最短路算法（三）FLOYD

一、Floyd 算法

Floyd 算法是一种用于求多源最短路问题的算法。

Floyd 算法要求图以矩阵方式存储，采取动态规划的思想来计算。

算法框架：

```
void floyd(){
    for(int k=1;k<=n;k++)     //枚举中间点
     for(int i=1;i<=n;i++)      //枚举起点
       for(int j=1;j<=n;j++)  //枚举终点
         if((dist[i][k]!=inf)&&(dist[k][j]!=inf) &&(dist[i][k]+dist[k][j]<dist[i][j]))
                dist[i][j]=dist[i][k]+dist[k][j];
```

floyd 算法本质是动态规划的思想求最短路。设 d[k,i,j]表示经过不超过 k 个节点，从 i 到 j 的最短路长度。

该状态定义下，可以划分为两个子问题：

1、经过不超过 k-1 个节点从 i 到 j

2、从 i 到 k，再到 j。（不超过 k-1 个节点）

可得：d[k,i,j]=min{d[k-1,i,j] , d[k-1,i,k]+d[k-1,k,j]}

初值：d[0,i,j]=map[i,j]

其中 k 为阶段，所以在转移时需要将 k 写在最外层循环。观察动态转移方程，发现第一维状态可以省略，于是得到：

d[i,j]=min{d[i,j],d[i,k]+d[k,j]}

二、Floyd 的应用

Flyod 不仅可以求多源最短路，还可以解决很多问题，比如判断连通性，求最小环等等

应用一：求传递闭包

1、 用于判断连通性（例 2）

2、 寻找满足条件的连通分支

应用二：计算有向无环图的最长路径问题

求最长路常见几种算法：

Floyd，变形的 spfa 或 Dijkstra，拓扑 DP

应用三：计算有向和无向带权图的最小环问题

1、 有向图的最小环

算法步骤：

Dist[i,i]就是包含 i 的最小环，初始时，dist[i,i]=+oo

1.读入数据建图

2.用 floyd 求出每对点最短路

3.打擂台求出最小 DIST[i， i]

2、 无向图的最小环

朴素算法：

1.删边求最短路，每条边都要求一次，最短路可用 dijkstra 或 spfa。

2.即令 g[i,j]表示 i 和 j 之间的连边，再令 dist[i,j]表示删除 i 和 j 之间的连边之后，i 和 j 之间的最短路

3.最小环则是 dist[i,j]+g[i,j]，时间复杂度是 EV^2

一个错误的算法：

若像有向图一样，直接用 Floyd 算法是错误的。即预处理出任意两点之间的最短路径，记作 dist[i,j]。枚举三个点 k,i,j,最小环则是 dist[i,k]+dist[k,j]+g[i,j]的最小值。正确吗？

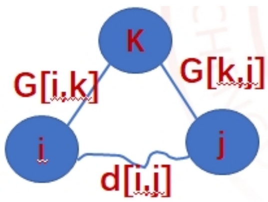
如果考虑 dist[i,k]包含边 i-j 的情况？如何改进？

改进算法：

仔细观察 Floyd 的三层循环，发现中间点枚举的顺序是从小到大，也就是当枚举到某个点 k，对所有小于它的点对 i 和 j，从 i 到 j 的最短路是不经过 k 的，如果此时有边 g[i,k]和 g[k,j]则可形成一个 ki....jk 的环，再用 k 去松弛所有的点对，以便下次枚举 k+1 时重新找与 1~k 之间的点形成的环，找出所有环中最小的。

g[][]表示原图中各点之间距离，dist[i,j]表示 i,j 之间最短距离

核心思想：随着 K 的递增,依次枚举顶点都小于 K 的环



应用四：任意两点间的最短路径条数 g[i][j]

