The Programming Assignment Report Instructions
CSCE 221

1. The description of an assignment problem.

   The purpose of this assignment is to be able to take an input of cities with info on where each city has a flight to another city. From this information, using a graph data structure and BFS, the shortest path from one city to another is determined.

2. The description of data structures and algorithms used to solve the problem.

   (a) Provide definitions of data structures by using Abstract Data Types (ADTs)

   - **Linked List**: a sequence of nodes where each node contains a pointer to the next node in the sequence. Since it is templated, a variety of data types can be passed in.
   - **Que:** a linear data structure where operations are performed based on first-in-first-out principle. Since it is templated, a variety of data types can be passed in.
   - **String:** a container that holds an array of characters
   - **Vector:** a container that can hold a variety of data types. Vector has several built in functions. Also provides bounds error checking.
   - **Graph represented by an adjacency list:** in this assignment the graph was implemented as a vector of linked list of edges. them.

   (b) Write about the ADTs implementation in C++.

   - **Linked List**: in this assignment the C++ standard library implementation of a linked list was used. It was used in the adjacency list.
   - **Que:** in this assignment the C++ standard library implementation of a que was used. It was used in implementing the BFS.
   - **String:** in this assignment the C++ standard library implementation of a string was used
   - **Vector:** in this assignment the C++ standard library implementation of a vector was used. Int was used for the ajacency list, container to hold vertices, and container for holding the edges.
   - **Graph represented by an adjacency list:** this was used to arrange all the cities and their connections.

   (c) Describe algorithms used to solve the problem.

   - **BFS/shortest distance:** A BFS traversal of a graph G visits all the vertices and edges of G, determines whether G is connected, computes the connected components of G, then finds the shortest path once the endpoint is found. This is done by first pushing the starting vertex to the queue. Also push this vertex to the visited vector with a NULL parent. While the queue is not empty, pop the queue. if this popped queue is the vertex being searched for, go to the vector and traverse up from the found vertex to its parents until the starting vertex is found, that is the shortest path. Else, go to the next level of the graph and enqueue each vertex in that level as well as pushing back each vertex and its parent to the visited vector.

- **BuildGraph:** read in the first line to determine the number of vertices and number of edges. Iterate through the remaining lines to determine what vertices are connected to v_{i-1}.
- **Determine Partitioning:** this algorithm purpose is to determine the possibility of partition the cities into two groups such that there is no edge between two vertices in each group. Iterate through each vertex. Check if the given vertex is not in group a or group b, if so add to group a. Check if vertex is in group a but not group b,

(d) Analyze the algorithms according to assignment requirements.

- **BFS/shortest distance:** $O(V+E)$
- **BuildGraph:** $O(V+E)$
- **Determine Partitioning:** $O(VE)$

3. A user guide description how to navigate your program with the instructions how to:

   (a) compile the program: specify the directory and file names, etc.
   'g++ *.cpp -o run.out' in root directory I submitted

   (b) run the program: specify the name of an executable file.
   ./run.out {file-name}

4. Specifications and description of input and output formats and files

   (a) The type of files: keyboard, text files, etc (if applicable).

   Input file containing the cities and the connections between the cities

   (b) A file input format: when a program requires a sequence of input items, specify the number of items per line or a line termination. Provide a sample of a required input format.

   Example input,
   ```
   7 6
   1 2 -1
   0 3 4 -1
   0 5 6 -1
   1 -1
   1 -1
   2 -1
   2 -1
   ```

