Ambulance Dispatch

Author:

Complete date:

Ambulance Dispatch

Chapter 1: Introduction

本次project我们需要利用所学的关于图的最短路径知识解决问题。根据一座城市的地图，将发生意外需要救护车的地点优先与到达用时最短的派遣中心相匹配；如果有两个中心的用时相同，则选择救护车较多的中心；如果还相同，则选择经过街道较少的路径。

Chapter 2: Algorithm Specification

基本思路：首先利用输入数据构造图，笔者利用的是邻接表构造图，注意城市节点之间互通，即应该是无向图；然后利用Dijkstra算法找出各派遣中心到各节点的最短路径，鉴于最终结果还与街道数有关，可以将街道数也加入计算，如此将各节点的最短路径、前一个点、最短路径经过的总街道数存入一个结构中，可以以此寻找各紧急地点匹配的派遣中心；寻找时需要将三种情况按优先级进行判断，遍历所有派遣中心对应的结构中存储的数据对比即可得出匹配的派遣中心或者无法得到结果结束。由于并未将最短路径直接存储起来，在打印路径时我们需要借助栈依次输出从派遣中心到紧急地点的路径节点。

数据存储：由于问题限定了数据规模，因此我们可以开固定大小(1011)的数组来存储各种数据，规定1~1000存储普通节点的数据，1001~1010存储派遣中心的数据，0位空出。同时使用全局变量来存储那些多个函数都要用到的数据。

下面是定义的两个数据结构：

```
typedef struct spot* spotp;//结构指针
typedef struct spot{
    int num;        //邻接点的索引值
    int length;     //到源点的时间
    spotp next;     //下一个邻接点
}spot;
```

```
//存储每个派遣中心的数据：各节点到该中心的最短路径，上一节点，街道数
typedef struct ADC{
    int dis[1011];    //最短路径
    int path[1011];   //经过的上一节点
    int strnum[1011]; //街道数
}ADC;
```

图的构建（getgraph）：需要注意的是，由于派遣中心的输入形式带有字母，所以获取输入时需要借助字符串，再转换为整型数据。然后更新图的数据。核心伪代码如下：

```
void getgraph()
{
    int Ns,Na;
    getint(Ns,Na);
    get the ambulance number of each center;
    int road;//总的路径数
    for(i=0 to road){
        gets(V1,V2);
        stringtoint(V1,V2);//获取索引值
        spotp p;//图节点指针

        p point to the graph[V1];
```

```
        find the tail of the list and add node to it.
        add data to new node.

        V2 is the same as the V1.
    }
}
```

Dijkstra 算法：对于将运行算法的派遣中心，步骤如下：1.初始化所需数据数组，除该中心对应的位置，最短距离都为无穷（可用某个极大的数代替，如两亿，但是谨慎使用INT_MAX，虽然它确实是可存储最大值，但是这也代表了如果有后续计算会导致溢出，即 INT_MAX+1<0<INT_MAX），初始经过街道数都为0，known数组初始全为0，即尚未判断；还有数组中多余的点known全部置为1（相当于我们已判断——无法到达）2.找到未判断的节点中已有路径最短的索引值（由于中心初始化0，第一个一定是该中心），该点"已判断"known置为1；3.遍历该点的所有邻接点，计算经过该店到达邻接点的路径时间和街道数并按情况更新数据（虽然题目中未特别指出，但是同一中心到达节点的路径应该也是按街道数较少来匹配的）4.重复2、3。伪代码如下：

```
void Dijkstra( int index)
{
    int known[1011];
    initialize the data arrays;
    while(1){
        int min,spotV;
        spotV=smallest unknown distance vertex;
        if(notfound)return ;
        known[spotV]=1;
        spotp p=the first vertex adjacent to spotV;
        while(!p){
            if(!known[p->num]){
                if(Dis(index to spotV)+length(spot to p)<Dis(the data stored before))
                    update the data;
                else if(the distace are same but the new one has less street cost)
                    update the data;
            }
        }
    }
}
```

派遣算法（call()）:遍历存储了最短路径的结构体，找到最短的那个派遣中心索引值即可，之后就是数据打印。伪代码：

```
void call(int spotV)
{
    for(every center structure){
        if(the ambulance number ){
            if(shorter path||same path,more amulance||same ambulance,less street cost)
                update the data we need;
        }
    }
    if(not found)All Busy ;return ;

    the ambulance number--;
    int stack[],top,element=spotV;
```

```
    while(element!=0){
        add to stack;
        element=the Vertex before element;
    }
    print the stack;
}
```

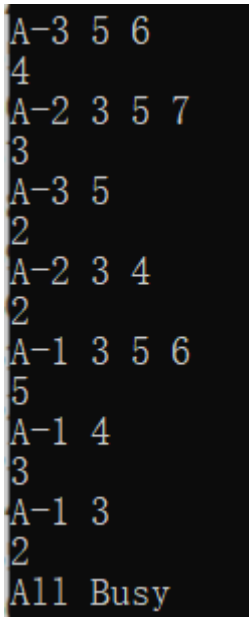
Chapter 3: Testing Result

特殊的情况基本如下：1.如题判断最短路径，较多救护车，较少街道 2.救护车为0的中心沦为普通点，需要经过 3.(题目未明确)同一中心到达同一节点有相同路径长甚至街道数（可能姥姥测试数据有点问题）4.图未完全联通（虽然同一城市不太可能，仅作参考）

