CSCE 221 - Programming Assignment 6

Due: April 30, 2017 at 11:59 pm

Program Description & Purpose

The purpose of this assignment was to further our knowledge of graph theory by implementing a graph data structure and algorithms so that a shortest path in the graph could be found. This program contains the implementation of a graph and an auxiliary queue data structure so that an input file containing a undirected graph's information can be passed through, parsed, stored, and utilized, in order to construct a graph and compute the shortest path by using Breadth-First Search on a starting and ending user-specified vertex. There are essentially 7 files, a main.cpp containing the driver program, the Graph.h and Graph.cpp files containing the graph data structure, the Vertex.h and Vertex.cpp containing the Vertex class, the Edge.h and Edge.cpp which contains the Edge class, and of course, the three instructor-given graph input files used to test the program. The program can be compiled and executed (directions shown down below in the **"How to Compile and Run"** section) in order to execute the graph building process, checking for the construction of a bipartite graph, and computing the shortest path based on user input starting and ending vertices.

Data Structure & Algorithms Description

The program essentially passes in an input file containing the graph's information to a stream so that the information can be parsed, stored, and utilized in order to construct a graph. Once the graph is constructed, a check is made to see if the constructed graph is a bipartite graph by checking if the graph's vertices can be divided into two disjoint sets. After this check, if the graph is a bipartite graph, the program then computes the shortest path based off of user-input starting and ending vertices, if it is not a bipartite graph, the shortest path is not calculated and the program terminates. If the user-input starting and ending vertices are valid (not out of range or equal to each other), and they are in the same disjoint group, the shortest path is then calculated by utilizing the Breadth-First-Search algorithm which uses an auxiliary queue data structure (STL) along with a back-tracking algorithm in order to find the shortest path by starting at the start vertex, using the BFS algorithm to arrive at the end vertex, assigning the respective parent to each currently visited vertex and assigning the starting vertex with a value of "-1", and then backtracking through the visited nodes starting from the ending vertex by their parent nodes, arriving back at the starting node and have thus computed the shortest path by only visiting parent nodes.

I/O Description

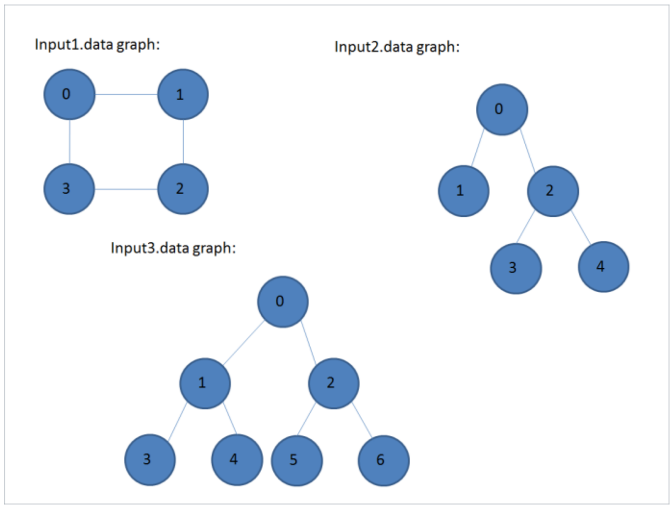
The input files utilized by the program are input1.data, input2.data, and input3.data which contain the graph's information, i.e., number of vertices; number of edges, each vertex's neighboring vertices, etc. The program parses the data files in order to extract the required information needed in order to construct the graph, determine if it is a bipartite graph, and to compute the shortest path based on user-specified input of starting and ending vertices.

C++/STL Features

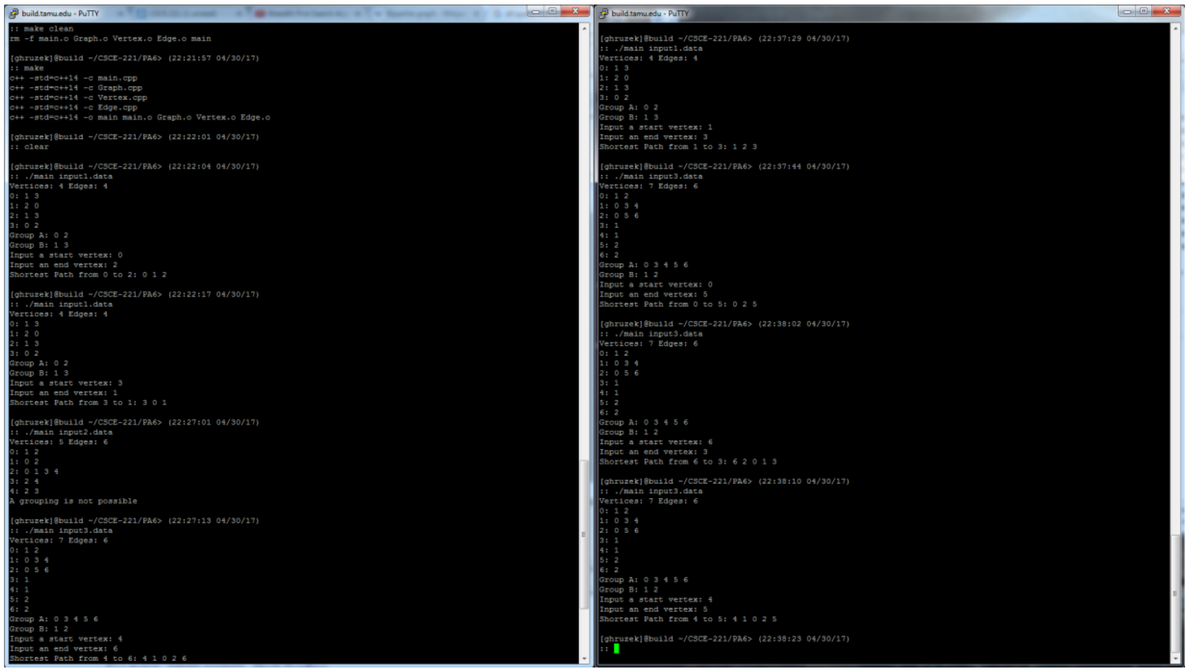
The primary C++ native features used in the program were the STL queue, vector, and the I/O stream in order to store, parse, and access various data regarding input data files, parsing said input data, and utilizing said data.

