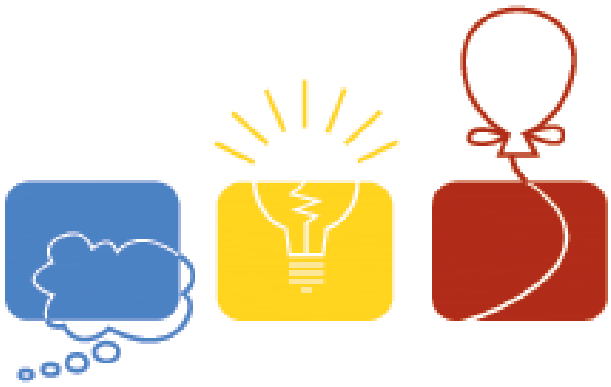
ACM Standard Code Library



夏天的风

Hangzhou Dianzi University

Ver 2.0

Update 2012/10/10

目录

图论

数据结构

动态规划

计算几何

数学

/**\

并查集

①**合并查找**:将2个集合合并成1个集合.(并到一个集合的根上)

②**种类并查集**:将不同种类的放到不同的层次上.

③**判正误**:利用种类并查集分层,然后每输入一句话,就判断是否矛盾.

**/

模板

==========

种类并查集(分层)

```
int find(int r) {
    if(r==bin[r]) return bin[r];
    int fx=bin[r];
    bin[r]=find(bin[r]);
    f[r]+=f[fx];
    return bin[r];
}
bool merge(int x,int y,int c) {
    int fx=find(x); int fy=find(y);
    if(fx==fy) {
        if(abs(f[y]-f[x])!=c) return 0;
        return 1;
    }else {
        bin[fx]=fy; f[fx]=(f[y]-f[x]+c);
        return 1;
    }
}
```

==========

/*--*\

并查集应用

--/

逆向并查集

[zoj3261] 给定N个点,每个点都有权值,给定M条边.

操作:

Q 询问点i连通的点中最大权值,且ID最小的点.

D 删除一条边.

【解析】

逆向并查集,将询问操作从最后一个到第一个.

先将询问中不会删除的边,全部并查集.

然后倒着询问,每次碰到D的时候,相当于加一条边...

这样题目就变得容易多了...

PS:因为点有1W,所以不可以开e[][]记录询问中会删掉的边.

所以我们可以先对M条边排序,然后每次询问中出现的删除的边.

我们可以通过二分查找到相应的边,进行-1标记.表示该边在询问时会删掉...

----------/

/**********\

最小生成树

①Kruskal:对边从小到大排序,贪心加边.

②Prim:每次在已选集合连出去的最小边对应的且不在集合点加入到集合中.

直到集合里元素个数为N个.

③最小树形图(有向图的MST):指定一个节点和其他节点连通的MST.

④K度限制生成树:指定一个节点的度不能超过k度.加边后,破圈.

⑤最小比率生成树:每条边有花费和距离.选取的N-1条边.

使得花费总和Σcost/Σdist距离总和 最小.

⑥斯坦纳树:给定一个图,问使得一些点(k个点)连通的MST.

PS:三点斯坦纳:先求出三个点的最短路.然后枚举'斯点'.

到三点的距离和最小.即为解.

**********/

模板

==========

Kruskal: 并查集+排序

Prim (n^2)

```
bool vis[N]; int dis[N];
int prim() {
    int i,j,u,v,w,ans=0;
    for(i=1;i<=n;i++) dis[i]=e[1][i];
    memset(vis,0,sizeof(vis)); vis[1]=1;
    for(i=2;i<=n;i++) {
        w=inf; u=-1;
        for(j=1;j<=n;j++) if(!vis[j]&&w>dis[j]) w=dis[j],u=j;
        if(u==-1) return -1;
        vis[u]=1; ans+=w;
        for(v=1;v<=n;v++) if(!vis[v])
            if(dis[v]>e[u][v]) dis[v]=e[u][v];
    }
    return ans;
}
```

==========

最小树形图(有向图的最小生成树)[hdu2121]

有根的情况: 直接求.

无根的情况: 建立超级根节点,向每个点连有向边,边权大于所有边权和即可.

```
struct Node {
    int oldu,oldv; //原始点
    int u,v,w; //缩图后的点
}e[N*N];
int n,m,rt;
int pre[N],ID[N],vis[N];
int in[N]; //最小入边权值
int Directed_MST(int root,int n,int m) {
    int i,oldroot=root;
    ll ret=0;
    while(1) {
        //1.找最小入边
        for(i=1;i<=n;i++) in[i]=inf;
        for(i=1;i<=m;i++) {
            int u=e[i].u;
            int v=e[i].v;
            if(u!=v&&e[i].w<in[v]) {
                pre[v]=u,in[v]=e[i].w;
                if(e[i].oldu==oldroot) rt=e[i].oldv;//记录最小根节点
编号          }
        }
        for(i=1;i<=n;i++) {
            if(i==root) continue;
            if(in[i]==inf)return -1;//除了根以外有点无入边,则根无法到达
它      }
        //2.找环
        int nn=0; //新图的节点个数
        memset(ID,-1,sizeof(ID));
        memset(vis,-1,sizeof(vis));
        in[root]=0;
        for(i=1;i<=n;i++) { //标记每个环
            ret+=in[i];
            int v=i;
            while(v!=root&&ID[v]==-1&&vis[v]!=i) {
                vis[v]=i; v=pre[v];
            }
            if(v!=root&&ID[v]==-1) { //将环缩点
                ID[v]=++nn;
                for(int u=pre[v];u!=v;u=pre[u]) ID[u]=nn;
            }
        }
        if(nn==0) break;//无环
        for(i=1;i<=n;i++) //每个不在环上的点重新加到图中
```

```
            if(ID[i]==-1) ID[i]=++nn;
        //3.缩点,重新标记
        for(i=1;i<=m;i++) {
            int u=e[i].u;
            int v=e[i].v;
            e[i].u=ID[u];
            e[i].v=ID[v];
            if(u!=v) e[i].w-=in[v];
        }
        n=nn;
        root=ID[root];
    }
    return ret;
}
int main() {
   int i,j,a,b,c,sum;
   while(scanf("%d%d",&n,&m)!=EOF) {
        sum=0;
        for(i=1;i<=m;i++) {
            scanf("%d%d%d",&e[i].u,&e[i].v,&e[i].w);
            e[i].u++; e[i].v++;
            e[i].oldu=e[i].u; e[i].oldv=e[i].v;
            sum+=e[i].w;
        }
        for(i=1;i<=n;i++) { //超级根节点 n+1
            e[++m].u=n+1; e[m].v=i; e[m].w=sum+1;
            e[m].oldu=e[m].u; e[m].oldv=e[m].v;
        }
        n++;
        int ret=Directed_MST(n,n,m);
        if(ret!=-1&&ret<2*sum+2)   printf("%d %d\n",ret-sum-1,rt-1);
        else                    puts("impossible");
    }
}
```

==========

K 度限制生成树

```
struct node {
   int a,b,c;
   friend bool operator<(const node &a,const node &b) {
     return a.c<b.c;
   }
}edge[N*N],ee; //edge 原图的边
int n,m,k,root; //点数,边数,限制度 k,根 root
map<string,int>mp;
```

```
vector<node>e[N],rt; //生成树的边,和 root 连出去的边
bool mark[N]; //标记 root 连出去的边
//------------------------- 加 MST 的边 -------------------------
void addedge(int u,int v,int w) {
   ee.b=v; ee.c=w; e[u].push_back(ee);
   ee.b=u; ee.c=w; e[v].push_back(ee);
}
//-------------------- Kruskal 求 MST(除 root) --------------------
int bin[N];
int find(int r);
bool merge(int x,int y);
int kruskal() {
   int i,j,u,v,mst=0;
   for(int i=0;i<=n;i++) bin[i]=i;
   sort(edge+1,edge+1+m);
   for(i=1;i<=m;i++) {
      u=edge[i].a; v=edge[i].b;
      if(u==root||v==root) { //将和 root 相连的边加到 rt 数组中
         if(v==root) swap(edge[i].a,edge[i].b);
         rt.push_back(edge[i]);
      }
      else if(merge(u,v)) { //加生成树的边(除 root)
         mst+=edge[i].c;
         addedge(u,v,edge[i].c);
      }
   }
   return mst;
}
//------------------------- 求 K 度生成树 -------------------------
bool vis[N];
node best[N]; //best[v]为从 root 到 v 的链上最大边
void dfs(int u,node f) { //搜索最大边 O(n)
   if(vis[u]) return;
   vis[u]=1; best[u]=f;
   for(int i=0;i<e[u].size();i++) {
      node ff=f; //父节点的最大边
      int v=e[u][i].b , w=e[u][i].c;
      if(w>ff.c) ff.a=u,ff.b=v,ff.c=w;
      dfs(v,ff);
   }
   vis[u]=0;
}
void Remove(int u,int v) { //删除最大边
   for(int i=0;i<e[u].size();i++)
```

```
      if(e[u][i].b==v) { e[u].erase(e[u].begin()+i); return; }
}
int solve(int mst) {
   int i,u=root,v,w;
   int d=0; //记录度
   for(i=0;i<rt.size();i++) { //求最小生成树
      v=rt[i].b; w=rt[i].c;
      if(merge(u,v)) {
         d++,mst+=w;
         addedge(u,v,w);
         mark[i]=1; //标记已经选中
      }
   }

   while(d<k) {
      int tmp=0,j=0; //tmp 最大差,j 为和 root 相连要加入 Tree 的边
      node max_e; //记录替换边
      ee.c=-1; dfs(root,ee); //求最大边
      for(i=0;i<rt.size();i++) { //寻找替换边
         if(mark[i]) continue; //已经在 MST 中
         v=rt[i].b; w=rt[i].c;
         if(tmp<best[v].c-w) {
            tmp=best[v].c-w;
            j=i,max_e=best[v];
         }
      }
      if(tmp==0) break; //说明已经是 MST 了
      mark[j]=1;
      Remove(max_e.a,max_e.b);
      Remove(max_e.b,max_e.a);
      e[root].push_back(rt[j]);
      d++,mst-=tmp;
   }
   return mst;
}
//------------------------- 初始化 -------------------------
void init() {
   mp.clear(); mp["Park"]=(n=1);
   for(int i=0;i<=m*2;i++) e[i].clear();
   rt.clear();
   memset(mark,0,sizeof(mark));
   memset(vis,0,sizeof(vis));
}
int main(){
```

```
   int i,j,t,cas=0;
   char s1[22],s2[22];
   int a,b,c;
   while(scanf("%d",&m)!=EOF) {
      init();
      for(i=1;i<=m;i++) {
         scanf("%s%s%d",&s1,&s2,&edge[i].c);
         if(!mp[s1]) mp[s1]=++n;
         if(!mp[s2]) mp[s2]=++n;
         edge[i].a=mp[s1]; edge[i].b=mp[s2];
      }
      scanf("%d",&k);
      root=1;
      int ret=solve(kruskal());
      printf("Total miles driven: %d\n",ret);
   }
}
```

==========

最小比率生成树

```
//kruskal+Dinke(可用二分)
struct po{int x,y,h;}f[1005]; //点坐标,高度
struct node {
   int a,b;
   double s,h,w;
   friend bool operator<(const node &a,const node &b) {
      return a.w<b.w;
   }
}dis[1000005];
int n,m;
bool mak;
int find(int r);
bool merge(int x,int y);
int main() {
   int i,j,k;
   while(scanf("%d",&n),n) {
      for(i=1;i<=n;i++) scanf("%d%d%d",&f[i].x,&f[i].y,&f[i].h);
      m=0;
      for(i=1;i<=n;i++)
         for(j=i+1;j<=n;j++) {
            dis[++m].a=i; dis[m].b=j;
            dis[m].s=dist(f[i],f[j]); //求距离,边长
            dis[m].h = fabs(f[i].h-f[j].h); //求高度差
         }
      double mid=0,b;
```

```
      int cnt;
      while(1) {
         for(i=1;i<=m;i++) //w=a-ans*b
            dis[i].w=dis[i].h-mid*dis[i].s;
         sort(dis+1,dis+1+m);
         for(i=0;i<=n;i++) bin[i]=i;
         cnt=1;
         double cost=0,dist=0;
         for(i=1;i<=m;i++) {
            if(merge(dis[i].a,dis[i].b)) {
               cost+=dis[i].h; dist+=dis[i].s;
               if(++cnt==n) break;
            }
         }
         b=cost/dist;
         if(fabs(b-mid)<1e-5) break;
         mid=b;
      }
     printf("%.3f\n",mid);
   }
}
```

==========

斯坦纳树

```
//--------------------- Minimal Steiner Tree --------------------
//G(V,E),A 是 V 的一个子集，求至少包含 A 中所有点的最小子树
//时间复杂度 O( N*M + N*2^A*(2^A+N) )
//INIT: dis[][]距离矩阵; id[]置为集合 A 中点的标号;
//CALL: steiner(int n,int m);
const int inf=1000000000;
const int N=1010; //总顶点数
const int A=8; //子点数
int id[A]; //包含顶点的序号
int dp[1<<A][N]; //dp[i][j]表示点 j 到联通的集合 i 的最短距离
bool vis[N];
int dis[N][N]; //距离矩阵
void spfa(int s); //求 dis[s][v];
int steiner(int n,int m) {
    int i,j,k,top=1<<m;
    for(i=0;i<n;i++) spfa(i);
    for(i=0;i<top;i++)
        for(j=0;j<n;j++) dp[i][j]=inf;
    for(i=0;i<m;i++)
        for(j=0;j<n;j++) dp[1<<i][j]=dis[id[i]][j];
    for(i=1;i<top;i++) {
```

```
        if((i&(i-1))==0) continue;
        for(k=0;k<n;k++) //init
            for(j=(i-1)&i;j>0;j=(j-1)&i)
                dp[i][k]=min(dp[i][k],dp[j][k]+dp[i^j][k]);
        for(j=0;j<n;j++) //update
            for(k=0;k<n;k++)
            dp[i][j]=min(dp[i][j],dp[i][k]+dis[k][j]);
    }
    int ret=inf;
    for(i=0;i<n;i++) ret=min(ret,dp[top-1][i]);
    return ret;
}
```

==========

/*--*

最小生成树应用

***--*/**

寻找 MST 每条边的最佳替换边[hdu4126]

正反 dfs,O(N^2).求每条 MST 边的最佳替换边.

1.正着,求出 u 节点到 v 节点的树(子树)的非 MST 的最小边. g[u][v]
即(u,v)边将其分成 2 个区域.

2.反着,前面求出 u 到 v 节点的树的最小边...那么从 v 搜 u 的子树.
遍历 u 那边所有的点到 v 节点的树的最小边,即为 (u,v)的最佳替换边.

==========

```
struct Edge {
   int b,c,nxt;
}e[N*2]; //保存 MST 的边
int p[N],cnt;
int n,m,q;
int g[N][N]; //图
int best[N][N]; //最佳替换边
bool mst_e[N][N]; //是否是 MST 的边
//----------------- (N^2)寻找每条边的最佳替换边 -------------------
int dfs1(int u,int fa,int rt) {
//一个点到一颗树的非 MST 的最小边
   for(int i=p[u];i!=-1;i=e[i].nxt) {
      int v=e[i].b;
      if(v!=fa) g[rt][u]=min(g[rt][u],dfs1(v,u,rt));
   }
   return g[rt][u];
}
int dfs2(int u,int fa,int rt) {
//一个树 v 到另一个树每个点 u 的最小边
```

```
   int ret=g[u][rt]; //即 u 到 rt 树的最小边
   for(int i=p[u];i!=-1;i=e[i].nxt) {
      int v=e[i].b;
      if(v!=fa) ret=min(ret,dfs2(v,u,rt));
   }
   return ret;
}
void get_best() {
   for(int i=0;i<n;i++) dfs1(i,i,i);
   for(int u=0;u<n;u++) { //对于每一条 MST 的边,求替换边
      for(int i=p[u];i!=-1;i=e[i].nxt) {
         int v=e[i].b;
         if(best[u][v]<inf) continue;
         int w=dfs2(u,v,v);
         best[u][v]=best[v][u]=w;
      }
   }
   for(int u=0;u<n;u++) { //恢复 MST 边的权值
      for(int i=p[u];i!=-1;i=e[i].nxt) {
         int v=e[i].b;
         g[u][v]=g[v][u]=e[i].c;
      }
   }
}
//------------------------ Prim()记录 MST 的边 ----------------------
int dis[N],pre[N]; bool vis[N];
int prim() {
   int i,j,u,v,mst=0;
   for(i=0;i<n;i++) dis[i]=g[0][i],pre[i]=0;
   memset(vis,0,sizeof(bool)*(n+1));
   vis[0]=1;
   for(i=1;i<n;i++) {
      int mindis=inf; u=-1;
      for(j=0;j<n;j++)
         if(!vis[j]&&dis[j]<mindis) mindis=dis[j],u=j;
      if(u==-1) return -1;
      vis[u]=1;
      addedge(pre[u],u,g[pre[u]][u]); //加 MST 边
      g[pre[u]][u]=g[u][pre[u]]=inf; //先删除 MST 边,dfs 时不选
      mst_e[pre[u]][u]=mst_e[u][pre[u]]=1; //标记为 MST 边
      mst+=mindis;
      for(v=0;v<n;v++)
         if(!vis[v]&&g[u][v]<dis[v]) dis[v]=g[u][v],pre[v]=u;
   }
```

```
   return mst;
}
//-------------------- 加 MST 的边 ------------------------------
void addedge(int a,int b,int c); //双向边 e
//--------------------- 初始化 ------------------------------
void init() {
   for(int i=0;i<=n;i++)
      for(int j=0;j<=n;j++) {
         g[i][j]=best[i][j]=inf;
         mst_e[i][j]=0;
      }
   memset(p,-1,sizeof(p));
   cnt=0;
}
int main() {
   int i,j,a,b,c;
   while(scanf("%d%d",&n,&m),n||m) {
      init();
      for(i=1;i<=m;i++) {
         scanf("%d%d%d",&a,&b,&c);
         g[a][b]=g[b][a]=c;
      }
      int mst=prim();
      get_best(); //求最佳替换边
      scanf("%d",&q);
      double ret=0;
      for(i=1;i<=q;i++) {
         scanf("%d%d%d",&a,&b,&c);
         if(!mst_e[a][b]) { //不是 mst 的边,直接加
            ret+=mst;
         }else { //加上最佳替换边
            ret+=mst+min(c,best[a][b])-g[a][b];
         }
      }
      printf("%.4f\n",ret/q);
   }
}
\*-----------------------------------------------------*/
```

/**\

LCA && RMQ

①LCA:树型图中,两节点的最近公共祖先.

②RMQ:区间最值问题.(题目中要不更新区间的值)
**********/

模板

==========

LCA 朴素法

```
/*
    从根节点开始遍历,求出每个点的深度dis,和父节点fa.
    然后进行朴素攀爬法找LCA(x,y)
PS:求深度也可以用bfs()
*/
```

==========

LCA 倍增法

```
/*
    首先我们构建一张表P[1,N][1,logN] (这里P[i][j]指的是结点i的第2^j个祖先)
    P[i][j] = pre[i] // j==0 时
    P[ P[i][j-1] ][j-1] // 等于P[i][j-1]的第2^(j-1)的父亲
*/
//-------------------- LCA倍增法<NlogN,logN> --------------------
int dis[N],dep[N]; //dis为到根节点的距离,dep为深度
int pa[N][20],pre[N];
void dfs(int u,int f,int deep) { //生成一颗dfs树
    dep[u]=deep; pre[u]=f;
    for(int i=p[u];i!=-1;i=e[i].nxt) {
        int v=e[i].b;
        int w=e[i].c;
        if(v==f) continue;
        dis[v]=dis[u]+w;
        dfs(v,u,deep+1);
    }
}
void make_pa() { //求倍增祖先
    int i,j;
    memset(pa,-1,sizeof(pa));
    for(i=1;i<=n;i++) pa[i][0]=pre[i];
    for(j=1;pw[j]<n;j++)
        for(i=1;i<=n;i++) if(pa[i][j-1]!=-1)
            pa[i][j]=pa[pa[i][j-1]][j-1];
}
int lca(int x,int y) { //查询LCA(x,y)
    int i,log=0;
    if(dep[x]<dep[y]) swap(x,y);
    while(pw[log+1]<=dep[x]) log++;
    for(i=log;i>=0;i--)
```

```
        if(dep[x]-pw[i]>=dep[y]) x=pa[x][i];
    if(x==y) return x;
    for(i=log;i>=0;i--)
        if(pa[x][i]!=-1&&pa[x][i]!=pa[y][i])
            x=pa[x][i],y=pa[y][i];
    return pre[x];
}
//------------------------- LCA上的RMQ -------------------------
//询问路径最大,最小值.
//注意: dis[u]为 v->u 的边权
int mx[N][20];
int tlog[N];
void make_rmq() {
    int i,j;
    memset(mx,-1,sizeof(mx));
    for(i=1;i<=n;i++) mx[i][0]=dis[i];
    for(j=1;pw[j]<n;j++)
        for(i=1;i<=n;i++) if(pa[i][j-1]!=-1)
            mx[i][j]=max(mx[i][j-1],mx[pa[i][j-1]][j-1]);
}
int get_rmq(int x,int f) {
    if(x==f) return -1;
    int len=dep[x]-dep[f];
    int k=tlog[len];
    int idx=x;
    for(int j=tlog[len-pw[k]];j>=0;j--)
        if(pw[j]&(len-pw[k])) idx=pa[idx][j];
    return max(mx[x][k],mx[idx][k]);
}
//------------------------- THE END -------------------------
int main() {
    int i,j,a,b,c;
    while(scanf("%d%d",&n,&m)!=EOF) {
        init();
        for(i=1;i<n;i++) {
            scanf("%d%d%d",&a,&b,&c);
            addedge(a,b,c);
        }
        dfs(1,1,0); //标记
        make_pa(); //建父亲表
        while(m--) {
            scanf("%d%d",&a,&b);
            int f=lca(a,b);
            printf("%d\n",dis[a]+dis[b]-2*dis[f]);
```

```
        }
    }
}
```

==========

一维 RMQ

```
int n,q;
int a[N];
int mx[20][N],mi[20][N];
int tlog[N];
int p[21];
void make_maxrmq() { //最大值
   int i,j,k;
   for(i=1;i<=n;i++) mx[0][i]=a[i];
   for(j=1;p[j]<=n;j++)
      for(i=1;i+p[j]-1<=n;i++)
         mx[j][i]=max(mx[j-1][i],mx[j-1][i+p[j-1]]);
}
void make_minrmq() { //最小值
   int i,j,k;
   for(i=1;i<=n;i++) mi[0][i]=a[i];
   for(j=1;p[j]<=n;j++)
      for(i=1;i+p[j]-1<=n;i++)
         mi[j][i]=min(mi[j-1][i],mi[j-1][i+p[j-1]]);
}
int maxrmq(int a,int b) {
   int k=tlog[b-a+1];
   return max(mx[k][a],mx[k][b-p[k]+1]);
}
int minrmq(int a,int b) {
   int k=tlog[b-a+1];
   return min(mi[k][a],mi[k][b-p[k]+1]);
}
void init() {
   for(int i=0;i<=20;i++) p[i]=(1<<i);
   tlog[0]=-1;
   for(int i=1;i<=50000;i++)
      tlog[i]=(i&(i-1))?tlog[i-1]:tlog[i-1]+1;
}
int main() {
   init(); //预处理
   int s,t;
   while(scanf("%d%d",&n,&q)!=EOF) {
      for(int i=1;i<=n;i++) scanf("%d",&a[i]);
      make_maxrmq();
```

```
      make_minrmq();
      while(q--) {
         scanf("%d%d",&s,&t);
         printf("%d\n",maxrmq(s,t)-minrmq(s,t));
      }
   }
}
```

==========

二维 RMQ

```
int n,m,q;
int a[N][N];
int mx[10][10][N][N];
int tlog[N],p[11];
void make_maxrmq() {
    int i,j,k,l;
    for(i=0;i<=n;i++)
        for(j=0;j<=m;j++) mx[0][0][i][j]=a[i][j];
    for(i=0;p[i]<=n;i++) {
        int di=p[i];
        for(j=0;p[j]<=m;j++) {
            if(i==0&&j==0) continue;
            int dj=p[j];
            for(k=1;k+di-1<=n;k++) {
                for(l=1;l+dj-1<=m;l++) {
                    if(i==0)
mx[i][j][k][l]=max(mx[i][j-1][k][l],mx[i][j-1][k][l+p[j-1]]);
                    else
mx[i][j][k][l]=max(mx[i-1][j][k][l],mx[i-1][j][k+p[i-1]][l]);
                }
            }
        }
    }
}
int maxrmq(int x1,int x2,int y1,int y2) {
    int kn=tlog[x2-x1+1]; //(int)(log(x2-x1+1.0)/log(2.0));
    int km=tlog[y2-y1+1]; //(int)(log(y2-y1+1.0)/log(2.0));
    int v1=mx[kn][km][x1][y1];
    int v2=mx[kn][km][x1][y2-p[km]+1];
    int v3=mx[kn][km][x2-p[kn]+1][y1];
    int v4=mx[kn][km][x2-p[kn]+1][y2-p[km]+1];
    return max(max(v1,v2),max(v3,v4));
}
void init_rmq() {
    for(int i=0;i<=10;i++) p[i]=1<<i;
```

```
    tlog[0]=-1;
    for(int i=1;i<=300;i++)
        tlog[i]=(i&(i-1))?tlog[i-1]:tlog[i-1]+1;
}
```
//优化版
```
int val[N][N];
int mx[10][N][N];
void make_maxrmq() {
    int i,j,k;
    int nn=max(n,m);
    for(k=0;pw[k]<=nn;k++)
        for(i=1;i<=n;i++)
            for(j=1;j<=m;j++) mx[k][i][j]=val[i][j];
    for(k=1;pw[k]<=nn;k++) {
        for(i=1;i<=n;i++) {
            for(j=1;j<=m;j++) {
                mx[k][i][j]=mx[k-1][i][j];
                if(i+pw[k-1]<=n)

mx[k][i][j]=max(mx[k][i][j],mx[k-1][i+pw[k-1]][j]);
                if(j+pw[k-1]<=m)

mx[k][i][j]=max(mx[k][i][j],mx[k-1][i][j+pw[k-1]]);
                if(i+pw[k-1]<=n&&j+pw[k-1]<=m)

mx[k][i][j]=max(mx[k][i][j],mx[k-1][i+pw[k-1]][j+pw[k-1]]);
            }
        }
    }
}
int maxrmq(int x1,int y1,int x2,int y2) {
    int i=tlog[x2-x1+1]; //(int)(log(x2-x1+1.0)/log(2.0));
    int j=tlog[y2-y1+1]; //(int)(log(y2-y1+1.0)/log(2.0));
    int k=min(i,j);
    if(i>j) i=j;
    int ret=mx[0][x1][y1];
    for(i=x1;i<=x2;i+=pw[k]) {
        if(i+pw[k]>x2) i=x2-pw[k]+1;
        for(j=y1;j<=y2;j+=pw[k]) {
            if(j+pw[k]>y2) j=y2-pw[k]+1;
            ret=max(ret,mx[k][i][j]);
        }
    }
    return ret;
```

```
}
```

==========

```
/*-----------------------------------------------------*\
```

LCA && RMQ 应用

```
\*-----------------------------------------------------*/
```

一维 RMQ 查找最大值下标

```
void make_maxrmq() {
   int i,j,k;
   for(i=1;i<=n;i++) mx[0][i]=i;
   for(j=1;p[j]<=n;j++)
      for(i=1;i+p[j]-1<=n;i++) {
         if(val[ mx[j-1][i] ]>val[ mx[j-1][i+p[j-1]] ])
            mx[j][i]=mx[j-1][i];
         else
            mx[j][i]=mx[j-1][i+p[j-1]];
      }
}
int maxrmq(int a,int b) {
   int k=tlog[b-a+1];
   int ret;
   if(val[ mx[k][a] ]>val[ mx[k][b-p[k]+1] ]) ret=mx[k][a];
   else ret=mx[k][b-p[k]+1];
   return ret;
}
```

```
\*-----------------------------------------------------*/
```

树上删一条边,M 条边中也删一条边[POJ3417]

[题意]给出一个 N 个点的树,再给出 M 条边.问从树上删一条边,再从 M 条边中删一条边,
使得图不连通,这样的删法共有多少种.(N,M <= 10^5)

[解析]

对于新边(a,b),会在 a->LCA(a,b)->b 这里形成一个环,
所以删除新边(a,b)以及这个环上的没有被其他环覆盖的边即可分成两部分.
所以问题转化为求每条边被环覆盖的次数
设 col[x]表示 x 所在的父边被覆盖的次数
引进一条新边(a,b)后,col[a]++,col[b]++,col[lca(a,b)]-=2
而这个环上的其他边的统计可以用 treeDP 解决,即 for(v) dp[u]+=dp[v]
注意到 LCA(a,b)的父边是不在环上的,所以每次引进新边(a,b),dp[LCA[a,b]]-=2
if(dp[i]==1) ans++; //删除该边及覆盖它的那个环
if(dp[i]==0) ans+=M //表明这条树边是桥,删除它及任意一条新边都可以.

```
int n,m,q;
int col[N];
//------------------------- LCA 倍增法<NlogN,logN> -------------------------
int dep[N]; //dis 为到根节点的距离,dep 为深度
```

```
int pa[N][20],pre[N];
void dfs(int u,int f,int deep) { //生成一颗 dfs 树
	dep[u]=deep; pre[u]=f;
	for(int i=p[u];i!=-1;i=e[i].nxt) {
		if(e[i].b==f) continue;
		dfs(e[i].b,u,deep+1);
	}
}
void make_pa(); //求倍增祖先
int lca(int x,int y); //查询 LCA(x,y)
void dfs(int u,int f) {
	for(int i=p[u];i!=-1;i=e[i].nxt) {
		int v=e[i].b;
		if(v==f) continue;
		dfs(v,u);
		col[u]+=col[v];
	}
}
//------------------------ THE END ------------------------
void init() { memset(col,0,sizeof(col)); }
int main() {
	int i,j,a,b,c;
	while(scanf("%d%d",&n,&m)!=EOF) {
		init();
		for(i=1;i<n;i++) { scanf("%d%d",&a,&b); addedge(a,b); }
		dfs(1,1,0);
		make_pa();
		for(i=1;i<=m;i++) {
			scanf("%d%d",&a,&b);
			int f=lca(a,b);
			col[a]++; col[b]++; col[f]-=2;
		}
		dfs(1,1);
		int ret=0;
		for(i=2;i<=n;i++) {
			if(col[i]==0) ret+=m;
			else if(col[i]==1) ret++;
		}
		printf("%d\n",ret);
	}
}
\*------------------------------------------------------*/
```

/**\

最短路

①Dijkstra:贪心,每次去最短的路去更新.
②Spfa:一般题目都能过.
③Floyd:插点法全源最短路.求最小环
④Bellman:主要用来判断负环.

**/

模板

==========

Dijkstra+优先队列 O(E*logE)

```
struct node {
   int id,c;
   friend bool operator<(const node &a,const node &b) { return a.c>b.c; }
}f;
int dis[N],path[N]; bool vis[N];
int dj(int s,int t,int n) {
    int i,u;
    priority_queue<node>q;
    for(i=0;i<=n;i++) dis[i]=inf;
    memset(vis,0,sizeof(vis));
    memset(path,-1,sizeof(path));
    dis[s]=0; f.c=0; f.id=s; q.push(f);
    while(!q.empty()) {
        f=q.top(); q.pop();
        u=f.id;
        if(vis[u]) continue;
        vis[u]=1;
        //if(vis[t]) return dis[t];
        for(i=p[u];i!=-1;i=e[i].nxt) {
            int v=e[i].b , w=e[i].c;
            if(!vis[v]&&dis[v]>dis[u]+w) {
                dis[v]=dis[u]+w; path[v]=u;
                f.id=v; f.c=dis[v]; q.push(f);
            }
        }
    }
    if(dis[t]==inf) return -1;
    return dis[t];
}
```

==========

SPFA+连接表 O(kE)

```
int dis[N],path[N]; bool vis[N];
```

```
void spfa(int s,int t,int n) {
    int i,u;
    queue<int>q;
    for(i=0;i<=n;i++) dis[i]=inf;
    memset(vis,0,sizeof(vis));
    memset(path,-1,sizeof(path));
    dis[s]=0; q.push(s);
    while(!q.empty()) {
        u=q.front(); q.pop();
        vis[u]=0;
        for(i=p[u];i!=-1;i=e[i].nxt) {
            int v=e[i].b , w=e[i].c;
            if(dis[v]>dis[u]+w) {
                dis[v]=dis[u]+w; path[v]=u;
                if(!vis[v]) q.push(v),vis[v]=1;
            }
        }
    }
    if(dis[t]==inf) return -1;
    return dis[t];
}
```

==========

Floyd O(n^3)

```
//INIT: e[][]边表,dis[][]任意两点距离,path[][]  路径
//          d 最小环,s[]最小环路径
void floyd() {
    int i,j,k,w,v;
    d=inf; //  记录最小环
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++) path[i][j]=j,dis[i][j]=e[i][j];
    for(k=1;k<=n;k++) {
        //  新增部分,求最小环
        for(i=1;i<k;i++) if(e[i][k]!=inf)
            for(j=1;j<i;j++) if(e[k][j]!=inf) {
                if(dis[i][j]==inf) continue;
                if(d>dis[i][j]+e[i][k]+e[k][j]) {
                    d=dis[i][j]+e[i][k]+e[k][j];
                //  记录最小环路径,保存点到 s 中  i->j
                    len=0; s[0]=k; v=i;
                    while(v!=j) s[++len]=v,v=path[v][j];
                    s[++len]=j;
                }
            }
        //  普通部分,更新最短路
```

```
        for(i=1;i<=n;i++) if(dis[i][k]!=inf)
            for(j=1;j<=n;j++) if(dis[k][j]!=inf) {
                int w=dis[i][k]+dis[k][j];
                if(dis[i][j]>w) dis[i][j]=w , path[i][j]=path[i][k];
                else if(dis[i][j]==w)   // 最短路相等,按字典序排
                    if(path[i][j]>path[i][k]) path[i][j]=path[i][k];
            }
    }
}
```

==========

```
/*-----------------------------------------------------*\
```

最短路的应用

```
\*-----------------------------------------------------*/
```

带限制最短路

A.二分枚举答案,如差值最小(二分枚举差值,最短路判断)
B.dis[][]开二维,记录状态.如dis[v][t]表示到达v用掉t的权限的最短路.
C.有很多条件的，求最短路，可以用广搜来做
D.每条边有开放时间(t1,t2)才能走. 枚举出发时间,求最短路.

```
\*-----------------------------------------------------*/
```

对偶图(平面最小割)

A.从源点到汇点连一条边（此条边不加进去，只是为了形成一个面）
B.新形成的面是最短路的起点,无界的面是最短路的汇点,
C.将面看做点,边看做相连2个面的权值,建图.
D.用Dijkstra+优先队列求最短路.

```
\*-----------------------------------------------------*/
```

最小路径树+删边最短路

```
//给出一边权均为1的无向图 求删掉第1,2...M条边时的所有最短路之和
//先用spfa求所有点最短路 记录所有点的最短路径树
//删边时枚举所有点 如果这条边不在最短路径树上 则不影响其sum
//否则 重新计算该点的最短路径之和
```

==========

```
int diss[N];
int qiao[N][3001];
int dis[N],pre[N];
int spfa(int s,int mak=0) {
   int i,u;
   queue<int>q;
   for(i=0;i<=n;i++) dis[i]=inf;
   memset(pre,-1,sizeof(pre));
   q.push(s); dis[s]=0;
   while(!q.empty()) {
      u=q.front(); q.pop();
```

```
      for(i=p[u];i!=-1;i=e[i].nxt) {
         int v=e[i].b;
         if(e[i].id==mak) continue;  // 删掉第 mak 那条边
       // 有多条边能到 v 最短，说明v点没有最短桥
         if(!mak&&dis[v]==dis[u]+1) {
            pre[v]=-1; continue;
         }
         if(dis[v]!=inf) continue;  // 已经算出最短路
         dis[v]=dis[u]+1;
         if(!mak) pre[v]=e[i].id;   // 记录到达v的边
         q.push(v);
      }
   }
   int res=0;
   for(i=1;i<=n;i++) {
      if(dis[i]==inf) return -1;
      res+=dis[i];
      if(!mak&&pre[i]!=-1) qiao[s][pre[i]]=1; //标记到达v的桥边
   }
   return res;
}
void addedge(int a,int b,int c,int id); //双向边
void init() {
   memset(qiao,0,sizeof(qiao));
   memset(p,-1,sizeof(p));
   cnt=0;
}
int main() {
   int i,j,a,b,ans,tmp;
   bool mak;
   while(scanf("%d%d",&n,&m)!=EOF) {
      init();
      for(i=1;i<=m;i++) {
         scanf("%d%d",&a,&b);
         addedge(a,b,1,i);
      }
      mak=0;
      for(i=1;i<=n;i++) {
         diss[i]=spfa(i);
         if(diss[i]==-1) { mak=1; break; }
      }
//--------------------图不连通 --------------------------
      if(mak) {
         for(i=1;i<=m;i++) printf("INF\n");
```

```
        continue;
      }
//-------------------------------------------------------
      for(i=1;i<=m;i++) {
        ans=0; mak=0;
        for(j=1;j<=n;j++) {
          if(qiao[j][i]) {
            tmp=spfa(j,i);
            if(tmp==-1) { mak=1; break; }
          }else tmp=diss[j];
          ans+=tmp;
        }
        if(mak) puts("INF");
        else    printf("%d\n",ans);
      }
   }
}
\*-------------------------------------------------------*/
```

