

7.7 课堂笔记 18

上周作业答案：

1、P1341:

本质是欧拉路和欧拉回路

①、这里就可以发现实际上就是在找欧拉路，首先每个字符就是代表的图中的某一个点，不

断输入字符串，就代表两点相邻即有连通。

构造输出的字符串就是在找输出一笔画回路，

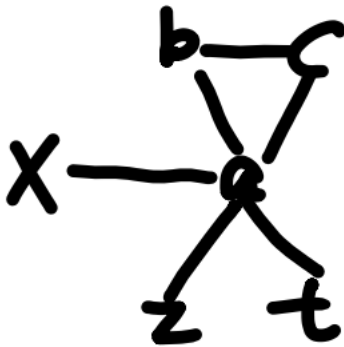
②、构建邻接矩阵的大小需要计算，方法如下：

大写字母和小写字母共计 52 个，因此最多无向边有： $52 \times 52 / 2 \approx 1500$

因此邻接矩阵开到 1500*1500 大小

③、一般在欧拉路问题中的 visit 数组针对边而言，因为如果针对点，有的点会重复走多次

例如下图：



输出的答案应该为：abcatzax，其中 a 走了 3 次

```
1  #include <iostream>
2  using namespace std;
3  int _g[1510][1510];
4  int _du[1510];
5  int _path[2000];
6  int _n, _m, _minSta = 1510, _en;
7
8  void Dfs(int i) {
9      for(int j = 1; j <= 1500; j++) {
10         if(_g[i][j] == 1) {
11             _g[i][j] = 0;
12             _g[j][i] = 0;
13             Dfs(j);
14         }
15     }
16     // 出栈顺序
17     _en++;
18     _path[_en] = i;
19 }
20
```

```

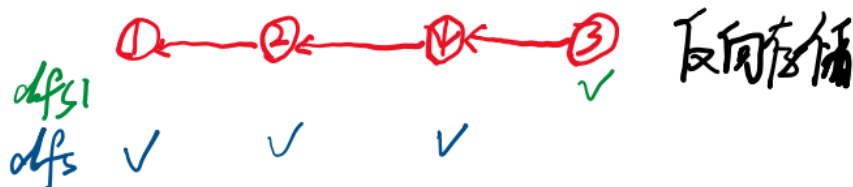
21 int main () {
22     cin >> _n;
23     char x, y;
24     for(int i = 1; i <= _n; i++) {
25         cin >> x >> y;
26         // cout << min((int)x, (int)y);
27         _minSta = min( min((int)x, (int)y), _minSta);
28         _g[(int)x][(int)y] = 1;
29         _g[(int)y][(int)x] = 1;
30         _du[(int)x]++;
31         _du[(int)y]++;
32     }
33     int cc = 0;
34     for(int i = 1; i <= 1500; i++) {
35         if(_du[i] % 2 == 1) {
36             _minSta = i; // 奇点开始
37             break;
38         }
39     }
40     for(int i = 1; i <= 1500; i++) {
41         if(_du[i] % 2 == 1) {
42             cc++;
43         }
44     }
45     if(cc != 0 && cc != 2) { // 奇点不为0或者不为2
46         cout << "No Solution";
47         return 0;
48     }
49     Dfs(_minSta);

50     for(int i = _en; i >= 1; i--) {
51         cout << (char)_path[i];
52     }
53
54     return 0;
55 }

```

2、P3916:

方法二：存图的时候反向存储，并从最大的点开始搜索，当某点的 visit 数组已经非零，证明已经有更大的点可以走到该处，因此无需再继续深搜该点：



如上图所示，只需要进行两次 dfs

```

1  #include <iostream>
2  #include <string.h>
3  using namespace std;
4  int numEdge = 0, n, m;
5  bool visit[100010];
6  int num;
7  int head[100010];
8  int res[100010];
9  struct edge{
10     int next;
11     int to;
12     int w;
13 } e[100010];
14
15 void addEdge(int from, int to){
16     numEdge++;
17     e[numEdge].next = head[from];
18     e[numEdge].to = to;
19     head[from] = numEdge;
20 }
21 void dfs(int i){
22     visit[i] = true;
23     res[i] = num;
24     for(int j = head[i]; j != 0; j = e[j].next){
25         int y = e[j].to;
26         if( !visit[y]){
27             dfs(y);
28         }
29     }
30 }

