

**数据结构课程设计报告**

**题目：有向加权图**

学 院 计算机学院

专 业 信息安全

年级班别 17级（2）班

学 号 3117005278

学生姓名 林俊涵

指导教师 李小妹

编 号 19

成 绩 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2019年 1月

**报告：**

**报告内容：** □详细　 □完整　 □基本完整 □不完整

**设计方案：** □非常合理　 □合理　 □基本合理 □较差

**算法实现：** □全部实现　 □基本实现　 □部分实现 □实现较差

**测试样例：** □完备　 □比较完备　 □基本完备 □不完备

**文档格式：** □规范　 □比较规范　 □基本规范 □不规范

**答辩：**

□理解题目透彻，问题回答流利

□理解题目较透彻，回答问题基本正确

□部分理解题目，部分问题回答正确

□未能完全理解题目，答辩情况较差

**总评成绩：**

□优　　　□良　　　□中　　　□及格　　　□不及格

**1题目：**

元素类型：字符

存储结构：邻接矩阵

抽象数据类型：有向加权图

ADT List{

**数据对象**：D＝{ ai | ai∈ElemSet, i=1,2,...,n, n≥0 }

**数据关系**：R1＝{ <ai-1, ai>|ai-1, ai∈D, i=2,...,n }

**基本操作**：

Status InitGraph(MGraph \*G, VexType \*vexs, int n);

//初始化含有N个顶点且无边的图

Status CreatGraph(MGraph \*G, VexType \*vex, int n, ArcInfo \*arcs, int e);

//创建n个顶点和e条边的图，vexs为顶点信息，arcs为边信息

int LocateVex(MGraph G, VexType v);

//用来获取顶点的位置

VexType GetVex(MGraph G, int k); //取图G的k顶点的值到w

Status PutVex(MGraph G, int k, VexType w);

//对图G的k顶点赋值w

int FirstAdjVex(MGraph G, int k);

//求图G中k顶点的第一个邻接顶点的位序

int NextAdjVex(MGraph G, int k, int m);

//m顶点为k顶点到m顶点的边或者弧，求图中K顶点相对于m顶点的下一个邻接顶点的位序

Status AddArc(MGraph G, int k, int m, int info);

//在图G中增加k到m的线，和权值

Status Remove(MGraph G, int k, int m); //删除图中k到m的线

Status DFS(MGraph G, int k); //连通图的深度优先遍历

Status DFSTraverse(MGraph G); //完整图的优先遍历

LQueue\* InitQueue(); //构造一个空队列

Status EnQueue(LQueue \*Q, int e); //在队列的队尾插入

Status DeQueue(LQueue \*Q, int \*e); //是队列Q的队头出队并且返回到e

Status BFS(MGraph G, Status(\*visit)(int)); //广度优先遍历

} ADT List

**2．存储结构定义**

公用头文件DS0.h:

#include <stdio.h>

#include <stdlib.h>

#define OK 1

#define ERROR -1

#define OVERFLOW -1

typedef int Status;

#define UNVISITED 0

#define VISITED 1

#define INFINITY 65535 //取计算机允许的最大值，正无穷

typedef char VexType;/\* 元素类型为字符类型\*/

1. 采用邻接矩阵存储结构：

/\*有向加权图的邻接数组存取结构 \*/

typedef struct {

VexType \*vexs; /\*顶点数组，VexType是顶点类型，由用户定义\*/

int \*\*arcs; /\*关系数组，对于有权图，它的值是权值或者INFINITY\*/

int n, e; /\*定义顶点数n和边数e\*/

int \*tags; /\*标志数组，可用于在图的遍历中标记顶点访问与否\*/

}MGraph;

/\*弧线的存储结构\*/

typedef struct {

VexType v, w; //边弧的端点

int info; //边弧的权值

}ArcInfo;

/\*下列队列用来辅助图的遍历\*/

/\*定义队列的节点\*/

typedef struct LQNode {

int data;

struct LQNode \*next;

