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
2. 输入数据

3 3

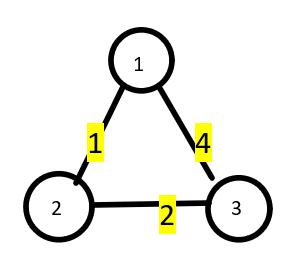
1 2 1

2 3 2

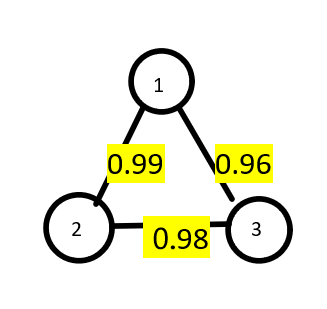
1 3 4

1 3

1. 将输入数据构建成如下图。



1. 体现边权与手续费的正向关系，修改图如下：



1. 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。
2. 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。
3. 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。
4. 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%。
5. 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)

1. 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)

1. 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)。