第 K 短路(A*)

```
=========================================================================
//先用dijkstra从t反向寻找最短路。然后使用A*算法，
//把f(i)=g(i) + h(i)。h(i)就是i点到t的最短距离。
//当某点出队次数达到k次的时候，结果为该点的
//当前路程＋该点到t的最短距离。（我没有判断不连通的情况）
//为什么这样做是对的呢？
//我们这样来思考，如果不实用最短路，而只使用A*那么t第x次出队的结果
//即为第x短路的距离。继而可以想到，从第一个出队次数达到x的点，
//沿着最短路走到t，一定是第x短路。
=========================================================================
struct po { int a,b,c,nxt;
}e[222222],re[222222];
int p[N],pr[N],cnt;
int n,m,k;
bool vis[N]; int dis[N];
void spfa(int s); //dis[] , pr[] , re[]
struct node {
   int v,c;
   friend bool operator<(const node a,const node b) {
      return a.c+dis[a.v]>b.c+dis[b.v];
  }
}f;
int Astar(int s,int t) {
   if(s==t) k++;
   if(k==1) return 0;
```

```
   if(dis[s]==inf) return -1;
   int i,u,len,cnt=0;
   priority_queue<node>q;
   f.v=s; f.c=0; q.push(f);
   while(!q.empty()) {
      f=q.top(); q.pop();
      u=f.v; len=f.c;
      if(u==t) { if(++cnt==k) return len; }
      for(i=p[u];i!=-1;i=e[i].nxt) {
         int v=e[i].b , w=e[i].c;
         f.v=v; f.c=len+w; q.push(f);
      }
   }
   return -1;
}
void addedge(int a,int b,int c) {
   e[cnt].b=b; e[cnt].c=c; e[cnt].nxt=p[a]; p[a]=cnt;
   re[cnt].b=a; re[cnt].c=c; re[cnt].nxt=pr[b]; pr[b]=cnt++;
}
void init() {
   memset(pr,-1,sizeof(pr));
   memset(p,-1,sizeof(p));
   cnt=0;
}
int main() {
   int i,a,b,c;
   while(scanf("%d%d",&n,&m)!=EOF) {
      init();
      for(i=1;i<=m;i++) {
         scanf("%d%d%d",&a,&b,&c);
         addedge(a,b,c);
      }
      scanf("%d%d%d",&a,&b,&k);
      spfa(b);
      int ans=Astar(a,b);
      printf("%d\n",ans);
   }
}
\*-------------------------------------------------------*/
```

最短路和次短路条数

```
int dj(int s,int t) {
   int i,j,k,u,min;
   for(i=0;i<=n;i++) {
      dis[i][0]=dis[i][1]=inf;
```

```
      cnt[i][0]=cnt[i][1]=inf;
   }
   memset(vis,0,sizeof(vis));
   dis[s][0]=0; cnt[s][0]=1;
   for(i=1;i<=2*n;i++) {
      min=inf; u=-1;
      for(j=1;j<=n;j++) {
         if(!vis[j][0]&&dis[j][0]<min) {
            min=dis[j][0]; k=0; u=j;
         }else if(!vis[j][1]&&dis[j][1]<min) {
            min=dis[j][1]; k=1; u=j;
         }
      }
      if(u==-1) break;
      vis[u][k]=1;
      for(j=p[u];j!=0;j=e[j].nxt) {
         int v=e[j].b , w=e[j].c;
         if(dis[v][0]>min+w) {
            dis[v][1]=dis[v][0];   dis[v][0]=min+w;
            cnt[v][1]=cnt[v][0];   cnt[v][0]=cnt[u][k];
         }else if(dis[v][0]==min+w) {
            cnt[v][0]+=cnt[u][k];
         }else if(dis[v][1]>min+w) {
            dis[v][1]=min+w;
            cnt[v][1]=cnt[u][k];
         }else if(dis[v][1]==min+c) {
            cnt[v][1]+=cnt[u][k];
         }
      }
   }
   int num=cnt[t][0];
   if(dis[t][0]+1==dis[t][1]) num+=cnt[t][1];
   return num;
}
\*-------------------------------------------------------*/
```

经过 N 条边的最短路

```
=====================================================================
//起点到终点，经过N条边的最短路
//(floyd + 二分矩阵)
=====================================================================
int N,T,S,E,n;
int V[N*10],e[N][N],ans[N][N],tmp[N][N];
void floyd(int a[][N],int b[][N],int c[][N]) {
   for(int k=1;k<=n;k++)
```

```
      for(int i=1;i<=n;i++) if(b[i][k]!=inf)
         for(int j=1;j<=n;j++) if(c[k][j]!=inf)
            if(a[i][j]>b[i][k]+c[k][j])
               a[i][j]=b[i][k]+c[k][j];
}
void copy(int a[][N],int b[][N]) {
   for(int i=1;i<=n;i++)
      for(int j=1;j<=n;j++)
         a[i][j]=b[i][j],b[i][j]=inf;
}
void mul(int k) {
   for(;k;k>>=1) {
      if(k&1) floyd(tmp,ans,e),copy(ans,tmp);
      floyd(tmp,e,e),copy(e,tmp);
   }
}
void init() {
   for(int i=0;i<N;i++) {
      for(int j=0;j<N;j++)
         ans[i][j]=tmp[i][j]=e[i][j]=inf;
      ans[i][i]=0;
   }
   memset(V,0,sizeof(V));
   n=0;
}
int main() {
   int a,b,c;
   while(scanf("%d%d%d%d",&N,&T,&S,&E)!=EOF) {
      init();
      while(T--) {
         scanf("%d%d%d",&c,&a,&b);
         if(!V[a]) V[a]=++n;
         if(!V[b]) V[b]=++n;
         if(c<e[V[a]][V[b]]) e[V[a]][V[b]]=e[V[b]][V[a]]=c;
      }
      mul(N);
      printf("%d\n",ans[ V[S] ][ V[E] ] );
   }
}
\*-------------------------------------------------------*/
```

最优比率环

```
=============================================================
//求一个环的｛点权和｝除以｛边权和｝最大
//F:点权
```

```
//F/C = ans --> F=ans*c
//二分枚举答案,判断是否存在负环
================================================================
int n,m;
int F[N];
bool vis[N];
int num[N];
double dis[N];
bool spfa(int s,double mid) {
   int u,i;
   queue<int>q;
   for(i=1;i<=n;i++) dis[i]=inf;
   for(i=0;i<cnt;i++) e[i].w=mid*e[i].c-F[e[i].b];
   memset(vis,0,sizeof(vis));
   memset(num,0,sizeof(num));
   dis[s]=0; q.push(s);
   while(!q.empty()) {
       //判负环 ++num[v]>=n  return 1;
   }
   return 0;
}
int main() {
   int i,a,b,c;
   while(scanf("%d%d",&n,&m)!=EOF) {
      init();
      for(i=1;i<=n;i++) scanf("%d",F+i);
      for(i=1;i<=m;i++) {
         scanf("%d%d%d",&a,&b,&c);
         addedge(加有向边);
      }
      double bot=0,top=1000.0,mid;
      do {
         mid=(top+bot)/2.0;
         if(spfa(1,mid)) bot=mid;
         else           top=mid;
      }while(top-bot>1e-5);
      printf("%.2f\n",mid);
   }
}
\*----------------------------------------------------*/
```

最短路+记忆化搜索

```
================================================================
1142 求从1到2有多少条,每步A到B,满足A到2的最短路比B长
2833 求两个起终点,都走最短路,最多有多少公共点.
```

uestc1053 最短路条数.
1688 最短路,和比最短路大1的条数.
==========
【hdu1142】
```
int dp[N];
int dfs(int u) {
    if(u==2)  return dp[u]=1;
    if(dp[u]!=-1) return dp[u];
    dp[u]=0;
    for(int i=p[u];i!=-1;i=e[i].nxt) {
        int v=e[i].b;
        if(dis[u]>dis[v]) dp[u]+=dfs(v);
    }
    return dp[u];
}
```
==========
【hdu2883】
```
int dp[N][N];
int dfs(int a,int b) {
    if(dp[a][b]!=-1) return dp[a][b];
    dp[a][b]=(a==b);
    int tmp=0;
    for(int i=p[a];i!=-1;i=e[i].nxt) { //a继续走
        int v=e[i].b;
        int w=e[i].c;
        if(dis1[v]+w==dis1[a]) tmp=max(tmp,dfs(v,b));
    }
    for(int i=p[b];i!=-1;i=e[i].nxt) { //b继续走
        int v=e[i].b;
        int w=e[i].c;
        if(dis2[v]+w==dis2[b]) tmp=max(tmp,dfs(a,v));
    }
    dp[a][b]+=tmp; //子状态中最多的公共点+本身
    return dp[a][b];
}
```
==========
【uestc1053】
```
ll dp[N];
ll dfs(int u) {
    if(u==T) return dp[u]=1;
    if(dp[u]!=-1) return dp[u];
    dp[u]=0;
    for(int i=p[u];i!=-1;i=e[i].nxt) {
        int v=e[i].b;
```

```
        int w=e[i].c;
        if(dis[v]+w==dis[u]) dp[u]=(dp[u]+dfs(v))%mod;
    }
    return dp[u];
}
```

==========

【hdu1688】

```
int dfs(int u,int k) {
    if(u==T) return num[u][k]=1;
    if(num[u][k]!=-1) return num[u][k];
    num[u][k]=0;
    for(int i=p[u];i!=-1;i=e[i].nxt) {
        int v=e[i].b;
        int w=e[i].c;
        if(dis[v]==dis[u]+w) { num[u][k]+=dfs(v,k); }
        if((!k)&&dis[v]+1==dis[u]+w) {
            num[u][k]+=dfs(v,1);
        }
    }
    return num[u][k];
}
```

***----------*/**

⒀有保护节点的最短路[hdu3873]

==========

```
//最短路的巧妙运用,本题较最短路问题多出了保护城市的条件，
//即一个城市x受若干城市保护，没有攻占所有保护x的“卫星城”就不可以攻占x
//dij过程中要维护其保护城市,用mx[x]记录x所有“卫星城”被攻占的最晚时间，
//那么能到x的最短时间为les[x]和到达x的最短路中的较大者.
```

【处理特点】

```
//①dij入队过程中只把in[v]==0（没有被包含的城市）入队.
//②对于出队的u,它的最短时间已经确定,表示已经被占领.
//它所保护的城市的保护度in[v]--;
//一旦某个被保护的城市的in[v]==0且已经到底（未占领，d[v]!=inf）
//就可以确定到达它的最短时间为dis[v]=max(mx[v],dis[v]),
//它也就到了入队的时机。
//[PS] dij贪心求最短路+拓扑关系...只对in[v]==0的才可以入队.
```

==========

```
int in[N];
vector<int>vec[N];
struct node {
   int x,c;
   friend bool operator<(node &a,node &b){return a.c>b.c;}
}f;
ll dis[N],mx[N]; //记录最短路,被保护的节点的'卫星城'中最远距离
```

```
bool vis[N];
ll dj(int s,int t) {
   int i,j,u;
   priority_queue<node>q;
   for(i=1;i<=n;i++) dis[i]=inf,mx[i]=0;
   memset(vis,0,sizeof(vis));
   dis[s]=0;
   f.x=s; f.c=0; q.push(f);
   while(!q.empty()) {
      f=q.top(); q.pop();
      u=f.x;
      if(vis[u]) continue;
      vis[u]=1;
      for(i=0;i<vec[u].size();i++) { //更新已经占领u节点
         int v=vec[u][i];
         in[v]--; mx[v]=max(mx[v],dis[u]);
         if(in[v]==0&&dis[v]!=inf) { //注意这里
            dis[v]=max(dis[v],mx[v]);
            f.x=v; f.c=dis[v]; q.push(f);
         }
      }
      for(i=p[u];i!=-1;i=e[i].nxt) {
         int v=e[i].b , w=e[i].c;
         if(!vis[v]&&dis[v]>dis[u]+w) {
            dis[v]=max(dis[u]+w,mx[v]);
            if(in[v]==0) f.x=v,f.c=dis[v],q.push(f);
         }
      }
   }
   return dis[t];
}
void init() {
   for(int i=0;i<=n;i++) vec[i].clear();
   memset(in,0,sizeof(in));
   memset(p,-1,sizeof(p));
   cnt=0;
}
int main() {
   int i,j,t,a,b,c;
   while(scanf("%d%d",&n,&m)!=EOF) {
      init();
      while(m--) {
         scanf("%d%d%d",&a,&b,&c);
         addedge(加有向边);
```

```
      }
      for(i=1;i<=n;i++) {
         scanf("%d",&c);
         while(c--) {
            scanf("%d",&b);
            vec[b].push_back(i);
            in[i]++;
         }
      }
      ll ret=dj(1,n);
      printf("%I64d\n",ret);
   }
}
\*-------------------------------------------------------*/
```

经过某些点,回到原点的最短路(状压 DP)

```
//原点为0,经过n个点回到原点的最短路
//首先缩图.再用状压DP求解.
//dp[i][j]:表示状态i,到达j点的最短路时间
//g[u][v] :表示缩图后的u->v的最短距离
//ret     :保存经过1~n的点后回到原点0的最短路
int state=(1<<n);
for(i=0;i<state;i++) {
   for(int u=1;u<=n;u++) {
      if(i==0) dp[i|pw[u-1]][u]=g[0][u];
      else {
         for(int v=1;v<=n;v++) {
            if(i&pw[v-1]) continue;
            int tmp=dp[i][u]+g[u][v];
            dp[i|pw[v-1]][v]=min(dp[i|pw[v-1]][v],tmp);
         }
      }
   }
}
int ret=oo;
for(i=1;i<=n;i++) ret=min(ret,dp[pw[n]-1][i]+g[i][0]);
\*-------------------------------------------------------*/
```

Bellman 寻找负权回路的点

```
struct node {
   int u,v,w,int nxt;
}e[99999];
int p[N],cnt;
bool hash[N];//hash表示是否在负权回路
void dfs(int u) {
   if(hash[u]) return;
```

```
   hash[u]=1;
   for(int i=p[u];i!=-1;i=e[i].nxt) {
      int v=e[i].v;
      if(!hash[v]) dfs(v);
   }
}
int dis[N];
void bellman() {
   int i,j;
   for(i=0;i<=n;i++) dis[i]=inf;
   dis[1]=0;
   for(i=0;i<=n;i++) {
      for(j=1;j<=m;j++) {
         int u=e[j].u,v=e[j].v,w=e[j].w;
         if(dis[u]!=inf&&dis[v]>dis[u]+w) {
            dis[v]=dis[u]+w;
            if(i==n) dfs(v); //和负权回路连接的点
         }
      }
   }
   for(i=1;i<=n;i++) //hash掉不可到达和最短路小于3的
      if(dis[i]==inf||dis[i]<3) hash[i]=1;
}
\*-----------------------------------------------------*/
```

双调旅行商问题

【问题】

平面上,有n个点.

严格的从最左的点走到最右的点 ， 再从最右的点走到最左的点.

从左往右,只能走x越来越大的点 ， 从右往左,只能走x越来越小的点.

问你每个点走,且只走一次.最短路径是多少?

【解析】

假设这两个人为A和B,且A总是走在B后面.

设Pij表示A走到pi,B走到pj时两人所经过的最短双调路径,根据假设,可得i<=j.

b[i,j]表示最短双调路径Pij的长度,

d[i,j]表示点pi到点pj的直线距离.

则: b[1, 2]=d[1, 2]

当i=j时　dp[i][j]=dp[i-1][j]+dis[i-1][j]

当i=j-1时　dp[i][j]=min(dp[k][j-1]+dis[k][j]) { 1<=k<j-1 }

当i<j-1时　dp[i][j]=dp[i][j-1]+dis[j-1][j]

PS:因为dp[i][i]转过去的状态肯定不是最短距离,所以整个过程2个人不会走相同的点.

```
struct node {int x,y;}p[N];
double dp[N][N],dis[N][N];
double TSP() {
   int i,j,k;
```

```
   dp[1][2]=dis[1][2];
   for(j=3;j<=n;j++) {
      dp[j-1][j]=inf;
      for(k=1;k<j-1;k++) {
         double tmp=dp[k][j-1]+dis[k][j];
         dp[j-1][j]=min(dp[j-1][j],tmp);
      }
      for(i=1;i<j-1;i++) dp[i][j]=dp[i][j-1]+dis[j-1][j];
      dp[j][j]=dp[j-1][j]+dis[j-1][j];
   }
   return dp[n][n];
}
int main() {
   int i,j,a,b;
   while(scanf("%d",&n)!=EOF) {
      for(i=1;i<=n;i++) scanf("%d%d",&p[i].x,&p[i].y);
      for(i=1;i<=n;i++)
         for(j=i+1;j<=n;j++)
            dis[i][j]=dis[j][i]=dist(p[i],p[j]);
      double ret=TSP();
      printf("%.2f\n",ret);
   }
}
\*-------------------------------------------------------*/
```

最小环条数

```
int n,m,ret,num;
int e[N][N];
int dis[N][N],way[N][N];
void floyd() {
    int i,j,k;
    ret=inf; num=0;
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++) dis[i][j]=e[i][j],way[i][j]=1;
    for(k=1;k<=n;k++) {
//新增部分:
        for(i=1;i<k;i++) if(e[i][k]!=inf)
            for(j=1;j<i;j++) if(e[k][j]!=inf)
                if(dis[i][j]!=inf) {
                    if(ret>dis[i][j]+e[i][k]+e[k][j]) {
                        ret=dis[i][j]+e[i][k]+e[k][j];
                        num=way[i][j];
                    }else if(ret==dis[i][j]+e[i][k]+e[k][j]) {
                        num+=way[i][j];
                    }
```

```
            }
//通常的 floyd 部分:
        for(i=1;i<=n;i++) if(dis[i][k]!=inf)
            for(j=1;j<=n;j++) if(dis[k][j]!=inf) {
                if(dis[i][j]>dis[i][k]+dis[k][j]) {
                    dis[i][j]=dis[i][k]+dis[k][j];
                    way[i][j]=way[i][k]*way[k][j];
                }else if(dis[i][j]==dis[i][k]+dis[k][j]) {
                    way[i][j]+=way[i][k]*way[k][j];
                }
            }
    }
}
\*-------------------------------------------------------*/
```

差分约束系统

求最大值：<=

求最小值：>=

| <1> 如果有负环,即表示不存在可行解,输出-1

| <2> 如果图不连通(即d[n]=INF),则表示这两个不连通的顶点可以任意排列,输出-2

| 3> 不满足上述两种情况时,输出d[n]

【建图】

1. 区间[a,b]最少有c个1，且每个点至少为0，至多为1。
2. 超级原点n+1，每个区间和都大于0。s[i]-s[n+1]>=0

```
s[b]-s[a-1]>=c      //==> add(a-1,b,c);
0<=s[i+1]-s[i]<=1   //==> add(i,i+1,0); add(i+1,i,-1);
s[i]-s[n+1]>=0      //==> add(n+1,i,0);
```

| hdu1275

```
ti[i] 表示从 i 开始工作的人数
s[i]  表示 0-i 总共雇佣了多少人
r[i]  表示 i 所需的至少人数
ti[i-7]+ti[i-6]+...ti[i] >= r[i]
设：s[i]-s[i-8] = ti[i-7]+ti[i-6]+...ti[i]
0 <= S[i]-S[i-1] <= ti[i]
①S[i]-S[i-1] >= 0
②S[i-1]-S[i] >= -ti[i]
③S[i]-S[i-8] >= r[i] ( i>=8 )
④S[23]+S[a]-S[a+16] >= r[i] ( i<=7 ) ==>
④S[a]-S[a+16] >= r[i]-S[23]
⑤*注意：S[24]-S[0]>=mid  //这句话要记得
```

| hdu1364

```
gt:s[a+b]-s[a-1]>c  ==>  s[a+b]-s[a-1]>=c+1
lt:s[a+b]-s[a-1]<c  ==> s[a-1]-s[a+b]>(-c)
```

超级源点： s[i]-s[n+1] >= 0

| hdu3666

注意（仅仅适用该题）

判断有无解（负环）的时候，如果用spfa，

不能用入队次数大于N来判断，会超时。

有如下两种比较可靠的方法（一般情况下）

1：某个点入队次数大于sqrt(N)的时候

2：所有入队次数大于T * （N + M)，其中T一般取2

L<= Xij * ai / bj <= R

log(L) <= log(Xij) + log(ai) - log(bj) <= log(R)

log(ai) - log(bj) <= log(R) - log(xij)

log(bj) - log(ai) <= log(xij) - log(L)

| hdu3440

【终点在起点左边】 坐标越小越好 ， 用 >= 求最小值

即：dis[v]<dis[u]+w 求最长路

【终点在起点右边】 坐标越大越好 ， 用 <= 求最大值

即：dis[v]>dis[u]+w 求最短路

----------/

/**********\

二分匹配

①男女匹配:X部和Y部最多能匹配多少.

②最少点覆盖:用最少的点覆盖所有的边.=最大匹配

③最少路径覆盖:用最少的不相交的路径覆盖所有的点.=n-最大匹配

④最大独立集:去掉最少的点,使得剩下的点没有边关联.=最少点覆盖

⑤多重匹配:Y部每个元素能容纳X部的cap[y]个.问X部最多多少个和Y部匹配.

即:多(X)对一(Y)的关系.

**********/

模板

==========

二分匹配(连接表)[一对一]

```
//n为X部节点个数,m为Y部节点个数
bool vis[N]; int mat[N];
bool dfs(int u) {
   for(int i=p[u];i!=-1;i=e[i].nxt) {
      int v=e[i].b;
      if(!vis[v]) {
         vis[v]=1;
         if(mat[v]==-1||dfs(mat[v])) {
            mat[v]=u; return 1;
         }
```

```
      }
   }
   return 0;
}
int match() {
   int i,ret=0;
   memset(mat,-1,sizeof(int)*(m+1));
   for(i=0;i<n;i++) {
      memset(vis,0,sizeof(bool)*(m+1));
      if(dfs(i)) ret++;
   }
   return ret;
}
```

==========

多重匹配(连接矩阵)[多对一]

```
//N=100050,M=10
//Y部能容纳X部的最多个数
int n,m; //X部点数n，Y部点数m
int cap[M]; //Y部每个能容纳的X部节点个数
int e[N][M]; //连接矩阵
bool vis[M];
int num[M]; //Y部已经接纳的X部的个数
int mat[M][N]; //多重匹配
bool dfs(int u) {
   for(int v=0;v<m;v++) {
      if(!vis[v]&&e[u][v]) {
         vis[v]=1;
         if(num[v]<cap[v]) { //容量没有满
            mat[v][num[v]++]=u; return 1;
         }
         for(int k=0;k<num[v];k++)
            if(dfs(mat[v][k])) { //是否可增广
               mat[v][k]=u; return 1;
            }
      }
   }
   return 0;
}
int match() {
   int i,ret=0;
   memset(num,0,sizeof(int)*(m+1));
   for(i=0;i<n;i++) {
      memset(vis,0,sizeof(bool)*(m+1));
      if(dfs(i)) ret++;
```

```
    }
    return ret;
}
```

==========

HK 匹配算法

```
int n,m;
bool vis[N];
int disx[N],disy[N];
int matx[N],maty[N];
bool bfs() {
    int i,u;
    queue<int>q;
    memset(disx,-1,sizeof(int)*(n+1));
    memset(disy,-1,sizeof(int)*(m+1));
    for(i=1;i<=n;i++)
        if(matx[i]==-1) q.push(i),disx[i]=0;
    bool flg=0;
    while(!q.empty()) {
        u=q.front(); q.pop();
        for(i=p[u];i!=-1;i=e[i].nxt) {
            int v=e[i].b;
            if(disy[v]==-1) {
                disy[v]=disx[u]+1;
                if(maty[v]==-1) flg=1;
                else {
                    disx[maty[v]]=disy[v]+1;
                    q.push(maty[v]);
                }
            }
        }
    }
    return flg;
}
bool dfs(int u) {
    for(int i=p[u];i!=-1;i=e[i].nxt) {
        int v=e[i].b;
        if(disy[v]==disx[u]+1) {
            disy[v]=0; //相当于 vis[v]=1;
            if(maty[v]==-1||dfs(maty[v])) {
                maty[v]=u; matx[u]=v; return 1;
            }
        }
    }
    return 0;
```

```
}
int match() {
    int i,ret=0;
    memset(matx,-1,sizeof(int)*(n+1));
    memset(maty,-1,sizeof(int)*(m+1));
    while(bfs()) {
        for(i=1;i<=n;i++)
            if(matx[i]==-1&&dfs(i)) ret++;
    }
    return ret;
}
\*-------------------------------------------------------*/

/*-------------------------------------------------------*\
```

二分匹配的应用

```
\*-------------------------------------------------------*/
```

奇偶分图匹配

A.行列匹配，一进一出匹配

```
\*-------------------------------------------------------*/
```

最小点覆盖(最大独立集)

二分图中，选取最少的点数，使这些点和所有的边都有关联
（把所有的边覆盖），叫做最小点覆盖。 最少点数=最大匹配数
举例：
图中有AB两组不相交的点。
A中的每个点需要B中的某些点同时来看守。
B中的每个点需要A中的某些点同时来看守。
选择最少的点，使所有的点都被看守。
A需要B中的点看守，则AB间相应的点连接。每条边就表示一个看守条件。
就是选择一些点，满足所有的边：和所有的边都有关联。

```
\*-------------------------------------------------------*/
```

最小路径覆盖

用尽量少的不相交简单路径覆盖有向无环图G中的所有结点，
解决此类问题可以建立一个二分图模型，把所有顶点i拆成两个，
X部结点集中的i和Y部结点集中的i’，
如果有点i->j，则在二分图中引入边i->j',设二分
图模型的最小匹配为m，则结果就是 ans = n-m

```
\*-------------------------------------------------------*/
```

二分+多重匹配[hdu1669]

[题目]有N个人,Group的个数为M.每个人可能属于一些group.
现在要你把N个人分配到M个group去.使得人数最多的group的人数尽少.
[解析]X部为N个人,Y部未M个group.每个M可以容纳X部的人.这样一想就是一个
多重匹配啦.只要二分枚举M部一个最大的group最多能容纳的人数mid.
然后用多重匹配去判断是否完全匹配即可.

--/

输出匹配情况(字典序最大最小)

A.任意解:输出 mat[v] 和 v 即可.

B.选的X部的字典序最小:for从1-n,小的优先匹配.就可以了.

C.选的X部的字典序最大:for从n-1,大的优先匹配.就可以了.

D.思考:如果保证能完美匹配的情况,使得X部从1-n对应的匹配Y部的节点编号
使得X按1-n顺序,输出其对应匹配Y部的节点字典序最小或最大?

--/

二分匹配必须边

所谓二分匹配的必须边,就是去掉该(u,v)的边,最大匹配树就会减少.

[解析]:先求原图的最大匹配ans.然后对于每条在匹配中的边,删掉,再求最大匹配.
如果删掉后,最大匹配数<ans.那么该边为一条必须边.

--/

二分匹配关键点[hdu1281]

所谓二分匹配中的关键点,就是去掉该点后,最大匹配数就会减少.

[解析]:先求原图的最大匹配ans.然后对于每个点,如果去掉后,最大匹配数<ans.
那么该点即为关键点.所有的点都如此操作,就可以求出所有的关键点.

--/

二分匹配好题[BNU14507]

[题意]给定N(<=100)个数A[i],要求从中选出尽可能多的数.
A[p1],A[p2],…,A[pm](0<=pi<N,若i!=j则pi!=pj).
满足对于任意的(pi!=pj)有A[pi]%A[pj]!=0.
求解,输出字典序最小的序列.
Sample Input
10
10 9 8 7 6 5 4 3 2 1
Sample Output
4 5 6 7 9

[解析]把每个数看做一个点,若A[j]%A[i]==0,则add(i,j).
则倘若选择了某个点就不能再选能到达该点或该点能到的所有点了.
即若选了某个点则把经过这个点的所有路径全部覆盖了.
所以这道题让求的就是尽可能多的点使得这些点两两之间不在同一条路径上.
进而想到最小路径覆盖.不过最小路径覆盖要求每个点只能用一次似乎和本题不符,
不过幸运的是这种倍数关系是满足传递性的so这点并会不影响到算法的正确性.

通过最小路径覆盖我们已经可以求得最多可以选多少个点了,
然后要解决的就是怎么能找到字典序最小的方案.

其实还是比较好想的,假设最后选的点数是m,
查看二分匹配出来的图(最小路径覆盖是用二分匹配解决的)
可以找到m条路径,那么最终答案一定是从这m条路径中每个路径上选一个点得到.
于是起始时每个路径都选最小的那个点,一但发现哪个被选的点能被其他选中的点
整除则把这个点改成这条路径上的后一个点,
直到找不到整除点对位置.最后把选中的点sort一遍就好了.

----------/

关键点&&必须边

关键点:

对于 matx[i]或者 maty[i] == -1 的,必定不是关键点.

对于 matx[i]!=-1,删除 matx[i]对应点,然后对 X 部的 i 增广 dfsx();

如果可以增广,那么 matx[i]对应的点不是关键点.

对于 maty[i]!=-1,删除 maty[i]对应点,然后对 Y 部的 i 增广 dfsy();

如果可以增广,那么 maty[i]对应的点不是关键点.

关键边:

对于每条匹配的边 x 匹配 y,删去 e[x][y]=0;

并且还要标记 xy 成未匹配: matx[x]=-1;maty[y]=-1;

然后对边的两边,分别对 xy 进行增广.

如果都无法增广,那么 e[x][y]为:必须边(关键边)

```
//hdu3517 关键点(最大独立集的关键点)
int na,nb,nc,n,m;
int sub[N];
int idx[N],idy[N];
bool g[N][N],e[N][N];
bool hash[N],vis[N];
int matx[N],maty[N];
bool dfsx(int u) {
    for(int v=1;v<=m;v++) {
        if(e[u][v]&&!vis[v]) {
            vis[v]=1;
            if(maty[v]==-1||dfsx(maty[v])) return 1;
        }
    }
    return 0;
}
bool dfsy(int u) {
    for(int v=1;v<=n;v++) {
        if(e[v][u]&&!vis[v]) {
            vis[v]=1;
            if(matx[v]==-1||dfsy(matx[v])) return 1;
        }
    }
    return 0;
}
bool dfs(int u) {
    for(int v=1;v<=m;v++) {
        if(e[u][v]&&!vis[v]) {
            vis[v]=1;
            if(maty[v]==-1||dfs(maty[v])) {
                matx[u]=v; maty[v]=u; return 1;
```

```
                }
            }
        }
        return 0;
}
int match() {
        int i,ret=0;
        memset(matx,-1,sizeof(matx));
        memset(maty,-1,sizeof(maty));
        for(i=1;i<=n;i++) {
            memset(vis,0,sizeof(vis));
            if(dfs(i)) ret++;
        }
        return ret;
}
void init() {
        memset(hash,0,sizeof(hash));
        memset(g,0,sizeof(g));
        memset(e,0,sizeof(e));
}
void build_map() {
        for(int i=1;i<=n;i++) {
            for(int j=1;j<=m;j++) {
                int a=sub[idx[i]];
                int b=sub[idy[j]];
                if(g[idx[i]][b]||g[idy[j]][a]) e[i][j]=1; //建矛盾
            }
        }
}
int main() {
        int i,j,k,a,b,c;
        while(scanf("%d%d%d",&na,&nb,&nc)!=EOF) {
            init(); n=m=0;
            for(i=1;i<=nc;i++) {
                scanf("%d%d",&sub[i],&k);
                if(sub[i]<=na) idx[++n]=i;
                else idy[++m]=i;
                while(k--) {
                    scanf("%d",&b);
                    g[i][b]=1;
                }
            }
            build_map();
            int ret=match();
```

```
            printf("%d\n",nc-ret);
            for(i=1;i<=n;i++) { //dfsx()
                if(matx[i]!=-1) {
                    memset(vis,0,sizeof(vis));
                    vis[matx[i]]=1;
                    if(dfsx(i)) hash[sub[idy[matx[i]]]]=1;
                }else hash[sub[idx[i]]]=1;
            }
            for(i=1;i<=m;i++) { //dfsy()
                if(maty[i]!=-1) {
                    memset(vis,0,sizeof(vis));
                    vis[maty[i]]=1;
                    if(dfsy(i)) hash[sub[idx[maty[i]]]]=1;
                }else hash[sub[idy[i]]]=1;
            }
            int ct=0;
            for(i=1;i<=na+nb;i++) if(hash[i]) ct++;
            printf("%d",ct);
            for(i=1;i<=na+nb;i++) if(hash[i]) printf(" %d",i);
            puts("");
        }
}
==========================================================================
//fzu1202  必须边
int n,m;
bool e[N][N];
bool vis[N];
int matx[N],maty[N];
bool dfsx(int u) ; //同上，关键点
bool dfsy(int u) ; //同上，关键点
bool dfs(int u); //同上，关键点
int match(); //同上，关键点
void init() { memset(e,true,sizeof(e)); }
int main() {
    int i,j,k,a,b,c;
    while(scanf("%d",&n)!=EOF) {
        init(); m=n;
        while(scanf("%d%d",&a,&b),a||b) e[a][b]=0;
        int ret=match();
        bool flg=0;
        for(i=1;i<=n;i++) {
            if(matx[i]!=-1) {
                int x=i,y=matx[i];
                bool mak=0;
```

```
                    e[x][y]=0; matx[x]=-1; maty[y]=-1;
                    memset(vis,0,sizeof(vis));
                    if(dfsx(x)) mak=1; //dfsx()
                    memset(vis,0,sizeof(vis));
                    if(dfsy(y)) mak=1; //dfsy()
                    e[x][y]=1; matx[x]=y; maty[y]=x;
                    if(!mak) printf("%d %d\n",i,matx[i]),flg=1;
                }
            }
            if(!flg) puts("none");
        }
    }
\*-------------------------------------------------------*/
```

/**\

一般图的最大匹配(带花树)

介绍:主要运用匹配的交错轨的思想,本质也是找可增广路增广.

值得强调的是:可增广路的边数为奇数,

带花树的花是一个圈,圈内的边数也为奇数,这个性质很好.

**/

模板

==========

可搞定一类棋盘博弈问题[zoj3316]

```
int n,l;
bool g[N][N];
int block[N];
int pre[N],mat[N]; //匹配情况
int que[N],ss,ee; //模拟队列
bool inb[N],vis[N]; //inblock,vis
void modify(int u,int lca) {
    while(block[u]!=lca){
        int v=mat[u];
        vis[block[u]]=vis[block[v]]=1;
        u=pre[v];
        if(block[u]!=lca) pre[u]=v;
    }
}
void contract(int s,int u,int v){
    int lca;
    memset(vis,0,sizeof(vis));
    for(lca=u;;lca=pre[mat[lca]]) {
```

```
        lca=block[lca];
        vis[lca]=1;
        if(lca==s) break;
    }
    for(lca=v;;lca=pre[mat[lca]]){
        lca=block[lca];
        if(vis[lca]) break;
    }
    memset(vis,0,sizeof(vis));
    modify(u,lca); modify(v,lca);
    if(block[u]!=lca) pre[u]=v;
    if(block[v]!=lca) pre[v]=u;
    for(int i=0;i<n;i++){
        if(vis[block[i]]){
            block[i]=lca;
            if(inb[i]) continue;
            inb[i]=1; que[++ee]=i;
        }
    }
}
void find(int s) {
    memset(inb,0,sizeof(inb));
    memset(pre,-1,sizeof(pre));
    for(int i=0;i<n;i++) block[i]=i;
    que[ss=ee=0]=s; inb[s]=1;
    while(ss<=ee) {
        int u=que[ss++];
        for(int i=0;i<n;i++) {
            if(!g[u][i]) continue;
            if(block[u]==block[i]) continue;
            if(mat[u]==i) continue;
            if(i==s||(mat[i]!=-1&&pre[mat[i]]!=-1)){
                contract(s,u,i);
            }else if(pre[i]==-1) {
                pre[i]=u;
                if(mat[i]==-1){
                    while(i!=-1){
                        u=pre[i];
                        int tmp=mat[u];
                        mat[u]=i; mat[i]=u;
                        i=tmp;
                    }
                    return;
                }
```

```
                inb[mat[i]]=1; que[++ee]=mat[i];
            }
        }
    }
}
int gra_match() {
   int i,ret=0;
   memset(mat,-1,sizeof(mat));
   for(i=0;i<n;i++) {
      if(mat[i]==-1) find(i);
      if(mat[i]!=-1) ret++;
   }
    return ret;
}
int main() {
   int i,j,c;
   while(scanf(n)!=EOF) {
      for(i=0;i<n;i++) scanf(pt[i].x,pt[i].y);
      scanf(l);
      memset(g,0,sizeof(g));
      for(i=0;i<n;i++)
         for(j=i+1;j<n;j++)
            if(abs(pt[i].x-pt[j].x)+abs(pt[i].y-pt[j].y)<=l)
               g[i][j]=g[j][i]=1;
      if(gra_match()==n) puts("YES");
      else puts("NO");
   }
}
```

***--*/**

/*--*

一般图最大匹配的应用

***--*/**

棋盘取棋子博弈[zoj3316]

[题意]有N个棋子在棋盘上,2个人轮流拿走一个棋子,第一步可以拿任意一个,
而之后每一步必须拿上一步拿走的棋子曼哈顿长度L以内的棋子.
问:后手是否能赢.

