# 图论-Floyd算法及其扩展应用

## 一、AcWing 1125. 牛的旅行

【题目描述】

农民John的农场里有很多牧区，有的路径连接一些特定的牧区。

一片所有连通的牧区称为一个牧场。

但是就目前而言，你能看到**至少有两个牧区不连通**。

现在，John想在农场里添加一条路径（注意，恰好一条）。

一个牧场的直径就是牧场中**最远**的两个牧区的距离（本题中所提到的所有距离指的都是最短的距离）。

考虑如下的两个牧场，每一个牧区都有自己的坐标：

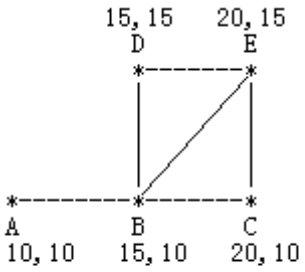



图1

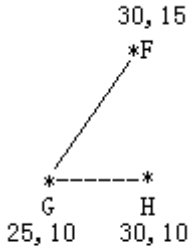



图2

图**1**是有**5**个牧区的牧场，牧区用`*`表示，路径用直线表示。

图**1**所示的牧场的直径大约是**12.07106**，最远的两个牧区是$A$和$E$，它们之间的最短路径是$A-B-E$。

图**2**是另一个牧场。

这两个牧场都在John的农场上。

John将会在两个牧场中各选一个牧区，然后用一条路径连起来，使得连通后这个新的更大的牧场有最小的直径。

注意，如果两条路径中途相交，我们不认为它们是连通的。

只有两条路径在同一个牧区相交，我们才认为它们是连通的。

现在请你编程找出一条连接两个不同牧场的路径，使得连上这条路径后，所有牧场（生成的新牧场和原有牧场）中直径最大的牧场的直径尽可能小。

输出这个直径最小可能值。

【输入格式】

第$1$行：一个整数$N$，表示牧区数；

第$2$到$N+1$行：每行两个整数$X,Y$，表示$N$个牧区的坐标。每个牧区的坐标都是不一样的。

第$N+2$行到第$2*N+1$行：每行包括$N$个数字（$0$或$1$）表示一个对称邻接矩阵。

例如，题目描述中的两个牧场的矩阵描述如下：

```
  A B C D E F G H
A 0 1 0 0 0 0 0 0
B 1 0 1 1 1 0 0 0
C 0 1 0 0 1 0 0 0
D 0 1 0 0 1 0 0 0
E 0 1 1 1 0 0 0 0
F 0 0 0 0 0 0 1 0
G 0 0 0 0 0 1 0 1
H 0 0 0 0 0 0 1 0
```

输入数据中至少包括两个不连通的牧区。

【输出格式】

只有一行，包括一个实数，表示所求答案。

数字保留六位小数。

【数据范围】

$1 \leq N \leq 150$

$0 \leq X,Y \leq 10^5$

【输入样例】

```
8
10 10
15 10
20 10
15 15
20 15
```

```
30 15
25 10
30 10
01000000
10111000
01001000
01001000
01110000
00000010
00000101
00000010
```

【输出样例】

```
22.071068
```

【分析】

---

首先，用Floyd算法求出任意两点之间的最短距离$dis[i][j]$，使用数组$maxd[i]$表示和$i$连通且距离$i$最远的点的距离，对于加边后图的直径有如下两种情况：

1. $res$为所有$maxd[i]$中的最大值，即在某两个非连通块之间加边后形成的新连通块的直径没有原先其它某个连通块的直径大；
2. 枚举在哪两个点之间加边，$i,j$满足$dis[i][j] \geq INF$，即$i,j$不连通，加边后形成的新的连通块直径为$maxd[i]+maxd[j]+i$与$j$之间的距离，$res$即为加边的所有情况中的最小值。

---

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
#include <cmath>
using namespace std;

typedef pair<int, int> PII;
const int N = 160;
const double INF = 1e20;
char g[N][N];
double dis[N][N], maxd[N];//maxd[i]表示和i连通且距离i最远的点的距离
PII q[N];
int n;
```

