## 实验9

8.1 实验目的

(1) 掌握图的基本概念。

(2) 掌握图的存储结构的设计与实现，基本运算的实现。

(3) 熟练掌握图的两种遍历算法、遍历生成树及遍历算法的应用。

8.2 实验任务

分别设计图（网）的邻接矩阵、邻接表存储结构，编写算法实现下列问题的求解。、

数据要求：可利用8.3中的数据，也可自行编写数据。

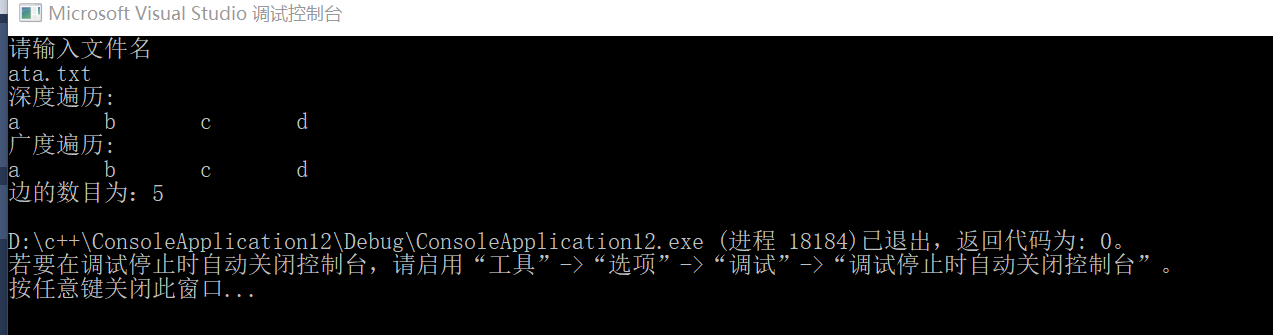
1．打印出图（网）的两种遍历序。

2．求给定图中的边（或弧）的数目。

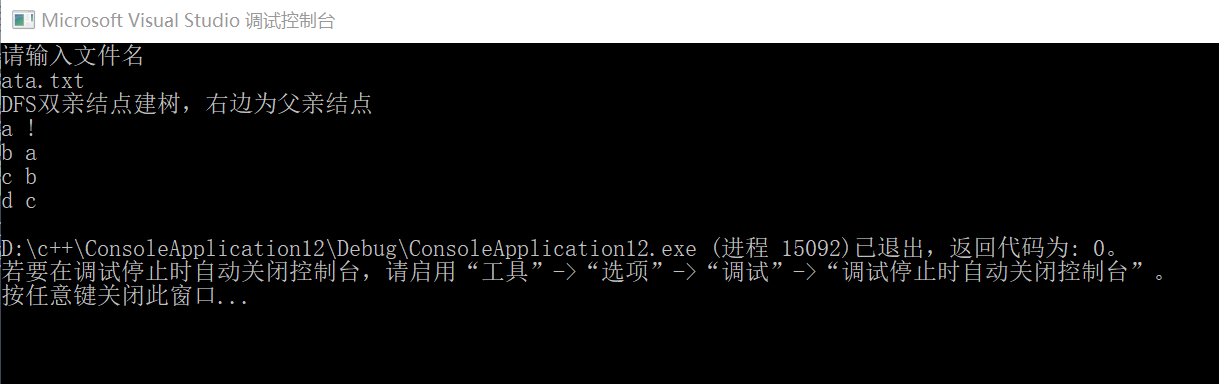
3．对给定的图G及出发点v0，设计算法从V0出发深度优先遍历图G，并构造出相应的生成树或生成森林。

