CSCI-311 Assignment 12 BFS, DFS

Download Lab13.tar and untar it using the command: tar -xvf Lab13.tar

You are provided the following files: *timestamp.h, graph.h, graph.cpp* and *main.cpp*.

In this assignment, you will work with a directed graph represented as Adjacency Lists and implemented using **vector<vector< *edge* > > *Adj*.** Instead of integers, each Adjacency List holds ***edge***(s), each ***edge*** has int *neighbor* and int *w* such that Adj[u][j] is an ***edge***, and **Adj[u][j].neighbor** returns a vertex *v* of the edge (u, v) in the graph. The integer *w* stands for *weight* and will be used to solve problems. For example, you can set *w* to -1 to mark the corresponding edge (u, v); here is how to access it: **Adj[u][j].w** = -1.

```
struct edge{
	int neighbor; // adjacent vertex
	int w; //weight, keeps auxiliary information
	edge(){
		neighbor = 0;
		w = 0;
	};
	edge(int i, int j){
		neighbor = i;
		w = j;
	};
};
```

You need to solve four problems for this assignment. Complete one problem at a time and test it using the provided test files.

Inside *main()* function, a graph is read from the standard input. The constructor and functions *addEdge, bfs, DFS, dfsVisit* are provided.

Problem 1.

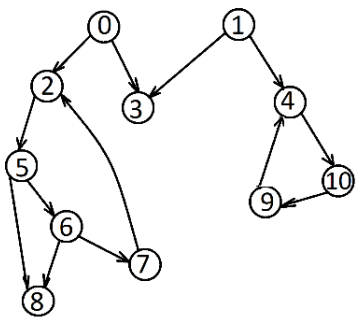
YouTube video: Same Cycle (25 min) by Elena.

Given a graph G and two integers, vertices *s* and *r*, write a member function(s) that returns *true* if the given vertices *s* and *r* lie on the same cycle. If these vertices do not lie on the same cycle, then your method should return *false*.

You must write the following public member function with the exact header as shown below:

bool *sameCycle*(int s, int r);

You may use more functions if needed to solve this problem, but they must be called from *sameCycle* method.



Example: Given a directed graph on Figure to the left and two vertices 5 and 7, *sameCycle* will return *true.*

With the same graph and two vertices 9 and 6, *sameCycle* will return *false.*

With the same graph and two vertices 1 and 10, *sameCycle* will return *false.*

To test your *sameCycle,* use test files t01.in and t01.out:
./run < tests/t01.in > t01.my
diff t01.my tests/t01.out

Problem 2.

YouTube video: Longest Cycle (12 min) by Elena.

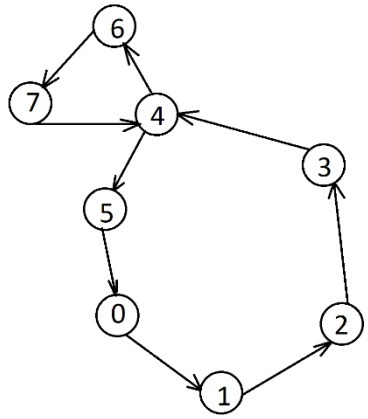
Given a graph G and a vertex *s*, write a member function that returns an integer L, the length of the longest simple cycle, on which *s* lies. If *s* does not lie on any cycle, your method will return 0. A cycle is *simple* if each vertex occurs in the cycle only once. The length of a cycle is measured in the number of edges on that cycle.

DFS might not discover all simple cycles, on which *s* lies, you only need to find the longest cycle that can be discovered by running dfsVisit on *s* (i.e. whatever cycle may be discovered, find it's length, and choose the max length out of all such cycles).

You must write a public member function with the exact header as shown below:

int longestCycle(int s);

You may write additional member functions, but call them from *longestCycle* function.



Example: Given a graph on Figure to the left and a vertex 4, your function will return 6. There are two simple cycles: 4, 6, 7 (the length is 3), and 4, 5, 0, 1, 2, 3 (the length of this cycle is 6). The longest cycle has length 6.