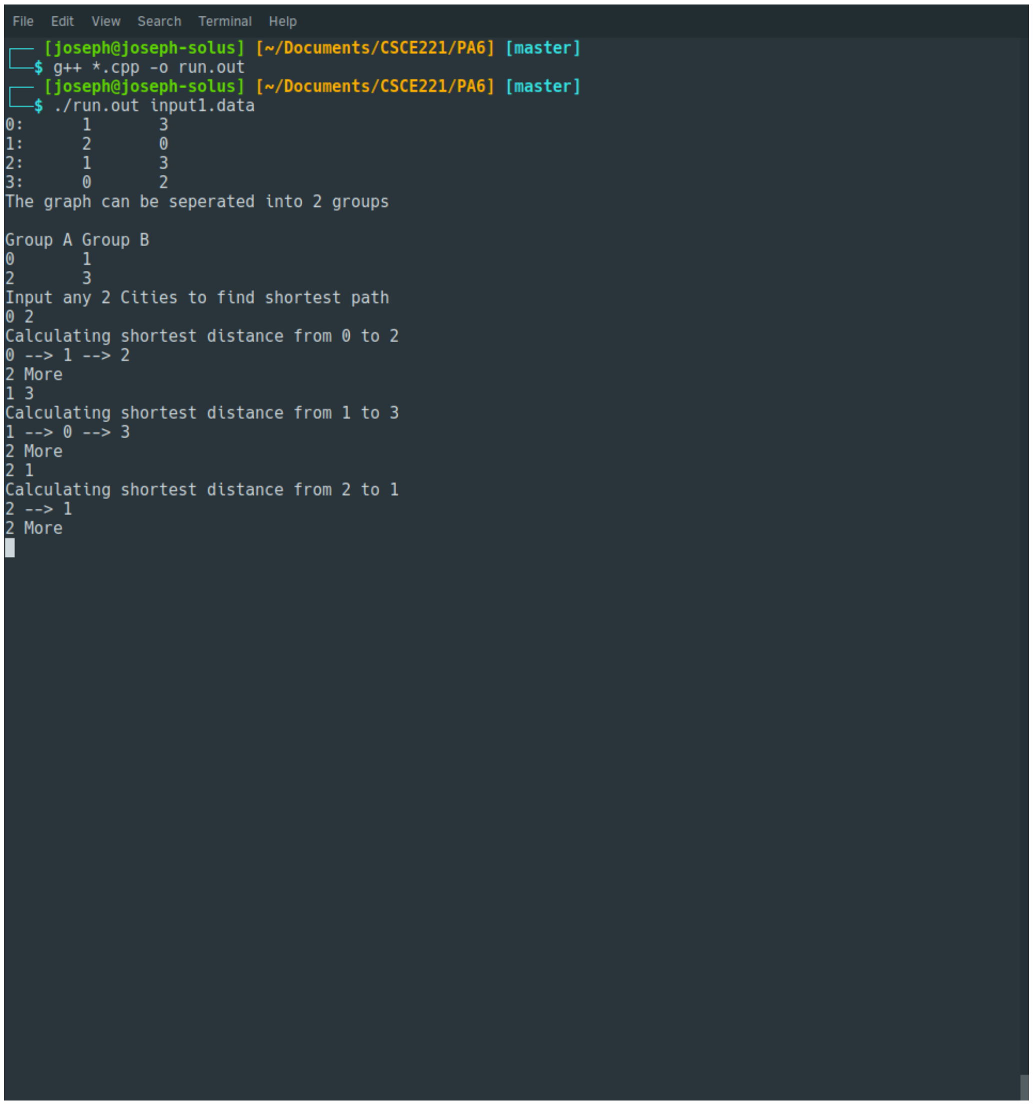
   (c) Discuss possible cases when your program could crash because of incorrect input (a wrong file name, strings instead of a number, or such cases when the program expects 10 items to read and it finds only 9.)

   The program will crash if the input first line does not specify the correct number of vertixes and edges.

5. Test your program for correctness using valid, invalid, and random inputs (e.g., insertion of an item at the beginning, at the end, or at a random place into a sorted vector). Include evidence of your testing, such as an output file or screen shots with an input and the corresponding output.