具体做法：设 D[i,j]表示最短路，g[i][j]表示最短路条数

```
void floyd(){
    for(int k=1;k<= n;k++){
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                if(d[i][k]!=INF&&d[k][j]!=INF){
                    if(d[i][j]==d[k][j]+d[i][k])
                        g[i][j]+=g[i][k]+g[k][j];
                    if(d[i][j]>d[k][j]+d[i][k]){
                        d[i][j]=d[i][k]+dist[k][j];
                        g[i][j]=g[i][k]*g[k][j];
                    }
                }
    }
}
```

三、实例应用

例 1：牛的旅行

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）。对这条路径有这样的限制：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，图 1 是有 5 个牧区的牧场，牧区用“*”表示，路径用直线表示。每一个牧区都有自己的坐标：

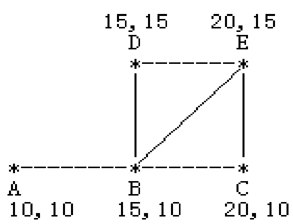


图1

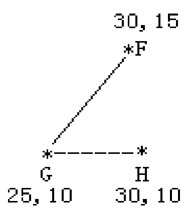


图2

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。

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，这个更大的新牧场有最小的直径。

【解析】

先用 floyed 计算任意两点最短路。（floyed 可以判断 2 点是否能够连通）

计算出每个点到可到达的点的最长路 m[i]

计算出原图的直径 L

枚举所有没连通的点，

假设在点 I，j 之间连一条边，这是两个原本分开的图的唯一联通的路径，所以通过该路径的直径为

距离 i 最远的点（点 a）——i——j——距离 j 最远的点（点 b）

我们只需要记录下 ai 的距离和 bj 的距离，用它们与 ij 的距离相加，即是当前的直径。（找一个最短的）

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
#define ML 501
#define INF 1e10
struct Ponit{
    int x,y;
};
int  n;
double f[ML][ML],m[ML]={0} , L=0,newL=INF;
//f[i][j]表示 ij 最短路，m[i]表示 i 到能连通最远距离，L 表示原图直径，newL 表示新图最小直径
Ponit a[ML];
double jisuan(const Ponit& aa,const Ponit& b){
    return sqrt( (aa.x-b.x)*(aa.x-b.x) + (aa.y - b.y)*(aa.y-b.y));
}
void readin(){
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i].x>>a[i].y;
    }
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++){
            char c;
            cin>>c;
            //f[i][i] = 0;
            if(c=='1'){
```

```
                f[i][j] = jisuan(a[i],a[j]);
            }
            else
                f[i][j] = INF;
        }
}
void floyed(){
    for(int k=1;k<=n;k++)
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
            if(i!=j&&i!=k&&k!=j)
                f[i][j]=min(f[i][j] , f[i][k]+f[k][j]);//求2点最短路

    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            if(f[i][j] != INF){//如果两点连通
             if(f[i][j] > m[i]) //找出i到联通点最远距离
                m[i] = f[i][j];
            if(f[i][j] > L)
                L = f[i][j];//求原图直径
        }


}
int main(){
    readin();
    floyed();
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++){
            if(f[i][j]==INF&&i!=j){//找所有不连通的路
                double t = jisuan(a[i],a[j]) + m[i] + m[j];
                if(t < newL) newL=t;
                //cout<<t<<endl;
            }
        }
    if(L>newL) newL=L;
    printf("%.6lf",newL);
    return 0;
}
```

例 2：产生数 P3063

给出一个整数 n 和 k 个规则。求出：经过任意次的变换（0 次或多次），能产生出多少个不同整数。仅要求输出个数。

【解析】

认真分析题目之后发现，本题搜索显然是不行的，而且对于只需计数而不需求具体方案的题目，一般都不会用搜索解决，其实本题不难看出，可以用乘法原理直接进行计数，用 F[i]表示数字 i

包括本身可以变成的数字总个数（这里的变成可以是直接变成也可以是间接变成，比如 35,57,那么 37），那么对于一个数 a[i]，它可以变成的数字总个数为 F[a[i]]，根据乘法原理它能产生出 F[a[1]]*F[a[2]]*…*F[a[n]]个不同整数，相信这一点大家不难理解。那么现在的关键就是如何求 F[i]。

由于这些变换规则都是反应的数字与数字之间的关系，这很容易让我们想到用图来表示这种关系，算法步骤如下：

- ①建立一个有向图 G，初始化 map[i][j]=0，如果数字 i 能直接变成数字 j，那么 map[i][j]=1；再把 map[i][i]=1。
- ②用 Floyd 求传递闭包，注意数字只为 0-9；
- ③统计 f[i]，即数字 i 可以变成的数字；

