CSCE 221 Cover Page
Programming Assignment # 6

First Name: Devin Last Name: Tuchsen UIN: 121000270

User Name: devint1 E-mail address: devint1@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 in the Aggie Honor System Office http://aggiehonor.tamu.edu/

Type of sources		
People		
Web pages (provide URL)	cplusplus.com stackoverflow.com	http://courses.cs.tamu.edu/teresa/csce221/
Printed material	Class Text	
Other Sources		

I certify that I have listed all the sources that I used to develop the solutions/code 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 (signature) Date

Report

The implementation of Dijkstra's algorithm in this assignment starts by setting $d[u]$ to `INT_MAX` (closest thing to infinity for integers) for each vertex in `vertList` (except the one given), which 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 `updateKey` 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 support, `updateKey` 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 $O(n^2)$.

For simple.txt given vertex 1:

iter	d[1]/π[1]	d[2]/π[2]	d[3]/π[3]	d[4]/π[4]	d[5]/π[5]
0	0/NULL	∞/NULL	∞/NULL	∞/NULL	∞/NULL
1	0	4/1	∞/NULL	∞/NULL	∞/NULL
2	0	4/1	9/2	∞/NULL	∞/NULL
3	0	4/1	9/2	11/3	∞/NULL
4	0	4/1	9/2	11/3	17/4
5	0	4/1	9/2	11/3	17/4

Computational Result (to vertex 5): `1 -[4]-> 2 -[5]-> 3 -[2]-> 4 -[6]-> 5`

All above data is verified.

For complex.txt given vertex 1:

iter	d[1]/π[1]	d[2]/π[2]	d[3]/π[3]	d[4]/π[4]	d[5]/π[5]	d[6]/π[6]	d[7]/π[7]	d[8]/π[8]
0	0	∞/NULL	∞/NULL	∞/NULL	∞/NULL	∞/NULL	∞/NULL	∞/NULL
1	0	10/1	4/1	∞/NULL	2/1	∞/NULL	∞/NULL	∞/NULL
2	0	6/5	4/1	∞/NULL	2/1	∞/NULL	7/5	∞/NULL
3	0	6/5	4/1	5/3	2/1	∞/NULL	7/5	14/3
4	0	6/5	4/1	5/3	2/1	16/4	7/5	14/3
5	0	6/5	4/1	5/3	2/1	16/4	7/5	8/7
6	0	6/5	4/1	5/3	2/1	16/4	7/5	8/7
7	0	6/5	4/1	5/3	2/1	16/4	7/5	8/7
8	0	6/5	4/1	5/3	2/1	16/4	7/5	8/7

Computational results from vertex 1:

To 2:

```
1 -[2]-> 5 -[4]-> 2
Total weight of the shortest path from 1 to 2:  6
```

To 3:

```
1 -[4]-> 3
Total weight of the shortest path from 1 to 3:  4
```

To 4:

```
1 -[4]-> 3 -[1]-> 4
Total weight of the shortest path from 1 to 4:  5
```

To 5:

```
1 -[2]-> 5
Total weight of the shortest path from 1 to 5:  2
```

To 6:

```
1 -[4]-> 3 -[1]-> 4 -[11]-> 6
Total weight of the shortest path from 1 to 6:  16
```

To 7:

```
1 -[2]-> 5 -[5]-> 7
Total weight of the shortest path from 1 to 7:  7
```

To 8:

```
1 -[2]-> 5 -[5]-> 7 -[1]-> 8
Total weight of the shortest path from 1 to 8:  8
```

This verifies the above results.

Dijkstra's algorithm can be useful in many different applications. For example, an airline network might represent airports as nodes in a graph, and flight routes as 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.

Running Instructions

1. Open a Unix terminal and change the current directory to that which contains the source files by typing `cd PATH_TO_SOURCE_FILES`
2. Type `make` to build the program.
3. Run the program by typing `./graph`.
4. The program will ask for a file in which to load the graph. Afterwards, the user is presented with various options to be performed on the graph.

Data Structures Used

- Graph
- Binary Heap
- Minimum Priority Queue
- Vector

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)
```