   With the following input,
   ```
   4 4
   1 3 -1
   ```

2 0 -1
1 3 -1
0 2 -1
Results in the following output

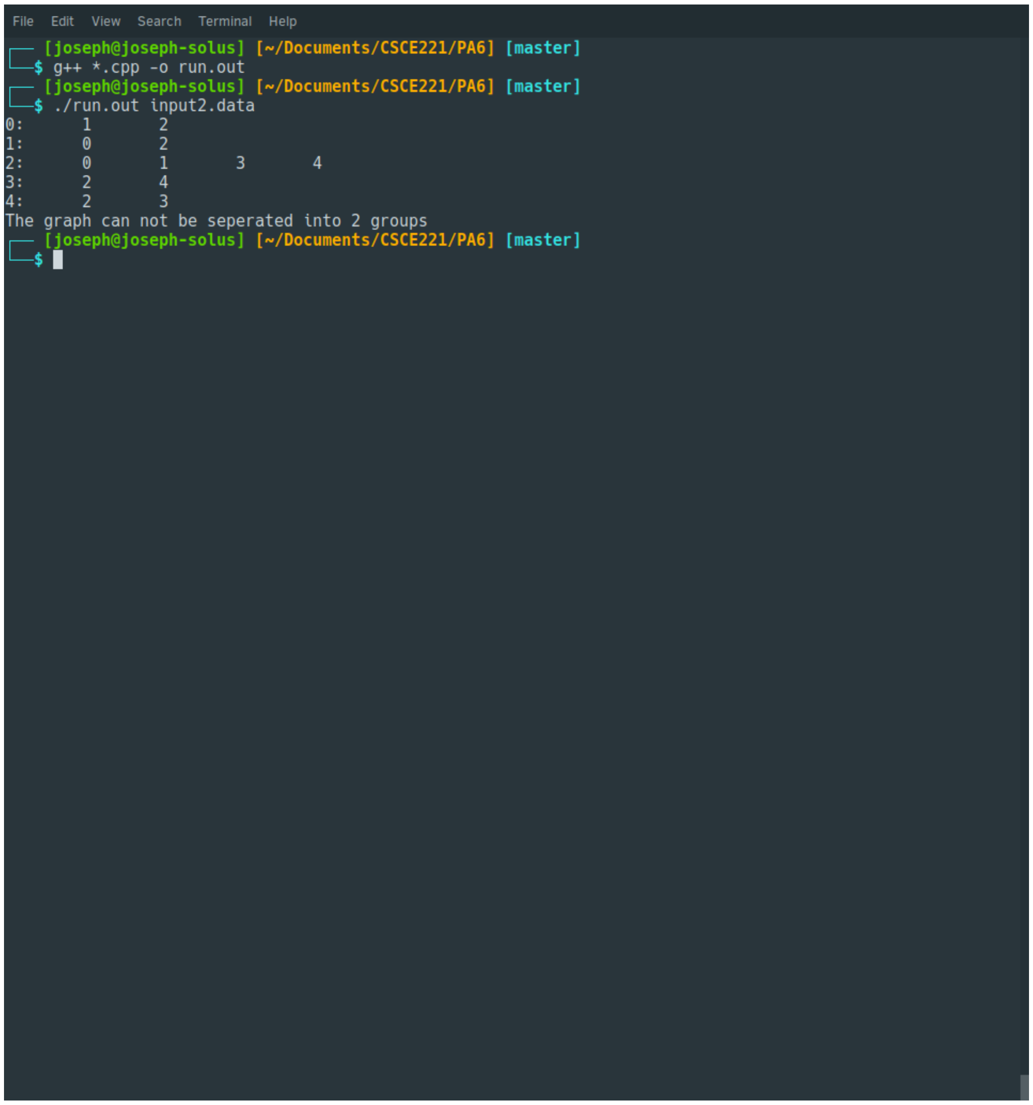


```
┌─ [joseph@joseph-solus] [~/Documents/CSCE221/PA6] [master]
└─$ g++ *.cpp -o run.out
┌─ [joseph@joseph-solus] [~/Documents/CSCE221/PA6] [master]
└─$ ./run.out input1.data
0:      1       3
1:      2       0
2:      1       3
3:      0       2
The graph can be seperated into 2 groups

Group A Group B
0       1
2       3
Input any 2 Cities to find shortest path
0 2
Calculating shortest distance from 0 to 2
0 --> 1 --> 2
2 More
1 3
Calculating shortest distance from 1 to 3
1 --> 0 --> 3
2 More
2 1
Calculating shortest distance from 2 to 1
2 --> 1
2 More
```