Notice that a cycle 4, 5, 0, 1, 2, 3, 4, 6, 7 is not *simple* because 4 occurs on this cycle more than once. So, we don't consider the length of this cycle.

To test your *longestCycle,* use test files t02, t03, t04 and t05:

./run < tests/t02.in > t02.my
diff t02.my tests/t02.out

./run < tests/t03.in > t03.my
diff t03.my tests/t03.out

./run < tests/t04.in > t04.my
diff t04.my tests/t04.out

./run < tests/t05.in > t05.my
diff t05.my tests/t05.out

Problem 3.

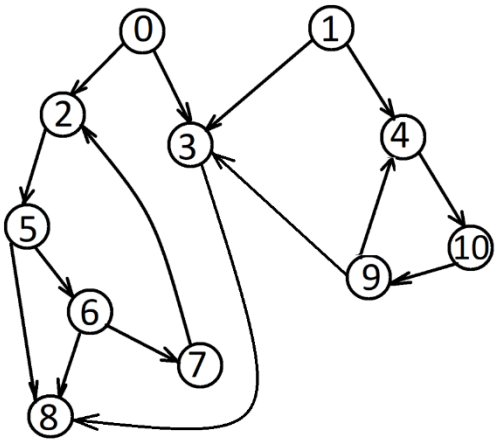
YouTube video: Two Paths (12 min) by Elena.

Given a directed graph *G* and two vertices *s* and *r* in *G*, write a member function that returns *true* if there exist at least two distinct paths from *s* to *r.* Two paths are called *distinct* if they do not share any edges.

You must write a public member function with the exact header as shown below:

bool *twoPaths*(int s, int r);

You may write additional member functions, but call them from *twoPaths* function.



Example: Given the graph to the left and two vertices 0 and 8, your program will return *true* because there are two paths from 0 to 8, namely, 0, 3, 8 and 0, 2, 5, 8. These paths are distinct because they do not share any edges.

Given the same graph and two vertices 1 and 8, your program will return *false.* There are two paths from 1 to 8: the first is 1, 3, 8, and the second is 1, 4, 10, 9, 3, 8, but these paths are not distinct because they share the same edge (3, 8).

To test your *twoPaths,* use test files t06:
./run < tests/t06.in > t06.my
diff t06.my tests/t06.out

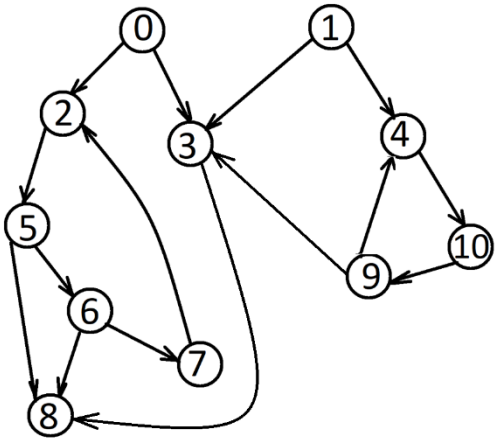
Problem 4.

Given a directed graph, and three vertices *s*, *r* and *q*, write a member function that returns true if *q* is on the shortest path from *s* to *r.*

You must write a public member function with the exact header as shown below:

bool *isOnPath*(int s, int r, int q);

You may write additional member functions, but call them from *isOnPath* function.



Example: Given the graph on Figure to the left, and three vertices 0, 8 and 5, your program will return *false.* There are two paths from 0 to 8: the first is 0, 3, 8 and the second is 0, 2, 5, 8. The shortest path 0, 3, 8 has length 2 (the length is measured in the number of edges on the path), and the vertex 5 is not on the shortest path.

Given the same graph and three vertices 1, 3 and 3, your program will return *true* because the shortest path from 1 to 3 has only one edge and the third given vertex 3 lies on this path.

Given the same graph and three vertices 0, 7 and 5, your program will return *true* because the shortest path from 0 to 7 is 0, 2, 5, 6, 7 and the vertex 5 is on this path.

To test your *isOnPath,* use test files t07:
./run < tests/t07.in > t07.my
diff t07.my tests/t07.out