[解析]把每一个棋子与周围距离为L的棋子都连上边后,形成一些联通块.
易知,一轮游戏只能在某一个联通块里开始直到结束.
那么,如果有一个联通块不是完美匹配,先手就可以走那个没被匹配到的点,
后手不论怎么走,必然走到一个被匹配的点上,先手就可以顺着这个交错路走下去.
最后一定是后手没有路可走,因为如果还有路可走,这一条交错路,就是一个增广路.
必然有更大的匹配.故判断是否是完美匹配即可.(gra_match()==n)

--/

棋盘移动 king 博弈[hdu3446]

[题意]有一个R*C的棋盘,棋盘上有一些格子是幸运的格子,棋盘上只有一个棋子:king.
king有一定的走路范围:20个格子如下所示.
dx[]={-2,-2,-2,-2,-1,-1,-1,-1,-1,0,0,1,1,1,1,1,2,2,2,2}
dy[]={-2,-1,1,2,-2,-1,0,1,2,-1,1,-2,-1,0,1,2,-2,-1,1,2}
两个人轮流走,每个人可以移动king走到没被走过的幸运的格子上.
问:先手是否能赢.

[解析]这题与上一题某方面是一样的,都是与增广路有关.
把所有能连的边先连上,我们先不算king,求一次匹配.
然后我们再算上king,求一次匹配.如果第二次匹配的结果比第一次大,
说至少明存在一条增广路.且增广路的起点为king所在的点.
那么,我们沿着这条路走下去,最后后手必然无路可走.
总结:2次匹配.

--/

/**\

KM 匹配

①最大权匹配:边有权值的匹配.使得匹配的权值和最大.

INIT memset(lx,-0x7f,sizeof(lx));
memset(ly,0,sizeof(ly));
w[][] 初始化为 0

②最小权匹配:边有权值的匹配.使得匹配的权值和最小.

INIT memset(lx,-0x7f,sizeof(lx));
memset(ly,0,sizeof(ly));
w[][] 初始化为 负的很大
解法二: 用一个很大的数 -w , w 初始化还是 0
建边 a 和 b 连 -c 的边

**/

模板

==========

最小权匹配

```
//下面的模板求的是最大权.
//因为将边权变为负的,相当于求负的最大.即为最小权了.
const int inf=100000000;
int w[N][N],aug,mat[N],lx[N],ly[N];
bool visx[N],visy[N];
int slack[N];
bool dfs(int u) {
   visx[u]=1;
   for(int v=1;v<=n;v++) {
```

```
      int t=lx[u]+ly[v]-w[u][v]; //注意-w[u][v]溢出!
      if(t) slack[v]=min(slack[v],t);
      else {
         if(!visy[v]) {
            visy[v]=1;
            if(mat[v]==-1||dfs(mat[v])) {
               mat[v]=u; return 1;
            }
         }
      }
   }
   return 0;
}
int km() {
   int i,j,ret=0;
   for(i=1;i<=n;i++)
      for(j=1;j<=m;j++) lx[i]=max(lx[i],w[i][j]);
  for(int x=1;x<=n;x++) {
      for(i=0;i<=m;i++) slack[i]=inf;
      while(1) {
         memset(visx,0,sizeof(visx));
         memset(visy,0,sizeof(visy));
         if(dfs(x)) break;
         aug=inf;
         for(i=1;i<=m;i++)
            if(!visy[i]&&slack[i]<aug) aug=slack[i];
         for(i=1;i<=n;i++)
            if(visx[i]) lx[i]-=aug;
         for(i=1;i<=m;i++)
            if(visy[i]) ly[i]+=aug;
            else       slack[i]-=aug;
      }
   }
   int cnt=0;
   for(i=1;i<=m;i++) { //最优解
      if(mat[i]==-1||w[mat[i]][i]==-inf) continue;
      cnt++; re+=w[mat[i]][i];
   }
   if(cnt!=n) return -1; //X部未完全匹配
   return -ret; //返回最小权
}
void init() {
   memset(mat,-1,sizeof(mat));
   memset(lx,-0x7f,sizeof(lx));
```

```
   memset(ly,0,sizeof(ly));
   memset(w,-0x7f,sizeof(w));
}
int main() {
   int i,j,t,a,b,c;
   scanf("%d",&t);
   while(t--) {
      scanf("%d%d",&n,&m);
      init();
      for(i=1;i<=m;i++) {
         scanf("%d%d%d",&a,&b,&c);
         if(w[a][b]<(-c)) w[a][b]=w[b][a]=-c;
      }
      int ret=km();
      if(ret==-1) puts("NO");
      else       printf("%d\n",ret);
   }
}
```

==========

套模板注意事项

把不存在的边权值赋为0.

要注意的是n<=m(否则错误)

如果n>m则把n和m交换集合交换

***----------*/**

/*----------*

KM 匹配的应用

***----------*/**

边权之积最大

KM算法求得的最大权匹配是边权值和最大，

如果我想要边权之积最大，又怎样转化？

还是不难办到，每条边权取自然对数，然后求最大和权匹配，

求得的结果a再算出e^a就是最大积匹配。

至于精度问题则没有更好的办法了。

***----------*/**

收益最大,且边的改动最少(1)

[题意]:给定任务能在哪些机器上工作和收益,且每个机器只能接一个任务.

再给定每个任务目前在哪个机器上工作.

问:改变任务工作的机器,使收益最大,且改动最小?

[解析]:由于要求满足收益最大情况下改变顺序最少的方案,

因此我们在建图的时候可以将权值乘以一个常数(*1000),

然后对原有边的权值自加一个较小量来体现出对该边的"偏好"

这样边权相同的情况下,就会优先选择已经工作的任务与机器.

```
void init() {
   memset(mat,-1,sizeof(mat));
   memset(w,-0x7f,sizeof(w));
   memset(lx,-0x7f,sizeof(lx));
   memset(ly,0,sizeof(ly));
}
int main() {
   while(scanf("%d%d",&n,&m)!=EOF) {
      init();
      for(int i=1;i<=n;i++)
         for(int j=1;j<=m;j++) {
            scanf("%d",&w[i][j]);
            w[i][j]*=1000; //这里
         }
      int sum=0,a;
      for(i=1;i<=n;i++) {//已在机器工作的任务
         scanf("%d",&a);
         sum+=w[i][a];
         w[i][a]++;  //这里
      }
      int ans=km();
      //最小改动 , 最大增加的效益
      printf("%d %d\n",n-ans%1000,(ans-sum)/1000);
   }
}
\*-------------------------------------------------------*/
```

收益最大,且边的改动最少(2)

[题意]举行一场比赛,每对N个人,编号都1...N.
每个人都有HP值和攻击值,i号赢了得Vi分,输了失去Vi分.
如果自己队的i号和对方队的j号对打(i!=j),则错位值+1.
(即i与i对打是不错位的).
问:在得分最多的情况下,使得错位值最小.
输出:最大得分&&未错位的对战数/总人数.

[解析]由于要求求满足收益最大情况下改变顺序最少的方案,
因此我们在建图的时候可以将权值乘以一个常数(*100)
然后对原有边的权值自加一个较小量来体现出对该边的'偏好'.

```
int n,m,e;
int w[N][N];
int val[N]; //score
int h1[N],h2[N]; //HP
int a1[N],a2[N]; //atk
void init() {
   memset(w,-0x7f,sizeof(w));
   memset(lx,-0x7f,sizeof(lx));
```

```
   memset(ly,0,sizeof(ly));
   memset(mat,-1,sizeof(mat));
}
int main() {
   while(scanf("%d",&n),n) {
      init(); m=n;
      for(i=1;i<=n;i++) scanf("%d",&val[i]);
      for(i=1;i<=n;i++) scanf("%d",&h1[i]);
      for(i=1;i<=n;i++) scanf("%d",&h2[i]);
      for(i=1;i<=n;i++) scanf("%d",&a1[i]);
      for(i=1;i<=n;i++) scanf("%d",&a2[i]);
/********************** 建图 ***************************/
      for(i=1;i<=n;i++)
         for(j=1;j<=n;j++) {
            int ct1= (h2[j]+a1[i]-1)/a1[i];
            int ct2= (h1[i]+a2[j]-1)/a2[j];
            if(ct1<=ct2) w[i][j]=val[i]*100;
            else        w[i][j]=-val[i]*100;
         }
      for(i=1;i<=n;i++) w[i][i]++; //偏好
/********************** **** ***************************/
      int ans=km();
      if(ans<0) puts("Oh, I lose my dear seaco!");
      else     printf("%d %.3lf%%\n",ans/100,(ans%100)/(n*0.01));
   }
}
\*-----------------------------------------------------*/
```

其他类型

[题意]输入M个Q序列(1~n的序列)

要你求一个T序列(1~n的序列)

使得Q都变成T的总价值最小

假设: T 数列为 T1 T2 T2

其中一个: Q 数列为 Q1 Q2 Q3

d = sum(abs(T1-Q1))

我们假设T数列第i个位置数为j,(w[i][j]).

那么把所有Q的第i个位置数变为j,花费就tot=sum(abs(j-Q[k][i]))

{ Q[k][i] 表示第k个Q的第i为为多少 }

[解析]因为每一个位置上的rank不重复,所以就变成了一个最小权二分匹配

题目意思是定义两个排列a,b的距离d=sum(ai-bi),

现在给出m个排列,求一个排列t,使其与m个排列的d值的和最小.

建立二分图:

左边节点表示m个排列第i个位置，右边就是1到n，n个数

i到j连边，边权为-sum(abs(Aij-j))

求最小权匹配,匹配边i-->j代表j这个数是在i这个位置,

这样一个匹配就代表一个n个数的排列，并且sum(d)最小.

--/

[招聘员工]

【题意】有n个部门和n个待业员工，每个部门要雇佣一个员工，各个部门希望雇佣各个员工的希望值不同，各个员工希望被雇佣到各个部门的希望值也不同。

请给出一种匹配方案，使得所有人的希望值之和最大，如果最大的情况有多种，请输出所有的情况。

【思路】第i个员工和第j的部门之间的边权为两者各自的希望值之和，

先用KM算法求出最大的希望值之和。

然后再用深搜构造所有能达到这个值的匹配情况。

X对Y有个评价 ， Y对X有个评价. 求X和Y配对,使得评价和最小

X1对Y1有评价 w[X1][Y1]+=c; Y2对X2有评价 w[X2][Y2]+=c;

即Xi和Yi配对时,的最小评价. 使得X和Y全部配对的最小评价

--/

[货物运输]

n m k //n个商店,m个仓库,k类物品.

n行,每行k个数字,表示第i个物品需要ci个.

m行,每行k个数字,表示第i个仓库总共有ci个.

k*n*m的矩阵,表示第k个物品从i运到j的费用.

对于每个物品k,求一次最小花费.

然后全部加起来即为结果.

--/

[最短匹配]

【题意】给定2*N个点坐标,让你用前N个点和后N个点一一配对,使得相连边没有相交的.

【分析】首先有一个性质必须知道，那就是最短的配对，必然是没有相交的。

于是我们就可以构建二分图，求最小权匹配。（用最小费用流求解）

--/

[工作分配]

N个任务,交给M个人做

map[i][j] 表示第i个任务交给第j个人做的时间.

每个人必须完成一个任务后,才能继续去做另一个任务.

问N个任务完成时间的平均值最少为多少?

[解析]

设N个任务的执行时间分别为T1,T2…TN,则N个订单的总的执行时间是

T1*N + T2*(N-1) + … + TN-1*2 + TN

从倒数第k个开始考虑.

构图:

将每台机器拆成N个点。第k个点表示倒数第k个完成的任务。

将任务i和机器j的第k个分点连权值为map[i][j]*k的边。

--/

/**\

一般图的最小权匹配

介绍: 解决带权无向图的中国邮路问题

**/

模板

```
==========================================================
一般完全图最小权匹配
新图的点数cnt_node(表示原图奇度点个数),,0,...cnt_node-1.
边权关系保存在G[][](两两奇度点的距离)中,调用match,返回sum
[注意]Minimum_Match_Value()求的是最大权!!!
     所以边权G要为负的,即求最小权了.
//下标从0开始!!!
int n,m; //原图点数,边数
int deg[N]; //原点度数
int e[N][N]; //原图连接矩阵
struct Minimum_Match_Value {
   int cnt_node;
   int g[N][N];
   int dis[N];
   bool vis[N];
   int rec[N],cr,M[N];
   bool spfa(int u) {
      rec[cr++]=u;
      if(vis[u]) return 1;
      vis[u]=1;
      for(int v=0;v<cnt_node;v++) {
         if(v!=u&&M[u]!=v&&!vis[v]) {
            int w=M[v];
            if(dis[w]<dis[u]+g[u][v]-g[v][w]) {
               dis[w]=dis[u]+g[u][v]-g[v][w];
               if(spfa(w)) return 1;
            }
         }
      }
      cr--;
      vis[u]=0;
      return 0;
   }
   int match() {
      int i,j;
      for(i=0;i<cnt_node;i+=2) M[i]=i+1,M[i+1]=i;
      int cnt=0;
```

```
      while(1) {
         cr=0;
         bool found=0;
         memset(dis,0,sizeof(dis));
         memset(vis,0,sizeof(vis));
         for(i=0;i<cnt_node;i++) {
            if(spfa(i)) {
               found=1;
               int nx=M[rec[cr-1]];
               for(j=cr-2;rec[j]!=rec[cr-1];j--) {
                  M[nx]=rec[j];
                  swap(nx,M[rec[j]]);
               }
               M[nx]=rec[j];
               M[rec[j]]=nx;
               break;
            }
         }
         if(!found) {
            cnt++;
            if(cnt>=3) break;
         }
      }
      int sum=0;
      for(i=0;i<cnt_node;i++) {
         int v=M[i];
         if(i<v) sum+=g[i][v];
      }
      return sum;
   }
}G;
void init() {
   memset(deg,0,sizeof(deg));
   for(int i=0;i<n;i++)
      for(int j=0;j<n;j++) e[i][j]=inf;
}
int main() {
   int i,j,a,b,c;
   while(scanf("%d",&n),n) {
      init();
      scanf("%d",&m);
      int sum=0;
      while(m--) {
         scanf("%d%d%d",&a,&b,&c);
```

```
            e[a][b]=e[b][a]=min(e[a][b],c);
            sum+=c; //每条边经过一次的距离和
            deg[a]++; deg[b]++; //计算度数
        }
        int V[N],ct=0; //记录奇度点
        for(i=0;i<n;i++) if(deg[i]&1) V[ct++]=i;
        floyd(); //求任意两点距离
        G.cnt_node=ct; //构建新图,求最小全匹配
        for(i=0;i<ct;i++)
            for(j=0;j<ct;j++)
                G.g[i][j]=-e[V[i]][V[j]]; //负的
        printf("%d\n",sum-G.match());
    }
}
\*-------------------------------------------------------*/

/*-------------------------------------------------------*\
```

一般图的最小权匹配的应用

```
\*-------------------------------------------------------*/
```

无向图中国邮路问题

[题意]邮递员从任意一点出发,经过每条边至少一次,回到原点的最短路程.

[解析]

A.若原图不连通则无解.

B.若所有点的度数均为偶数,则存在欧拉回路,所有边权和即为结果.

C.否则肯定存在偶数个奇度点,将这些奇度点拉出来重新构建新图.
任意两点之间的边权为原图的两点之间的最短距离.
然后最新图求一次一般图的最小权完美匹配,加上原图的所有边权和.
即为最终结果.

```
\*-------------------------------------------------------*/
```

输出最小权值的回路

将每条匹配边在原图中对应的最短路上的每条边都额外增加一条,
这样原图便成为欧拉图,求一次欧拉回路即可.

```
\*-------------------------------------------------------*/


/*********************************************************\
```

最大团&&稳定婚姻

①最大团:图中最大的完全匹配子图. 即两两之间都有边.

②稳定婚姻:每个人都对另一半的所有人都有一个偏好值, 求是的其稳定的匹配.

```
\*********************************************************/
```

模板

==========

最大团

```
//注意:下标从0开始!!!
int e[N][N]; //连接矩阵
int set[N],cs[N]; //临时变量
int ans[N],num; //保存答案
bool mak;
void clique(int n,int depth,int *u) {
   int i,j,v[N],vn;
   if(n) {
      if(depth+cs[u[0]]<=num) return;
      for(i=0;i<n+depth-num;i++) {
         for(j=i+1,vn=0;j<n;j++)
            if(e[u[i]][u[j]]) v[vn++]=u[j];
            set[depth]=u[i];
            clique(vn,depth+1,v);
            if(mak) return;
      }
   }
   else if(depth>num) {
      num=depth; mak=1;
      for(i=0;i<depth;i++) ans[i]=set[i];
   }
}
int reclique(int n) {
   int vn,v[N],i,j;
   num=0;
   for(cs[i=n-1];i>=0;i--) {
      for(vn=0,j=i+1;j<n;j++)
         if(e[i][j]) v[vn++]=j;
      mak=0;
      set[0]=i;
      clique(vn,1,v);
      cs[i]=num;
   }
   return num;
}
```

==========

稳定婚姻匹配(白爷模板)

稳定婚姻问题是一个很有意思的匹配问题，有n位男士和n位女士，
每一个人都对每个异性有一个喜好度的排序，代表对他的喜爱程度，
现在希望给每个男士找一个女士作配偶，使得每人恰好有一个异性配偶。

如果男士 u 和女士 v 不是配偶但喜欢对方的程度都大于喜欢各自当前配偶的程度，则称他们为一个不稳定对。稳定婚姻问题就是希望找出一个不包含不稳定对的方案。

算法非常简单，称为求婚－拒绝算法，每位男士按照自己喜欢程度从高到低依次给每位女士主动求婚直到有一个女士接受他，
对于每个女士，如果当前向她求婚的配偶比她现有的配偶好则抛弃当前配偶，否则不予理睬，
循环往复直到所有人都有配偶。有趣的是，看起来是女士更有选择权，但是实际上最后的结果是男士最优的(man-optimal)。

首先说明最后匹配的稳定性，随着算法的执行，每位女士的配偶越来越好，而每位男士的配偶越来越差。
因此假设男士 u 和女士 v 形成不稳定对，u 一定曾经向 v 求过婚，但被拒绝。这说明 v 当时的配偶比 u 更好，
因此算法结束后的配偶一定仍比 u 好，和不稳定对的定义矛盾，类似的，
方式我们考虑最后一个被抛弃的男士和抛弃这位男士的女士，不难得出这个算法一定终止的结论。

如果存在一个稳定匹配使得男士 i 和女士 j 配对，则称(i,j)是稳定对。对于每个男士 i，
设所有稳定对(i,j)中 i 最喜欢的女士为 best(i)，则可以证明这里给出的算法对让每位男士 i 与 best(i)配对。
对于所有男士来说，不会有比这更好的结果了，而对于女士则恰恰相反，对于她们来说不会有比这更糟的结果了，
因此这个算法是男士最优的。

算法一定得到稳定匹配，并且复杂度显然是 O(n^2)，因为每个男士最多考虑每个女士一次，
考虑的时间复杂度是 O(1)，当然了，需要作一定的预处理得到这个复杂度。

```
//方法类似于二分匹配
//like数组记录i第j喜欢的是哪一个（通常是输入）
//rank数组记录i喜欢j的程度排名，0为最大
//rank[i][like[i][j]]=j
int xlike[N][N],ylike[N][N];
int xrank[N][N],yrank[N][N];
int xmat[N],ymat[N]; //匹配
bool find(int x,int y) {
   if(ymat[y]==-1) {
      xmat[x]=y; ymat[y]=x;
      return 1;
   }
   int tx=ymat[y];
   if(yrank[y][tx]>yrank[y][x]){
      xmat[x]=y; ymat[y]=x;
      int ty=xlike[tx][xrank[tx][y]+1];
      if(!find(tx,ty)) return 0;
      return 1;
```

```
   }else {
      for(int i=xrank[x][y]+1;i<n;i++){
         int ty=xlike[x][i];
         if(!find(x,ty)) continue;
         return 1;
      }
      return 0;
   }
}
bool stable_match(){
   memset(xmat,-1,sizeof(xmat));
   memset(ymat,-1,sizeof(ymat));
   for(int i=0;i<n;i++) {
      if(!find(i,xlike[i][0])) return 0;
   }
   return 1;
}

int main() {
   int i,j;
   while(scanf("%d",&n)!=EOF) {
      for(i=0;i<n;i++) //xlike
         for(j=0;j<n;j++) scanf("%d",&xlike[i][j]);
      for(i=0;i<n;i++) //ylike
         for(j=0;j<n;j++) scanf("%d",&ylike[i][j]);
      for(i=0;i<n;i++) //get_rank
         for(j=0;j<n;j++) {
            xrank[i][xlike[i][j]]=j;
            yrank[i][ylike[i][j]]=j;
         }
      if(!stable_match()) puts("Impossible");
      else {
         for(i=0;i<n;i++) printf("%d %d\n",i,xmat[i]);
      }
   }
}
\*-------------------------------------------------------*/

/*-------------------------------------------------------*\
```

最大团&&稳定婚姻的应用

```
\*-------------------------------------------------------*/
```

最大独立集[poj1419]

给定一个无向图,选取最多的点,要求这些点都没有边相连.即相互独立.

定理:最大独立集=图的补图(完全图去掉已有的边)的最大团.

要求输出点集.(我觉得用二分匹配也可以)

***--*/**

二分+最大团

给你N个点,要你选出k个点.使得选出的k个点中最近的点对的距离尽量大.

[题目]There are n points in the plane.

Your task is to pick k points (k>=2),

and make the closest points in these k points as far as possible.

Input: N,K.

Output:K个点中最近点对距离的最大值.

[解析]要选出的k个点最近点对距离最大.那么只要二分距离.

然后距离大于mid的连一条边.求最大团.

如果>=k,说明还可以增大最小距离(增大,最大团个数就会变少)

如果<k,说明mid太大了,以至于一些点无法连接.减少最小距离.

***--*/**

计算极大团个数

即最大团的个数

```
//Bron_Kerbosch
int n,m;
int g[N][N];
int cnt;
void countClique(int *p,int ps,int *x,int xs) {
  if(ps==0) {
    if(xs==0) cnt++;
    return;
  }
  for(int i=0;i<xs;i++) {
    int j,v=x[i];
    for(j=0;j<ps&&g[p[j]][v];j++);
    if(j==ps) return;
  }
  int tmp_p[N],tmp_x[N];
  int tmp_ps=0,tmp_xs=0;
  for(int i=0;i<ps;i++) {
    int v=p[i];
    tmp_ps=tmp_xs=0;
    for(int j=i+1;j<ps;j++) {
      int u=p[j];
      if(g[v][u]) tmp_p[tmp_ps++]=u;
    }
    for(int j=0;j<xs;j++) {
      int u=x[j];
      if(g[v][u]) tmp_x[tmp_xs++]=u;
    }
```

```
      countClique(tmp_p,tmp_ps,tmp_x,tmp_xs);
      if(cnt>1000) return;
      x[xs++]=v;
   }
}
void solve() {
   int p[N],x[N];
   for(int i=0;i<n;i++) p[i]=i;
   countClique(p,n,x,0);
}
void init() {
   memset(g,0,sizeof(g));
   cnt=0;
}
int main() {
   int a,b;
   while(scanf("%d%d",&n,&m)!=EOF) {
      init();
      while(m--) {
         scanf("%d%d",&a,&b);
         g[a-1][b-1]=g[b-1][a-1]=1;
      }
      solve();
      if(cnt>1000) puts("Too many maximal sets of friends.");
      else  printf("%d\n",cnt);
   }
}
\*-----------------------------------------------------*/
```

着色问题(四色定理)[最大团的应用]

[题意]给定无向图,为每个点涂上颜色,要求相连的点不能涂相同的颜色.
问:至少需要多少种不同的颜色去涂这些点.

[解析]最大团中,两两涂上不同的颜色后.其他的点涂色的不同颜色数
肯定不大于最大团的不同颜色数,故直接求最大团即可.

```
\*-----------------------------------------------------*/
```

```
/*************************************************************\
```

强连通&&双连通

①强连通:将图缩成一个个互相连通的强连通分量,缩图后的图就不存在环了.

②双连通:A.点双连通,边双连通

B.求割点和桥边

```
\*************************************************************/
```

模板

==========

强连通

```
stack<int>s;
int DNF[N],low[N],belong[N];
bool instack[N],in[N],out[N]; //vis标志,是否有入,出度
int scc,dep; //强连通个数,访问顺序
void tarjan(int u) {
   DNF[u]=low[u]=dep++;
   s.push(u); instack[u]=1;
   int i,v;
   for(i=p[u];i!=-1;i=e[i].nxt) {
      v=e[i].b;
      if(DNF[v]==-1) tarjan(v),low[u]=min(low[u],low[v]);
      else if(instack[v]) low[u]=min(low[u],DNF[v]);
   }
   if(low[u]==DNF[u]) {
      scc++;
      do {
         v=s.top(); s.pop();
         belong[v]=scc;
         instack[v]=0;
      }while(v!=u);
   }
}
void init() {
   memset(DNF,-1,sizeof(DNF));
   memset(instack,0,sizeof(instack));
   while(!s.empty()) s.pop();
   scc=dep=0;
   memset(in,0,sizeof(in));
   memset(out,0,sizeof(out));
   memset(p,-1,sizeof(p));
   cnt=0;
}
int main() {
    int i,j,a,b;
    while(scanf("%d%d",&n,&m)!=EOF) {
        init();
        while(m--) {
            scanf("%d%d",&a,&b);
            addedge(a,b);
        }
```

```
//强连通缩点
        for(i=1;i<=n;i++)
            if(DFN[i]==-1) tarjan(i);
//缩图建边
        for(int u=1;u<=n;u++) {
            for(i=p[u];i!=-1;i=e[i].nxt) {
                int v=e[i].b;
                if(blg[u]==blg[v]) continue;
                in[blg[v]]++; out[blg[u]]++;
            }
        }
//求入度出度为0的点数
        int ru=0,chu=0;
        for(i=1;i<=scc;i++) {
            if(in[i]==0) ru++;
            if(out[i]==0) chu++;
        }
//求需要至少多少条边,使得原图成为强连通分量
        int ret=max(ru,chu);
        if(scc==1) ret=0;
        printf("%d\n",ret);
    }
}
```

==========

边双连通

//先用并查集处理平行边!!!

求桥边:

一条无向边(u,v)是桥.

当且仅当(u,v)为树枝边,且满足DFN(u)<Low(v).

边双连通:

求法同强连通.

```
struct Edge { int b,nxt; bool vis; } e[200005];
int n,m;
stack<int>s;
int DFN[N],low[N],blg[N];
int qiao[N];
int scc,dep,qcnt;
void tarjan(int u) {
    DFN[u]=low[u]=dep++;
    s.push(u);
    int i,v;
    for(i=p[u];i!=-1;i=e[i].nxt) {
        if(e[i].vis) continue;
        e[i].vis=e[i^1].vis=1;
```

```
        v=e[i].b;
        if(DFN[v]==-1) {
            tarjan(v);
            low[u]=min(low[u],low[v]);
            if(DFN[u]<low[v]) qiao[qcnt++]=i; //求桥边
        }else low[u]=min(low[u],DFN[v]);
    }
    if(DFN[u]==low[u]) { ... } //求边连通分量,缩点。同强连通
}
void addedge(int a,int b); //双向边，e[cnt].vis=0
void init() {
    memset(DFN,-1,sizeof(DFN));
    while(!s.empty()) s.pop();
    scc=dep=qcnt=0;  cnt=0;
    memset(p,-1,sizeof(p));
}
int main() {
    int i,j,a,b;
    while(scanf("%d%d",&n,&m)!=EOF) {
        init();
        while(m--) {
            scanf("%d%d",&a,&b);
            addedge(a,b);
        }
        int num=0;
        for(i=1;i<=n;i++)
            if(DFN[i]==-1) tarjan(i),num++;
        if(num!=1) { puts("0"); continue; } //图不连通
        sort(qiao,qiao+qcnt);
        printf("%d\n",qcnt);
        for(i=0;i<qcnt;i++) { //输出桥边
            int id=(qiao[i]|1);
            printf("%d %d\n",e[id].b,e[id^1].b);
        }
    }
}
```

==========

点双连通

```
/*
//根节点如果有多个树枝边则为割点
//普通点如果存在low[v]>=dfn[u]的树枝边的点即为割点
//blg[][]保存第i个双连通块是否包含点j
//如果真要缩成树的话割点不变，每个连通块中除割点外的点缩为一点
//向每个割点连边,得到一棵树
```

求割点:
    一个顶点u是割点,当且仅当满足(1)或(2)
    (1)u为树根,且u有多于一个子树.
    (2)u不为树根,且满足存在(u,v)为树枝边,使得DFN(u)<=Low(v).

点双连通(求块):
    将遍历到的边加入栈中.
    对于割点u.将栈中的边弹出,一直到边i为止.
    这些边组成一个连通分块.
    blg[scc][]表示scc分块的节点.
*/
```
struct Edge { int b,nxt; bool vis; } e[99999];
int n,m;
stack<int>s;
int DFN[N] , low[N];
bool iscut[N];
bool blg[N][N];
int scc,dep,root;
void tarjan(int u) {
    DFN[u]=low[u]=dep++;
    int ct=0; //记录子树个数
    for(int i=p[u];i!=-1;i=e[i].nxt) {
        int v=e[i].b;
        if(e[i].vis) continue;
        e[i].vis=e[i^1].vis=1;
        s.push(i);
        if(DFN[v]==-1) {
            ct++;
            tarjan(v);
            low[u]=min(low[u],low[v]);
            //求割点
            if(u==root) { if(ct>1) iscut[u]=1; } //根节点
            else if(DFN[u]<=low[v]) iscut[u]=1; //其他节点
            //求连通分块
            if(DFN[u]<=low[v]) {
                scc++;
                int ii=-1;
                while(ii!=i) { //保存连通分块
                    ii=s.top(); s.pop();
                    blg[scc][e[ii].b]=1;
                    blg[scc][e[ii^1].b]=1;
                }
            }
        }else {
```

```
            low[u]=min(low[u],DFN[v]);
        }
    }
}
void addedge(int a,int b); //双向边，e[cnt].vis=0
void init() {
    memset(DFN,-1,sizeof(DFN));
    memset(iscut,0,sizeof(iscut));
    memset(blg,0,sizeof(blg));
    while(!s.empty()) s.pop();
    scc=dep=0;
    memset(p,-1,sizeof(p));
    cnt=0;
}
int main() {
    int i,j,a,b;
    while(scanf("%d%d",&n,&m)!=EOF) {
        init();
        while(m--) {
            scanf("%d%d",&a,&b);
            addedge(a,b);
        }
        for(i=1;i<=n;i++)
            if(DFN[i]==-1) root=i,tarjan(i);
        for(i=1;i<=scc;i++) { //输出连通块
            printf("%d: ",i);
            for(j=1;j<=n;j++) {
                if(blg[i][j]) printf("%d.",j);
            } puts("");
        }
    }
}
\*------------------------------------------------------*/

/*------------------------------------------------------*\
```

强连通&&双连通的应用

--/

强连通初步

A.判断是否是一个强连通：scc == 1

B.找出度为0的集合,入度为0的集合

①入度为0的点有几个 == 相互传递信息

②入度和出度最大的有几个 == 构成一个强连通

（至少加几条边让整个图变成强连通）

C.判断有几个环:分量中元素大于1的个数.

--/

受欢迎的牛(只有 1 个度数为 0)

[题意]每一头牛的愿望就是变成一头最受欢迎的牛.

现在有N头牛,给你M对整数(A,B),表示牛A认为牛B受欢迎.

这种关系是具有传递性的,如果A认为B受欢迎,B认为C受欢迎,

那么牛A也认为牛C受欢迎.

你的任务是求出有多少头牛被所有的牛认为是受欢迎的.

[解析]受所有的牛欢迎,那么所有其他的强连通分图都指向它

(出度为0的只有一个)

(因为缩点后是一个无环图,而后形成一个有向无环图,

可以发现唯一的出度为0的点即为所求.

若有多个出度为0的点,则这些点定无法互相连通,无解,输出0)

--/

强连通缩点+树形 dp(累加子节点的总权值)

[题意]受最多人支持的人 (输出其编号)

[思路]缩点,建有向无环反向图,对每个入度为0的点进行dfs,

找出最大的值即可.注意到,答案只能是在反向图入度为0的点中.

--/

缩点+拓扑

[题意]是给你一个有向图, 问你这个图是否符合以下的条件:

对于图上任意两点x,y.都存在一条有向路径,从x到y或从y到x.

[思路]先缩点,变成一个有向无环图,那么题目条件成立的充要条件就是:

缩点后的节点构成一条链,知道这个之后,

就可以选取任意一点往两个方向各找一条最长量,

看是否加起来后是否就是包含所有缩点的长链.

求任意两点间是否都有 u->v 或者 v->u 的路径

--/

仙人掌图

1. 图为强连通图

2. 每条边都属于一个环

这题巧妙的利用了Tarjan强连通算法low[]数组,根据仙人掌图的三个性质

判断是否是仙人掌图, 详细证明这里

性质1 仙人掌图的DFS树没有横向边.

性质2 Low(v)<=DFN(u) (v是u的儿子)

性质3 设某个点v有a(u)个儿子的Low值小于DFS(u),

同时u自己有b(u)条逆向边.那么a(u)+b(u)<2.

```
stack<int>s;
int DFN[N],low[N],belong[N];
bool instack[N],hash[N];
int in[N],out[N];
int scc,dep;
bool tarjan(int u) {
  DFN[u]=low[u]=dep++;
```

```
   s.push(u); instack[u]=1;
   int i,v,cn=0;
   for(i=p[u];i!=0;i=e[i].nxt) {
      v=e[i].b;
      if(hash[v]) return 0;
      if(DFN[v]==-1) {
         if(!tarjan(v))    return 0;
         //子孩子的low值大于当前点的dfn值，说明没有指出去的逆向边。
         //这样<v, u>就成了桥,不是连通图，返回false，性质1.2
         //u 有两条逆向边(本身或儿子，指回去父节点），成了两个圈。
         if(low[v]>DFN[u]) return 0;
         if(low[v]<DFN[u]) cn++;
         low[u]=min(low[u],low[v]);
      }else if(instack[v]) {
         cn++;
         low[u]=min(low[u],DFN[v]);
      }
      if(cn>=2) return 0;
   }
   if(low[u]==DFN[u]) {
      scc++;
      do {
         v=s.top(); s.pop();
         belong[v]=acc;
         instack[v]=0;
      }while(v!=u);
   }
   hash[u]=1; //该点dfs结束，标记
   return 1;
}
int main() {
  int i,j,t,a,b;
  while(scanf("%d",&n)!=EOF) {
      init();
      memset(hash,0,sizeof(hash));
      while(scanf("%d%d",&a,&b),a||b) addedge(a+1,b+1);
      bool ans=tarjan(1);
      for(i=1;i<=n;i++)//若还存在没被遍历到的点,则图不连通
         if(DFN[i]==-1) { ans=0; break; }
      if(ans) puts("YES");
      else    puts("NO");
   }
}
\*------------------------------------------------------*/
```

[双连通寻找奇圈]

经过每条**"边"**一次,含有奇数条边的圈.**(边圈)**

求边双连通,对每一个分量进行交叉染色,同时被染成黑白两色的点,即可以在奇圈内.

(即要存在边圈,就不能有桥边.)

经过每个**"点"**一次,含有奇数个点的圈.**(点圈)**

求点双连通,对每一个块进程交叉染色,同时被染成黑白两色的点,即可以再奇圈内.

(即要存在点圈,就不能有割点.)

***----------*/**

[混合图改有向图]

[题意]

给定一副图,有的边是有向的,有的边是无向的,

是强连通的要求尽可能多的将其中的无向边变成有向边,

使得变过的图任然是一个强连通图.

[思路]

先把所有的有向边都看成是无向边,同样的对于图中的桥,是不能定向的,之后同样对每个边双联通分量进行考虑(是将有向边当成无向边后的边双联通分量)

但是现在这个连通分量内有些边是已经定向的,我们需要找到一颗dfs生成树,使得按照之前的构造方法不与已经定向的边冲突

其实同样按照tarjan算法的方法去进行就可以了,在访问的同时定向,对于有向边,如果访问的方向相反,那么就不予以访问,这样在每条边(假设是now->v)进行完low的函数计算的时候(其实是计算low[v])我们都会判一次是否会有low[v]>low[now],如果上述关系式成立,那就说明按照现在的定向方法,v是不能访问到它的前驱的,所以对于当前边,就对它取反向,这样当前的v节点就可以访问到前驱节点了,因为这个连通分支原来就是双联通的(将边全部看成无向边)所以对当前边的改变,不会影响v的前驱节点到v的连通性(可以用反证法,显然如果改变了这条边就使得v的前驱节点不能访问到v,那么之前从v的前驱到v就只有一条路,而v到v前驱没路,这与双联通矛盾)所以这样构造可以取得双联通图,这样的构造对于所有非桥的无向边都定向了,所以是最优解

```
int n,m;
bool e[N][N];
int DFN[N],low[N];
int scc,dep;
void tarjan(int u,int f) {
    DFN[u]=low[u]=dep++;
    int i,v;
    for(v=1;v<=n;v++) {
        if(v==f) continue;
        if(e[u][v]==0&&e[v][u]==0) continue;
        if(DFN[v]==-1) {
            tarjan(v,u);
            low[u]=min(low[u],low[v]);
            if(DFN[u]<low[v]) { //桥边
                if(e[v][u]) {
                    printf("%d %d %d\n",u,v,2);
                    e[u][v]=e[v][u]=0;
```

```
            }
        }
    }else low[u]=min(low[u],DFN[v]);
  }
}
void dfs(int u,int f) {
  DFN[u]=low[u]=dep++;
  int i,v;
  for(v=1;v<=n;v++) {
    if(v==f||!e[u][v]) continue;
    if(DFN[v]==-1) {
      dfs(v,u);
      low[u]=min(low[u],low[v]);
      if(DFN[u]<low[v]) { //uv定向时,u的子树无法返祖
        if(e[v][u]) {
          printf("%d %d %d\n",v,u,1);
          e[u][v]=e[v][u]=0;
        }
      }else { //uv定向,u的子树可以返祖
        if(e[v][u]) {
          printf("%d %d %d\n",u,v,1);
          e[u][v]=e[v][u]=0;
        }
      }
    }else {
      low[u]=min(low[u],DFN[v]);
      if(e[v][u]) { //返祖边
        printf("%d %d %d\n",u,v,1);
        e[u][v]=e[v][u]=0;
      }
    }
  }
}
void init() {
  memset(DFN,-1,sizeof(DFN));
  scc=dep=0;
  memset(e,0,sizeof(e));
}
int main() {
  int i,j,a,b,c;
  while(scanf("%d%d",&n,&m)!=EOF) {
    init();
    while(m--) {
      scanf("%d%d%d",&a,&b,&c);
```

```
            if(c==1) e[a][b]=1;
            else e[a][b]=e[b][a]=1;
        }
        for(i=1;i<=n;i++)
            if(DFN[i]==-1) tarjan(i,i);
        memset(DFN,-1,sizeof(DFN));
        scc=dep=0;
        for(i=1;i<=n;i++)
            if(DFN[i]==-1) dfs(i,i);
    }
}
\*-------------------------------------------------------*/
```

[最少支撑点]

题意：给定一个连通的无向图,在里面标记一些点,使得这个图在去除任意一个点后,
所有的点和被标记的点是连通的,求出最少要标记多少个点,和方案数.

