \to 4 \to 2 \to 8 \to 0 \to 9 \to 1 \to 3 \to 6 \to 7$$
$$0 \to 9 \to 1 \to 2 \to 8 \to 3 \to 4 \to 6 \to 5 \to 7$$
$$6 \to 0 \to 9 \to 1 \to 2 \to 8 \to 3 \to 4 \to 5 \to 7$$
$$3 \to 4 \to 2 \to 8 \to 0 \to 9 \to 1 \to 6 \to 5 \to 7$$
$$5 \to 4 \to 2 \to 8 \to 0 \to 9 \to 1 \to 3 \to 6 \to 7$$
$$7 \to 8 \to 0 \to 9 \to 1 \to 2 \to 3 \to 4 \to 6 \to 5$$
$$9 \to 1 \to 2 \to 8 \to 0 \to 3 \to 4 \to 6 \to 5 \to 7$$
$$3 \to 4 \to 2 \to 8 \to 0 \to 9 \to 1 \to 6 \to 5 \to 7$$
$$8 \to 0 \to 9 \to 1 \to 2 \to 3 \to 4 \to 6 \to 5 \to 7$$
$$5 \to 4 \to 2 \to 8 \to 0 \to 9 \to 1 \to 3 \to 6 \to 7$$
$$0 \to 9 \to 1 \to 2 \to 8 \to 3 \to 4 \to 6 \to 5 \to 7$$
$$2 \to 8 \to 0 \to 9 \to 1 \to 3 \to 4 \to 6 \to 5 \to 7$$

Problem #2.2

For Breadth-First Search, we get the following ordering (with the root taken at random).

$$3 \to 4 \to 6 \to 0 \to 8 \to 5 \to 7 \to 9 \to 1 \to 2$$
$$0 \to 9 \to 3 \to 4 \to 1 \to 2 \to 8 \to 5 \to 6 \to 7$$
$$8 \to 0 \to 3 \to 5 \to 6 \to 7 \to 9 \to 1 \to 4 \to 2$$
$$5 \to 4 \to 2 \to 6 \to 0 \to 8 \to 3 \to 7 \to 9 \to 1$$
$$0 \to 9 \to 3 \to 4 \to 1 \to 2 \to 8 \to 5 \to 6 \to 7$$
$$6 \to 0 \to 8 \to 3 \to 5 \to 7 \to 9 \to 1 \to 4 \to 2$$
$$3 \to 4 \to 6 \to 0 \to 8 \to 5 \to 7 \to 9 \to 1 \to 2$$
$$5 \to 4 \to 2 \to 6 \to 0 \to 8 \to 3 \to 7 \to 9 \to 1$$
$$7 \to 8 \to 0 \to 3 \to 5 \to 6 \to 9 \to 1 \to 4 \to 2$$
$$9 \to 1 \to 4 \to 6 \to 0 \to 8 \to 3 \to 5 \to 7 \to 2$$
$$3 \to 4 \to 6 \to 0 \to 8 \to 5 \to 7 \to 9 \to 1 \to 2$$
$$8 \to 0 \to 3 \to 5 \to 6 \to 7 \to 9 \to 1 \to 4 \to 2$$
$$5 \to 4 \to 2 \to 6 \to 0 \to 8 \to 3 \to 7 \to 9 \to 1$$
$$0 \to 9 \to 3 \to 4 \to 1 \to 2 \to 8 \to 5 \to 6 \to 7$$
$$2 \to 8 \to 0 \to 3 \to 5 \to 6 \to 7 \to 9 \to 1 \to 4$$

Question #3

Our graph is exactly the same as the previous version; however, this one has the boolean property (`is_directed` set to false). We test examples with the following adjacency list.

$$\begin{bmatrix}
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

Problem #3.1

For Depth-First Search, we get the following ordering (with the root taken at random).

$$3 \to 9 \to 0 \to 8 \to 1 \to 4 \to 2 \to 5 \to 7 \to 6$$
$$9 \to 0 \to 8 \to 1 \to 3 \to 4 \to 2 \to 5 \to 7 \to 6$$
$$2 \to 0 \to 8 \to 1 \to 3 \to 9 \to 4 \to 5 \to 6 \to 7$$
$$4 \to 9 \to 0 \to 8 \to 1 \to 3 \to 6 \to 7 \to 2 \to 5$$
$$9 \to 0 \to 8 \to 1 \to 3 \to 4 \to 2 \to 5 \to 7 \to 6$$
$$0 \to 8 \to 1 \to 3 \to 9 \to 4 \to 2 \to 5 \to 7 \to 6$$
$$4 \to 9 \to 0 \to 8 \to 1 \to 3 \to 6 \to 7 \to 2 \to 5$$
$$2 \to 0 \to 8 \to 1 \to 3 \to 9 \to 4 \to 5 \to 6 \to 7$$
$$4 \to 9 \to 0 \to 8 \to 1 \to 3 \to 6 \to 7 \to 2 \to 5$$
$$1 \to 8 \to 0 \to 9 \to 3 \to 4 \to 2 \to 5 \to 7 \to 6$$
$$0 \to 8 \to 1 \to 3 \to 9 \to 4 \to 2 \to 5 \to 7 \to 6$$
$$7 \to 8 \to 0 \to 9 \to 3 \to 1 \to 4 \to 2 \to 5 \to 6$$
$$7 \to 8 \to 0 \to 9 \to 3 \to 1 \to 4 \to 2 \to 5 \to 6$$
$$5 \to 0 \to 8 \to 1 \to 3 \to 9 \to 4 \to 2 \to 7 \to 6$$
$$4 \to 9 \to 0 \to 8 \to 1 \to 3 \to 6 \to 7 \to 2 \to 5$$

Problem #3.2

For Breadth-First Search, we get the following ordering (with the root taken at random).

$$3 \to 9 \to 1 \to 4 \to 6 \to 7 \to 8 \to 2 \to 0 \to 5$$
$$9 \to 0 \to 3 \to 4 \to 2 \to 5 \to 1 \to 8 \to 7 \to 6$$
$$2 \to 0 \to 8 \to 4 \to 5 \to 7 \to 1 \to 3 \to 9 \to 6$$
$$4 \to 9 \to 2 \to 3 \to 5 \to 1 \to 8 \to 7 \to 0 \to 6$$
$$9 \to 0 \to 3 \to 4 \to 2 \to 5 \to 1 \to 8 \to 7 \to 6$$
$$0 \to 8 \to 9 \to 2 \to 5 \to 4 \to 3 \to 1 \to 7 \to 6$$
$$4 \to 9 \to 2 \to 3 \to 5 \to 1 \to 8 \to 7 \to 0 \to 6$$
$$2 \to 0 \to 8 \to 4 \to 5 \to 7 \to 1 \to 3 \to 9 \to 6$$
$$4 \to 9 \to 2 \to 3 \to 5 \to 1 \to 8 \to 7 \to 0 \to 6$$
$$1 \to 8 \to 3 \to 4 \to 7 \to 2 \to 0 \to 5 \to 9 \to 6$$
$$0 \to 8 \to 9 \to 2 \to 5 \to 4 \to 3 \to 1 \to 7 \to 6$$
$$7 \to 8 \to 1 \to 2 \to 3 \to 9 \to 4 \to 6 \to 5 \to 0$$
$$7 \to 8 \to 1 \to 2 \to 3 \to 9 \to 4 \to 6 \to 5 \to 0$$
$$5 \to 0 \to 2 \to 4 \to 9 \to 3 \to 1 \to 8 \to 7 \to 6$$
$$4 \to 9 \to 2 \to 3 \to 5 \to 1 \to 8 \to 7 \to 0 \to 6$$

Question #4

```
1  class ParentheticalOrder(Graph):
2      def print_parenthetical_order(self, start_node):
3          colors = {}
4          for vertex in self.adjacency_list:
5              colors[vertex] = "white"
6
7          for node in sorted(self.adjacency_list):
8              if colors[node] is "white":
9                  self.__print_parenthetical_order(node, colors)
10
11     def __print_parenthetical_order(self, node, colors):
12         colors[node] = "grey"
13         print("({} ".format(node), end='')
14         for neighbor in sorted(self.adjacency_list[node]):
15             if colors[neighbor] is "white":
16                 self.__print_parenthetical_order(neighbor, colors)
17         colors[node] = 'black'
18         print(" {})".format(node), end='')
```

We get the following string:
(u (v (y (x x) y) v) u) (w (z z) w)

Question #5

Question #6

```
class WrestleMania(Graph):
    def determine_valid_rivalry(self):
        rivalry, not_visited = {}, list(self.adjacency_list.keys())

        for vertex in self.adjacency_list:
            rivalry[vertex] = "none"

        while "none" in rivalry.values():
            current_depth, start = 0, not_visited[-1]
            queue = [start]

            while queue:
                current_depth += 1
                node = queue.pop(0)

                for unvisited_node in list(self.adjacency_list[node]):
                    if rivalry[unvisited_node] is "none":
                        if current_depth % 2 == 0:
                            rivalry[unvisited_node] = "good guy"
                        else:
                            rivalry[unvisited_node] = "bad guy"

                        not_visited.remove(unvisited_node)
                        queue.insert(0, unvisited_node)

        for wrestler, adjacency_wrestlers in self.adjacency_list.items():
            for adjacent_wrestler in adjacency_wrestlers:
                if rivalry[wrestler] == rivalry[adjacent_wrestler]:
                    return False

        return True
```

For the following sample inputs

$$X_1 = \begin{bmatrix} & x & y & z \\ x & 1 & 0 & 1 \\ y & 0 & 0 & 1 \\ z & 1 & 1 & 0 \end{bmatrix} \qquad X_2 = \begin{bmatrix} & w & x & y & z \\ w & 0 & 1 & 0 & 1 \\ x & 1 & 0 & 1 & 0 \\ y & 0 & 1 & 0 & 1 \\ z & 1 & 0 & 1 & 0 \end{bmatrix} \qquad X_3 = \begin{bmatrix} & u & v & w & x & y & z \\ u & 0 & 1 & 0 & 0 & 0 & 0 \\ v & 1 & 0 & 0 & 0 & 0 & 0 \\ w & 0 & 0 & 0 & 1 & 0 & 0 \\ x & 0 & 0 & 1 & 0 & 0 & 0 \\ y & 0 & 0 & 0 & 0 & 0 & 1 \\ z & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

```
No configuration could be found.
=======================================
Wrestler y Is A Baby Face
```

```
Wrestler x Is A Heel
Wrestler z Is A Heel
Wrestler w Is A Baby Face
=======================================
Wrestler u Is A Heel
Wrestler w Is A Heel
Wrestler v Is A Baby Face
Wrestler y Is A Heel
Wrestler x Is A Baby Face
Wrestler z Is A Baby Face
```

Question #7

Problem #7.1

For this problem we have three cases.

No Children If the root has no children it has no edges so it cannot disconnect the graph.

One Child The only edge it breaks is paths to itself so it cannot disconnect graph.

≥ 2 *Children* Because there are no cross edges between the children, and because the children nodes are in the subtrees of the root, no path exists between them. Deleting the root will delete the link between the two or more children; thus, disconnecting the graph and making the root G_π an articulation point.

Problem #7.2

Suppose there is a vertex $v \in G_\pi$, where v is a non-root. Furthermore, suppose v has a child s such that there is no back edge from or any descendent of s to ancestor of v. Thus, the removal of v will lead the disconnection of the sub-tree rooted at v from the graph G_π.

Therefore, the absence of an edge between the descendants of s and ancestors of v resulted in the disconnection of the graph after removal of non-root vertex v. This implies v is an articulation point.

Problem #7.3

Because v is discovered before all of its descendants, the only edges that could affect the minimum are ancestors of v. Thus, we can do an augmented depth-first search to determine all of the low values.

```
class ArticulateGraph(Graph):
    time = 0
    times, colors, lows = {}, {}, {}

    def find_minimum(self, start_node, visited_nodes=None):
```

```
6           if visited_nodes is None:
7               self.times, self.colors, self.lowes = {}, {}, {}
8               self.time = 0
9
10              for vertex in self.adjacency_list:
11                  self.colors[vertex] = "white"
12
13              visited_nodes = []
14
15          for unvisited_node in [x for x in self.adjacency_list[start_node]
    if x not in visited_nodes]:
16              if self.colors[unvisited_node] is "white":
17                  self.colors[start_node] = "grey"
18                  self.time += 1
19                  visited_nodes += [start_node]
20
21                  self.times[start_node] = self.time
22                  self.lows[start_node] = self.times[start_node]
23
24                  for adjacent_node in self.adjacency_list[start_node]:
25                      self.find_minimum(adjacent_node, visited_nodes)
26
27                      if self.colors[adjacent_node] is "white":
28                          if self.time[start_node] < self.lows[self.
    adjacency_list]:
29                              self.lows[start_node] = self.times[
    adjacent_node]
30
31          self.colors[start_node] = "black"
32          self.time += 1
33
34          return self.lows
```

With the following graph,

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

We get the following:

$$8 \to 3$$
$$2 \to 2$$
$$3 \to 1$$
$$4 \to 4$$
$$7 \to 9$$

Problem #7.4

After applying the algorithm mentioned above for all $v \in V$, we check to see if $v.low = v.d$. If it is, no descendants of v has a back edge to a proper ancestor of v, implying v is not a articulation point.

