# 附录 基于链表的树、图实现的源程序

## 1.基于链表的建图

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

//用四个链表来完成此关卡

typedef struct Node //双向链表节点

{

char name[20];

double distance;

struct Node\* pre;

struct Node\* next;

}Node,\*NodePtr;

typedef struct DoubleLinklist //双向循环链表

{

int line; //地铁线路号

NodePtr head;

NodePtr end;

}DoubleLinklist,\*DoubleLinklistPtr;

DoubleLinklistPtr CreatNew(int line) //输入地铁线路号 ,创建一个空的双向链表

{

DoubleLinklistPtr q=(DoubleLinklistPtr)malloc(sizeof(DoubleLinklist));

q->line=line;

NodePtr head=(NodePtr)malloc(sizeof(Node));

head->distance=0;

head->next=NULL;

head->pre==NULL;

q->head=head;

q->end=head;

return q;

}

void AddNode(DoubleLinklistPtr q,char name[20],double distance) //双向链表中加入节点

{

NodePtr p=(NodePtr)malloc(sizeof(Node));

strcpy(p->name,name);

p->distance=distance;

q->end->next=p;

p->pre=q->end;

p->next=q->head;

q->head->pre=p;

q->end=p;

}

void read(DoubleLinklistPtr q) //将数据读入q

{

char name[20];

double distance;

while(1)

{

scanf("%s %lf",name,&distance);

AddNode(q,name,distance); //读入节点

if(distance==0) break;

}

}

void Display(DoubleLinklistPtr q) //输出数据

{

NodePtr head=q->head;

NodePtr temp=head->next;

printf("%d ",q->line);

while(temp!=head)

{

if(temp->next==head) //判断是否到表尾，若是则不输出距离0

{

printf("%s\n",temp->name);

break;

}

printf("%s %.2f ",temp->name,temp->distance);

temp=temp->next;

}

}

int main()

{

int n,line; //有n条线路,线路名字为line

scanf("%d",&n);

DoubleLinklistPtr q[n];

int i=0;

while(i<n)

{

scanf("%d",&line);

q[i]=CreatNew(line); //依次将n条线路读入n条双向链表中

read(q[i]);

i++;

}

i=0;

while(i<n)

{

Display(q[i]); //依次输出n条线路

i++;

}

return 0;

}

## 2.增加站点，删除

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

//用四个链表来完成此关卡

typedef struct Node //双向链表节点

{

char name[20];

double distance;

struct Node\* pre;

struct Node\* next;

}Node,\*NodePtr;

typedef struct DoubleLinklist //双向循环链表

{

int line; //地铁线路号

NodePtr head;

NodePtr end;

}DoubleLinklist,\*DoubleLinklistPtr;

DoubleLinklistPtr CreatNew(int line) //输入地铁线路号 ,创建一个空的双向链表

{

DoubleLinklistPtr q=(DoubleLinklistPtr)malloc(sizeof(DoubleLinklist));

q->line=line;

NodePtr head=(NodePtr)malloc(sizeof(Node));

head->distance=0;

head->next=NULL;

head->pre==NULL;

q->head=head;

q->end=head;

return q;

}

void AddNode(DoubleLinklistPtr q,char name[20],double distance) //双向链表中加入节点

{

NodePtr p=(NodePtr)malloc(sizeof(Node));

strcpy(p->name,name);

p->distance=distance;

q->end->next=p;

p->pre=q->end;

p->next=q->head;

q->head->pre=p;

q->end=p;

}

void read(DoubleLinklistPtr q) //将数据读入q

{

char name[20];

double distance;

while(1)

{

scanf("%s %lf",name,&distance);

AddNode(q,name,distance); //读入节点

if(distance==0) break;

}

}

void Display(DoubleLinklistPtr q) //输出数据

{

NodePtr head=q->head;

NodePtr temp=head->next;

printf("%d ",q->line);

while(temp!=head)

{

if(temp->next==head) //判断是否到表尾，若是则不输出距离0

{

printf("%s\n",temp->name);

break;

}

printf("%s %.2f ",temp->name,temp->distance);

temp=temp->next;