即缩图后,叶子节点个数,和叶子节点乘积.

①图中不存在割点：ans = n*(n-1)/2

②存在割点：每个连通分块中,如果只含一个割点,那么方案数 ans*=size-1. 需要标记的点数tot++.

PS：原先以为去掉所有割点,然后剩下的每个连通块中节点个数乘积即为结果.
但是发现错了.

***---*/**

[LOW 和 DFN 的应用]

low和DFN的应用

生成一个dfs树.

DFN[u]<low[v] //uv为桥边

DFN[u]<=low[v] //u为割点

记录final[u]为子树中最后被遍历的时间戳.

AB CD:

设这个边为g1-g2,其中DFN[g2]>DFN[g1]

①a,b都在g2的子树内,那么ab可以到达

②a,b都不在g2的子树内,那么ab可以到达

③a,b至少有一个在g2的子树内,则判断DFN[g1]>=low[g2]
如果是的话,那么ab可以到达

④其他情况下不能到达.

AB C:

①ab都不在g1的子树内.则可行

②ab有一个在g1的子树内,求出在a-g1路径上的倒数第二个点k
如果DFN[g1]>low[a].则可行

③ab都在g1的子树内.求出在a-g1路径上的倒数第二个点k1,在b-g1路径上的倒数第二个点k2,
判断是否两者的low都<dfn[g1].则可行

④其他情况下不能到达。

判断一个点v是否在另一个点u的子树内:(所谓的欧拉序列)
    DFN[v]>=DFN[u]&&final[v]<=final[u]

```
int n,m,q;
int DFN[N],low[N];
int fa[N],final[N];
int scc,dep;
void tarjan(int u,int f,int d) {
    DFN[u]=low[u]=++dep;
    fa[u]=f;
    int i,v;
    for(i=p[u];i!=-1;i=e[i].nxt) {
        v=e[i].b;
        if(e[i].vis) continue;
        e[i].vis=e[i^1].vis=1;
        if(DFN[v]==-1) {
            tarjan(v,u,d+1);
            low[u]=min(low[u],low[v]);
        }else low[u]=min(low[u],DFN[v]);
    }
    final[u]=dep;
}
bool ing(int a,int g) { //判断a是否在g的子树内
    if(DFN[g]<=DFN[a]&&final[g]>=final[a]) return 1;
    return 0;
}
bool judge(int a,int b,int g1,int g2) { //AB CD
    if(DFN[g1]>DFN[g2]) swap(g1,g2);
    if(ing(a,g2)==ing(b,g2)) return 1; //都在(或都不在),g2的子树
    if(DFN[g1]>=low[g2]) return 1; //g1-g2不是树枝边(桥边),有返祖边
    return 0;
}
bool judge(int a,int b,int c) { //AB C
    bool x=ing(a,c),y=ing(b,c);
    int flg1=1,flg2=1;
    int v1=a,v2=b;
    if(x==0&&y==0) return 1; //都不在c的子树中
    if(x) { //x在c的子树中
        flg1=0;
        while(fa[v1]!=c) v1=fa[v1];
        if(DFN[c]>low[v1]) flg1=1;
    }
    if(y) { //y在c的子树中
        flg2=0;
```

```
        while(fa[v2]!=c) v2=fa[v2];
        if(DFN[c]>low[v2]) flg2=1;
    }
    if(x&&y&&v1==v2) flg1=flg2=1; //a和b在相同的子树中
    return flg1&&flg2;
}
int main() {
    int i,j,t,cas=0;
    int a,b,c,d,op;
    while(scanf("%d%d",&n,&m)!=EOF) {
        init();
        while(m--) {
            scanf("%d%d",&a,&b);
            addedge(a,b);
        }
        for(i=1;i<=n;i++)
            if(DFN[i]==-1) tarjan(i,i,1);
        scanf("%d",&q);
        while(q--) {
            scanf("%d",&op);
            bool flg=0;
            if(op==1) { //AB CD
                scanf("%d%d%d%d",&a,&b,&c,&d);
                flg=judge(a,b,c,d);
            }else {
                scanf("%d%d%d",&a,&b,&c);
                flg=judge(a,b,c);
            }
            if(flg)  puts("yes");
            else    puts("no");
        }
    }
}
\*-------------------------------------------------------*/
```

[强连通总结]

DFN[u] 时间戳,其节点访问的时间.

low[u] 记录最早的时间戳.

树枝边 u->v, v没遍历过

后向边 v遍历过，且在栈里

(也叫返祖边)

横叉边 v的子树已经完全遍历过,u再访问到v时,叫横叉边

平行边 u到v有两条边

[]当DFN(u)=Low(u)时，以u为根的搜索子树上所有节点是一个强连通分量

基本都在入度为0,或者出度为0的集合中做文章...

至少加几条边,使得原图变成强连通分量.

==> 入度出度0的最大值(注意:scc==1的时候,不需要加边)

[hdu3072] 缩点应用 ==> 最小树形图

最小树形图:每次找最小入边(E),找环(V),收缩O(E).

因为缩点后,环内的传递不需要能量,所以缩图后无环,直接记录最小入边即可.

[poj1904]求二分完美匹配的每个点的可行匹配.

就是图为完美匹配的,要你求:比如X=1 , Y=2 进行匹配的话,图还是可以进行完美匹配.那么 X=1,Y=2为可行匹配.

[解法]:

对于X和Y连边, 然后最初的匹配情况Yi与Xi连边.

然后做强连通缩点.

在同一块强连通分量的男女可以形成可行匹配.

***----------*/**

[双连通总结]

==========

【求割点】:

一个顶点u是割点,当且仅当满足(1)或(2)

(1)u为树根,且u有多于一个子树.

(2)u不为树根,且满足存在(u,v)为树枝边,使得DFN(u)<=Low(v).

==========

【求桥边】:

一条无向边(u,v)是桥.

当且仅当(u,v)为树枝边,且满足DFN(u)<Low(v).

==========

【点连通分支】(叫做:块)

将遍历到的边加入栈中.

对于割点u.将栈中的边弹出,一直到边i为止.

这些边和关联的点组成一个连通分块.割点可以属于多个分块.

点联通求块:

0个环:点数大于边数(只有一种情况,2个点1条边)

1个环:点数等于边数

多个环:点数小于边数

==========

【边双连通分支】: 求法同强连通.

==========

【构造双连通图】:

①一个有桥的连通图，如何把它通过加边变成边双连通图?

方法为首先求出所有的桥,然后删除这些桥边,剩下的每个连通块都是一个双连通子图。

把每个双连通子图收缩为一个顶点，再把桥边加回来，最后的这个图一定是一棵树，边连通度为1。

统计出树中度为1的节点的个数，即为叶节点的个数，记为leaf。

则至少在树上添加(leaf+1)/2条边，就能使树达到边二连通，所以至少添加的边数就是(leaf+1)/2。

具体方法为：

首先把两个最近公共祖先最远的两个叶节点之间连接一条边，
这样可以把这两个点到祖先的路径上所有点收缩到一起，
因为一个形成的环一定是双连通的。
然后再找两个最近公共祖先最远的两个叶节点，这样一对一对找完，
恰好是(leaf+1)/2次，把所有点收缩到了一起。

②无向图改有向强连通图：

直接tarjan出来的即为无向边的定向.

对于桥边,需要正向和反向的有向边.

③混合图改有向强连通图：

先将有向图看做无向图,按照无向图的方式来做,先求出桥边,因为桥边必须是双向的.

然后删除桥边后,再对原图进行tarjan,对于所有无向边进行定向.

如果：

```
(u,v)为树枝边:(DFN[v]==-1)
    if(DFN[u]<low[v]) //v无法返祖;双向边必须定向为: v->u
    else //可以返祖;双向边定向为: u->v
(u,v)为返祖边:
    //可以返祖;定向为: u->v
```

==

【寻找奇圈】

①经过每条"边"一次,含有奇数条边的圈.(边圈)

求边双连通,对于每一个分量进行交叉染色,同时被染成黑白两色的点,即可以在奇圈内.

(即要存在边圈,就不能有桥边.)

②经过每个"点"一次,含有奇数个点的圈.(点圈)

求点双连通,对于每一个块进程交叉染色,同时被染成黑白两色的点,即可以再奇圈内.

(即要存在点圈,就不能有割点.)

---/

/**\

2-SAT

基本思路： 有很多对物品，每对物品只能选其中一个。用强连通来做

建图： 有矛盾的建图 a 和 b 有矛盾，

则选了 a 就只能选~b，选了 b 就只能选~a

**/

/*---*\

2-SAT 的应用

***--*/**

题目分析

==========

判断矛盾:

有矛盾的建边,

每个节点都有两种状态: 选 , 不选 (A 和 非 A)

然后判断选了 A 的话,怎么样.不选 A 的话怎么样.

注意: 一定要把图建完整,不然会有遗漏的矛盾!!!

==========

输出可行解:

对于 tarjan 缩图后的新图.

①求每个 u 对应的矛盾所在分量 opp[u] (为染色做准备)

②缩图后,建反向图.

③进行 topsort()染色;

每次将未染色的 u 染成 W,并且 opp[u]染成 B.

④输出一个可行解. (被染成 W 的点)

==========

A. 钥匙开门[POJ 2723]Get Luffy Out[较难]

两把钥匙为一组，以一个为a，另一个就为~a

对于每一扇门的两把锁X和Y,对应钥匙为x和y,

如果我们用了与x一组的钥匙~x,那么x钥匙就不能用了,

我们就必须去打开Y锁,使用y钥匙,即边 ~x->y,~y->x.

二分能打开门的个数,强连通判解

B. HDU1816[Get Luffy Out][较难]

上题的加强版,这题中一把钥匙可以同时属于多组,

我们把每把钥匙使用(a)和不使用(~a)为两个状态,

先根据组建边,如果一个组x和y两把钥匙,

那么有x->~y,y->~x,然后就和上题一样了

[解析]把钥匙分成:选a或者不选~a,然后制造矛盾.

就变成简单的2-SAT了.这是本质.

C. [题意]平面上,一个圆,圆的边上按顺时针放着n个点.

现在要连m条边,比如a,b,那么a到b可以从平面正面连接,

也可以从平面另一面连接.给你的信息中,每个点最多只会连接的一条边.

问能不能连接这m条边,使这些边都不相交.

[思路]每个边看成2个点:分别表示在正面连接和在反面连接,只能选择一个.

表示为点i和点i'.

如果两条边i和j必须一个画在正面,一个画在反面,有矛盾,那么连边:

i->j',表示i画正面的话,j只能画反面,即j'.

j->i',同理 i'->j, 同理 j’ ->i,同理,然后就是2-sat

D. [题意]给你一张无向图,图的边权为逻辑算符opt值c∈{0, 1},图的点权

为一个0/1的值.对于边w(A,B),需要满足A opt B == c.

你需要确定能不能给所有的顶点都填上一个值使整张图都被满足。

本质:【选了一个东西，必须选另一个东西】

① A or B == 1

那么 A == 1 B的取值无所谓，所以没有约束

A == 0 B必须取1，所以连一条边 A0 -> B1

那么 B == 1 A的取值无所谓，所以没有约束

B == 0 A必须取1，所以连一条边 B0 -> A1

② A and B == 1

那么 A 和 B 都必须取 1 才可以

所以 A0 ->A1 , B0 ->B1 （表示一定要选 A1，B1）

***--*/**

二分+2-SAT

==========

A.Bomb Game

[题意]XoY坐标中,有N对点,给定这N对点的坐标.

现在要求找出一个最大的半径R,使得可以从每对点中选择一个点,

并且这N个点以自己为圆心,半径为R的圆两两不相交.

[思路]二分枚举半径,然后找有矛盾的炸弹建图

[构图]第i个炸弹和第j个炸弹的距离小于2*R (i和j不能为同类)

那么:(选i就不能选j,选j就不能选i)

选i就必须选j^1,即i->(j^1)

选j就必须选i^1,即j->(i^1)

==========

B.Building roads

[题意]是说每个仓库都必须与两个传送点其中的一个且只有一个建路

(仓库与传送点以及传送点与传送点之间的路的距离是曼哈顿距离)

在满足一些like与hate的关系的情况下:

要求任意两个仓库的距离的最大值最小(这里指沿道路的距离)

[解析]题目理解的话就很简单了,在二分前先把like和hate的关系全部建好

(连在s1(a),连在s2(~a))

like: x->y,~x->~y,y->x,y->~x

hate: x->~y,~x->y,y->~x,~y->x;

***--*/**

2-Sat 输出路径

具体操作就是:求强联通,缩点重新建图(这里建反向图,参见论文)

然后给图中的点着色,将一个未着色点x上色,同时把与它矛盾的点y以及y的

所有子孙节点上另外一种颜色.上色完成后,进行拓扑排序,选择一种颜色的

点输出就是一组可行解.

[准备阶段]为着色做准备.

opp[x]保存和 x 的矛盾的连通分量的编号.

在赵爽的论文中有证明,这样可以保证拓扑求解的可行性

b 和 b' 的连通分量矛盾

```
for(i=0;i<n;i++) {
   opp[belong[i*2]]=belong[i*2+1];
   opp[belong[i*2+1]]=belong[i*2];
```

```
    }
```

==========

例题:

[题意]一堆夫妇去参加一对新人的婚礼.(h:husband,w:wife)

人们坐在一个很长很长的桌子的两侧(面对面),新郎新娘在桌子头面对面座.

新娘不希望看见她对面的一排有一对夫妇坐着(夫妇需要分开两排座)

同时,一些人之间有通奸关系,新娘不希望有通奸关系的人同时坐在她对面一排.

问你可否满足新娘的要求,可以的话,输出一种方案:

是要求新娘对面(也就是新郎那边)不能有矛盾,而输出是新娘这边坐着的人.

如果把没有矛盾的染成W色,那么输出的是被染成B的集合.

```
//THE code
const int W=1;
const int B=2;
int opp[N];
int in[N];
int col[N];
void addedge(int u,int v); //有向边
void addedge_topsort(int u,int v); //有向边
void topsort() {
   int i,j,u,v;
   queue<int>q;
   memset(col,0,sizeof(col));
   for(i=1;i<=scc;i++)
      if(!in[i]) q.push(i);
   while(!q.empty()) {
      u=q.front(); q.pop();
      if(col[u]) continue;
      col[u]=W; col[opp[u]]=B;
      for(i=pp[u];i!=-1;i=ee[i].nxt) {
         int v=ee[i].b;
         if(!(--in[v])) q.push(v);
      }
   }
}
int main() {
   int i,j;
   int a,b,c1,c2;
   char s1,s2;
   while(scanf("%d%d",&n,&m),n||m) {
      init();
      for(i=0;i<m;i++) {
         scanf("%d%c%d%c",&a,&s1,&b,&s2);
         c1=(s1=='h')?0:1; c2=(s2=='h')?0:1;
         addedge( a*2+c1 , (b*2+c2)^1 ); //有向边
```

```
        addedge( b*2+c2 , (a*2+c1)^1 );
    }
    addedge(1,0); // 必须选新浪
    //----- 强连通分量 -----
    for(i=0;i<2*n;i++)
      if(DFN[i]==-1) tarjan(i);
    //----- 判断有无解 -----
    bool mak=0;
    for(i=0;i<n;i++)
      if(belong[i*2]==belong[i*2+1]) { mak=1; break; }
    if(mak) { puts("bad luck"); continue; }

    //----- 寻求可行解 -----
    //为着色做准备,
    //opp[x]保存和 x 的矛盾的连通分量的编号
    for(i=0;i<n;i++) {
      opp[belong[i*2]]=belong[i*2+1];
      opp[belong[i*2+1]]=belong[i*2];
    }
    //缩点,反向建边
    for(i=0;i<2*n;i++) {
      int u=belong[i];
      for(j=p[i];j!=-1;j=e[j].nxt) {
        int v=belong[e[j].b];
        if(u==v) continue;
        addedge_topsort(v,u);
        in[u]++;
      }
    }
    //拓扑染色
    topsort();
    for(i=2;i<2*n;i+=2) {
      if(i!=2) printf(" ");
      if(col[belong[i]]==B) printf("%dh",i/2);
      else                  printf("%dw",i/2);
    }puts("");
  }
}
\*----------------------------------------------------------*/
```

输出字典序最小(只能爆搜)

```
==============================================================
```

[题目]某国有n个党派,每个党派在议会中恰有2个代表.
现在要成立和平委员会,该会满足:
每个党派在和平委员会中有且只有一个代表,

如果某两个代表不和,则他们不能都属于委员会
代表的编号从1到2n,编号为2a-1,2a的代表属于第a个党派.
求和平委员会是否能创立.若能.输出字典序最小的解.

```
//THE code
const int R=1;
const int B=2;
const int W=0;
vector<int> gra[16005];
int col[16005],n;
int cnt,ans[16005];
bool dfs(int u) {
   int size=gra[u].size();
   if(col[u]==B) return 0;
   if(col[u]==R) return 1;
   col[u]=R; col[u^1]=B;
   ans[cnt++]=u;
   for(int i=0;i<size;i++) {
      int v=gra[u][i];
      if(!dfs(v)) return 0;
   }
   return 1;
}
bool solve() {
   memset(col,0,sizeof(col));
   for(int i=0;i<n;i++) {
      if(col[i]) continue;
      cnt=0;
      if(!dfs(i)) {
         for(int j=0;j<cnt;j++) {
            col[ans[j]]=W;
            col[ans[j]^1]=W;
         }
         if(!dfs(i^1)) return 0;
      }
   }
   return 1;
}
int main() {
   int i,a,b,m;
   while(~scanf("%d%d",&n,&m)) {
      n<<=1;
      for(int i=0;i<n;i++) gra[i].clear();
      for(int i=0;i<m;i++) {
         scanf("%d%d",&a,&b);
```

```
        a--; b--;
        gra[a].push_back(b^1);
        gra[b].push_back(a^1);
      }
      if(solve()) {
        for(int i=0;i<n;i++) {
          if(col[i]==R) printf("%d\n",i+1);
        }
      }else puts("NIE");
    }
}
\*-------------------------------------------------------*/
```

/**\

欧拉回路

1. 用并查集或dfs判断连通性
2. 根据出入度来判断其性质
   - **有向图回路:** 入度=出度
   - **无向图回路:** 每个点度为偶数
   - **有向图路径:** 入度=出度,或只有2个点,a的出-入=1，且b的入-出=1
   - **无向图路径:** 度为偶数,或只有2个点度为奇数
3. 一些题型:
   - A. 单词接龙
   - B. 并查集判断连通性
   - C. 判断入度和出度的关系
   - D. dfs 搜索欧拉路径
   - E. 如果要输出最小的字典序,只要事先对str排序后,dfs出来的就是最小的
4. 输出路径:

   输出点:

```
for(...) { ... }
ans[ct++]=点;
```

   输出边:

```
for(...) {
    ...
    ans[ct++]=边;
}
```

**/

模板

```
==========================================================================
//下面的是按路径输出点的.
int n;
```

```
int in[N],out[N];
int bin[N],f[N];
int ans[1005],ct;
struct node {
   int s,e,id,vis;
}e[1005];
struct po {
   int id;
   char s[25];
}str[1005];
void dfs(int u) {
   for(int i=1;i<=n;i++) {
      if(!e[i].vis&&e[i].s==u) {
         e[i].vis=1;
         dfs(e[i].e);
         ans[++ct]=i;
      }
   }
}
void init() {
   ct=0;
   memset(in,0,sizeof(in));
   memset(out,0,sizeof(out));
   for(int i=0;i<=27;i++) bin[i]=i,f[i]=0;
}
int main() {
   int i,t;
   while(scanf("%d",&n)!=EOF) {
      for(i=1;i<=n;i++) scanf("%s",str[i].s);
      sort(str+1,str+1+n,cmp);
      for(i=1;i<=n;i++) {
         int a=str[i].s[0]-'a'+1;
         int b=str[i].s[strlen(str[i].s)-1]-'a'+1;
         e[i].s=a; e[i].e=b; e[i].vis=0;
         merge(a,b);
         out[a]++; in[b]++;
         f[a]=1; f[b]=1;
      }
      int scc=0; //判断有几个联通快
      bool mak=0;
      int ru=0,chu=0;
      for(i=1;i<=27;i++) {
         if(bin[i]==i&&f[i]) scc++;
         if(in[i]==out[i]) continue;
```

```
        if(in[i]-out[i]==1) ru++;
        else if(out[i]-in[i]==1) chu++;
        else { mak=1; break; }
      }
      // 判断欧拉回路 or 欧拉路
      if(mak||scc!=1||ru!=chu||ru>1||chu>1) {
        puts("***");
        continue;
      }
      int pos=1; //如果都为偶度,则从1开始
      for(i=1;i<=n;i++) //找度为奇数的点
        if(out[e[i].s]-in[e[i].s]==1) { pos=i; break; }
      dfs(e[pos].s);
      for(i=ct;i>=1;i--) pr intf("%s.",str[ans[i]].s);
    }
}
\*-------------------------------------------------------*/

/*-------------------------------------------------------*\
```

欧拉回路的应用

```
\*-------------------------------------------------------*/
```

每条边来回进过两次

[解析]建单向边,正和反各当做有向边

```
\*-------------------------------------------------------*/
```

磁鼓欧拉回路

[题意]用最长的 01 二进制编码.DeBruijin 图问题.

给定一个数 N,求一个长为 2^N 的 01 串,使得任意 N 个子串

表示 0 到 2^N-1 所有的数恰好一次,且要求字典序最小.

A.要输出串

```
int n,p;
bool hash[2222];
int ans[2222];
void dfs(int u) {
    int t=((u<<1)&((1<<n)-1));
    if(!hash[t]){hash[t]=1;dfs(t);ans[++p]=0;}
    if(!hash[t+1]){hash[t+1]=1;dfs(t+1);ans[++p]=1;}
}
int main() {
    int i;
    while(scanf("%d",&n)!=EOF) {
        memset(hash,0,sizeof(hash));
        p=0;
        dfs(0);
```

```
            printf("%d ",1<<n);
            for(i=1;i<n;i++) printf("0");
            for(i=p;i>=n;i--)printf("%d",ans[i]);
            puts("");
        }
    }
```

==========

B.求长度为 n 的第 k 个值, 00110

```
    //第三个十进制是 3
    int n,k,p;
    bool hash[33333];
    int ans[16][33333];
    void dfs(int u) {
        int t=((u<<1)&((1<<n)-1));
        if(!hash[t]) {hash[t]=1;dfs(t);ans[n][++p]=t;}
        if(!hash[t+1]) {hash[t+1]=1;dfs(t+1);ans[n][++p]=t+1;}
    }
    int main() {
        int i;
        for(n=1;n<=15;n++) {
            p=0; memset(hash,0,sizeof(hash));
            dfs(0);
        }
        while(scanf("%d%d",&n,&k),n||k) {
            printf("%d\n",ans[n][(1<<n)-k]);
        }
    }
\*----------------------------------------------------*/
```

00-99 连接成最短串

```
    [题意]用 0-9 的数字,给你密码的位数,如 n=2
            问你怎么连接数字,每 n 个数字遍历 00-99
            且长度为  10^n+n-1
    [思路]对于每一个长度为 n 的串,让该串的前 n-1 位为一个节点,
            后 n-1 位为另一个节点,那么他俩之间的边也就确定了该串.
            在该图中遍历,每一条边遍历一次,寻找欧拉回路,
            找其中字典序最小的输出即是.这里得用非递归的算法.
            递归的算法会导致栈溢出.
            如 n=6,对于密码 123456,则在 12345 和 23456 之间建立一条边
            (前者到后者|  的有向边),这条边就代表 123456 这个密码.边上的值是 6.
//需要手动模拟栈
bool vis[9999999];
void dfs(int u) {
    for(int i=p[u];i!=-1;i=e[i].nxt)
        if(!e[i].vis) {
```

```
            e[i].vis=1;
            dfs(e[i].b);
            printf("%d",e[i].b%10);
        }
}
int main() {
    int i,j,k;
    while(scanf("%d",&n),n) {
        if(n==1) { puts("0123456789"); continue; }
        for(k=1,i=1;i<n;i++) k*=10;
        int mod=k/10;
        init(k);
        for(i=0;i<k;i++) {
            int tmp=i%mod;
            tmp*=10;
            for(j=9;j>=0;j--) addedge(i,tmp+j);
        }
        memset(vis,0,sizeof(bool)*(cnt+1));
        dfs(0);
        for(i=1;i<n;i++) putchar('0');
        for(i=len;i>=1;i--) putchar(ans[i]);
        puts("");
    }
}
\*------------------------------------------------------*/
```

混合欧拉回路

A.并查集判断连通性

B.mak 判断是 欧拉回路 还是 欧拉路!

C.处理欧拉路,即加一条边使首尾相连,就变成欧拉回路了!!!)

D.网络流是否满流来判断是否是欧拉回路

```
//判断是否是 欧拉回路
//Code1
int n,m;
int in[N],out[N];
int e[N][N];
void init() {
    memset(in,0,sizeof(in));
    memset(out,0,sizeof(out));
    memset(e,0,sizeof(e));
}
int main() {
    int i,t,a,b,c;
    while(scanf("%d%d",&n,&m)!=EOF) {
        init();
```

```
        for(i=1;i<=m;i++) {
            scanf("%d%d%d",&a,&b,&c);
            if(a==b) continue;
            in[b]++; out[a]++;
            if(c!=1) e[a][b]++;
        }
        int mak=0;
        int sum=0;
        for(i=1;i<=n;i++) {
            out[i]-=in[i];
            if(out[i]%2==1) {mak=1;break;}
            out[i]/=2;
            if(out[i]>0) {
                e[0][i]=out[i];
                sum+=out[i];
            }else e[i][n+1]=-out[i];
        }
        if(mak) { puts("impossible"); continue; }
        int ans=sap(0,n+1,n+2);
        if(ans==sum) puts("possible");
        else         puts("impossible");
    }
}
=========================================================
//Code2
int e[N][N];
int in[N],out[N];
int bin[N],f[N];
void init() {
    memset(in,0,sizeof(in));
    memset(out,0,sizeof(out));
    memset(e,0,sizeof(e));
    for(int i=0;i<=27;i++) bin[i]=i,f[i]=0;
}
int main() {
    int i,a,b,c;
    char ss[1111];
    int op;
    while(scanf("%d",&n)!=EOF) {
        init();
        for(i=1;i<=n;i++) {
            scanf("%s%d",ss,&op);
            a=ss[0]-'a'+1; b=ss[strlen(ss)-1]-'a'+1;
            out[a]++; in[b]++;
```

```
            merge(a,b); f[a]=f[b]=1;
            if(op==1) e[a][b]++;
        }
        int scc=0;
        for(i=1;i<=27;i++)
            if(bin[i]==i&&f[i]) scc++;
        if(scc!=1) { puts("Poor boy!"); continue; }
        int mak=0,sum=0;
        int x=-1,y;
        for(i=1;i<=27;i++)
            if((in[i]+out[i])%2==1) {
                mak++;
                if(x==-1)    x=i;
                else             y=i;
            }
        //判断奇数度的个数
        if(mak!=2&&mak!=0) { puts("Poor boy!"); continue; }
        if(mak==2) { //连接两奇度点,使之变成欧拉回路
            out[x]++; in[y]++;
            e[x][y]++;
        }
        for(i=1;i<=27;i++) {
            out[i]-=in[i];
            out[i]/=2;
            if(out[i]>0) {
                e[0][i]=out[i];
                sum+=out[i];
            }else e[i][27]=-out[i];
        }
        int ans=sap(0,27,28);
        if(ans==sum) puts("Well done!");
        else                puts("Poor boy!");
    }
}
\*----------------------------------------------------*/


/**********************************************************************\
```

竞赛图

有向图中, 任意两个点不是u->v就是v->u的完全图

性质:　竞赛图肯定有哈密顿路

若图为强连通图, 则有哈密顿回路

证明:

对一个已得到的一条链,一点没有在链中的点一定能连在这条链上,
设插入之前,该链表示: a1 -> a2 -> .... ak .... an-1 -> an.
现在要插入a[n+1]
假设a[n+1]->a[1] 或 a[n]->a[n+1] 直接查到首尾.
否则:有a[1]->a[n+1] 且 a[n+1]->a[n]
假如有an+1->a2 或 a[n-1]->a[n+1] 可以插入,否则...
最后最迟一定在中间能插入.

题目: [zoj3332]就是一道竞赛图中求哈密尔顿路的题.
[hdu3414]是一道竞赛图中求哈密尔顿图的题.
同样找链,看找到的链首尾能否相连,能则能成环.
整个过程枚举每个点首先插入得到的链.

**********/

/*----------*\

竞赛图的应用

----------/

找三元环

```
bool a[5010][5010];
char s[5050];
int main() {
   int n,i,j;
   scanf("%d",&n);
   for(i=0;i<n;i++) {
      scanf("%s",s);
      for(j=0;j<n;j++) a[i][j]=(s[j]-'0');
   }
   vector<int>b(1,0);
   for(i=1;i<n;i++) {
      int l=0,r=i-1;
      while(l<i&&a[i][b[l]]!=1) l++;
      while(r>=0&&a[b[r]][i]!=1) r--;
      if(l==r+1) b.insert(b.begin()+l,i);
      else { printf("%d %d %d\n",i+1,b[l]+1,b[r]+1);  break;}
   }
   if(i==n) printf("-1\n");
   return 0;
}
```

----------/

输出哈密顿路

```
//用静态链表,因为插入频繁
struct node {
   int c,nxt;
```

```
}res[N];
int ct,p;
bool e[N][N];
int n,m;
int main() {
   int i,j,c;
   while(scanf("%d",&n)!=EOF) {
      for(i=1;i<=n;i++)
         for(j=1;j<=n;j++)
            scanf("%d",&c),e[i][j]=c;
      res[ct=1].c=1; res[1].nxt=-1;
      p=1;
      for(i=2;i<=n;i++) {
         bool mak=0;
         if(e[i][res[p].c]) { // 插入前面
            res[++ct].c=i;
            res[ct].nxt=p;
            p=ct; mak=1;
         }else {
            int tmp=p;
            while(res[tmp].nxt!=-1) { //插入中间
               int pp=res[tmp].nxt;
               if(e[res[tmp].c][i]&&e[i][res[pp].c]) {
                  res[++ct].c=i;
                  res[ct].nxt=pp;
                  res[tmp].nxt=ct;
                  mak=1; break;
               }
               tmp=pp;
            }
            if(mak==0&&e[res[tmp].c][i]) { //插入最后
               res[++ct].c=i;
               res[ct].nxt=-1;
               res[tmp].nxt=ct;
               mak=1;
            }
         }
         if(!mak) break;
      }
      if(i<=n) {puts("Impossible");continue;}
      while(res[p].nxt!=-1)
         printf("%d ",res[p].c),p=res[p].nxt;
      printf("%d\n",res[p].c);
   }
```

```
    }
\*-------------------------------------------------------*/
```

输出哈密顿回路

[解析]①强连通判断有无哈密顿回路

②插入点

3

010

000

110

1 2 3 4 5

1 2 3 6 4 5

前面 1 2 3 对6 e[1][6]=1

后面 4 5 对6 e[6][4]=1

1 2 3 4 5 ->6 ==1

[具体步骤]

A.判断强连通

B.先找三元环 S

C.对于i有进有出,插入

D.剩下 X只进,Y只出

E.找Y到X的任意两点的链随意加入S

```
//THE code
struct node {
   int b;
   int nxt;
}e[2222222],res[N];
int p[N],pi;
int cnt,ct;
int S[N],T[N],st,tt;
bool g[N][N];
bool vis[N];
int n,m;
int in[N],out[N];
void find_cir() { //找简单三元环
   int h[1111];
   h[ct=1]=1;
   for(int i=2;i<=n;i++) {
      int l=1,r=i-1;
      while(l<i&&g[i][h[l]]!=1)  l++;
      while(r>=1&&g[h[r]][i]!=1) r--;
      if(l==r+1) { //插入 r l 之间,把l-ct往后移
         ct++;
         for(int j=ct;j>l;j--) h[j]=h[j-1];
         h[l]=i;
      }else { // a->b->c->a
```

```
            res[1].b=i; res[pi=1].nxt=2;
            res[2].b=h[l]; res[2].nxt=3;
            res[3].b=h[r]; res[3].nxt=-1;
            vis[i]=vis[h[l]]=vis[h[r]]=1;
            return;
        }
    }
}
bool insert(int i) { //插点
    int tmp=pi;
    while(res[tmp].nxt!=-1) {
        int pp=res[tmp].nxt;
        if(g[res[tmp].b][i]&&g[i][res[pp].b]) {
            res[++ct].b=i;
            res[ct].nxt=pp;
            res[tmp].nxt=ct;
            return 1;
        }
        tmp=pp;
    }
    if(g[res[tmp].b][i]&&g[i][res[pi].b]) {
        res[++ct].b=i;
        res[ct].nxt=-1;
        res[tmp].nxt=ct;
        return 1;
    }
    return 0;
}
int path[N];
bool hh[N];
int spfa(int s) { //找S->T的链
    int i,u;
    queue<int>q;
    memset(hh,0,sizeof(hh));
    memset(path,-1,sizeof(path));
    hh[s]=1; q.push(s);
    while(!q.empty()) {
        u=q.front(); q.pop();
        for(i=p[u];i!=-1;i=e[i].nxt) {
            int v=e[i].b;
            if(vis[v]) continue;
            if(hh[v]) continue;
            hh[v]=1; path[v]=u;
            q.push(v);
```

```
        if(in[v]) return v;
      }
   }
}
void init() {
   memset(DFN,-1,sizeof(DFN));
   memset(low,-1,sizeof(low));
   memset(instack,0,sizeof(instack));
   while(!s.empty()) s.pop();
   scc=num=dep=0;
   memset(p,-1,sizeof(p));
   cnt=0;
   memset(vis,0,sizeof(vis));
   memset(in,0,sizeof(in));
   memset(out,0,sizeof(out));
}
int main()
{
   int i,j;
   int a,b,c;
   bool mak;
   while(scanf("%d",&n),n) {
      init();
      for(i=1;i<=n;i++)
         for(j=1;j<=n;j++) {
            scanf("%d",&c);
            g[i][j]=c;
            if(c) addedge(i,j);
         }
      //----- 判断是否有回路 -----
      if(n==1)     { puts("1");  continue; }
      else if(n==2) { puts("-1"); continue; }
      tarjan(1);
      if(!(scc==1&&num==n)) { puts("-1"); continue; }
      //----- 找三元环 -----
      find_cir();
      //----- 在三元环内差点 -----
      //----- 并且分出只出和只入的点 -----
      ct=3; st=tt=0;
      for(i=1;i<=n;i++) {
         if(vis[i]) continue;
         mak=insert(i);
         if(mak) { vis[i]=1; }
         else {
```

```
      if(g[res[pi].b][i]) //只出
        S[++st]=i,out[i]=st;
      else //只入
        T[++tt]=i,in[i]=tt;
    }
  }
  //----- 对于只出或只入的点进行插入 -----
  while(ct!=n) {
    bool flg=1;
    for(i=1;i<=st;i++) { //重新插
      mak=insert(S[i]);
      if(mak) {
        vis[S[i]]=1;
        out[S[i]]=0;
        S[i]=S[st]; out[S[st]]=i;
        i--; st--;
        flg=0;
      }
    }
    for(i=1;i<=tt;i++) { //重新插
      mak=insert(T[i]);
      if(mak) {
        vis[T[i]]=1;
        in[T[i]]=0;
        T[i]=T[tt]; in[T[tt]]=i;
        i--; tt--;
        flg=0;
      }
    }
    if(flg) { //找链
      int v=spfa(S[1]);
      T[in[v]]=T[tt]; in[T[tt]]=in[v]; //弹出
      tt--; in[v]=0; vis[v]=1;
      // 出去-> a->b ->进去 a S[1] , b v
      res[++ct].b=v; res[ct].nxt=-1; b=ct;
      while(path[v]!=-1) {
        int u=path[v];
        res[++ct].b=u;
        res[ct].nxt=ct-1;
        out[u]=0; vis[u]=1;
        v=u;
      } a=ct;
      for(i=1;i<=st;i++)
        if( out[S[i]]==0 ) {
```

```
                while(out[S[st]]==0&&st>0) st--;
                if(st<i) break;
                S[i]=S[st]; out[S[st]]=i;
                st--;
              }
            // 将 a->...->b 插入 C
            int pp=res[pi].nxt;
            res[pi].nxt=a;
            res[b].nxt=pp;
          }
        }
        //----- output -----
        while(res[pi].nxt!=-1)
          printf("%d ",res[pi].b),pi=res[pi].nxt;
        printf("%d\n",res[pi].b);
    }
}
\*-------------------------------------------------------*/
```

[构造哈密顿路,每个点度 d(v)>=(n+1)/2]

则对于任意的u,v 都有d(u)+d(v)>=n

(题目所说的条件强于Ore条件,Ore条件是,对于一个无向图,任意两个互异不相邻的点的度数(去除自环和重边)和不小于点数).

