**课程号：314035040 班级 02 学号 2022141530052 姓名 林宸**

**实验3**

**实验题目：图的遍历操作**

**实验目的：**

1）进一步熟悉利用Code::Blocks，VC++等集成编译环境调试C/C++代码；

2）熟练掌握有向图和无向图的概念；

3）熟练掌握邻接矩阵和邻接链表建立图的存储结构；

4）熟练掌握图的深度优先遍历（DFS）与广度优先遍历算法（BFS）的实现。

**实验要求：**

阅读所提供的代码，根据提示补全缺少的部分，完成下面的功能：

采用邻接矩阵和邻接链表作为图的存储结构，完成有向图和无向图的DFS和BFS操作。

**实验主要步骤：**

1. 分析、理解给出的程序，根据提示填写缺少部分的代码；
2. 调试程序，并且编译通过；
3. 实现功能：采用邻接矩阵和邻接链表，要求输入的内容如下：

Input VertexNum(n) and EdgesNum(e): 8,9

Input Vertex string: 01234567

Input edges,Creat Adjacency Matrix

0 1

0 2

1 3

1 4

2 5

2 6

3 7

4 7

5 6

然后输出该图的DFS（深度优先遍历）和BFS（广度优先遍历）的结果。

4、完成思考题

**课后实验报告提交要求：**

1. 使用邻接矩阵的实现与使用邻接表的实现分别写在一个源文件(.c文件或者.cpp文件)中（即，一共两个源文件）；
2. 实验报告打印纸质版提交（11月27日）。
3. 将源文件与实验报告word文档放在一个以“学号+姓名”命名的文件夹中提交到邮箱(769674197@qq.com)，邮件命名为“第三次上机实践+学号+姓名”。**邮件提交时word文档中只需粘贴补全的代码及运行结果，无需填写心得体会，邮件提交截止日期为11月27日**。所有同学课后均需要提交实验报告。

程序代码（请同学们填写 **程序1 以及 程序2** 空缺部分的代码）:

**程序1：邻接矩阵作为存储结构**

#include"stdio.h"

#include"stdlib.h"

#define MaxVertexNum 100 //定义最大顶点数

typedef struct{

char vexs[MaxVertexNum]; //顶点表

int edges[MaxVertexNum][MaxVertexNum]; //邻接矩阵

int n, e; //图中的顶点数n和边数e

}MGraph;

//=========建立邻接矩阵=======

void CreatMGraph(MGraph &G)

{

int i,j,k;

char a;

printf("Input VertexNum(n) and EdgesNum(e): ");

scanf("%d,%d",&G.n,&G.e); //输入顶点数和边数

scanf("%c",&a);

printf("Input Vertex string:");

for(i=0;i<G.n;i++)

{

scanf("%c",&a);

G.vexs[i]=a; //读入顶点信息，建立顶点表

}

for(i=0;i<G.n;i++)

for(j=0;j<G.n;j++)

G.edges[i][j]=0; //初始化邻接矩阵

printf("Input edges,Creat Adjacency Matrix\n");

for(k=0;k<G.e;k++) { //读入e条边，建立邻接矩阵

scanf("%d%d",&i,&j); //输入边（Vi，Vj）的顶点序号

G.edges[i][j]=1;

G.edges[j][i]=1;

}

}

//=========定义标志向量，为全局变量=======

typedef enum{FALSE,TRUE} Boolean;

Boolean visited[MaxVertexNum];

//========DFS：深度优先遍历的递归算法======

void DFSM(MGraph & G,int i)

{

printf("%c",G.vexs[i]);  
visited[i]= *TRUE*;  
for(int j=0;j<G.n;j++) {  
 if (!visited[j]&&G.edges[i][j]) {  
 DFSM(G, j);  
 }  
}

}

void DFS(MGraph &G)

{

int i;

for(i=0;i<G.n;i++)

visited[i]=FALSE; //标志向量初始化

for(i=0;i<G.n;i++)

if(!visited[i]) //Vi未访问过

DFSM(G,i); //以Vi为源点开始DFS搜索

}

//===========BFS：广度优先遍历=======

void BFS(MGraph &G, int k)

{ //以Vk为源点对用邻接矩阵表示的图G进行广度优先搜索

int i,j,f=0,r=0;

int cq[MaxVertexNum]; //定义队列

for(i=0;i<G.n;i++)

visited[i]=FALSE; //标志向量初始化

for(i=0;i<G.n;i++)

cq[i]=-1; //队列初始化

printf("%c",G.vexs[k]); //访问源点Vk

visited[k]=TRUE;

cq[r]=k; //Vk已访问，将其入队。

while(cq[f]!=-1 && f<G.n) {

i=cq[f]; f=f+1;

for(j=0;j<G.n;j++)

if(!visited[j]&&G.edges[i][j]){  
 printf("%c",G.vexs[j]);  
 cq[r++]=j;  
 visited[j]=*TRUE*;  
}

