CSCE 221 Cover Page
Programming Assignment #6
Due April 30 by midnight to CSNet

First Name: Leilani Last Name: Horlander-Cruz UIN: 523008771
User Name: leilanihc112 E-mail address: leilanihc112@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 System Office website: http://aggiehonor.tamu.edu/

Type of sources				
People				
Web pages (provide URL)	http://piazza.com	http://cplusplus.com		
Printed material				
Other Sources	Lecture slides from class			

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	Leilani Horlander-Cruz			Date	4-30-17

Programming Assignment #6

Due **April 30** *to submit to CSNet*

Data Structures

The undirected graph data structure is used throughout the program. Graphs contain vertices and edges. Vertices are connected to other vertices by edges, and these connections create a graph.

Vectors are implemented as dynamic arrays; Just as regular arrays, vector containers have their elements stores in contiguous storage locations, which means that their elements can be accessed using iterators and offsets on regular pointers to elements. Unlike regular arrays, however, storage in vectors is handled automatically, allowing it to be expanded and contracted as needed. The main vectors used in the program were an adjacency list, which was a vector of edge lists; a vertices vector, which contained all of the vertices in the graph; a group 1 vector, which contained all vertices in group 1 in Part 2 of the assignment; a group 2 vector, which contained all vertices in group 2 in Part 2 of the assignment; a visited vector, which contained all vertices visited during Part 3 of the assignment to find the shortest path; and a parent vector, which contained all parents of the vertices visited during Part 3 of the assignment.

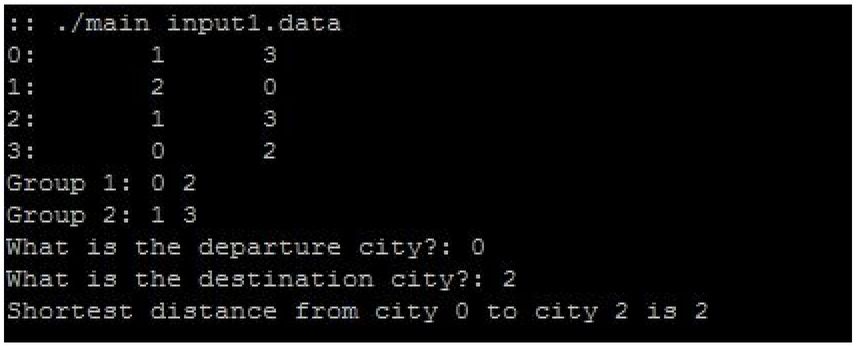
Lists are abstract data types that can be used to store multiple pieces of information at once. The lists used in the program were an edge list, which contained a list of edges in the graph, and a queue implemented as a list used in Part 3 of the assignment.

Algorithms

In Part 2, 0 was used as a source vertex. There was a vector that contained all the labels of the vertices in the graph called tempLabels, in which all visited vertices were removed so that only unvisited vertices remained and would be searched. Two group vectors were made, the first initially containing the source. The source was then explored, and any adjacent vertices connected to it were added to the group 2 vector. The same happened for the group 2 vector; Each vertex was explored, and its adjacent vertices were added to the opposite group vector, and those vertices were removed from the tempLabels vector, essentially marking them as "visited." After the tempLabels vector was empty, meaning all vertices had been visited and added to a group, the groups were added together and compared to the original vector containing the vertices in the graph to see if all vertices truly were in a group. If they were, then the vertices in each group were checked. If any vertices in the groups connected to each other, grouping was concluded to be not possible for the graph. Otherwise, grouping was possible, and the groups were displayed.

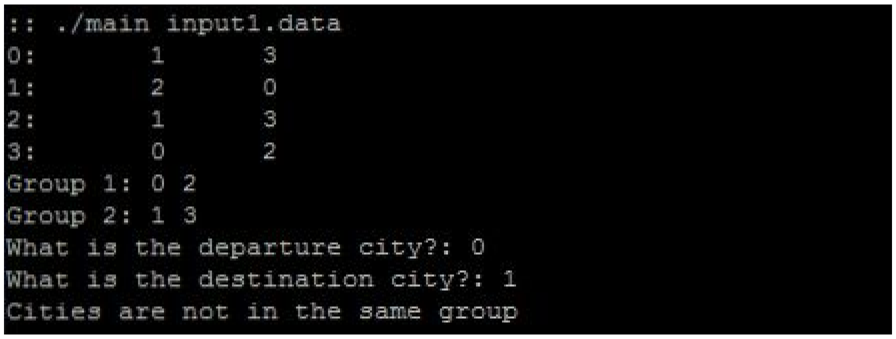
In Part 3, the Breadth-First Search algorithm is used to find the shortest path from the source node to the end node. In order to do this, each node a distance of d is visited before any nodes a distance of d+1 are visited. These nodes are all enqueued as they are visited, and then they are dequeued one at time. Each dequeued node is explored and its adjacent nodes are enqueued and this continues until all nodes have been explored. This started at the source vertex occurred until the inputted end vertex was enqueued, or visited. During this process, the parents of each vertex were saved. After this, the vertices were searched based on their parents, backtracking until reaching the source vertex. Distance was incremented each time a vertex was visited during this process.

