数据结构实验报告

（2017-2018第一学期）

**实验题目：** 实验六 图及其应用

**学 号：**  20151305212

**姓 名：**  崔健骅

**专业年级：** 信息安全16-1

**指导教师：** 崔青

**实验日期：**  2017-12-1

实验六 图及其应用

**实验目的：**

1. 理解图的基本概念，掌握用邻接矩阵和邻接表的方法描述图的存储结构。
2. 掌握在邻接链表存储结构下的图的深度优先递归遍历，广度优先遍历。
3. 应用图的结构解决具体问题。

**实验内容：**

1. 建立二建立一个包含6个结点的无向图的邻接矩阵或邻接表，实现插入、删 除边的功能，并实现该图的深度优先搜索遍历和广度优先搜索遍历。

程序源代码：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define MAXVEX 100

#define MAXSIZE 100

int visited[MAXSIZE];

typedef int datatype;

typedef int Status;

typedef char VertexType;

typedef int EdgeType;

typedef struct SQueue{

datatype data[MAXSIZE];

int front;

int rear;

}SQueue;

typedef struct EdgeNode {

int adjvex;

struct EdgeNode \*next;

}EdgeNode;

typedef struct VertexNode {

VertexType data;

struct EdgeNode \*firstedge;

}VertexNode;

typedef struct

{

VertexNode AdjList[MAXVEX];

int NumOfVertexes;

int NumOfEdges;

}GraphAdjList;

void Menu(GraphAdjList \*G);

void InitQueue(SQueue \*Q);

int IsEmpty(SQueue \*Q);

Status EnQueue(SQueue \*Q, datatype x);

Status DeQueue(SQueue \*Q, datatype \*x);

void CreateGraph(GraphAdjList \*G);

void DFS(GraphAdjList \*G, int i);

void DFSTraverse(GraphAdjList \*G);

void BFSTraverse(GraphAdjList \*G);

Status DeleteEdge(GraphAdjList \*G, int i, int j);

Status InsertEdge(GraphAdjList \*G, int i, int j);

int main(void)

{

GraphAdjList G;

Menu(&G);

return 0;

}

void Menu(GraphAdjList \*G)

{

char cmd;

const char \*legal="cCiIrRdDbBqQ";

int i, j;

do

{

putchar('\n');

printf("c,C 创建\n");

printf("i,I 插入边\n");

printf("r,R 删除边\n");

printf("d,D 深度优先遍历\n");

printf("b,B 广度优先遍历\n");

printf("q,Q 退出\n");

putchar('\n');

do

{

printf("cmd: ");

fflush(stdin);

scanf("%c", &cmd);

}

while( strchr(legal, cmd)==NULL );

putchar('\n');

switch(cmd)

{

case 'c':

case 'C':

CreateGraph(G);

break;

case 'i':

case 'I':

printf("Input the Edge: ");

scanf("%d%d", &i, &j);

InsertEdge(G, i, j);

printf("DFS: ");

DFSTraverse(G);

printf("BFS: ");

BFSTraverse(G);

break;

case 'r':

case 'R':

printf("Input the Deleted Edge: ");

scanf("%d%d", &i, &j);

DeleteEdge(G, i, j);

printf("DFS: ");

DFSTraverse(G);

printf("BFS: ");

BFSTraverse(G);

break;

case 'd':

case 'D':

DFSTraverse(G);

break;

case 'b':

case 'B':

BFSTraverse(G);

break;

}

}

while( (cmd!='q') && (cmd!='Q') );

}

void InitQueue(SQueue \*Q)

{

Q->front = 0;

Q->rear = 0;

}

int IsEmpty(SQueue \*Q)

{

return Q->rear==Q->front;

}

Status EnQueue(SQueue \*Q, datatype x)

{

if((Q->rear+1) % MAXSIZE == Q->front)

return ERROR;

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

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

return OK;

}

Status DeQueue(SQueue \*Q, datatype \*x)