}

}

//==========main=====

int main()

{

MGraph G;

CreatMGraph(G); //建立邻接矩阵

printf("Print Graph DFS: ");

DFS(G); //深度优先遍历

printf("\n");

printf("Print Graph BFS: ");

BFS(G,3); //以序号为3的顶点开始广度优先遍历

printf("\n");

}

**程序2：邻接链表作为存储结构**

#include"stdio.h"

#include"stdlib.h"

#define MaxVertexNum 50 //定义最大顶点数

typedef struct node{ //边表结点

int adjvex; //邻接点域

struct node \*next; //链域

}EdgeNode;

typedef struct vnode{ //顶点表结点

char vertex; //顶点域

EdgeNode \*firstedge; //边表头指针

}VertexNode;

typedef VertexNode AdjList[MaxVertexNum]; //AdjList是邻接表类型

typedef struct {

AdjList adjlist; //邻接表

int n,e; //图中当前顶点数和边数

} ALGraph; //图类型

//=========建立图的邻接表=======

void CreatALGraph(ALGraph &G)

{

int i,j,k;

char a;

EdgeNode \*s; //定义边表结点

printf("Input VertexNum(n) and EdgesNum(e): ");

scanf("%d,%d",&G.n,&G.e); //读入顶点数和边数

scanf("%c",&a);

printf("Input Vertex string:");

//建立边表

for(i=0;i<G.n;i++)

{

char c;  
scanf("%c",&c);  
VertexNode tp;  
tp.firstedge=**NULL**;  
tp.vertex=c;  
G.adjlist[i]=tp;

}

printf("Input edges,Creat Adjacency List\n");

for(k=0;k<G.e;k++) {

scanf("%d%d",&i,&j); //读入边（Vi，Vj）的顶点对序号

s=(EdgeNode \*)malloc(sizeof(EdgeNode)); //生成边表结点

s->adjvex=j; //邻接点序号为j

s->next=G.adjlist[i].firstedge;

G.adjlist[i].firstedge=s; //将新结点\*S插入顶点Vi的边表头部

s=(EdgeNode \*)malloc(sizeof(EdgeNode));

s=(EdgeNode \*)malloc(sizeof(EdgeNode));  
s->adjvex=i;  
s->next=G.adjlist[j].firstedge;  
G.adjlist[j].firstedge=s;

}

}

//=========定义标志向量，为全局变量=======

typedef enum{FALSE,TRUE} Boolean;

Boolean visited[MaxVertexNum];

//========DFS：深度优先遍历的递归算法======

void DFSM(ALGraph &G, int i)

{ //以Vi为出发点对邻接链表表示的图G进行DFS搜索

EdgeNode \*p;

printf("%c",G.adjlist[i].vertex); //访问顶点Vi

visited[i]=TRUE; //标记Vi已访问

p=G.adjlist[i].firstedge; //取Vi边表的头指针

while(p) { //依次搜索Vi的邻接点Vj，这里j=p->adjvex

int ne=p->adjvex;  
p=p->next;  
if(!visited[ne])DFSM(G,ne);

}

}

void DFS(ALGraph &G)

{

int i;

for(i=0;i<G.n;i++)

visited[i]=FALSE; //标志向量初始化

for(i=0;i<G.n;i++)

if(!visited[i]) //Vi未访问过

DFSM(G,i); //以Vi为源点开始DFS搜索

}

//==========BFS：广度优先遍历=========

void BFS(ALGraph &G, int k) { //以Vk为源点对用邻接链表表示的图G进行广度优先搜索

int i,f=0,r=0; EdgeNode \*p; int cq[MaxVertexNum]; //定义FIFO队列

for(i=0;i<G.n;i++)

visited[i]=FALSE; //标志向量初始化

for(i=0;i<=G.n;i++)

cq[i]=-1; //初始化标志向量

printf("%c",G.adjlist[k].vertex); //访问源点Vk

visited[k]=TRUE;

cq[r]=k; //Vk已访问，将其入队。注意，实际上是将其序号入队

while(cq[f]!=-1) { //队列非空则执行

i=cq[f]; f=f+1; //Vi出队

p=G.adjlist[i].firstedge; //取Vi的边表头指针

while(p) { //依次搜索Vi的邻接点Vj（令p->adjvex=j）

int ne=p->adjvex;  
p=p->next;  
if(!visited[ne]){  
 printf("%c",G.adjlist[ne].vertex);  
 visited[ne]=*TRUE*;  
 cq[r++]=ne;  
}

}

}//endwhile

}

//==========主函数===========

int main()

{

ALGraph G;

CreatALGraph(G);

printf("Print Graph DFS: ");

DFS(G);

printf("\n");

printf("Print Graph BFS: ");