```cpp

//求两点间距离
double get_dist(PII u, PII v)
{
    return sqrt(pow(u.first - v.first, 2) + pow(u.second - v.second, 2));
}

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++) cin >> q[i].first >> q[i].second;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
        {
            cin >> g[i][j];
            if (i != j)
                if (g[i][j] == '1') dis[i][j] = get_dist(q[i], q[j]);
                else dis[i][j] = INF;
        }
    //Floyd算法求出任意两点间的距离
    for (int k = 0; k < n; k++)
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
    //求出maxd数组
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (dis[i][j] < INF) maxd[i] = max(maxd[i], dis[i][j]);
    double res = INF;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (dis[i][j] >= INF)//判断在不连通的两点间加边的情况
                res = min(res, maxd[i] + maxd[j] + get_dist(q[i], q[j]));
    for (int i = 0; i < n; i++) res = max(res, maxd[i]);//判断每个连通块的直径是否比加边更大
    printf("%lf\n", res);
    return 0;
}
```

# 二、AcWing 343. 排序（传递闭包）

【题目描述】

给定$n$个变量和$m$个不等式。其中$n \leq 26$，变量分别用前$n$的大写英文字母表示。

不等式之间具有传递性，即若$A > B$且$B > C$，则$A > C$。

请从前往后遍历每对关系，每次遍历时判断：

- 如果能够确定全部关系且无矛盾，则结束循环，输出确定的次序；
- 如果发生矛盾，则结束循环，输出有矛盾；
- 如果循环结束时没有发生上述两种情况，则输出无定解。

【输入格式】

输入包含多组测试数据。

每组测试数据，第一行包含两个整数$n$和$m$。

接下来$m$行，每行包含一个不等式，不等式全部为小于关系。

当输入一行`0 0`时，表示输入终止。

【输出格式】

每组数据输出一个占一行的结果。

结果可能为下列三种之一：

- 如果可以确定两两之间的关系，则输出`Sorted sequence determined after t relations: yyy...y.`，其中`t`指迭代次数，`yyy...y`是指升序排列的所有变量。
- 如果有矛盾，则输出：`Inconsistency found after t relations.`，其中`t`指迭代次数。
- 如果没有矛盾，且不能确定两两之间的关系，则输出`Sorted sequence cannot be determined.`。

【数据范围】

$2 \leq n \leq 26$，变量只可能为大写字母$A \sim Z$。

【输入样例1】

```
4 6
A<B
A<C
B<C
C<D
B<D
A<B
3 2
A<B
B<A
26 1
A<Z
0 0
```

【输出样例1】

```
Sorted sequence determined after 4 relations: ABCD.
Inconsistency found after 2 relations.
Sorted sequence cannot be determined.
```

【输入样例2】

```
6 6
A<F
B<D
C<E
F<D
D<E
E<F
0 0
```

【输出样例2】

```
Inconsistency found after 6 relations.
```

【输入样例3】

```
5 5
A<B
B<C
C<D
D<E
E<A
0 0
```

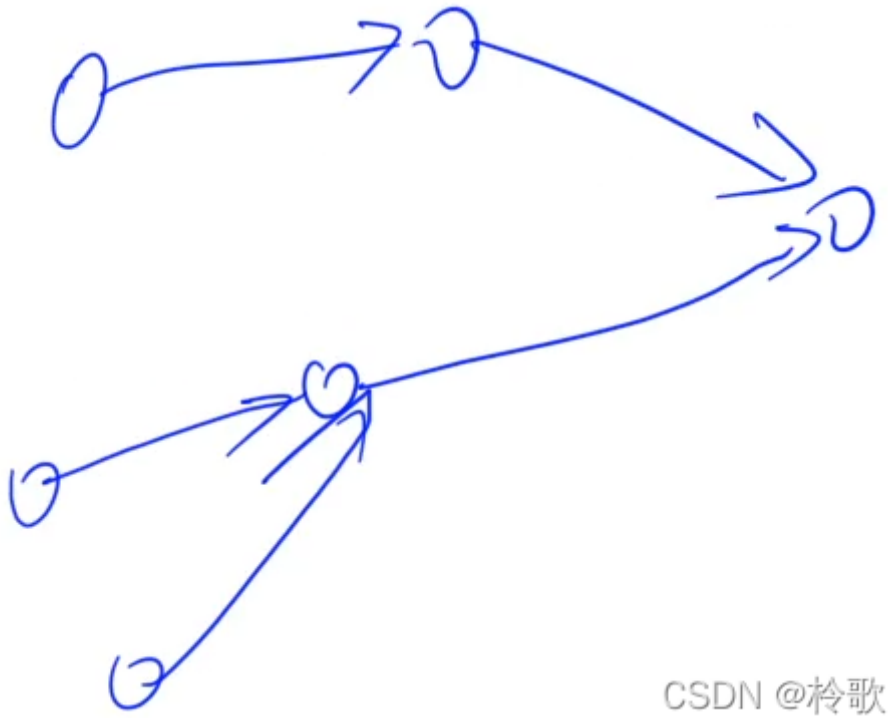
【输出样例3】