Problem #7.5

If there is a cycle from $u \to v \to w$, where u and v have unique partitions, then removing any edge of w does not disconnect the graph. Therefore, the edges from w to u and v cannot be bridges.

Problem #7.6

A simple circuit contains an edge (u, v) if and only if

- Both of its endpoints are articulation points.
- One ifs endpoints is an articulation point and the other is a vertex of degree 1.

Using our previous algorithm above, we can compute this by running `find_minimum(self, v)`, and deciding if one of the two criteria are met.

Problem #7.7

Because we have already stated that nay two edges of any bi-connected components lie on a command cycle, we know all edges are lying inside a components of a graph will not be bridges (via Part E). Therefore, all bi-connected components of a graph partition the non-bridges of said graph.

Problem #7.8

Locate all bridge edges using the algorithm described in Part F. Remove each bridge, the connected components are all biconnected components.

Homework #8

Analysis of Algorithms

Illya Starikov

Due Date: November 27th, 2017

Contents

Listings

Question #1

Let the samples space $S = \{X_{\text{free}} \implies$ prisoner X is going free, $Y_{\text{free}} \implies$ prisoner Y is going free, Z_{fr} prisoner Z is going free, $\}$. Furthermore, let y be event that Y is to be executed. Using Bayes Theorem, we get the following:

$$\begin{aligned}
\Pr(X_{\text{free}}|y) &= \frac{\Pr(y|X_{\text{free}}) \times \Pr(X_{\text{free}})}{\Pr(y \cap X_{\text{free}}) + \Pr(y \cap Y_{\text{free}}) + \Pr(y \cap Z_{\text{free}})} \\
&= \frac{\frac{1}{3} \times \frac{1}{2}}{\frac{1}{3} \times \frac{1}{2} + 0 + \frac{1}{3} \times 1} \\
&= \frac{1}{3}
\end{aligned}$$

We see that the prisoner X still has a probability of $\frac{1}{3}$.

Question #2

$$\text{minimum} = 2^h \qquad \text{maximum} = 2^{h+1} - 1$$

Question #3

A simple way to achieve an $\mathcal{O}(n \lg k)$ algorithm.

1. Sort all of the arrays in ascending order.
2. Create a minimum heap with all of the minimum elements in the k individual lists. Remove these elements from the respective arrays.
3. Remove the minimum element (name it $d_{\min}$) from the heap mentioned in Step 2, and replace it with the minimum from it's respective list. Add $d_{\min}$ to the solution.
4. While the heap is not empty, repeat Step 3.

Listing 1: Problem #8.2

```
#!/usr/local/bin/python3
#
# problem-3.py
# source
#
# Created by Illya Starikov on 11/16/17.
# Copyright 2017. Illya Starikov. All rights reserved.
#

from collections import defaultdict


class MinHeap():
    """A Min-Max Heap Implementation"""
    __heap_list = []
    __current_size = 0

    def __init__(self):
        self.__heaplist = [0]
        self.__current_size = 0

    def heapify(self, list_):
        i = len(list_) // 2
        self.__current_size = len(list_)
        self.__heap_list = [0] + list_[:]
        while (i > 0):
            self.percolate_down(i)
            i = i - 1

    def percolate_down(self, i):
        while (i * 2) <= self.__current_size:
            minimum_child = self.min_child(i)
            if self.__heap_list[i] > self.__heap_list[minimum_child]:
                tmp = self.__heap_list[i]
                self.__heap_list[i] = self.__heap_list[minimum_child]
```

```
                self.__heap_list[minimum_child] = tmp
            i = minimum_child

    def percolate_up(self, i):
        while i // 2 > 0:
            if self.__heap_list[i] < self.__heap_list[i // 2]:
                tmp = self.__heap_list[i // 2]
                self.__heap_list[i // 2] = self.__heap_list[i]
                self.__heap_list[i] = tmp
            i = i // 2

    def insert(self, k):
        self.__heap_list.append(k)
        self.__current_size = self.__current_size + 1
        self.percolate_up(self.__current_size)

    def min_child(self, i):
        if i * 2 + 1 > self.__current_size:
            return i * 2
        else:
            if self.__heap_list[i*2] < self.__heap_list[i*2+1]:
                return i * 2
            else:
                return i * 2 + 1

    def delete_min(self):
        to_return = self.__heap_list[1]

        self.__heap_list[1] = self.__heap_list[self.__current_size]
        self.__current_size = self.__current_size - 1
        self.__heap_list.pop()
        self.percolate_down(1)

        return to_return

    def print_heap(self):
        print(self.__heap_list)

    def __len__(self):
        return self.__current_size

def empty_tree(input_list):
    """Recursively iterate through values in nested lists."""
    for item in input_list:
        if not isinstance(item, list) or not empty_tree(item):
             return False
    return True

def k_way_merge(enumerables):
    heap_values = defaultdict(list)
    heap = MinHeap()
    solution = []
```

```
88
89      for list_ in enumerables:
90          list_ = sorted(list_)
91          min_value = list_.pop(0)
92
93          heap_values[min_value].append(list_)
94
95      heap.heapify(list(heap_values.keys()))
96
97      while len(heap) > 0:
98          minimum = heap.delete_min()
99          solution.append(minimum)
100
101         minimum_list = [] if heap_values[minimum] == [] else
        heap_values[minimum][-1]
102         del heap_values[minimum]
103
104         if minimum_list != []:
105             new_minimum = minimum_list.pop(0)
106             heap_values[new_minimum].append(minimum_list)
107
108             heap.insert(new_minimum)
109
110     return solution
111
112 def main():
113     lists = [[3, 2, 1, 0], [5, 4], [9, 8, 7, 6]]
114     print(k_way_merge(lists))
115
116
117 if __name__ == "__main__":
118     main()
```

Question #4

Recall Stirling's Approximation for factorials:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \tag{1}$$

We skip the $\frac{1}{2}$ and $\frac{1}{n}$ case; for if we can't prove $\frac{1}{2^n} \times n! \in \Omega(n)$, then neither are $\frac{1}{2}$ and $\frac{1}{n}$.

Theorem 1.

$$\forall n \in \mathbb{R}^+, \frac{1}{2^n} \times n! \notin \Omega(n)$$

Proof. Suppose not. That is, suppose $\exists c \in \mathbb{R}, \frac{1}{2^n} \times n! \leq c\,n$. Therefore, $\frac{1}{2^n} \times n!$ must be in the same growth class as n. By definition, the limit of the two functions must be convergent to some number M.

Because we cannot directly take the limit, we use Equation 1. From this we get the result

$$\lim_{n\to\infty} \frac{\frac{1}{2^n} \times n!}{c\,n} = \frac{\frac{1}{2^n} \times \sqrt{2\pi n}\left(\frac{n}{e}\right)^n}{c\,n} = \infty \qquad \forall c \in R^+$$

This has led us to a contradiction. □

Question #5

Proof. Suppose we have n integers of length L (we can make this assumption because we can always pad the leading digits with 0s).

Base Case Solving with $L = 1$ is trivial. Because it assumed our sorting algorithm is correct, this will simply sort the digits.

Inductive Hypothesis Suppose that it is true for $n - 1$; we will show it to be true for n.

When comparing the nth digit, call it d_1 and d_2 from the two respective lists, it appears something like such:

$\cdots$	$\cdots$	d_1	$\cdots$	$\cdots$
$\cdots$	$\cdots$	d_2	$\cdots$	$\cdots$

From this, there are three cases:

$d_1 < d_2$ Then d_1 is placed before d_2.

$d_1 > d_2$ Then d_1 is placed after d_1.

$d_1 = d_2$ Then the numbers go unchanged. **Because our sorting algorithm is stable**, the order is preserved from the lower digits. This is shown below.

$\cdots$	$\cdots$	42	64	$\cdots$
$\cdots$	$\cdots$	42	16	$\cdots$

□

Question #6

We can accomplish $n + \lg n - 2$ runtime as such:

1. Split all elements into pairwise elements ($S \implies (s_1, s_2), (s_3, s_4), \ldots, (s_{n-1}, s_n)$).
2. Compare all of the elements with respect to the smallest element in the pair. This will get the smallest. This requires n iterations.
3. The second smallest value must have been an element that lost to the original minimum, so we cache all of the lost contenders. We then run the previous step on this to get the second minimum. This requires at most $\lg n - 1$ iterations.

We first the following sample input $n = 10$:

342	133	40	365	796	716	354	38	256	614

We get the following output:

$$\begin{aligned} \text{Number of Comparison} &\in \Omega\left(n + \lceil \lg n \rceil - 2\right) \\ &\leq 10 + \lg 10 - 2 \\ &\leq 12 \\ &= 7 \end{aligned}$$

Listing 2: Problem #5

```
#!/usr/local/bin/python3
#
# problem-5.py
# source
#
# Created by Illya Starikov on 11/24/17.
# Copyright 2017. Illya Starikov. All rights reserved.
#

from random import sample
from math import log2, ceil


def static_vars(**kwargs):
    def decorate(func):
        for k in kwargs:
            setattr(func, k, kwargs[k])
        return func
    return decorate

```

```
21
22  def pairwise(iterable):
23      """s -> (s0, s1), (s2, s3), (s4, s5), ..."""
24
25      a = iter(iterable)
26      return list(zip(a, a))
27
28
29  @static_vars(comparison_counter=0)
30  def find_smallest_value(iterable, pairwised=False):
31
32      if iterable != []:
33          if pairwised is False:
34              iterabale_pairwise = pairwise(iterable)
35          else:
36              iterabale_pairwise = iterable
37
38          minimum = iterabale_pairwise.pop(0)
39
40          for element in iterabale_pairwise:
41              find_smallest_value.comparison_counter += 1
42
43              if min(element) < min(minimum):
44                  minimum = element
45
46          return min(minimum), max(minimum), [x for x in iterable if x
        not in minimum], find_smallest_value.comparison_counter
47
48      return None, None, None, None
49
50
51  def find_second_smallest_element(iterable):
52      """Returns second smallest of two inputs"""
53      smallest, pair_of_smallest, new_iterable, _ = find_smallest_value(
        iterable)
54      smallest_second_iteration, _, _, comparison_counter =
        find_smallest_value(new_iterable)
55
56      second_smallest = pair_of_smallest if smallest_second_iteration is
        None else min(pair_of_smallest, smallest_second_iteration)
57      return second_smallest, comparison_counter
58
59
60  def generate_input(of_length):
61      """Generates random input for problem #7. Returns (iterable,
        second_smallest_element)"""
62      input_ = sample(range(of_length * 100), of_length)
63
64      smallest = min(input_)
65      second_smallest = min([x for x in input_ if smallest != x])
66
67      return input_, second_smallest
```

```
68
69
70  def main():
71      length_of_input = 10
72
73      input_iterable, second_smallest_from_input = generate_input(
        length_of_input)
74      second_smallest, comparisons = find_second_smallest_element(
        input_iterable)
75      print("Input: {}".format(input_iterable))
76
77      print("Length: {}\nElements {} = {} => {}\nNumber of Comparisons
        ({}) <= n + lg(n) - 2 ({}) => {}".format(
78          length_of_input,
79          second_smallest_from_input, second_smallest,
        second_smallest_from_input == second_smallest,
80          comparisons, length_of_input + ceil(log2(length_of_input)) - 2,
         (comparisons <= length_of_input + ceil(log2(length_of_input)) - 2)
81          ))
82
83
84  if __name__ == "__main__":
85      main()
```

Question #7

Theorem 2. *The Set-Partition Problem is NP-Complete.*

Proof. It is trivial to prove that The Set-Partition problem is NP. Given two candidate solutions A and $\bar{A}$, verify that following things:

$$\sum_{x \in A} x - \sum_{x \in \bar{A}} x = 0 \quad \text{and} \quad A \cup \bar{A} = S$$

Recall The Subset Sum problem as follows:

> Given a set of natural numbers S and a natural number t, is there a subset of S that sums to t

We will prove that The Set-Partition problem reduces down to The Subset Sum problem.

