1

Chính xác

Điểm 1,00 của 1,00

Cờ câu hỏi

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)

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`

Câu hỏi 2

Chính xác

Điểm 1,00 của 1,00

Cờ câu hỏi

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)

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, ... %)

Câu hỏi 3

Chính xác

Điểm 1,00 của 1,00

Cờ câu hỏi

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

Câu hỏi 4

Chính xác

Điểm 1,00 của 1,00

Cờ câu hỏi

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`

Câu hỏi 5

Chính xác

Điểm 1,00 của 1,00

Cờ câu hỏi

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;
        }
};
```

```
        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