```
Sorted sequence determined after 4 relations: ABCDE.
```

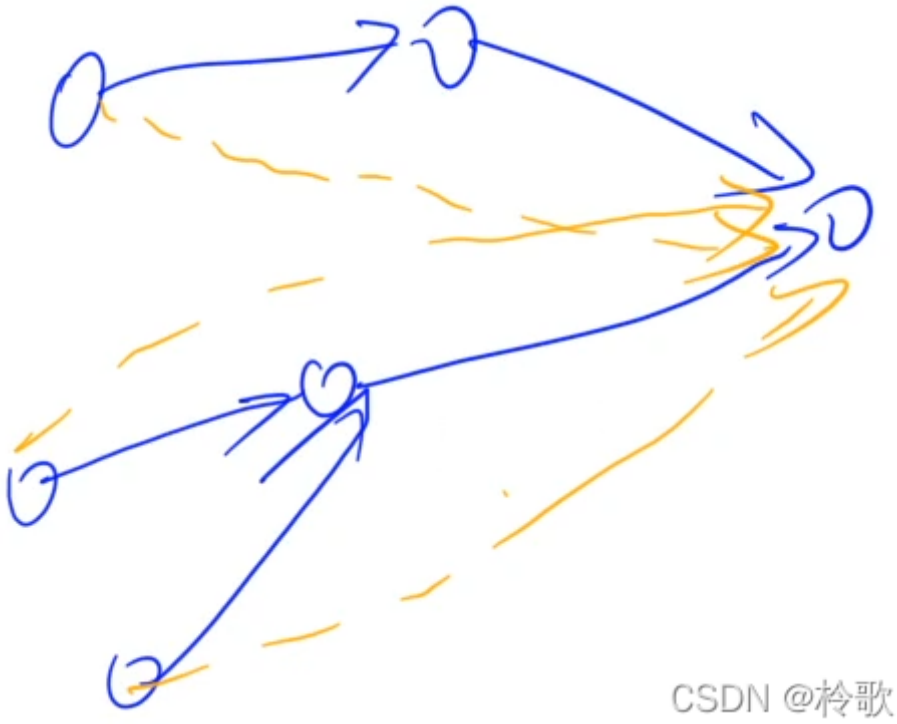
【分析】

---

在做本题之前首先需要了解**传递闭包**的概念，传递闭包显示的是传递关系，若$a$能到$b$，$b$能到$c$，那么$a$也能到$c$，假设有一个有向图如下图所示：




则该有向图的传递闭包为：




Floyd算法可以在$O(n^3)$的时间复杂度内求出一个有向图的传递闭包，假设图的邻接矩阵$g[i][j] == true$表示$i$能够走到$j$，在更新时，若$g[i][k] == true$且$g[k][j] == true$，则$g[i][j] = true$。

---

如何将时间复杂度降一维变成$O(n^2)$呢？观察到本题在添加一条边$a \to b$时只有以下三种情况：

1. 如果$x$能走到$a$，加了$a \to b$边后$x$也能走到$b$；
2. 如果$b$能走到$x$，加了$a \to b$边后$a$也能走到$x$；
3. 如果$x$能走到$a$，$b$能走到$y$，加了$a \to b$边后$x$也能走到$y$。

每次添加新的条件时只需对传递闭包进行这三种情况的更新即可形成新的传递闭包。

---

**【Floyd代码】**

