2022212153 陈祥烨 计科22-2班 第九章作业

**第一题**

**题目：**

**（2）设计算法, 按照广度遍历 算法的基本方法判断有向图G是否是一棵以v0为根的有向树。**

**思路：**

将所有结点设置为未被访问0，用广度遍历算法的遍历后，遍历后标志变为1。遍历结束后，判断是否每个元素的标志都为1，如果都为1，则v0是根，反之不是。

**代码：**

#include<iostream>

#include"queue"

using namespace std;

bool visited[10] = { false };

void visit(int i)

{

visited[i] = true;

}

void bfs(int v0)

{

visit(v0);

visited[v0] = true;

Queue Q;

Q.Append(v0);

while (!Q.Empty()) {

v = Q.Serve();

int w = firstadj(G, v);

while (w != 0) {

if (!visited[w]) {

visit(w);

visited[w] = true;

Q.Append(w);

}

w = nextadj(G, v, w);

}

}

}

bool isroot(int vo)

{

for (int i = 0; i < 10; i++)

{

if (!visited[i])return false;

}

return true;

}

**第二题**

**题目：**

**（7）设计算法求顶点v0到每个顶点的最短路径（以弧数为单位）, 要求时间尽可能地少。**

**思路：**

利用弗洛伊德算法，利用邻接矩阵储存

**代码：**

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

#define MAXVEX 100//最大顶点数

typedef char VertexType;//顶点类型

typedef int EdgeType;//边上的权值类型

struct MGraph

{

VertexType vexs[MAXVEX];//顶点表

EdgeType arc[MAXVEX][MAXVEX];//邻接矩阵

int numVertexte;//当前顶点数

int numEdges;//当前边数

};

void ShortestPath\_Floyd(MGraph G, vector<vector<int>>& P, vector<vector<int>>& D)

{

for (int i = 0; i < G.numVertexte; ++i)

{

for (int j = 0; j < G.numVertexte; ++j)

{

D[i][j] = G.arc[i][j];//用G的邻接矩阵初始化D

P[i][j] = j;

}

for (int i = 0; i < G.numVertexte; ++i)

{

for (int j = 0; j < G.numVertexte; ++j)

{

for (int w = 0; w < G.numVertexte; ++w)

{

if (D[j][w] > D[j][i] + D[i][w])//如果经过下标为i的顶点路径比原两点间路径更短

{

D[j][w] = D[j][i] + D[i][w];//更新D的值

P[j][w] = P[j][i];//更新P的值

}

}

}

}

}

}

void Print(MGraph G, vector<vector<int>>& P, vector<vector<int>>& D)

{

for (int i = 0; i < G.numVertexte; ++i)

{

for (int j = i + 1; j < G.numVertexte; ++j)

{

cout << i << " " << j <<" " << D[i][j]<<endl; //打印源点 终点 以及他们之前的权

int k = P[i][j];//第一个路径顶点下标

cout << "path" << i << endl;//打印源点

while (k != j)//如果没有到终点

{

printf("-> %d", k);//打印路径顶点

k = P[k][j];//获取下一个路径顶点下标

}

cout << "->" << j << endl; //打印终点

}

cout << endl;

}

}

**第三题**

**题目：**

**9.5 已知图G用邻接矩阵存储，设计算法以分别实现函数firstadj(G, *v*)和nextadj(G, *v*, *w*)。**

**代码：**

#include<iostream>

#include<vector>

#include<set>

using namespace std;

struct Edge;

struct Edge {

int from;//起点

int to;//终点

int weight;//权

Edge();

Edge(int \_weight, int \_from, int \_to) {

weight = \_weight;

from = \_from;

to = \_to;

}

};

struct Graph {

public:

int nodes[1000][1000];//储存所有点

set<Edge\*> edges;//储存所有边

int nodecount;//结点数

};

Graph\* CreateGraph(int matrix[][3], int n,int nc) //n表示边数 即matrix.size(),nc为结点数

{

Graph\* graph = new Graph;

graph->nodecount = nc;

for (int i = 0; i < nc; i++) {

for (int j = 0; j < nc; j++) {

graph->nodes[i][j] = 0;

}

}

for (int i = 0; i != n; i++)

{

int weight = matrix[i][0];

int from = matrix[i][1];

int to = matrix[i][2];

graph->nodes[from-1][to-1] = weight;

Edge\* newedge = new Edge(weight, from, to);

graph->edges.insert(newedge);

}

return graph;

}

int firstadj(Graph\* graph,int node) {

for (int i = 0; i < graph->nodecount; i++) {

if (graph->nodes[node-1][i] != 0)return i+1;

}

return 0;

}

int nextadj(Graph\* graph,int node, int wnode) {

for (int i = wnode; i < graph->nodecount; i++) {

if (graph->nodes[node-1][i] != 0)return i+1;

}

return 0;

}

void print(Graph\* a) {

cout << "邻接矩阵为：" << endl;

for (int i = 0; i < a->nodecount; i++) {

for (int j = 0; j < a->nodecount; j++) {

cout << a->nodes[i][j] << " ";

}

cout << endl;

