Lab 5

Steve Boucher

December 10, 2022

Bags and Graphs

1 Introduction

The goal of lab 3 was to code multiple weighted graphs that could be used to perform A single source shortest path algorithm following the Bellman Fords algorithm as well as a modified version of the 0-1 knapsack problem.

2 File Reading

Unlike my previous lab assignments, I figured out a way to read commands given from a text file. It took me days and countless hours looking up and scrathing my head trying to figure out how to read the commands before I settled on using and abusing a function that selects a character in a string.

`String.charAt(int);`

Using this command. I was able to set up a file reader which converts each line of a text file and holds onto it as a string temporarily. Using that I was able to run test cases to look for key letters that could tip off the code on what it needed to do.

```
43 if (textCommand != "") {
44                 // Check for comments
45
46                 if ((textCommand.charAt(0)) == '-') {
47
48                   System.out.println(textCommand);
49                 }
50                 // Checks to see if we are making a new graph
 51                if (textCommand.charAt(0) == 'n') {
52
53                     graph = new Graph();
54              }
55                 if (textCommand.charAt(0) != '-' && textCommand.charAt(0) != 'n'){
...
{
```

In the code above. I used multiple if statements along with charAt() to look to see if the first digit is a -. If so, skip/print it as it is a comment. An 'n' means the starting letter is an n, and we will be constructing a new graph. adding vertexes and edges uses a similar tactic, checking for the first letter, but then also position 4 and later to make sure we are adding a vertex or edge and the full number for the vertex.

3 Struggle

Where I struggled to read the file for lab 4 but was able to perform the majority of tasks, the opposite follows for 5. I was able to read the text file, But I can't for the life of me figure out how to actually implement the pseudo-code provided on the slide deck.

The Bellman-Ford, Single-Source, Shortest, Path Algorithm is an algorithm for finding the shortest pathway to get from one vertex to another in a weighted directional graph. It does this by initiating its distance edge counters to be as high as possible before relaxing itself to calculate what the actual distances are, The function repeats this step several times before it knows the path to each point. This algorithm is not greedy, meaning it would rather take the overall best option as opposed to the fastest decisions. Due to this nature, this has a asymptotic Worst time complexity because the program needs to run through as many V-1 possibilities or even recalculate due to a negative number edge throwing off the rest of the program (I know it should be able to handle negative numbers . as possible to make the correct decision on what the shortest path is.

4 Handy Haversack

I employed the same methodologies I used for the Graphs File reader for the partial haversack problem. The haversack problem is a simple one of having too much of a good thing. We were supposed to write an algorithm that selects the best combination of spices in order to maximize the value of what's in the bag. My code accomplishes this by calculating the unit price for each item, selectes a spice for the output, and decreasing the quantity of the respective dust. The only issue I have with my spice bag is that it won't look to the red and green spices despite having no issues grabbing orange and blue spices.

```
Main.java
240     System.out.println("An optimized knapsack size holds: " +  bagSize + ": ");
242     for (int i = 0; i < bagSize; i++) {
243      TotalPlunder = TotalPlunder + spiceList.Snatch();
244     }
245   System.out.println("Valuing: " + TotalPlunder + " Gold Pieces");
246   }

List.java
50    public int Snatch() {
51        Spice spice = head;
52        int max = 0;
53        String nameMax = " ";
54        int TotalPlunder = 0;
55        while (spice.next != null) {
56            max = 0;
57            if (spice.unitprice > max && spice.ammount > 0) {
58                max = spice.unitprice;
59                nameMax = spice.name;
60                spice.ammount--;
61            }
62            spice = spice.next;
63        }
64        if (spice.unitprice > max && spice.ammount > 0) {
65            max = spice.unitprice;
66            nameMax = spice.name;
67            spice.ammount--;
68
69        }
70        TotalPlunder = TotalPlunder + max;
71        System.out.println(nameMax);
72        return TotalPlunder;
73    }
{
```

The knapsack problem has an asymptotic run time of O(NlogN) as it takes N time to go through the list of spices and sort them, and N time to go through the rest of the problem of filling the bag.

5 Conclusion

This One took a lot of coding to achieve between the lines and lines of code to read the file along with the actual code for the problems. It felt good each step of the way setting up the file reader as each if statement got me closer to getting into the actual program. Once I got there, It was a fun journey troubleshooting and coming up with a plan to sort the items. I had a bunch of fun during part two of this, but could not figure out any of the wording/symbols/variables used in the pseudo-code. I will be trying to look at this and fix it at some point in the near future if I have time because I would love for this to work accordingly.