CMPS 2200 Introduction to Algorithms

Programming Project 2

Part 1 is due 2019-11-20 at 11:59pm on Canvas
Part 2 is due 2019-11-25 at 11:59pm on Canvas
Part 3 is due 2019-12-06 at 11:59pm on Canvas

Dijkstra's Algorithm Applied (50 points + 30 bonus points)

In this assignment, you will be implementing Dijkstra's Algorithm and applying it to real-world road map data.

Input:

Your program will read in DIMACS format .gr files containing United States road map data. These input files may be found here. Distance and travel time graphs are provided. You may test your program using either. These files contain the list of edges that make up a road map graph, each is denoted by node IDs and the weight (either distance or time) of the edge between them. There are also .co files which specify the latitude and longitude coordinates for each node ID. Your program is not required to read in the .co files.

Output:

Your program will write to an output file the shortest path between a source node ID and a target node ID. On the first line will output the total weight of the path. After that you will output the shortest path, one node ID per line, source node first and target node last.

Example output file (generated from the New York City distance graph):

```
6529
1
1363
1358
1357
1356
1276
```

Program:

You will write a program `FindShortestPath` that takes in four command line arguments: the input graph file, the source node ID, the target node ID, and the outputFileName. As specified above, it will write the total cost and the shortest path from source to target to the output file.

Part 1: (15 points)

Implement code that reads in a directed weighted graph from a graph input file into an adjacency list structure. You are required to implement your own graph data structure.

Part 2: (10 points)

Programming Project 2

Part 1 is due 2019-11-20 at 11:59pm on Canvas
Part 2 is due 2019-11-25 at 11:59pm on Canvas
Part 3 is due 2019-12-06 at 11:59pm on Canvas

Implement a `PriorityQueue` using a Min-Heap. You may adapt your code from Project 1 for this purpose. Recall that Dijkstra's algorithm requires operations `poll()` to remove & return the smallest value in the queue and `decreaseKey()` to decrease the key of some value in the queue. These are the operations that you will need to add to your implementation.

Part 3: (25 points)

Implement Dijkstra's Algorithm and produce the output file.

Bonus:

Fibonacci Heap: (10 points)

Instead of adapting your heap from Project 1 to serve as your Priority Queue, implement a Fibonacci Heap for your Priority Queue.

Draw Routes on a Map: (20 points)

Write a second program that takes as input the output file of your `FindShortestPath` program and a .co file (see input section above) and draws that route on a map. Which mapping software to use and how perform the drawing are up to you. Provide instructions on how to use your program and view its output in a README.

Notes:

- I have provided for you a program `FindClosestNode` which takes in a .co file and latitude and longitude coordinates and prints the node ID of the closest node to those coordinates.
- The example output given above is the shortest path from node 1 to 1276 in the New York distances graph.

Comment your code thoroughly. Points will be deducted for poorly documented code. Projects must be completed individually; group work is not allowed.

Submission

- You may use Python, Java, C, or C++ for this project.
- Turn in all individual source files on Canvas.
- For your final submission submit ALL files to Project 1 – Part 3 on Canvas.
- All projects must compile and run. If you program does not compile, you will receive 0 points for it.
- Do not use any fancy libraries. You are required to implement your own Graph and PriorityQueue data structures. We should be able to compile your programs under the standard installs of Python, Java, C, or C++.