```

```

31
32
33 int main(){
34     int v1 = 0, v2 = 0;
35     cin >> n >> m;
36     for(int i = 1; i <= m; i++){
37         cin >> v1 >> v2;
38         addEdge(v2, v1);
39     }
40
41     for(int i = n; i >= 1; i--){
42         num = i;
43         if( !visit[i]){
44             dfs(i);
45         }
46     }
47     for(int i = 1; i <= n; i++){
48         cout << res[i] << " ";
49     }
50 }

```

一、邻接表 dfs-模板

```
1  #include<iostream>
2  using namespace std;
3  int head[100];
4  struct edge{
5      int next;
6      int to;
7      int w;
8  }e[10000];
9  int v1,v2,dis;
10 int n,m;
11 int numEdge;//统计边的数量
12 bool vis[100];//用来标记走过的点
13
14 void addEdge(int from, int to, int w){
15     numEdge++;
16     e[numEdge].next = head[from];
17     e[numEdge].to = to;
18     e[numEdge].w = w;
19     head[from] = numEdge;
20 }
21
22 void dfs(int x){
23     vis[x] = 1;//第一步就把走过的这个点给标记起来
24     cout << x << " ";
25     for(int i=head[x];i!=0;i=e[i].next){ //遍历当前这个点作为起点，它的所有边
26         int y = e[i].to; //当前这条边的终点
27         if(!vis[y]){ //这个点没有走过
28             dfs(y);
29         }
30     }
31 }
32
33 int main(){
34     cin >> n >> m;
35     for(int i=1; i<=m; i++){
36         cin >> v1 >> v2 >> dis;
37         addEdge(v1,v2,dis);
38     }
39     dfs(1);
40     return 0;
41 }
42
```

练习：2853

解题思路：

通过牛所在的点开始进行深搜，找到其能到达的点，设置一个数组来记录不同点的到达次数。

例如测试用例中的 2 号牛能到达 2 3 4， 3 号牛能到达 3 4

因此 3 4 位置能到达两次，和牛的数量相同。因此最终可以作为进餐的地点为 2 个。

```

15 void addEdge(int from, int to){
16     numEdge++;
17     e[numEdge].next = head[from];
18     e[numEdge].to = to;
19     head[from] = numEdge;
20 }
21
22 void dfs(int x){
23     vis[x] == 1;
24     num[x]++;
25     for(int i=head[x]; i!=0; i=e[i].next){
26         int y = e[i].to;
27         if(!vis[y]){
28             dfs(y);
29         }
30     }
31 }

```

```

33 int main(){
34     cin >> k >> n >> m;
35     for(int i=1; i<=k; i++){
36         cin >> niu[i];
37     }
38     for(int i=1; i<=m; i++){
39         cin >> v1 >> v2;
40         addEdge(v1, v2);
41     }
42     for(int i=1; i<=k; i++){
43         memset(vis,0,sizeof(vis));
44         dfs(niu[i]);
45     }
46     for(int i=1; i<=n; i++){
47         if(num[i] == k){
48             cnt++;
49         }
50     }
51     cout << cnt << endl;
52     return 0;
53 }

```

二、邻接表—dfs、bfs—5318

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  int numEdge = 0, n, m;
4  int vis[100010];
5  int head[100000];
6  queue <int> que;
7  struct pst{
8      int u;
9      int v;
10 }poi[1000010];
11
12 struct edge{
13     int next;
14     int to;
15 } e[2000010];
16
17 bool cmp(pst a, pst b){
18     if(a.u == b.u){//起点相同的时候, 要保证终点降序
19         return a.v > b.v;
20     }
21     return a.u < b.u;
22 }
23
24 void addEdge(int from, int to){
25     numEdge++;
26     e[numEdge].next = head[from];
27     e[numEdge].to = to;
28     head[from] = numEdge;
29 }
```