}

}

DoubleLinklistPtr DeleteStation(DoubleLinklistPtr q,int\* flag)

{

DoubleLinklistPtr qNew=CreatNew(q->line);

char a[20];

\*flag=0; //判断是否删除了值 ,flag为0没有删除

scanf("%s",a);

NodePtr temp=q->head->next;

while(temp!=q->head)

{

if(!strcmp(temp->name,a)) //如果相等就遍历下一个，不加入到qNew中

{

temp=temp->next;

\*flag=1; //标志找到了要删除的站 ，flag为1删除成功

}

if(!strcmp(temp->next->name,a)) //如果下个站点要删去，则这个站点的距离需要修改

{

AddNode(qNew,temp->name,temp->distance+temp->next->distance);

}

else AddNode(qNew,temp->name,temp->distance);

temp=temp->next;

}

return qNew;

}

DoubleLinklistPtr AddStation(DoubleLinklistPtr q,int\* flag)

{

DoubleLinklistPtr qNew=CreatNew(q->line);

\*flag=2; //flag=0删除失败；flag=1删除成功或增加成功；flag=2；flag=3增加失败，已有同名站点

float a,b;

char c[20],d[20];

NodePtr temp=q->head->next;

scanf("%f%f",&a,&b);

if(a==0) //即可能在第一个位置加入站点

{

scanf("%s",c) ;//将可能加入的站点读入c

while(temp!=q->head)

{

if(!strcmp(temp->name,c))

{

\*flag=3; //增加失败，已有同名站点,将flag设置为3.直接返回

return qNew;

}

AddNode(qNew,temp->name,temp->distance);

temp=temp->next;

}

\*flag=1;

NodePtr p=(NodePtr)malloc(sizeof(Node)); //没有同名站点，开始插入操作 ,且将flag设置为1

strcpy(p->name,c);

p->distance=b;

temp=qNew->head->next;

p->next=temp;

p->pre=qNew->head;

temp->pre=p;

qNew->head->next=p; //flag=1,插入成功，正常返回

return qNew;

}

else

{

scanf("%s%s",c,d); //在c站点后加入可能添加的d站点

while(temp!=q->head)

{

if(!strcmp(temp->name,d))

{

\*flag=3; //增加失败，已有同名站点,将flag设置为3.直接返回

return qNew;

}

temp=temp->next;

}

temp=q->head->next;

while(temp!=q->head) //先将原路线复制到qNew中

{

AddNode(qNew,temp->name,temp->distance);

temp=temp->next;

}

temp=qNew->head->next;

while(temp!=qNew->head) //再在qNew中做插入操作

{

if(!strcmp(temp->name,c)) //找到了c

{

\*flag=1;

temp->distance=a;

NodePtr p=(NodePtr)malloc(sizeof(Node)); //开始插入操作

strcpy(p->name,d);

p->distance=b;

NodePtr temp2=temp->next; //即在temp和temp2中插入一个p

p->next=temp2;

p->pre=temp;

temp->next=p;

temp2->pre=p; //flag=1,插入成功，正常返回

return qNew;

}

temp=temp->next;

} //flag为2，没有找到c

return qNew;

}

}

void Display\_qNew(DoubleLinklistPtr qNew,int flag) //flag=0删除失败；flag=1删除成功或增加成功；

{ //flag=2，没有与输入的增加位置前一站点同名的站点，flag=3增加失败，已有同名站点

if(flag==0) printf("删除失败，没有同名站点");

if(flag==1) Display(qNew);

if(flag==2) printf("增加失败，没有与输入的增加位置前一站点同名的站点");

if(flag==3) printf("增加失败，已有同名站点");

}

int main()

{

int n,line,flag; //有n条线路,线路名字为line,输出的几种情况用flag标识

char pattern[20];

scanf("%d",&n);

DoubleLinklistPtr q[n];

int i=0;

while(i<n)

{

scanf("%d",&line);

q[i]=CreatNew(line); //依次将n条线路读入n条双向链表中

read(q[i]);

i++;

}

scanf("%s%d",pattern,&line);