}LQNode, \*QueuePtr;

/\*定义队列\*/

typedef struct {

QueuePtr front;

QueuePtr rear;

}LQueue;

**3. 算法设计**

Status InitGraph(MGraph \*G, VexType \*vexs, int n) {

/\*初始化含有N个顶点且无边的图\*/

int i, j;

if (n < 0 || (n > 0 && vexs == NULL)) return ERROR;

G->n = n;

G->e = 0;

if (0 == n) return OK;

if (NULL == (G->vexs = (VexType\*)malloc(n \* sizeof(VexType))))

return OVERFLOW;

for (i = 0; i < G->n; i++) G->vexs[i] = vexs[i];

if (NULL == (G->arcs = (int\*\*)malloc(n \* sizeof(int\*))))

return OVERFLOW;

for (i = 0; i < n; i++) {

if (NULL == (G->arcs[i] = (int\*)malloc(n \* sizeof(int))))

return OVERFLOW;

}

if (NULL == (G->tags = (int\*)malloc(n \* sizeof(int)))) return OVERFLOW;

for (i = 0; i < n; i++) {

G->tags[i] = UNVISITED;

for (j = 0; j < n; j++) {

G->arcs[i][j] = INFINITY;

}

}

return OK;

}

Status CreatGraph(MGraph \*G, VexType \*vex, int n, ArcInfo \*arcs, int e) {

//创建n个顶点和e条边的图，vexs为顶点信息，arcs为边信息

int i, j, k;

if (InitGraph(G, vex, n) != OK) return ERROR;

G->e = e;

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

i = LocateVex(\*G, arcs[k].v);

j = LocateVex(\*G, arcs[k].w);

if (i < 0 || j < 0) return ERROR;

G->arcs[i][j] = arcs[k].info;

}

return OK;

}

int LocateVex(MGraph G, VexType v) {

//用于获取顶点的位置

int i;

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

if (G.vexs[i] == v) return i;

}

return ERROR;

}

VexType GetVex(MGraph G, int k) {

//取图G的k顶点的值到w

if (k<0 || k>G.e) return;

return G.vexs[k];

}

Status PutVex(MGraph G, int k, VexType w) {

//对图G的k顶点赋值w

if (k<0 || k>G.e) return ERROR;

G.vexs[k] = w;

return OK;

}

int FirstAdjVex(MGraph G, int k) {

int i;

if (k < 0 || k >= G.n) return ERROR;

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

if (G.arcs[k][i] != INFINITY) return i;

}

return ERROR;

}

int NextAdjVex(MGraph G, int k, int m) {

m = m + 1;

// //m顶点为k顶点到m顶点的边或者弧，求图中K顶点相对于m顶点的下一个邻接顶点的位序

if (k < 0 || k >= G.n || m<0 || m >= G.n || m <= k) return ERROR;

for (; m<G.n; m++)

{

if (G.arcs[k][m] != INFINITY) return m;

}

return ERROR;

}

Status AddArc(MGraph G, int k, int m, int info) {

//在图G中增加k到m的线，和权值

if (k < 0 || k >= G.n || m < 0 || m >= G.n || m <= k) return ERROR;

if (G.arcs[k][m] == INFINITY)

{

G.arcs[k][m] = info;

}

else

return ERROR;

}

Status Remove(MGraph G, int k, int m) {

//删除图中k到m的线

if (k < 0 || k >= G.n || m < 0 || m >= G.n || m <= k) return ERROR;

if (G.arcs[k][m] != INFINITY)

{

G.arcs[k][m] = INFINITY;

}

else

return ERROR;

}

Status DFS(MGraph G, int k) {

//连通图的深度优先遍历

int i;

if (k < 0 || k >= G.n) return ERROR;

printf("%c", G.vexs[k]);

G.tags[k] = VISITED;

for (i = FirstAdjVex(G, k); i >= 0; i = NextAdjVex(G, k, i)) {

if (UNVISITED == G.tags[i])

if (ERROR == DFS(G, i))

return ERROR;