{

if(Q->front==Q->rear)

return ERROR;

\*x = Q->data[Q->front];

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

return OK;

}

void CreateGraph(GraphAdjList \*G)

{

int i, j, k;

EdgeNode \*e;

printf("Input the Num of Vertexes and Edges: ");

scanf("%d%d", &(G->NumOfVertexes), &(G->NumOfEdges));

for(i=0; i<G->NumOfVertexes; i++)

{

printf("Input the Data of the %dth Vertex: ", i+1);

fflush(stdin);

scanf("%c", &(G->AdjList[i].data));

G->AdjList[i].firstedge = NULL;

}

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

{

printf("Input the %dth Edge: ", k+1);

scanf("%d%d", &i, &j);

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

e->adjvex = j;

e->next = G->AdjList[i].firstedge;

G->AdjList[i].firstedge = e;

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

e->adjvex = i;

e->next = G->AdjList[j].firstedge;

G->AdjList[j].firstedge = e;

}

}

void DFS(GraphAdjList \*G, int i)

{

EdgeNode \*p;

visited[i] = TRUE;

printf("%c ", G->AdjList[i].data);

p = G->AdjList[i].firstedge;

while(p)

{

if(!visited[p->adjvex])

DFS(G, p->adjvex);

p = p->next;

}

}

void DFSTraverse(GraphAdjList \*G)

{

int i;

for(i=0; i<G->NumOfVertexes; i++)

visited[i] = FALSE;

for(i=0; i<G->NumOfVertexes; i++)

if(!visited[i]) DFS(G, i);

putchar('\n');

}

void BFSTraverse(GraphAdjList \*G)

{

int i;

EdgeNode \*p;

SQueue Q;

for(i=0; i<G->NumOfVertexes; i++)

visited[i] = FALSE;

InitQueue(&Q);

for(i=0; i<G->NumOfVertexes; i++)

{

if(!visited[i])

{

visited[i] = TRUE;

printf("%c ", G->AdjList[i].data);

EnQueue(&Q, i);

while(!IsEmpty(&Q))

{

DeQueue(&Q, &i);

p = G->AdjList[i].firstedge;

while(p)

{

if(!visited[p->adjvex])

{

visited[p->adjvex] = TRUE;

printf("%c ", G->AdjList[p->adjvex].data);

EnQueue(&Q, p->adjvex);

}

p = p->next;

}

}

}

}

putchar('\n');

}

Status DeleteEdge(GraphAdjList \*G, int i, int j)

{

EdgeNode \*p=NULL;

EdgeNode \*tmp=NULL;

if(i>=G->NumOfVertexes || j>=G->NumOfVertexes)

{

printf("At Least One Vertex Doesn't Exist!");

return ERROR;

}

p = G->AdjList[i].firstedge;

if(p->adjvex==j)

{

G->AdjList[i].firstedge = p->next;

free(p);

return OK;

}

while(p && (p->next) && ((p->next->adjvex)!=j))

p = p->next;

if(p==NULL || p->next==NULL)

{

printf("The Edge Doesn't Exist!\n");

return ERROR;

}

tmp = p->next;

p->next = tmp->next;

free(tmp);

p = G->AdjList[j].firstedge;

if(p->adjvex==i)

{

G->AdjList[j].firstedge = p->next;

free(p);

return OK;

}

while(p && (p->next) && ((p->next->adjvex)!=i))

p = p->next;

if(p==NULL || p->next==NULL)

{

printf("The Edge Doesn't Exist!\n");

return ERROR;

}

tmp = p->next;

p->next = tmp->next;

free(tmp);

return OK;

}

Status InsertEdge(GraphAdjList \*G, int i, int j)

{

EdgeNode \*tmp=NULL;

if(i>=G->NumOfVertexes || j>=G->NumOfVertexes)

{

printf("At Least One Vertex Doesn't Exist!");

return ERROR;

}

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

tmp->adjvex = j;

tmp->next = G->AdjList[i].firstedge;

