实验 6 图的应用(通信网络)

计科 1903 201914020128 陈旭

一. 问题分析

1). 问题与功能描述

1. 需要处理的数据是一串成对的整型(int 型)数字，他们共同组成了图的元素(结点和边)。
2. 实现的功能有:
   - 读入两个整数(n，m)，前者作为路口的数量(结点个数)，后者作为(每个路口之间的)结点之间的边数；
   - 循环。循环 m 次，将每一个路线信息，连接的结点，路口之间长度输入图中，同时判断，大路则直接作为权值插入图中，小路则取平方之后插入；
   - Dijkstra 算法求出单源最短路径；
   - 输出最优路径下的总疲劳值。
3. 使用标准输入输出。

2). 样例分析

1. **求解方法:** 该题所对应的图是无向图。图建好后，使用 Dijkstra 算法算出从 1 号路口到 n 号路口的最优路径对应的疲劳值
2. **样例求解:**

【样例输入】 6 7

1 1 2 3

0 2 3 2

0 1 3 30

0 3 4 20

0 4 5 30

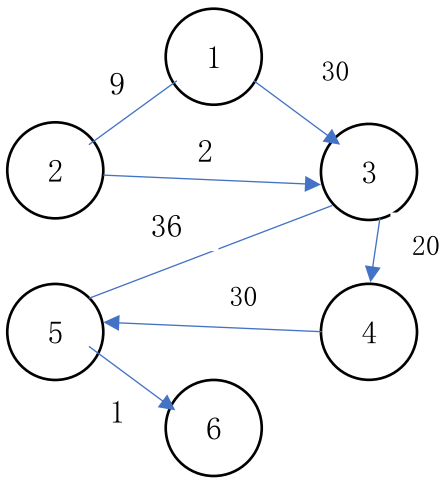
1 3 5 6

0 5 6 1

【样例输出】 48

【求解过程】 求解过程如下

样例对应的图示意如下(箭头仅表示对应大道并不表示是有向图)



先使用一个数组 D，初始化为 D[0]=0，其他元素值为 99999(infinite)，即此时 D 为

0 inf inf inf inf inf

执行 Dijkstra 算法，1 作为源点并

标记为被访问，目前 D:

0 9 30 inf inf inf

选定 2 标记访问，目前 D:

0 9 11 inf inf inf

选定 3 标记访问，目前 D:

0 9 11 31 47 inf

选定 4 标记访问，目前 D:

0 9 11 31 47 inf

选定 5 标记访问，目前 D:

0 9 11 31 47 48

数据结构分析

1) **数据对象** 本题处理的数据为整型(int)数组。

数据关系 本题处理的数据为整型有限数据，题目每组数据之间均可能有关系。因此使用图作为基本数据结构进行操作。

2) **基本操作**

- **【功能描述】**设置图的类型(有向或者无向)

  【名字】 setType

  【输入】 整数 n,1 代表有向，0 代表无向

  【输出】 无

- **【功能描述】**设置结点

  【名字】 setVex

【输入】 两个整数，分别代表结点编号和结点值

【输出】 无

- 【功能描述】设置图的边

  【名字】 setEdge

  【输入】 三个整数，分别代表出度，入度，权值(为处理方便本题直接统一赋值为 0)

  【输出】 无

- 【功能描述】求解最优路径

  【名字】 Dijkstra

  【输入】 一个整数 s 表示源点所在位置(本题固定为 0)

