**Data Structures Mini-Project**

Path Finder Using Dijkstra’s Algorithm

By

Nikhil Prabhu

01FB14ECS131

Nirmit Shetty

01FB14ECS134

Pramod KM

01FB14ECS154

Pradeep M

01FB14ECS149

Description

What would happen if you were given a choice of a million roads, but only a limited amount of time? What if you only needed to reach a certain place, but these roads seem to wind around endlessly? With enough data about them, computers can draw them into graphs and can even find the shortest path between two different places.

Graphs have been proven to be useful in all sorts of applications, ranging from Chemistry and Biology to even everyday Geography. The Dutch computer scientist and mathematician Edsger Wybe Dijkstra formulated a famous algorithm to find the shortest path between two nodes in a weighted graph.

Thus, our project is a visual demonstration of this incredibly useful algorithm, in the hopes that we make it easier to understand and visualize.

Data Structures Used

**Arrays** : They are used extensively throughout the code.

**Adjacency Matrix** : This is how the weighted graph is stored internally.

**Adjacency List :** The adjacency matrix is converted into adjacency lists for processing.

**Min Heap** : This is what is used to implement Dijkstra’s Algorithm. While a more primitive implementation is possible, using a min heap makes the algorithm notably faster; near logarithmic time.

**Priority Queue** : It is used to implement the min heap.

Design

The project program, in a nutshell, generates a random weighted graph for a certain number of nodes. A certain source node must be selected by the user. The algorithm then finds the shortest path from that node to all the other nodes in the graph. When the user selects another node, the shortest path from the source node to the selected destination is highlighted.

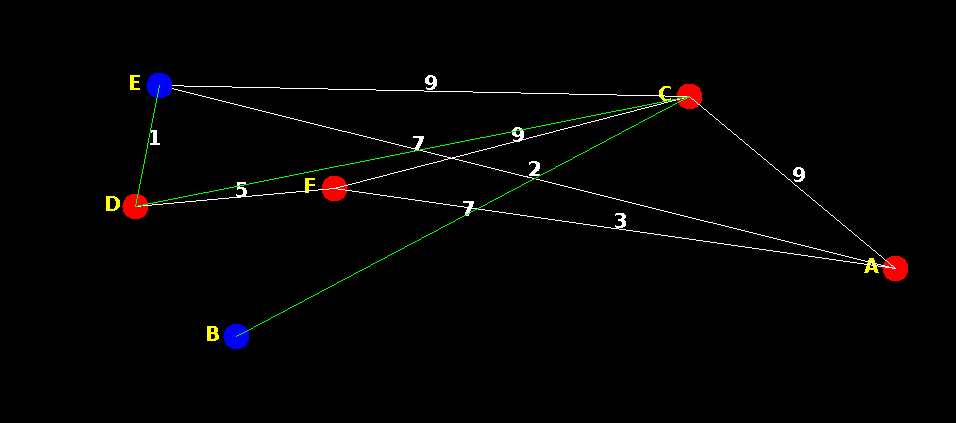
Going step by step:

1. A random adjacency matrix is generated for the nodes listed in the array ‘letters’ in the Client class.
2. The paint() method draws out the graph in the GUI using awt.
3. On clicking a node in the GUI, it is considered to be the source and is passed to the Dijkstra’s Algorithm code in the HeapPathFinder class.
4. The algorithm then finds the shortest paths from the source to all other nodes as follows:
5. The adjacency matrix and the source node are taken in as inputs/arguments. The program converts the adjacency matrix is converted into adjacency lists and processes the data using them.
6. The algorithm initializes 2 invariants: X, which contains all the nodes which have already been traversed through the course of the program, and a min heap which stores all the nodes which have not been traversed yet.
7. Dijkstra’s algorithm works by greedily selecting the shortest path from the nodes already visited. It compares the sums of the weights of every node in X to the min heap and picks the node with the smallest value. In our implementation, the nodes in the heap are sorted based on their Dijkstra scores, i.e. the shortest possible path from the source to that node through the nodes in X. Hence, the node at the top of the heap is added to X.
8. As the nodes are added to X, the paths are stored in the array shortestPath[]. Thus, the shortest paths from the source node to all other nodes are successively added to shortestPath[]. This array is then passed back to the Client program.
9. The shortest paths to all the nodes are kept ready as it waits for user input.
10. Once the user selects a node, the corresponding path is fetched from the shortestPath array and is then highlighted.
11. Selecting another node will now repeat the process from step 3.