```
#include <iostream>
#include <algorithm>
#include <cstring>
#include <string>
using namespace std;

const int N = 30;
bool g[N][N], d[N][N], st[N];//g表示原图，d表示传递闭包，st标记是否输出
int n, m;

```

```cpp
void floyd()
{
    memcpy(d, g, sizeof g);
    for (int k = 0; k < n; k++)
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                d[i][j] |= d[i][k] && d[k][j];
}

int check()
{
    for (int i = 0; i < n; i++) if (d[i][i]) return 2;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < i; j++)
            if (!d[i][j] && !d[j][i]) return 0;
    return 1;
}

char get_min()
{
    for (int i = 0; i < n; i++)
        if (!st[i])
        {
            bool flag = true;
            for (int j = 0; j < n; j++)
                if (!st[j] && d[j][i]) { flag = false; break; }
            if (flag) { st[i] = true; return i + 'A'; }
        }
}

int main()
{
    while (cin >> n >> m, n || m)
    {
        memset(g, false, sizeof g);
        int res = 0, t;//res为0表示不确定，1表示有唯一解，2表示有矛盾，t记录终止时的循环次数
        for (int i = 1; i <= m; i++)
        {
            string str;
            cin >> str;
            int a = str[0] - 'A', b = str[2] - 'A';

            if (!res)
```

```
            {
                g[a][b] = true;
                floyd();
                if (res = check()) t = i;
            }
        }
        if (!res) puts("Sorted sequence cannot be determined.");
        else if (res == 2) printf("Inconsistency found after %d
relations.\n", t);
        else
        {
            memset(st, false, sizeof st);
            printf("Sorted sequence determined after %d relations: ", t);
            for (int i = 0; i < n; i++) printf("%c", get_min());
            printf(".\n");
        }
    }
    return 0;
}
```

【优化代码】

```
#include <iostream>
#include <algorithm>
#include <cstring>
#include <string>
using namespace std;

const int N = 30;
bool g[N][N], st[N];
int n, m;

int check()
{
    for (int i = 0; i < n; i++) if (g[i][i]) return 2;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < i; j++)
            if (!g[i][j] && !g[j][i]) return 0;
    return 1;
}

char get_min()

{
```

```
    for (int i = 0; i < n; i++)
        if (!st[i])
        {
            bool flag = true;
            for (int j = 0; j < n; j++)
                if (!st[j] && g[j][i]) { flag = false; break; }
            if (flag) { st[i] = true; return i + 'A'; }
        }
}

int main()
{
    while (cin >> n >> m, n || m)
    {
        memset(g, false, sizeof g);
        int res = 0, t;//res为0表示不确定，1表示有唯一解，2表示有矛盾，t记录终止时的循环次数
        for (int i = 1; i <= m; i++)
        {
            string str;
            cin >> str;
            int a = str[0] - 'A', b = str[2] - 'A';
            if (!res)
            {
                g[a][b] = true;
                for (int x = 0; x < n; x++)
                {
                    if (g[x][a]) g[x][b] = true;//如果x能走到a，加了ab边后x也能走到b
                    if (g[b][x]) g[a][x] = true;//如果b能走到x，加了ab边后a也能走到x
                    for (int y = 0; y < n; y++)
                        if (g[x][a] && g[b][y]) g[x][y] = true;//如果x能走到a，b能走到y，加了ab边后x也能走到y
                }
                if (res = check()) t = i;
            }
        }
        if (!res) puts("Sorted sequence cannot be determined.");
        else if (res == 2) printf("Inconsistency found after %d relations.\n", t);
        else