除了各种多种选择的相同情况，应该基本无特殊的样例，所有数据的处理逻辑都是一样的，在此给出几个小数据样例说明特殊情况，最后pta验证大数据。

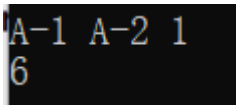
测试1：pta样例

```
7 3
3 2 2
16
A-1 2 4
A-1 3 2
3 A-2 1
4 A-3 1
A-1 4 3
6 7 1
1 7 3
1 3 3
3 4 1
6 A-3 5
6 5 2
5 7 1
A-2 7 5
A-2 1 1
3 5 1
5 A-3 2
8
6 7 5 4 6 4 3 2
```



测试2：对应情况2

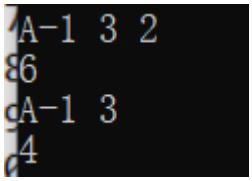
```
1 2
2 0
2
A-1 A-2 2
A-2 1 4
1
1
```



测试3：情况3

一个简单的图，从A-1到2有两种情况 A-1 3 2和A-2 1 2，但是由于Dijkstra 算法在比较判定街道数时采用的是”<=“而不是“<”，导致算法倾向于选择最后一段路径较短的路程替换较长的。（不要问为什么，问就是不这样过不了pta的最后一个点）。

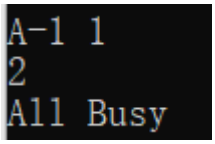
```
3 2
2 2
5
A-1 1 5
A-1 3 4
1 2 1
3 2 2
A-2 3 6
2
2 3
```



测试4：情况4

对于连通的点，按情况正常输出，不连通的点，无法派遣。

```
2 2
2 0
2
A-1 1 2
A-2 2 2
2
1 2
```



测试5：pta测试（增加一下正确性，同时说明大数据也可以跑）

提交时间	状态	分数	题目	编译器	耗时	用户
2021/11/28 20:40:39	答案正确	30	编程题	C (gcc)	68 ms	

测试点	结果	分数	耗时	内存
0	答案正确	15	2 ms	336 KB
1	答案正确	6	3 ms	348 KB
2	答案正确	3	3 ms	440 KB
3	答案正确	2	3 ms	324 KB
4	答案正确	4	68 ms	1016 KB

Chapter 4: Analysis and Comments

空间复杂度：构建图时使用邻接表，使用的空间大小与边的个数(E)是成正相关的，即O(E)，而Dijkstra算法和后续的派遣算法所需要和实际使用的空间都与节点总数(V)成正相关，还有使用的常数量中间变量，即空间复杂度为O(V).

时间复杂度：图的大小与边的多少有关，操作次数也相关，即构建图时时间复杂度为O(E)。

运行Dijkstra 时，需要每次寻找一条当前最短路径，再遍历所有邻接节点，可以得到最坏情况即所有节点两两之间都有一条边，这样遍历比较的次数最大。即时间复杂度为约为O(V*V+V*C(V,2))=O(V^3)（前面的V*V是寻找最短路径需要的时间，后面一部分是由已知最短路径再延申的计算时间）,实际上这种情况是不可能的，可以给出更加贴近的O(V*V+V*E)，当然可以思考这样也不是最为贴近的情况，但已经是可以给出的最贴近的复杂度了。

派遣时，只需要遍历数据结构中的存储数据，可以直接由索引值获取数据，最坏不过遍历所有结构，即O(Na^2);