G->AdjList[i].firstedge = tmp;

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

tmp->adjvex = i;

tmp->next = G->AdjList[j].firstedge;

G->AdjList[j].firstedge = tmp;

return OK;

}

调试分析：

运行结果1:

c,C 创建

i,I 插入边

r,R 删除边

d,D 深度优先遍历

b,B 广度优先遍历

q,Q 退出

cmd: c

Input the Num of Vertexes and Edges: 6 6

Input the Data of the 1th Vertex: a

Input the Data of the 2th Vertex: b

Input the Data of the 3th Vertex: f

Input the Data of the 4th Vertex: g

Input the Data of the 5th Vertex: i

Input the Data of the 6th Vertex: h

Input the 1th Edge: 0 1

Input the 2th Edge: 0 2

Input the 3th Edge: 1 3

Input the 4th Edge: 1 4

Input the 5th Edge: 2 3

Input the 6th Edge: 3 5

cmd: i

Input the Edge: 0 3

DFS: a g h f b i

BFS: a g f b h i

cmd: i

Input the Edge: 3 4

DFS: a g i b h f

BFS: a g f b i h

cmd: r

Input the Deleted Edge: 0 3

DFS: a f g i b h

BFS: a f b g i h

cmd: r

Input the Deleted Edge: 3 4

DFS: a f g h b i

BFS: a f b g i h

cmd: q

运行结果2:

2）用图实现校园导游程序，即校园里给定任意两个位置，编程求这两个位置间 的最短路径。

程序源代码：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAXVEX 100

#define INFINITY 65535

typedef char VertexType[20];

typedef int EdgeType;

typedef int Path[MAXVEX][MAXVEX];

typedef int ShortPathTable[MAXVEX][MAXVEX];

typedef struct

{

VertexType vexs[MAXVEX];

EdgeType arc[MAXVEX][MAXVEX];

int NumOfVertexes;

int NumOfEdges;

}MGraph;

void Menu(MGraph \*G, Path \*P, ShortPathTable \*D);

void CreateGraph(MGraph \*G);

void Floyd(MGraph \*G, Path \*P, ShortPathTable \*D);

void Search(MGraph G, Path P, ShortPathTable D, char \*Source, char \*Dest);

int main(void)

{

MGraph G;

Path P;

ShortPathTable D;

Menu(&G, &P, &D);

return 0;

}

void Menu(MGraph \*G, Path \*P, ShortPathTable \*D)

{

char cmd;

char Source[20];

char Dest[20];

const char \*legal = "cCsSqQ";

do

{

putchar('\n');

printf("c,C 创建\n");

printf("s,S 查询\n");

printf("q,Q 退出");

putchar('\n');

do

{

printf("cmd: ");

fflush(stdin);

scanf("%c", &cmd);

}

while( strchr(legal, cmd)==NULL );

putchar('\n');

switch(cmd)

{

case 'c':

case 'C':

CreateGraph(G);

Floyd(G, P, D);

break;

case 's':

case 'S':

printf("Input the Source: ");

scanf("%s", Source);

printf("Input the Dest: ");

scanf("%s", Dest);

Search(\*G, \*P, \*D, Source, Dest);

break;

}

}

while( (cmd!='q') && (cmd!='Q') );

}

void CreateGraph(MGraph \*G)

{

int i, j, k, w;

printf("Input the Num of Vertexes and Edges: ");

scanf("%d%d", &(G->NumOfVertexes), &(G->NumOfEdges));

for(i=0; i<G->NumOfVertexes; i++)

{

printf("Input the Data of the %dth Vertex: ", i+1);

scanf("%s", G->vexs[i]);

}

for(i=0; i<G->NumOfVertexes; i++)

for(j=0; j<G->NumOfVertexes; j++)

if(i!=j) G->arc[i][j] = INFINITY;

else G->arc[i][j] = 0;