```

31 void dfs(int x){
32     cout << x << " ";
33     vis[x] = true;
34     for(int i = head[x]; i != 0; i = e[i].next){
35         int y = e[i].to;
36         if(!vis[y]){
37             dfs(y);
38         }
39     }
40 }
41
42 void bfs(){
43     que.push(1);
44     vis[1] = true;
45     int h = 0;
46     while(!que.empty()){
47         h = que.front();
48         que.pop();
49         cout << h << " ";
50         for(int i = head[h]; i != 0; i = e[i].next){
51             int y = e[i].to;
52             if(!vis[y]){
53                 vis[y] = true;
54                 que.push(y);
55             }
56         }
57     }
58 }
59
60
61 int main(){
62     int u = 0, v = 0, cc = 0;
63     cin >> n >> m;
64     for(int i = 1; i <= m; i++){
65         cin >> u >> v;
66         poi[i].u = u;
67         poi[i].v = v;
68     }
69     sort(poi + 1, poi + 1 + m, cmp);
70     for(int i = 1; i <= m; i++){
71         addEdge(poi[i].u, poi[i].v);
72     }
73     dfs(1);
74     cout << endl;
75     memset(vis, 0, sizeof(vis));
76     bfs();
77 }

```


图的最短路问题:

图的最短路问题:

一、无权图

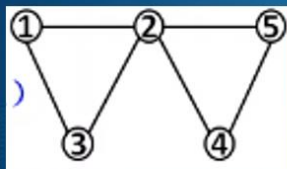
- 1、单源: 1次bfs
- 2、多源: 多次bfs

二、有权图

- 1、单源:
 - (1) 无负权: Dijkstra算法
 - (2) 有负权或负环: Ford算法/SPFA算法
- 2、多源: Floyd算法

图的最长路问题: 拓扑排序 (无环有向图)

三、最短路—bfs



【题目描述】

每个顶点都站了一位同学, john站在点1的位置, 他想知道其他同学距离他的最短距离分别是多少?

【输入】

第一行两个值分别为n, m表示n个顶点, m条边
接下来m行, 每行两个数据, 起点, 终点

【输出】

从顶点2至n距离1的最短距离

【输入样例】

5 6

1 2

1 3

2 3

2 4

2 5

4 5

【输出样例】

1 1 2 2


```

32 void Bfs() {
33     queue<int> que;
34     que.push(1);
35     _deep[1] = 1;
36     while(que.size() > 0) {
37         int x = que.front();
38         for(int i = _head[x]; i != 0; i = _e[i].next) {
39             int y = _e[i].to;
40             if(_deep[y] != 0) {
41                 continue;
42             }
43             _deep[y] = _deep[x] + 1;
44             que.push(y);
45         }
46         que.pop();
47     }
48 }

```

四、最短路—floyed 算法

任意节点 i 到 j 的最短路径两种可能：

- 1、直接从 i 到 j ;
- 2、从 i 经过若干个节点 k 到 j 。

在此 $d(i,k)$ 与 $d(k,j)$ 分别是目前为止所知道的 i 到 k 与 k 到 j 的最短距离。

若有 $d(i,j) > d(i,k) + d(k,j)$ ，就表示从 i 出发经过 k 再到 j 的距离要比原来的 i 到 j 距离短，自然把 i 到 j 的 $d(i,j)$ 重写为 $d(i,k) + d(k,j)$ ，每当一个 k 查完了， $d(i,j)$ 就是目前的 i 到 j 的最短距离。重复这一过程，最后当查完所有的 k 时， $d(i,j)$ 里面存放的就是 i 到 j 之间的最短距离了。

```

12 #include <cstring>
13 using namespace std;
14 int g[110][110];
15 int n, m;
16 void Floyed(){
17     for(int k = 1; k <= n; k++){
18         for(int i = 1; i <= n; i++){
19             for(int j = 1; j <= n; j++){
20                 if(i != j && i != k && j != k){
21                     g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
22                 }
23             }
24         }
25     }
26 }
27 void print(){
28     for(int i = 1; i <= n; i++){
29         for(int j = 1; j <= n; j++){
30             cout << g[i][j] << " ";
31         }
32         cout << endl;
33     }
34 }
35 int main(){
36     int u = 0, v = 0, w = 0;
37     memset(g, 0x3f, sizeof(g));
38     cin >> n >> m;
39     for(int i = 1; i <= m; i++){
40         cin >> u >> v >> w;
41         g[u][v] = w;
42     }
43     Floyed();
44     print();
45     return 0;

```

练习 1:

最短路径之Floyd算法

short.cpp

平面上有 n 个点 ($n \leq 100$), 每个点的坐标均在-10000~10000之间. 其中的一些点之间有连线. 若有连线, 则表示可从一个点到达另一个点, 即两点之间有通路, 通路的距离为两点间的直线距离. 现在的任务是找出从一点到另一点之间的最短距离.

输入(short.in):

第一行为整数 n ;

第2行到 $n+1$ 行 (共 n 行), 每行两个整数 x 和 y , 描述了一个点的坐标 (以一个空格分隔) .

第 $n+2$ 行为一个整数 m , 表示图中连线的个数.

此后的 m 行, 每行描述一条连线, 由两个整数 i 和 j 组成, 表示第 i 个点和第 j 个点之间有连线.

最后一行: 两个整数 s 和 t , 分别表示源点和目标点.

输出(short.out):

仅一行, 一个实数 (保留两位小数), 表示从 s 到 t 的最短路径长度

样例:

short.in

5

0 0

2 0

2 2

0 2

3 1

5

1 2

1 3

1 4

2 5

3 5

1 5

short.out

3.41