Suppose we have $p, t \in \mathbb{R}$ and a set P, where t is a particular number and p is $p = \sum_{x \in P} x$. We give input to Set-Partition $P^* = P \cup s - 2t$. From this, we have two cases.

1. If there exists $t \in P$, then remaining numbers in P sum to $s - t$. This satisfies the Set-Partition problem.
2. There exists a partition of X^* into two sets Q, Q^* such that the $\sum_{x \in Q} = \sum_{x \in Q^*} = s - t$. This implies $s - 2t \in Q * \cup Q$. Removing this number, one set sums to t, and we can use the previous case.

Because we have proved The Set-Partition Problem is NP and it reduces to an NP-Complete problem, we conclude that the Set-Partition problem is NP Complete. □

Question #8

Problem #8.1

There exists an algorithm. Because there are only two denominations, we can enumerate all possible solutions. By producing pairs of all values in amount of x and the amount of y, we get a solution that's $\mathcal{O}\left(\text{amount of x} \times \text{amount of y}\right) = \mathcal{O}\left(n^2\right)$.

Listing 3: Problem #8.1

```
#!/usr/local/bin/python3
#
# problem-8-1.py
# source
#
# Created by Illya Starikov on 11/26/17.
# Copyright 2017. Illya Starikov. All rights reserved.
#


def all_combinations(list_one, list_two):
    """Returns all two-element combinations of both lists"""

    all_combinations = []
    for element_one in list_one:
        for element_two in list_two:
            all_combinations += [[element_one, element_two]]

    return all_combinations


def bonnie_and_clyde(x_worth, y_worth, x_amount, y_amount):
    """Literally a brute force solution"""
    all_combinations_of_x_y = all_combinations(range(x_amount), range(
    y_amount))

    for candidate_x, candidate_y in all_combinations_of_x_y:
        if (candidate_x * x_worth + candidate_y * y_worth) == (x_worth
    * (x_amount - candidate_x) + y_worth * (y_amount - candidate_y)):
            return (True, candidate_x, candidate_y)

    return (False, None, None)


def main():
    # Impossible Case
    x_worth, y_worth = 100, 7
    x_amount, y_amount = 1, 2

    print(bonnie_and_clyde(x_worth, y_worth, x_amount, y_amount))

    # Possible Case (Literally one of each)
```

```
    x_worth, y_worth = 100, 7
    x_amount, y_amount = 2, 2

    print(bonnie_and_clyde(x_worth, y_worth, x_amount, y_amount))

if __name__ == "__main__":
    main()
```

Problem #8.2

There exists an algorithm. And it works like such:

1. Sort the coins in ascending order.
2. Pop off the largest coin, give it to Bonnie.
3. Keep taking smallest coins and assign them to Clyde. If finished, and still have coins to assign, go to Step 2.
4. If at any there is not enough coins to match Bonnie when assigning to Clyde in Step 3, there exists no such configuration.

Listing 4: Problem #8.2

```
#!/usr/local/bin/python3
#
# problem-8-2.py
# source
#
# Created by Illya Starikov on 11/26/17.
# Copyright 2017. Illya Starikov. All rights reserved.
#


def bonnie_and_clyde(coins):
    coins_sorted = sorted(coins)

    bonnie_coins, clyde_coins = [], []

    while coins_sorted != []:
        bonnie_coins += [coins_sorted.pop()]

        while sum(bonnie_coins) != sum(clyde_coins):
            if not coins_sorted:
                return [None, None]

            clyde_coins += [coins_sorted.pop(0)]

    return [bonnie_coins, clyde_coins]


def main():
    # possible configuration
    coins = [1, 1, 2, 4, 8]
    print(bonnie_and_clyde(coins))

    # impossible configuration
    coins = [1, 2, 4, 64]
    print(bonnie_and_clyde(coins))

```

```
37
38 if __name__ == "__main__":
39     main()
```

Problem #8.3

There exists no polynomial algorithm.

Proof. Note to Reader: We shall call this the Bonnie-Clyde Check problem, and the solution set of checks S_{Bonnie} for Bonnie and S_{Clyde} for Clyde.

It is trivial to prove that this Bonnie-Cylde check problem is NP. Given a solution S_{Bonnie} and S_{Clyde}, we must verify

$$\sum_{x \in S_{\text{Bonnie}}} x\text{'s worth} = \sum_{x \in S_{\text{Cylde}}} x\text{'s worth} \qquad |S_{\text{Bonnie}}| + |S_{\text{Clyde}}| = n$$

where $|A|$ is the cardinality of A[1]. This is clearly a polynomial algorithm.

We will now show that the Bonnie-Clyde Check problem reduces down to the Partition Problem.

Assign every checks value as it's representation in the input set S. Feed S into The Partition problem. All output from The Partition problem is also valid output for the Bonnie Clyde problem.

Because we have proved that the Bonnie-Clyde problem is NP and reduces to an NP Complete problem, we conclude that the Bonnie-Clyde problem is NP complete. □

[1] Cardinality meaning the amount of elements in the set

Problem #8.4

The problem is also NP Complete.

Proof. Note to Reader: We shall call this the Bonnie-Clyde 100 Check problem, and the solution set of checks S_{Bonnie} for Bonnie and S_{Clyde} for Clyde.

If the minimum check value is greater than 100$, the proof is identical to the previous.

If the minimum check is less than 100$, first we must prove it is in NP. Given a solution S_{Bonnie} and S_{Clyde}, we must verify

$$\left| \sum_{x \in S_{\text{Bonnie}}} x\text{'s worth} - \sum_{x \in S_{\text{Cylde}}} x\text{'s worth} \right| = 100 \qquad |S_{\text{Bonnie}}| + |S_{\text{Clyde}}| = n$$

where $|A|$ is the cardinality of A[2]. This is clearly a polynomial algorithm.

We will now show that the Bonnie-Clyde 100 Check problem reduces to the partition problem.

If there is a valid solution via Partition problem, we know there to be a solution. If not, we append a 1 to the distribution of checks, and rerun Set-Partition. Does this until either:

- We reach a valid solution.
- We reach 100.

Either way, this problem reduces to the Partition Problem.

Because we have proved that the Bonnie-Clyde problem is NP and reduces to an NP Complete problem, we conclude that the Bonnie-Clyde problem is NP complete. □

[2]Cardinality meaning the amount of elements in the set

Question #9

A greedy algorithm is as follows:

1. Pick two arbitrary vertices u and v.
2. Add both u and v to the vertex cover.
3. Delete all incident edges to u and v.
4. If the adjacency matrix still has edges, go to Step 1.

For the adjacency matrix

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

We get the minimum vertex cover:

$$\{1, 2, 3, 4, 5\}$$

Listing 5: Problem #9

```
#!/usr/local/bin/python3
#
# problem-9.py
# source
#
# Created by Illya Starikov on 11/27/17.
# Copyright 2017. Illya Starikov. All rights reserved.
#

import random


def vertex_cover(adjacency_matrix):
    if sum([item for sublist in adjacency_matrix for item in sublist])
    != 0:
        random_verties = random.sample(range(len(adjacency_matrix[0])),
     2)

        for i in range(len(adjacency_matrix[0])):
            adjacency_matrix[i][random_verties[0]] = 0
            adjacency_matrix[random_verties[0]][i] = 0

            adjacency_matrix[i][random_verties[1]] = 0
            adjacency_matrix[random_verties[1]][i] = 0

```

```
24          return vertex_cover(adjacency_matrix).union(set(random_verties)
        )
25
26      else:
27          return set()
28
29
30  def main():
31      matrix = [[0, 0, 1, 0, 1],
32                [1, 0, 0, 1, 0],
33                [0, 1, 0, 0, 1],
34                [1, 0, 1, 0, 0],
35                [1, 0, 0, 0, 1]]
36
37      print(vertex_cover(matrix))
38
39
40  if __name__ == "__main__":
41      main()
```

CS5400

Artificial Intelligence

Exercise #1

Introduction To Artificial Intelligence

Illya Starikov

Due Date: January 22nd, 2018

We defined a rational agent as follows:

> For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

We also have the following assumptions:

1. The performance measure awards one point for each clean square at each time step, over a "lifetime" of 1000 time steps.
2. The "geography" of the environment is known a priori (Figure 2.2) but the dirt distribution and the initial location of the agent are not. Clean squares stay clean and sucking cleans the current square. The Left and Right actions move the agent left and right except when this would take the agent outside the environment, in which case the agent remains where it is.
3. The only available actions are Left, Right, and Suck.
4. The agent correctly perceives its location and whether that location contains dirt.

Problem Statement. *Show that the simple vacuum-cleaner agent function described in Figure 2.3 is indeed rational under the assumptions listed on Page 38.*

We couple our assumptions with the fact the agent moves to the apposing square if there is no dirt, and "Suck"s if there is dirt. Also, we are under the assumption that the environment cannot become dirtier from some external agent within the 1000 time steps.

Under these assumptions, with our performance measure (see Item 1), we can conclude the agent is rational. Because the performance measure cannot decrease, it would always be beneficial for the agent to move to a new square to clean (since there is always a

possibility of increasing the performance measure). By design, our agent always moves to a new square, consequently, always having the possibility of increasing the performance measure.

Problem Statement. *Describe a rational agent function for the case in which each movement costs one point. Does the corresponding agent program require internal state?*

Yes. Because movement now costs one point, it is no longer in the agent's best interest to constantly move. However, because or agent is purely reflexive and we are unsure of the initial conditions, the agent will receive a negative performance measure; quite contrary to our definition of a rational agent.

For our agent to achieve maximum performance measure, the optimal situation would be to keep all clean squares in some sort of memory, and when the entirety of the vacuum world is clean, constantly perform a *NoOp*; a reflexive agent cannot do this.

Problem Statement. *Discuss possible agent designs for the cases in which clean squares can become dirty and the geography of the environment is unknown. Does it make sense for the agent to learn from its experience in these cases? If so, what should it learn? If not, why not?*

Learning would be optimal in this situation, for the following reasons:

Boundaries An agent with no clue about the world boundaries is doomed to constantly bump in the walls. This would waste precious time cycles on simply finding the boundaries.

Obstacles Just as one might bump in the boundaries, an agent might bump into obstacles.

Common Accumulation Some areas might become dirtier faster than others (i.e., a doorway). If an agent could determine where these areas are, it can check those areas more often that areas with less dirt.

In essence, learning some sort of a map would greatly improve the performance measure of the agent.

After learning some sort of map, we use the following agent design:

1. Start traversing to the most common dirty areas on the map in order of closeness.
2. With every step, if the current step is dirty, clean it; if not, *NoOp*.
3. If all common areas are **not** traversed, proceed to Step 1.
4. Proceed to the closest edge of the world boundary.
5. Proceed around the edges of the map.
6. If **not** reached original position, repeat Step 5.

7. Move one unit closer to the inside of the world. Treat the previous movement as the new world boundaries.
8. If **not** reached center of world, repeat Step 5.
9. Proceed to Step 1.

Exercise #2

Introduction To Artificial Intelligence

Illya Starikov

Due Date: January 24$^{\text{th}}$, 2018

For these problems, we split the river into two sections into two section: Left and Right. Left is assumed to be the initial state.

1 Problem Formulation

Problem Statement. *Precisely formulate the problem.*

1. A method of describing a state would be with the tuple: $\big(C_L,\, M_L,\, B_{L/R}\big)$, where:

$$\begin{aligned} C_L &= \text{The number of Cannibals on the Left} \\ M_L &= \text{The number of Missionaries on the Left} \\ B_{L/R} &= \text{The location (Left/Right) of the boat} \end{aligned}$$

   However, we restrict these numbers to as follows:

   We do not care about the number of Cannibals/Missionaries on the right, because we can calculate them as follows:

$$C_R = 3 - C_L \qquad M_R = 3 - M_L$$

   Although these values are not strictly within the state description, from here forth (for readability purposes) we act as if it is, mentally replacing C_R and M_R with these values. The final desciption is still $\big(C_L,\, M_L,\, B_{L/R}\big)$, and these are merely computed properties of the state.