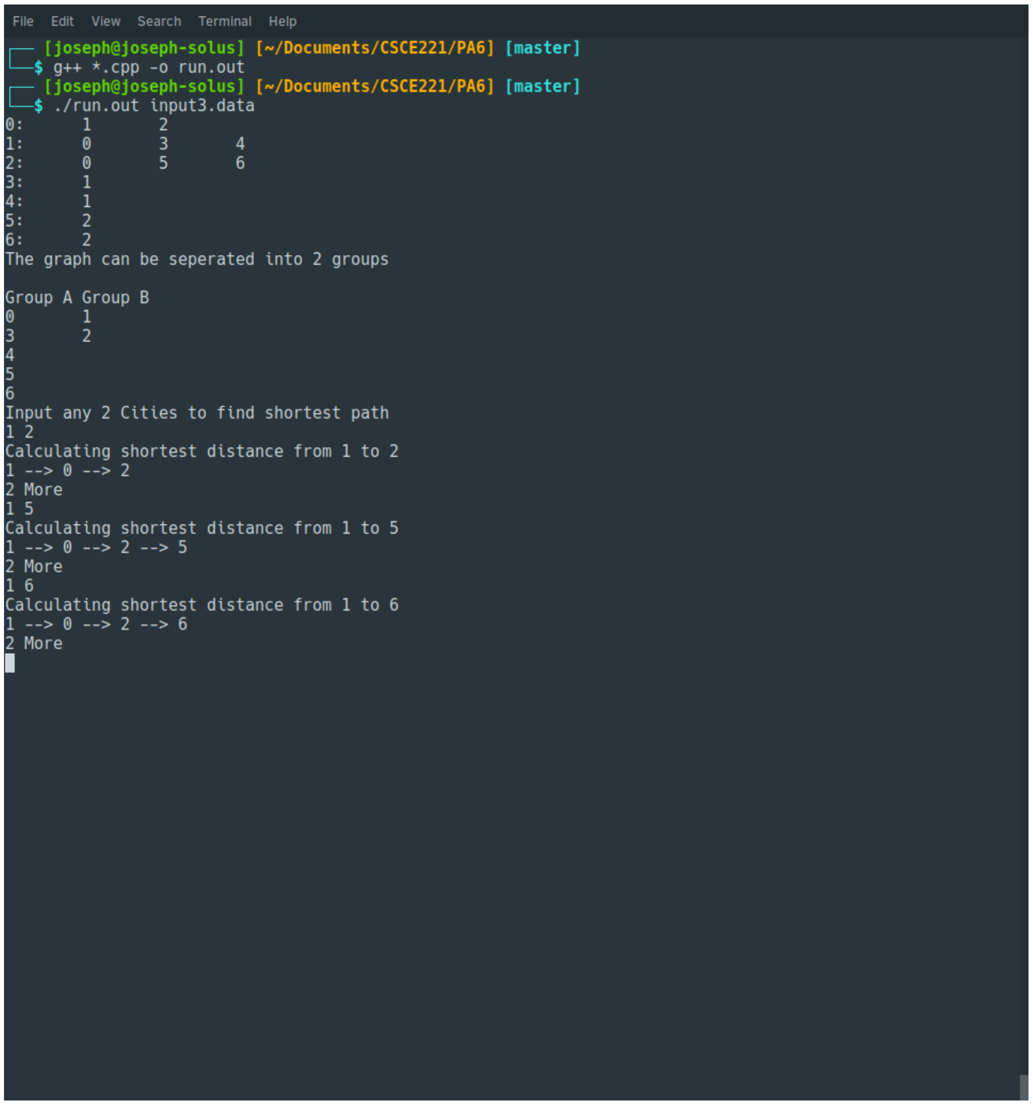
With the following input,
5 6
1 2 -1
0 2 -1
0 1 3 4 -1
2 4 -1
2 3 -1
Results in the following output

With the following input,
7 6
1 2 -1
0 3 4 -1



```
┌── [joseph@joseph-solus] [~/Documents/CSCE221/PA6] [master]
└─$ g++ *.cpp -o run.out
┌── [joseph@joseph-solus] [~/Documents/CSCE221/PA6] [master]
└─$ ./run.out input2.data
0:      1       2
1:      0       2
2:      0       1       3       4
3:      2       4
4:      2       3
The graph can not be seperated into 2 groups
┌── [joseph@joseph-solus] [~/Documents/CSCE221/PA6] [master]
└─$ 
```

0 5 6 -1
1 -1
1 -1
2 -1
2 -1
Results in the following output



```
┌── [joseph@joseph-solus] [~/Documents/CSCE221/PA6] [master]
└──$ g++ *.cpp -o run.out
┌── [joseph@joseph-solus] [~/Documents/CSCE221/PA6] [master]
└──$ ./run.out input3.data
0:      1       2
1:      0       3       4
2:      0       5       6
3:      1
4:      1
5:      2
6:      2
The graph can be seperated into 2 groups

Group A Group B
0       1
3       2
4
5
6
Input any 2 Cities to find shortest path
1 2
Calculating shortest distance from 1 to 2
1 --> 0 --> 2
2 More
1 5
Calculating shortest distance from 1 to 5
1 --> 0 --> 2 --> 5
2 More
1 6
Calculating shortest distance from 1 to 6
1 --> 0 --> 2 --> 6
2 More
```