```
int main() {
    cin >> n;
    for(int i = 1; i <= n; i++) {
        cin >> poi[i].x >> poi[i].y;
    }
    cin >> m;
    memset(g, 0x7f, sizeof(g));
    for(int i = 1; i <= m; i++) {
        cin >> u >> v; //poi[u].x, poi[u].y, poi[v].x, poi[v].y
        g[u][v] = g[v][u] = sqrt(pow(poi[u].x - poi[v].x, 2) + pow(poi[u].y - poi[v].y, 2));
    }
    Print();

    return 0;
}
```

练习 2:

有 n 个城市, 编号 $1 \sim n$. 其中 i 号城市的繁华度为 p_i . 省内有 m 条可以双向同行的高速公路, 编号 $1 \sim m$. 编号为 j 的高速公路连接编号为 a_j 和 b_j 两个城市, 经过高速公路的费用是 w_j . 若从城市 x 出发到某城市 y , 除了需要缴纳高速公路费用, 还要缴纳“城市建设费” (为从 x 城市到 y 城市所经过的所有城市中繁华度的最大值, 包括 x 和 y 在内) .

现提出 q 个询问, 每个询问给出一组 x 和 y , 你需要回答从 x 出发到 y 城市, 所需要的最低交通费 (高速公路费+城市建设费) 是多少.

【样例输入】

输入格式:

第一行三个整数 n, m, q .

第二行 n 个整数, 表示 $p_1 \sim p_n$.

接下来 m 行中, 每行 3 个正整数, 第 j 行包含 A_j, B_j, W_j .

随后 Q 行每组两个正整数 x, y 表示一组询问.

输出格式:

共 Q 行, 为对 Q 个问题的回答: x 城市到 y 城市的最小交通费用.

5 7 2

2 5 3 3 4

1 2 3

1 3 2

2 5 3

5 3 1

5 4 1

2 4 3

3 4 4

1 4

2 3

【样例输出】

8 9

数据范围及约定

$n \leq 250, m \leq 20000, Q \leq 10000, P_i \leq 10000, W_j \leq 2000,$

保证任意两个城市可以互相到达.

```

res[i][j] = {0};
for(int k = 1; k <= n; k++) { //中转点 (背包)
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) {
            if(g[i][k] + g[k][j] < g[i][j]) {
                g[i][j] = g[i][k] + g[k][j]; //最短路径
                res[i][j] = max(res[i][j], p[k]); //最大繁荣度
            }
        }
    }
}

for(int i = 1; i <= n; i++) {
    for(int j = 1; j <= n; j++) {
        res[i][j] += g[i][j];
    }
}

```

```

1  #include <iostream>
2  #include <algorithm>
3  using namespace std;
4  int n = 0;
5  int m = 0;
6  int q = 0;
7  int p[10010];
8  int list[2010];
9  int map[300][300];
10 int ans[300][300];
11 bool Cmp(int one, int two){
12     return p[one] < p[two];
13 }
14 void Floyed(){
15     for(int i = 1; i <= n; i++){
16         for(int j = 1; j <= n; j++){
17             ans[i][j] = map[i][j] + 10000; //初始化无穷大
18         }
19     }
20     for(int k = 1; k <= n; k++){
21         for(int i = 1; i <= n; i++){
22             for(int j = 1; j <= n; j++){
23                 if(i != j && j != list[k] && i != list[k]){
24                     map[i][j] = min(map[i][j], map[i][list[k]] + map[list[k]][j]);
25                     //难点: list[k]可能有多个, p必须满足升序
26                     ans[i][j] = min(ans[i][j], map[i][j] + max(p[i], max(p[j], p[list[k]])));
27                 }
28             }
29         }
30     }
31 }

```