   Note, however, because we want to restrict our state space to only valid states, we place the following restrictions:

$$M_L \geq C_L \quad \textbf{and} \quad M_R \geq C_R$$

   Furthermore,

$$B_{L/R} = \text{Left} \iff M_L + B_L = 6 \qquad B_{L/R} = \text{Right} \iff M_R + B_R = 6$$

2. The initial state is as follows:

$$\left(C_L,\, M_L,\, B_{L/R}\right) = (3,\, 3,\, L)$$

3. Our ACTIONS are defined as follow. We use synonyms $L \leftrightarrow$ Left and $R \leftrightarrow$ Right.

   Basically, we check to see if we can send one of both Cannibal and Missionary. If we can, add them to the solution set. Next, we check to see if we can send either 1 Cannibal and 1 Missionary. After, we check to see if we can send 2 Cannibal and 2 Missionary. Because there are rules we have to account for (i.e., there must be more Missionary than Cannibal *unless there's 0 Missionary*).

```
1: function ¬(direction)
2:     if direction = Left then
3:         return Right
4:     else
5:         return Left
6:     end if
7: end function
8:
9:
10: function ACTIONS(s)
11:     PossibleActions ← {}                          ▷ A set of sets of moves
12:
13:     for all direction d ∈ {Left, Right} do
14:         if a.M_d > 0 and a.C_d > 0 then
15:             NewAction ← {send Cannibal d → ¬d}
16:             NewAction ← {send Missionary d → ¬d}      ▷ Note this has the form {..., ...}
17:
18:             PossibleActions ← PossibleActions ∪ { NewAction} ▷ Note this has the form {..., {..., ...}, ...}
19:         end if
20:
21:         for i ← 1 to 2 do
22:             if a.M_d − i ≥ 0 then      ▷ Is there even enough missionaries to remove?
23:                 if (a.M_d − i ≥ a.C_d or a.M_d − i = 0) and a.M_¬d + i ≥ a.C_¬d then
24:                     NewAction ← {send i Missionary d → ¬d }[1]
25:                     PossibleActions ← PossibleActions ∪ { NewAction}
26:                 end if
27:             end if
```

[1] Note by saying "send i Missionary", we are just saying "send a missionary" i times. For $i = 2$, we have NewAction ← {send a missionary, send a missionary}.

```
28:
29:             if a.C_d − i ≥ 0 then                ▷ Is there even enough cannibals to remove?
30:                 if a.C_¬d + i ≤ a.M_¬d or a.M_¬d = 0 then
31:                     NewAction ← {send i Cannibal d → ¬d}[2]
32:                     PossibleActions ← PossibleActions ∪ { NewAction}
33:                 end if
34:             end if
35:         end for
36:     end for
37:
38:     return PossibleActions
39: end function
```

4. We define Results(s, a) as follows.

```
1: function Results(s, a)
2:     a′ ← a
3:     a′.B_L/R = ¬a′.B_L/R
4:
5:     for all action ∈ a do                ▷ We defined an action as a set of steps
6:         if action = send Cannibal Left → Right then
7:             a′.C_L = a′.C_L − 1
8:         end if
9:
10:        if action = send Cannibal Right → Left then
11:            a′.C_L = a′.C_L + 1
12:        end if
13:
14:        if action = send Missionary Left → Right then
15:            a′.M_L = a′.M_L − 1
16:        end if
17:
18:        if action = send Missionary Right → Left then
19:            a′.M_L = a′.M_L + 1
20:        end if
21:    end for
22:
23:    return a′
24: end function
```

5. We define IsGoal(s) as follows.

```
1: function IsGoal(s)
```

[2]See previous footnote.

2: **return** $s = (0, 0, \text{Right})$
3: **end function**

6. We define $\text{C}(s, a, s')$

1: **function** $\text{C}(s, a, s')$
2: **return** $|\text{a}|$[3]
3: **end function**

2 State Space

Figure 1 provides the complete space for the problem, with M representing Missionaries and C representing cannibals.

3 Possible Solution

Figure 2 outlines a path for a solution, with M representing Missionaries and C representing cannibals.

[3]We define the $|x|$ operator to return the length of x.

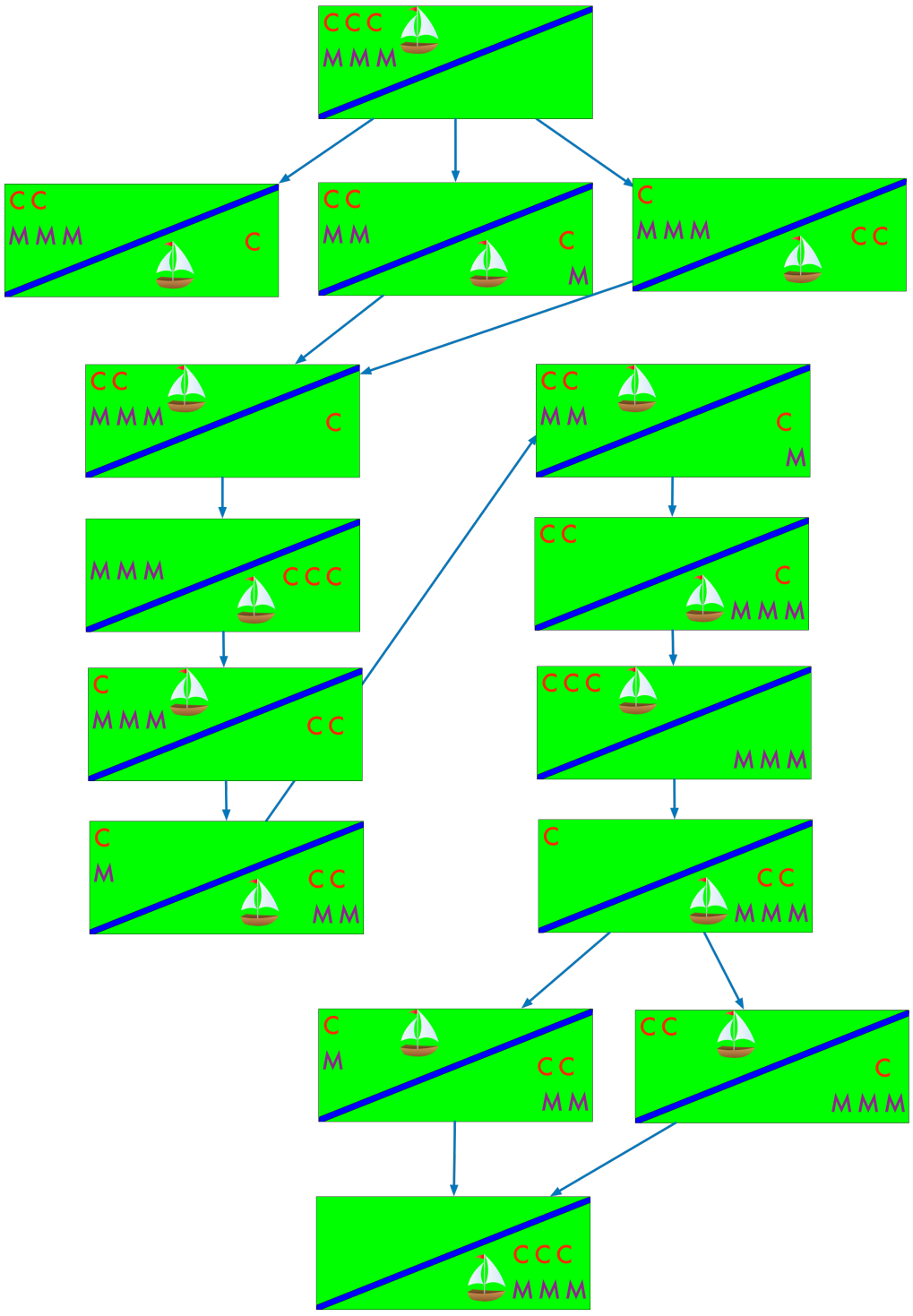


Figure 1: The Complete State Space

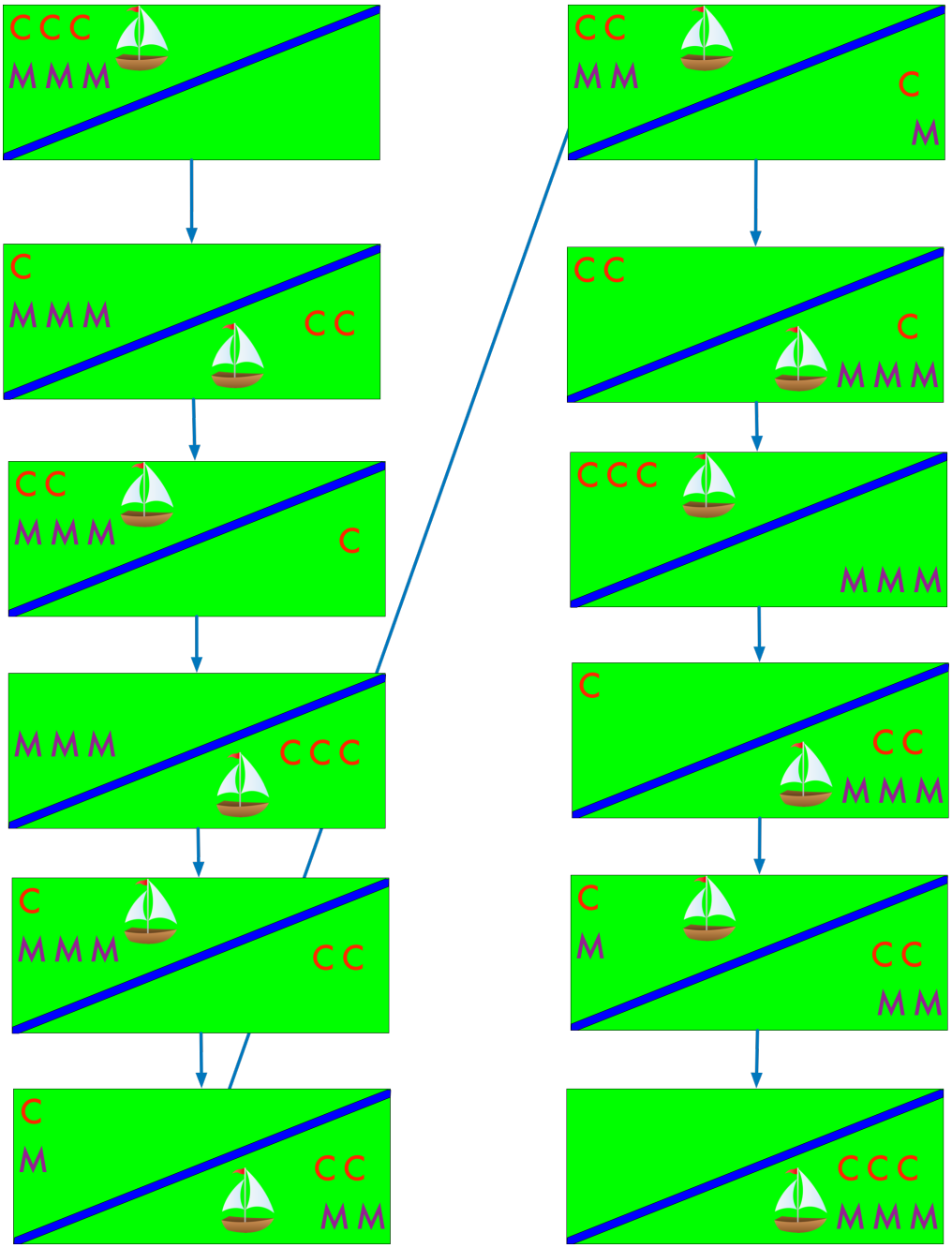


Figure 2: Path To A Solution

Exercise #3

Introduction To Artificial Intelligence

Illya Starikov

Due Date: February 5^{th}, 2018

1 Negative Path Optimality

Problem Statement. *Suppose that actions can have arbitrarily large negative costs; explain why this possibility would force any optimal algorithm to explore the entire state space.*

Even if for any path we have an arbitrary path $P_{\max}$ with largest path cost $N_{\max}$,

$$N_{\max} = \sum_{\text{vertex } v \in P_{\max}} \textsc{path-cost}\,(v)$$

There might be a singular path P after it,

$$N = \sum_{\text{vertex } v \in P} \textsc{path-cost}\,(v)$$

Such that

$$N_{\max} + N = \text{Minimum Path Cost}$$

Therefore, any algorithm that is to find the optimal path cost must search the entire state space.

Also, there is the possibility of a negative cycle in a graph, which can only be discovered by exploring the entire state space.

2 Bounds on Negative Path Cost

Problem Statement. *Does it help if we insist that step costs must be greater than or equal to some negative constant c? Consider both trees and graphs.*

For trees, it can be helpful in limiting the search space, because there can only be so much gained from negative paths.

For graphs, there is the aforementioned issue of negative cycles, where the lowest path cost could be $-\infty$ (i.e., keep repeating the negative cycle infinitely).

3 Loops In Negative Path Cost

Problem Statement. *Suppose that a set of actions forms a loop in the state space such that executing the set in some order results in no net change to the state. If all of these actions have negative cost, what does this imply about the optimal behavior for an agent in such an environment?*

This implies that the optimal behavior in this environment is to loop around the states in the negative cycle.