算法:

(1) 随意找到一条路.goto (2).

(2) 如果这条路的首尾相邻且路长 != n(n为顶点数),goto (a);

若首尾相邻且路长==n,输出;

否则,先将此路向两侧延伸至无法延伸,goto (3).

(a) 随便找出一条连接路中顶点和路外顶点的边,把它加入路,并从路中删去一条和新加入的边相邻的边,从而使得路变成一个更长的路.

(3) 设路径为a-k1-k2-...-km-b,找到路中一个顶点kj,要求满足a和kj相邻且k(j-1)和b相邻.

这样删除边k(j-1)-kj,加入边a-kj和k(j-1)-b,从而使得路变成一个首尾相邻的等长的路.

证明:

首先,满足Ore条件的图是连通的.设图中有n个点.否则若图不连通,则可以分为两个互不连通部分,分别有s个点和(n-s)个点.

我们任取这两个互不连通部分中各一点,它们的度数和最多s-1+n-s-1=n-2<n.这不满足Ore条件.得证.

既然满足Ore条件的图是连通的,(2)算法总可以成功找到这样一条边. 再证明(3).

由于这个路是不能再延伸的,所以路的头尾两点发出的边都指向路中的点,且这两个点的度数和>=n(这两个点一定不相邻,见(2)).

(3)中可能成为j的下标有2...m这m-1个,由于m-1<n,由抽屉原理,总存在满足条件的j.

这样整个算法就能正确的执行.每次(3)开始之前都一定要延伸路径,强烈提示注意.

--/

/**\

拓扑排序

方法如下:

(1)从有向图中选择一个没有前驱(即入度为0)的顶点并且输出它.

(2)从网中删去该顶点,并且删去从该顶点发出的全部有向边.

(3)重复上述两步,直到剩余的网中不再存在没有前驱的顶点为止.

**/

模板

```
==========================================================================
//mak=0  合法; mak=1  不完整; mak=2  冲突;
int topsort() {
   int i,j,u;
   int mak=0;
   int ct=0,tot=0;
   queue<int>q;
   for(i=1;i<=n;i++)
      if(!in[i]) q.push(x),ct++,tot++;
   if(ct>1) mak=1; //不完整
   ct=0;
   while(!q.empty()) {
      u=q.front(); q.pop();
      for(i=p[u];i!=-1;i=e[i].nxt) {
         int v=e[i].b;
         if((--in[v])==0) q.push(v),ct++,tot;
      }
      if(ct>1) mak=1; //不完整
      ct=0;
   }
   if(tot!=n) mak=2; //存在环,无法拓扑
   return mak;
}
```

--/

/*--*\

拓扑排序的应用

--/

拓扑判环

若图是连通图,用topsort判断被拓扑到的节点数是否等于n.

如果小于n,则说明存在环.

***----------*/**

三维拓扑排序

[题意]给出N个长方体,以及这些长方体之间的M个关系;

要你输出满足做给关系的 N 个长方体的坐标:x1,y1,z1,x2,y2,z2;

长方体的内部个点(x,y,z)满足 x1<=x<=x2,y1<=y<=y2,z1<=z<=z2.

I i j: Ci和Cj可能相交.

X i j: 相交面积为0,并且Ci内部所有的点的X坐标小于Cj内部所有点的X坐标.

Y i j: 同理.

Z i j: 同理.

```
vector<int>g[4][N];
int in[4][N];
int rank[4][N];
int ans[4][N];
int cnt[4];
bool topsort(int id) {
   int i,u;
   priority_queue<int>q;
   for(i=1;i<=2*n;i++)
      if(!in[id][i]) q.push(i);
   while(!q.empty()) {
      u=q.top(); q.pop();
      rank[id][++cnt[id]]=u;
      int sz=g[id][u].size();
      for(i=0;i<sz;i++) {
         int v=g[id][u][i];
         if((--in[id][v])==0) q.push(v);
      }
   }
   return cnt[id]==2*n;
}
void init() {
   for(int i=0;i<4;i++)
      for(int j=0;j<=2*n;j++)
         in[i][j]=0,g[i][j].clear();
   cnt[1]=cnt[2]=cnt[3]=0;
   for(i=1;i<4;i++)
      for(j=1;j<=n;j++)
         g[i][j].push_back(j+n),in[i][j+n]++;
}
int main() {
   int i,j,a,b;
   char op[11];
   while(scanf("%d%d",&n,&m),n||m) {
```

```
      init();
      while(m--) {
         scanf("%s%d%d",op,&a,&b);
         if(op[0]=='I') {
            g[1][b].push_back(a+n);
            g[2][b].push_back(a+n);
            g[3][b].push_back(a+n);
            in[1][a+n]++; in[2][a+n]++; in[3][a+n]++;
            g[1][a].push_back(b+n);
            g[2][a].push_back(b+n);
            g[3][a].push_back(b+n);
            in[1][b+n]++; in[2][b+n]++; in[3][b+n]++;
         }else if(op[0]=='X') {
            g[1][a+n].push_back(b);
            in[1][b]++;
         }else if(op[0]=='Y') {
            g[2][a+n].push_back(b);
            in[2][b]++;
         }else if(op[0]=='Z') {
            g[3][a+n].push_back(b);
            in[3][b]++;
         }
      }
      bool mak=1;
      for(i=1;i<=3;i++)
         if(!topsort(i)) { mak=0; break; }
      if(!mak) puts("IMPOSSIBLE\n");
      else {
         puts("POSSIBLE");
         for(i=1;i<=2*n;i++)
            for(j=1;j<4;j++)
               ans[j][rank[j][i]]=i;
         for(i=1;i<=n;i++) {
            printf("%d %d %d ",ans[1][i],ans[2][i],ans[3][i]);
            printf("%d %d %d\n",ans[1][i+n],ans[2][i+n],ans[3][i+n]);
         }
      }
   }
}
\*-------------------------------------------------------*/
```

[执行任务]

给定 Pi , Di //任务花费时间,任务估计完成时间.

//Ci 为实际完成时间.

求解的是 max{Cj-dj,0} 最小.而不是总和最小.

所以按照贪心,拓扑的时候.
没有约束的多个任务,先执行deadline小的.
--/

/**\

网络流

网络流的变换很多.所以要靠多做题来找感觉.

①最大流:S-T最大流; 分层最大流;

②最小割:最大点权独立集;最大权闭包;最小割点集;最小割边集;二者取其一问题;

③费用流:分阶段费用流;

④全局最小割

⑤上下界网络流

==========

[INIT]

```
void addedge(int a,int b,int c) {
    e[cnt].b=b; e[cnt].c=c; e[cnt].nxt=p[a]; p[a]=cnt++;
    e[cnt].b=a; e[cnt].c=0; e[cnt].nxt=p[b]; p[b]=cnt++; }
void init() { memset(p,-1,sizeof(p)); cnt=0; }
```

**/

模板

==========

EK(连接表)

```
//call: ek(S,T,N);
int pre[N],path[N];
bool vis[N];
bool spfa(int s,int t,int n) {
    int i,u;
    queue<int>q;
    q.push(s);
    memset(vis,0,sizeof(vis));
    while(!q.empty()) {
        u=q.front(); q.pop();
        for(i=p[u];i!=-1;i=e[i].nxt) {
            int v=e[i].b;
            if(e[i].c>0&&!vis[v]) {
                pre[v]=u , path[v]=i;
                vis[v]=1; q.push(v);
                if(v==t) return 1;
            }
        }
    }
```

```
    return 0;
}
int ek(int s,int t,int n) {
    int i,aug,maxflow=0;
    while(spfa(s,t,n)) {
        aug=inf;
        for(i=t;i!=s;i=pre[i]) aug=min(aug,e[path[i]].c);
        for(i=t;i!=s;i=pre[i])
            e[path[i]].c-=aug,e[path[i]^1].c+=aug;
        maxflow+=aug;
    }
    return maxflow;
}
```

==========

SAP(连接表)

```
//call: sap(S,T,N);
int pre[N],dis[N],gap[N],cur[N];
int sap(int s, int t,int n) {
    int i,v,u,c;
    int aug=-1,maxflow=0;
    for(i=0;i<=n;i++) cur[i]=p[i];
    memset(dis,0,sizeof(dis));
    memset(gap,0,sizeof(gap));
    u=pre[s]=s,
    gap[0]=n;
    while(dis[s]<n) {
loop:
        for(int &ee=cur[u];ee!=-1;ee=e[ee].nxt) {
            v=e[ee].b; c=e[ee].c;
            if(c>0&&dis[u]==dis[v]+1) {
                if(aug==-1||aug>c) aug=c;
                pre[v]=u;
                u=v;
                    if(v==t) {
                        maxflow+=aug;
                        for(u=pre[u];v!=s;v=u,u=pre[u])
                            e[ cur[u] ].c-=aug , e[ cur[u]^1 ].c+=aug;
                        aug=-1;
                    }
                    goto loop;
            }
        }
        int mindis=n;
        for(i=p[u];i!=-1;i=e[i].nxt) {
```

```
            v=e[i].b;
            if(e[i].c>0&&mindis>dis[v])
                cur[u]=i , mindis=dis[v];
        }
        if((--gap[dis[u]])==0) break;
        gap[dis[u]=mindis+1]++;
        u=pre[u];
    }
    return maxflow;
}
```

==========

费用流(连接表)

```
//call: mfmc(S,T,N);
void addedge(int a,int b,int c,int w) {
   e[cnt].b=b; e[cnt].c=c; e[cnt].w=w;  e[cnt].nxt=p[a]; p[a]=cnt++;
   e[cnt].b=a; e[cnt].c=0; e[cnt].w=-w; e[cnt].nxt=p[b]; p[b]=cnt++;
}
struct Edge {
    int b,c,w,nxt;
}e[999999];
int p[N];
int cnt;

int pre[N],path[N],dis[N];
bool vis[N];
bool spfa(int s,int t,int n) {
    int i,u;
    queue<int>q;
    for(i=0;i<=n;i++) dis[i]=inf;
    memset(vis,0,sizeof(vis));
    dis[s]=0; q.push(s);
    while(!q.empty()) {
        u=q.front(); q.pop();
        vis[u]=0;
        for(i=p[u];i!=-1;i=e[i].nxt) {
            int v=e[i].b;
            int c=e[i].c;
            int w=e[i].w;
            if(c>0&&dis[v]>dis[u]+w) {
                dis[v]=dis[u]+w;
                pre[v]=u , path[v]=i;
                if(!vis[v]) q.push(v),vis[v]=1;
            }
        }
```

```
    }
    if(dis[t]==inf) return 0;
    return 1;
}
int mfmc(int s,int t,int n) {
    int i;
    int aug,maxflow=0,maxcost=0;
    while(spfa(s,t,n)) {
        aug=inf;
        for(i=t;i!=s;i=pre[i])
            if(aug>e[path[i]].c) aug=e[path[i]].c;
        for(i=t;i!=s;i=pre[i])
            e[ path[i] ].c-=aug , e[ path[i]^1 ].c+=aug;
        maxcost+=dis[t]*aug;
        maxflow+=aug;
    }
    return maxcost;
}
```

==========

无向图最小割

```
//下标从1开始
//call: Stoer_Wagner(N);
int e[N][N];
int v[N];
int dis[N];
bool vis[N];
int Stoer_Wagner(int n) {
   int i,j;
   int res=inf;
   for(i=0;i<=n;i++) v[i]=i; //保存顶点 , 固定顶点为0
   while(n>1) {
      int k=2,pre=1;
      for(i=2;i<=n;i++) {
         dis[v[i]]=e[v[1]][v[i]]; //初始化距离数组dis[]
         if(dis[v[i]]>dis[v[k]]) k=i; //寻找最大距离——求最大生成树
      }
      memset(vis,0,sizeof(vis));
      vis[v[1]]=1;
      //求最大生成树，并进行最小割操作。
      for(i=2;i<=n;i++) {
         if(i==n) {
            //只剩最后一个没加入集合的点，更新最小割
            res=min(res,dis[v[k]]);
            //合并最后一个点以及推出它的集合中的点
```

```
            for(j=1;j<=n;j++) {
               e[v[pre]][v[j]]+=e[v[j]][v[k]];
               e[v[j]][v[pre]]+=e[v[j]][v[k]];
            }
            v[k]=v[n--]; //缩点后的图
         }
         vis[v[k]]=1;
         pre=k;
         k=-1;
         for(j=2;j<=n;j++)
            if(!vis[v[j]]) {
               //将上次求的maxj加入集合，合并与它相邻的边到割集
               dis[v[j]]+=e[v[pre]][v[j]]; //dis[]保存的是一个积累量。
               if(k==-1||dis[v[k]]<dis[v[j]]) k=j;
            }
      }
   }
   return res;
}
int main()
{
   int n,m,a,b,c;
   while(scanf("%d%d",&n,&m)!=EOF) {
      memset(e,0,sizeof(e));
      while(m--) {
         scanf("%d%d%d",&a,&b,&c);
         e[a][b]+=c; e[b][a]+=c;
      }
      int ans=Stoer_Wagner(n);
      printf("%d\n",ans);
   }
}
```

==========

上下界网络流

A.[求可行流]

如果是有源汇的,就连一条addedge(T,S,inf); 变成无源无汇.

新建超级源汇点 SS , TT.

①对于每条边 (a,b,low,up) :

```
addedge(SS,b,low);
addedge(a,TT,low);
addedge(a,b,up-low);
```

②跑一次SS到TT最大流,判断SS到TT是否满流,不满流则无解;

③残留网络中,将原图中每条边的流量+该边的下界low.即为该边的可行流.

B.[求源汇最大流]

如果是有源汇的,就连一条addedge(T,S,inf);

新建超级源汇点 SS , TT.

①对于每条边,同理.

②跑一次SS到TT最大流,判断SS到TT是否满流,不满流则无解;

③再从S到T的最大流.

④残留网络中,将原图中每条边的流量+该边的下界low.即为该边的最大流的可行流.

⑤最大流的答案为:S到T的最大流.

C.[求源汇最小流]

先不加T到S的边.

①对于每条边,同理.

②跑一次SS到TT最大流;

③再加上addedge(T,S,inf) , 再跑一次SS到TT的最大流.

④两次流之和如果不是满流,则无解.

⑤残留网络中,将原图中每条边的流量+该边的下界low.即为该边的最小流的可行流.

⑥最小流的答案为:T到S边流过去的流量.

CODE:

最大流:

```
int SS=n+1,TT=SS+1;
int tot=0; //记录下界流的总和
for(i=1;i<=m;i++) {
    int a=road[i].a;
    int b=road[i].b;
    int low=road[i].low;
    int up=road[i].up;
    addedge(a,b,up-low);
    tot+=low;
}
for(i=1;i<=m;i++) {
    addedge(SS,road[i].b,road[i].low);
    addedge(road[i].a,TT,road[i].low);
}
int ret=sap(SS,TT,TT);
if(ret!=tot) puts("NO"); //无解
else { puts("YES");
    for(i=1;i<=m;i++) { //每条边的流量
        printf("%d\n",road[i].low+e[i*2-1].c);
    }
}
// addedge(T,S,inf); //有源汇的情况,要变成无源汇
```

//--

最小流:

```
int SS=n+1,TT=SS+1;
int tot=0;
for(i=1;i<=m;i++) {
```

```
            int a=road[i].a;
            int b=road[i].b;
            int low=road[i].low;
            int up=road[i].up;
            addedge(a,b,up-low);
            tot+=low;
        }
        for(i=1;i<=m;i++) {
            addedge(SS,road[i].b,road[i].low);
            addedge(road[i].a,TT,road[i].low);
        }
        int ret=sap(SS,TT,TT);
        addedge(T,S,inf); //加边,再跑一次
        ret+=sap(SS,TT,TT);
        if(ret!=tot) puts("Impossible"); //无解
        else {
            printf("%d\n",e[cnt-1].c); //最小流
            for(i=1;i<=m;i++) printf("%d\n",road[i].low+e[i*2-1].c);
        }
\*-------------------------------------------------------*/

/*-------------------------------------------------------*\
```

网络流建图总汇

```
\*-------------------------------------------------------*/
```

inf 表示不割 , 拆点 , 拆边

```
\*-------------------------------------------------------*/
```

点有权值,边没权值

拆点,拆成入和出. i 和 i' .

建边 add(i,i',val[i]); add(i,j,inf);

| **A问:**攻占点能获得的**最大收益** hdu3061 小白攻城

解:点权正的连S(容量val),负的连T(-val),有关系的a连b(inf)

总收益-最大流

| **B问:最小割点集**(割掉最少的点(权值总和),使S和T分割) hdu3491 小偷

解:拆点+最大流

拆成i和i+n,(本身真身(入)连替身(出),不同的两个点替身连真身)

对于本身i连i+n,流:i的权值

对于其他i+n连j,流:INF

最后求S+n到T的流

| **C问:最大点权独立集**,在带权无向图G中,点权之和最大的独立集.方格取数

(不能取相连的)

```
// 定理:
// 1. 最小点权覆盖集  =  最小割  =  最大流
// 2. 最大点权独立集  =  总权  -  最小点权覆盖集
```

// 解题:
// 1.先染色,取一个点染白色,和它相邻的点染黑色(i+j为奇数的变黑色,偶数的白色)
// 2.每个白点向它相邻的黑点连一条边,容量为inf
// 3.增加源点S,向每一个白色点连一条边,容量为白点的权
// 4.增加汇点T,每个黑点向T连一条边,容量为黑点的权
// 5.求MaxWVIS = TotalFlow - MaxFlow
// (i+j奇数连源点,偶数连汇点)

| ④问:③的变形,相连的同时取,收益是为2*(x&y) hdu3657
最大点权独立集=所有点的权值之和-最大流.

***----------*/**

二分图的点权覆盖于点权独立集

建立二分图 S,T
S连x部 流:x的点权
T连y部 流:y的点权
x连y 流:inf

***----------*/**

点有权值,边也有权值(最大权闭包)

A. [题目]每个点有费用,连通一条边,可得到相应收益,问建多少个点,使收益最大
[建图]把边看做点
S连点,流为点权(费用)
两节点连相应的边,inf
边连T,流为边的收益
最大权闭包=总收益-最大流

B. [题目]公司收益①的变形,投资一个公司收益,帮公司建路(必须建所有的费用)
[建图] 源点连每个公司,容量为收益(税收)
汇点连每个公司,容量为总费用(每个公司所commit的)
存在关系的公司之间连边,容量正无穷.
最后答案=收益总和-最大流.

***----------*/**

最小权的环覆盖整个图

[思路]
观察被环覆盖后的图,图中每个点的出入度都为一,考察每个点,如果有边连入,
它的入度就加1,如果有边连出,它的出度就加1(废话).
所以将每个点拆成两个,一个表示出度,一个表示入度,用网络流模型.
源点和全部点中的其中一个点连边,流量为1,另外那个点和汇点连边,流量也为1.
如果点i和j之间都存在有向边,就连边i->j',流量也为1,这样建图就可以求解了.
但是这样建的图一定存在满流,于是还要对每个节点的流量进行限制,
所以对每个节点还要再拆点,也就是将每个点拆成4个,i拆成i0,i1,i2,i3.
其中i0,i1表示入度;i2,i3表示出度;连边i0->i1,i2->i3;
i->j'就改成连边i1->j2,流量为1.同时题目中还要求使得所有环的权值和最小,
所以套用费用流模型,i0->i1,i2->i3,s->i0,i3->t的费用都为0;
i1->j2为i到j在原图中的权值,对新建的网络跑一次最小费用流.
如果流量等于节点数就是存在解,最小费用就是最优解了.

[hdu3435]也是一样的,变成了无向图,连正反两条边就好了.

最小环覆盖(判断是否存在环)(每个点只经过一次,除了出发点和终止点)

一.如果几个点构成一个环的话，那么这每一个点的入度与出度都是为1的.

二.设一个源点0,汇点2*n+1,

源点连接每一个u,容量为1,费用为0;

汇点连接每一个v+n,容量也为1,费用为0;

从u到v建一条边,容量为1,费用为w;那么这就转换成了最小费用最大流的模板题,假设最后最大流为n,

那么说明恰好每个点都是出入度为1,即构成了环,从而得到题目做要求的.

[建图]拆点+费用流

S连i,流:1,费用:0

i+n连T流:1,费用:0

有关系的点a连b+n,流:1,费用:边权

***----------*/**

[网络流关键割边][ZOJ2532]

①第一类:增加哪些边的容量,可以使得最大流增大

分别对源点和汇点染色。一条边如果连着两个集合，说明是关键的割边

```
int col[N];
void dfs1(int u) { //S
    if(col[u]) return;
    col[u]=1;
    for(int i=p[u];i!=-1;i=e[i].nxt) {
        if(e[i].c>0) dfs1(e[i].b);
    }
}
void dfs2(int u) { //T
    if(col[u]) return;
    col[u]=2;
    for(int i=p[u];i!=-1;i=e[i].nxt) {
        if(e[i^1].c>0) dfs2(e[i].b);
    }
}
```

②第二类:减少哪些边的容量,可以使得最大流减少(无向图)

先跑一次最大流,然后对于残留网络中,流量大于0的边建边,跑tarjan();

然后对于原图的每条边(a,b),如果属于不同分量,那么该边为关键割边.

***----------*/**

[网络流输出割边集]

对源点染色(或者对汇点染色)，被染色的为源集合,其他的为汇集合.

```
int col[N];
void dfs1(int u) { //S
   if(col[u]) return;
   col[u]=1;
   for(int i=p[u];i!=-1;i=e[i].nxt) {
     if(e[i].c>0) dfs1(e[i].b);
```

```
    }
}
\*-------------------------------------------------------*/
```

[分层建图思想]

当不知道怎么建图的时候,可以将点拆成不同的形态,或者拆边.

如:按时间拆成,(1,1) (1,2)...分别表示第一个节点第一秒和第二秒状态的点.

```
\*-------------------------------------------------------*/
```

[费用相同的优先选一些点]

每个费用的值都乘*100(比点数多),后两位加上一个数作为优先选择.

```
\*-------------------------------------------------------*/
```

[最大费用优先(可以不满流)]

利用EK算法不断增广最长路,直到如果dis[t]<=0时,就break;

```
\*-------------------------------------------------------*/
```

[最大权闭合子图]

构造出收益和花费的状态. 源点连收益,汇点连花费.

求一次最小割以后,dfs(S),被标记的点即为我们选择的点达到最大效益.

```
\*-------------------------------------------------------*/
```

[左上到右下走多次的最大价值](一个点的价值取了,第二次取的价值为 0)

费用流拆边.

第一条边: 容量为1 , 费用为点的价值,

另一条边: 容量为inf , 费用为0.

这样求最长路时,肯定先走第一条边,然后走第二条边,保证每个点价值取一次.

```
\*-------------------------------------------------------*/
```

[选取区间问题][POJ3680],[POJ3762](工作安排)

给你好多个区间,选择一个区间可以获得一个利益Wi.

但是每个点最多只能被覆盖K次.问在限制条件下能获得的最大利益.

【解析】先将所有区间端点离散化到整数1..M , 另加源s=0 , 汇t=M+1;

对每个点i (0 <= i <= M) 加边(i, i+1, K, 0);

对每个区间(ai, bi)加边(ai', bi', 1, wi);

(其中ai', bi'分别表示ai, bi离散化后对应的数值)

求一次最大费用流即为结果。

```
\*-------------------------------------------------------*/
```

[按字典序输出割集]

无向图使得S-T不连通的割集.[POJ1815]

把一个点拆成两个点，由该点的入点向该点的出点连一条容量为1的边，其余的原来有边的要连正无穷的边...

先做一遍网络流，求出最小割，就是我们要的最少的点的个数

i 和 i+n

建立超级源点汇点S,T

【建图】:

S	连	s	流 inf	
t+n	连	T	流 inf	
s	连	s+n	流 inf	(不割s和t)
t	连	t+n	流 inf	(不割s和t)

有关系的: i+n 连 j 流 inf
其他点 : i 连 i+n 流 1 (只割这些点!!!)

PS:: 按字典序输出 , 只要从小到大枚举 每个点 是否是割点即可
即去掉点i->i+n 的边,最大流减少,说明 点i是一个割点,
去掉该点hash[i]=1;
不断修改最大流为割点后的最大流,
重新建图时去掉和前面已经选择出来的割点所连的边 e[i].c=0;

----------/

[最小点权覆盖][POJ3308][行列覆盖]

N行 , M列
每行Ri被覆盖的费用为ri , 每列Ci被覆盖的费用为 ci .
N*M的矩阵中有一些点,要你花费最少的费用,使得每个点都被覆盖.
【建图】
S连行i , 容量为 ri
T连列j+N , 容量为 ci
矩阵中每个点(x,y) : 行x 连 列y+N , 容量为inf .
最小割 = 最大流 = 最小点权覆盖.
最大独立集 = 总容量 - 最小点权覆盖

----------/

[项目最小花费问题][最小割]

[POJ3469] N个项目, 每个项目有2个模式,分别的费用为ai 和 bi.
并且有M对依赖关系: a b w ,表示项目a和b如果不用同一个模式,需要额外的w花费.
问怎样安排每个项目的模式,使得最小花费.
【建图】
S连模式1,容量为ai.
T连模式2,容量为bi.
a和b如果存在依赖关系: a连b,容量为w. (双向边)
S->T 求最小割,即为最小花费.

[ZOJ2930] N个项目, 每个项目有2个模式,分别的费用为ai 和 bi.
M对依赖关系 a b : a选了模式2,那么b也必须选模式2.
问怎样安排模式,使得最小花费.
【建图】
Wi = ai - bi
Wi>0 , S 连 i , 容量 Wi.
Wi<0 , i 连 T , 容量 -Wi.
每对依赖干系 a b : a 连 b , 容量为 inf.
tot记录选择模式1的费用,
sum记录选择正Wi的费用.
最后的结果即为: ans = tot - (sum-sap(S,T))

----------/

[无向图判断边割集是否唯一]

N个点, M条带权边。
先跑一边最大流, dfs1(S) , dfs2(T);

如果有点没被遍历到,说明不唯一,否则唯一.

【解析】一个点没被遍历到,说明和该点连的边都是割边,

S->v 和 v->T的两条边就是任选一个就是割边.

--/

[求有多少条边,减少 1 的流量,最大流减少?]RQNOJ 石油运输

无向图.

先跑一次最大流.

然后对于每条边(反向边也算),如果剩余流量不为0,则建一条有向边.

建图后,进行强连通缩点.

然后对于输入的每条边,判断两个端点是否在同一个联通分量.

如果在不同的联通分量,那么该边为割边.

--/

[输出边不相交的两条最短路径][SGU185]

分别从起点和终点做最短路, 加最短路上的边到网络流上.

跑一次起点到终点的最短路,

残留网络上,从起点沿着正向边流量为0 (i%2==0) 的路径一直到终点,标记掉该路径的所有边.再做相同操作.

[加最短路上的边,也可以这么加]: 只做源点的最短路.

```
a=road[i].a; b=road[i].b; c=road[i].c;
if(ds[a]+c==ds[b]) addedge(a,b,1);
if(ds[b]+c==ds[a]) addedge(b,a,1);
```

--/

[混合欧拉定向, 输出路径][UVA10735] Euler Circuit

建完混合欧拉图后,跑一次最大流.

残留网络上,边流大于0的即为无向边定向的方向.

然后把每条定向的无向边和原来输入的有向边建图加起.

跑一次欧拉回路即可.

--/

/*--*\

网络流的应用

--/

来回不重复最短路

无向图,从起点到终点,再从终点到起点.中间不能走重复的边,求最短路程.

相当于找两条从起点到终点不重复的路径.

[建图] S连起点,容量2

终点连T,容量2

i和j add(i,j,1,w) add(j,i,1,w) 容量1

--/

二分+网络流(挤奶,floyd+最大流+二分)

A.[题意]有c头奶牛和k台挤奶机,每台挤奶机最多为m头奶牛工作,挤奶机与奶牛间互相存在一定的距离.现在求一种方案,使所有的奶牛完成挤奶工作,并且使到挤奶机的距离最远的奶牛走的距离最小,输出此距离.

[建图]二分枚举答案+网络流

对答案ans进行二分,然后删掉牛到挤奶器距离大于ans的边,

距离小于ans的就连边,容量1

建立S连牛,容量 1

建立T连挤奶器,容量 m

判断 S->T 的最大流是否为 牛的数量

如果等于牛的数量,说明ans为一个可行解

B.[题意]有n块田地,已知每块田地上面牛的数量和雨篷能遮蔽的牛的数量;

有m路无向边连接任意两块田地，每条路有固定的长度.

问如果下雨了,所有的牛要怎么走,才能使得在最短的时间

(最后的牛进入雨篷)内让所有的牛进入雨篷,如果不能的话输出-1.

[建边]把一个点拆成i,i+n分别表示牛i,棚子i+n

S->i 牛的数量,i+n->T棚子容纳牛的数量

枚举答案ans,如果i牛到j棚距离小于ans,

就连一条边i->j+n 容量为inf,表示可以流无限的牛

如果i牛到j棚距离大于ans,就不连

然后判断最大流是否等于牛的数量

maxflow==cow,表示ans为一个可行解

不断枚举ans,直到找到最小的ans满足 maxflow==cow

***----------*/**

最小点权覆盖+输出割点(Destroying The Graph)

将点权转换成边权,拆点i,i+n 表示出和入

S连上所有出边,T连上所有入边,

根据最小点权覆盖,最小割就是答案.

这是个简单割,因为u-v不可能是割,以割必定是s-u,v-t中的一个,

那么按照覆盖路径的含义,每条边中至少有一个点的情况也满足了.

```
//THE code
   bool vis[N];
    void dfs(int u) {
       vis[u]=1;
       for(int i=p[u];i!=-1;i=e[i].nxt) {
          int v=e[i].b;
          if(!vis[v]&&e[i].c>0) dfs(v);
       }
    }
    int main()
    {
       int i;
       int a,b,c;
       while(scanf("%d%d",&n,&m)!=EOF){
          init();
          S=n+n+1; T=n+n+2;
          for(i=1;i<=n;i++) //入
             scanf("%d",&c),addedge(i+n,T,c);
```

```
        for(i=1;i<=n;i++) //出
           scanf("%d",&c),addedge(S,i,c);
        for(i=1;i<=m;i++) //有向边
           scanf("%d%d",&a,&b),addedge(a,b+n,inf);
        int ans=sap(S,T,T); //SAP模板
        //dfs求割集
        memset(vis,0,sizeof(vis));
        dfs(S);
        int ct=0;
        for(i=1;i<=n;i++) {
          if(!vis[i])  ct++;//与S不连通,割出边
           if(vis[i+n]) ct++;//与T不连通,割入边
        }
        printf("%d\n",ans);
        printf("%d\n",ct);
        for(i=1;i<=n;i++) {
           if(!vis[i])  printf("%d -\n",i);
           if(vis[i+n]) printf("%d +\n",i);
        }
      }
   }
\*-------------------------------------------------------*/
```

最大密度子图+输出子图

边数/点数 最大.输出子图中的点.

[建边]

```
   for(i=1;i<=m;i++) addedge(x[i],y[i],1.0,1.0);   (双向边)
   for(i=1;i<=n;i++) {
      addedge(S,i,m+0.0);
      addedge(i,T,m+2.0*mid-d[i]+0.0);
   }
```

[判断] f=(n*m-maxflow)/2.0

f>0 bot=mid;

f<0 top=mid;

[PS]:搜出答案后， 还要对 sap(bot) 流一次，防止答案满流

```
//THE code
struct Edge {
  int b,nxt
  double c,w;
}e[N*N];
int x[N],y[N];
int d[N];
bool hash[N];
int ans[N];
void addedge(int a,int b,double c,double cc=0) {
```

```
    e[cnt].b=b; e[cnt].c=c; e[cnt].nxt=p[a]; p[a]=cnt++;
    e[cnt].b=a; e[cnt].c=cc; e[cnt].nxt=p[b]; p[b]=cnt++;
  }
  void make_g(double mid) {
    int i;
    init();
    for(i=1;i<=m;i++) addedge(x[i],y[i],1.0,1.0);
    for(i=1;i<=n;i++) {
      addedge(S,i,m+0.0);
      addedge(i,T,m+2.0*mid-d[i]+0.0);
    }
  }
  void dfs(int u) {
    hash[u]=1;
    for(int i=p[u];i!=-1;i=e[i].nxt)
      if(e[i].c>0.0&&!hash[e[i].b]) dfs(e[i].b);
  }
  int main() {
    int i;
    while(scanf("%d%d",&n,&m)!=EOF) {
      if(m==0) {puts("1\n1");continue;}
      memset(d,0,sizeof(d));
      S=n+1,T=n+2;
      for(i=1;i<=m;i++) {
        scanf("%d%d",&x[i],&y[i]);
        d[x[i]]++; d[y[i]]++;
      }
      double top=m+0.0,bot=0.0,mid;
      while(top-bot>1e-5) {
        mid=(top+bot)/2.0;
        make_g(mid);
        double f=(m*n-sap(S,T,T)+0.0)/2.0;
        if(f>1e-10) bot=mid;
        else       top=mid;
      }
      make_g(bot); sap(S,T,T);
      dfs(S); int num=0;
      for(i=1;i<=n;i++)
        if(hash[i]) ans[++num]=i;
      printf("%d\n",num);
      for(i=1;i<=num;i++) printf("%d\n",ans[i]);
    }
  }
\*------------------------------------------------------*/
```

无向图全局最小割

拆点,一进一出i和i+n

枚举源汇点,求最大流,记录最小值

***--*/**

01 规划+输边割集

[题意]给出一个带权无向图,求割集,且割集的平均边权最小.

```
struct po {
  int a,b,c,id;
  double w;
  int nxt;
}e[N*N];
int p[N];
int cnt;
int ans[N*N],num;
double sap(int s, int t,double mid) {
   for(i=0;i<cnt;i++) {
      e[i].w=e[i].c-mid;
      if(e[i].w<0) maxflow+=e[i].w;
   }
   maxflow/=2.0;
  //......
}
bool vis[N];
void dfs(int u) {
   vis[u]=1;
   for(int i=p[u];i!=-1;i=e[i].nxt)
      if(e[i].w>ep&&!vis[e[i].b]) dfs(e[i].b);
}
void find() {
   double top=1000000.0,bot=0,mid;
   while(top-bot>=ep){
      mid=(top+bot)/2.0;
      double ans=sap(1,n,mid);
      if(ans<0) top=mid;
      else     bot=mid;
   }
   memset(vis,0,sizeof(vis));
   dfs(1);
   num=0;
   memset(ans,0,sizeof(ans));
   for(int i=1;i<=n;i++)
      for(int j=p[i];j!=-1;j=e[j].nxt)
         if((vis[i]&&vis[e[j].b]==0)||e[j].w<0)
            ans[e[j].id]=1;
```

```
}
int main() {
   int i,j,t,a,b,c;
   while(scanf("%d%d",&n,&m)!=EOF) {
      init();
      for(i=1;i<=m;i++) {
         scanf("%d%d%d",&a,&b,&c);
         addedge(a,b,c,i);
         addedge(b,a,c,i);
      }
      find();
      for(i=1;i<=m;i++)
         if(ans[i]) ans[++num]=i;
      printf("%d\n",num);
      for(i=1;i<=num;i++) printf(i==num?"%d\n":"%d ",ans[i]);
   }
}
\*-------------------------------------------------------*/
```

求最小点割集(最大点权独立集)

割掉最少的点(权值总和),使S和T分割.

[题意]有n个城市和m条道路(双向),一伙thieves准备从S城出发到H城盗窃,
police为了将这伙thieves抓住,需要在这n个城市中的每一个城市安排
一定数量的police(已知条件).但police不希望在S城或H城邂逅thieves.
求总共需要的最少police数.

[考察点]拆点+最大流

[思路]把城市作为点拆开(把城市i拆成cap数组中的cap[i][i+n].
其中i作为城市i的入,i+n作为城市i的出),原来的街道仍然作为边,
但是边权是无穷大.只要注意到BFS的时候 起点/终点是不需要拆的.
(在cap数组中就直接把起点/终点的入和出之间的边权赋值为0,
使得BFS的时候不走起点/终点,实际上就相当于没有把起点/终点拆开.
程序中的处理方法就是 cap[s][s+n]=0;cap[e][e+n]=0;)

[建图] 拆成i和i+n(本身真身连替身,不同的两个点.替身连真身)
对于本身 i 连 i+n 流 i 的权值
对于其他 i+n 连 j 流 INF
最后 求 S+n 到 T 的流

```
\*-------------------------------------------------------*/
```

数和思想(行进列出)

行进列出,正好一流

```
\*-------------------------------------------------------*/


/*************************************************************\
```

组合博弈

在子状态中找最小不能取到的数为SG值.

[本质]

A.必胜点1至少可以有一步方法进入必败点0

(儿子状态中有0，可以进入必败点)

B.必败点0只能进入必胜点1

(儿子状态中都只有1，即只能进入必胜点)

博弈类型:

巴什博弈 ， 威佐夫博奕 ， 尼姆博弈(SG函数) ， 反NIM博弈

扩展NIMk博弈(n堆石子,每次从最多k堆中取若干石子)

不平等博弈,棋盘博弈,阶梯博弈,二分图与博弈

翻硬币游戏,K倍动态减法游戏,Every-Game

有向无环图博弈,砍树博弈,无向图删边博弈

Nim积

**********/

模板

==========

SG 函数

```
//每次只能取s[i]个,这里为fibo数列做例子
int sg[10005],s[20],n,m,p;
int getsg(int x) {
   if(sg[x]!=-1) return sg[x];
    bool hash[20]={0};
    for(int i=1;i<=16;i++) {
      if(x-s[i]<0) break;
      if(sg[x-s[i]]==-1) sg[x-s[i]]=getsg(x-s[i]);
      hash[sg[x-s[i]]]=1;
   }
    for(int i=0;;i++)
      if(!hash[i]) return sg[x]=i;
}
```

==========

威佐夫博奕

```
//有两堆各若干个物品,两个人轮流从某一堆或同时从两堆中取同样多的物品,
//规定每次至少取一个,多者不限,最后取光者得胜.
//(ak,bk)(ak ≤ bk ,k=0,1,2,...,n)表示奇异局势
//求法:
//ak =[k(1+√5)/2], bk= ak + k (k=0,1,2,...,n 方括号表示取整函数)
```

[判断]

Gold=(1+sqrt(5.0))/2.0;

1)假设(a,b)为第k种奇异局势(k=0,1,2...) 那么k=b-a;

2)判断其a==(int)(k*Gold),相等则为奇异局势

[注意]

采用适当的方法,可以将非奇异局势变为奇异局势.