4．对给定的图G及出发点v0，设计算法从V0出发广度优先遍历图G，并构造出相应的生成树或生成森林。

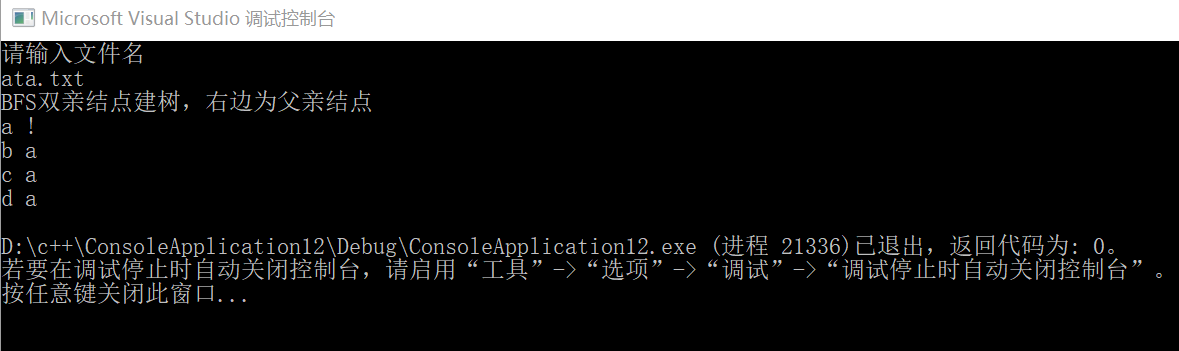
**截图**

****

**1 2**

****

**3**

****

**4**

**源代码**

#include "pch.h"

#include <iostream>

#define INF 65535

#define MaxVerNum 1000

#define maxnum 100

typedef char elementType;

typedef int cellType;

typedef enum{UDG,UDN,DG,DN}GraphKind;

using namespace std;

int E = 0;

int A = 0;

bool visited[MaxVerNum];

bool visitedd[MaxVerNum];

typedef struct GraphAdjMatrix

{

elementType Data[MaxVerNum];

cellType AdjMatrix[MaxVerNum][MaxVerNum];

int ArcNum;

int VerNum;

GraphKind gKind;

}Graph;

typedef struct sQueue

{

int data[MaxVerNum];

int front;

int rear;

}seqQueue;

struct tnode

{

char data;

char parent;

};

struct tnode treelist[maxnum];

void initialQueue(seqQueue\*Q)

{

Q->front = 0;

Q->rear = 0;

}

bool empty(seqQueue&Q)

{

if (Q.front == Q.rear)

return true;

else

return false;

}

void getFront(seqQueue&Q, int&x)

{

if (empty(Q))

cout << "队空" << endl;

else

{

x = Q.data[(Q.front + 1) % MaxVerNum];

}

}

bool full(seqQueue&Q)

{

if (((Q.rear + 1) % MaxVerNum) == Q.front)

return true;

else

return false;

}

void enQueue(seqQueue\*Q,int x)

{

if (full(\*Q))

cout << "队满！" << endl;

else

{

Q->rear = ((Q->rear + 1) % MaxVerNum);

Q->data[Q->rear] = x;

}

}

void outQueue(seqQueue\*Q)

{

if (empty(\*Q))

cout << " 队空 !" << endl;

else

{

Q->front = (Q->front + 1) % MaxVerNum;

}

}

void strLTrim(char\*str)

{

char\*p;

if (str == NULL)

return;

for (p = str; \*p == ' '; ++p);

if (\*p == 0)

{

\*str = 0;

return;

}

}

int CreateGraphFromFile( Graph &G)

{

FILE\* pFile; //定义文件指针

char str[1000]; //存放读出一行文本的字符串

char strTemp[10]; //判断是否注释行

cellType eWeight; //边的信息，常为边的权值

GraphKind graphType; //图类型枚举变量

cout << "请输入文件名" << endl;

char fileName[50];

cin >> fileName;

pFile = fopen(fileName, "r");

if (!pFile)

{

printf("错误：文件%s打开失败。\n", fileName);

return false;

}

while (fgets(str, 1000, pFile) != NULL)

{

strLTrim(str); //删除字符串左边空格，这是一个自定义的函数

if (str[0] == '\n') //空行，继续读取下一行

continue;

strncpy(strTemp, str, 2);

if (strstr(strTemp, "//") != NULL) //跳过注释行

continue;

else //非注释行、非空行，跳出循环

break;

