Jeff Hildebrandt

COT5405 Programming Assignment 2

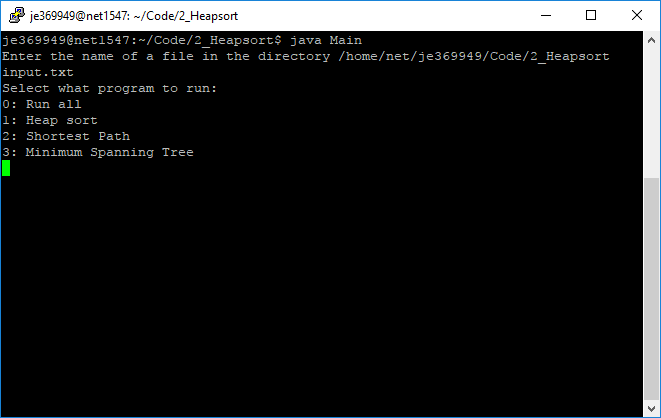
**Purpose:**

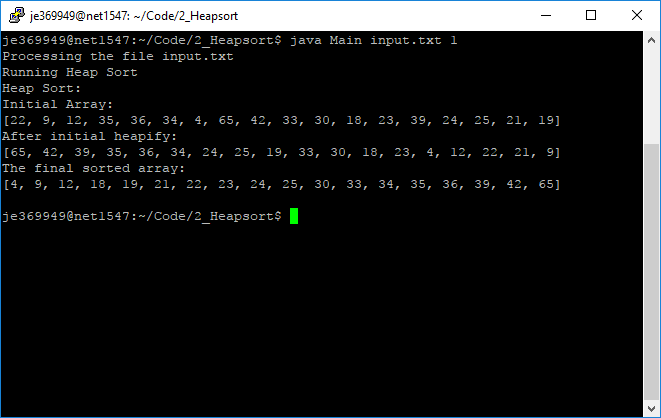
This assignment goes over heapsort, shortest path, and minimum spanning tree. It demonstrates how these common problems can be solved in a relatively quick running time, using long established algorithms.

**Language:**

Java

**Compilation Instructions:**

* All three problems were done in the same file, they can be run one at a time, or all at once. The program is copied three times for each problem. The folders are:
  + 2\_Heapsort
  + 3\_ShortestPath
  + 4\_MinimumSpanningTree
* Each program can be run by doing a cd into the folder then running java Main
* If you do java Main it gives you a prompt for selecting a file, then selecting the problem to run
* 
* You can also pass the file and selection in as a parameter for example:
  + Java Main input.txt 1 will run heapsort



For Heapsort:

cd 2\_Heapsort

javac Main.java

java Main input.txt 1

For Shortest Path:

cd 3\_ShortestPath

javac Main.java

java Main input.txt 2

For Minimum spanning Tree:

cd 4\_MinimumSpanningTree

javac Main.java

java Main input.txt 3

java Main good.txt 3

Note: good.txt is an example of a tree that is not strongly connected

For all three run from the same file

Java Main input.txt 0

**Format of Input:**

Here is an example of good input:

A, 22 B, 9 C, 12 D

B, 22 A, 35 C, 36 F, 34 H

C, 9 A, 35 B, 4 D, 65 E, 42 F

D, 12 A, 4 C, 33 E, 30 I

E, 65 C, 33 D, 18 F, 23 G

F, 36 B, 42 C, 18 E, 39 G, 24 H

G, 23 E, 39 F, 25 H, 21 I

H, 34 B, 24 F, 25 G, 19 I

I, 30 D, 21 G, 19 H

It also supports non-linked nodes, and directional inputs represented here:

A, 22 B, 9 C, 12 D

B, 35 C, 36 F, 34 H

C, 9 A, 35 B, 4 D, 65 E, 42 F

D, 12 A, 4 C, 33 E, 30 I

E, 65 C, 33 D, 23 G

F, 36 B, 42 C, 18 E, 39 G, 24 H

G, 23 E, 39 F, 25 H, 21 I

H, 34 B, 24 F, 25 G, 19 I

I, 30 D, 21 G, 19 H

J, 5 K

K, 5 J, 49 L

L, 49 K

M, 9 N

N, 9 M

O

For the first line A, 22 B, 9 C, 12 D: A represents the node being defined. 22 B represents the edge from A to B and the weight of that edge. Each edge and the node being defined are separated by commas, the weight and edge node are separated by a space. The nodes can be strings of any length.