        {
```

```
            memset(st, false, sizeof st);
            printf("Sorted sequence determined after %d relations: ", t);
            for (int i = 0; i < n; i++) printf("%c", get_min());
            printf(".\n");
        }
    }
    return 0;
}
```

# 三、AcWing 344. 观光之旅（最小环）

【题目描述】

给定一张无向图，求图中一个至少包含3个点的环，环上的节点不重复，并且环上的边的长度之和最小。

该问题称为无向图的最小环问题。

你需要输出最小环的方案，若最小环不唯一，输出任意一个均可。

【输入格式】

第一行包含两个整数$N$和$M$，表示无向图有$N$个点，$M$条边。

接下来$M$行，每行包含三个整数$u,v,l$，表示点$u$和点$v$之间有一条边，边长为$l$。

【输出格式】

输出占一行，包含最小环的所有节点（按顺序输出），如果不存在则输出`No solution.`。

【数据范围】

$1 \le N \le 100$

$1 \le M \le 10000$

$1 \le l < 500$

【输入样例】

```
5 7
1 4 1
1 3 300
3 1 10
1 2 16
2 3 100
2 5 15
5 3 20
```

【输出样例】

```
1 3 5 2
```

【分析】

---

设环的形式是：$i-k-j$（$i,j,k$不同）

Floyd是典型的插点算法，每次插入点$k$，为此，在点$k$被插入前可计算$i-k-j$这个环，即此时中间节点为：$1\sim k-1$，即我们已经算出了任意$i\sim j$的最短道路，中间经过的节点可以为（$1,2,3,\ldots,k-1$）我们只需枚举所有以$k$为环中最大节点的环即可，设$g$表示原邻接矩阵，$dis$表示最短距离，则每个环$i-k-j$的距离为：$dis[i][j]+g[i][k]+g[k][j]$。

设$pos[i][j]$表示$i\sim j$的最短路中经过的点是$k$（即由这个状态转移过来），且这个$k$是此路径中编号最大的点（除$i,j$），如果$pos[i][j]==0$，说明$i\sim j$的最短路没有经过其他节点。

这条道路存在以下两条性质：

1. 在$i\sim j$的最短道路中，一定没有环（显然）；
2. 设$i,j$之间的最短道路经过点$k$（不同于$i,j$），则$i\sim k,k\sim j$之间必然没有交集。

因此利用此性质可求解整条路径，假设以$k\to i\to j$的路径行走，那么整条路径为：$path=\{k,i,$利用$pos$数组递归求出$i\sim j$的中间路径$,j\}$。

---

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 110, INF = 0x3f3f3f3f;
int g[N][N], dis[N][N];
int pos[N][N], path[N], cnt;//pos[i][j]表示i到j的最短路所经的点k
```

```cpp
int n, m;

void get_path(int i, int j)
{
    if (!pos[i][j]) return;
    int k = pos[i][j];
    get_path(i, k);
    path[cnt++] = k;
    get_path(k, j);
}

int main()
{
    cin >> n >> m;
    memset(g, 0x3f, sizeof g);
    for (int i = 1; i <= n; i++) g[i][i] = 0;
    while (m--)
    {
        int a, b, c;
        cin >> a >> b >> c;
        g[a][b] = g[b][a] = min(g[a][b], c);
    }
    memcpy(dis, g, sizeof g);
    int res = INF;
    for (int k = 1; k <= n; k++)
    {
        for (int i = 1; i < k; i++)
            for (int j = i + 1; j < k; j++)
                if ((long long)dis[i][j] + g[i][k] + g[k][j] < res)
                {
                    res = dis[i][j] + g[i][k] + g[k][j];
                    cnt = 0;
                    path[cnt++] = k;//假定从k走到i再走到j
                    path[cnt++] = i;
                    get_path(i, j);
                    path[cnt++] = j;
                }
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                if (dis[i][k] + dis[k][j] < dis[i][j])
                {
                    dis[i][j] = dis[i][k] + dis[k][j];

                    pos[i][j] = k;
```

```
                }
    }
    if (res == INF) puts("No solution.");
    else for (int i = 0; i < cnt; i++) cout << path[i] << ' ';
    return 0;
}
```

