DFS算法

1,C++代码.

//DFS，使用 邻接矩阵+栈 实现  
  
#include <iostream>  
  
#include <stack>  
  
using namespace std;  
  
#define MAX\_VERTS 20  
  
class Vertex  
  
{  
  
public:  
  
Vertex(char lab)  
  
{  
  
Label = lab;  
  
wasVisited = false;  
  
}  
  
public:  
  
bool wasVisited;  
  
char Label;  
  
};  
  
class Graph  
  
{  
  
public:  
  
Graph();  
  
~Graph();  
  
void addVertex(char lab);  
  
void addEdge(int start, int end);  
  
void printMatrix();  
  
void showVertex(int v);  
  
void DFS();  
  
private:  
  
Vertex\* vertexList[MAX\_VERTS];  
  
int nVerts;  
  
int adjMat[MAX\_VERTS][MAX\_VERTS];  
  
int getAdjUnvisitedVertex(int v);  
  
};  
  
Graph::Graph()  
  
{  
  
nVerts = 0;  
  
for (int i = 0; i < MAX\_VERTS; i++)  
  
{  
  
for (int j = 0; j < MAX\_VERTS; j++)  
  
{  
  
adjMat[i][j] = 0;  
  
}  
  
}  
  
}  
  
Graph::~Graph()  
  
{  
  
//delete[] vertexList;  
  
}  
  
//添加顶点  
  
void Graph::addVertex(char lab)  
  
{  
  
vertexList[nVerts++] = new Vertex(lab);  
  
}  
  
//添加边  
  
void Graph::addEdge(int start, int end)  
  
{  
  
adjMat[start][end] = 1;  
  
adjMat[end][start] = 1;  
  
}  
  
void Graph::printMatrix()  
  
{  
  
for (int i = 0; i < nVerts; i++)  
  
{  
  
for (int j = 0; j < nVerts; j++)  
  
{  
  
cout << adjMat[i][j] << " ";  
  
}  
  
cout << endl;  
  
}  
  
}  
  
//显示顶点标签  
  
void Graph::showVertex(int v)  
  
{  
  
cout << vertexList[v]->Label << " ";  
  
}  
  
//获得未访问过的下一个顶点  
  
int Graph::getAdjUnvisitedVertex(int v)  
  
{  
  
for (int j = 0; j < nVerts; j++)  
  
{  
  
//邻接的并且没被访问过  
  
if ((adjMat[v][j] == 1) && (vertexList[j]->wasVisited == false))  
  
return j;  
  
}  
  
return -1;  
  
}  
  
//深度优先搜索  
  
void Graph::DFS()  
  
{  
  
stack<int> gStack;  
  
vertexList[0]->wasVisited = true;  
  
showVertex(0);  
  
//要把访问过的顶点压入栈中  
  
gStack.push(0);  
  
int v;  
  
while (gStack.size() > 0)  
  
{  
  
//访问当前顶点的下一个  
  
v = getAdjUnvisitedVertex(gStack.top());  
  
//如果没找到就利用栈向回找  
  
if (v == -1)  
  
gStack.pop();  
  
else  
  
{  
  
vertexList[v]->wasVisited = true;  
  
showVertex(v);  
  
gStack.push(v);  
  
}  
  
}  
  
cout << endl;  
  
//为了下一次搜索再把wasVisited变成false  
  
for (int j = 0; j < nVerts; j++)  
  
vertexList[j]->wasVisited = false;  
  
}  
  
int main()  
  
{  
  
Graph g;  
  
g.addVertex(&apos;A&apos;);  
  
g.addVertex(&apos;B&apos;);  
  
g.addVertex(&apos;C&apos;);  
  
g.addEdge(0, 1);  
  
g.addEdge(0, 2);  
  
  
  
g.printMatrix();  
  
g.DFS();  
  
system("pause");  
  
return 0;  
  
}