DoubleLinklistPtr qNew=NULL; //用一个qNew链表保存被改动后的那条线路的结果

for(i=0;i<n;i++) //找到对应line的的数组，然后赋给qNew传入函数中

{

if(q[i]->line==line)

{

qNew=q[i];

break;

}

}

if(!strcmp(pattern,"add")) qNew=AddStation(qNew,&flag);

if(!strcmp(pattern,"delete")) qNew=DeleteStation(qNew,&flag);

i=0;

while(i<n)

{

Display(q[i]); //依次输出n条线路

i++;

}

Display\_qNew(qNew,flag); //输出新的修改后的线路

return 0;

}

## 3.从指定站点出发，计算到另一站点的最短距离和途径站点序列

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define MAXVEX 21

#define INFINITY 65535

//用四个链表来完成此关卡

typedef struct Node //双向链表节点

{

char name[20];

double distance;

struct Node\* pre;

struct Node\* next;

}Node,\*NodePtr;

typedef struct DoubleLinklist //双向循环链表

{

int line; //地铁线路号

NodePtr head;

NodePtr end;

}DoubleLinklist,\*DoubleLinklistPtr;

typedef char VertexType[20]; //顶点类型为字符串，因为要传入站名

typedef float EdgeType;

typedef struct //用邻接矩阵建立图

{

VertexType vexs[MAXVEX];

EdgeType arc[MAXVEX][MAXVEX];

int numVertexes,numEdges;

}MGraph;

typedef int Patharc[MAXVEX]; //用于存储最短路径下标的数组

typedef float ShortPathTable[MAXVEX]; //用于存储到各点最短路径的权值和

DoubleLinklistPtr CreatDoubleLinklist(int line) //输入地铁线路号 ,创建一个空的双向链表

{

DoubleLinklistPtr q=(DoubleLinklistPtr)malloc(sizeof(DoubleLinklist));

q->line=line;

NodePtr head=(NodePtr)malloc(sizeof(Node));

head->distance=0;

head->next=NULL;

head->pre==NULL;

q->head=head;

q->end=head;

return q;

}

void AddNode(DoubleLinklistPtr q,char name[20],double distance) //双向链表中加入节点

{

NodePtr p=(NodePtr)malloc(sizeof(Node));

strcpy(p->name,name);

p->distance=distance;

q->end->next=p;

p->pre=q->end;

p->next=q->head;

q->head->pre=p;

q->end=p;

}

void read(DoubleLinklistPtr q) //将数据读入q

{

char name[20];

double distance;

while(1)

{

scanf("%s %lf",name,&distance);

AddNode(q,name,distance); //读入节点

if(distance==0) break;

}

}

void transfer(int\* i,int\* j,char a[],char b[],VertexType vexs[]) //将a，b在顶点表中的下标找到，并以i，j返回

{

int k=0;

for(k=0;k<MAXVEX;k++)

{

if(!strcmp(a,vexs[k])) \*i=k;

if(!strcmp(b,vexs[k])) \*j=k;

}

}

void CreatMGraph(MGraph \*G,DoubleLinklistPtr q[]) //将四个链表创立一个图

{

G->numEdges=0;

G->numVertexes=0;

int i,j,k;

int flag=1; //flag=0则表示之前有这个站了，不用加入顶点表

//建立顶点表

for(i=0;i<MAXVEX;)

{

for(j=0;j<4;j++) //依次将四个链表读入

{

NodePtr head=q[j]->head;

NodePtr temp=q[j]->head->next;

while(temp!=head)

{

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

{

if(!strcmp(G->vexs[k],temp->name))

{

flag=0;

break;

}

}

if(flag==1) //之前没有相等的站

{

strcpy(G->vexs[i],temp->name);

G->numVertexes++;

i++;

}

temp=temp->next;

flag=1;

}

}

}

//初始化邻接矩阵

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

{

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

{

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

}

}

//建立邻接矩阵

for(k=0;k<4;k++) //将四个链表的边是关系输入邻接矩阵中

{

NodePtr head=q[k]->head;

NodePtr temp=q[k]->head->next;

while(temp->next!=head)