Appendix: Source Code (in C)

```c
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define INF 2000000000

typedef struct spot* spotp;
typedef struct spot{
    int num;        //the index of the ambulance dispatch centers and pick-up spots.
    int length;     //the time needed to pass the street
    spotp next;     //next spot connect to the sourse spot.
}spot;
//I use Adjacency List to store the graph.

typedef struct ADC{
    int dis[1011];//the least time we need to the i-th adc from dis[j] spot.
    int path[1011];// the one just before j-th spot when we take the shortest path.
    int strnum[1011];//the number of streets we pass.
}ADC;

void getgraph(); //generate the graph.
int spotnum(char ch[]);//Converts a string to an integer number.
void Dijkstra( int index);//record the shortest path to the adc[index]
void call(int spot);//call the ambulance

int Ns,Na;//the total number of pick-up spots(Ns)and the number of ambulance dispatch centers(Na).
spot graph[1011];//1~1000 are the space of pick-up spots,1001~1010 are the space of ambulance dispatch centers.
int amb_ava[11];//the number of available ambulances at the i-th center.
ADC adc[11];//record the shortest path to adc[i] from each spot.

void getgraph()
{
    int i,road;
    char get1[5],get2[5];
    int  spot1,spot2;
    int  length;
    spotp p;

    scanf("%d %d",&Ns,&Na);//the total number of pick-up spots
                           //and the number of ambulance dispatch centers
    for(i=1;i<=Na;i++){
        scanf("%d",&amb_ava[i]);//the number of the ambulance in each center.
    }
    scanf("%d",&road);//the number of all the streets.

    for(i=0;i<road;i++){
```

```
        scanf("%s",get1);scanf("%s",get2);//get the two indices of two
spots.
        scanf("%d",&length);//get the time
        spot1=spotnum(get1);spot2=spotnum(get2);//get the index we use in
this program

        //the next eight lines update the graph.
        p=&graph[spot1];
        while(p->next) p=p->next;// get the tail of the list.
        p->next=(spotp)malloc(sizeof(spot));
        p->next->num=spot2; p->next->length=length; p->next->next=NULL;

        p=&graph[spot2];
        while(p->next) p=p->next;
        p->next=(spotp)malloc(sizeof(spot));
        p->next->num=spot1; p->next->length=length; p->next->next=NULL;
    }
}
int spotnum(char ch[])
{
    int spot;
    if(ch[0]=='A')spot=atoi(&ch[2])+1000;
                //if it is a center,we need add 1000 to get its index.
    else spot=atoi(ch);
    return spot;
}
void Dijkstra( int index)
{
    int known[1011];
    int i;
    for(i=1;i<=1010;i++){
        adc[index].dis[i]=INF;//the time is infnite for all spots at first
        adc[index].strnum[i]=0;//the number is 0.
        known[i]=0;//we don't know the spot
        if(i==index+1000){
            adc[index].dis[i]=0;//the sourse center spot is initialized to
be 0;
        }
        if((i>Ns&&i<=1000)||(i>Na+1000&&i<=1010))known[i]=1;
        //we know we can't reach these spots.
    }//initialize the data.
    while(1){
        int min=INF,spotV=0;
        for(i=1;i<=1010;i++){
            if(adc[index].dis[i]<min&&known[i]==0){
                spotV=i;
                min=adc[index].dis[i];
            }
        }//we get the spot which has the shortest path .
        if(spotV==0)break;//if spotV==0,then all the spots are known.
        known[spotV]=1;
        spotp p=graph[spotV].next;

        while(p){
            if(known[p->num]==0){
                int Dis1=adc[index].dis[spotV];//the time we need to reach
spotV.
```

```
                int Dp=p->length;//the time we need from spotV to p adjacent to V
                int Dis2=adc[index].dis[p->num];//the shortest time p stored at first.
                if(Dis1+Dp<Dis2){//the new one is shorter ,update the data
                    adc[index].dis[p->num]=Dis1+Dp;
                    adc[index].strnum[p->num]=adc[index].strnum[spotV]+1;
                    adc[index].path[p->num]=spotV;
                }
                else if(Dis1+Dp==Dis2&&adc[index].strnum[spotV]+1<=adc[index].strnum[p->num]){
                    //they have the same time,but new one has less streets cost.
                    adc[index].strnum[p->num]=adc[index].strnum[spotV]+1;
                    adc[index].path[p->num]=spotV;
                }
            }
            p=p->next;// the next one adjacent to V .
        }
    }
}
void call(int spotV)
{
    int i,Dmin=INF,Smin,index=0;

    for(i=1;i<=Na;i++){
        if(amb_ava[i]!=0){//it need to has at least one ambulance.
        //three condition:1.shorter time 2.same time but more ambulance 3.same ambulance but less street cost.
            if((adc[i].dis[spotV]<Dmin)||
            (adc[i].dis[spotV]==Dmin&&amb_ava[i]>amb_ava[index])||
            (amb_ava[i]==amb_ava[index]&&adc[i].strnum[spotV]<Smin)){
                //update the chosen center data
                index=i;
                Dmin=adc[i].dis[spotV];
                Smin=adc[i].strnum[spotV];
            }
        }
    }
    if(index==0||adc[index].dis[spotV]==INF){//there are no center connect to it.
        printf("All Busy\n");
        return ;
    }
    amb_ava[index]--;//the number needs to -1.

    int stack[1012],top=-1,element=spotV;

    while(adc[index].path[element]){//use the stack to record the path.
        stack[++top]=element;
        element=adc[index].path[element];
    }
    //print the path.
    printf("A-%d",index);
    while(top>=0){
        if(stack[top]>1000)printf(" A-%d",stack[top--]-1000);
        else printf(" %d",stack[top--]);
    }
```

```
    printf("\n");
    printf("%d\n",adc[index].dis[spotV]);
}
int main()
{
    int i;

    getgraph();
    for(i=1;i<=Na;i++){
        Dijkstra(i);
    }
    int Snum;//the number of spot need ambulance.
    int spotV;//the index of the spot.
    scanf("%d",&Snum);
    for(i=0;i<Snum;i++){
        scanf("%d",&spotV);
        call(spotV);
    }
    return 0;
}
```

Declaration

I hereby declare that all the work done in this project titled "Ambulance Dispatch" is of my independent effort.

我在此声明，在这个名为“Ambulance Dispatch”的项目中所做的所有工作都是我独立完成的。