  【输出】 无

- 【功能描述】求解未加入路径的目前对应的最小权值结点

  【名字】 minVertex

  【输入】 无

  【输出】 该结点对应的下标

- 【功能描述】输出第 n 号结点对应的总疲劳数

  【名字】 getFinalD

  【输入】 无

  【输出】 输出最后结点对应的总疲劳值

3) 物理实现

```
void setType(int flag){
    if (flag==1){
        isDirected=true;
    }
    else{
        isDirected=false;
    }
} //给结点赋值

void setVex(int v, E value){
    assert(v<numVertex);
    vertex[v]=value;
}//给结点赋值


void setEdge(int v1, int v2, int wght){
    assert(v1<numVertex&&v2<numVertex);
    if (v1==v2){
        Comnetwork[v1][v2]=1;
        matrix[v1][v2]=1;
    }
    if (matrix[v1][v2]==0){
        numEdge++;
        matrix[v1][v2]=1;
        Comnetwork[v1][v2]=1;
    }
    weight[v1][v2]=wght;
    if(!isDirected){
        matrix[v2][v1]=1;
        weight[v2][v1]=wght;
        Comnetwork[v2][v1]=1;
    }
}//建立一个边，也就是把权值和坐标赋值

void Dijkstra(int s){
    for (int i=0; i<numVertex; i++){
        int v=minVertex();
        if(D[v]==99999){
            return ;
        }
        setMark(v, 1);
        for (int j=first(v);j<numVertex;j=next(v, j)){
            if(D[j]>D[v]+getWeight(v, j)){
                D[j]=D[v]+getWeight(v, j);
```

```
            }
        }
    }
}//Dijkstra 算法函数

int minVertex(){
    int v;
    for (int i=0; i<numVertex; i++){
        if(getMark(i)==0){
            v=i;
            break;
        }
    }
    for (int i=0; i<numVertex; i++){
        if(getMark(i)==0&&D[i]<D[v]){
            v=i;
        }
    }
    return v;
}//返回当前距离最小的结点

int getFinalD(){
    return D[numVertex-1];
}//返回最终最优路径对应的疲劳程度
```

二. 算法分析

- **算法思想:**

Dijkstra 算法思想:设 G=(V,E)是一个带权有向图,把图中顶点集合 V 分成两组，第一组为已求出最短路径的顶点集合（用 S 表示，初始时 S 中只有一个源点，以后每求得一条最短路径，就将加入到集合 S 中，直到全部顶点都加入到 S 中，算法就结束了），第二组为其余未确定最短路径的顶点集合（用 U 表示），按最短路径长度的递增次序依次把第二组的顶点加入 S 中。在加入的过程

中，总保持从源点 v 到 S 中各顶点的最短路径长度不大于从源点 v 到 U 中任何顶点的最短路径长度。此外，每个顶点对应一个距离，S 中的顶点的距离就是从 v 到此顶点的最短路径长度，U 中的顶点的距离，是从 v 到此顶点只包括 S 中的顶点为中间顶点的当前最短路径长度。

(1) 初始时，S 只包含起点 s；U 包含除 s 外的其他顶点，且 U 中顶点的距离为"起点 s 到该顶点的距离"[例如，U 中顶点 v 的距离为(s,v)的长度，然后 s 和 v 不相邻，则 v 的距离为∞。

(2) 从 U 中选出"距离最短的顶点 k"，并将顶点 k 加入到 S 中；同时，从 U 中移除顶点 k。

(3) 更新 U 中各个顶点到起点 s 的距离。之所以更新 U 中顶点的距离，是由于上一步中确定了 k 是求出最短路径的顶点，从而可以利用 k 来更新其它顶点的距离；例如，(s,v)的距离可能大于(s,k)+(k,v)的距离。

(4) 重复步骤(2)和(3)，直到遍历完所有顶点。

- **性能分析**

【空间复杂度】原题代码可以分为三个部分，开辟了一个个个一维动态数组，三个二维动态数组，除此之外还开辟了一些单独的变量，三者空间复杂度为 O(n), O(n^2),O(1);取最大值得到空间复杂度为 O(n^2)

【时间复杂度】Dijkstra 算法对应的两个函数各有两个 for 循环，一个嵌套，一个并列。相当于在外层 for 循环的内部加入了三个并列的 for 循环，即时间复杂度为 O(n^2)