假设面对的局势是(a,b)

若b=a,则同时从两堆中取走 a 个物体,就变为了奇异局势(0,0);

1.如果a=ak,

1.1 b>bk, 那么取走b - bk个物体,即变为奇异局势(ak, bk);

1.2 b<bk,则同时从两堆中拿走ak - a[b - ak]个物体,

变为奇异局势(a[b - ak],a[b - ak]+b-ak);

2.如果a=bk,

2.1 b>ak,则从第二堆中拿走多余的数量b - ak

2.2 b<ak,则:若b=aj(j<k)从第一堆中拿走多余的数量a - bj;(a>bj)

若b=bj(j<k)从第一堆中拿走多余的数量a - aj;(a>aj)

```
//THE code
int main() {
   int t,m,k,a,b;
   while(scanf("%d%d",&a,&b)!=EOF) {
      if(a>b) swap(a,b);
      k=b-a;
      m=k*(1+sqrt(5.0))/2;
      if(m==a) puts("0");
      else    puts("1");
   }
   return 0;
}
```

==========

不平等博弈

[题意]有N堆方块,方块分为有黑白两色,W选手只能取白色方块,B选手只能取黑色方块.

如:一堆方块从下到上分别是 WBWB,那么如果 W 选手取了最下面的 W,

同时会去掉 W 上面的所有方块.

[解题]

下面我们来考虑初始局面 G 有 n 座塔的情况.

不妨设局面 G 由 n 座塔 T1,T2...Tn 组成,且与 T1,T2...Tn 相对应的

surreal number 为 x1,x2...xn.

由 2.2 节的定理 8 可知,与 G 相对应的 surreal number x=x1+x2...+xn.

题目要求我们判断子局面 C1 是否不差于子局面 C2,也就是要求我们判断是

否对于任意的塔 H,当 C2+H 大于 0 时,C1+H 也大于 0.

实际上这就等价于要求我们判断 C1 是否大于等于 C2.

[surreal number]

只要计算以 W 为基准的 surreal number x

x>0 , 局面为 W 态,即不管 W 是先手后手,都必胜.

x<0 , 局面为 B 态,即不管 B 是选手后手,都必胜.

x=0 , 后手赢.

//THE code

```
int sz[N];
```

```
char T[N][N];
//计算W态的surreal number x
//x1+x2..+xn>0,局面为W态.
//x1+x2..+xn<0,局面为B态.
//x1+x2..+xn=0,后手赢.
double surreal(char T[],int n) {
    double x=0,k=0.5;
    int i=1;
    while(i<=n&&T[i]==T[1]) {
        if(T[i]=='W') x+=1.0; //最下面都是白色 n
        else x-=1.0; //最下面都是黑色 -n
        i++;
    }
    while(i<=n) { // n-1/2-1/4+1/8-1/16
        if(T[i]=='W') x+=k; //白色,+1/(2^k)
        else x-=k; //黑色,-1/(2^k)
        i++; k*=0.5;
    }
    return x;
}
int main() {
    int i,j; char s[55];
    double x1=0.0,x2=0.0;
    for(i=1;i<=3;i++) scanf("%d",&sz[i]);
    for(i=1;i<=3;i++) {
        for(j=1;j<=sz[i];j++) {
            scanf("%s",s);
            T[i][j]=s[0];
        }
        x1+=surreal(T[i],sz[i]);
    }
    for(i=1;i<=3;i++) scanf("%d",&sz[i]);
    for(i=1;i<=3;i++) {
        for(j=1;j<=sz[i];j++) {
            scanf("%s",s);
            T[i][j]=s[0];
        }
        x2+=surreal(T[i],sz[i]);
    }
    if(x1>=x2) puts("Yes");
    else puts("No");
}
```

==========

K 倍动态减法游戏

```
int n,k;
int f[N],id[N],top;
//int getsg(int n) { //单调队列O(n)
//  int i;
//  f[0]=0; id[0]=0;
//  f[top=1]=1; id[top]=1;
//  for(i=2;i<=n;i++) {
//      while(top&&f[top]<=(i-id[top])*k) top--;
//      f[top+1]=i-id[top];
//      id[++top]=i;
//  }
//  return f[top];
//}
int getsg(int n) { //标记所有必败点
    int i,j;
    f[1]=1; f[2]=2;
    for(i=2,j=1;f[i]<n;i++) {
        while(f[j]*k<f[i]) j++;
        f[i+1]=f[i]+f[j];
        printf("%d\n",f[i]);
    }
    while(f[i]>n) i--;
    while(f[i]!=n) {
        n-=f[i];
        while(f[i]>n) i--;
    }
    return f[i];
}
int main() {
    while(scanf("%d%d",&n,&k)!=EOF) {
        int nim=getsg(n);
        if(nim==n) printf("lose\n");
        else      printf("%d\n",nim);
    }
}
```

==========

Nim 积

Turning Corners:

[游戏规则]

三条基本规则

[初始条件]一组成矩阵摆放的硬币

[行动]每次选四枚硬币翻转,必须是四个角落(a,b)(a,y)(x,b)(x,y)

(0<=a<x,0<=b<y)且(x,y)是正面

[引入概念]Nim Multiplication.
SG(x,y)=x⊙y
[定理]
1.x⊙(y⊙z)=(x⊙y)⊙z
2.(x?y)⊙z=(x⊙z)?(y⊙z)
3.X⊙y=X*y
4.X⊙X=X*3/2
(大写的X为2^(2^n),如2,4,16,256,65536...)

```
//THE code
```

运算规则:类似四则运算的加和乘.
Nim积的性质: (+表示Nim和运算,x表示Nim积运算)
1.X x 2^(2^a) = 2^(2^a) * X
2.X x Y < 2^(2^a)
3.2^(2^a) x 2^(2^a) = (3/2) * 2^(2^a)
如果慢的话,可以先预处理出小数据的Nim积表sg[][];

[PS]如果是多维的话, X x Y x Z...
因为Nim积满足结合律,
只要先计算出 G(X,Y)的Nim积,然后再求 G(X,Y)与Z的Nim积即可.
即: nim_multi(nim_multi(x,y) , z)

```
const int pw[]={2,4,16,256,65536}; //2^(2^a)
int n,m;
int nim_multi_power(int x,int y) { //确保x是一个2的幂
    if(x<2) return (x==1)&&(y==1); //查表
    int a=0; //找到 2^(2^a) <= x < 2^(2^(a+1))
    while(pw[a+1]<=x) a++;
    int M=pw[a]; //M=2^(2^a);
    int p=x/M; //x=p*M;
    int s=y/M,t=y-s*M; //y=s*M+t;
    int d1=nim_multi_power(p,s);
    int d2=nim_multi_power(p,t);
    return (d1^d2)*M^nim_multi_power(M/2,d1);
}
int nim_multi(int x,int y) {
    if(x<y) return nim_multi(y,x);
    if(x<2) return (x==1)&&(y==1); //查表
    int a=0; //找到 2^(2^a) <= x < 2^(2^(a+1))
    while(pw[a+1]<=x) a++;
    int M=pw[a]; //M=2^(2^a);
    int p=x/M,q=x-p*M; //x=p*M+q;
    int s=y/M,t=y-s*M; //y=s*M+t;
    int c1=nim_multi(p,s);
    int c2=nim_multi(p,t)^nim_multi(q,s);
    int c3=nim_multi(q,t);
```

```
        return (c1^c2)*M^c3^nim_multi_power(M/2,c1);
    }
    int main() {
        int i,j,x,y;
        while(scanf("%d",&n)!=EOF) {
            int nim=0;
            for(i=1;i<=n;i++) {
                scanf("%d%d",&x,&y);
                nim^=nim_multi(x,y);
            }
            if(nim) puts("Have a try, lxhgww.");
            else puts("Don't waste your time.");
        }
    }
\*--------------------------------------------------------*/

/*--------------------------------------------------------*\
```

博弈的应用

```
\*--------------------------------------------------------*/
```

二维博弈(体现 SG 的本质)

[题意]放正方形,不能能放就结束

0000

1001

1000

0000

0 能放,1 不能放

[解析]在子状态中找有无必败状态

必胜点:至少有一种方法进入必败点.

必败点:只能进入必胜点.

```
//THE code
    bool dfs() {
        int i,j;
        for(i=0;i<n-1;i++)
            for(j=0;j<m-1;j++)
                if(map[i][j]=='0'&&map[i][j+1]=='0'&&
                    map[i+1][j]=='0'&&map[i+1][j+1]=='0') {
                    map[i][j]=map[i][j+1]=map[i+1][j]=map[i+1][j+1]='1';
                    bool sg=dfs();
                    map[i][j]=map[i][j+1]=map[i+1][j]=map[i+1][j+1]='0';
                    if(!sg) return 1;
            }
       return 0;
    }
```

--/

SG 状态(字符串)

一般我们都用一个数字x表示剩下还有几个石子,表示状态.

这题用字符串表示一个状态!!!

一个状态从子状态求的SG函数.

--/

有向图 SG 函数

[题意]游戏描述是你在一个有向图上有N棋子,你能将棋子进行移动,到棋子都移动到出度为0的顶点时就不能再移动,此时不能再移动的player就算输.

有向图SG,出度为0的为必败点.看完 getsg 函数 就有启发了

```
int getsg(int x) {
    if(sg[x]!=-1) return sg[x];
    bool hash[1005]={0};
    for(int i=p[x];i!=-1;i=e[i].nxt)
        hash[getsg(e[i].b)]=1;
    for(int i=0;;i++)
        if(!hash[i]) return sg[x]=i;
}
```

--/

[砍树博弈]

一棵树 dfs 等价转换成一堆石子后,石子数可能为 0

多颗树同理...

```
//nim^=dfs(root,root)
int dfs(int u,int f) { //从根节点开始搜
    int nim=0;
    for(int i=p[u];i!=-1;i=e[i].nxt) {
        int v=e[i].b;
        if(v==f) continue;
        nim^=(dfs(v,u)+1);
    }
    return nim;
}
```

--/

[无向图删边博弈]

无向图,每次可以删除一条和root相连的边,并且去掉和root不相连的部分.

不能删了,就算输了.

[解析]先用双连通缩点,然后就变成一棵树的删边游戏了.

①若分量中边的条数为奇数,则该分量缩成一条新边.

②所分量中边的条数为偶数,则该分量缩成一个点.

③把桥边加回去.

--/

[威佐夫博弈输出方案]

```
a=k*gold , b=a+k
k=a/gold+1; k=b/(gold+1)+1;
```

--/

[反 Nim 博弈]

最后不能走的赢.

对于任意一个Anti-SG游戏，如果我们规定当局面中所有的单一游戏的SG值为0时，游戏结束，则先手必胜当且仅当：

1、游戏的SG函数不为0且游戏中某个单一游戏的SG函数大于1；

2、游戏的SG函数为0且游戏中没有单一游戏的SG函数大于1。

--/

[扩展 NIMk 博弈]

n 堆石子,每次从最多 k 堆中取若干石子.

[NP分析] K=1，为Nim问题.

对于K>1的情况,我们令把P1~Pn这n个数,转成二进制,

然后每位分别相加,每位最后结果mod(K+1)即可.

如果每一位结果都是0,则为P局面,否则是N局面

[取石子方法]

设P1P2P3 … Pn为n堆石子数目.Pi已标记的石子堆,

D0[i]和D1[i]分别表示所有已标记的石子堆中第i位为0和1的总数.

1.找出加法结果非0的最高位,设为W.

2.找出一个二进制第W位为1,而且未标记的石子堆Pi,将Pi标记,

并把它的第W位由1改为0.

对于Pi的第1到(W-1)位,逐个判断,第j位如果为0则Inc(D0[j]),

否则Inc(D1[j]).

3.若更新后的S某一位非0(即S[i]≠0),且S[i]+D0[i]>K,

或S[i]-D1[i]<1,

可以通过修改以前已标记的石子堆将S[i]修正为0.

4.如果加法结果已经全部为0,则确认所有已经做的更改,并结束;否则转到1.

--/

[Every-Game]

N个单一的游戏,决策者要对所有还没结束的单一游戏都进行操作. 无路可走输.

[分析]游戏者需要做的:

就是让自己可以取得胜利的游戏尽可能长的玩下去.

让自己不能取得胜利的游戏尽可能短的玩下去.

[解决]

对于SG值为0的点,我们需要知道最少几步能将游戏带入终止状态;

对于SG值不为0的点,我们需要知道最多几步游戏会被带入终止状态.

以上两个值，我们都用step来表示

```
            0    ,v为终结点
step[u] = max(step[v])+1    ,u必胜,v必败.
          min(step[v])+1    ,u必败,v必胜.
```

[定理]对于 Every-SG 游戏先手必胜当且仅当单一游戏中最大的step为奇数.

(因为必胜状态step为奇数,必败状态step为偶数)

```
//THE CODE
int step[N][N]; //步数
char sg[N][N];
```

```
bool dfs(int x,int y) {
    if(sg[x][y]!=2) return sg[x][y];
    if(x==0) {
        sg[x][y]=step[x][y]=0;
        return sg[x][y];
    }
    int mx=-inf,mi=inf;
    for(int i=1;i*x<=y;i++) {
        int xx=min(x,y-i*x),yy=max(x,y-i*x);
        if(dfs(xx,yy)) mi=min(mi,step[xx][yy]);
        else mx=max(mx,step[xx][yy]);
    }
    if(mx!=-inf) step[x][y]=mx+1,sg[x][y]=1;
    else step[x][y]=mi+1,sg[x][y]=0;
    return sg[x][y];
}
int main() {
    int i,j,t,cas=0;
    int a,b;
    memset(sg,2,sizeof(sg));
    for(i=0;i<=1000;i++)
        for(j=i;j<=1000;j++) sg[i][j]=dfs(i,j);
    while(scanf("%d",&n)!=EOF) {
        int ret=0;
        for(i=1;i<=n;i++) {
            scanf("%d%d",&a,&b);
            if(a>b) swap(a,b);
            ret=max(ret,step[a][b]);
        }
        if(ret&1) puts("MM");
        else puts("GG");
    }
}
```

***--*/**

[棋盘博弈]

往上走和往左走,相当于从一堆中取若干石子.

往左上走,相当于桶两堆中取相同个石子.

即最终转为威佐夫博弈.

***--*/**

[阶梯博弈]

首先是对阶梯博弈的阐述...博弈在一列阶梯上进行...

每个阶梯上放着自然数个点..两个人进行阶梯博弈...

每一步则是将一个集体上的若干个点(>=1)移到前面去..

最后没有点可以移动的人输..

如这就是一个阶梯博弈的初始状态2 1 3 2 4,只能把后面的点往前面放,

如何来分析这个问题呢? 其实阶梯博弈经过转换可以变为Nim.

把所有奇数阶梯看成N堆石子,做nim.

把石子从奇数堆移动到偶数堆可以理解为拿走石子,相当于几个奇数堆石子在做Nim.

(如所给样例..2^3^4=5 不为零所以先手必败)

为什么可以这样来转化?

假设我们是先手,所给的阶梯石子状态的奇数堆做Nim先手能必胜.

我就按照能赢的步骤将奇数堆的石子移动到偶数堆.

如果对手也是移动奇数堆.,我们继续移动奇数堆.

如果对手将偶数堆的石子移动到了奇数堆.

那么我们紧接着将对手所移动的这么多石子从那个偶数堆移动到下面的奇数堆.

两次操作后,相当于偶数堆的石子向下移动了几个,而奇数堆依然是原来的样子,即为必胜的状态.

就算后手一直在移动偶数堆的石子到奇数堆,我们就一直跟着他将石子继续往下移,保持奇数堆不变.

如此做下去,我可以跟着后手把偶数堆的石子移动到0,然后你就不能移动这些石子了.

所以整个过程,将偶数堆移动到奇数堆不会影响奇数堆做Nim博弈的过程.

整个过程可以抽象为奇数堆的Nim博弈.

其他的情况,先手必输的,类似推理,只要判断奇数堆做Nim博弈的情况即可.

为什么是只对奇数堆做Nim就可以,而不是偶数堆呢?

因为如果是对偶数堆做Nim,对手移动奇数堆的石子到偶数堆,我们跟着移动这些石子到下一个奇数堆.

那么最后是对手把这些石子移动到了0,我们不能继续跟着移动,就只能去破坏原有的Nim而导致胜负关系的不确定.

所以只要对奇数堆做Nim判断即可知道胜负情况.

[POJ1704]

我们把棋子按位置升序排列后，从后往前把他们两两绑定成一对。如果总个数是奇数，就把最前面一个和边界（位置为0）绑定。

在同一对棋子中，如果对手移动前一个，你总能对后一个移动相同的步数，

故一对棋子的前一个和前一对棋子的后一个之间有多少个空位对最终结果没有影响

于是只需考虑同一对的两个棋子之间有多少空位。这样一来就成了N堆取石子游戏了.

[HDU4315]

考虑King的情况和上述版本几乎一致，只要把King当作普通人一样处理即可。

除了两种特殊情况：

1．当King是第一个人时，Alice直接胜

2．当King是第二个人且一共有奇数个人时，第一堆的大小需要减1。

因为如果king在奇数石子上,那么king前面的那一对石子k1,k2.

当对方把k1移到top时,我可以把k2移到top前的一个位置.

那么对于k2石子,对方如果碰了它,那么我肯定会把king移到top的.

所以k2也相当于是一颗废子而已,不影响最终Nim的结果.

***----------*/**

[翻硬币游戏]

一般的翻硬币游戏的规则是这样的：N 枚硬币排成一排，有的正面朝上，有的反面朝上。我们从左开始对硬币按1 到N 编号.

第一，游戏者根据某些约束翻硬币，但他所翻动的硬币中，最右边那个硬币的必须是从正面翻到反面。例如，只能翻3个硬币的情况，那么第三个硬币必须是从正面翻到反面。如果局面是正正反，那就不能翻硬币了，因为第三个是反的。

第二，谁不能翻谁输。

有这样的结论：局面的SG 值为局面中每个正面朝上的棋子单一存在时的SG 值的异或和。即一个有k个硬币朝上，朝上硬币位置分布在的翻硬币游戏中，SG值是等于k个独立的开始时只有一个硬币朝上的翻硬币游戏的SG值异或和。比如THHTTH这个游戏中，2号、3号、6号位是朝上的，它等价于TH、TTH、TTTTTH三个游戏和，即sg[THHTTH]=sg[TH]^sg[TTH]^sg[TTTTTH].我们的重点就可以放在单个硬币朝上时的SG值的求法。

约束条件一：每次只能翻一个硬币。

一般规则中，所翻硬币的最右边必须是从正面翻到反面，因为这题是只能翻一个硬币，那么这个硬币就是最右边的硬币，所以，每次操作是挑选一个正面的硬币翻成背面。

对于任意一个正面的硬币，SG值为1。

有奇数个正面硬币，局面的SG值==1，先手必胜，有偶数个正面硬币，局面的SG值==0，先手必败。

约束条件二：每次能翻转一个或两个硬币。(不用连续)

每个硬币的SG值为它的编号，初始编号为0，与NIM游戏是一样的。

如果对于一个局面，把正面硬币的SG值异或起来不等于0，既a1^a2^a3^…^an==x，对于an来说一定有an'=an^x<an。

如果an'==0，意思就是说，把an这个值从式子中去掉就可以了。对应游戏，就是把编号为an的正面硬币翻成背面就可以了。因为an^x==0，而a1^a2^a3^…^an==x，即an^a1^a2^a3^…^an==0，即a1^a2^a3^…^an-1==0，只要在原来的x里面去掉an就可以了。

如果an'!=0，意思就是说，把an这个值从式子中去掉后再在式子中加上an'，an'<an。对应游戏，去掉an就是把编号为an的正面硬币翻成背面，加上an'，如果编号为an'的硬币是正面，我们就把它翻成背面，是背面就翻成正面，总之，就是翻转编号为an'的硬币。因为an^x!=0，所以an^a1^a2^a3^…^an!=0，即a1^a2^a3^…^an-1!=0，而这里的an'=a1^a2^a3^…^an-1，所以在x中去掉an后，要对an'进行异或，也就是翻转，正转反，反转正。

约束条件三：每次必须连续翻转 k 个硬币。

我们以k==3为例。

我们计算的是个数为N的硬币中，其中最后一个硬币为正面朝上，的sg值。

N=1时，硬币为:正，先手必输，所以sg[1]=0。

N=2时，硬币为:反正，先手必输，所以sg[2]=0。

N=3时，硬币为:反反正，先手必胜，所以sg[3]=1。

N=4时，硬币为:反反反正，先手操作后为:反正正反，子状态局面SG=0^1=1，则sg[4]=0。

N=5时，硬币为:反反反反正，操作后:反反正正反，子状态局面的SG=1^0=1，则sg[5]=0。

N=6时，硬币为:反反反反反正，操作后:反反反正正反，子局面SG=0^0=0，则sg[6]=1。

根据观察，可以知道，从编号为1开始，sg值为：001 001 001 001……

根据观察，可以知道，sg的形式为000…01 000…01，其中一小段0的个数为k-1。

约束条件4：每次翻动一个硬币后，必须翻动其左侧最近三个硬币中的一个，即翻动第x个硬币后，必须选择x-1，x-2，x-3中的其中一个硬币进行翻动，除非x是小于等于3的。

（Subtraction Games）

当N==1时，硬币为：正，先手必赢，所以sg[1]=1。
当N==2时，硬币为：反正，先手必赢，因为先手可以翻成反反或正反，可能性为2，所以sg[2]==2。
当N==3时，硬币为：反反正，先手操作后可以为：反正
位置x：1 2 3 4 5 6 7 8 9 10 11 12 13 14...
sg[x]： 1 2 3 0 1 2 3 0 1 2 3 0 1 2…
这个与每次最多只能取3个石子的取石子游戏的SG分布一样，同样还有相似的这类游戏，约束条件5也是一样。
约束条件5：每次必须翻动两个硬币，而且这两个硬币的距离要在可行集S={1,2,3}中，硬币序号从0开始。(Twins游戏)
当N==1时，硬币为：正，先手必输，所以sg[0]=0。
当N==2时，硬币为：反正，先手必赢，所以sg[1]=1。
当N==3时，硬币为：反反正，先手必赢，所以sg[2]=2。
当N==4时，硬币为：反反反正，先手必赢，所以sg[3]=3。
当N==5时，硬币为：反反反反正，先手必输，所以sg[4]=0。
位置x：0 1 2 3 4 5 6 7 8 9 10 11 12 13 14...
sg[x]： 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2…
约束条件6：每次可以翻动一个、二个或三个硬币。（Mock Turtles游戏）
初始编号从0开始。
当N==1时，硬币为：正，先手必胜，所以sg[0]=1.
当N==2时，硬币为：反正，先手必赢，先手操作后可能为：反反或正反，方案数为2，所以sg[1]=2。
当N==3时，硬币为：反反正，先手必赢，先手操作后可能为：反反反、反正反、正反正、正正反，方案数为4，所以sg[2]=4。
位置x： 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14...
sg[x]：1 2 4 7 8 11 13 14 16 19 21 22 25 26 28…
看上去sg值为2x或者2x+1。我们称一个非负整数为odious，当且仅当该数的二进制形式的1出现的次数是奇数，否则称作evil。所以1，2，4，7是odious因为它们的二进制形式是1,10,100,111.而0,3,5,6是evil，因为它们的二进制形式是0,11,101,110。而上面那个表中，貌似sg值都是odious数。所以当2x为odious时，sg值是2x，当2x是evil时，sg值是2x+1.
这样怎么证明呢？我们会发现发现，
evil^evil=odious^odious=evil
evil^odious=odious^evil=odious
假设刚才的假说是成立的，我们想证明下一个sg值为下一个odious数。注意到我们总能够在第x位置翻转硬币到达sg为0的情况；通过翻转第x位置的硬币和两个其它硬币，我们可以移动到所有较小的evil数，因为每个非零的evil数都可以由两个odious数异或得到；但是我们不能移动到下一个odious数，因为任何两个odious数的异或都是evil数。
假设在一个Mock Turtles游戏中的首正硬币位置x1,x2,…,xn是个P局面，即sg[x1]^…^sg[xn]=0.那么无可置疑的是n必定是偶数，因为奇数个odious数的异或是odious数，不可能等于0。而由上面可知sg[x]是2x或者2x+1，sg[x]又是偶数个，那么x1^x2^…^xn=0。相反，如果x1^x2^…^xn=0且n是偶数，那么sg[x1]^…^sg[xn]=0。这个如果不太理解的话，我们可以先这么看下。2x在二进制当中相当于把x全部左移一位，然后补零，比如说2的二进制是10，那么4的二进制就是100。而2x+1在二进制当中相当于

把x全部左移一位，然后补1，比如说2的二进制是10，5的二进制是101。现在看下sg[x1]^…^sg[xn]=0，因为sg[x]是2x或者2x+1，所以式子中的2x+1必须是偶数个（因为2x的最后一位都是0，2x+1的最后一位都是1，要最后异或为0，2x+1必须出现偶数次）。实际上的情况可能是这样的：MT游戏当中的P局面是拥有偶数堆石子的Nim游戏的P局面。

约束条件 7：每次可以连续翻动任意个硬币，至少翻一个。（Ruler 游戏）

初始编号从1开始。

那么这个游戏的SG函数是g(n)=mex{0,g(n-1),g(n-1)^g(n-2),…,g(n-1)^…^g(1)}

根据SG函数可以得到SG值表如下。

位置x：1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16...

g(x)：1 2 1 4 1 2 1 8 1 2 1 4 1 2 1 16…

所以sg值为x的因数当中2的能达到的最大次幂。比如14=2*7，最大1次幂，即2；16=2*2*2*2，最大4次幂，即16。

这个游戏成为尺子游戏是因为SG函数很像尺子上的刻度。

约束条件8：每次必须翻转4个对称的硬币，最左与最右的硬币都必须是从正翻到反。（开始的时候两端都是正面）（Grunt游戏）

这是Grundy游戏的变种，初始编号从0开始。

当首正硬币位置为0,1,2时是terminal局面，即 终结局面，sg值都是0。当首正硬币位置n大于等于3的时候的局面可以通过翻0,x,n-x,n四个位置得到(其中x<n/2可保证胜利)。

这就像是把一堆石子分成两堆不同大小石子的游戏，也就是Grundy游戏。

***----------*/**

[寻找"平衡态"]

即对称态(如:两堆相同的石子,那么对方怎么取,我就从另一堆怎么取)

对称化分析: 相当于两堆相同的石子,对方在一堆怎么取,

我只要再将其去成对称即可,这样我就处于不败之地了.

***----------*/**

/**********

极大极小过程

二进制(三进制)状态压缩

a-b 剪枝

************/**

模板

==========

例题:给出一个 n*n 的棋盘(n<=8).

0和1两个玩家轮流操作，0先

0玩家在棋盘的空位上放置0，1玩家放置1

当棋盘放满时查询两个玩家最大连通块中棋子的个数

玩家得分为比对方多的棋子的个数

[分析]对于搜索中的节点,可以通过递归得到它所有子节点的评估分数值，

评估分数转移的规则：

0玩家一定会选择分数最高的子节点走(Max 局面)

1玩家一定会选择分数最低的子节点走(Min 局面)

```
//state 可以放的位置为1; who 轮到谁; now:摆放的状态;
//pos[]:记录选手放的位置;
//check()为检查最后局面的分值;
int maxmin(int state,int who,int now,int alpha,int beta,int dep) {
    if(hash[now]!=inf) return hash[now];
    if(state==0) {
        for(int i=0;i<dep;i++) {
            int x=pt[pos[i]].x,y=pt[pos[i]].y;
            g[x][y]=(i&1)+'0';
        }
        int num=check();
        return hash[now]=num;
    }
    int maxval=-inf,minval=inf,val;
    for(int st=state;st;st-=st&(-st)) {
        int i=st&(-st),x=rpw2[i];
        pos[dep]=x;
        val=maxmin(state-i,who^1,now+pw3[x]*(who+1),maxval,minval,dep+1);
        if(who==0&&val>beta) return val;
        if(who==1&&val<alpha) return val;
        maxval=max(maxval,val);
        minval=min(minval,val);
    }
    if(who==0) return hash[now]=maxval;
    else return hash[now]=minval;
}
\*------------------------------------------------------*/

/*------------------------------------------------------*\
```

极大极小过程的应用

```
\*------------------------------------------------------*/
```

[三子棋 ，四子棋]

```
//    下面以四子棋为例
//    X子先走,给定一个局面.
//    问当前下棋的玩家.
//        WIN :能否赢
//        LOSE:能否输
//        TIE :能否打平局
int pw2[25],pw3[25]; map<int,int>rpw2;
struct point { int x,y; }pt[N];
int n,m,op;
char g[5][5];
bool check(int x,int y) { //放了(x,y)是否赢了
    int i;
    for(i=0;i<4&&g[i][y]==g[x][y];i++); //每行
    if(i==4) return 1;
    for(i=0;i<4&&g[x][i]==g[x][y];i++); //每列
    if(i==4) return 1;
    if(x==y) { //对角线
        for(i=0;i<4&&g[i][i]==g[x][y];i++);
        if(i==4) return 1;
    }
    if(x+y==3) { //斜对角线
        for(i=0;i<4&&g[i][3-i]==g[x][y];i++);
        if(i==4) return 1;
    }
    return 0;
}
int hash[1<<16];
int maxmin(int state,int who,int now,int alpha,int beta) {
    if(hash[now]!=inf) return hash[now];
    if(state==0) return hash[now]=0;
    int maxval=-inf,minval=inf,val;
    for(int st=state;st;st-=st&(-st)) {
        int i=st&(-st),x=rpw2[i];
        g[pt[x].x][pt[x].y]=(who==0?'x':'o'); //标记走法
        if(check(pt[x].x,pt[x].y))val=who?-1:1;//检查是否赢了
        else val=maxmin(state-i,who^1,now+pw3[x]*(who+1),maxval,minval);
        g[pt[x].x][pt[x].y]='.'; //回溯
        if(op<=1) {
            maxval=max(maxval,val);
            minval=min(minval,val);
        }else { //能平则平
            if(who==0) { if(val==0) return hash[now]=0; }
            else { if(val!=0) return hash[now]=-1; }
```

```
        }
    }
    if(op==0) { //先手要赢
        if(who==0) return hash[now]=maxval;
        else return hash[now]=minval;
    }else if(op==1) { //先手要输
        if(who==0) return hash[now]=minval;
        else return hash[now]=maxval;
    }else { //先手要平局
        if(who==0) return hash[now]=-1;
        else return hash[now]=0;
    }
}
void init() {
    pw2[0]=pw3[0]=1; rpw2.clear();
    for(int i=0;i<=20;i++) pw2[i]=(1<<i),rpw2[1<<i]=i;
    for(int i=1;i<=12;i++) pw3[i]=pw3[i-1]*3;
}
int main() {
    int i,j; char s[22];
    init();
    while(scanf("%s",s)!=EOF) { //WIN,LOSE,TIE
        m=0; n=4;
        for(i=0;i<n;i++) {
            scanf("%s",g[i]);
            for(j=0;j<n;j++) {
                if(g[i][j]=='.') pt[m].x=i,pt[m++].y=j;
            }
        }
        if(strcmp(s,"WIN")==0) op=0;
        else if(strcmp(s,"LOSE")==0) op=1;
        else op=2;
        int st=pw2[m]-1;
        for(i=0;i<pw3[m];i++) hash[i]=inf;
        int ret=maxmin(st,0,0,-inf,inf);
        if(op==0) {
            if(ret>0) puts("YES");
            else puts("NO");
        }else if(op==1) {
            if(ret<0) puts("YES");
            else puts("NO");
        }else {
            if(ret==0) puts("YES");
            else puts("NO");
```

```
            }
        }
    }
    //    LOSE
    //    .o.x
    //    .o.o
    //    x...
    //    .x..
\*-------------------------------------------------------*/
```

```
/*************************************************************************\
```

dancing links(DLX)

精确覆盖(部分覆盖) : 选择行, 使得每一列都恰好有一个1.

重复覆盖 : 选择行, 使得每一列都至少有一个1.

重复覆盖+精确覆盖(判独)

重复覆盖部分每一列至少有一个1. 精确覆盖不需要完全覆盖, 只是用来判独.

最小支配集 : 用最少的点去支配所有的点.

```
\*************************************************************************/
```

模板

```
=======================================================================
```

精确覆盖

```
int L[N],R[N],U[N],D[N]; //左右上下
int C[N],H[N]; //C[i],H[i]表示i对应的哪一列,哪一行
int row[N]; //节点属于哪一行
int S[N]; //列链表中节点的总数
int O[N]; //记录答案
int head,num; //head, num为总节点数(即1的总个数)
void remove(int c) { //删除第c列
    L[R[c]]=L[c]; R[L[c]]=R[c];
    for(int i=D[c];i!=c;i=D[i]) { //删除对应c列有1的行
        for(int j=R[i];j!=i;j=R[j]) {
            U[D[j]]=U[j]; D[U[j]]=D[j];
            S[C[j]]--;
        }
    }
}
void resume(int c) { //恢复第c列
    for(int i=U[c];i!=c;i=U[i]) { //恢复c对应的有1的行
        for(int j=L[i];j!=i;j=L[j]) {
            U[D[j]]=j; D[U[j]]=j;
            S[C[j]]++;
```

```
        }
    }
    L[R[c]]=c; R[L[c]]=c;
}
bool dfs(int k)  {
    if(R[head]==head) { //全部填完
        //输出选择了哪些行
        //for(int i=0;i<k;i++) printf("%d",row[O[i]]);
        return 1;
    }
    int s=inf,c; //找1最少的列对应的第一个1的位置i
    for(int i=R[head];i!=head;i=R[i])
        if(S[i]<s) s=S[i],c=i;

    remove(c); //删除c对应的列
    for(int i=D[c];i!=c;i=D[i]) {
        O[k]=i;
        for(int j=R[i];j!=i;j=R[j]) remove(C[j]); //删除那一行有1的列
        if(dfs(k+1)) return 1;
        for(int j=L[i];j!=i;j=L[j]) resume(C[j]); //恢复那一行有1的列
    }
    resume(c); //恢复c对应的列
    return 0;
}
void addedge(int r,int c) {
    num++;
    U[num]=U[c]; D[U[c]]=num;  //老的最后行指向新的最后行
    U[c]=num; D[num]=c;   //新的最后行指向列首

    if(H[r]==-1) H[r]=L[num]=R[num]=num; //行的第一个1
    else { //插入到行列表中
        L[num]=L[H[r]]; R[num]=H[r];
        R[L[H[r]]]=num; L[H[r]]=num;
    }
    C[num]=c;   //列指针,表示num所在的列
    row[num]=r; //表示num所在的行
    S[c]++;     //相应的列1的个数++
}
void init() {
    head=0;
    memset(H,-1,sizeof(int)*(n+1));
    for(int i=0;i<=m;i++) {
        U[i]=i; D[i]=i;
        L[i+1]=i; R[i]=i+1;
```

```
        S[i]=0; C[i]=i;
    }
    L[0]=m; R[m]=0;
    num=m;
}
```

==========

重复覆盖

```
//一般求至少选择多少行,使得每一列至少有一个1.
int L[N],R[N],U[N],D[N]; //左右上下
int C[N],H[N]; //C[i],H[i]表示i对应的哪一列,哪一行
int S[N]; //列链表中节点的总数
int O[N]; //记录答案
int head,num,lim; //head, num为总节点数(即1的总个数),lim为至少选择的行数
void remove(int c) {
    for(int i=D[c];i!=c;i=D[i]) {
        L[R[i]]=L[i]; R[L[i]]=R[i];
    }
}
void resume(int c) {
    for(int i=U[c];i!=c;i=U[i]) {
        R[L[i]]=i; L[R[i]]=i;
    }
}
bool use[250]; //大小为列数
int h() {
    memset(use,0,sizeof(use));
    int ret=0;
    for(int c=R[head];c!=head;c=R[c])
        if(!use[c]) {
            ret+=1;
            use[c]=1;
            for(int i=D[c];i!=c;i=D[i])
                for(int j=R[i];j!=i;j=R[j]) use[C[j]]=1;
        }
    return ret;
}
void dfs(int k) {
    if(k+h()>=lim) return;
    if(R[head]==head) {
        lim=min(lim,k);
        return;
    }

    int s=inf,c;
```

```
    for(int i=R[head];i!=head;i=R[i])
        if(S[i]<s) s=S[i],c=i;

    for(int i=D[c];i!=c;i=D[i]) {
        remove(i);
        for(int j=R[i];j!=i;j=R[j]) remove(j);
        dfs(k+1);
        for(int j=L[i];j!=i;j=L[j]) resume(j);
        resume(i);
    }
}
void addedge(int r,int c) {
    num++;
    U[num]=U[c]; D[U[c]]=num;  //老的最后行指向新的最后行
    U[c]=num; D[num]=c;   //新的最后行指向列首

    if(H[r]==-1) H[r]=L[num]=R[num]=num; //行的第一个1
    else { //插入到行列表中
        L[num]=L[H[r]]; R[num]=H[r];
        R[L[H[r]]]=num; L[H[r]]=num;
    }
    C[num]=c;   //列指针,表示num所在的列
    S[c]++;     //相应的列1的个数++
}
void init() {
    head=0;
    memset(H,-1,sizeof(int)*(n+1));
    for(int i=0;i<=m;i++) {
        U[i]=i; D[i]=i;
        L[i+1]=i; R[i]=i+1;
        S[i]=0; C[i]=i;
    }
    L[0]=m; R[m]=0;
    num=m;
}
void build_map() {
    int i,j,x;
    for(i=1;i<=n;i++) {
        scanf("%d",&j);
        while(j--) {
            scanf("%d",&x);
            addedge(i,x);
        }
    }
```

```
    lim=n;
}
int main() {
    int i,j,c;
    while(scanf("%d%d",&m,&n)!=EOF) {
        init();
        build_map();
        dfs(0);
        printf("%d\n",lim);
    }
}
```

==========

精确+重复覆盖

//前 n 列为重复覆盖 ，后 nn 列为精确覆盖(一般用来判独)

```
    bool dfs(int k) {
        if(R[head]==head||R[head]>n) {
            //输出解
            return 1;
        }
        int s=inf,c;
        for(int i=R[head];i!=head;i=R[i])
            if(S[i]<s&&i<=n) s=S[i],c=i;
        for(int i=D[c];i!=c;i=D[i]) {
            O[k]=row[i];
            remove(i);
            for(int j=R[i];j!=i;j=R[j])
                if(j<=n) remove(j);
            for(int j=R[i];j!=i;j=R[j])
                if(j>n) remove2(C[j]);
            if(dfs(k+1)) return 1;
            for(int j=L[i];j!=i;j=L[j])
                if(j>n) resume2(C[j]);
            for(int j=L[i];j!=i;j=L[j])
                if(j<=n) resume(j);
            resume(i);
        }
        return 0;
    }
```

***--*/**

/*--*

DLX 的应用

***--*/**

数独建模

[数独模板]

dep为depXdep小方格边长.

sz为整个数独的边长.