Here are examples of bad input:

1. Switching edge and node locations

A, F 33, 22 B, 9 C, 12 D

B, 22 A, 35 C, 36 F, 34 H

C, 9 A, 35 B, 4 D, 65 E, 42 F

D, 12 A, 4 C, 33 E, 30 I

E, 65 C, 33 D, 18 F, 23 G

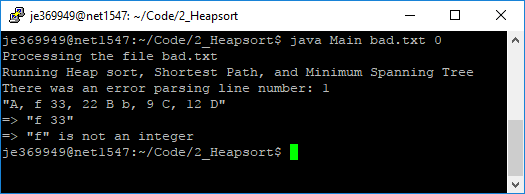
F, 36 B, 42 C, 18 E, 39 G, 24 H

G, 23 E, 39 F, 25 H, 21 I

H, 34 B, 24 F, 25 G, 19 I

I, 30 D, 21 G, 19 H

Output:



1. Too many inputs for an edge

A, 22 B, 9 C, 12 D

B, 22 A, 35 C, 36 F, 34 H

C, 9 A, 35 B, 4 D, 65 E, 42 F

D, 12 A, 4 C, 33 E, 30 I

E, 65 C, 33 C D, 18 F, 23 G

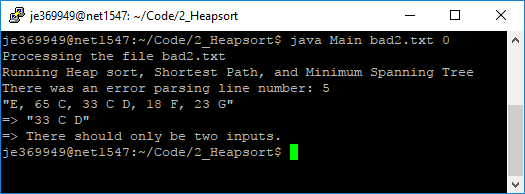
F, 36 B, 42 C, 18 E, 39 G, 24 H

G, 23 E, 39 F, 25 H, 21 I

H, 34 B, 24 F, 25 G, 19 I

I, 30 D, 21 G, 19 H

Output:



1. Defining nodes more than once

A, 22 B, 9 C, 12 D

B, 22 A, 35 C, 36 F, 34 H

C, 9 A, 35 B, 4 D, 65 E, 42 F

D, 12 A, 4 C, 33 E, 30 I

E, 65 C, 33 D, 18 F, 23 G

F, 36 B, 42 C, 18 E, 39 G, 24 H

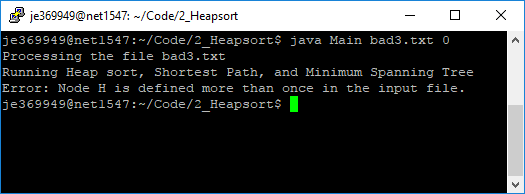
G, 23 E, 39 F, 25 H, 21 I

H, 34 B, 24 F, 25 G, 19 I

H, 34 B, 24 F, 25 G, 19 I

I, 30 D, 21 G, 19 H

Output:



1. Having an edge that links to a node that doesn’t exist

A, 22 B, 9 C, 12 D

B, 22 A, 35 C, 36 F, 34 H

C, 9 A, 35 B, 4 D, 65 E, 42 F

D, 12 A, 4 C, 33 N, 30 I

E, 65 C, 33 D, 18 F, 23 G

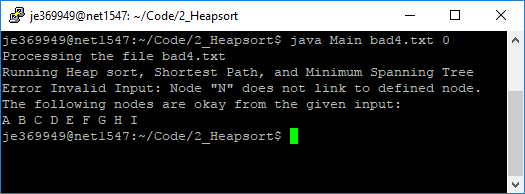
F, 36 B, 42 C, 18 E, 39 G, 24 H

G, 23 E, 39 F, 25 H, 21 I

H, 34 B, 24 F, 25 G, 19 I

I, 30 D, 21 G, 19 H

Output:



**Problems:**

adjacency matrix

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F | G | H | I |
| A | 0 | 22 | 9 | 12 | 0 | 0 | 0 | 0 | 0 |
| B | 22 | 0 | 35 | 0 | 0 | 36 | 0 | 34 | 0 |
| C | 9 | 35 | 0 | 4 | 65 | 42 | 0 | 0 | 0 |
| D | 12 | 0 | 4 | 0 | 33 | 0 | 0 | 0 | 30 |
| E | 0 | 0 | 65 | 33 | 0 | 18 | 23 | 0 | 0 |
| F | 0 | 36 | 42 | 0 | 18 | 0 | 39 | 24 | 0 |
| G | 0 | 0 | 0 | 0 | 23 | 39 | 0 | 25 | 21 |
| H | 0 | 34 | 0 | 0 | 0 | 24 | 25 | 0 | 19 |
| I | 0 | 0 | 0 | 30 | 0 | 0 | 21 | 19 | 0 |