Exercise #4

Introduction To Artificial Intelligence

Illya Starikov

Due Date: February 12th, 2018

Problem Statement. *On page 90, we mentioned iterative lengthening search, an iterative analog of uniform cost search. The idea is to use increasing limits on path cost. If a node is generated whose path cost exceeds the current limit, it is immediately discarded. For each new iteration, the limit is set to the lowest path cost of any node discarded in the previous iteration.*

1 Iterative Lengthening Optimality

Problem Statement. *Show that this algorithm is optimal for general path costs.*

Because the nodes are discovered in order of path cost, naturally the node with the lowest path cost will be discovered first.

2 Iterative Lengthening Performance

Problem Statement. *Consider a uniform tree with branching factor b, solution depth d, and unit step costs. How many iterations will iterative lengthening require?*

The runtime will be $\mathcal{O}\left(b^d\right)$.

3 Iterative Lengthening Performance II

Problem Statement. *Now consider step costs drawn from the continuous range $[\epsilon, 1]$, where $0 < \epsilon < 1$. How many iterations are required in the worst case?*

The runtime will be $\mathcal{O}\left(\frac{d}{\epsilon}\right)$.

Exercise #5

Introduction To Artificial Intelligence

Illya Starikov

Due Date: February 19th, 2018

1 A^* vs. DFS

Problem Statement. *True/False: Depth-first search always expands at least as many nodes as A search with an admissible heuristic.*

False. DFS has the best case of $\Theta(d)$ (for the case where there is no backtracking to find a solution). A^* can make no such guarantee, because it ranges by heuristic.

2 $h(n) = 0$ For 8-Puzzle

Problem Statement. *True/False:* $h(n) = 0$ *is an admissible heuristic for the 8-puzzle.*

True. For positive path costs, $h(n) = 0$ will always be an admissible heuristic.

$$\forall \epsilon \in \mathbb{R}^+, h(n) = 0 \leq C^*(n) = \epsilon$$

3 A^* In Robotics

Problem Statement. *True/False: A is of no use in robotics because percepts, states, and actions are continuous.*

True. A^* operates on discrete states. However, there is always the possibility to create discrete states from continuous states.

4 Breadth-First Search Completeness

Problem Statement. *True/False: Breadth-first search is complete even if zero step costs are allowed.*

True. Per our definition of completeness:

> A search algorithm is complete iff it will find a goal when one exists.

5 Rook Admissibility

Problem Statement. *True/False: Assume that a rook can move on a chessboard any number of squares in a straight line, vertically or horizontally, but cannot jump over other pieces. Manhattan distance is an admissible heuristic for the problem of moving the rook from square A to square B in the smallest number of moves.*

False. Using the chess depicted in Figure 1, the Manhattan Distance from $A1$ to $A8$ would be 8; however, a rook could get there in one move.

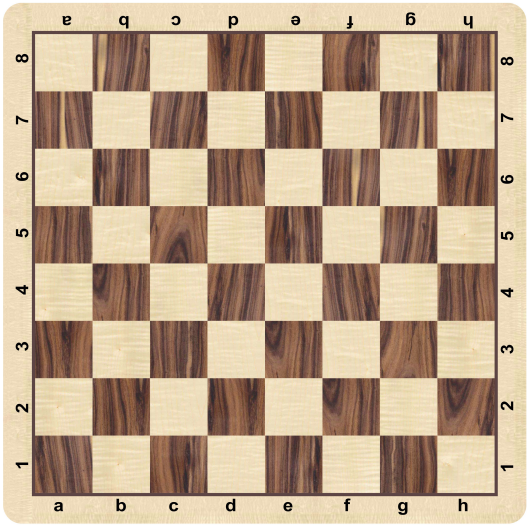


Figure 1: A Standard Chessboard

Exercise #6

Introduction To Artificial Intelligence

Illya Starikov

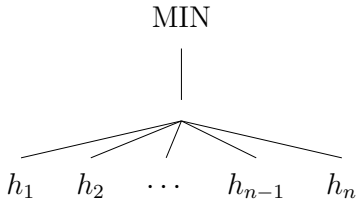
Due Date: March 13th, 2018

1 Suboptimal Min vs Optimal Min

Problem Statement. *Prove the following assertion: For every game tree, the utility obtained by MAX using minimax decisions against a suboptimal MIN will be never be lower than the utility obtained playing against an optimal MIN. Can you come up with a game tree in which MAX can do still better using a suboptimal strategy against a suboptimal MIN?*

We will do so by induction.

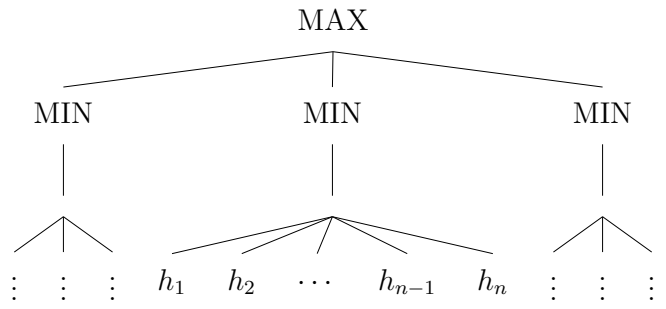
Basis Case The base case is a *MIN* node with all terminals. Suppose we have the following leaf such that for any node h_x, they are ordered in ascending order based on the utility value $h_1 \leq h_2 \leq \cdots \leq h_{n-1} \leq h_n$:



An optimal *MIN* would always pick h_1, while a suboptimal would pick from $\{h_1, h_2, \ldots, h_{n-1}, h_n\}$. From this, we can conclude

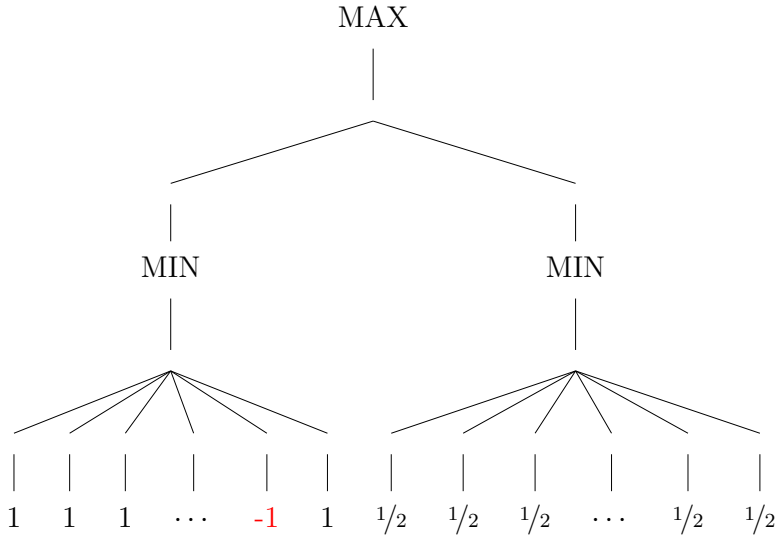
$$\text{Optimal } \textit{MIN}\text{'s Utility} \leq \text{Suboptimal } \textit{MIN} \text{ Utility} \tag{1}$$

Inductive Case The recursive case is for *MIN* with parents and children.



However, because the game tree can be reduced to a *MIN* node with all terminals (such as the base case), our inductive step holds.

For a game tree of a suboptimal *MAX* vs a *MIN*, we can produce the following chess game:



An optimal *MAX* would pick the rightmost *MIN*, which would result in only a draw; however, a suboptimal *MIN* would only pick the leftmost *MIN*, resulting in in almost all wins, with only one possibility of losing.

Exercise #7

Introduction To Artificial Intelligence

Illya Starikov

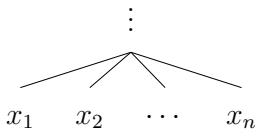
Due Date: March 9th, 2018

1 x To $ax + b$ Leaf Node Transformation

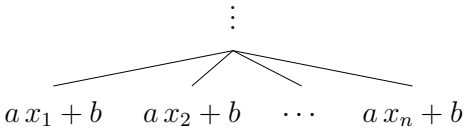
Problem Statement. *Prove that with a positive linear transformation of leaf values (i.e., transforming a value x to $a\,x + b$ where $a > 0$), the choice of move remains unchanged in a game tree, even when there are chance nodes.*

Proof. We will prove the following via proof of mathematical induction.

Base Case We will take the base case with a lead nodes, so the transformation would be as follows:



would transform into:



Recalling the definition of Minimax, we get the following:

$$\textsc{Minimax}(\textsc{s}) = \begin{cases} \textsc{Max'sUtility}(\textsc{s}) & \leftrightarrow \textsc{Terminal-Test}(\textsc{s}) \\ \max_{a \in \textsc{Actions}(\textsc{s})} \textsc{Minimax}(\textsc{Result}(\textsc{s},\, \textsc{a})) & \leftrightarrow \textsc{Player}(\textsc{s}) = \textsc{Max} \\ \min_{a \in \textsc{Actions}(\textsc{s})} \textsc{Minimax}(\textsc{Result}(\textsc{s},\, \textsc{a})) & \leftrightarrow \textsc{Player}(\textsc{s}) = \textsc{Min} \end{cases} \tag{1}$$

Recalling that Actions returns a set of values (let us name them $x_1, x_2, \ldots, x_n$) our $\min()$ and $\max()$ values can be rewritten as:

$$\min(x_1, x_2, \ldots, x_n) = x_p$$
$$\max(x_1, x_2, \ldots, x_n) = x_p$$

where x_p is the chosen node. For the probability, we get

$$p(x_1) + p(x_2) + \cdots + p(x_n)$$

Applying the transformation, we get the following:

$$\begin{aligned}
\min(a\,x_1 + b, a\,x_2 + b, \ldots, a\,x_n + b) &= a\,x_p + b \\
\max(a\,x_1 + b, a\,x_2 + b, \ldots, a\,x_n + b) &= a\,x_p + b \\
p_1(a\,x_1 + b) + p_2(a\,x_2 + b) + \cdots + p_n(a\,x_n + b)
\end{aligned}$$

From this, we factor out a and b:

$$\begin{aligned}
a\,\min(x_1, x_2, \ldots, x_n) + b &= a\,x_p + b \\
a\,\max(x_1, x_2, \ldots, x_n) + b &= a\,x_p + b \\
a(p_1\,x_1 + p_2\,x_2 + \cdots + p_n\,x_n) + b
\end{aligned}$$

Upon which we can clearly see a ($\forall a \in \mathbb{R}^+$) and b have no choice on which node gets chosen.

Inductive Step As we scale our problem up for MINIMAX (Equation 1), we see that if the decision of the terminal node is unchanged, the decision at the root will be changed. The same terminal value will be returned; and as we go up the game tree, each ply acts as a terminal.

□

Exercise #8

Introduction To Artificial Intelligence

Illya Starikov

Due Date: April 22nd, 2018

1 State-Space Search Algorithms

Problem Statement. *Give the name of the algorithm that results from each of the following special cases.*

1. Local beam search with $k = 1$. **Steepest-Ascent Hill-Climbing.**
2. Local beam search with one initial state and no limit on the number of states retained. **Breadth First Tree Search.**
3. Simulated annealing with $T = 0$ at all times (and omitting the termination test). **Steepest-Ascent Hill-Climbing.**
4. Simulated annealing with $T = \infty$ at all times. **Random Walk.**
5. Genetic algorithm with population size $N = 1$. **Random Walk.**

Bonus #1

Illya Starikov

Due Date: Sunday, February 18$^{\text{th}}$, 2018

We know the definition of branching factor b^* to be defined as

$$N = b^* + (b^*)^2 + \cdots + (b^*)^d$$

With N being the total number of nodes being generated, and d being the solution depth. From this, we can get an estimate for the effective branching factor as follows:

$$b^* = N^{\frac{1}{d}}$$

Note this is not a perfect equation, and has the possibility for high error tolerances; for our purposes, it should be okay. The two heuristics for this assignment are as follows:

$$\begin{aligned} h_1 &= \text{Score Difference} \\ &= \frac{\text{Quota} - \text{Current Score}}{\text{Max Swaps} - \text{Current Swaps}} \\ h_2 &= \text{Homogeneous Board} \\ &= \text{Standard Deviation of the} \\ &= \text{Number of Devices Types on the Board} \end{aligned}$$

Results can be found in Table 1. Essentially, the branching factors are similar, but the Homogeneous Board heuristic has a tendency to explore more of the board.

Table 1: The computations for the effective branching factor.

