Đã bắt đầu vào lúc	Thứ năm, 14 Tháng mười hai 2023, 2:37 PM
Tình trạng	Đã hoàn thành
Hoàn thành vào lúc	Thứ năm, 14 Tháng mười hai 2023, 8:50 PM
Thời gian thực hiện	6 giờ 13 phút
Điểm	7,00/7,00
Điểm	**10,00** của 10,00 (**100**%)

Câu hỏi 1

Chính xác

Điểm 1,00 của 1,00

Implement Breadth-first search

```
Adjacency *BFS(int v);
```

where Adjacency is a structure to store list of number.

```
#include <iostream>
#include <list>
using namespace std;

class Adjacency
{
private:
        list<int> adjList;
        int size;
public:
        Adjacency() {}
        Adjacency(int V) {}
        void push(int data)
        {
                adjList.push_back(data);
                size++;
        }
        void print()
        {
                for (auto const &i : adjList)
                        cout << " -> " << i;
        }
        void printArray()
        {
                for (auto const &i : adjList)
                        cout << i << " ";
        }
        int getSize() { return adjList.size(); }
        int getElement(int idx)
        {
                auto it = adjList.begin();
                advance(it, idx);
                return *it;
        }
};
```

And Graph is a structure to store a graph (see in your answer box)

For example:

Test	Result

int V = 6;
int visited = 0;
 	
Graph g(V);
Adjacency* arr = new Adjacency(V);
int edge[][2] = {{0,1},{0,2},{1,3},{1,4},{2,4},{3,4},{3,5},{4,5}};
 	
for(int i = 0; i < 8; i++)
{
  g.addEdge(edge[i][0], edge[i][1]);
}
 	
arr = g.BFS(visited);
arr->printArray();
delete arr;
	0 1 2 3 4 5

int V = 6;
int visited = 2;
 	
Graph g(V);
Adjacency* arr = new Adjacency(V);
int edge[][2] = {{0,1},{0,2},{1,3},{1,4},{2,4},{3,4},{3,5},{4,5}};
 	
for(int i = 0; i < 8; i++)
{
  g.addEdge(edge[i][0], edge[i][1]);
}
 	
arr = g.BFS(visited);
arr->printArray();
delete arr;
	2 0 4 1 3 5

Answer: (penalty regime: 0, 0, 5, 10, ... %)

Reset answer

```
class Graph
{
private:
    int V;
    Adjacency *adj;

public:
    Graph(int V)
    {
        this->V = V;
        adj = new Adjacency[V];
    }

    void addEdge(int v, int w)
    {
        adj[v].push(w);
        adj[w].push(v);
    }

    void printGraph()
    {
        for (int v = 0; v < V; ++v)
        {
            cout << "\nAdjacency list of vertex " << v << "\nhead ";
            adj[v].print();
        }
    }

    Adjacency *BFS(int v)
    {
        // v is a vertex we start BFS
```

```
32          Adjacency* traversedList = new Adjacency;
33          bool visited[V];
34          for(int i = 0; i < V; ++i) visited[i] = false;
35          list<int> queue;
36          visited[v] = true;
37          queue.push_back(v);
38          while(!queue.empty()) {
39              v = queue.front();
40              traversedList->push(v);
41              queue.pop_front();
42              for(int i = 0; i < int(adj[v].getSize()); ++i) {
43                  int adjacent = adj[v].getElement(i);
44                  if(!visited[adjacent]) {
45                      visited[adjacent] = true;
46                      queue.push_back(adjacent);
47                  }
48              }
49          }
```

	Test	Expected	Got	

✔	`int V = 6;`
	`int visited = 0;`
	
	`Graph g(V);`
	`Adjacency* arr = new Adjacency(V);`
	`int edge[][2] = {{0,1},{0,2},{1,3},{1,4},{2,4},{3,4},{3,5},{4,5}};`
	
	`for(int i = 0; i < 8; i++)`
	`{`
	`  g.addEdge(edge[i][0], edge[i][1]);`
	`}`
	
	`arr = g.BFS(visited);`
	`arr->printArray();`
	`delete arr;`
		0 1 2 3 4 5	0 1 2 3 4 5	✔

✔	`int V = 6;`
	`int visited = 2;`
	
	`Graph g(V);`
	`Adjacency* arr = new Adjacency(V);`
	`int edge[][2] = {{0,1},{0,2},{1,3},{1,4},{2,4},{3,4},{3,5},{4,5}};`
	
	`for(int i = 0; i < 8; i++)`
	`{`
	`  g.addEdge(edge[i][0], edge[i][1]);`
	`}`
	
	`arr = g.BFS(visited);`
	`arr->printArray();`
	`delete arr;`
		2 0 4 1 3 5	2 0 4 1 3 5	✔

	Test	Expected	Got	

✔	`int V = 8, visited = 5;`
	
	`Graph g(V);`
	`Adjacency *arr;`
	`int edge[][2] = {{0,1}, {0,2}, {0,3}, {0,4}, {1,2}, {2,5}, {2,6}, {4,6}, {6,7}};`
	`for(int i = 0; i < 9; i++)`
	`{`
	`\tg.addEdge(edge[i][0], edge[i][1]);`
	`}`
	
	`// g.printGraph();`
	`// cout << endl;`
	`arr = g.BFS(visited);`
	`arr->printArray();`
	`delete arr;`
		`5 2 0 1 6 3 4 7`	`5 2 0 1 6 3 4 7`	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 2

Chính xác

Điểm 1,00 của 1,00

Given a graph represented by an adjacency-list `edges`.

Request: Implement function:

```
int connectedComponents(vector<vector<int>>& edges);
```

Where `edges` is the adjacency-list representing the graph (this list has between 0 and 1000 lists). This function returns the number of connected components of the graph.

Example:

Given a adjacency-list: `[[1], [0, 2], [1], [4], [3], []]`

There are `3` connected components: `[0, 1, 2], [3, 4], [5]`

Note:

In this exercise, the libraries `iostream, string, cstring, climits, utility, vector, list, stack, queue, map, unordered_map, set, unordered_set, functional, algorithm` has been included and `namespace std` are used. You can write helper functions and classes. Importing other libraries is allowed, but not encouraged, and may result in unexpected errors.

For example:

Test	Result

`vector<vector<int>> graph {`
`{1},`
`{0, 2},`
`{1, 3},`
`{2},`
`{}`
`};`
`cout << connectedComponents(graph);`
	2

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
void dfs(int e, vector<vector<int>>& edges, vector<bool>& visited) {
    visited[e] = true;
    for(int i = 0; i < int(edges[e].size()); ++i) {
        int adjacent = edges[e][i];
        if(!visited[adjacent]) dfs(adjacent, edges, visited);
    }
}
int connectedComponents(vector<vector<int>>& edges) {
    // STUDENT ANSWER
    vector<bool> visited(edges.size(), false);
    int ComponentCount = 0;
    for(int i = 0; i < int(edges.size()); ++i) {
        if(visited[i] == false) {
            dfs(i, edges, visited);
            ++ComponentCount;
        }
    }
    return ComponentCount;
}
```

	Test	Expected	Got	

✔	vector<vector<int>> graph {
	\t{1},
	\t{0, 2},
	\t{1, 3},
	\t{2},
	\t{}
	};
	cout << connectedComponents(graph);
		2	2	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 3

Chính xác

Điểm 1,00 của 1,00

Implement Depth-first search

```
Adjacency *DFS(int v);
```

where Adjacency is a structure to store list of number.

```
#include <iostream>
#include <list>
using namespace std;

class Adjacency
{
private:
        list<int> adjList;
        int size;
public:
        Adjacency() {}
        Adjacency(int V) {}
        void push(int data)
        {
                adjList.push_back(data);
                size++;
        }
        void print()
        {
                for (auto const &i : adjList)
                        cout << " -> " << i;
        }
        void printArray()
        {
                for (auto const &i : adjList)
                        cout << i << " ";
        }
        int getSize() { return adjList.size(); }
        int getElement(int idx)
        {
                auto it = adjList.begin();
                advance(it, idx);
                return *it;
        }
};
```

And Graph is a structure to store a graph (see in your answer box)

For example:

Test	Result

int V = 8, visited = 0;
 	
Graph g(V);
Adjacency *arr;
int edge[][2] = {{0,1}, {0,2}, {0,3}, {0,4}, {1,2}, {2,5}, {2,6}, {4,6}, {6,7}};
for(int i = 0; i < 9; i++)
{
g.addEdge(edge[i][0], edge[i][1]);
}
 	
// g.printGraph();
// cout << endl;
arr = g.DFS(visited);
arr->printArray();
delete arr;
	0 1 2 5 6 4 7 3

Answer: (penalty regime: 0, 0, 5, ... %)

Reset answer

```
class Graph
{
private:
    int V;
    Adjacency *adj;

public:
    Graph(int V)
    {
        this->V = V;
        adj = new Adjacency[V];
    }

    void addEdge(int v, int w)
    {
        adj[v].push(w);
        adj[w].push(v);
    }

    void printGraph()
    {
        for (int v = 0; v < V; ++v)
        {
            cout << "\nAdjacency list of vertex " << v << "\nhead ";
            adj[v].print();
        }
    }
    void DFSHelp(int v, bool visited[], Adjacency* traversedList) {
       visited[v] = true;
       traversedList->push(v);
       for(int i = 0; i < adj[v].getSize(); ++i) {
           int adjacent = adj[v].getElement(i);
           if(!visited[adjacent]) {
               DFSHelp(adjacent, visited, traversedList);
           }
       }
    }
    Adjacency *DFS(int v)
    {
        // v is a vertex we start DFS
        bool visited[V] = {false};
        Adjacency* traversedList = new Adjacency;
        DFSHelp(v, visited, traversedList);
        return traversedList;
    }
};
```

	Test	Expected	Got	

✔	`int V = 8, visited = 0;`
	
	`Graph g(V);`
	`Adjacency *arr;`
	`int edge[][2] = {{0,1}, {0,2}, {0,3}, {0,4}, {1,2}, {2,5}, {2,6}, {4,6}, {6,7}};`
	`for(int i = 0; i < 9; i++)`
	`{`
	`\tg.addEdge(edge[i][0], edge[i][1]);`
	`}`
	
	`// g.printGraph();`
	`// cout << endl;`
	`arr = g.DFS(visited);`
	`arr->printArray();`
	`delete arr;`
		`0 1 2 5 6 4 7 3`	`0 1 2 5 6 4 7 3`	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 4

Chính xác

Điểm 1,00 của 1,00

Given a graph and a source vertex in the graph, find shortest paths from source to destination vertice in the given graph using Dijsktra's algorithm.

Following libraries are included: iostream, vector, algorithm, climits, queue

For example:

Test	Result

int n = 6;
int init[6][6] = {
{0, 10, 20, 0, 0, 0},
{10, 0, 0, 50, 10, 0},
{20, 0, 0, 20, 33, 0},
{0, 50, 20, 0, 20, 2},
{0, 10, 33, 20, 0, 1},
{0, 0, 0, 2, 1, 0} };
 	
int** graph = new int*[n];
for (int i = 0; i < n; ++i) {
graph[i] = init[i];
}
 	
cout << Dijkstra(graph, 0, 1);
	10

Answer: (penalty regime: 0 %)

Reset answer

```
// Some helping functions

int Dijkstra(int** graph, int src, int dst) {
    // TODO: return the length of shortest path from src to dst.
    // khởi tạo tập S khoảng cách vô cực
    int n = 6;
    vector<int> dist(n, INT32_MAX);
    vector<bool> visited(n, false);
    // Khởi tạo min-heap Q
    // pair<đỉnh 1, đỉnh 2>
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

    // Đưa đỉnh đầu tiên vào min-heap
    pq.push(make_pair(0, src));
    dist[src] = 0;

    while(!pq.empty()) {
        // Lấy phần tử đầu tiên ra
        int u = pq.top().second;
        pq.pop();
        visited[u] = true;

        for(int v = 0; v < n; ++v) {
             if (!visited[v] && graph[u][v] && dist[u] != INT32_MAX && dist[u] + graph[u][v] < dist[v]) {
                 dist[v] = dist[u] + graph[u][v];
                 pq.push(make_pair(dist[v], v));
             }
        }
    }
    return dist[dst];
}

```

	Test	Expected	Got	

✔	int n = 6;
	int init[6][6] = {
	\t{0, 10, 20, 0, 0, 0},
	\t{10, 0, 0, 50, 10, 0},
	\t{20, 0, 0, 20, 33, 0},
	\t{0, 50, 20, 0, 20, 2},
	\t{0, 10, 33, 20, 0, 1},
	\t{0, 0, 0, 2, 1, 0} };
	
	int** graph = new int*[n];
	for (int i = 0; i < n; ++i) {
	\tgraph[i] = init[i];
	}
	
	cout << Dijkstra(graph, 0, 1);
		10	10	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 5

Chính xác

Điểm 1,00 của 1,00

The relationship between a group of people is represented by an adjacency-list `friends`. If `friends[u]` contains `v`, `u` and `v` are friends.Friendship is a two-way relationship. Two people are in a friend group as long as there is some path of mutual friends connecting them.

Request: Implement function:

```
int numberOfFriendGroups(vector<vector<int>>& friends);
```

Where `friends` is the adjacency-list representing the friendship (this list has between 0 and 1000 lists). This function returns the number of friend groups.

Example:

Given a adjacency-list: `[[1], [0, 2], [1], [4], [3], []]`

There are `3` friend groups: `[0, 1, 2], [3, 4], [5]`

Note:

In this exercise, the libraries `iostream, string, cstring, climits, utility, vector, list, stack, queue, map, unordered_map, set, unordered_set, functional, algorithm` have been included and `namespace std` is used. You can write helper functions and class. Importing other libraries is allowed, but not encouraged.

For example:

Test	Result

vector<vector<int>> graph {
{1},
{0, 2},
{1},
{4},
{3},
{}
};
cout << numberOfFriendGroups(graph);
	3

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
void dfs(int u, vector<vector<int>>& friends, vector<bool>& visited) {
    visited[u] = true;

    for(int i = 0; i < int(friends[u].size()); ++i) {
        int adjacent = friends[u][i];
        if (!visited[adjacent]) {
            dfs(adjacent, friends, visited);
        }
    }
}
int numberOfFriendGroups(vector<vector<int>>& friends) {
    // STUDENT ANSWER
    vector<bool> visited(friends.size(), false);
    int count = 0;
    for(int i = 0; i < int(friends.size()); ++i) {
         if (!visited[i]) {
            dfs(i, friends, visited);
            ++count;
        }
    }
    return count;
}
```

	Test	Expected	Got	

✔	vector<vector<int>> graph {
	\t{1},
	\t{0, 2},
	\t{1},
	\t{4},
	\t{3},
	\t{}
	};
	cout << numberOfFriendGroups(graph);
		3	3	✔

✔	vector<vector<int>> graph {
	
	};
	cout << numberOfFriendGroups(graph);
		0	0	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 6

Chính xác

Điểm 1,00 của 1,00

Implement function to detect a cyclic in Graph

```
bool isCyclic();
```

Graph structure is defined in the initial code.

For example:

Test	Result

DirectedGraph g(8);
int edege[][2] = {{0,6}, {1,2}, {1,4}, {1,6}, {3,0}, {3,4}, {5,1}, {7,0}, {7,1}};
 	
for(int i = 0; i < 9; i++)
g.addEdge(edege[i][0], edege[i][1]);
 	
if(g.isCyclic())
cout << "Graph contains cycle";
else
cout << "Graph doesn't contain cycle";
	Graph doesn't contain cycle

Answer: (penalty regime: 0, 0, 5, ... %)

Reset answer

```
#include <iostream>
#include <vector>
#include <list>
using namespace std;

class DirectedGraph
{
    int V;
    vector<list<int>> adj;
public:
    DirectedGraph(int V)
    {
        this->V = V;
        adj = vector<list<int>>(V, list<int>());
    }
    void addEdge(int v, int w)
    {
        adj[v].push_back(w);
    }
    bool dfs(int u, vector<bool>& visited, vector<bool> recurs) {
        visited[u] = true;
        recurs[u] = true;
        for(list<int>::iterator it = adj[u].begin(); it != adj[u].end(); ++it) {
            int adjacent = *it;
            if (!visited[adjacent]) {
                if (dfs(adjacent, visited, recurs)) {
                    return true;
                }
            } else if (recurs[adjacent]) {
                return true;
            }
        }
        recurs[u] = false;
        return false;
    }
    bool isCyclic()
    {
```

```
38          // Student answer
39          vector<bool> visited(V, false);
40          vector<bool> recurs(V, false);
41          for(int i = 0; i < V; ++i) {
42              if(!visited[i] && dfs(i, visited, recurs)) return true;
43          }
44          return false;
45      }
46  };
```

	Test	Expected	Got	

✔	DirectedGraph g(8);
	int edege[][2] = {{0,6}, {1,2}, {1,4}, {1,6}, {3,0}, {3,4}, {5,1}, {7,0}, {7,1}};
	
	for(int i = 0; i < 9; i++)
	\tg.addEdge(edege[i][0], edege[i][1]);
	
	if(g.isCyclic())
	\tcout << "Graph contains cycle";
	else
	\tcout << "Graph doesn't contain cycle";
		Graph doesn't contain cycle	Graph doesn't contain cycle	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 7

Chính xác

Điểm 1,00 của 1,00

Implement **topologicalSort** function on a graph. (Ref here)

```
void topologicalSort();
```

where Adjacency is a structure to store list of number. Note that, the vertex index starts from 0. **To match the given answer, please always traverse from 0 when performing the sorting.**

```
#include <iostream>
#include <list>
using namespace std;

class Adjacency
{
private:
        list<int> adjList;
        int size;
public:
        Adjacency() {}
        Adjacency(int V) {}
        void push(int data)
        {
                adjList.push_back(data);
                size++;
        }
        void print()
        {
                for (auto const &i : adjList)
                        cout << " -> " << i;
        }
        void printArray()
        {
                for (auto const &i : adjList)
                        cout << i << " ";
        }
        int getSize() { return adjList.size(); }
        int getElement(int idx)
        {
                auto it = adjList.begin();
                advance(it, idx);
                return *it;
        }
};
```

And Graph is a structure to store a graph (see in your answer box). You could write one or more helping functions.

For example:

Test	Result

Graph g(6);
g.addEdge(5, 2);
g.addEdge(5, 0);
g.addEdge(4, 0);
g.addEdge(4, 1);
g.addEdge(2, 3);
g.addEdge(3, 1);
 	
g.topologicalSort();
	5 4 2 3 1 0

Answer: (penalty regime: 0, 0, 5, 10, ... %)

Reset answer

```
class Graph {

    int V;
    Adjacency* adj;

public:
    Graph(int V){
        this->V = V;
        adj = new Adjacency[V];
    }
    void addEdge(int v, int w){
        adj[v].push(w);
    }

    //Heling functions
    void dfs(int u, bool visited[], list<int>& Stack) {
        visited[u] = true;
        for(int i = 0; i < adj[u].getSize(); ++i) {
            int adjacent = adj[u].getElement(i);
            if(!visited[adjacent]) {
                dfs(adjacent, visited, Stack);

            }
        }
        Stack.push_back(u);
    }
    void topologicalSort(){
        //TODO
        bool* visited = new bool[V];
        list<int> Stack;
        for(int i = 0; i < V; ++i) {
            visited[i] = false;
        }
        for(int i = 0; i < V; ++i) {
            if(visited[i] == false) dfs(i, visited, Stack);
        }
        while(!Stack.empty()) {
            cout << Stack.back() << " ";
            Stack.pop_back();
        }
        delete[] visited;
    }
};
```

	Test	Expected	Got	

✔	`Graph g(6);`
	`g.addEdge(5, 2);`
	`g.addEdge(5, 0);`
	`g.addEdge(4, 0);`
	`g.addEdge(4, 1);`
	`g.addEdge(2, 3);`
	`g.addEdge(3, 1);`
	
	`g.topologicalSort();`
		`5 4 2 3 1 0`	`5 4 2 3 1 0`	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.