实验六 图的应用实验报告

一、问题分析

- 要处理的对象（数据）：

  2 个整数：总人数和可以互相转账的人的对数

  由 3 个正整数组成的 m 组数：互相转账的人，转账需要扣除的手续 z%(z<100)。

  2 个整数：所求的转账对象

- 要实现的功能：

  构建转账扣除手续费的关系

  求 A 最少需要多少钱使得转账后 B 收到 100 元

- 处理后的结果如何显示：通过屏幕输出结果，精确到小数点后 8 位
- 样例求解过程:

1. 样例一

① 输入数据

3 3

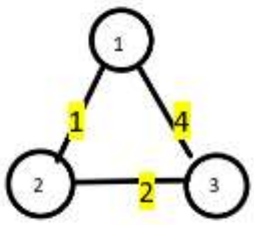
1 2 1

2 3 2

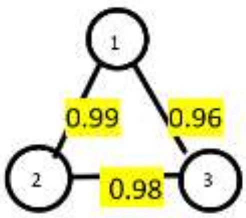
1 3 4

1 3

② 将输入数据构建成如下图。



③ 体现边权与手续费的正向关系，修改图如下：



④ 从起点 1 开始，读取相邻边入距离数组，到 2 距离为 99，到 3 距离为 96。修改 1 状态。

⑤ 1 已被访问。继而访问 2，2 到 3 距离为 98。比较 0.99*0.98=0.9702>0.96，更新 1 到 3 距离为 0.9702。修改 2 访问状态。

⑥ 3 节点图已访问 1 个中间节点（2），判断最短路径已找到，结束访问。

⑦ 读取距离数组，下标 2 的位置值为 0.9702，即 1 向 3 转账手续费最低为（1-0.9702）*100%。

⑧ 计算 100/0.9702，去尾法保留 8 位小数，输出结果：103.07153164。

二、数据结构和算法设计

抽象数据类型：

数据对象：n 个数字构成顶点集，m 组数字构成弧集

数据关系：两点之间由有弧长的无向边连接。

基本操作：

```
//初始化图
virtual void Init(int n) =0;
//返回顶点数和边数
virtual int n() =0;
virtual int e() =0;
//返回顶点 v 的第一个邻接点
virtual int first(int v) =0;
//返回顶点 v 的下一个邻接点
virtual int next(int v, int w) =0;
//给边赋权值
virtual void setEdge(int v1, int v2, double wght) =0;
//判断两点间是否有边
virtual bool isEdge(int i, int j) =0;
//返回边的权值
virtual double weight(int v1, int v2) =0;
//获取/设置点的标记量
virtual int getMark(int v) =0;
virtual void setMark(int v, int val) =0;
```

物理数据类型：

选用基于邻接矩阵实现的无向图

算法思想：

可以理解为迪克斯特拉算法的变式求单源最短路径。

①边的权值修改为 1-手续费。

②构建一个储存 1-手续费 （最短距离）的数组。

③外层循环从起点开始依次访问图的节点，内层循环将访问节点的相邻节点依次当作目标节点。

④若起点与目标节点之间无距离，直接读入访问节点到目标节点的权值；若起点与目标节点之间有距离，比较起点到访问节点的距离*访问节点到目标节点的距离和起点到目标节点的距离。若前者大于后者，代表这条转账路径花费更少手续费，将该值更新为目标节点的最短距离。

⑤外层循环遍历完全图后，最短数组下标为目标节点的储存的值即为所求。

关键功能的算法步骤：

1. 构建图，将顶点和权值存入图中。注意存进的权值为(100-手续费)/100。

```
Graph* G;
int vert_num=0,enu=0;     //  图的顶点数，编号从 0 开始
cin >> vert_num>>enu;
G = createGraph(1, vert_num);
int fr_vert=0, to_vert=0;
double wt=0;
while(enu>0)
{
    double wt1=0;
    cin >> fr_vert >> to_vert >> wt;
    wt1=double((100-wt)/100);
    G->setEdge(fr_vert-1, to_vert-1, wt1);
    G->setEdge(to_vert-1, fr_vert-1, wt1);
    enu--;
}
int ae=0,be=0;
cin>>ae>>be;
```

构件图，时间/空间复杂度都和|v|+|E|相关，化简为线性阶。

时间复杂度：O(n)，空间复杂度：O(n)

2. 构建最短距离数组，对用迪克斯特拉变式算法处理图。代码具体思想见上。

```
double D[G->n()];
for (int i = 0; i < G->n(); i++) // Initialize
    D[i] = INFINITY;
it->Dijkstral_(D, ae-1,be-1);

void Dijkstral_(double* D, int s,int ss)
{
    for(int i=0;i<G->n();i++)
        D[i]=G->weight(s,i);
    D[s]=1;
    G->setMark(s,VISITED);
    for(int i=0;i<G->n()-2;i++)
    {
        double mm=0;
        int k;
        for(int j=0;j<G->n();j++)
        {
            if(G->getMark(j)==UNVISITED&&D[j]>mm)
            {
                k=j;
                mm=D[j];
            }
        }
```

```
            G->setMark(k,VISITED);
            if(k==ss) return ;
            for(int j=0;j<G->n();j++)
                if(G->getMark(j)==UNVISITED&&D[j]<D[k]*G->weight(k,j) )
                    D[j]=D[k]*G->weight(k,j);
        }
    }
```

对图的处理的时间代价为线性阶。迪克斯特拉近似算法中，时间代价最大的部分为双重 for 循环，时间代价为平方阶。构造最短路径数组，空间代价为线性阶。

时间复杂度：O(n^2)，空间复杂度：O(n)

3. 读取最短路径长度，计算输出结果。

三、算法性能分析

见上，时间复杂度为 O(n^2)，空间复杂度为 O(n)。