BFS(G,3);

printf("\n");

}

**思考题：**

1. 根据实验步骤中提供的顶点与边分别画出对应无向图和有向图的示意图。

图示

描述已自动生成

1. 若要将**程序1和程序2**应用于有向图的遍历，则需要分别对代码做什么修改？

程序1中输入边时只需把[i][j]置为1,[j][i]依旧为0

程序2中生成边表时只需处理i到j的情形,相当于用原来的代码(无向图把j到i的情形按照i到j的照抄一遍即可)

1. 分析程序1和程序2中DFS和BFS（顶点数为n，边数为e）的时间复杂度，并指出决定算法时间复杂度的主要代码段。

1DFS:O(n^2),每个顶点访问一次,访问每个顶点时判断其余所有点是否为它的邻接点.

for(int j=0;j<G.n;j++) {  
 if (!visited[j]&&G.edges[i][j]) {  
 DFSM(G, j);  
 }  
}

1BFS:O(n^2) ,每个顶点访问一次,访问每个顶点时判断其余所有点是否为它的邻接点.

for(j=0;j<G.n;j++){  
 if(!visited[j]&&G.edges[i][j]){  
 printf("%c",G.vexs[j]);  
 cq[r++]=j;  
 visited[j]=*TRUE*;  
 }  
}

2DFS:O(n+2\*e),每个顶点访问一次,每条边访问两次(边的两个顶点被访问时都会循环遍历所有关联的边)

while(p) { *//依次搜索Vi的邻接点Vj，这里j=p->adjvex* int ne=p->adjvex;  
 p=p->next;  
 if(!visited[ne])DFSM(G,ne);  
}

2BFS:O(n+2\*e) ,每个顶点访问一次,每条边访问两次(边的两个顶点被访问时都会循环遍历所有关联的边)

while(p) { *//依次搜索Vi的邻接点Vj（令p->adjvex=j）* int ne=p->adjvex;  
 p=p->next;  
 if(!visited[ne]){  
 printf("%c",G.adjlist[ne].vertex);  
 visited[ne]=*TRUE*;  
 cq[r++]=ne;  
 }  
}

**实验结果：**

根据实验步骤中的提示，将屏幕输出内容截取，并粘贴到word文档中。

(输入采用重定向输入)

图形用户界面, 文本

中度可信度描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

**心得体会：**

**标题：图论实验心得体会：探索图的创建、存储与遍历算法**

**在数据结构与算法的学习中，图论是一个极为重要的领域，涵盖了图的创建、存储和遍历等多个方面。通过进行图的编程上机实验，我深刻体会到了图在计算机科学中的重要性，并对图的相关算法有了更为深入的理解。以下是我在实验中的心得体会：**

**### 图的创建与存储**

**实验的第一部分涉及图的创建与存储，主要包括邻接矩阵和邻接表两种方式。在实际编程中，我发现选择合适的存储结构对后续操作的效率产生了显著的影响。**

**1. \*\*邻接矩阵：\*\* 通过二维数组表示图的邻接关系，易于理解和实现。但对于稀疏图而言，会浪费大量空间。在实际应用中，邻接矩阵更适合密集图。**

**2. \*\*邻接表：\*\* 使用链表表示每个顶点的邻接点，对于稀疏图占用空间较小。在实验中，我更倾向于选择邻接表作为图的存储结构，尤其是对于大规模的图数据。**

**### 图的遍历**

**图的遍历是解决许多实际问题的基础，而深度优先搜索（DFS）和广度优先搜索（BFS）是两种常用的图遍历算法。在实验中，我对这两种算法的应用场景和实现原理有了更深入的了解。**

**1. \*\*DFS：\*\* 通过递归或栈实现，DFS更适合解决一些路径问题，如寻找从起点到终点的路径。在实验中，我发现DFS的递归实现方式清晰简洁，但需要注意递归深度，避免栈溢出。**

**2. \*\*BFS：\*\* 采用队列实现，BFS更适合解决最短路径问题，如找到起点到终点的最短路径。在实验中，我深刻体会到BFS的层次遍历特性，对于一些问题，BFS的效率更高。**

**### 实验总结**

**通过这次图论的编程实验，我不仅学到了图的创建与存储方法，还深入了解了DFS和BFS这两种重要的图遍历算法。在编写代码的过程中，我更加熟练地运用了数据结构和算法的知识，提高了对图问题的分析和解决能力。**

**此外，实验过程中的调试与优化也让我更加注重代码的健壮性和效率。对于图这种复杂的数据结构，编写清晰、高效的代码是至关重要的。**

**总的来说，这次图论实验让我在算法和数据结构的学习中迈出了重要的一步，为解决实际问题提供了更多的思路和方法。我相信这些在实验中积累的经验将对我的编程能力和算法思维产生深远的影响。**