Input	**Heuristic**	**Nodes Generated** (N)	**Solution Depth** (d)	**Branching Factor** (b^*)
Puzzle #1	Score Difference	4	1	4
	Homogeneous Board	4	1	4
Puzzle #2	Score Difference	99	11	1.52
	Homogeneous Board	144	10	1.64
Puzzle #3	Score Difference	94	12	1.46
	Homogeneous Board	890	25	1.31

CS5402

Data Mining

Homework #1

CS5402 — Intro To Data Mining

Illya Starikov

Due Date: July 6th, 2018

1 1-Rule Method

Problem Statement. *Use the 1-rule (1R) method to find the best single attribute to determine restaurant. In order to demonstrate that you actually know how this method works (and aren't just guessing at which attribute is best), you must fill in ALL of the blank values in the table below; otherwise, you will not receive any credit for this problem.*

Attribute	Attribute Value	# Rows	Most Frequent Value	Errors	Total Errors
Meal Preference	Hamburger	3	McDonalds (3)	0	2
	Fish	2	Burger King (2)	0	
	Chicken	4	Wendy's (2)	2	
Gender	M	5	Burger King/McDonald's (2/2)	3	5
	F	4	McDonald's (2)	2	
Drink Preference	Pepsi	3	Burger King (2)	1	3
	Coke	6	McDonald's (4)	2	

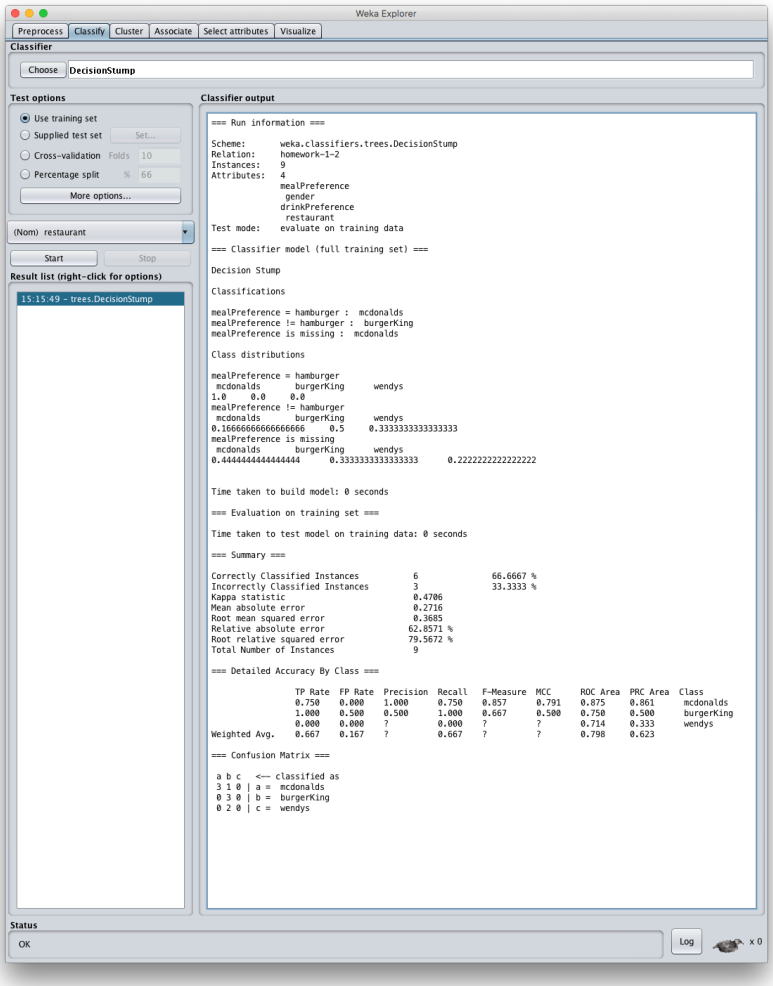
From this, the following rules would be generated:

$$\begin{aligned} \text{Meal Preference} = \textit{Hamburger} &\implies \text{McDonald's} \\ \text{Meal Preference} = \textit{Fish} &\implies \text{Burger King} \\ \text{Meal Preference} = \textit{Chicken} &\implies \text{Wendy's} \end{aligned}$$

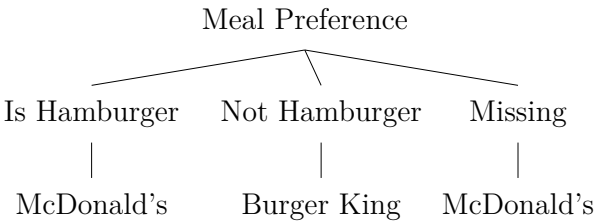
2 Decision Stump

Problem Statement. *Create the dataset given in problem 1. as an arff or csv file, and run DecisionStump on it in Weka. List the classification rules that are produced by Weka (you can just include a screenshot of your Weka output) AND hand-draw the decision tree that corresponds to those rules.*

From Weka's Decision Stump, we get the following output:



For the figure, we get the following decision tree:



3 Statistical Modeling

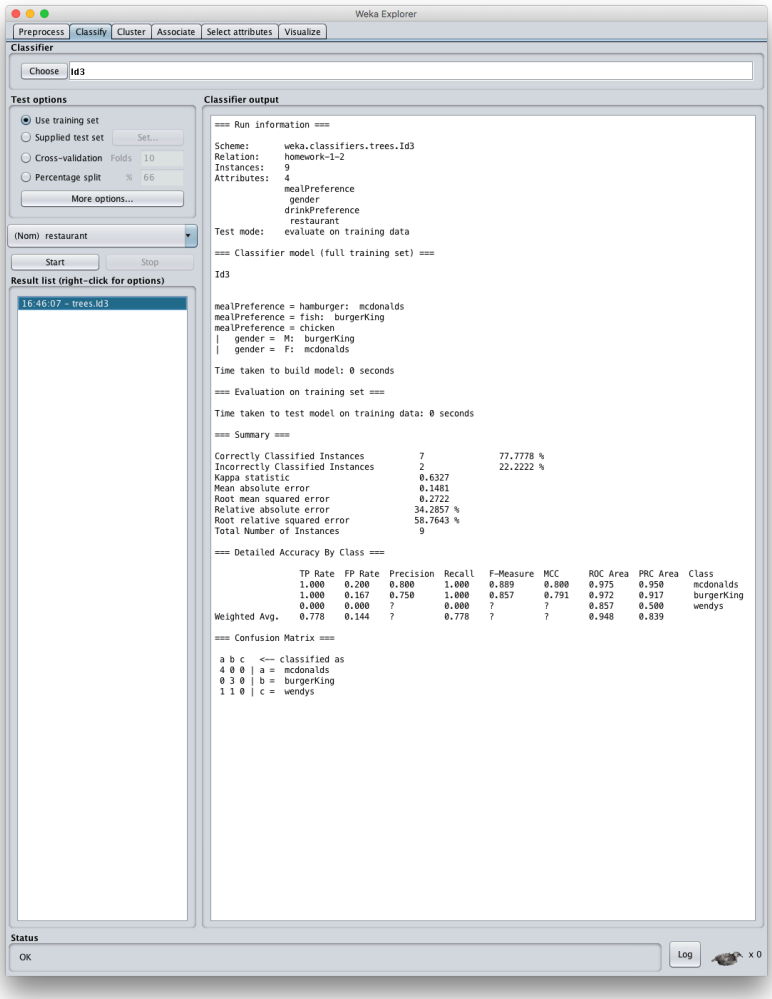
Problem Statement. *Statistical modeling can be used to compute the probability of occurrence of an attribute value. Based on the data given in the table below, if we have a new instance where ageGroup = youngAdult, gender = M, and bookPreference = nonFiction, what is the likelihood that musicPreference = country? Just set up the equation to compute this with the appropriate values; you don't have to actually calculate the final answer.*

$$\text{Likelihood of Country} = {}^{2}/_{3} \times {}^{1}/_{3} \times {}^{1}/_{3} \times {}^{3}/_{8} = {}^{1}/_{36} \approx 0.02\bar{7}$$

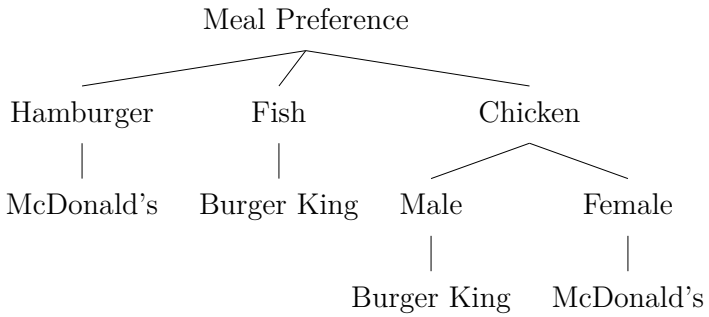
4 Id3

Problem Statement. *Create the dataset given in problem 1. as an arff or csv file, and run Id3 on it in Weka. Include a screenshot of the decision tree output that is produced by Weka AND hand-draw the decision tree. Note: You may have to install the package containing Id3.*

From Weka's Id3, we get the following output:



For the figure, we get the following decision tree:



5 Entropy

Problem Statement. *If we want to make a decision tree for determining restaurant, we must decide which of the three non-decision attributes (mealPreference, gender, or drinkPreference) to use as the root of the tree.*

- *Set up the equation to compute what in lecture we called entropyBeforeSplit for restaurant. You do not have to actually solve (i.e., calculate the terms in) the equation, just set up the equation with the appropriate values.*

$$\text{Entropy Before Split} = -{}^{3}/_{10}\log_2({}^{3}/_{10}) - {}^{5}/_{10}\log_2({}^{5}/_{10}) - {}^{2}/_{10}\log_2({}^{2}/_{10}) \approx 1.485$$

- *Set up the equation to compute entropy for mealPreference when its value is chicken. That is, a tree with mealPreference at the root would have three branches (one for hamburger, one for chicken, and one for fish), requiring us to compute entropyHamburger, entropyChicken, and entropyFish; here we only want you to set up the equation to compute entropyChicken. You do not have to actually solve (i.e., calculate the terms in) the equation, just set it up using the appropriate values.*

$$\text{Entropy Chicken} = -{}^{1}/_{4}\log_2({}^{1}/_{4}) - {}^{1}/_{4}\log_2({}^{1}/_{4}) - {}^{2}/_{4}\log_2({}^{2}/_{4}) \approx 1.040$$

- *Suppose that instead of considering mealPreference to be the root of this decision tree, we had instead considered drinkPreference. Set up the equation to compute information gain for drinkPreference given the variables specified below.*

$$\text{Information Gain} = X - ({}^{7}/_{10}\,C + {}^{3}/_{10}\,P)$$

6 isSticky

Problem Statement. *Create the dataset shown below as an arff or csv file and run the Prism algorithm on it in Weka specifying isSticky as the decision attribute. List the classification rules that are produced (you can just include a screenshot of your Weka output). Then work out the Prism algorithm by hand to show what classification rules you would get; who knows, they might be different!*

6.1 Is Sticky = No

consistency	packaging	chocolate	roundShaped	isSticky
soft	individuallyWrapped	yes	no	no
soft	box	yes	no	no
hard	sack	no	yes	no
hard	box	yes	yes	no
hard	individuallyWrapped	no	no	yes
hard	box	no	yes	yes
soft	individuallyWrapped	no	no	yes

Homework #2

CS5402 — Intro To Data Mining

Illya Starikov

Due Date: July 10th, 2018

1 Association Rules

Problem Statement. *Compute the coverage (i.e., support) of each item set listed below.*

drinkPref = pepsi 2

married = no, pet = dog 3

sportPref = football, musicPref = classical, drinkPref = tea 0

Problem Statement. *Write down every association rule that could be generated from the item set listed below, regardless of whether or not there are actually any instances of that rule in our given dataset.*

$$
\begin{aligned}
&\textbf{married} = \text{no} &&\implies \textbf{pet} = \text{dog} \\
&\textbf{pet} = \text{dog} &&\implies \textbf{married} = \text{no} \\
&\_ &&\implies \textbf{married} = \text{no and } \textbf{pet} = \text{dog}
\end{aligned}
$$

Problem Statement. *Compute the accuracy (i.e., confidence) of each rule listed below. Express accuracy as a fraction (e.g., 2/3, 2/2, etc.), NOT as a decimal number (e.g., 0.67, 1.0, etc.).*

With the case *If pet = dog then income = middle*, we get as follows:

$$\text{Accuracy} = \frac{1}{4}$$

With the case *If married = no and sportPref = football, then pet = dog and musicPref = rock*, we get as follows:

$$\text{Accuracy} = \frac{2}{3}$$

With the case *If _, then drinkPref = coke and married = yes*, we get as follows:

$$\text{Accuracy} = 0$$

Problem Statement. *In each rule listed below, specify whether any condition(s) in the antecedent (i.e., the "if' part) can be dropped.*

If married = no and sportPref = football then pet = dog and drinkPref = coke No.