{

transfer(&i,&j,temp->name,temp->next->name,G->vexs);

G->arc[i][j]=temp->distance;

G->arc[j][i]=temp->distance;

G->numEdges++;

temp=temp->next;

}

}

}

void ShortestPath\_Dijkstra(MGraph G, int v0, Patharc \*P, ShortPathTable \*D)

{

int v,w,k,min;

int final[MAXVEX]; // final[w]=1表示求得顶点v0至vw的最短路径

for(v=0; v<G.numVertexes; v++) //初始化数据

{

final[v] = 0; //全部顶点初始化为未知最短路径状态

(\*D)[v] = G.arc[v0][v]; //将与v0点有连线的顶点加上权值

(\*P)[v] = -1; //初始化路径数组P为-1

}

(\*D)[v0] = 0; //v0至v0路径为0

final[v0] = 1; // v0至v0不需要求路径

//开始主循环，每次求得v0到某个v顶点的最短路径

for(v=1; v<G.numVertexes; v++)

{

min=INFINITY; //当前所知离v0顶点的最近距离

for(w=0; w<G.numVertexes; w++) //寻找离v0最近的顶点

{

if(!final[w] && (\*D)[w]<min)

{

k=w;

min = (\*D)[w]; //w顶点离v0顶点更近

}

}

final[k] = 1; //将目前找到的最近的顶点置为1

for(w=0; w<G.numVertexes; w++) // 修正当前最短路径及距离

{

//如果经过v顶点的路径比现在这条路径的长度短的话

if(!final[w] && (min+G.arc[k][w]<(\*D)[w]))

{ // 说明找到了更短的路径，修改D[w]和P[w]

(\*D)[w] = min + G.arc[k][w]; //修改当前路径长度

(\*P)[w]=k;

}

}

}

}

//Dijkstra算法最后得到的结果为：

//顶点a（下标）到其余顶点v的最短路径P[v]以及带权长度D[v]

// P[v]的值为前驱顶点下标，D[v]表示a到v的最短路径之和

void InvertedSequence(int a[],int n) //将有n个元素的数组逆序排列

{

int temp=0;

int i=0;

for(;i<n-1;i++,n--)

{

temp=a[i];

a[i]=a[n-1];

a[n-1]=temp;

}

}

int main()

{

int n,line; //有n条线路,线路名字为line

scanf("%d",&n);

DoubleLinklistPtr q[n];

int i=0; //i就起一个计数的作用

while(i<n)

{

scanf("%d",&line);

q[i]=CreatDoubleLinklist(line); //依次将n条线路读入n条双向链表中

read(q[i]);

i++;

}

char origin[20],terminal[20]; //起点和终点

scanf("%s%s",origin,terminal);

MGraph G;

CreatMGraph(&G,q);

int a,b; //用a，b接受起始和终点对应的数组下标

Patharc P; //用于存储最短路径下标的数组

ShortPathTable D; //用于存储到各点最短路径的权值和

transfer(&a,&b,origin,terminal,G.vexs);

ShortestPath\_Dijkstra(G,a,&P,&D);

printf("%.2f ",D[b]); //将到b的总路程输出

int m=0;

int b\_temp1=b; //b的值需要保存下来，所以后面两个要用到b的地方，分别用b\_temp1和b\_temp2

int b\_temp2=b;

while(P[b\_temp1]!=-1) //判断中间经过了几个站点 ,记录在m中

{

m++;

b\_temp1=P[b\_temp1];

}

int index[m];

i=0;

while(P[b\_temp2]!=-1) //将i个站点的index逆序存入数组中

{

index[i]=P[b\_temp2];

b\_temp2=P[b\_temp2];

i++;

}

InvertedSequence(index,m); //将数组逆序，变成正常的顺序

printf("%s %.2f ",origin,G.arc[a][index[0]]);

for(i=0;i<m-1;i++)

{

printf("%s %.2f ",G.vexs[index[i]],G.arc[index[i]][index[i+1]]);

}

printf("%s %.2f ",G.vexs[index[m-1]],G.arc[index[m-1]][b]);

printf("%s",terminal);

return 0;

}