Testing

Testing of the program relies mainly on the 3 instructor-provided input files as testing cases. Extensive testing has been implemented in order to test the reliability of the program via input1.data, input2.data, and input3.data, below is a visualization of the 3 graphs generated by the 3 input files:



Here, you can see that input1.data and input3.data graphs are bipartite, and input2 is not. Testing for the generation of shortest path in a graph based on user-input is also shown below:



```
:: make clean
rm -f main.o Graph.o Vertex.o Edge.o main

[ghruzek]@build ~/CSCE-221/PA6> (22:21:57 04/30/17)
:: make
c++ -std=c++14 -c main.cpp
c++ -std=c++14 -c Graph.cpp
c++ -std=c++14 -c Vertex.cpp
c++ -std=c++14 -c Edge.cpp
c++ -std=c++14 -o main main.o Graph.o Vertex.o Edge.o

[ghruzek]@build ~/CSCE-221/PA6> (22:22:01 04/30/17)
:: clear

[ghruzek]@build ~/CSCE-221/PA6> (22:22:04 04/30/17)
:: ./main input1.data
Vertices: 4 Edges: 4
0: 1 3
1: 2 0
2: 1 3
3: 0 2
Group A: 0 2
Group B: 1 3
Input a start vertex: 0
Input an end vertex: 2
Shortest Path from 0 to 2: 0 1 2

[ghruzek]@build ~/CSCE-221/PA6> (22:22:17 04/30/17)
:: ./main input1.data
Vertices: 4 Edges: 4
0: 1 3
1: 2 0
2: 1 3
3: 0 2
Group A: 0 2
Group B: 1 3
Input a start vertex: 3
Input an end vertex: 1
Shortest Path from 3 to 1: 3 0 1

[ghruzek]@build ~/CSCE-221/PA6> (22:27:01 04/30/17)
:: ./main input2.data
Vertices: 5 Edges: 6
0: 1 2
1: 0 2
2: 0 1 3 4
3: 2 4
4: 2 3
A grouping is not possible

[ghruzek]@build ~/CSCE-221/PA6> (22:27:13 04/30/17)
:: ./main input3.data
Vertices: 7 Edges: 6
0: 1 2
1: 0 3 4
2: 0 5 6
3: 1
4: 1
5: 2
6: 2
Group A: 0 3 4 5 6
Group B: 1 2
Input a start vertex: 4
Input an end vertex: 6
Shortest Path from 4 to 6: 4 1 0 2 6
```

```
[ghruzek]@build ~/CSCE-221/PA6> (22:37:29 04/30/17)
:: ./main input1.data
Vertices: 4 Edges: 4
0: 1 3
1: 2 0
2: 1 3
3: 0 2
Group A: 0 2
Group B: 1 3
Input a start vertex: 1
Input an end vertex: 3
Shortest Path from 1 to 3: 1 2 3

[ghruzek]@build ~/CSCE-221/PA6> (22:37:44 04/30/17)
:: ./main input3.data
Vertices: 7 Edges: 6
0: 1 2
1: 0 3 4
2: 0 5 6
3: 1
4: 1
5: 2
6: 2
Group A: 0 3 4 5 6
Group B: 1 2
Input a start vertex: 0
Input an end vertex: 5
Shortest Path from 0 to 5: 0 2 5

[ghruzek]@build ~/CSCE-221/PA6> (22:38:02 04/30/17)
:: ./main input3.data
Vertices: 7 Edges: 6
0: 1 2
1: 0 3 4
2: 0 5 6
3: 1
4: 1
5: 2
6: 2
Group A: 0 3 4 5 6
Group B: 1 2
Input a start vertex: 6
Input an end vertex: 3
Shortest Path from 6 to 3: 6 2 0 1 3

[ghruzek]@build ~/CSCE-221/PA6> (22:38:10 04/30/17)
:: ./main input3.data
Vertices: 7 Edges: 6
0: 1 2
1: 0 3 4
2: 0 5 6
3: 1
4: 1
5: 2
6: 2
Group A: 0 3 4 5 6
Group B: 1 2
Input a start vertex: 4
Input an end vertex: 5
Shortest Path from 4 to 5: 4 1 0 2 5

[ghruzek]@build ~/CSCE-221/PA6> (22:38:23 04/30/17)
::
```

How to Compile and Run

1. In a UNIX terminal, navigate to the respective source directory **PA6/** using the **cd** command.
2. Once in the **PA6/** directory type **make** to compile the respective program.
3. Run the executable for the program by typing **./main -"inputfile"** for the program ("input file" is the name of your data file you wish to use).
4. Compiled on build.tamu.edu host

Conclusion

A greater understanding of the Graph and Queue Data structures, Trees, the Breadth-First-Search algorithm, I/O streams, and Bipartite graphs was achieved by implementing the graph data structure along with implementing the BFS & a back-tracking algorithm in order to construct a graph, find disjoint groups of a bipartite graph, and computing the shortest path based on user-input data files.