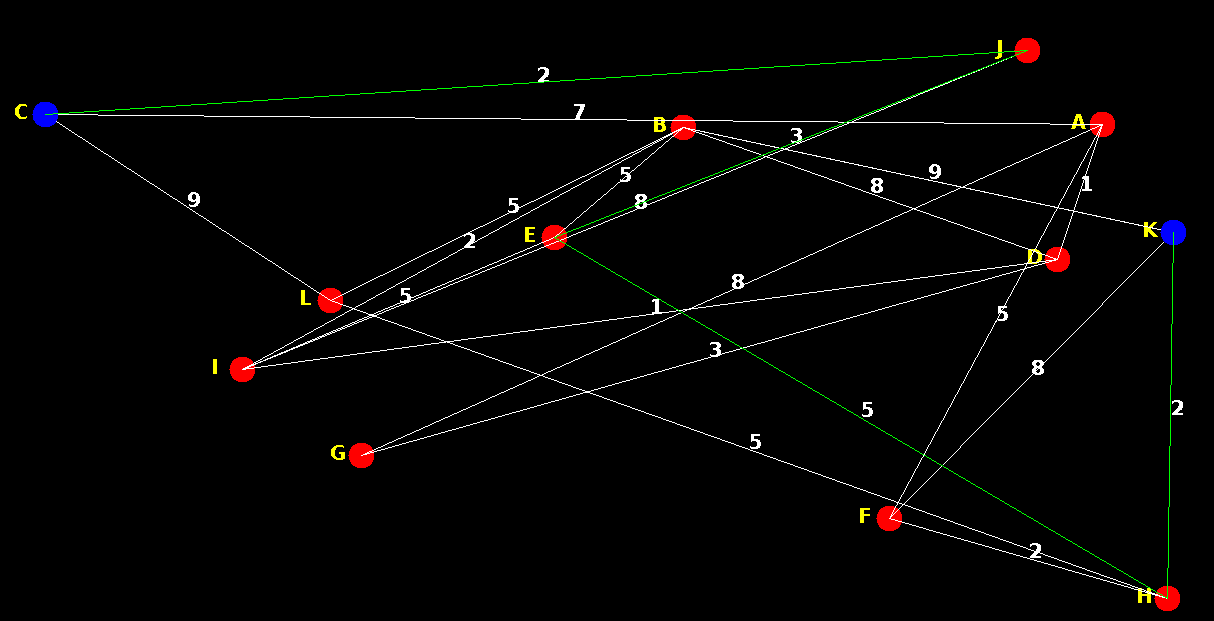
Tool(s) Used

Abstract Window Toolkit (AWT)

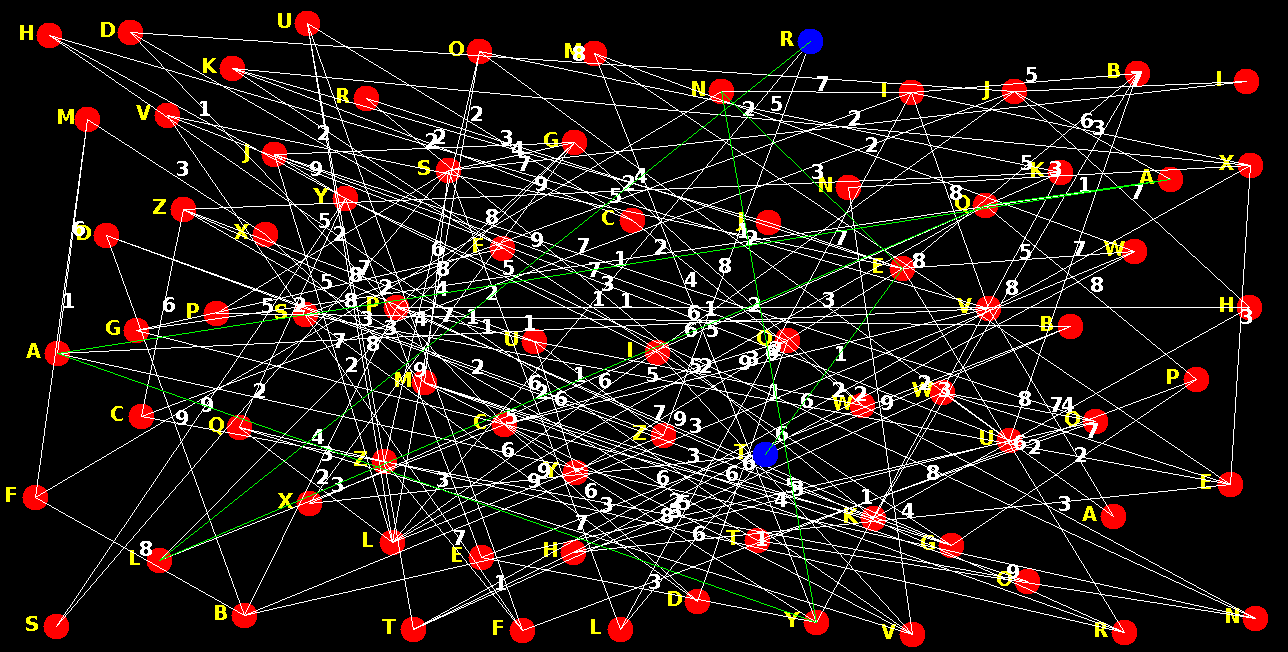
Screen Shots



A Simple and Easy to View Case



A Somewhat More Standard Case



A Very Complex Case Which Shows The Scalability of the Program, but Makes it Hard to see on the GUI

Conclusion

While the algorithm is highly scalable, the GUI limited us from using even a slightly larger data set.

We learned how to use AWT to draw basic shapes, the implementation and uses of Dijkstra’s algorithm, the use of heaps in practical applications and usefulness of Collections.

Further improvements could include using image processing to process maps and find shortest routes between places.