}

//循环结束，str中应该已经是图的标识Graph，判断标识是否正确

if (strstr(str, "Graph") == NULL)

{

printf("错误：打开的文件格式错误！\n");

fclose(pFile); //关闭文件

return false;

}

//读取图的类型，跳过空行

while (fgets(str, 1000, pFile) != NULL)

{

strLTrim(str); //删除字符串左边空格，这是一个自定义函数

if (str[0] == '\n') //空行，继续读取下一行

continue;

strncpy(strTemp, str, 2);

if (strstr(strTemp, "//") != NULL) //注释行，跳过，继续读取下一行

continue;

else //非空行，也非注释行，即图的类型标识

break;

}

//设置图的类型

if (strstr(str, "UDG"))

graphType = UDG; //无向图

else if (strstr(str, "UDN"))

graphType = UDN; //无向网

else if (strstr(str, "DG"))

graphType = DG; //有向图

else if (strstr(str, "DN"))

graphType = DN; //有向网

else

{

printf("错误：读取图的类型标记失败！\n");

fclose(pFile); //关闭文件

return false;

}

//读取顶点行数据到str。跳过空行

while (fgets(str, 1000, pFile) != NULL)

{

strLTrim(str); //删除字符串左边空格，这是一个自定义函数

if (str[0] == '\n') //空行，继续读取下一行

continue;

strncpy(strTemp, str, 2);

if (strstr(strTemp, "//") != NULL) //注释行，跳过，继续读取下一行

continue;

else //非空行，也非注释行，即图的顶点元素行

break;

}

//顶点数据放入图的顶点数组

char\* token = strtok(str, " ");

int nNum = 0;

while (token != NULL)

{

G.Data[nNum] = \*token;

token = strtok(NULL, " ");

nNum++;

}

//图的邻接矩阵初始化

int nRow = 0; //矩阵行下标

int nCol = 0; //矩阵列下标

if (graphType == UDG || graphType == DG)

{

for (nRow = 0; nRow < nNum; nRow++)

for (nCol = 0; nCol < nNum; nCol++)

G.AdjMatrix[nRow][nCol] = 0;

}

else

{

for (nRow = 0; nRow < nNum; nRow++)

for (nCol = 0; nCol < nNum; nCol++)

G.AdjMatrix[nRow][nCol] = INF; //INF表示无穷大

}

//循环读取边的数据到邻接矩阵

int edgeNum = 0; //边的数量

elementType Nf, Ns; //边或弧的2个相邻顶点

while (fgets(str, 1000, pFile) != NULL)

{

strLTrim(str); //删除字符串左边空格，这是一个自定义函数

if (str[0] == '\n') //空行，继续读取下一行

continue;

strncpy(strTemp, str, 2);

if (strstr(strTemp, "//") != NULL) //注释行，跳过，继续读取下一行

continue;

char\* token = strtok(str, " "); //以空格为分隔符，分割一行数据，写入邻接矩阵

if (token == NULL) //分割为空串，失败退出

{

printf("错误：读取图的边数据失败！\n");

fclose(pFile); //关闭文件

return false;

}

Nf = \*token; //获取边的第一个顶点

token = strtok(NULL, " "); //读取下一个子串，即第二个顶点

if (token == NULL) //分割为空串，失败退出

{

printf("错误：读取图的边数据失败！\n");

fclose(pFile); //关闭文件

return false;

}

Ns = \*token; //获取边的第二个顶点

//从第一个顶点获取行号

for (nRow = 0; nRow < nNum; nRow++)

{

if (G.Data[nRow] == Nf) //从顶点列表找到第一个顶点的编号

break;

}

//从第二个顶点获取列号

for (nCol = 0; nCol < nNum; nCol++)

{

if (G.Data[nCol] == Ns) //从顶点列表找到第二个顶点的编号

break;

}

//如果为网，读取权值

if (graphType == UDN || graphType == DN)