If married = yes and pet = cat then musicPref = country Yes, **married** = yes can be dropped.

2 RICO

Problem Statement. *Using the Rule Induction from Coverings algorithm discussed in class, find each of the partitions specified below.*

$$\begin{aligned}
\{a\}^* &= \{\{x_1, x_5\}, \{x_2, x_3, x_4\}\} \\
\{c\}^* &= \{\{x_1\}, \{x_2, x_3\}, \{x_4\}, \{x_5\}\} \\
\{d\}^* &= \{\{x_1, x_2\}, \{x_3, x_4\}, \{x_5\}\} \\
\{d,\, e\}^* &= \{\{x_1\}, \{x_2\}, \{x_3, x_4\}, \{x_5\}\}
\end{aligned}$$

Problem Statement. *Suppose that we are looking for a covering for decision attribute f which has partition {f}* = {{x1, x2}, {x3, x4, x5}}.*

- No, the block $\{x_1, x_5\}$ is not a subset of any blocks in $\{f\}^*$.
- No, the block $\{x_2, x_3\}$ is not a subset of any blocks in $\{f\}^*$.
- Yes, all blocks are subsets of all blocks in $\{f\}^*$.
- No, because $\{d,\, e\}^*$ is not minimal.

Problem Statement. *Find a covering for f. If you found one in part b that worked, just use one of those. Using your covering, give a set of rules for decision attribute f.*

$$\begin{aligned}
d = 0 &\implies f = True \\
d = 1 &\implies f = False \\
d = 2 &\implies f = False
\end{aligned}$$

3 KD-Tree

Problem Statement. *Consider the dataset given below where the decision attribute is the one labeled decision. Build a kd-tree where k = 3. No partial credit will be given unless you show your work!*

Sorting the tuples by x, we get as follows:

$$[(1, 7, 9), (2, 5, 8), (3, 6, 7), (4, 4, 0), (5, 3, 4), (6, 1, 6), (7, 2, 2)]$$

Seeing a our median = 4, we partition as follows:

$$[(1, 7, 9), (2, 5, 8), (3, 6, 7)] \quad [(4, 4, 0), (5, 3, 4), (6, 1, 6), (7, 2, 2)]$$

Sorting these tuples by y, we get as follows:

$$[(2, 5, 8), (3, 6, 7), (1, 7, 9)] \quad [(6, 1, 6), (7, 2, 2), (5, 3, 4), (4, 4, 0)]$$

With our medians as follows, we get as follows:

median = 6		median = 2.5	
[(2, 5, 8)]	[(3, 6, 7), (1, 7, 9)]	[(6, 1, 6), (7, 2, 2)]	[(5, 3, 4), (4, 4, 0)]

Sorting these tuples by z, we get as follows:

$$[(2, 5, 8)] \quad [(3, 6, 7), (1, 7, 9)] \quad [(7, 2, 2), (6, 1, 6)] \quad [(4, 4, 0), (5, 3, 4)]$$

With our medians as follows, we get as follows:

median = 8	median = 8		median = 4		median = 2	
[(2, 5, 8)]	[(3, 6, 7)]	[(1, 7, 9)]	[(7, 2, 2)]	[(6, 1, 6)]	[(4, 4, 0)]	[(5, 3, 4)]

To produce the following KD-Tree:

Homework #3

CS5402 — Intro To Data Mining

Illya Starikov

Due Date: July 17th, 2018

1 C4.5

Split	Entropy Less Than	Entropy Greater Than	Entropy Before Split	Entropy After Split	Information Gain
8	0	1.55459	1.579	1.435	0.14400
10	1	1.57262	1.579	1.48453	0.09447
15	0.9183	1.52193	1.579	1.38263	0.19637
20	0.81128	1.39215	1.579	1.21342	0.36558
24	0.97095	1.40564	1.579	1.23845	0.34055
28	0.9183	0.98523	1.579	0.95434	0.62466
28	1.37878	1	1.579	1.20396	0.37504
28	1.5	0.97095	1.579	1.29652	0.28248
29	1.53049	1	1.579	1.36726	0.21174
30	1.52193	0.9183	1.579	1.38263	0.19637
30	1.49492	0	1.579	1.26493	0.31407
31	1.55459	0	1.579	1.435	0.14400
33	1.57662	0	1.579	1.57662	0.00238

2 Grouping

Grouping	Entropy	Entropy Before Split	Entropy After Split	Information Gain
Coke-Pepsi	1.5219	1.565	1.2652	0.2998
Mountain Dew	1.5850	1.565		
Coke-Mountain Dew	1.4591	1.565	1.0943	0.4706
Pepsi	0	1.565		
Pepsi-Mountain Dew	1.3710	1.565	0.9623	0.6028
Coke	0.9183	1.565		

No, they should not be grouped; the respective information gains for the groupings are less than the information gain without grouping.

3 J48

For the deepest subtree, `credit_history`, we get the following error for the individual leaves:

$$U_{90\%}(0,\,1) + U_{90\%}(0,\,1) + 7 \times U_{90\%}(1,\,7) + 0 + 2 \times U_{90\%}(0,\,2) \approx$$
$$0.95 + 0.95 + 7 \times 0.5207 + 0 + 2 \times 0.77639 \approx 3.19709$$

If the subtree would just be replaced by a leaf **good (9/2)**, predicted error would be

$$9 \times U_{90\%}(2,\,9) \approx 9 \times 0.47009 \approx 4.23081$$

Therefore, the new tree would be

```
property_magnitude = car
|    personal_status = male div/sep: bad (1.0)
|    personal_status = female div/dep/mar: good (7.0)
|    personal_status = male single: good (9.0/2.0)
|    personal_status = male mar/wid: good (1.0)
|    personal_status = female single: good (0.0)
```

For this subtree, we get the following error for individual leaves:

$$U_{90\%}(0,\,1) + 7 \times U_{90\%}(0,\,7) + 9 \times U_{90\%}(2,\,9) + U_{90\%}(0,\,1) + 0 \approx$$
$$0.95 + 7 \times 0.348169 \times 0.470090 \approx 7.61793$$

Again, if the subtree would be replaced by a leaf **good (17/2)**, predicted error would be

$$17 \times U_{90\%}(2,\,17) \approx 17 \times 0.70420 \approx 5.0286$$

Therefore, the new tree would be

```
property_magnitude = car: good (17/2)
```

4 Missing Value

4.1 Outdoor

4.1.1 Hair = Short

$$\text{entropy} = -\frac{1 + 2/7}{2 + 2/7} \log_2 \frac{1 + 2/7}{2 + 2/7} - \frac{1}{2 + 2/7} \log_2 \frac{1}{2 + 2/7} = 0.9887$$

4.1.2 Hair = Long

$$\text{entropy} = \frac{0}{\vdots} \cdots = 0$$

4.2 Indoor

4.2.1 Hair = Short

$$\text{entropy} = -\frac{1}{1+{}^{2}\!/_{7}}\log_2\frac{1}{1+{}^{2}\!/_{7}} - \frac{{}^{2}\!/_{7}}{1+{}^{2}\!/_{7}}\log_2\frac{{}^{2}\!/_{7}}{2+{}^{2}\!/_{7}} = 0.7642$$

4.2.2 Hair = Long

$$\text{entropy} = -\frac{1}{1}\log_2\frac{1}{1} = 0$$

4.3 Both

4.3.1 Hair = Short

$$\text{entropy} = -\frac{1+{}^{3}\!/_{7}}{1+{}^{3}\!/_{7}}\log_2\frac{1+{}^{3}\!/_{7}}{2+{}^{3}\!/_{7}} = 0 = 0$$

4.3.2 Hair = Long

$$\text{entropy} = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1$$

Homework #5

CS5402 — Intro To Data Mining

Illya Starikov

Due Date: July 24th, 2018

1 DBSCAN

a) The table would look as follows:

Point	x	y	Density	Designation
A1	1	4	3	Core Point
A2	5	2	3	Core Point
A3	4	3	3	Core Point
A4	5	6	2	Border Point
A5	2	5	2	Border Point
A6	5	4	4	Core Point
A7	1	2	2	Border Point
A8	3	1	1	Noise

b) The points in the two clusters would be:

- $\{A1, A5, A7\}$
- $\{A2, A3, A4, A6\}$

The clusters are formed by first identifying the main points; this is done via calculating the ϵ, or the density of the points (i.e., how many points are around those points). After the core points have been identified, the border points are found via observing which points are close (within an ϵ value) to the core points. Finally, the remaining points are simply noise.

2 Bagging Vs Boosting

2.1 Iris

Looking at the confusion matrix of using bagging:

```
  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
  0  0 50 |  b = Iris-versicolor
  0  0 50 |  c = Iris-virginica
```

And comparing to the confusion matrix of boosting:

```
  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
  0 45  5 |  b = Iris-versicolor
  0  1 49 |  c = Iris-virginica
```

We see that boosting is far more accurate; the kappa statistic agrees, where bagging had a kappa statistic of 0.5 versus the boosting kappa statistic of 0.94.

2.2 Iris

Looking at the confusion matrix of bagging:

```
 445  13 |   a = benign
   7 234 |   b = malignant
```

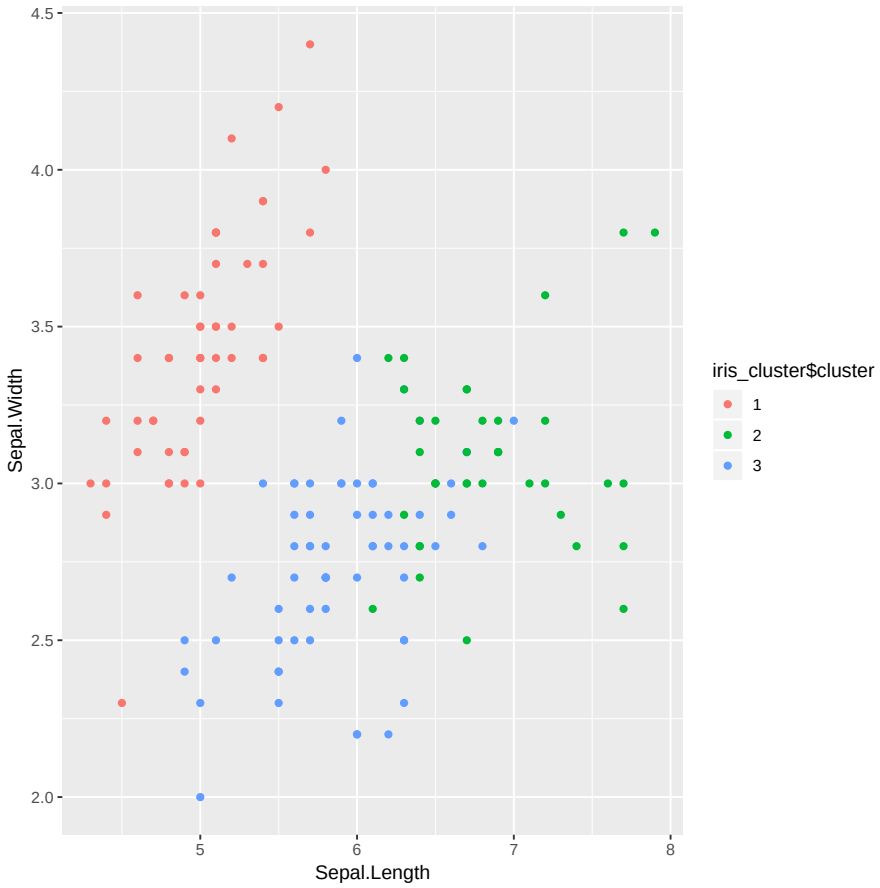
And comparing to the confusion matrix of the boosting:

```
   a   b   <-- classified as
 445  13 |   a = benign
  17 224 |   b = malignant
```

We see there is very little difference in accuracy. The kappa statistic agrees; where bagging had a kappa statistic of 0.937 versus the boosting kappa statistic of 0.9046.