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

{

printf("Input the %dth Edge and its weight: ", k+1);

scanf("%d%d%d", &i, &j, &w);

G->arc[i][j] = G->arc[j][i] = w;

}

}

void Floyd(MGraph \*G, Path \*P, ShortPathTable \*D)

{

int v, w, k;

for(v=0; v<G->NumOfVertexes; v++)

{

for(w=0; w<G->NumOfVertexes; w++)

{

(\*D)[v][w] = G->arc[v][w];

(\*P)[v][w] = w;

}

}

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

{

for(v=0; v<G->NumOfVertexes; v++)

{

for(w=0; w<G->NumOfVertexes; w++)

{

if( (\*D)[v][w] > (\*D)[v][k]+(\*D)[k][w] )

{

(\*D)[v][w] = (\*D)[v][k]+(\*D)[k][w];

(\*P)[v][w] = (\*P)[v][k];

}

}

}

}

}

void Search(MGraph G, Path P, ShortPathTable D, char \*Source, char \*Dest)

{

int v, w, k;

for(v=0; v<G.NumOfVertexes; v++)

if(strcmp(Source, G.vexs[v]) == 0) break;

for(w=0; w<G.NumOfVertexes; w++)

if(strcmp(Dest, G.vexs[w]) == 0) break;

if(v==G.NumOfVertexes+1 || w=G.NumOfVertexes+1)

{

printf("The Source or Dest Doesn't Exist\n");

return ;

}

printf("%s-%s length: %d\n", G.vexs[v], G.vexs[w], D[v][w]);

k = P[v][w];

printf("Path: %s", G.vexs[v]);

while(k!=w)

{

printf("->%s", G.vexs[k]);

k = P[k][w];

}

printf("->%s\n", G.vexs[w]);

putchar('\n');

}

调试分析：

运行结果1:

c,C 创建

s,S 查询

q,Q 退出

cmd: c

Input the Num of Vertexes and Edges: 8 9

Input the Data of the 1th Vertex: 正门

Input the Data of the 2th Vertex: 教学楼群

Input the Data of the 3th Vertex: 11号教学楼

Input the Data of the 4th Vertex: 图书馆

Input the Data of the 5th Vertex: 17号宿舍楼

Input the Data of the 6th Vertex: 餐饮广场

Input the Data of the 7th Vertex: 科技楼

Input the Data of the 8th Vertex: 红湖

Input the 1th Edge and its weight: 0 1 159

Input the 2th Edge and its weight: 1 3 224

Input the 3th Edge and its weight: 2 3 99

Input the 4th Edge and its weight: 0 3 231

Input the 5th Edge and its weight: 1 5 133

Input the 6th Edge and its weight: 4 5 79

Input the 7th Edge and its weight: 5 6 318

Input the 8th Edge and its weight: 2 6 128

Input the 9th Edge and its weight: 6 7 139

cmd: s

Input the Source: 正门

Input the Dest: 17号宿舍楼

正门-17号宿舍楼 length: 371

Path: 正门->教学楼群->餐饮广场->17号宿舍楼

cmd: s

Input the Source: 17号宿舍楼

Input the Dest: 11号教学楼

17号宿舍楼-11号教学楼 length: 525

Path: 17号宿舍楼->餐饮广场->科技楼->11号教学楼

cmd: s

Input the Source: 图书馆

Input the Dest: 红湖

图书馆-红湖 length: 366

Path: 图书馆->11号教学楼->科技楼->红湖

cmd: s

Input the Source: 教学楼群

Input the Dest: 红湖

教学楼群-红湖 length: 590

Path: 教学楼群->图书馆->11号教学楼->科技楼->红湖

cmd: q

运行结果2：

实验心得体会：

1. 对于使用邻接表表示的图进行边的删除时，若链表未使用头结点，则要注意对 链表的第一个结点进行单独处理。
2. 由于链表中插入结点使用头插法，故经过相逆的删除、插入相同边的操作后， 图的遍历结果可能不同。