# 四、AcWing 345. 牛站（倍增，快速幂）

【题目描述】

给定一张由$T$条边构成的无向图，点的编号为$1 \sim 1000$之间的整数。

求从起点$S$到终点$E$恰好经过$N$条边（可以重复经过）的最短路。

注意：数据保证一定有解。

【输入格式】

第$1$行：包含四个整数$N,T,S,E$。

第$2 \sim T+1$行：每行包含三个整数，描述一条边的边长以及构成边的两个点的编号。

【输出格式】

输出一个整数，表示最短路的长度。

【数据范围】

$2 \leq T \leq 100$

$2 \leq N \leq 10^6$

【输入样例】

```
2 6 6 4
11 4 6
4 4 8
8 4 9
6 6 8
2 6 9
3 8 9
```

【输出样例】

```
10
```

【分析】

- 状态表示：$f[k][i][j]$表示$i$走到$j$恰好经过$k$条路径的最短距离；
- 状态转移方程：$f[a+b][i][j]=min(f[a+b][i][j],f[a][i][k]+f[b][k][j])$。

本题可以采取快速幂的思想：因为要走$k$条边，所以可以先将$k$转化为二进制数，例如$k=(38)_D=(100110)_B$，设$g$数组为走过一条边的最短距离，则可以看成：答案数组`res=g数组走过2条边的最短距离+g数组走过4条边的最短距离+g数组走过32条边的最短距离`。按照此类倍增的思想即可大幅降低时间复杂度，以指数的形式逼近答案。

由于$g$数组表示的是经过一条边后各点之间最短距离的矩阵，即为图的领接矩阵，但是因为要经过一条边，因此$g[i][i]$不能初始化为$0$，只有当$i\to i$存在一条边的时候才能更新$g[i][i]$。

$g$数组的倍增：

只要将两个$g$数组单独分开来看即可。

例如$g$数组从经过两条边的最短距离转变为经过四条边的最短距离：$temp[i][j]=min(temp[i][j],g[i][k]+g[k][j])$。

为什么这样做是可行的呢？

因为$g[i][k]$表示的是从$i\to k$经过两条边的最短距离，$g[k][j]$表示的是从$k\to j$经过两条边的最短距离，所以$g[i][k]+g[k][j]$表示的是$i\to k\to j$经过四条边的最短距离。

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <map>
using namespace std;

const int N = 210;//最多100条边,200个点
int g[N][N], res[N][N];//g为图的邻接矩阵,即只经过一条边的最短距离
int k, n, m, S, E;
map<int, int> ids;

void mul(int a[][N], int b[][N], int c[][N])
{
    int temp[N][N];
    memset(temp, 0x3f, sizeof temp);
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
```

```
            for (int j = 1; j <= n; j++)
                temp[i][j] = min(temp[i][j], b[i][k] + c[k][j]);
    memcpy(a, temp, sizeof temp);
}

void qmi()
{
    memset(res, 0x3f, sizeof res);//只经过0条边时只有自己走到自己距离为0
    for (int i = 1; i <= n; i++) res[i][i] = 0;
    while (k)
    {
        if (k & 1) mul(res, res, g);//更新答案数组,即res = res * g
        mul(g, g, g);//将g数组倍增,即g = g * g
        k >>= 1;
    }
}

int main()
{
    cin >> k >> m >> S >> E;
    if (!ids.count(S)) ids[S] = ++n;
    if (!ids.count(E)) ids[E] = ++n;
    S = ids[S], E = ids[E];
    memset(g, 0x3f, sizeof g);//因为g数组必须走过一条边,所以g[i][i]不初始化为0,除非i->i之间有边
    while (m--)
    {
        int a, b, c;
        cin >> c >> a >> b;
        if (!ids.count(a)) ids[a] = ++n;//离散化过程,因为题目中给的编号并不是从1开始而是随机的
        if (!ids.count(b)) ids[b] = ++n;
        a = ids[a], b = ids[b];
        g[a][b] = g[b][a] = min(g[a][b], c);
    }
    qmi();
    cout << res[S][E] << endl;
    return 0;
}
```