Trạng thái	Đã xong
Bắt đầu vào lúc	Thứ Ba, 16 tháng 4 2024, 1:49 PM
Kết thúc lúc	Thứ Tư, 24 tháng 4 2024, 2:37 PM
Thời gian thực hiện	8 Các ngày

Câu hỏi 1

Đúng

Đạt điểm 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];
    }
```

```
13
14      void addEdge(int v, int w)
15      {
16          adj[v].push(w);
17          adj[w].push(v);
18      }
19
20      void printGraph()
21      {
22          for (int v = 0; v < V; ++v)
23          {
24              cout << "\nAdjacency list of vertex " << v << "\nhead ";
25              adj[v].print();
26          }
27      }
28
29      Adjacency *BFS(int v)
30      {
31          // v is a vertex we start BFS
32          Adjacency* result = new Adjacency();
33          list<int> l;
34          bool check[V] = {false};
35          l.push_back(v);
36          check[v] = true;
37          while (!l.empty()) {
38              int e = l.front();
39              l.pop_front();
40              result->push(e);
41              for (int i = 0; i < adj[e].getSize(); ++i) {
42                  int temp = adj[e].getElement(i);
43                  if (check[temp] == false) {
44                  l.push_back(temp);
45                  check[temp] = true;
46                  }
47              }
48          }
49          return result;
50      }
51  };
```

	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! ✓

Câu hỏi **2**

Đúng

Đạt điểm 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 v, vector<vector<int>>& edges, vector<bool>& check) {
    if (check[v] == false) {
    check[v] = true;
        for (int i : edges[v])
            dfs(i,edges,check);
    }
}

int connectedComponents(vector<vector<int>>& edges) {
    // STUDENT ANSWER
    vector<bool> check(edges.size(), false);
    int size = edges.size();
    int cnt = 0;
    for (int i = 0 ; i < size; ++i) {
        if (check[i] == false) {
        dfs (i, edges, check);
```

```
        dfs (i, edges, check);
        cnt++;
        }
    }
    return cnt;
}
```

	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! ✓

Câu hỏi **3**

Đúng

Đạt điểm 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 dfs(int v, Adjacency* result, bool* check) {
```

```
30          if (check[v] == false) {
31          check[v] = true;
32          result->push(v);
33              for (int i = 0; i < adj[v].getSize(); ++i) {
34              int tmp = adj[v].getElement(i);
35              dfs(tmp, result, check);
36                  check[tmp] = true;
37              }
38          }
39      }
40
41      Adjacency *DFS(int v)
42      {
43          bool check[V] = {false};
44          Adjacency* result = new Adjacency();
45          dfs(v, result, check);
46          return result;
47      }
48  };
```

	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();
		0 1 2 5 6 4 7
		3
			0 1 2 5 6 4 7
			3
				✓

Passed all tests! ✓

Câu hỏi **4**

Đúng

Đạt điểm 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.
    vector<int> v(6, INT_MAX);
    v[src] = 0;
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
    pq.push({0, src});
        while (!pq.empty()) {
            int i = pq.top().second;
            pq.pop();
            for (int j = 0; j < 6; ++j) {
                if (graph[i][j] && v[i] + graph[i][j] < v[j]) {
                v[j] = v[i] + graph[i][j];
                pq.push({v[j], j});
            }
        }
    }
    return v[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! ✓

Câu hỏi **5**

Đúng

Đạt điểm 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 v, vector<vector<int>>& friends, vector<bool>& check) {
    if (check[v] == false) {
    check[v] = true;
        for (int i : friends[v])
        dfs(i,friends,check);
    }
}

int numberOfFriendGroups(vector<vector<int>>& friends) {
    vector<bool> check(friends.size(), false);
    int size = friends.size();
    int cnt = 0;
```

```
        for (int i = 0 ; i < size; ++i) {
            if (check[i] == false) {
            dfs (i, friends, check);
            cnt++;
        }
    }
return cnt;
}
```

	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! ✓

Câu hỏi **6**

Đúng

Đạt điểm 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 v, bool* check, int e) {
        int i = v;
        if (!check[i]) {
            check[i] = true;
            bool tmp = false;
                for (int i : adj[i]) {
```

```
                for (int i : adj[i]) {
                tmp = tmp || dfs(i, check, e);
                    if(tmp)
                break;
                }
            return tmp;
            }
            else if (v == e)
        return true;
    return false;
    }

    bool isCyclic()
    {
        for (int i = 0; i < V; ++i) {
        bool check[V] = {false};
            if (dfs(i, check, i))
        return true;
        }
    return false;
    }
};
```

	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! ✓

Câu hỏi **7**

Đúng

Đạt điểm 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 topologicalSortUtil(int v, bool check[], list<int>& st) {
    check[v] = true;
        for (int i = 0; i < adj[v].getSize(); ++i) {
        int tmp = adj[v].getElement(i);
            if (!check[tmp])
            topologicalSortUtil(tmp, check, st);
            }
        st.push_back(v);
    }

    void topologicalSort(){
        list<int> st;
        bool* check = new bool[V];
            for (int i = 0; i < V; ++i) {
            check[i] = false;
        }
        for (int i = 0; i < V; ++i) {
            if (check[i] == false)
            topologicalSortUtil(i, check , st);
```

```
        }
        while (!st.empty()) {
            cout << st.back() << " ";
            st.pop_back();
        }
    }
};
```

	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! ✓