}

return OK;

}

Status DFSTraverse(MGraph G) {

//完整图的优先遍历

int i = 0;

for (i = 0; i < G.n; i++) G.tags[i] = UNVISITED;

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

if (UNVISITED == G.tags[i])

if (ERROR == DFS(G, i)) return ERROR;

}

return OK;

}

LQueue\* InitQueue() {

//构造一个空队列

LQueue \*Q = (LQueue \*)malloc(sizeof(LQueue));

if (Q == NULL) return ERROR;

Q->front = NULL;

Q->rear = NULL;

return Q;

}

Status EnQueue(LQueue \*Q, int e) {

//在队列的队尾插入e

LQNode \*p;

p = (LQNode\*)malloc(sizeof(LQNode));

if (p == NULL) return ERROR;

p->data = e;

p->next = NULL;

if (NULL == Q->front) {

Q->front = p;

Q->rear = p;

}

else

{

Q->rear->next = p;

Q->rear = p;

}

return OK;

}

Status DeQueue(LQueue \*Q, int \*e) {

//是队列Q的队头出队并且返回到e

LQNode \*p;

if (NULL == Q->front) return ERROR;

p = Q->front;

\*e = p->data;

Q->front = p->next;

if (Q->rear == p) Q->rear = NULL;

free(p);

return OK;

}

Status BFS(MGraph G) {

//广度优先遍历

int i, j, k;

int \*p\_k = &k;

LQueue \*Q = InitQueue();

for (i = 0; i < G.n; i++) G.tags[i] = UNVISITED;

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

if (UNVISITED == G.tags[i]) {

printf("%c", G.vexs[i]);

G.tags[i] = VISITED;

EnQueue(Q, i);

while (DeQueue(Q, p\_k) == OK)

{

for (j = FirstAdjVex(G, k); j >= 0; j = NextAdjVex(G, k, j))

if (UNVISITED == G.tags[j] && j >= 0) {

printf("%c", G.vexs[j]);

G.tags[j] = VISITED;

EnQueue(Q, j);

}

}

}

}

return OK;

}

**4．测试**

void main() {

printf("有向加权图测试：\n");

MGraph M;

MGraph \*p\_M = &M;

VexType vexs[5] = { 'A','C','R','F','E' };

int n = 5;

int k = 3;;

ArcInfo \*p = (ArcInfo\*)malloc(3 \* sizeof(ArcInfo));

p[0].v = 'A';

p[0].w = 'R';

p[0].info = 32;

p[1].w = 'F';

p[1].v = 'A';

p[1].info = 34;

p[2].v = 'R';

p[2].w = 'E';

p[2].info = 45;

VexType w = 'c';

CreatGraph(p\_M, vexs, n, p, 3);

printf("图的顶点如下：\n");

for (int s = 0; s < M.n; s++) {

printf("\t\t%c", M.vexs[s]);

}

printf("\n图的关系如下：\n");

int h, s;

for (h = 0; h < M.n; h++) {

printf("%c：", M.vexs[h]);

for (s = 0; s < M.n; s++)

{

printf("%d\t\t", M.arcs[h][s]);

}

printf("\n");

}

printf("\n获取指定位置顶点元素的测试：\n 位标1的元素是：");

w = GetVex(M, 1);

printf("%c\n", w);

printf("\n将指定元素放入指定位标测试（将k放到o号位）:\n");

PutVex(M, 0, 'K');

for (int s = 0; s < M.n; s++) {

printf("\t\t%c", M.vexs[s]);

}

printf("\n获取指定位标的第一个邻接点元素测试（0号位标的第一个邻接点）：\n");

int i = FirstAdjVex(M, 0);

printf("%c\t下一个邻接点是：", M.vexs[i]);

i = NextAdjVex(M, 0, 2);

printf("%c\n", M.vexs[i]);

printf("在1和2之间增加连线，并加权20:\n");

