HW3

Profit Maximization Problem by recursive and non-recursive version program

B10532011 高靖雅

程式摘要：

此程式使用 C++語言

判斷一個 directed graph 是否為 singly connected

程式內容說明：

一開始輸入的時候如果有重複的邊，我的程式會讀完測資然後直接判斷 NO。如果沒有，那就會進入 recursive，把每一個點當作 root，並沿著箭頭一路走下去，如果走到之前走過的點就直接 return，如果之前沒走過，但是之前 root 可以直接或間接到 pointToNext，那就判定不是 singly connected，如果上述情況都沒有發生，那就標記 root 可以間接到達這個點，然後繼續往下走，直到可以走的都走到底了，那就在換一個點當作 root

圖解:

root=0

0→1

check	0	1	2
	1	1	0

adj	0	1	2
0		1	
1			1
2	1	1	

1→2

check	0	1	2
	1	1	1

adj	0	1	2
0		1	1
1			1
2	1	1	

2→0 return

root=1

1→2

check	0	1	2
	0	1	1

adj	0	1	2
0		1	1
1			1
2	1	1	

2→0

check	0	1	2
	1	1	1

adj	0	1	2
0		1	1
1	1		1
2	1	1	

0→1 return

root=2

2→0

check	0	1	2
	1	0	1

adj	0	1	2
0		1	1
1	1		1
2	1	1	

0→1

check	0	1	2
	1	1	1

adj	0	1	2
0		1	1
1	1		1
2	1	1	

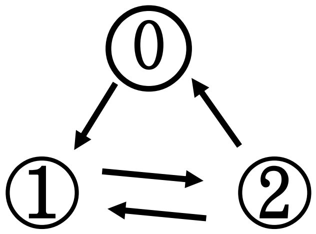
1→2 return

2→1

check	0	1	2
	0	1	1

adj	0	1	2
0		1	
1			1
2	1	$\boxed{1}$	

has been visited



Pseudocode:

```
void goDeep(int k, int l) {                            //recursive
    check[k] = 1;                                       //denote k is visited
    int pointToNext = v[k][l];                          //k  指向的第l個點
    if the next vertex that k point to is visited
        return;
    if root visited pointToNext before and k is not root
        this is not a singly connected graph
        return;
    adjacent[root][pointToNext] = 1;                    // set root visit pointToNext
    check[pointToNext] = 1;                             //denote pointToNext is visited
    for (int j = 0; j < v[pointToNext].size(); j++) {
        goDeep(pointToNext, j);
    }
    return;
}

int main()
    input set;                                          // how many test case
    for i to set
        input vec, edg                                  // how many vertices, edges
        initialize adjacent[900][900]
        ans = 1;
        for j to edg
            input x, y                                  //pair of connected vertex
            v[x].push_back(y);
            adjacent[x][y] = 1;

        for every vertex be root
            for every vertex that can be directly reached from current root vertex
                check.reset();
                root = k;
                goDeep(k, l);
        if (ans)
            print "YES"
        else
            print "NO"
```

程式:

```cpp
#include <iostream>
#include <map>
#include <vector>
#include <bitset>
using namespace std;
int adj[1000][1000];
int root;
bool ans;
map<int, vector<int> > node;
bitset<900> check;

void goDeep(int k, int l) {
    check[k] = 1;
    int pointTo = node[k][l];
    if (check[pointTo] == 1) {
        return;
    }
    if (adj[root][pointTo] == 1 && k != root) {
        ans = false;
        return;
    }
    adj[root][pointTo] = 1;
    check[pointTo] = 1;
    for (int j = 0; j < node[pointTo].size(); j++) {
        goDeep(pointTo, j);
    }
    return;
}
int main() {

    int n, vec, edg;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> vec >> edg;
        for (int s = 0; s < 1000; s++) {
            for (int d = 0; d < 1000; d++) {
                adj[s][d] = 0;
            }
        }
        int j;
        int x, y;
        ans = true;
        for (j = 0; j < edg; j++) {
            cin >> x >> y;
            if (adj[x][y]) {
                ans = false;
                break;
            }
            node[x].push_back(y);
            adj[x][y] = 1;
        }
        for (; j < edg; j++)
            cin >> x >> y;
        for (int k = 0; k < vec; k++) {
            if (!ans)
                break;
            for (int l = 0; l < node[k].size(); l++) {
                check.reset();
                root = k;
                goDeep(k, l);
            }
        }
        if (ans)
            cout << i + 1 << " YES" << endl;
        else
            cout << i + 1 << " NO" << endl;
        node.clear();
    }
    //system("pause");
    return 0;
}
```