**CSCE 221 Assignment 6 Cover Page**

First Name: **Kaivan** Last Name: **Shah** UIN: **522006920**

User Name: **kks6920**  E-mail address: **kaivan29@tamu.edu**

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more on Aggie Honor.

|  |  |  |  |
| --- | --- | --- | --- |
| Type of Sources |  |  |  |
| People | Kyle Almeida |  |  |
| Web Pages (provide URL) | http://www.cplusplus.com/ | others |  |
| Printed Material | Data Structures and Algorithms in C++ |  |  |
| Other Sources | PowerPoint |  |  |

System Office website: http://aggiehonor.tamu.edu/

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.

On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.

Your Name Kaivan Shah Date 12/8/14

🡪Program Description:

The implementation of Dijkstra’s algorithm in this assignment starts by setting d[u] to 1000000 (that we say as infinity) for each vertex in vertList, while for the source it is 0. This takes O(n) operations.

Then, a priority queue is initialized with all the vertices in vertList, which is O(nlogn) operations. Once, the priority queue is initialized, each minimum element is stored and removed, giving a total of O(nlogn) operations. Finally, each edge is relaxed and keys are updated. In this implementation, this operation is O(n^2), since decreaseKey uses a linear search algorithm. This gives an overall complexity that is O(n^2), while the most efficient complexity is O(nlogn).

With locator, decreaseKey can be executed in constant time, as opposed to linear time. This would reduce the relaxation step to a complexity of O(n), giving Dijkstra’s algorithm an overall complexity of O(nlogn).

While this may not be the most efficient implementation of Dijkstra’s algorithm, the priority queue found in the algorithm is essential to its function. Without it, removing all minimum elements could take more than O(nlogn) operations. For example, if linear search were used with a vector to find the minimum element with each iteration, the cost to remove all elements of the vector would be O(n^2).

🡪Data Structures Used

• Graph

• Binary Heap

• Minimum Priority Queue

• Vector

🡪Dijkstra’s algorithm can be useful in many different applications. For example, an airline network named delta might represent the different airports as nodes in a graph, and flight routes to the airports as its edges with distances represented by weight. This algorithm could then find the shortest route to get from one airport to another. The algorithm could also be used in GPS systems, in which intersections are nodes in a graph and roadways are edges. The algorithm could then calculate the shortest route to your destination. Additionally, weights could also be something else besides distance; for example, if weight represents time or fuel cost, then Dijkstra’s algorithm could compute a path that is either the quickest or most economical.

Dijkstra’s Algorithm is a greedy algorithm in that it looks at the shortest path at that particular instant and does not look into the future which could have an alternate better path.

For simple.txt given vertex 1:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| iter | d[1]/m[1] | d[2]/m[2] | d[3]/m[3] | d[4]/m[4] | d[5]/m[5] | d[6]/m[6] |
| 0 | 0/NULL | ∞/NULL | ∞/NULL | ∞/NULL | ∞/NULL | ∞/NULL |
| 1 | ↓ | 3/[1] | 9/[1] | ∞/NULL | ∞/NULL | ∞/NULL |
| 2 |  | ↓ | 8/[2] | 7/[2] | 4/[2] | ∞/NULL |
| 3 |  |  | 8/[2] | 7/[2] | ↓ | ∞/NULL |
| 4 |  |  | 8/[2] | ↓ |  | 13/[4] |
| 5 |  |  | ↓ |  |  | 13/[4] |
| 6 | 0 | 3/[1] | 8/[2] | 7/[2] | 4/[2] | 13/[4] |

The result for source: 1 and destination: 5 is :-

The shortest path tree is:

3 1

1--->2--->5

The result for source: 1 and destination: 5 is :-

The shortest path tree is:

3 4 6

1--->2--->4--->6

🡪Algorithm Description  
Pseudocode for Dijkstra’s algorithm (member function of Graph class)

Dijkstra(Vertex V):

for(every vertex U in vertList)

d[U] = ∞

π[U] = NULL

d[V] = 0

Initialize priority queue Q to contain all vertices in vertList

while(Q is not empty)

Remove minimum element from Q and call it U;

for(every outgoing vertex Z in U)

relax(U,Z,w(U,Z)) // w(U,Z) is the weight of edge U->Z

Q.updateKey(d[Z], Z)

🡪Running Instructions

Compile the Graphtest.cpp with **make** and run it with **: g++ -std=c++11 GraphTest.cpp**

To run them use: ./a.out.

🡪Bonus part: Initialization then vertex 1