Testing



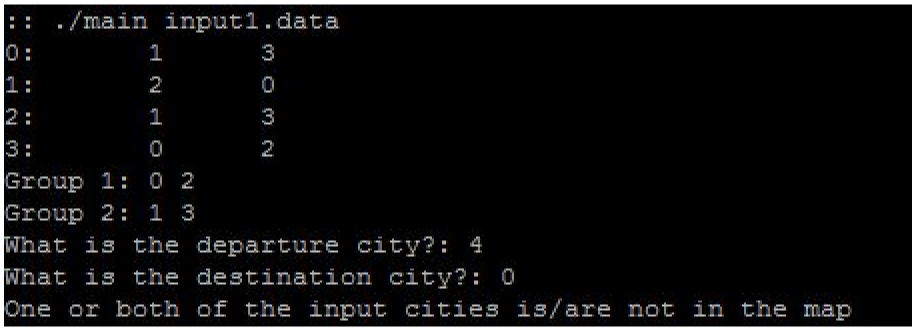
```
:: ./main input1.data
0:       1      3
1:       2      0
2:       1      3
3:       0      2
Group 1: 0 2
Group 2: 1 3
What is the departure city?: 0
What is the destination city?: 2
Shortest distance from city 0 to city 2 is 2
```

With input1.data, testing was done to see if the program output the correct adjacent vertices to each vertex, to see if the program recognized that grouping was possible, that the groups were correct, and then to see if the distance between two vertices in the same group was correct.



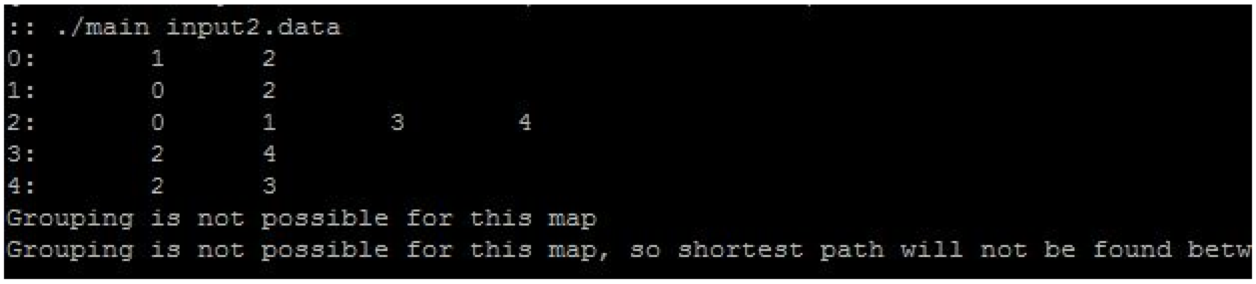
```
:: ./main input1.data
0:       1      3
1:       2      0
2:       1      3
3:       0      2
Group 1: 0 2
Group 2: 1 3
What is the departure city?: 0
What is the destination city?: 1
Cities are not in the same group
```

Then, two vertices in different groups were input to see if the program would recognize that the vertices were not in the same group.



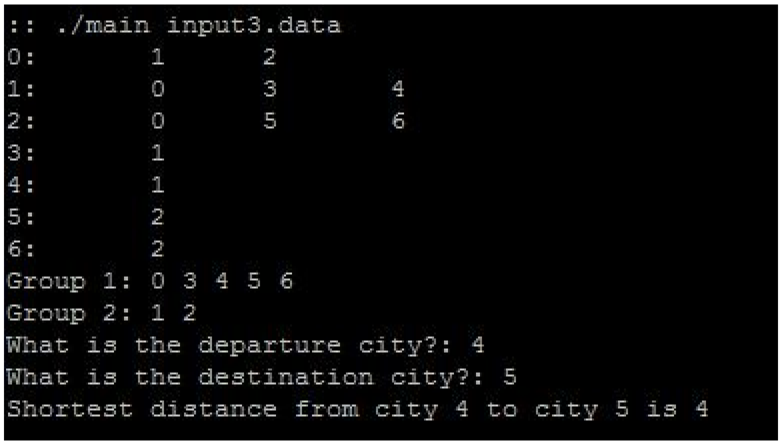
```
:: ./main input1.data
0:       1      3
1:       2      0
2:       1      3
3:       0      2
Group 1: 0 2
Group 2: 1 3
What is the departure city?: 4
What is the destination city?: 0
One or both of the input cities is/are not in the map
```

Then, one vertex that was not in the graph was input to see if the program would recognize that the vertex did not exist in the graph.



```
:: ./main input2.data
0:       1      2
1:       0      2
2:       0      1       3       4
3:       2      4
4:       2      3
Grouping is not possible for this map
Grouping is not possible for this map, so shortest path will not be found betw
```

Then, input2.data was tested to see if the program would recognize that grouping was not possible for this graph.



```
:: ./main input3.data
0:       1      2
1:       0      3       4
2:       0      5       6
3:       1
4:       1
5:       2
6:       2
Group 1: 0 3 4 5 6
Group 2: 1 2
What is the departure city?: 4
What is the destination city?: 5
Shortest distance from city 4 to city 5 is 4
```

Finally, testing was done with input3.data to see if the program output the correct adjacent vertices to each vertex, to see if the program recognized that grouping was possible, that the groups were correct, and then to see if the distance between two vertices in the same group was correct.