```
for(i=0;i<=9;i++)
    for(j=0;j<=9;j++)f[i]+=map[i][j];
```

- ④求方案数。for(i=0;i<n;i++)chengdan(ans,f[s[i]-'0']);
- 最后值得注意的是当 n 很大时,要使用**高精度**乘法

```
#include<bits/stdc++.h>
using namespace std;
inline int read(){
    char ch;bool op=0;int sum=0;ch=getchar();
    while(ch!='-' && (ch<'0' || ch>'9'))ch=getchar();
    if(ch=='-')op=1,ch=getchar();
    while(ch<='9' && ch>='0')sum=(sum<<3)+(sum<<1)+ch-'0',ch=getchar();
    return op ? -sum : sum;
}
int ans[10010],now=1,t,len,x,y,num[10];
char c[35];
bool flag[10][10];
void mul(int a[],int &len,int x){
    for(int i=1;i<=len;i++)a[i]*=x;
    for(int i=1;i<=len;i++){
        a[i+1]+=a[i]/10;
        a[i]%=10;
    }
    while(a[len+1]>0){
        len++;
        a[len+1]=a[len]/10;
        a[len]%=10;
    }
}
int main(){
    cin>>c+1;
    len=strlen(c+1);
    t=read();
    for(int i=1;i<=t;i++){
        x=read();y=read();
```

```
        flag[x][y]=true;
    }
    for(int k=0;k<=9;k++)
        for(int i=0;i<=9;i++)
            for(int j=0;j<=9;j++)
             if(i!=k && i!=j && j!=k)flag[i][j]=flag[i][j] || (flag[i][k]
&& flag[k][j]);
    for(int i=0;i<=9;i++)
        for(int j=0;j<=9;j++)if(i!=j)num[i]+=flag[i][j] ? 1 : 0;
    ans[1]=1;now=1;
    for(int i=1;i<=len;i++){
        int u=num[(int)(c[i]-'0')]+1;
        mul(ans,now,u);
    }
    for(int i=now;i>=1;i--)printf("%d",ans[i]);
    return 0;
}
```

例 3： P3061．观光旅游

题意：求无向图的最小环

【解析】

无向图最小环模板题。

【参考答案】

```
#include<bits/stdc++.h>
using namespace std;

const int N=110,INF=1e9;
int n,m,x,y,f[N][N],dis[N][N],ans=INF;

#define in read()
inline int read(){
    int f=1,k=0;char cp=getchar();
    while(cp!='-'&&(cp<'0'||cp>'9')) cp=getchar();
    if(cp=='-') f=-1,cp=getchar();
    while(cp>='0'&&cp<='9') k=(k<<3)+(k<<1)+cp-48,cp=getchar();
    return f*k;
}
int main(){
    n=in,m=in;
    for(int i=1;i<+n;i++)
        for(int j=1;j<=n;j++)
            f[i][j]=INF;
    while(m--) x=in,y=in,dis[x][y]=dis[y][x]=f[x][y]=f[y][x]=in;
    for(int k=1;k<=n;k++){
```

```
        for(int i=1;i<=k-1;i++)
            for(int j=i+1;j<=k-1;j++)
                if(f[i][j]!=INF&&dis[i][k]&&dis[k][j])
                    ans=min(ans,f[i][j]+dis[i][k]+dis[k][j]);
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                f[i][j]=min(f[i][j],f[i][k]+f[k][j]);
    }
    if(ans!=INF) printf("%d\n",ans);
    else printf("No solution.\n");

    return 0;
}
```