INDIAN INSTITUTE OF TECHNOLOGY

KHARAGPUR

COMPUTER COMMUNICATION LABORATORY
A REPORT ON

"Simulation of Dijkstra's routing algorithm"

November 24, 2020

Y SAI SANJEET
(16EC35025)

DEPT OF ELECTRONICS AND ELECTRICAL COMMUNICATION ENGINEERING

Contents

OBJECTIVE

The aim is to simulate Dijkstra's routing algorithm to find the shortest path from one node to another in a graph.

DISCUSSION

- A Graph object is created and the shortest path from one node to every other node is computed using Dijkstra's algorithm.
- The input graph can contain any arbitrary number of vertices and can have any amount of edges. Checks haven't been made to ensure two different instances of the same edge with different weights do not occur.
- Shortest distances are computing using a priority queue instance from the C++ Standard Template Library.
- The implementation of the algorithm has a time complexity of $O(E \cdot log(V))$, where E is the number of edges in the graph and V is the number of vertices in the graph.
- The implementation stores the information about the edges and weights by making a pair and storing them in each of the corresponding vertice's lists.
- This results in a space complexity of $O(E)$, which is always less than or equal to the space complexity of storing an adjacency matrix that is $O(V^2)$.

CODE EXPLANATION

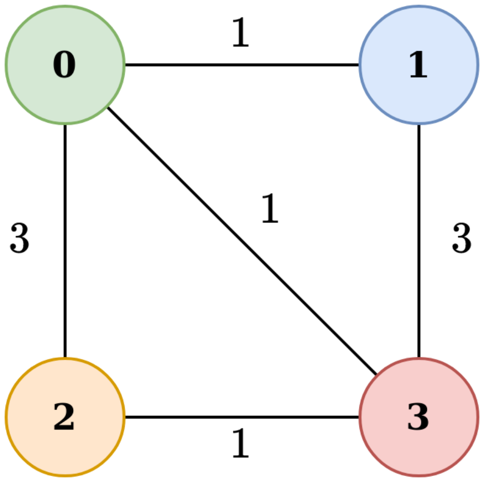
This section gives a brief overview of the code present in `Graph.cpp`.

- The `Graph` class has an adjacency list that stores the edge information of each vertex as an edge is added (the destination node ID and the weight of the edge).
- Edges are added using the `AddEdge` method, which creates two node-weight pairs and stores them in each of the two vertices.
- Dijkstra's algorithm is implemented in the `ShortestPath` method. A priority queue is created and the distances to all vertices are set to infinity.
- The distance to the source node is set to `0` and pushed into the priority queue.

- Single every element at the top of the priority queue is the shortest distance to the corresponding vertex, the topmost element is popped in every iteration and its distance is set.
- Now the distance to every other vertex is computed through this popped vertex and if the distance is smaller than it was earlier, the vertex weight is updated and pushed into the priority queue.
- Once the priority queue is empty, all the updated distances would be the minimum possible distances from the source.

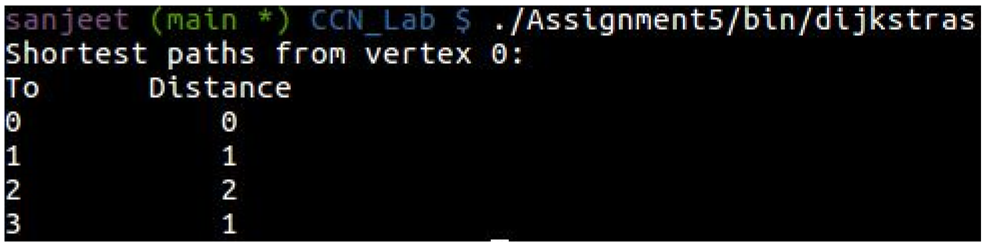
RESULTS

Dijkstra’s algorithm is tested on a simple 4-vertex graph as shown below:



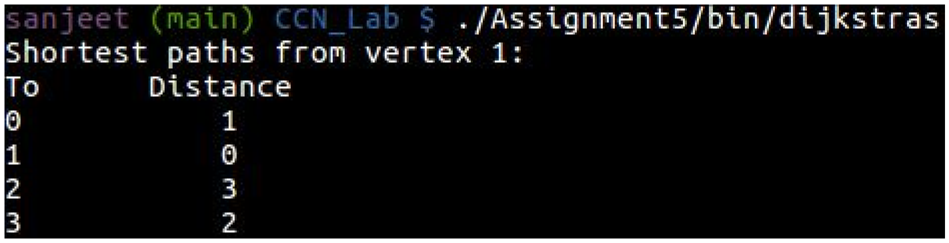
The output of the code:

- The distance of all vertices from Vertex 0:



```
sanjeet (main *) CCN_Lab $ ./Assignment5/bin/dijkstras
Shortest paths from vertex 0:
To      Distance
0           0
1           1
2           2
3           1
```

- The distance of all vertices from Vertex 1:



```
sanjeet (main) CCN_Lab $ ./Assignment5/bin/dijkstras
Shortest paths from vertex 1:
To      Distance
0           1
1           0
2           3
3           2
```

CONCLUSION

- Dijkstra’s routing algorithm is implemented.
- The shortest paths to all vertices are shown from two different vertices.

REFERENCES

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-using-priority_queue-stl/

The graph is drawn using https://app.diagrams.net/