```

31 int main(){
32     for(int i = 0; i < 300; i++){
33         for(int j = 0; j < 300; j++){
34             map[i][j] = 11451419;
35         }
36     }
37     cin >> n >> m >> q;
38     for(int i = 1; i <= n; i++){
39         cin >> p[i];
40     }
41     for(int i = 1; i <= m; i++){
42         int a = 0;
43         int b = 0;
44         int w = 0;
45         cin >> a >> b >> w;
46         map[a][b] = min(w, map[a][b]);
47         map[b][a] = min(w, map[b][a]);
48     }
49     for(int i = 1; i <= n; i++){
50         list[i] = i;
51     }
52     sort(list + 1, list + 1 + n, Cmp);
53     Floyed();
54     for(int i = 1; i <= q; i++){
55         int x = 0;
56         int y = 0;
57         cin >> x >> y;
58         cout << ans[x][y] << endl;
59     }
60     system("pause");
61     return 0;
62 }

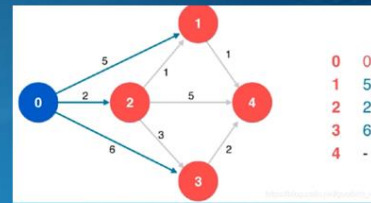
```

五、最短路—Dijkstra

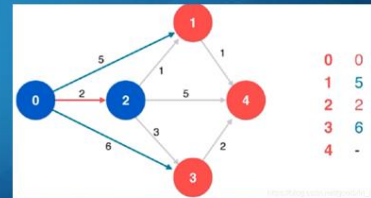
是从一个顶点到其余各顶点的最短路径算法，解决的是有权图中最短路径问题。迪杰斯特拉算法主要特点是从起始点开始，采用贪心算法的策略，每次遍历到始点距离最近且未访问过的顶点的邻接节点，直到扩展到终点为止。

1.初始化。V为G中所有顶点集合， $S=\{v\}$ 。D[x]表示从源点到x的已知路径，初始D[v]为0，其余为无穷大。

2.从源点v开始找其相邻点。如右图中从源点0开始，找到的可见点为1,2,3。



3.计算可见点到源点v的路径长度，更新D[x]。然后对路径进行排序，选择最短的一条作为确定找到的最短路径，将其终点加入到S中，如此处找到的点为2，故将2加入S。S={v, 2}。



4.从S中选择新加入的点找其相邻点，重复step3。直至所有点已加入S或者再搜索不到新的可见点（图中存在不联通的点，此时 $S<V$ ）终止算法。

- 不断运行找相邻点，计算可见点到源点的距离长度
- 从当前已知的路径中选择长度最短的将其顶点加入S作为确定找到的最短路径的顶点。

算法：

```

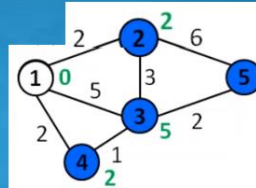
1  设起点为v |
2  _w[i][j] = 权值或oo;
3  _dis[i] = oo; 其中 _dis[v] = 0; //源点v到i的距离
4  _pre[i] = 0; //i的前驱点,用于输出路径
5  _b[j] = false; //初始为蓝点false, 求解过程中逐渐转为白点true

```

```

for(int i = 1; i <= _n; i++) {
    _minx = 1000;
    _k = 0; //中转点
    for(int j = 1; j <= _n; j++) {
        //蓝点中找到源点v的最小距离点
        if(!_b[j] && _dis[j] < _minx) {
            _minx = _dis[j];
            _k = j;
        }
    }
    //中转点k蓝变白
    _b[_k] = true;
    //通过中转点更新其余蓝点
    for(int j = 1; j <= _n; j++) {
        if(_dis[_k] + _w[_k][j] < _dis[j]) {
            _dis[j] = _dis[_k] + _w[_k][j];
            _pre[j] = _k; //记录到j点的中转点
        }
    }
}

```



初始	1	2	3	4	5
dis	0	oo	oo	oo	oo
pre	0	0	0	0	0

白1	1	2	3	4	5
dis	0	2	5	2	oo
pre	0	1	1	1	0

白2	1	2	3	4	5
dis	0	2	5	2	8
pre	0	1	1	1	2

白4	1	2	3	4	5
dis	0	2	3	2	8
pre	0	1	4	1	2

白3	1	2	3	4	5
dis	0	2	3	2	5
pre	0	1	4	1	3

