



CSCI 2270 – Data Structures

Instructors: Christopher Godley, Maciej Zagrodzki

Assignment 8 - Graph

OBJECTIVES

1. Applications of BFS

Overview

In this assignment, you will apply BFS for finding connected cities in a graph and check whether a graph is Bipartite graph or not.

Graph Class

Your code should implement graph traversal for cities. A header file that lays out this graph can be found in [Graph.hpp](#) on Moodle. *As usual, do not modify the header file. You may implement helper functions in your .cpp file if you want as long as you don't add those functions to the Graph class.*

Your graph will utilize the following struct:

```
struct vertex;

struct adjVertex{
    vertex *v;
};

struct vertex{
    vertex() {
        this->visited = false;
        this->color = "";
        this->distance = 0;
    }
    string name;
    bool visited;
    string color;
    int distance;
    vector<adjVertex> adj;
};
```

void addVertex(string name);

→ Add new vertex 'name' to the graph.



CSCI 2270 – Data Structures

Instructors: Christopher Godley, Maciej Zagrodzki

void addEdge(string v1, string v2);

→ Make a connection between v1 and v2.

void displayEdges();

→ Display the all the edges in the graph.

Format for printing:

If we create a graph with the following structure

```
graph.addVertex("Boulder");  
graph.addVertex("Denver");  
graph.addVertex("Las Vegas");  
  
graph.addEdge("Boulder", "Denver");  
graph.addEdge("Las Vegas", "Denver");
```

We print the edges in the following manner.

```
Boulder --> Denver  
Denver --> Boulder Las Vegas  
Las Vegas --> Denver
```

The order of vertices printed is the same as the order in which they were added to the graph. Similarly, the order of vertices to the right of "-->" sign is the same as the order in which the corresponding edge was added to the graph.

void breadthFirstTraverse(string sourceVertex);

→ Breadth first traversal from sourceVertex. Format for printing:

```
// for the source vertex in the graph  
cout<< "Starting vertex (root): " << vStart->name << "->";  
// for other vertex traversed from source vertex with distance  
cout << n->adj[x].v->name << "(" << n->adj[x].v->distance << ")" << " " << " ";
```

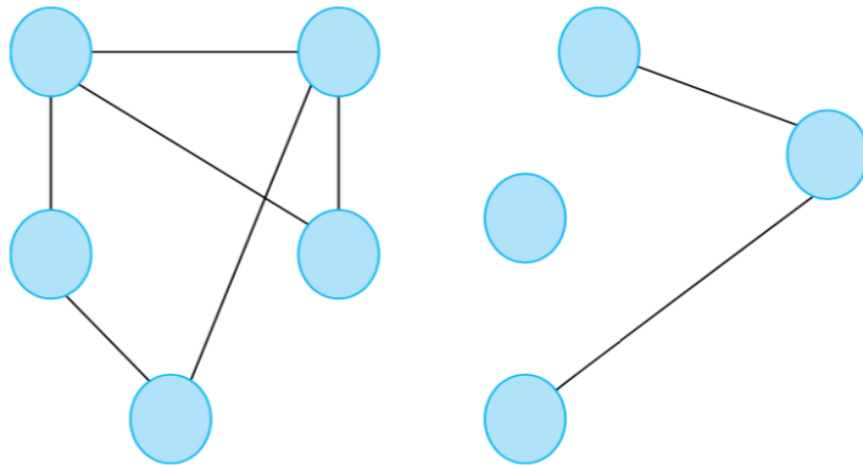
int getConnectedComponents();



CSCI 2270 – Data Structures

Instructors: Christopher Godley, Maciej Zagrodzki

- This method will provide the number of connected components in the graph i.e. the number of distinct subgraphs where no edges exist which connect vertices in two distinct subgraphs. In the following graph, the number of connected components is 3.



bool checkBipartite();

- This method will check whether a graph is bipartite or not. Will return true if the graph is Bipartite, otherwise false.

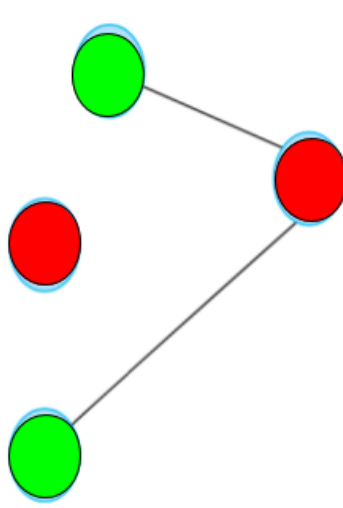
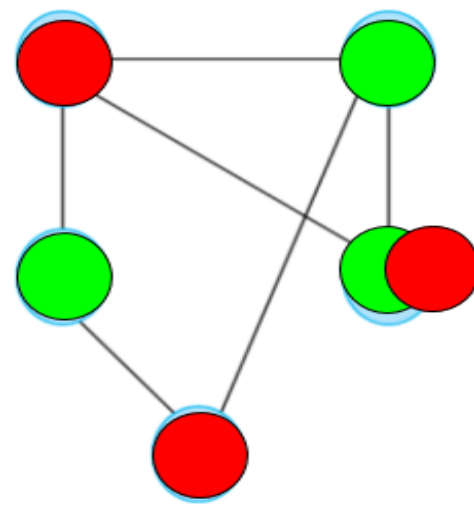
A [Bipartite graph](#) is a graph whose vertices can be divided into two sets and there will not be any edge among the vertices in the same set. We can say a graph is bipartite, if we can color the graph with two colors where vertices in the same set will be same colored.

In other words, in a Bipartite graph, no two adjacent vertices will have the same color.



CSCI 2270 – Data Structures

Instructors: Christopher Godley, Maciej Zagrodzki

 <p>In this graph, we color the vertices with two colors, so, it's a bipartite graph.</p>	 <p>In this graph, if we try to color the vertices with two colors, we can't do that. As you can see, one vertex has one neighbor with the color red and the other with green. Therefore, for the graph to be bipartite, that node would have to be two colors at the same time. So, it's not a bipartite graph.</p>
--	--

Please note that once you are done with your assignment on code runner you need to click on **'finish attempt'** and the **'submit all and finish'**. If you don't do this, you will not get graded."