}

}

int main() {

Graph\* a;

int m[6][3] = { {1,1,3},{2,1,4},{1,3,2},{2,4,5},{3,2,4},{1,4,3} };

a = CreateGraph(m, 6,5);

print(a);

cout << "4的firstadj(a,4)为：" << firstadj(a, 4) << endl;

cout << "4的nextadj(a,4,firstadj(a,4))为：" << nextadj(a, 4, firstadj(a, 4)) << endl;

cout << "4的nextadj(a,4,5)为：" << nextadj(a, 4, 5) << endl;

}

**第四题**

**题目：**

**9.10 设计算法以判断无向图G是否是连通的，若连通，返回TRUE，否则返回FALSE；**

**思路：**

采用[深度优先遍历](https://so.csdn.net/so/search?q=%E6%B7%B1%E5%BA%A6%E4%BC%98%E5%85%88%E9%81%8D%E5%8E%86&spm=1001.2101.3001.7020)的方式判断无向图G是否连通。若[遍历](https://so.csdn.net/so/search?q=%E9%81%8D%E5%8E%86&spm=1001.2101.3001.7020)之后，访问标记数组visited[]中的所有元素值均为1，则图G是连通的，否则不连通。

**代码：**

#define maxvertexnum 30 //最大顶点个数

typedef char VertexType; //顶点的类型

typedef int EdgeType;

typedef struct {

VertexType Vertex[maxvertexnum]; //顶点表

EdgeType Edge[maxvertexnum][maxvertexnum]; //边表

int vexnum, edgenum; //顶点数和边数

}MGraph;

//返回顶点的存储下标，将顶点值转换为顶点号

int getPos(MGraph G, VertexType v) {

int i;

for (i = 0; i < G.vexnum; i++) {

if (G.Vertex[i] == v) {

break;

}

}

return i;

}

int visited[maxvertexnum]; //访问标记数组

//深度优先遍历

void DFS(MGraph G, VertexType v) {

// cout<<v;

visited[getPos(G, v)] = 1;

for (VertexType w = FirstNeighbor(G, v); w != '0'; w = NextNeighbor(G, v, w)) {

if (!visited[getPos(G, w)])

DFS(G, w);

}

}

void DFSTraverse(MGraph G) {

for (int i = 0; i < G.vexnum; i++) {

visited[i] = 0; //初始化标记数组

}

for (int i = 0; i < G.vexnum; i++) {

if (!visited[i]) {

DFS(G, G.Vertex[i]);

}

}

}

//判断是否连通

int IsConnect(MGraph G) {

DFSTraverse(G);

for (int i = 0; i < G.vexnum; i++) {

if (visited[i] = 0)

return 0;

}

return 1;

}

**第五题**

**题目：**

**9.15 设连通图用邻接表A表示，设计算法以产生dfs（1）的dfs生成树，并存储到邻接矩阵B中。**

**思路：**

利用深度优先算法遍历连通图，成功遍历到时，用于存储连通图的邻接矩阵visited[i][j] = 1,表示已被访问过

**代码：**

int visited[20][20] = { 0 }; //全局量数组,用以标记某个顶点是否被访问过

void DFSl(ALGraph G, int i)

{ //从顶点Vi出发,深度优先搜索遍历图G(邻接表结构)

EdgeNode\* p;

int j;

visited[i] [j] = 1; //标记vi已访问过

p = G[i].link; //取Vi邻接表的表头指针

while (p != NULL) //依次搜索vi的每个邻接点

{

j = p->adjvex; // j为vi的一个邻接点序号

if (!visited[j])

DFSl(G, j); //若(vi,vj)∈E(G),且vj未被访问过,则从开始递归调用

p = p->next; //使p指向vi的下一个邻接点

} // End-while

}

**第六题**

**题目：**

**9.19 设计算法以求解二叉树T中层次最小的一个叶子结点的值。**

**代码：**

#include<iostream>

#include<vector>

#include<map>

#include<string>

using namespace std;

int mindeep = 0;

char Data;

struct Node {

char data;

Node\* Lchild, \* Rchild;

};

struct Tree {

Node\* root;

int count;

Tree() { root = new Node; }

~Tree() { delete root; }

};

void CreateTree(Node\*& node) {

char x;

cin >> x;

if (x == '#') {

node=NULL;

return;

}

else {

node= new Node;

node->data = x;

CreateTree(node->Lchild);

CreateTree(node->Rchild);

}

}

void Pre\_Traversal(Node\* node,int deep) {//deep为当前结点的层次

cout << node->data << " ";

if(node->Lchild!=NULL)

Pre\_Traversal(node->Lchild,deep+1);

if(node->Rchild!=NULL)

Pre\_Traversal(node->Rchild,deep+1);

if (node->Lchild == NULL && node->Rchild == NULL) {

if (mindeep == 0) {

mindeep = deep;

Data = node->data;

}

else {

if (mindeep > deep) {

mindeep = deep;

Data = node->data;

}

}

}

}

int main() {

Tree T;

CreateTree(T.root);

Pre\_Traversal(T.root,1);

cout << "\n最小的层次为："<<mindeep << endl;

cout << "层次最小的一个叶子为" << Data << endl;