AddArc(M, 1, 2, 20);

for (h = 0; h < M.n; h++) {

printf("%c：", M.vexs[h]);

for (s = 0; s < M.n; s++)

{

printf("%d\t\t", M.arcs[h][s]);

}

printf("\n");

}

printf("深度优先遍历结果:\n");

DFSTraverse(M, 0);

printf("\n");

printf("广度优先遍历结果:\n");

BFS(M);

printf("\n\n");

printf("移除2号和4号之间的连线：\n");

Remove(M, 2, 4);

for (h = 0; h < M.n; h++) {

printf("%c：", M.vexs[h]);

for (s = 0; s < M.n; s++)

{

printf("%d\t\t", M.arcs[h][s]);

}

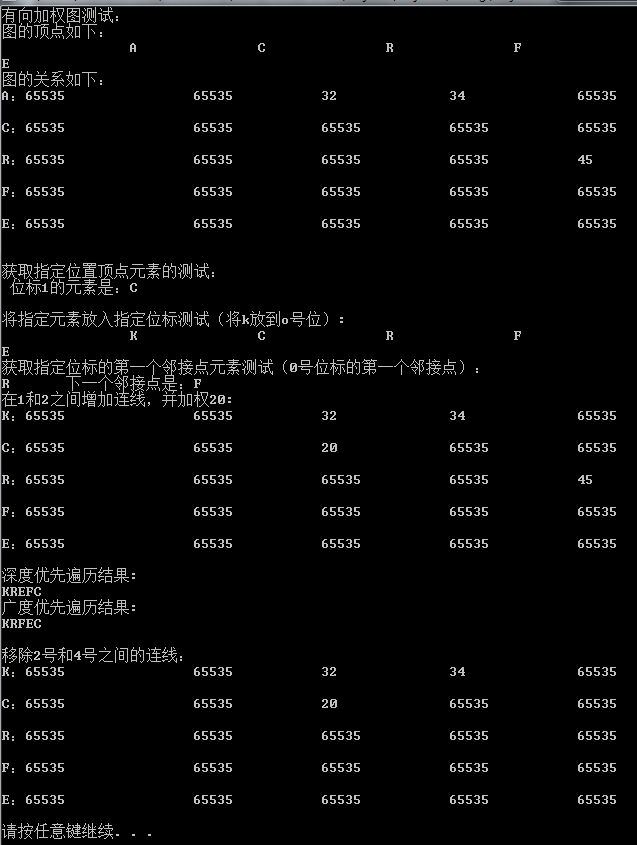
printf("\n");

}

system("pause");

}

测试结果：



**5．三种存储结构的比较**

|  |  |  |
| --- | --- | --- |
|  | 基本操作 | 时间复杂度 |
| 邻接矩阵图的基本操作 | InitGraph(MGraph \*G, VexType \*vexs, int n) | ***O***(n^2) |
| CreatGraph(MGraph \*G, VexType \*vex, int n, ArcInfo \*arcs, int e) | ***O***(n^2) |
| LocateVex(MGraph G, VexType v) | ***O***(n) |
| GetVex(MGraph G, int k) | ***O***(1) |
| PutVex(MGraph G, int k, VexType w) | ***O***(1) |
| FirstAdjVex(MGraph G, int k) | ***O***(n) |
| NextAdjVex(MGraph G, int k, int m) | ***O***(n) |
| AddArc(MGraph G, int k, int m, int info) | ***O***(1) |
| Remove(MGraph G, int k, int m) | ***O***(1) |
| DFS(MGraph G, int k) | ***O***(n^2) |
| DFSTraverse(MGraph G) | ***O***(n^2) |

六、思考和总结

（1）在算法设计时，要注意判断有关参数值的可用性，避免访问一些没有指向内存的指针内容。

（2）在实行相关操作时需要注意指针是否为空的情况，增加程序的健壮性。

（3） 运行测试时要考虑测试的全面性，使用一些极端情况，才能发现程序隐藏的bug。