Adjacency list

A, 22 B, 9 C, 12 D

B, 22 A, 35 C, 36 F, 34 H

C, 9 A, 35 B, 4 D, 65 E, 42 F

D, 12 A, 4 C, 33 E, 30 I

E, 65 C, 33 D, 18 F, 23 G

F, 36 B, 42 C, 18 E, 39 G, 24 H

G, 23 E, 39 F, 25 H, 21 I

H, 34 B, 24 F, 25 G, 19 I

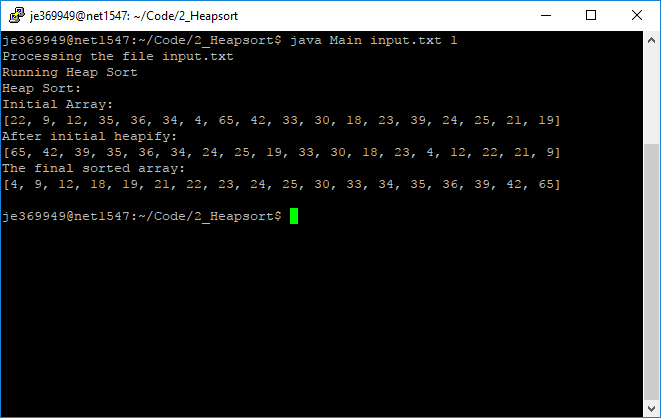
I, 30 D, 21 G, 19 H

1. **Heap and Heapsort**

Array after heapified:

The runtime of the algorithm was done in O(nlogn) time, heapify was used rather than siftUp and siftDown. Using heapify actually decreases the runtime slightly as the beginning siftUp command runs the entire n length of the array and the heapify runs at n/2. It’s a slight performance increase, but not enough to change the time complexity.

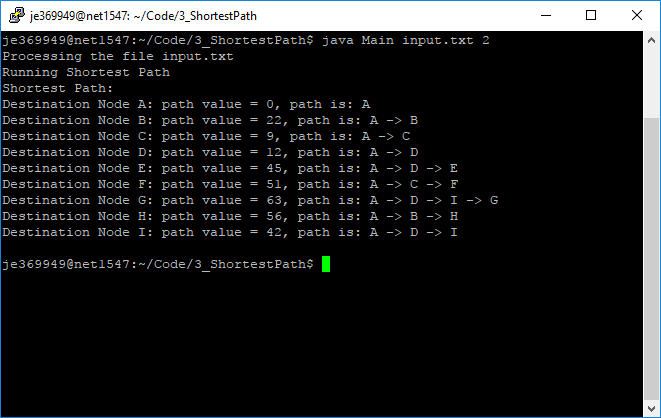
Output:



1. **Shortest path**

Dijkstra’s Algorithm done using Java’s PriorityQueue. It’s a close implementation of Dijkstra’s Algorithm as represented by the pseudo code on Wikipedia here https://en.wikipedia.org/wiki/Dijkstra%27s\_algorithm. However, all adjacent nodes are stored in a hashmap instead of an array due to the node names being saved as strings rather than integers.

Output:

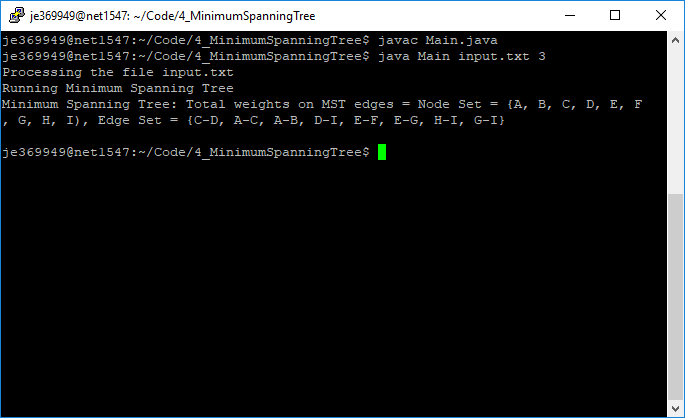


1. **Minimum Spanning Tree**

I implemented Kruskal’s Algorithm somewhat differently from any other implementation I’ve seen. I used a list of hashsets to represent the subsets. For each check it goes through all the sets O(logn) then checks to see if the source and destination nodes both exist in the same subset. Since it is a hashset it can be compared in O(1) time complexity. However, since it’s a hashset when two subsets are unioned it is done in O(n) time. So, any time saved checking the adjacent nodes is lost at this step.

Output:

With input.txt



With good.txt