```

1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4  int _w[110][110];
5  int _dis[110];
6  int _pre[110];
7  bool _b[110];
8  int _n, _m;
9  int _k, _minx;
10 void Print(int num) {
11     if(_pre[num] == 0) {
12         cout << num << "->";
13         return;
14     }
15     Print(_pre[num]);
16     cout << num << "->";
17 }
18 void Dijkstra() {
19     _dis[1] = 0; //起点为1
20     for(int i = 1; i <= _n; i++) {
21         _minx = 2e9;
22         _k = 0; //中转点
23         for(int j = 1; j <= _n; j++) {
24             //蓝点中找到源点v的最小距离点
25             if(!_b[j] && _dis[j] < _minx) {
26                 _minx = _dis[j];
27                 _k = j;
28             }
29         }
30         //中转点k蓝变白
31         _b[_k] = true;
32         //通过中转点更新其余蓝点
33         for(int j = 1; j <= _n; j++) {
34             if(_dis[_k] + _w[_k][j] < _dis[j]) {
35                 _dis[j] = _dis[_k] + _w[_k][j];
36                 _pre[j] = _k; //记录到j点的中转点
37             }
38         }
39     }
40 }

```

```

41 int main () {
42     cin >> _n >> _m; // 顶点数和边数
43     memset(_w, 0x3f, sizeof(_w));
44     memset(_dis, 0x3f, sizeof(_dis));
45     int v1 = 0, v2 = 0, d = 0;
46     for(int i = 1; i <= _m; i++) {
47         cin >> v1 >> v2 >> d;
48         // 防止重边
49         _w[v1][v2] = min(d, _w[v1][v2]);
50         _w[v2][v1] = min(d, _w[v2][v1]);
51     }
52     Dijkstra();
53     // for(int i = 1; i <= _n; i++) {
54     //     cout << _pre[i] << endl;
55     // }
56     // cout << "1到5的距离为: " << _dis[5] << endl;
57     // Print(5);
58
59     for(int i = 1; i <= _n; i++) {
60         cout << _dis[i] << " ";
61     }
62     return 0;
63 }

```

练习：

最小花费：

在n个人中，某些人的银行账号之间可以互相转账。这些人之间转账的手续费各不相同。给定这些人之间转账时需要从转账金额里扣除百分之几的手续费，请问A最少需要多少钱使得转账后B收到100元。

输入格式

第一行输入两个用空格隔开的正整数n和m，分别表示总人数和可以互相转账的人的对数。以下m行每行输入三个用空格隔开的正整数x,y,z，表示标号为x的人和标号为y的人之间互相转账需要扣除z%的手续费(z<100)。最后一行输入两个用空格隔开的正整数A和B。数据保证A与B之间可以直接或间接地转账。

输出格式

输出A使得B到账100元最少需要的总费用。精确到小数点后8位。

样例输入

```

3 3
1 2 1
2 3 2
1 3 3
1 3

```

样例输出

```

103.07153164

```



```

1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4  double g[110][110];
5  double dis[110]; //初始化为0
6  bool b[110];
7  int n, m, A, B;
8  void dijkstra(){
9      dis[A] = 1;
10     for(int i = 1; i <= n; i++){
11         double maxx = 0;
12         int k = 0;
13         for(int j = 1; j <= n; j++){
14             if(!b[j] && dis[j] > maxx){//找蓝点里面的最大值
15                 maxx = dis[j];
16                 k = j;
17             }
18         }
19         b[k] = true;
20         for(int j = 1; j <= n; j++){
21             if(!b[j] && dis[k] * g[k][j] > dis[j]){//找白点里面的最大值
22                 dis[j] = dis[k] * g[k][j];
23             }
24         }
25     }
26 }

27 int main(){
28     cin >> n >> m;
29     int x, y, z;
30     for(int i = 1; i <= m; i++){
31         cin >> x >> y >> z;
32         double w = (100 - z) / 100.0;
33         g[x][y] = w;
34         g[y][x] = w;
35     }
36     cin >> A >> B;//a是源点
37     for(int i = 1; i <= n; i++){
38         g[i][i] = 1;//自己到自己转账钱不变
39     }
40     for(int i = 1; i <= n; i++){
41         dis[i] = g[A][i];//直接转账可以先赋值
42     }
43     dijkstra();
44     cout << fixed << setprecision(8) << 100.0/dis[B];
45     return 0;
46 }

```