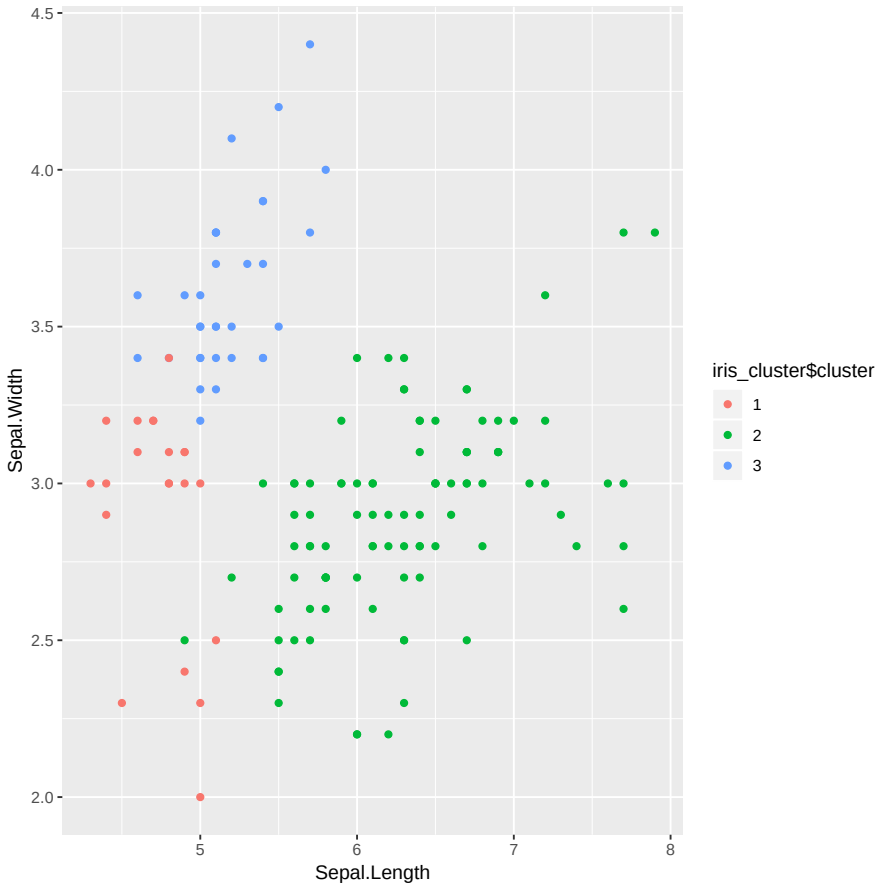
3 KMeans In R

```r
library(ggplot2)

iris_copy <- data.frame(iris[1:4])
iris_cluster = kmeans(iris_copy, 3)

table(iris_cluster$cluster, iris$Species)

# could not get regular plot to work so this is what i got to work
# requires packages ggplot2 + labeling (and dependencies)
iris_cluster$cluster <- as.factor(iris_cluster$cluster)
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = iris_cluster$cluster))
    + geom_point()
```



4 DBSCAN In R

```
library(fpc)
iris_cluster = dbscan(iris_cluster, eps=0.42, MinPts = 5)

table(iris_cluster$cluster, iris$Species)

# could not get regular plot to work so this is what i got to work
# requires packages ggplot2 + labeling (and dependencies)
iris_cluster$cluster <- as.factor(iris_cluster$cluster)
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = iris_cluster$cluster))
    + geom_point()
```



5 Frames For Days

```
refund = c(
    "y",
    "n",
    "n",
    "y",
    "n",
    "n",
    "y",
    "n",
    "n",
    "n"
)
status = c(
    "single",
    "married",
    "single",
    "married",
    "divorced",
```

```
    "married",
    "divorced",
    "single",
    "married",
    "single"
)

income_k = c(
    125,
    100,
    70,
    120,
    95,
    60,
    220,
    85,
    75,
    90
)
class = c(
    FALSE,
    FALSE,
    FALSE,
    FALSE,
    TRUE,
    FALSE,
    FALSE,
    TRUE,
    FALSE,
    TRUE
)

tax_info = data.frame(refund, status, income_k, class)
```

6 She's So Mean

```
tax_info <- data.frame(refund, status, income_k, class)
total <- 0

for (i in 1:nrow(tax_info)) {
    total <- total + tax_info$income_k[i]
}

average <- total / nrow(tax_info)
print(average)
```

7 Animals

```
library(party)

data <- read.csv("C:\\temp\\animal data.csv")
data_frame = data.frame(data)

tree = ctree(Name ~ BloodType + GivesBirth + CanFly + LivesInWater, data=
    data_frame)
plot(tree, type="simple")
```

Homework #6

CS5402 — Intro To Data Mining

Illya Starikov

Due Date: July 27th, 2018

1 Support Vectors

```
library(kernlab)

xy = matrix(c(1, 2, 4, 1, 1, 5, 2, 1, 2, 0, 1, 2),  nrow=6,  ncol=2)
z =  matrix(c(1, 1, -1, 1, 1, -1),  nrow=6,  ncol=1)

svp = ksvm(xy, z, type="C-svc", kernel='vanilladot', C=100, scaled=c())

print(xmatrix(svp))
print(predict(svp, matrix(c(0, 1), nrow=1, ncol=2)))
print(predict(svp, matrix(c(4, 1), nrow=1, ncol=2)))
```

The output is as follows:

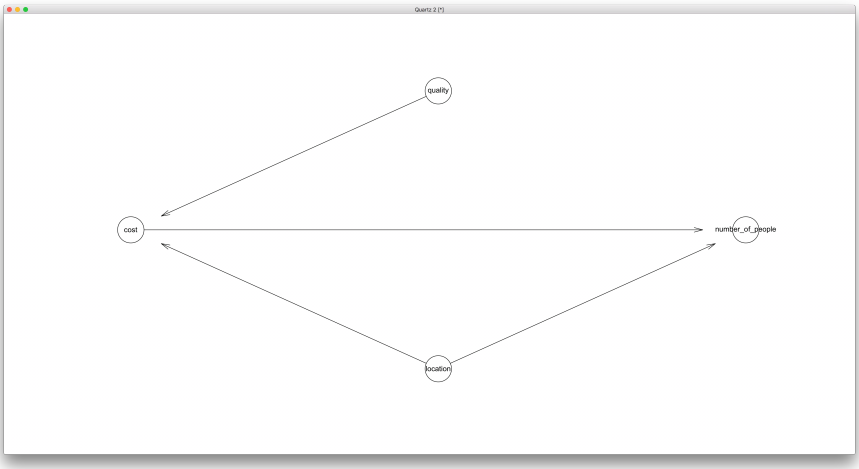
```
 Setting default kernel parameters
[[1]]
  X1 X2
2  2  1
3  4  2

[1] 1
[1] -1
```

2 Bayes Network

```
library(bnlearn)
dag = model2network("[location][cost|location:quality][quality][number_of_
    people|location:cost]")

location.values         = factor(c("Good", "Bad"))
quality.values          = factor(c("Good", "Normal", "Bad"))
```

```
cost.values             = factor(c("High", "Low"))
number_of_people.values = factor(c("High", "Low"))

plot(dag)

location.prob         = array(c(0.6, 0.4), dim=2, dimnames=list(location=
    location.values))
quality.prob          = array(c(0.3, 0.5,0.2), dim=3, dimnames=list(
    quality=quality.values))
cost.prob             = array(c(0.8, 0.2, 0.6, 0.4, 0.1, 0.9, 0.6, 0.4,
    0.6, 0.4, 0.05, 0.95), dim=c(2,3,2), dimnames=list(cost=cost.values,
    quality=quality.values, location=location.values))
number_of_people.prob = array(c(0.6, 0.4, 0.8, 0.2, 0.1, 0.9, 0.6, 0.4),
    dim=c(2, 2, 2), dimnames=list(number_of_people=number_of_people.values,
     cost=cost.values, location=location.values))

conditional_probability_table = list(location=location.prob, quality=
    quality.prob, cost=cost.prob, number_of_people=number_of_people.prob)
print(conditional_probability_table)

bayes_network = custom.fit(dag, conditional_probability_table)

location         = factor(c("Good", "Bad"))
quality          = factor(c("Normal", "Good"))
ccost            = factor(c("High", "Low"))
number_of_people = factor(c("Low", "High"))

d_test        = data.frame(location, quality, ccost, number_of_people)
names(d_test) = c("location", "quality", "cost", "number_of_people")
prediction    = predict(bayes_network, "quality", d_test, debug=FALSE)

table(prediction, d_test[,"quality"])
print(prediction)
```

The network looks as follows:



The output looks as follows:

```
$location
location
Good  Bad
 0.6  0.4

$quality
quality
  Good Normal    Bad
   0.3    0.5    0.2

$cost
, , location = Good

      quality
cost   Good Normal Bad
  High  0.8    0.6 0.1
  Low   0.2    0.4 0.9

, , location = Bad

      quality
cost   Good Normal  Bad
  High  0.6    0.6 0.05
  Low   0.4    0.4 0.95


$number_of_people
, , location = Good

                cost
number_of_people High Low
            High  0.6 0.8
            Low   0.4 0.2

, , location = Bad

                cost
number_of_people High Low
            High  0.1 0.6
            Low   0.9 0.4
```

```
prediction Good Normal
    Good      0      0
    Normal    1      1
    Bad       0      0
[1] Normal Normal
Levels: Good Normal Bad
```

3 DBScan

```
using RDatasets, Clustering, Distances, Gadfly

cars = dataset("datasets", "mtcars")
x = convert(Array, cars[:,3])'
y = convert(Array, cars[:,11])'
distances = pairwise(Euclidean(), x, y)

cluster = dbscan(distances, 2, 5)

assignments = assignments(cluster)
cluster_plot = plot(x=x, y=y, color=assignments, Geom.point);

print(cluster)
```

```
Clustering.DbscanResult(
    [1, 5, 7, 12, 13, 14, 15, 16, 17, 22, 23, 24, 25, 29, 31],
    [1, 1, 1, 1, 2, 1, 3, 1, 1, 1, 1, 4, 5, 6, 7, 8, 9, 1, 1, 1,
        1, 10, 11, 12, 13, 1, 1, 1, 14, 1, 15, 1],
    [18, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

Gadfly could not produce a graph.

4 Decision Tree

```
using RDatasets, DecisionTree, DataFrames

cars = dataset("datasets", "mtcars")

features = convert(Array, cars[:, [12, 3, 5, 11]])
classification = convert(Array, cars[:, 2])
model = build_tree(classification, features)

print_tree(model)
```

```
Feature 3, Threshold 118.0
L-> Feature 3, Threshold 96.0
    L-> Feature 3, Threshold 65.5
        L-> Feature 1, Threshold 1.5
            L-> 33.9 : 1/1
            R-> Feature 3, Threshold 57.0
                L-> 30.4 : 1/1
                R-> 24.4 : 1/1
        R-> Feature 3, Threshold 92.0
            L-> Feature 1, Threshold 1.5
                L-> 27.3 : 1/2
                R-> 26.0 : 1/1
            R-> 22.8 : 2/2
    R-> Feature 1, Threshold 3.0
        L-> Feature 3, Threshold 107.0
            L-> Feature 2, Threshold 5.0
                L-> 21.5 : 1/1
                R-> 18.1 : 1/1
            R-> Feature 4, Threshold 4.5
                L-> 21.4 : 2/2
                R-> 30.4 : 1/1
        R-> 21.0 : 2/2
R-> Feature 3, Threshold 192.5
    L-> Feature 2, Threshold 7.0
        L-> Feature 4, Threshold 4.5
            L-> 17.8 : 1/2
            R-> 19.7 : 1/1
        R-> Feature 1, Threshold 2.5
            L-> Feature 3, Threshold 162.5
                L-> 15.2 : 1/2
                R-> 19.2 : 1/2
            R-> 16.4 : 1/3
    R-> Feature 3, Threshold 237.5
        L-> Feature 3, Threshold 222.5
            L-> 10.4 : 2/2
            R-> 14.7 : 1/1
        R-> Feature 3, Threshold 254.5
            L-> 14.3 : 1/2
            R-> Feature 3, Threshold 299.5
                L-> 15.8 : 1/1
                R-> 15.0 : 1/1
```

5 KMeans

```
using CSV, DataFrames
using Clustering
using Gadfly

data_frame = CSV.read("./problem-5.csv"; types=[Float64, Float64, Float64
    ])
data = convert(Array, data_frame[:, [1, 2]])'

k_means = kmeans(data, 3; maxiter=200, display=:iter)

a = assignments(k_means)
c = counts(k_means)

print("Assignments: ")
println(a')

print("Counts: ")
println(c')

plot(x=data[1, :], y = data=[2, :], color=a, Geom.point)
```

```
  Iters               objv        objv-change | affected
------------------------------------------------------------
      0       1.000000e+02
      1       4.880000e+01      -5.120000e+01 |        0
      2       4.880000e+01       0.000000e+00 |        0
K-means converged with 2 iterations (objv = 48.80000000000001)
Assignments: [1 1 1 1 2 1 3]
Counts: [5 1 1]
```