[建图]

行:sz*sz*sz.

即:sz*sz个格子,(1~sz)种可能

列:(sz+sz+sz)*sz+sz*sz

sz行,sz列,sz个depXdep格子乘以(1~sz)数字加上(sz*sz)每个格子填一个

(比如列中,第1行填数字9这个状态:即第一行只能选一次数字9)

如: 行9*9*9=729;81个格子,(1~9)种可能

列(9+9+9)*9+81=324;9行,9列,9个3X3格子*9数字+81每个格子填一个

[比如列中,第1行填数字9这个状态:即第一行只能选一次数字9]

```
int to[5000][3]; //属于那一行,那一列,填什么数字
int n,m;
int sz,dep; //整个大小,和小depXdep的大小
int g[20][20]; //原图
char ans[22][22]; //输出答案
int cnt;
bool dfs(int k) {
   if(R[head]==head) {//全部填完
      if(++cnt>1) return 1;
      for(int i=1;i<=sz;i++)
         for(int j=1;j<=sz;j++) {
            char c=(g[i][j]<=9)?(g[i][j]+'0'):(g[i][j]+'A'-10);
            ans[i][j]=c;
         }
      return 0;
   }
   //...
   for(int i=D[c];i!=c;i=D[i]) {
      int x=to[row[i]][0],y=to[row[i]][1],z=to[row[i]][2];
      g[x][y]=z; //保存答案
      //...
   }
   //...
}
int solve() {
   int i,j,k,nn=0; //nn为第nn行
   init();
   for(i=1;i<=sz;i++)
      for(j=1;j<=sz;j++) { //g[i][j]
         for(k=1;k<=sz;k++) { //选数字k
            nn++;
```

```
            if(g[i][j]==0||g[i][j]==k) { //未确定 或是 确定的数字
               int box=(i-1)/dep*dep+(j-1)/dep+1; //第几个depXdep的格子
               addedge(nn,(i-1)*sz+k); //行
               addedge(nn,sz*sz+(j-1)*sz+k); //列
               addedge(nn,2*sz*sz+(box-1)*sz+k); //3X3的小格子
               addedge(nn,3*sz*sz+(i-1)*sz+j); //每个格子选一个数字
            }
         }
      }
   cnt=0; dfs(0);
   return cnt;
}
int main() {
   int i,j,k,t,cas=0; char s[20];
   while(scanf("%d",&dep)!=EOF) {
      sz=dep*dep;
      n=sz*sz*sz; //行
      m=(sz+sz+sz)*sz+sz*sz*sz; //列
      //----- 表示构图后行的状态 -----
      t=0;
      for(i=1;i<=sz;i++)
         for(j=1;j<=sz;j++)
            for(k=1;k<=sz;k++) to[++t][0]=i,to[t][1]=j,to[t][2]=k;
      //----- 将A~Z转换成数字 -----
      for(i=1;i<=sz;i++) {
         scanf("%s",s+1);
         for(j=1;j<=sz;j++) {
            if(s[j]=='.')    g[i][j]=0;
            else if(s[j]<'A') g[i][j]=s[j]-'0';
            else            g[i][j]=s[j]-'A'+10;
         }
      }
      //----- 输出答案 -----
      int ret=solve();
      if(ret==0) puts("No Solution"); //无解
      else if(ret>1) puts("Multiple Solutions"); //多解
      else { //唯一解
         for(i=1;i<=sz;i++) {
            for(j=1;j<=sz;j++) printf("%c",ans[i][j]);
            puts("");
         }
      }
   }
}
```

```
\*------------------------------------------------------*/
```

N 皇后建模

```
    行: N*N个格子
    列: 行,列,左斜,右斜.(左右斜用来判冲突,不需要完全覆盖.只要覆盖行和列即可)
    PS: 查找列的时候,只查找前 2*N 的列
        for(int i=R[head];i<=2*sz;i=R[i]) //在前面2*SZ行找!!!
            if(S[i]<s) s=S[i],c=i;
//计算方法数
int sz,n,m,cnt;
int dfs(int k) {
   if(k==sz){cnt++;return 1;} //选了K个行,说明放了K个皇后
   if(R[head]==head) return 1;
   int s=inf,c; //找1最少的列对应的第一个1的位置i
   for(int i=R[head];i<=2*sz;i=R[i])//在前面2*SZ行找!!!
      if(S[i]<s) s=S[i],c=i;
   //...
}
int solve() {
   int i,j,nn=0;
   init();
   for(i=1;i<=sz;i++) {
      for(j=1;j<=sz;j++) {
         nn++;
         addedge(nn,i); //行
         addedge(nn,sz+j); //列
         addedge(nn,2*sz+(i-1)+(sz-j)+1); //右斜
         addedge(nn,4*sz-1+(i-1)+(j-1)+1); //左斜
      }
   }
   cnt=0; dfs(0);
   return cnt;
}
int main() {
   int ans[12];
   for(sz=1;sz<=10;sz++) { //打表预处理
      n=sz*sz; m=6*sz-2;
      ans[sz]=solve();
   }
}
\*------------------------------------------------------*/
```

N 皇后的直接构造(非 DLX)

一、当 n%6!=2 或 n%6!=3 时，有一个解为：

2,4,6,8,...,n,1,,3,5,7,...,n-1(n 为偶数)

2,4,6,8,...,n-1,,1,3,5,7,...,n(n 为奇数)

(上面序列第 i 个数为 ai，表示在第 i 行 ai 列放一个皇后；...
省略的序列中，相邻两数以 2 递增。下同)

二、当 n%6==2 或 n%6==3 时，

(当 n 为偶数,k=n/2；当 n 为奇数,k=(n-1)/2)

我看了这个题目，也想找找规律的理论依据

k,k+2,k+4...n;2,4...k-2;k+3,k+5...n-1;1,3,5...k+1 (k 为偶数,n 为偶数)

k,k+2,k+4...n-1;2,4...k-2;k+3,k+5...n-2;1,3,5...k+1,n (k 为偶数,n 为奇数)

k,k+2,k+4...n-1;1,3,5...k-2;k+3...n;2,4...k+1 (k 为奇数,n 为偶数)

k,k+2,k+4...n-2;1,3,5...k-2;k+3...n-1;2,4...k+1,n (k 为奇数,n 为奇数)

----------/

/**********\

高级搜索

BFS(广度优先搜索)

DFS(深度优先搜索)

记忆化搜索(基于深搜)

就是用一个数组,dp[state] 表示state这个状态的结果,如果进行深搜时,发现已经得出dp[state]的结果了,就直接 return dp[state];

双向广搜

从初始结点和目标结点开始分别作两次BFS,每次选择队列中结点少的一边进行扩展,并且检测两边是否出现了状态重合

二分状态搜索

多半应用于背包的搜索,就是给你n<=30个物品,每个物品有一定的价值(Value<=10^9),问你将其分成两堆,使得两堆的总价值和差值最小是多少.

如果我们直接dfs搜索的话,复杂度会达到O(2^n).极限复杂度就是 2^30 !!!

我们可以先记录sum为所有物品的价值总和,然后先将前15个物品的所有组合状态用一个 hash[state]记录下来,然后按价值排序.

再对后15个物品枚举组合状态,记其组合的价值为 Val,那么我们在 hash[]中二分查找一个状态 state,使得 Val+hash[state]最接近 sum/2(就是总价值的一半)...然后最小差值就在 O((2^15) * log2(2^15)) + O((2^15)*log2(2^15)) [前15个状态排序+后15个状态二分的复杂度] 的时间内完美解决了...

启发式搜索

启发式合并,很好很强大.Nlog(N)的复杂度.

A*搜索

定义一个估计函数 G = g(x) + h(x) . 其中 g(x)为你实际已经走过的距离,然后其重点就在 h(x) 的选择,比如迷宫搜索最短路时,可以选择 h(x)为到终点的曼哈顿距离.那么如果用 BFS 实现,那么用一个优先队列,使得每次增广时选择 G 小的先增广,这样搜最优解的希望更大一些.如果是 DFS 的话,假设我们目前已经搜到的最优解为 Ans,那么我们再搜另一个状态他的 G 如果大于 Ans,那么就没有必要再搜下去了,剪掉...

A*的关键就在 h(x)的选择,一般选择的都是:离目标状态最少还要花费多少步数.(当然这并不是说这样是最好的 h(x).只是普遍的选择...).根据不同的需要,你也可以定义一个更强大的 h(x)来,让你的搜索快的飞起来~~~

IDA*搜索

就是从小到大(当然你也可以二分枚举)不断限制搜索的深度,然后去做DFS半的A*,当搜到一个解的时候,那就是最优解了(如果是二分的话,还需缩小深度继续搜).因为我们是从小到大枚举的...

h()函数:

①曼哈顿距离

②魔方旋转类,每次如果会改变c个数字,那么 (曼哈顿距离和+(c-1))/c 作为估计函数.

反正就是当前状态到大最终状态最理想情况下最少需要多少步来当做估计函数h().

************/**

模板

==========

双向广搜

```
struct State { }; //状态
queue<State>que[2];
bool vis[2];
bool flag;
void bfs(int d) {
    int size=que[d].size();
    while(size--) {
        //普通单广转移新状态v
        if(vis[d][v]) continue;
        if(vis[d^1][v]) { flag=true; return; }
        //新状态入队
    }
}
int dbfs() {
    //初始化
    int cnt=0;
    while(true) {
        cnt++;
        if(que[0].size()<que[1].size()) bfs(0);
        else bfs(1);
        if(flag) break;
    }
    return cnt;
}
```

***----------*/**

/*----------*

高级搜索的技巧

--/

哈希方法:

朴素的哈希：将这 9 个数的排列转化为一个 int 范围的值，范围过大

散列表哈希：转化为一个 int 值后对大素数取模，然后线性解决冲突

map 哈希：直接把状态使用 map 进行 hash，太慢了

字符串哈希：将状态转换成字符串，用字符串 hash，可能会有冲突

逆序数哈希：康托展开的应用

形状哈希：特殊点的位置及物体的形状进行 hash.

(例如:贪吃蛇,对其蛇头位置及弯曲形状进行 hash.)

搜索技巧:

1.预处理一些状态.(先搜一半)

2.先对状态进行排序,然后再搜索,可以以此作为剪枝的工具.

(从大往小搜:容量,边的度)

3.记忆化搜索

4.从终点扩展到每个点(x,y).记录距离为 len[x][y],作为 A*的估计函数.

5.缩图后,再进行搜索.

6.流氓剪枝.

7.正搜不行,进行反搜.

8.根据最优解应该具有的特性,按顺序搜索.

9.状态压缩(二进制,三进制,或者哈希不同形态...)

10.是否能将所有数据的目标状态转换成一样.

--/

/**\

参数搜索(01 分数规划)

求最大比率 mid*b[i]-a[i]

求最小比率 a[i]-mid*b[i]

形如: a[i]/b[i] 比率最大或最小

a[i]/b[i]=ans ==> a[i]=ans*b[i] ==> ans*b[i]-a[i] = 0

```
//下面为每个课程a[i](收益费),b[i](花费)
//要选出k个课程,使得 a[i]/b[i] 最大
int n,k;
double a[N],b[N];
double t[N];
int main() {
    while(scanf("%d%d",&n,&k),n||k) {
        for(i=1;i<=n;i++) scanf("%lf",&a[i]);
        for(i=1;i<=n;i++) scanf("%lf",&b[i]);
        double top=1.0,bot=0,mid;
        while(top-bot>1e-4) {
```

```
            mid=(top+bot)/2.0;
            for(i=1;i<=n;i++) t[i]=mid*b[i]-a[i];
            sort(t+1,t+n+1);
            double sum=0;
            for(i=1;i<=n-k;i++) sum+=t[i];
            if(sum>=0) top=mid;
            else       bot=mid;
        }
        printf("%lf\n",mid);
    }
}
\*------------------------------------------------------*/
```

/**\

图的连通度

| 在图中删去部分元素（点或边），使得图中指定的两个点s和t不连通
| （不存在从s到t的路径），求至少要删去几个元素。
| 图的连通度分为点连通度和边连通度：
| **（1）点连通度：**只许删点，求至少要删掉几个点（当然，s和t不能删去，这
| 里保证原图中至少有三个点）；
| **（2）边连通度：**只许删边，求至少要删掉几条边。
| 并且，有向图和无向图的连通度求法不同，因此还要分开考虑（对于混合图，
| 只需将其中所有的无向边按照
| 无向图的办法处理、有向边按照有向图的办法处理即可）。
| **【1】有向图的边连通度：**
| 这个其实就是最小割问题。以s为源点，t为汇点建立网络，原图中的每条边
| 在网络中仍存在，容量为1，
| 求该网络的最小割（也就是最大流）的值即为原图的边连通度。
| **【2】有向图的点连通度：**
| 需要拆点。建立一个网络，原图中的每个点i在网络中拆成i’与i’’，有一条
| 边<i’，i’’>，容量为1
| （<s’，s’’>和<t’，t’’>例外，容量为正无穷）。原图中的每条边<i，j>
| 在网络中为边<i’’，j’>，
| 容量为正无穷。以s’为源点、t’’为汇点求最大流，最大流的值即为原图的
| 点连通度。
| 说明：最大流对应的是最小割。显然，容量为正无穷的边不可能通过最小割，
| 也就是原图中的边和s、t两个点
| 不能删去；若边<i，i’’>通过最小割，则表示将原图中的点i删去。
| **【3】无向图的边连通度：**
| 将图中的每条边(i，j)拆成<i，j>和<j，i>两条边，再按照有向图的办法
| （【1】）处理；
| **【4】无向图的点连通度：**

| 将图中的每条边(i, j)拆成<i, j>和<j, i>两条边，再按照有向图的办法
| （【2】）处理。
**/

数据结构

/**\

后缀数组

**/

模板

```
======================================================================
int n,m;
char s[N];
int r[3*N]; //3倍
int wa[N],wb[N],wc[N],wv[N];
int sa[3*N]; //3倍
int rk[N],height[N];
//dc3
#define F(x) ((x)/3+((x)%3==1?0:tb))
#define G(x) ((x)<tb?(x)*3+1:((x)-tb)*3+2)
int c0(int *r,int a,int b) {
    return r[a]==r[b]&&r[a+1]==r[b+1]&&r[a+2]==r[b+2];
}
int c12(int k,int *r,int a,int b) {
    if(k==2) return (r[a]<r[b])||(r[a]==r[b]&&c12(1,r,a+1,b+1));
    else return (r[a]<r[b])||(r[a]==r[b]&&wv[a+1]<wv[b+1]);
}
void sort(int *r,int *a,int *b,int n,int m) {
    int i;
    for(i=0;i<n;i++) wv[i]=r[a[i]];
    for(i=0;i<m;i++) wc[i]=0;
    for(i=0;i<n;i++) wc[wv[i]]++;
    for(i=1;i<m;i++) wc[i]+=wc[i-1];
    for(i=n-1;i>=0;i--) b[--wc[wv[i]]]=a[i];
}
void dc3(int *r,int *sa,int n,int m) {
    int i,j,*rn=r+n,*san=sa+n,ta=0,tb=(n+1)/3,tbc=0,p;
    r[n]=r[n+1]=0;
    for(i=0;i<n;i++) if(i%3!=0) wa[tbc++]=i;
```

```
    sort(r+2,wa,wb,tbc,m);
    sort(r+1,wb,wa,tbc,m);
    sort(r,wa,wb,tbc,m);
    for(p=1,rn[F(wb[0])]=0,i=1;i<tbc;i++)
        rn[F(wb[i])]=c0(r,wb[i-1],wb[i])?p-1:p++;
    if(p<tbc) dc3(rn,san,tbc,p);
    else for(i=0; i<tbc; i++) san[rn[i]]=i;
    for(i=0;i<tbc;i++) if(san[i]<tb) wb[ta++]=san[i]*3;
    if(n%3==1) wb[ta++]=n-1;
    sort(r,wb,wa,ta,m);
    for(i=0;i<tbc;i++) wv[wb[i]=G(san[i])]=i;
    for(i=0,j=0,p=0;i<ta&&j<tbc;p++)
        sa[p]=c12(wb[j]%3,r,wa[i],wb[j])?wa[i++]:wb[j++];
    for(;i<ta;p++) sa[p]=wa[i++];
    for(;j<tbc;p++) sa[p]=wb[j++];
}
void calheight(int r[],int sa[],int n) {
    int i,j,k=0;
    for(i=1;i<=n;i++) rk[sa[i]]=i;
    for(i=0;i<n;height[rk[i++]]=k)
        for(k?k--:0,j=sa[rk[i]-1];r[i+k]==r[j+k];k++);
}
\*------------------------------------------------------*/

/*------------------------------------------------------*\
```

后缀数组的应用

```
\*------------------------------------------------------*/
```

重复 2 次以上的子串的个数(不重叠)【单串】

枚举字串长度h ， 对于每一次的h,利用height数组,
找出连续的height大于等于h的里面最左端和最右端得为之l和r.
如果l + h - 1 < r的话,说明没有重叠,答案加1.
答案：ans+=check(i); 1<=i<=n

```
\*------------------------------------------------------*/
```

重复 k 次以上最大子串(可重叠)【单串】

二分枚举长度.len
连续height大于等于len的个数超过k个,说明长度为len的存在重复k次的子串

```
\*------------------------------------------------------*/
```

最长重复 k 次可重叠子串长度及最右边的位置【单串】

同上,过程中记录位置即可

```
\*------------------------------------------------------*/
```

最大不重叠重复子串【单串】

二分+判断连续 height>=mid 的最up和low位置差值是否大于mid

```
\*------------------------------------------------------*/
```

最长回文字串+LCP【单串】

后缀数组+RMQ

LCP(i,j)=min(height[rank[i+1]],height[rank[i+2]]
...height[rank[j]])

枚举回文串中心

```
//THE code
int st[20][N];
int tlog2[N];
void rmq(int n) {
   for(int i=1;i<=n;i++) st[0][i]=height[i];
   for(int j=1;j<=tlog2[n];j++)
      for(int i=1;j<=tlog2[n+1-i];i++)
         st[j][i]=min(st[j-1][i],st[j-1][i+(1<<j>>1)]);
}
int get_rmq(int l,int r) {
   if(l>r) swap(l,r);
   int tmp=tlog2[r-l+1];
   return min(st[tmp][l],st[tmp][r-(1<<tmp)+1]);
}
int main() {
   int i;
   tlog2[0]=-1;
   for(i=1;i<N;i++)
      tlog2[i]=(i&(i-1))?tlog2[i-1]:tlog2[i-1]+1;
   while(scanf("%s",s)!=EOF) {
      n=strlen(s);
      for(i=0;i<n;i++) r[i]=s[i];
      for(i=0;i<n;i++) r[n+1+i]=s[n-1-i];
      r[n]=1; r[2*n+1]=0;
      da(r,sa,2*n+2,128);
      calheight(r,sa,2*n+1);
      rmq(2*n+1);
      int ans=0,k=0;
      for(i=0;i<n;i++) {
         int l=min(rk[i],rk[2*n-i]);
         int r=max(rk[i],rk[2*n-i]);
         int tmp=get_rmq(l+1,r)*2-1;
         if(tmp>ans) {ans=tmp;k=i;}
         if(i>0) {
            l=min(rk[i],rk[2*n-i+1]);
            r=max(rk[i],rk[2*n-i+1]);
            tmp=get_rmq(l+1,r)*2;
            if(tmp>ans) {ans=tmp;k=i;}
         }
```

```
      }
      if(ans%2) { //奇数
         for(i=k-ans/2;i<=k;i++) printf("%c",s[i]);
         for(;i<=k+ans/2;i++) printf("%c",s[i]);
         puts("");
      }else { //偶数
         for(i=k-ans/2;i<k;i++) printf("%c",s[i]);
         for(;i<k+ans/2;i++) printf("%c",s[i]);
         puts("");
      }
   }
}
\*-------------------------------------------------------*/
```

回文子串个数【单串】

```
     S + (不在S的字符) + S反串
//THE code
int main() {
   int i,t;
   init();
   while(scanf("%s",s)!=EOF) {
      n=strlen(s);
      for(i=0;i<n;i++) r[i]=s[i];
      for(i=0;i<n;i++) r[n+1+i]=s[n-1-i];
      r[n]=128; r[2*n+1]=0;
      da(r,sa,2*n+2,130);
      calheight(r,sa,2*n+1);
      rmq(2*n+1);
      int ans=0;
      for(i=0;i<n;i++) {
         ans+=get_rmq(rk[i],rk[2*n-i]);  //奇数
         if(i>0) ans+=get_rmq(rk[i],rk[2*n-i+1]); //偶数
      }
      printf("%d\n",ans);
   }
}
\*-------------------------------------------------------*/
```

不同子串个数【单串】

```
      int sum=0;
      for(i=1;i<=n;i++)
        sum+=n-sa[i]-height[i];
\*-------------------------------------------------------*/
```

求重复次数最多的连续重复子串【单串】

[题意]给定一个字符串，求重复次数最多的连续重复子串。

[解析]先穷举长度L,然后求长度为L的子串最多能连续出现几次.

首先连续出现1次是肯定可以的,所以这里只考虑至少2次的情况.
假设在原字符串中连续出现2次,记这个子字符串为S.
那么S肯定包括了字符r[0],r[L],r[L*2],r[L*3]...中的某相邻的两个.
所以只须看字符r[L*i]和r[L*(i+1)]往前和往后各能匹配到多远,
记这个总长度为K,那么这里连续出现了K/L+1 次.
最后看最大值是多少.
穷举长度L的时间是n,每次计算的时间是n/L.所以整个做法的时间
复杂度是O(n/1+n/2+n/3+……+n/n)=O(nlogn)

--/

给两串字符串,求最长公共子串【两串】

```
  S + 分隔符 + ss   //len即为最长长度
//THE code
  for(i=0;i<x;i++) r[i]=s[i];
  for(i=0;i<y;i++) r[x+i+1]=ss[i];
  r[x]=1; r[n]=0;
  int len=0,idx=0;
  for(i=1;i<=n;i++)
    if(sa[i-1]<x&&sa[i]>x || sa[i-1]>x&&sa[i]<x ) {
      if(height[i]>len) {
        len=height[i];
        idx=min(sa[i-1],sa[i]);
      }
    }
  for(i=1;i<=len;i++) printf("%c",s[idx++]);
```

--/

两个串中长度>=k 的公共子串的数量【两串】

[题意]给定两个字符串A和B,求长度不小于k的公共子串的个数(可以相同)

样例1:

A="xx",B="xx",k=1.长度不小于k的公共子串的个数是5.

样例2:

A="aababaa",B="abaabaa",k=2.个数是22.

[解析]基本思路是计算A的所有后缀和B的所有后缀之间的最长公共前缀长度,
把最长公共前缀长度不小于k的部分全部加起来.
先将两个字符串连起来,中间用一个没有出现过的字符隔开.
按height值分组后,接下来的工作便是快速的统计每组中后缀之间的
最长公共前缀之和.扫描一遍,每遇到一个B 的后缀就统计与前面的A的
后缀能产生多少个长度不小于k的公共子串,这里A的后缀需要用一个
单调的栈来高效的维护.然后对A 也这样做一次.

--/

不小于 K 个字符的最长子串【多串】

将n个串连起来,中间用不同的且没有出现在串中的字符隔开,求后缀数组.
然后二分答案,用和④同样的方法将后缀分成若干组,判断每组的后缀
是否出现在不小于k个的原串中.这个做法的时间复杂度为O(nlogn).

--/

求 n 个字符串的最长公共字串(或反转)【多串】

求n个字符串的最长公共字串(或反转)长度

[解法]合并所有串(+反串).

对合并后串的每个字符用who[]数组记录它原来属于哪个串.

(正反串属于同一串)

仍然二分枚举长度mid,若连续一段height[]>=mid中,

后缀所属的原串种类串等于n个,说明找到.

返回出现串S[1]中的最小字典位置,mid为长度.

```
//THE code 1
bool vis[105];
int who[105];
int check(int mid) {
   int i,j;
   int cnt=0;
   int low=inf,pos=inf;
   memset(vis,0,sizeof(vis));
   for(i=1;i<=nn;i++) {
       if(height[i]>=mid) cnt++;
       else {
           int ct=0;
           for(j=i-cnt;j<i;j++)
            if(!vis[who[sa[j]]]) vis[who[sa[j]]]=1,ct++;
           if(ct==n) return 1;
           for(j=i-cnt;j<i;j++) vis[who[sa[j]]]=0;
           cnt=1;
       }
    }
   return 0;
}
int main() {
   int i,j;
   while(scanf("%d",&n)!=EOF) {
      int w=130; nn=0;
      int l=222;
      for(i=1;i<=n;i++) {
         scanf("%s",s);
         int x=strlen(s);
         l=min(l,x);
         for(j=0;j<x;j++) r[nn+j]=s[j],who[nn+j]=i;
         nn+=x; r[nn++]=w++;
         for(j=0;j<x;j++) r[nn+j]=s[x-1-j],who[nn+j]=i;
         nn+=x; r[nn++]=w++;
      } r[nn]=0;
      da(r,sa,nn+1,w);
```

```
      calheight(r,sa,nn);
      int top=l,bot=1,mid;
      int ct=0;
      while(top>=bot) {
         mid=(top+bot)/2;
         if(check(mid)) bot=mid+1,ct=mid;
         else          top=mid-1;
      }
      printf("%d\n",ct);
   }
}
=====================================================================
//THE code 2
//输出位置
int check(int mid) {
   int i,j;
   int cnt=0;
   int low=inf;
   memset(vis,0,sizeof(vis));
   for(i=1;i<=nn;i++) {
      if(height[i]>=mid) {
         cnt++;
         //记录出现在串1,且字典序最小的位置
         if(who[sa[i]]==1&&low==inf) low=sa[i];
      }else {
         //判断是否有n个种类串
         if(cnt>=n) {
            int ct=0;
            for(j=i-cnt;j<i;j++)
              if(!vis[who[sa[j]]]) vis[who[sa[j]]]=1,ct++;
            if(ct==n) return low;
            for(j=i-cnt;j<i;j++) vis[who[sa[j]]]=0; //注意还原!
         }
         cnt=1;
         if(who[sa[i]]==1) low=sa[i];
         else            low=inf;
      }
   }
   return -1;
}
```

***--*/**

出现在多于一半以上的最长公共子串【多串】

同上.

***--*/**

每个字符串至少出现两次且不重叠的最长子串【多串】

[题意]给定n个串,求在每个串中至少出现两次且不重叠的最长子串.

[解析]做法和[1]大同小异,也是先将n个字符串连起来,中间用不相同的且没有出现在字符串中的字符隔开,求后缀数组.

然后二分答案,再将后缀分组.判断的时候,要看是否有一组后缀在每个原来的字符串中至少出现两次,并且在每个原来的字符串中.

后缀的起始位置的最大值与最小值之差是否不小于当前答案

(判断能否做到不重叠,若题中没有不重叠要求,,那么不用做此判断)

这个做法的时间复杂度为O(nlogn).

----------/

/**********\

划分树

**********/

询问[l,r]区间,找一个X, Σ|X-xi| 最小.

[解析] X为区间的中位数.

```
struct seg_tree {
    int l,r,mid;
}tree[N*4];
int sa[N];
int val[20][N];
int toleft[20][N]; //放到左边的个数
ll sleft[20][N]; //放到左边的和
ll sum[N];
ll lsum;
void build(int l,int r,int id,int d) {
    tree[id].l=l;
    tree[id].r=r;
    tree[id].mid=(l+r)>>1;
    if(l==r) return;
    int mid=tree[id].mid;
    int same=mid-l+1;
    for(int i=l;i<=r;i++)
        if(val[d][i]<sa[mid]) same--;
    int lpos=l,rpos=mid+1;
    for(int i=l;i<=r;i++) {
        toleft[d][i]=(i==l)?0:toleft[d][i-1];
        sleft[d][i]=(i==l)?0:sleft[d][i-1];
        if(val[d][i]<sa[mid]||(val[d][i]==sa[mid]&&same)){//分到左边
            if(val[d][i]==sa[mid]) same--;
            toleft[d][i]++;
            sleft[d][i]+=val[d][i];
```

```
            val[d+1][lpos++]=val[d][i];
        }else { //划分到右边
            val[d+1][rpos++]=val[d][i];
        }
    }
    build(l,mid,id*2,d+1);
    build(mid+1,r,id*2+1,d+1);
}
int query(int l,int r,int id,int d,int k) {
    if(l==r) return val[d][l];
    int s,ss; // [l,r] [tree[id].l,l-1] 被划分到左边的个数
    ss=(tree[id].l==l)?0:toleft[d][l-1];
    s=toleft[d][r]-ss;
    if(s>=k) { //搜左边
        int ll=tree[id].l+ss;
        int rr=tree[id].l+ss+s-1;
        return query(ll,rr,id*2,d+1,k);
    }else {
        //加上放到左边的和
        lsum+=sleft[d][r]-((tree[id].l==l)?0:sleft[d][l-1]);
        int mid=tree[id].mid;
        int ll=mid+1+(l-tree[id].l-ss); //[T[id].l,l-1]分到右边的个数
        int rr=ll+(r-l-s); //[l,r] 划分到右边的个数
        return query(ll,rr,id*2+1,d+1,k-s);
    }
}
int main() {
    int i,j,a,b,k;
    while(scanf("%d%d",&n,&m)!=EOF) {
        sum[0]=0;
        for(i=1;i<=n;i++) {
            scanf("%d",&sa[i]);
            val[0][i]=sa[i];
            sum[i]=sum[i-1]+sa[i];
        }
        sort(sa+1,sa+1+n);
        build(1,n,1,0);
        while(m--) {
            scanf("%d%d",&a,&b);
            lsum=0;
            int k=(b-a)/2+1;
            int avg=query(a,b,1,0,k);
            ll lx=((ll)(k-1)*(ll)avg-lsum); //左边和右边的差和
            ll rx=(sum[b]-sum[a-1]-lsum-(ll)(b-a+1-k+1)*(ll)avg);
```

```
            ll res=lx+rx;
            printf("%I64d\n",res);
        }
    }
}
\*------------------------------------------------------*/
```

/**\

树链剖分

**/

```
/*
    [HDU3966] 区间更新,单点求值的代码.
        第一步:dfs(); fa,son,dep,sz,val
        第二步:build_tree(); top,w,who
        第三步:build(); 建立线段树
        第四步:Update(); 进行更新,或者查询
    [最大连续自序和问题]
        1 a b:询问a->b路径的最大连续子序列和.
        2 a b c:a->b的路径点权变成c.
    [PS]注意询问的时候,a->f->b
        注意 f->a爬坡时,要翻转区间的左连续和右连续!!!
*/
int n,m,q;
int val[N];
//------------------------- Init_Tree -------------------------
//val[]当为边时,val保存v与父亲的边权.
//fa[]父亲,son[]重儿子,dep[]深度,sz[]子树节点数;
//top[]u所在链的顶端节点;
//w[]节点(边)在线段树中的位置;
//who[]线段树节点对应树中的位置;
//z线段树的节点个数.
int fa[N],son[N],dep[N],sz[N];
int top[N],w[N],who[N],z;
void dfs(int u,int f,int deep) {
    fa[u]=f; son[u]=0; dep[u]=deep; sz[u]=1;
    for(int i=p[u];i!=-1;i=e[i].nxt) {
        int v=e[i].b;
        if(v==f) continue;
        dfs(v,u,deep+1);
        //val[v]=e[i].c //边的时候
        if(sz[v]>sz[son[u]]) son[u]=v;
        sz[u]+=sz[v];
```

```
    }
}
void build_tree(int u,int tp) {
    w[u]=++z; who[z]=u; top[u]=tp;
    if(son[u]) build_tree(son[u],top[u]);
    for(int i=p[u];i!=-1;i=e[i].nxt) {
        int v=e[i].b;
        if(v!=son[u]&&v!=fa[u]) build_tree(v,v);
    }
}
//------------------------- SEG_TREE -------------------------
struct Seg_Tree {
    int l,r,mid,c;
}tree[N<<2];
void up(int id) {
    if(tree[id].c!=0) {
        tree[id*2].c+=tree[id].c;
        tree[id*2+1].c+=tree[id].c;
        tree[id].c=0;
    }
}
void build(int l,int r,int id) {
    tree[id].l=l; tree[id].r=r;
    tree[id].mid=(l+r)>>1;
    tree[id].c=0;
    if(l==r) { tree[id].c=val[who[l]]; return; }
    int mid=tree[id].mid;
    build(l,mid,id*2);
    build(mid+1,r,id*2+1);
}
void update(int l,int r,int id,int c) {
    if(tree[id].l==l&&tree[id].r==r) { tree[id].c+=c; return; }
    up(id);
    int mid=tree[id].mid;
    if(r<=mid) update(l,r,id*2,c);
    else if(l>mid) update(l,r,id*2+1,c);
    else {
        update(l,mid,id*2,c);
        update(mid+1,r,id*2+1,c);
    }
}
int query(int pos,int id) {
    if(tree[id].l==tree[id].r) { return tree[id].c; }
    up(id);
```

```
    int mid=tree[id].mid;
    if(pos<=mid)return query(pos,id*2);
    else       return query(pos,id*2+1);
}
//------------------------- Update() ------------------------
void Update(int va,int vb,int c) {
    int f1=top[va],f2=top[vb];
    while(f1!=f2) {
        if(dep[f1]<dep[f2]) swap(f1,f2),swap(va,vb);
        update(w[f1],w[va],1,c);
        va=fa[f1]; f1=top[va];
    }
    //下面为在同一条链上的情况
    //if(va==vb) return; //当线段树的节点保存的是边权时.
    if(dep[va]>dep[vb]) swap(va,vb);
    update(w[va],w[vb],1,c);
    //update(w[son[va]],w[vb],1,c); //边的时候
}
void init() { z=0; sz[0]=0; }
//------------------------- THE END ------------------------
int main() {
    int i,a,b,c; char op[10];
    while(scanf("%d%d%d",&n,&m,&q)!=EOF) {
        init();
        for(i=1;i<=n;i++) scanf("%d",&val[i]);
        while(m--) {
            scanf("%d%d",&a,&b);
            addedge(a,b);
        }
        dfs(1,1,1);
        build_tree(1,1);
        build(1,z,1);
        while(q--) {
            scanf("%s",op);
            if(op[0]=='I') { //a到b路径权值增加c
                scanf("%d%d%d",&a,&b,&c);
                Update(a,b,c);
            }else if(op[0]=='D') { //a到b路径权值减少c
                scanf("%d%d%d",&a,&b,&c);
                Update(a,b,-c);
            }else { //询问点a的权值
                scanf("%d",&a);
                int ret=query(w[a],1);
                printf("%d\n",ret);
```

```
            }
        }
    }
}
\*------------------------------------------------*/

/**********************************************************\
```

Splay-Tree

```
\**********************************************************/
```

模板

==========

[操作及模板]

插入,删除,区间旋转,区间删除并插入到第 k 位置后面.

①单点插入:

插入到第 k 位置后面,将 k 提根,再将 k+1 提到根下.再将要插入的放到 k+1 的左子树中.

②单点删除:

将 k-1 提跟,k+1 提到根下.再将 k+1 的左子树删去.

