CS4310 Assignment 6: (Programming)

Dijkstra’s Shortest Path Algorithm

In this assignment, you are asked to find and print out the shortest path between two places on a graph of US road network using Dijkstra's shortest-path algorithm.

The data you will work on is a subset of major roads from the National Highway Planning Network. There are two data files, Road.txt and Place.txt, in the attached USRoads.zip. Both files are in plain text format. Each line in Road.txt represents one road segment. Each line in Place.txt represents a place. Data values are separated by comma (,). Meanings of data values are given as follows:

Road.dat:
#1 (integer), place ID of one end of the road segment
#2 (integer), place ID of the other end of the road segment
#3 (float), length in miles
#4 (string), signs and/or description of the road segment, somewhat garbled.

Place.dat:
#1 (integer), place ID
#2 (string), place name.

The file Road.txt contains 127,806 road segments and 91,203 place IDs. Only some places have names given in Place.txt. The file Place.txt contains 25252 place IDs and their names.

All road segments are two-way roads. The record (A,B,miles,name) represents both the road from A to B and from B to A. Therefore, you will work on an undirected graph.

The graph has more than one component. Road.txt contains roads in Alaska and Hawaii.

Implementation Notes:

For solving real world problems, choosing the right data representation is very important. Observe that the graph has 91,203 nodes and 127,806 edges. You should choose the right data structure to represent the data as a graph.

At the core of Dijkstra’s shortest path algorithm is a priority queue. For big problems of this size, you want to use an efficient data structure.

The graph has disconnected components. If the user asks how to go from Kalamazoo to Honolulu, your program should know when to stop the search.

The road graph can be constructed by reading the file Road.txt. You are suggested to read a line each time into a string, split the string by comma, and convert each substring into a value of the corresponding data type.

Your program should ask the user for the source place name and the destination place name. Given the place names, your program should use the data in Place.txt to search for source and destination place IDs. You can expect exact string match of user inputs of place names to find place IDs. Do not spend too much time on matching inaccurate place names.

If you use Java, Java class String has useful methods for string processing. **More importantly, Java Collection Framework (JCF)** has useful classes and interfaces that may help to greatly reduce your programming work. These classes/interfaces include ArrayList, PriorityQueue, HashMap, HashSet, etc.

Your program should print out the result path of road segments in sequence and the total distance of the path. For each segment of the path, print using the following format:

```
#: from_placeID (place name) -> to_placeID (place name), road description, length
```

Your output should follow the following format:

```
C:\Users\yang\Desktop\Roads>java DSP
Enter the Source Name:
MIKALAMAZOO N
Enter the Destination Name:
MIANN ARBOR N
Searching from 39087(MIKALAMAZOO N) to 39059(MIANN ARBOR N)
        1: 39087(MIKALAMAZOO N) -> 39123(null), U131 BARK ST, 1.11 mi.
        2: 39123(null) -> 39122(null), O94  BALAMAZOO AV, 0.38 mi.
        3: 39122(null) -> 39121(null), O94  BALAMAZOO AV, 0.21 mi.
        4: 39121(null) -> 39119(null), O94  BALAMAZOO AV, 0.16 mi.
        5: 39119(null) -> 39156(null), O94  BMVET MEMORIAL PKWY, 1.23 mi.
        6: 39156(null) -> 39206(null), O94  BMVET MEMORIAL PKWY, 1.82 mi.
        7: 39206(null) -> 39217(MI I94 MP081), O94  BMVET MEMORIAL PKWY, 0.82 mi.
        8: 39217(MI I94 MP081) -> 39213(null), I94, 0.37 mi.
        9: 39213(null) -> 39212(null), I94, 2.32 mi.
        10: 39212(null) -> 39193(null), I94, 8.57 mi.
        11: 39193(null) -> 39194(null), I94, 0.05 mi.
        12: 39194(null) -> 39196(MI I94 MP092), I94, 0.08 mi.
        13: 39196(MI I94 MP092) -> 39236(null), I94, 5.01 mi.
        14: 39236(null) -> 39222(MIBATTLE CREEK S), I94, 1.14 mi.
        15: 39222(MIBATTLE CREEK S) -> 39152(null), I94, 2.93 mi.
        16: 39152(null) -> 39146(null), I94, 0.68 mi.
        17: 39146(null) -> 39103(MI I94 MP104), I94, 1.72 mi.
        18: 39103(MI I94 MP104) -> 39112(MIMARSHALL NW), I94, 4.59 mi.
        19: 39112(MIMARSHALL NW) -> 39108(null), I69, 0.07 mi.
        20: 39108(null) -> 39127(null), I94, 1.72 mi.
        21: 39127(null) -> 39160(null), I94, 1.48 mi.
        22: 39160(null) -> 39163(null), I94, 0.21 mi.
        23: 39163(null) -> 39162(null), I94, 0.4 mi.
        24: 39162(null) -> 39228(MIALBION N), I94, 9.31 mi.
        25: 39228(MIALBION N) -> 39230(null), I94, 2.0 mi.
        26: 39230(null) -> 39232(MI I94 MP124), I94, 0.46 mi.
        27: 39232(MI I94 MP124) -> 39218(null), I94, 11.54 mi.
        28: 39218(null) -> 39219(MIJACKSON WNW), I94, 0.35 mi.
        29: 39219(MIJACKSON WNW) -> 39220(null), I94, 1.23 mi.
        30: 39220(null) -> 39215(null), I94, 0.69 mi.
        31: 39215(null) -> 39198(MIJACKSON NW), I94, 0.66 mi.
        32: 39198(MIJACKSON NW) -> 39202(null), I94, 1.13 mi.
        33: 39202(null) -> 39201(MIJACKSON N), I94, 0.06 mi.
        34: 39201(MIJACKSON N) -> 39187(null), I94, 1.08 mi.
        35: 39187(null) -> 39179(null), I94, 1.18 mi.
        36: 39179(null) -> 39178(MIJACKSON NE), I94, 0.07 mi.
        37: 39178(MIJACKSON NE) -> 39176(MI I94 MP145), I94, 1.99 mi.
        38: 39176(MI I94 MP145) -> 39167(null), I94, 0.95 mi.
        39: 39167(null) -> 39109(null), I94, 8.96 mi.
        40: 39109(null) -> 39120(null), I94, 5.4 mi.
        41: 39120(null) -> 39107(null), I94, 6.52 mi.
        42: 39107(null) -> 39105(null), I94, 1.1 mi.
        43: 39105(null) -> 39149(null), I94, 3.87 mi.
        44: 39149(null) -> 39139(null), S14, 0.45 mi.
        45: 39139(null) -> 39100(MIANN ARBOR C-N), S14, 2.9 mi.
        46: 39100(MIANN ARBOR C-N) -> 39059(MIANN ARBOR N), U23  B, 1.52 mi.
It takes 100.49 miles from 39087(MIKALAMAZOO N) to 39059(MIANN ARBOR N).
```

Report:

Prepare a report and submit with your code. The report should consist of the following parts:

- Ideas and experimental procedures: Explain your main ideas and hardware/software configurations for the experiment.
- Include one example output of running your program, as shown above.
- Analysis: How did you represent the graph and how did you implement the priority queue? What packages/libraries did you use? Why did you do that way? What is the asymptotic complexity of your run time? What is the experimental run time?
- Conclusion: what did you learn from this homework assignment?

Grading:

- Report: 30%
- Code: 70%
    - construction of the graph: 20%
    - Dijkstra’s algorithm: 50% (20% on priority queue)

Pack your answers in one zip file and submit to elearning.