{ //读取下一个子串，即边的附加信息，常为边的权重

token = strtok(NULL, " ");

if (token == NULL) //分割为空串，失败退出

{

printf("错误：读取图的边数据失败！\n");

fclose(pFile); //关闭文件

return false;

}

eWeight = atoi(token); //取得边的附加信息

}

if (graphType == UDN || graphType == DN)

G.AdjMatrix[nRow][nCol] = eWeight;

//如果为网，邻接矩阵中对应的边设置权值，否则置为1

else

G.AdjMatrix[nRow][nCol] = 1;

edgeNum++; //边数加1

}

G.VerNum = nNum; //图的顶点数

if (graphType == UDG || graphType == UDN)

G.ArcNum = edgeNum / 2; //无向图或网的边数等于统计的数字除2

else

G.ArcNum = edgeNum;

G.gKind = graphType; //图的类型

fclose(pFile); //关闭文件

return true;

}

void DFS(Graph&G, int verID)

{

cout << G.Data[verID - 1] << "\t";

visited[verID - 1] = true;

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

{

E++;

if ((G.AdjMatrix[verID - 1][w] >= 1) && (G.AdjMatrix[verID - 1][w] < INF) && (!visited[w]))

{

DFS(G,w + 1);

}

}

}

void DFStree(Graph&G, int verID,tnode\*treelist)

{

treelist[A].data= G.Data[verID - 1];

treelist[0].parent ='!';

visited[verID - 1] = true;

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

{

if ((G.AdjMatrix[verID - 1][w] >= 1) && (G.AdjMatrix[verID - 1][w] < INF) && (!visited[w]))

{

A++;

treelist[A].parent = G.Data[verID - 1];

DFStree(G, w + 1,treelist);

}

}

}

void BFS(Graph &G, int verID)

{

int u;

seqQueue Q;

initialQueue(&Q);

cout << G.Data[verID - 1] << "\t";

visitedd[verID - 1] = true;

enQueue(&Q, verID);

while (!empty(Q))

{

getFront(Q, u);

outQueue(&Q);

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

{

if ((G.AdjMatrix[u - 1][w] >= 1) && (G.AdjMatrix[u - 1][w] < INF) && (!visitedd[w]))

{

cout << G.Data[w] << "\t";

visitedd[w] = true;

enQueue(&Q, w + 1);

}

}

}

}

void BFStree(Graph &G, int verID, tnode\*treelist)

{

int u;

seqQueue Q;

initialQueue(&Q);

treelist[A].data = G.Data[verID - 1];

treelist[A].parent = '!';

visitedd[verID - 1] = true;

enQueue(&Q, verID);

while (!empty(Q))

{

getFront(Q, u);

outQueue(&Q);

char B = G.Data[u-1];

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

{

if ((G.AdjMatrix[u - 1][w] >= 1) && (G.AdjMatrix[u - 1][w] < INF) && (!visitedd[w]))

{

A++;

treelist[A].parent = B;

treelist[A].data =G.Data[w];

visitedd[w] = true;

enQueue(&Q, w + 1);

}

}

}

}

void test1()

{

Graph G;

CreateGraphFromFile(G);

cout << "深度遍历:" << endl;

DFS(G, 1);

cout << endl;

cout <<"广度遍历:"<< endl;

BFS(G, 1);

cout << endl;

cout << "边的数目为：" << E / 4<< endl;

}

void test2()

{

Graph G;

CreateGraphFromFile(G);

DFStree(G, 1,treelist);

cout << "DFS双亲结点建树，右边为父亲结点" << endl;

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

{

cout << treelist[i].data << " ";

cout << treelist[i].parent << endl;

}

}

void test3()

{

Graph G;

CreateGraphFromFile(G);

BFStree(G, 1, treelist);

cout << "BFS双亲结点建树，右边为父亲结点" << endl;

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

{

cout << treelist[i].data << " ";

cout << treelist[i].parent << endl;

}

}

int main()

{

test1();

test2();

test3();

}