CS 4310-100
Assignment 6: Dijkstra's Shortest Path Algorithm

Chandler Klein

04/17/2020

Ideas and Experimental Procedures

This project focuses on solving the shortest path problem by utilizing Dijkstra's algorithm, using a binary min-heap representing a priority queue, as well as various hash maps. The data in USRoads/Place.txt file, the final distances from the source vertex to each other vertex, and the parent of each vertex are stored in the maps.

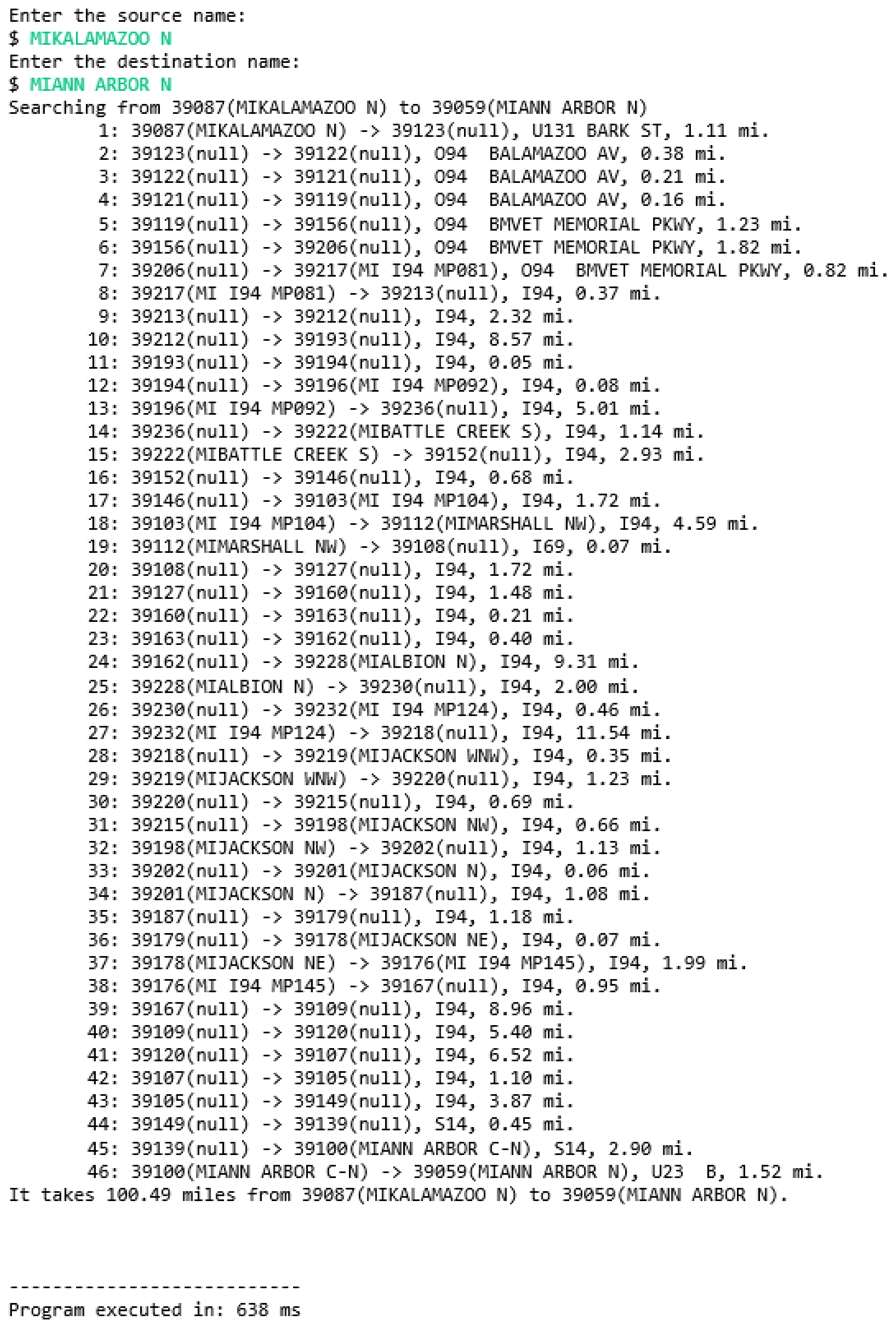
Environment:

- Windows
- Eclipse IDE version 4.14.0
- Java – JDK 13.0.2

Modified code used for binary min-heap (priority queue) implementation - reference on last page. Also included is the Apache v2 License.

The next page shows example output after running the program.

Example Output



```
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, 0.40 mi.
        24: 39162(null) -> 39228(MIALBION N), I94, 9.31 mi.
        25: 39228(MIALBION N) -> 39230(null), I94, 2.00 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, 5.40 mi.
        41: 39120(null) -> 39107(null), I94, 6.52 mi.
        42: 39107(null) -> 39105(null), I94, 1.10 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, 2.90 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).



---------------------------
Program executed in: 638 ms
```

Analysis

The graph in this program is represented by data structures storing all of the edges, all of the vertices, and a map that contains the places and their IDs. I was going to originally use the PriorityQueue included in the Java Collection Framework (JCF), but it was missing a couple of methods that made Dijkstra's algorithm even simpler. The priority queue that I have included in the program is a modified version of a binary min-heap - the source is included in the references at the end of this document.

The binary min-heap mainly has functions for adding to the heap, getting and extracting the minimum value, and decreasing the weight of a given key to a new weight. These functions are required when performing the main operations of Dijkstra's algorithm, which includes retrieving minimum-valued keys and decreasing the weight - or distance - of keys in the "relaxation" step.

Since the minimum value is retrieved efficiently from a binary min-heap, this decreases the overall complexity of the program compared to the complexity of a linear search method of an adjacency list or an ordinary linked list. There are multiple auxiliary data structures used in this program, resulting in a space complexity of $\Theta(E + V)$, where E is the number of edges, and V is the number of vertices. The nature of the binary min-heap data structure requires $\Theta((E + V) \log V)$ as the time complexity, where E is the number of edges, and V is the number of vertices. This time complexity could be improved even further by using a data structure called a Fibonacci heap, which results in a time complexity of $\Theta(E+V \log V)$.

Conclusion

I learned a lot from this homework assignment, including greater usage of priority queues (using binary min-heap), and general format and complexity of Dijkstra's algorithm. I became more comfortable with using data structures in general after finishing this project.

References

Modified code for binary min-heap (priority queue): https://github.com/mission-peace/interview

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm