Câu hỏi 1

Không hoàn thành

Chấm điểm của 1,00

Implement Depth-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

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 %)

Reset answer

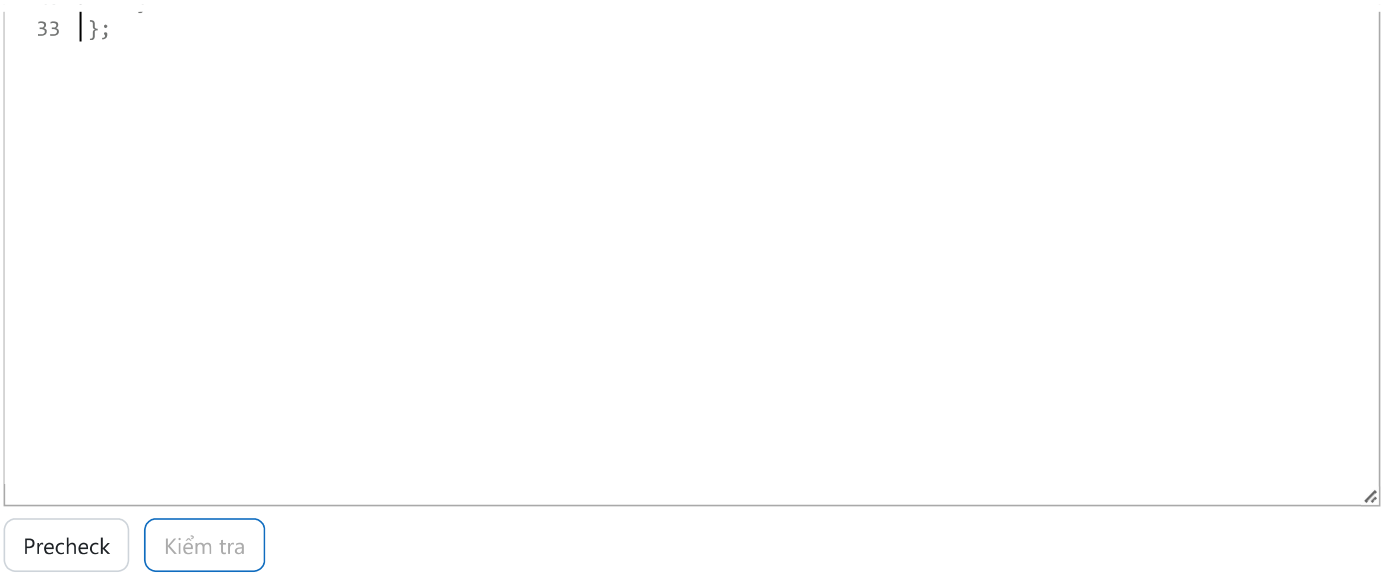
```
class Graph
{
private:
    int V;
    Adjacency *adj;

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
    }
```



```
};
```

Precheck
Kiểm tra

Câu hỏi 2

Không hoàn thành

Chấm điểm 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

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 %)

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 *DFS(int v)
    {
        // v is a vertex we start DFS
    }
};
```

Precheck

Kiểm tra

Câu hỏi 3

Không hoàn thành

Chấm điểm 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 %)

Reset answer

```
int connectedComponents(vector<vector<int>>& edges) {
    // STUDENT ANSWER
}
```

Precheck

Kiểm tra

Câu hỏi 4

Không hoàn thành

Chấm điểm 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.

For example:

<table>
<tr><th>Test</th><th>Result</th></tr>
<tr><td><pre>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, 0);</pre></td><td><pre>0</pre></td></tr>
<tr><td><pre>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);</pre></td><td><pre>10</pre></td></tr>
</table>

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.

}

```

Kiểm tra

Câu hỏi 5

Không hoàn thành

Chấm điểm 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 %)

Reset answer

```
int numberOfFriendGroups(vector<vector<int>>& friends) {
    // STUDENT ANSWER
}
```

Kiểm tra

Câu hỏi 6

Không hoàn thành

Chấm điểm của 1,00

Implement function to detect a cyclic in Graph

```
bool isCyclic();
```

Graph structure in this lab is slightly different from previous labs.

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

class DirectedGraph
{
        int V;
        list<int> *adj;
        bool isCyclicUtil(int v, bool visited[], bool *rs);
public:
        DirectedGraph(){
        V = 0;
        adj = NULL;
    }
        DirectedGraph(int V)
        {
                this->V = V;
                adj = new list<int>[V];
        }
        void addEdge(int v, int w)
        {
                adj[v].push_back(w);
        }
        bool isCyclic();
};
```

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 %)

Reset answer

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

class DirectedGraph
{
    int V;
    list<int> *adj;
    bool isCyclicUtil(int v, bool visited[], bool *rs);
public:
    DirectedGraph(){
        V = 0;
        adj = NULL;
    }
    DirectedGraph(int V)
    {
        this->V = V;
        adj = new list<int>[V];
    }
    void addEdge(int v, int w)
    {
        adj[v].push_back(w);
    }
    bool isCyclic()
    {
        // Student answer
    }
};
```

Precheck

Kiểm tra

Câu hỏi 7

Không hoàn thành

Chấm điểm của 1,00

Given an undirected, connected and weighted graph, find Minimum Spanning Tree (MST) of the graph using Kruskal’s algorithm.

Below are the steps for finding MST using Kruskal’s algorithm:

1. Sort all the edges in non-decreasing order of their weight.

2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.

3. Repeat step#2 until there are (V-1) edges in the spanning tree.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <algorithm>
#include <iostream>
#include <utility>
#include <map>
#include <vector>
#include <set>
using namespace std;

struct Graph
{
    int V, E;
    vector< pair<int, pair<int, int>> > edges;
    // Constructor
    Graph(int V, int E)
    {
        this->V = V;
        this->E = E;
    }

    void addEdge(int u, int v, int w)
    {
        edges.push_back({ w, {u, v} });
    }

    //YOUR CODE HERE

};
```

For example:

Test	Result

`int V = 2, E = 1;`
`Graph g(V, E);`
	2

BÁCH KHOA E-LEARNING

WEBSITE

HCMUT

MyBK

BKSI

LIÊN HỆ

268 Lý Thường Kiệt, P.14, Q.10, TP.HCM

(028) 38 651 670 - (028) 38 647 256 (Ext: 5258, 5234)

elearning@hcmut.edu.vn

Reset answer

```
// Some helping functions

int kruskalMST() {
    // TODO: return weight of the minimum spanning tree.

}
```

Kiểm tra