③a b c 截取区间 a-b,在剩下的数列中,将 a-b 放到 c 后面

[做法]提根(a-1,b+1),截取 a-b.push_up().提根(c,c+1),插入 a-b

④a b 旋转 a-b

[做法]提根(a-1,b+1).放 lazy

[注意!!!]下放标记 lazy 的时候要注意儿子节点是否为空(0),否则会影响结果

```
//下面为区间求和
int n,m;
int num[N];
struct SplayTree {
    void Rotate(int x,int f) { //旋转
        int y=pre[x],z=pre[y];
        push_down(x);
        push_down(y);
        ch[y][!f]=ch[x][f]; pre[ch[x][f]]=y;
        ch[x][f]=y; pre[y]=x;
        pre[x]=z;
        if(pre[x]) ch[z][ch[z][1]==y]=x;
        push_up(y);
    }
    void Splay(int x,int goal) { //提根
        push_down(x);
        while(pre[x]!=goal) {
            if(pre[pre[x]]==goal) {
                Rotate(x,ch[pre[x]][0]==x);
```

```
            }else {
                int y=pre[x],z=pre[y];
                int f=(ch[z][0]==y);
                if(ch[y][f]==x) Rotate(x,!f),Rotate(x,f);
                else            Rotate(y,f),Rotate(x,f);
            }
        }
        push_up(x);
        if(goal==0) root=x;
    }
    void RotateTo(int k,int goal) {//把第 k 位的数转到 goal 下边
        int x=root;
        push_down(x);
        while(sz[ch[x][0]]!=k) {
            if(k<sz[ch[x][0]]) {
                x=ch[x][0];
            }else {
                k-=(sz[ch[x][0]]+1);
                x=ch[x][1];
            }
            push_down(x);
        }
        Splay(x,goal);
    }
//-------------------------------------------------------------
    void clear() {
        ch[0][0]=ch[0][1]=pre[0]=sz[0]=0;
        root=n=0;
        //信息.....
        val[0]=sum[0]=lazy[0]=0;

        NewNode(root,-inf);
        NewNode(ch[root][1],inf);
        pre[n]=root;
        sz[root]=2;
    }
    void NewNode(int &x,int c) {
        x=++n;
        ch[x][0]=ch[x][1]=pre[x]=0;
        sz[x]=1;
        //信息.....
        val[x]=sum[x]=c;
        lazy[x]=0;
    }
```

```
    void push_up(int x) {
        sz[x]=1+sz[ch[x][0]]+sz[ch[x][1]];
        sum[x]=val[x]+sum[ch[x][0]]+sum[ch[x][1]];
    }
    void push_down(int x) {
        //lazy
        if(lazy[x]) { //注意 val[]的更新!
            if(ch[x][0]) { //注意!!!
                sum[ch[x][0]]+=lazy[x]*sz[ch[x][0]];
                val[ch[x][0]]+=lazy[x];
                lazy[ch[x][0]]+=lazy[x];
            }
            if(ch[x][1]) { //注意!!!
                sum[ch[x][1]]+=lazy[x]*sz[ch[x][1]];
                val[ch[x][1]]+=lazy[x];
                lazy[ch[x][1]]+=lazy[x];
            }
            lazy[x]=0;
        }
    }
    void init(int pos,int tot) { //初始化建树
        clear(); cnt=tot;
        RotateTo(pos,0);
        RotateTo(pos+1,root);
        build(ch[ch[root][1]][0],1,tot,ch[root][1]);
        push_up(ch[root][1]);
        push_up(root);
    }
    void build(int &x,int l,int r,int f) {
        if(l>r) return;
        int mid=(l+r)>>1;
        NewNode(x,num[mid]);
        build(ch[x][0],l,mid-1,x);
        build(ch[x][1],mid+1,r,x);
        pre[x]=f;
        push_up(x);
    }
//------------------------------------------------------------
/* 遍历结果 */
    void out() { //遍历
        RotateTo(0,0);
        RotateTo(cnt+1,root);
        ct=0;
        travel(ch[ch[root][1]][0]);
```

```
    }
    void travel(int id) {
        if(id==0) return;
        //push_down(id);
        travel(ch[id][0]); ct++;
        printf(ct==cnt?"%d\n":"%d ",val[id]);
        travel(ch[id][1]);
    }
//-------------------------------------------------------------
    int root,n,cnt,ct; //cnt 为总节点数
    int ch[N][2]; //左右儿子
    int pre[N]; //父节点
    int sz[N]; //子树大小
    int val[N]; //点权
    int sum[N]; //求和
    int lazy[N]; //延迟标志
}spt;
//spt.init(0,n); //读入数据 num[i]后,先要初始化
```

==========

维护序列

```
/*
    插入:INSERT pos n c1...cn (pos 后面插入 n 个数字)
    删除:DELETE pos n (删除 pos 开始的 n 个数字)
    修改:MAKE-SAME pos n c (pos 开始的 n 个数字修改为 c)
    翻转:REVERSE pos n (pos 开始的 n 个数字翻转)
    求和:GET-SUM pos n (pos 开始的 n 个数字求和)
    求和最大的子序列:MAX-SUM (整个区间的和最大的子序列)
*/
int n,m;
int num[N];
struct SplayTree {
    void Rotate(int x,int f); //旋转
    void Splay(int x,int goal); //提根
    void RotateTo(int k,int goal); //把第 k 位的数转到 goal 下边
    void build(int &x,int l,int r,int f) ;
    void init(int pos,int tot);
    void clear() {
        ch[0][0]=ch[0][1]=pre[0]=sz[0]=0;
        root=n=0;
        //信息.....
        val[0]=ls[0]=rs[0]=ss[0]=-inf;
        sum[0]=0;
        lazy1[0]=0; lazy2[0]=-inf;
        NewNode(root,-inf);
```

```
    NewNode(ch[root][1],-inf);
    pre[n]=root;
    sz[root]=2;
}
void NewNode(int &x,int c) {
    if(top) x=pool[--top];
    else x=++n;
    ch[x][0]=ch[x][1]=pre[x]=0;
    sz[x]=1;
    //信息.....
    val[x]=ls[x]=rs[x]=ss[x]=sum[x]=c;
    lazy1[x]=0; lazy2[x]=-inf;
}
void push_up(int x) {
    int lx=ch[x][0],rx=ch[x][1];
    sz[x]=1+sz[lx]+sz[rx];
    sum[x]=val[x]+sum[lx]+sum[rx];
    ls[x]=max(ls[lx],sum[lx]+val[x]+max(0,ls[rx]));
    rs[x]=max(rs[rx],sum[rx]+val[x]+max(0,rs[lx]));
    ss[x]=max(0,rs[lx])+val[x]+max(0,ls[rx]);
    ss[x]=max(ss[x],max(ss[lx],ss[rx]));
}
void update_rev(int x) {
    if(!x) return;
    swap(ch[x][0],ch[x][1]);    swap(ls[x],rs[x]);
    lazy1[x]^=1;
}
void update_same(int x,int v) {
    if(!x) return;
    val[x]=v; sum[x]=v*sz[x];
    ss[x]=ls[x]=rs[x]=max(v,v*sz[x]);
    lazy2[x]=v;
}
void push_down(int x) {
    //lazy
    if(lazy1[x]) {
        update_rev(ch[x][0]);    update_rev(ch[x][1]);
        lazy1[x]=0;
    }
    if(lazy2[x]!=-inf) {
        update_same(ch[x][0],lazy2[x]);    update_same(ch[x][1],lazy2[x]);
        lazy2[x]=-inf;
    }
}
```

```
void ins(int pos,int len) { //插入
	cnt+=len;
	RotateTo(pos,0);
	RotateTo(pos+1,root);
	build(ch[ch[root][1]][0],1,len,ch[root][1]);
	push_up(ch[root][1]);
	push_up(root);
}
void erase(int x) { //删除
	if(!x) return;
	pool[top++]=x;
	erase(ch[x][0]);
	erase(ch[x][1]);
}
void del(int l,int r) { //删除
	RotateTo(l-1,0);
	RotateTo(r+1,root);
	int key=ch[ch[root][1]][0];
	ch[ch[root][1]][0]=0;
	cnt-=sz[key];
	erase(key);
	push_up(ch[root][1]);
	push_up(root);
}
void change(int l,int r,int c) { //修改
	RotateTo(l-1,0);
	RotateTo(r+1,root);
	int key=ch[ch[root][1]][0];
	update_same(key,c);
	push_up(ch[root][1]);
	push_up(root);
}
void flip(int l,int r) { //翻转
	RotateTo(l-1,0);
	RotateTo(r+1,root);
	int key=ch[ch[root][1]][0];
	update_rev(key);
}
int get_sum(int l,int r) { //求和
	RotateTo(l-1,0);
	RotateTo(r+1,root);
	int key=ch[ch[root][1]][0];
	return sum[key];
}
```

```
    int max_sum() { //求最大连续子序和
        RotateTo(0,0);
        RotateTo(cnt+1,root);
        int key=ch[ch[root][1]][0];
        return ss[key];
    }
//-------------------------------------------------------
    int root,n,cnt,top; //cnt 为总节点数
    int ch[N][2]; //左右儿子
    int pre[N]; //父节点
    int sz[N]; //子树大小
    int val[N],sum[N],ls[N],rs[N],ss[N];
    bool lazy1[N]; //翻转
    int lazy2[N]; //修改
    int pool[N]; //内存池
}spt;
int main() {
    int i,j,a,b,c; char op[30];
    while(scanf("%d%d",&n,&m)!=EOF) {
        for(i=1;i<=n;i++) scanf("%d",&num[i]);
        spt.init(0,n);
        while(m--) {
            scanf("%s",op);
            if(op[0]=='I') { //插入
                scanf("%d%d",&a,&n);
                for(i=1;i<=n;i++) scanf("%d",&num[i]);
                spt.ins(a,n);
            }else if(op[0]=='D') { //删除
                scanf("%d%d",&a,&b);
                spt.del(a,a+b-1);
            }else if(op[0]=='M'&&op[2]=='K') { //修改
                scanf("%d%d%d",&a,&b,&c);
                spt.change(a,a+b-1,c);
            }else if(op[0]=='R') { //翻转
                scanf("%d%d",&a,&b);
                spt.flip(a,a+b-1);
            }else if(op[0]=='G') { //sum
                scanf("%d%d",&a,&b);
                int ret=spt.get_sum(a,a+b-1);
                printf("%d\n",ret);
            }else if(op[0]=='M') { //max-sum
                int ret=spt.max_sum();
                printf("%d\n",ret);
            }
```

```
        }
    }
}
\*------------------------------------------------------*/

/**********************************************************************\
```

KD 树

```
\**********************************************************************/
```

模板

```
struct kdnode {
   int l,r,p,split,x[2];
   bool flg; //标记
}kdtree[N],q[N];
int n,m,D,root,splitD;
double avg[2],var[2];
bool operator==(const kdnode &a,const kdnode &b) {
    for(int i=0;i<D;i++)
        if(a.x[i]!=b.x[i]) return 0;
    return 1;
}
bool cmp(kdnode a,kdnode b) {
    return a.x[splitD]<b.x[splitD];
}
void calAvg(int l,int r) {
    for(int i=0;i<D;i++) avg[i]=0;
    for(int i=l;i<=r;i++)
        for(int j=0;j<D;j++)
            avg[j]+=1.0*kdtree[i].x[j]/(r-l+1);
}
void calVar(int l,int r) {
    for(int i=0;i<D;i++) var[i]=0;
    for(int i=l;i<=r;i++)
        for(int j=0;j<D;j++)

var[j]+=(kdtree[i].x[j]-avg[j])/n*(kdtree[i].x[j]-avg[j]);
}
int construct(int p,int l,int r) {
    if(r<l) return -1;
    int root=(l+r)/2;
    calAvg(l,r); calVar(l,r);
    double maxVar=-1.0;
```

```
    for(int i=0;i<D;i++)
        if(var[i]>maxVar) maxVar=var[i],splitD=i;
    sort(kdtree+l,kdtree+r+1,cmp);
    kdtree[root].split=splitD;
    kdtree[root].l=construct(root,l,root-1);
    kdtree[root].r=construct(root,root+1,r);
    kdtree[root].p=p;
    return root;
}
int find(int root,kdnode x) {
    if(root==-1) return -1;
    if(x==kdtree[root]) return root;

    int d=kdtree[root].split;
    if(x.x[d]>kdtree[root].x[d]) {
        return find(kdtree[root].r,x);
    }
    else if(x.x[d]<kdtree[root].x[d]) {
        return find(kdtree[root].l,x);
    }
    else {
        int l=find(kdtree[root].l,x);
        int r=find(kdtree[root].r,x);
        return (l==-1?r:l);
    }
}
int find_min(int root,int d) {
    int ret=root;
    if(kdtree[root].l!=-1) {
        int v=find_min(kdtree[root].l,d);
        if(kdtree[ret].x[d]>kdtree[v].x[d]) ret=v;
    }
    if(kdtree[root].r!=-1) {
        int v=find_min(kdtree[root].r,d);
        if(kdtree[ret].x[d]>kdtree[v].x[d]) ret=v;
    }
    return ret;
}
int find_max(int root,int d) {
    int ret=root;
    if(kdtree[root].l!=-1) {
        int v=find_max(kdtree[root].l,d);
        if(kdtree[ret].x[d]<kdtree[v].x[d]) ret=v;
    }
```

```
    if(kdtree[root].r!=-1) {
        int v=find_max(kdtree[root].r,d);
        if(kdtree[ret].x[d]<kdtree[v].x[d]) ret=v;
    }
    return ret;
}
void del_node(int v) {
    int p=kdtree[v].p;
    kdtree[v].p=-1;
    if(kdtree[p].l==v) kdtree[p].l=-1;
    else kdtree[p].r=-1;
}
void remove(int root,kdnode x) {
    int pos=find(root,x);
    if(kdtree[pos].l==-1&&kdtree[pos].r==-1) {
        del_node(pos);
    }
    else if(kdtree[pos].l==-1) {
        int alt=find_min(kdtree[pos].r,kdtree[pos].split);
        for(int i=0;i<D;i++) kdtree[pos].x[i]=kdtree[alt].x[i];
        remove(alt,kdtree[alt]);
    }
    else {
        int alt=find_max(kdtree[pos].l,kdtree[pos].split);
        for(int i=0;i<D;i++) kdtree[pos].x[i]=kdtree[alt].x[i];
        remove(alt,kdtree[alt]);
    }
}
void insert(int root,int x) {
    int d=kdtree[root].split;
    if(kdtree[root].x[d]<kdtree[x].x[d]) {
        if(kdtree[root].r==-1) {
            kdtree[root].r=x;
            kdtree[x].p=root;
        }
        else insert(kdtree[root].r,x);
    }
    else {
        if(kdtree[root].l==-1) {
            kdtree[root].l=x;
            kdtree[x].p=root;
        }
        else insert(kdtree[root].l,x);
    }
```

```
}
void add(int root,kdnode x) {
    int pos=n;
    kdtree[n++]=x;
    insert(root,pos);
}
ll dist(kdnode a,kdnode b) {
    ll ret=0;
    for(int i=0;i<D;i++)
        ret+=(ll)(a.x[i]-b.x[i])*(ll)(a.x[i]-b.x[i]);
    return ret;
}
int idx; ll ds; //下标和最小距离的平方
void query(int root,kdnode x) {
    if(root==-1) return;

    ll dss=dist(kdtree[root],x);
    if(ds>dss) idx=root,ds=dss;

    int d=kdtree[root].split;
    if(x.x[d]<kdtree[root].x[d]) {
        query(kdtree[root].l,x);
        double dd=1.0*x.x[d]+sqrt(1.0*ds);
        if(dd>=1.0*kdtree[root].x[d]) {
            query(kdtree[root].r,x);
        }
    }
    else if(x.x[d]>kdtree[root].x[d]) {
        query(kdtree[root].r,x);
        double dd=1.0*x.x[d]-sqrt(1.0*ds);
        if(dd<=1.0*kdtree[root].x[d]) {
            query(kdtree[root].l,x);
        }
    }
    else {
        query(kdtree[root].l,x);
        query(kdtree[root].r,x);
    }
}

int main()
{
    int i,j,t,cas=0;
    scanf("%d",&t);
```

```
    while(t--) {
        scanf("%d",&n); D=2;
        for(i=0;i<n;i++) {
            kdtree[i].split=0;
            kdtree[i].p=kdtree[i].l=kdtree[i].r=-1;
            for(j=0;j<D;j++) scanf("%d",&kdtree[i].x[j]);
            q[i]=kdtree[i];
        }

        root=construct(-1,0,n-1);

        m=n;
        for(i=0;i<m;i++) {
            remove(root,q[i]);
            idx=root; ds=dist(kdtree[root],q[i]);
            query(root,q[i]);
            add(root,q[i]);
            printf("%I64d\n",ds);
        }
    }
    return 0;
}
```

==========

最近点对问题

```
/*
    求每个点的最近点距离
    D表示维数
    flg等价于删点标记
*/
struct kdnode {
    int l,r,x[2]; //维数
    bool flg; //标记
}kdt[N],pt;
int cnt,D,rt;
int n,m;
void init() { D=2; cnt=0; rt=-1; }
int add(kdnode pt) {
    kdt[cnt].flg=0;
    kdt[cnt].l=kdt[cnt].r=-1;
    for(int i=0;i<D;i++) kdt[cnt].x[i]=pt.x[i];
    return cnt++;
}
void insert(int rt,kdnode pt,int d) {
    d%=D;
```

```
    if(pt.x[d]<kdt[rt].x[d]) {
        if(kdt[rt].l==-1) kdt[rt].l=add(pt);
        else insert(kdt[rt].l,pt,d+1);
    }else {
        if(kdt[rt].r==-1) kdt[rt].r=add(pt);
        else insert(kdt[rt].r,pt,d+1);
    }
}
int dist(kdnode a,kdnode b) {
    int ret=0;
    for(int i=0;i<D;i++)
        ret+=(a.x[i]-b.x[i])*(a.x[i]-b.x[i]);
    return ret;
}
int idx,ds; //下标和最近距离
void query(int rt,kdnode pt,int d) {
    d%=D;
    if(rt==-1) return;
    if(!kdt[rt].flg) {
        int dss=dist(kdt[rt],pt);
        if(ds>dss) idx=rt,ds=dss;
    }
    if(pt.x[d]<=kdt[rt].x[d]) {
        query(kdt[rt].l,pt,d+1);
        int dd=(kdt[rt].x[d]-pt.x[d])*(kdt[rt].x[d]-pt.x[d]);
        if(dd<ds) query(kdt[rt].r,pt,d+1);
    }
    if(pt.x[d]>=kdt[rt].x[d]) {
        query(kdt[rt].r,pt,d+1);
        int dd=(kdt[rt].x[d]-pt.x[d])*(kdt[rt].x[d]-pt.x[d]);
        if(dd<ds) query(kdt[rt].l,pt,d+1);
    }
}
int main() {
    int i,j;
    while(scanf("%d",&n)!=EOF) {
        init();
        for(i=0;i<n;i++) {
            for(j=0;j<D;j++) scanf("%d",&pt.x[j]);
            if(rt==-1) rt=add(pt);
            else insert(rt,pt,0);
        }
        for(i=0;i<n;i++) {
            idx=rt; ds=inf;
```

```
            kdt[i].flg=1;
            query(rt,kdt[i],0);
            kdt[i].flg=0;
            printf("%d\n",ds);
        }
    }
}
```

==========

KDT 查询区间

```
/*
    查询区间内有多少个点.
*/
struct kdnode {
    int l,r,x[5];
}kdt[N];
int cnt,D,rt;
int n,m;
int ret;
int pt[5],pt1[5],pt2[5];
void init() { cnt=0; rt=-1; }
int add(int pt[]) { //插入
    kdt[cnt].l=kdt[cnt].r=-1;
    for(int i=0;i<D;i++) kdt[cnt].x[i]=pt[i];
    return cnt++;
}
void insert(int rt,int pt[],int d) { //插入
    d%=D;
    if(pt[d]<kdt[rt].x[d]) {
        if(kdt[rt].l==-1) kdt[rt].l=add(pt);
        else insert(kdt[rt].l,pt,d+1);
    }else {
        if(kdt[rt].r==-1) kdt[rt].r=add(pt);
        else insert(kdt[rt].r,pt,d+1);
    }
}
void query(int rt,int pt1[],int pt2[],int d) {
    d%=D;
    if(rt==-1) return;
    int i;
    for(i=0;i<D;i++) //判断
        if(kdt[rt].x[i]<pt1[i]||kdt[rt].x[i]>pt2[i]) break;
    if(i==D) ret++; //在区间内
    if(pt2[d]<kdt[rt].x[d]) { //全部在左边
        query(kdt[rt].l,pt1,pt2,d+1);
```

```
    }else if(pt1[d]>=kdt[rt].x[d]) { //全部在右边
        query(kdt[rt].r,pt1,pt2,d+1);
    }else { //横跨两边
        query(kdt[rt].l,pt1,pt2,d+1);
        query(kdt[rt].r,pt1,pt2,d+1);
    }
}
int main() {
    int i,j; char op[5];
    while(scanf("%d%d",&D,&m)!=EOF) {
        init();
        while(m--) {
            scanf("%s",op);
            if(op[0]=='I') { //增加点
                for(j=0;j<D;j++) scanf("%d",&pt[j]);
                if(rt==-1) rt=add(pt);
                else insert(rt,pt,0);
            }else { //查询区域内节点个数
                for(j=0;j<D;j++) scanf("%d",&pt1[j]);
                for(j=0;j<D;j++) scanf("%d",&pt2[j]);
                ret=0;
                query(rt,pt1,pt2,0);
                printf("%d\n",ret);
            }
        }
    }
}
\*--------------------------------------------------------*/
```

/**\

块状链表

```
/*
    1 区间修改成某个值
    2 区间查找是否存在某个值
*/
const int mod=1007;
struct Block {
    int l,r,c; //左右端点,c为lazy标记
    int hash[1007]; //整数hash
    int num[1007]; //每个值对于的个数
    void init() { //初始化
        memset(hash,-1,sizeof(hash));
```

```
        memset(num,0,sizeof(num));
    }
    int HASH(int x) {
        int p=x%mod;
        while(hash[p]!=-1&&hash[p]!=x) {
            p++;
            if(p==mod) p=0;
        }
        return p;
    }
    void insert(int c) { //添加一个值为c
        int p=HASH(c);
        if(hash[p]==-1) hash[p]=c;
        num[p]++;
    }
    void remove(int c) { //删除一个值为c
        int p=HASH(c);
        num[p]--;
        if(num[p]==0) hash[p]=-1,num[p]=0;
    }
    int find(int c) { //值为c的个数
        int p=HASH(c);
        return num[p];
    }
}blk[350];
int n,m,sqr,cnt;
int col[N];
void lazy(int d) { //lazy标记
    if(blk[d].c!=-1) {
        blk[d].init();
        for(int i=blk[d].l;i<blk[d].r;i++) {
            col[i]=blk[d].c;
            blk[d].insert(col[i]);
        }
        blk[d].c=-1;
    }
}
void build() { //初始化
    sqr=sqrt(n+eps); //每个块大小
    cnt=(n-1)/sqr; //块数
    for(int i=0;i<=cnt;i++) {
        blk[i].l=sqr*i;
        blk[i].r=min(n-1,sqr*(i+1));
        blk[i].c=-1;
```

```
        blk[i].init();
    }
    for(int i=0;i<n;i++) { //插入
        int d=i/sqr;
        blk[d].insert(col[i]);
    }
}
void update(int l,int r,int c) { //[l,r]都变成c
    int i,p;
    int d1=l/sqr,d2=r/sqr;
    lazy(d1); lazy(d2);
    if(d1==d2) { //一段
        for(i=l;i<=r;i++) {
            blk[d1].remove(col[i]);
            col[i]=c;
            blk[d1].insert(col[i]);
        }
    }else { //多段
        for(i=d1+1;i<d2;i++) blk[i].c=c;
        for(i=l;i<blk[d1].r;i++) { //左端
            blk[d1].remove(col[i]);
            col[i]=c;
            blk[d1].insert(col[i]);
        }
        for(i=blk[d2].l;i<=r;i++) { //右端
            blk[d2].remove(col[i]);
            col[i]=c;
            blk[d2].insert(col[i]);
        }
    }
}
int query(int l,int r,int c) { //[l,r]有多少个值为c
    int i,p,ret=0;
    int d1=l/sqr,d2=r/sqr;
    if(d1==d2) { //一段
        if(blk[d1].c!=-1) {
            if(blk[d1].c==c) ret=r-l+1;
        }else {
            for(i=l;i<=r;i++) ret+=(col[i]==c);
        }
    }else { //多段
        for(i=d1+1;i<d2;i++) {
            if(blk[i].c!=-1) {
                if(blk[i].c==c) ret+=sqr;
```

```
            }else {
                ret+=blk[i].find(c);
            }
        }
        if(blk[d1].c!=-1) { //左块
            if(blk[d1].c==c) ret+=blk[d1].r-l;
        }else {
            for(i=l;i<blk[d1].r;i++) ret+=(col[i]==c);
        }
        if(blk[d2].c!=-1) { //右块
            if(blk[d2].c==c) ret+=r-blk[d2].l+1;
        }else {
            for(i=blk[d2].l;i<=r;i++) ret+=(col[i]==c);
        }
    }
    return ret;
}
int main() {
    int i,op,a,b,z;
    while(scanf("%d%d",&n,&m)!=EOF) {
        for(i=0;i<n;i++) scanf("%d",&col[i]);
        build();
        while(m--) {
            scanf("%d%d%d%d",&op,&a,&b,&z);
            if(op==1) {
                update(a,b,z);
            }else {
                int ret=query(a,b,z);
                printf("%d\n",ret);
            }
        }
    }
}
\*-------------------------------------------------------*/
```

```
/*************************************************************\
```

动态树

```
/*
    1.合并a,b子树
    2.以a为根,删除b与父亲的边,分成两个子树
    3.更新a->b路径权值加c
    4.询问a->b路径最大权值
```

```
*/
struct LCT {
    void Rotate(int x,int f) {
        int y=pre[x],z=pre[y];
        push_down(x);
        ch[y][!f]=ch[x][f]; pre[ch[x][f]]=y;
        ch[x][f]=y; pre[y]=x;
        pre[x]=z;
        if(z) ch[z][ch[z][1]==y]=x;
        push_up(y);
    }
    void Splay(int x) {
        int rt=x;
        while(pre[rt]) rt=pre[rt];
        if(rt!=x) {
            bef[x]=bef[rt];
            bef[rt]=0;
            while(pre[x]) {
                push_down(pre[x]);
                int y=pre[x];
                int f=(ch[y][0]==x);
                Rotate(x,f);
            }
            push_up(x);
        }else push_down(x);
    }
    void Link(int x) {
        int y=ch[pre[x]][1];
        bef[y]=0;
        ch[pre[x]][1]=x;
        bef[x]=1;
    }
    void Access(int x) {
        int y=0;
        while(x) {
            Splay(x);
            bef[ch[x][1]]=x; pre[ch[x][1]]=0;
            bef[y]=0; pre[y]=x;
            ch[x][1]=y;
            push_up(x);
            y=x; x=bef[x];
        }
    }
    int Getroot(int x) {
```

```
    Access(x),Splay(x);
    while(ch[x][0]) x=ch[x][0];
    return x;
}
void MakeRoot(int x){
    Access(x),Splay(x);
    rev[x]^=1;
}
void Merge(int x,int y) {
    if(Getroot(x)==Getroot(y)) { puts("-1"); return; }
    MakeRoot(x),MakeRoot(y);
    bef[x]=y;
}
void Cut(int x,int y) {
    if(x==y||Getroot(x)!=Getroot(y)) { puts("-1"); return; }
    MakeRoot(x);
    Access(y),Splay(y);
    bef[ch[y][0]]=bef[y]; bef[y]=0;
    pre[ch[y][0]]=0; ch[y][0]=0;
    push_up(y);
}
//------------------------------------------------------------
void update_add(int x,int v) {
    if(!x) return;
    val[x]+=v; mx[x]+=v;
    lazy[x]+=v;
}
void push_up(int x) {
    mx[x]=max(val[x],max(mx[ch[x][0]],mx[ch[x][1]]));
}
void push_down(int x) {
    if(lazy[x]) {
        update_add(ch[x][0],lazy[x]);
        update_add(ch[x][1],lazy[x]);
        lazy[x]=0;
    }
    if(rev[x]) {
        rev[ch[x][0]]^=1; rev[ch[x][1]]^=1;
        swap(ch[x][0],ch[x][1]);
        rev[x]=0;
    }
}
void init() {
    memset(ch,0,sizeof(ch));
```

```
        memset(pre,0,sizeof(pre));
        rev[0]=rev[1]=0;
        lazy[0]=lazy[1]=0;
        bef[0]=bef[1]=0;
        val[0]=mx[0]=0;
    }
    void update(int x,int y,int v) {
        if(Getroot(x)!=Getroot(y)) { puts("-1"); return; }
        Access(y); y=0;
        while(x) {
            Splay(x);
            if(!bef[x]) {
                val[x]+=v;
                update_add(y,v);
                update_add(ch[x][1],v);
                return;
            }
            bef[ch[x][1]]=0; ch[x][1]=y;
            bef[y]=0; pre[y]=x;
            push_up(x);
            y=x; x=bef[x];
        }
    }
    int query(int x,int y) {
        if(Getroot(x)!=Getroot(y)) return -1;
        Access(y); y=0;
        while(x) {
            Splay(x);
            if(!bef[x])
                return max(val[x],max(mx[ch[x][1]],mx[y]));
            bef[ch[x][1]]=0; ch[x][1]=y;
            bef[y]=0; pre[y]=x;
            push_up(x);
            y=x; x=bef[x];
        }
    }
    int ch[N][2];
    int pre[N];
    int bef[N];
    int val[N];
    int mx[N];
    int lazy[N];
    int rev[N];
}lct;
```

```
void dfs(int u,int f){ //生成树
    for(int i=p[u];i!=-1;i=e[i].nxt) {
        int v=e[i].b;
        if(v==f) continue;
        lct.bef[v]=u;
        dfs(v,u);
    }
}
int main() {
    int i,j,op,a,b,c;
    while(scanf("%d",&n)!=EOF) {
        init(); lct.init();
        for(int i=1;i<n;i++) {
            scanf("%d%d",&a,&b);
            addedge(a,b);
        }
        for(int i=1;i<=n;i++) scanf("%d",&lct.val[i]);
        dfs(1,0);
        scanf("%d",&m);
        while(m--) {
            scanf("%d",&op);
            if(op==1) {
                scanf("%d%d",&a,&b);
                lct.Merge(a,b);
            } else if(op==2) {
                scanf("%d%d",&a,&b);
                lct.Cut(a,b);
            } else if(op==3) {
                scanf("%d%d%d",&c,&a,&b);
                lct.update(a,b,c);
            } else {
                scanf("%d%d",&a,&b);
                printf("%d\n",lct.query(a,b));
            }
        }
    }
}
\*-------------------------------------------------------*/

/*********************************************************\
```

树同构(树的最小表示法)

```
/* BY 菊花 */
```

```
//判断树同构
//树的最小表示法
//我们先用DFS构造出这棵树,在构造的途中,记录每个节点其儿子的括号序列
//(为左括号,为右括号)
//(()(()))类似于这种
//如果最小表示法序列相同则两棵树同构
//对于无根树,只要确定一个根,就转换成有根树同构的判断了
//将树看成无向图,进行拓扑排序(每次删除度为的点),最后剩余或个点.
//如果是个点,以该点为根建立有根树
//如果是个点,一棵树以任一点为根建树,另一棵树分别以个点建立树,判断两次.
#define maxm 1000000
#define maxn 1011
struct TREEMIN {
   string ans;
   struct EDGE {
      int v,next;
   }E[maxm];
   int head[maxn];
   int num;
   int NV;
   void addedge(int u,int v) {
      E[num].v=v;E[num].next=head[u];head[u]=num++;
      E[num].v=u;E[num].next=head[v];head[v]=num++;
   }
   void init(int n) {
      NV=n;
      memset(head,-1,sizeof(head[0])*NV);
      num=0;
   }
   string dfs(int s,int pre) {
      vector<string>temp;
      string ret="0";
      for(int i=head[s];i!=-1;i=E[i].next) {
         int v=E[i].v;
         if(v==pre) continue;
         temp.push_back(dfs(v,s));
      }
      sort(temp.begin(),temp.end());
      for(int i=0;i<temp.size();i++) ret=ret+temp[i];
      ret+="1";
      return ret;
   }
   string work(int s) {
      //s is the root
```

```
      return dfs(s,-1);
   }
}G;
\*-------------------------------------------------------*/
```

```
/*************************************************************\
```

动态规划

```
\*************************************************************/
```

树形背包

```
int dp[N][M];
void dfs(int u) {
    for(int i=p[u];i!=-1;i=e[i].nxt) {
        int v=e[i].b;
        int c=e[i].c;
        for(int j=0;j<=m;j++) dp[v][j]=dp[u][j];
        dfs(v);
        for(int j=c;j<=m;j++)
            if(dp[v][j-c]+pt[v].w>dp[u][j]) dp[u][j]=dp[v][j-c]+pt[v].w;
    }
}
```

```
\*-------------------------------------------------------*/
```

①单调栈

```
//求i左右延伸到最远比h[i]大的位置
l[1]=1; r[n]=n;
for(i=2;i<=n;i++) {
   k=i;
   while(k>1&&h[k-1]>=h[i]) k=l[k-1];
   l[i]=k;
}
for(i=n-1;i>=1;i--) {
   k=i;
   while(k<n&&h[k+1]>=h[i]) k=r[k+1];
   r[i]=k;
}
```

```
\*-------------------------------------------------------*/
```

②单调队列

```
//max
top=0; bot=0;
for(i=1;i<=n;i++) {
   while( top>bot && a[i]>=a[q[top-1]] ) top--;
   q[top++]=i;
   while( top>bot && i-q[bot]>=k ) bot++;
```

```
      mx[i]=a[q[bot]];
    }
//min
    top=0; bot=0;
    for(i=1;i<=n;i++) {
      while( top>bot && a[i]<=a[q[top-1]] ) top--;
      q[top++]=i;
      while( top>bot && i-q[bot]>=k ) bot++;
      mi[i]=a[q[bot]];
    }
\*-----------------------------------------------------------*/
```

③树中的最远距离

```
=========================================================================
计算每个节点的子树中,最远和次最短.
=========================================================================
int f1[N]; //子树中最大距离
int f2[N]; //子树中次大距离
int dir[N];  //记录最大距离的方向
int dfs(int u,int father) {
   for(int i=p[u];i!=-1;i=e[i].nxt) {
      int v=e[i].b;
      int w=e[i].c;
      if(v==father) continue;
      int tmp=w+dfs(v,u);
      if(tmp>f1[u]) {
         f2[u]=f1[u];
         f1[u]=tmp;
         dir[u]=v;
      }else if(tmp>f2[u]) {
         f2[u]=tmp;
      }
   }
   return f1[u];
}
void dfs2(int u,int father) {
   for(int i=p[u];i!=-1;i=e[i].nxt) {
      int v=e[i].b;
      int w=e[i].c;
      if(v==father) continue;
      int l1=f1[u]+w; //父节点最大
      int l2=f2[u]+w; //父节点次大
      if(dir[u]==v) { //v在l1上
         if(l2>f1[v]) {
            f2[v]=f1[v];
```

```
            f1[v]=l2;
            dir[v]=u;
         }else if(l2>f2[v]) {
            f2[v]=l2;
         }
      }else {//v不在l1上
         if(l1>f1[v]) {
            f2[v]=f1[v];
            f1[v]=l1;
            dir[v]=u;
         }else if(l1>f2[v]) {
            f2[v]=l1;
         }
      }
      dfs2(v,u);
   }
}
int main() {
   int i,j,b,c;
   while(scanf("%d",&n)!=EOF) {
      init();
      for(i=2;i<=n;i++) {
         scanf("%d%d",&b,&c);
         addedge(i,b,c); //无向边
      }
      memset(f1,0,sizeof(f1));
      memset(f2,0,sizeof(f2));
      memset(dir,0,sizeof(dir));
      dfs(1,1);
      dfs2(1,1);
      for(i=1;i<=n;i++) ret=max(ret,f1[i]+f2[i]);
      printf("%d\n",ret);
   }
}
\*-------------------------------------------------------*/
```

④子段划分问题

[1]最大连续子段和(最大不连续呢?)

【 dp[i] 以i结尾最大值,dp[i-1]转来 】

【 dp[i] 以i结尾最大值,dp[k] {k<i} 】a

[2]最大m段连续子段和 (各子段长度固定或者给定范围呢?)

【 dp[i][j] 划分成i段以j结尾的最大值;

dp[i][j-1]接在后面 , dp[i-1][j-1]独立成段 】

[3]子段划分

【dp[i][j] 以j结尾划分成i段的最大值;

```
      dp[i-1][k]前面k划分成i-1段 , (k+1)~j独立成为第i段 】
     for(i=0;i<=ki;i++)
       for(j=0;j<=n;j++) dp[i][j]=inf;
     dp[0][0]=0;
     for(i=1;i<=ki;i++) {//划分成i段
       for(j=i*l;j<=n;j++) { //第i段以j结尾
         for(k=(i-1)*l;k<=j-l;k++) {//前面i-1段以k结尾
           dp[i][j]=min( dp[i][j],dp[i-1][k]+(sum[j]-sum[k])*i);
         }
       }
     }
\*-------------------------------------------------------*/
```

```
/*************************************************************\
```

计算几何部分公式

```
\*************************************************************/
```

平行四边形个数

Q:给你 n 个点,求组成平行四边形的个数.

A:对每条直线将一个点移到原点.

```
\*-------------------------------------------------------*/
```

圆的最多覆盖次数

```
=====================================================================
A.半径一样的 O(N^2*logN)
=====================================================================
const double R=1.0;
struct point {
  double x,y;
}p[N];
struct alpha {
  int flg;
  double v;
  bool friend operator<(const alpha &a,const alpha &b) {
    return a.v<b.v;
  }
}b[N*2];
int n;
inline double dis(point a,point b) {
  return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}
int solve() {
  int i,j;
  int ret=0;
```

```
   for(i=0;i<n;i++) {
      int sz=0;
      for(j=0;j<n;j++) {
         if(i==j) continue;
         if(dis(p[i],p[j])>2.0*R) continue;
         //关键部分
         double d=dis(p[i],p[j]);
         double o=atan2(p[j].y-p[i].y,p[j].x-p[i].x);
         double phi=acos(d/(2.0*R));
         if(o<0) o+=2*pi;

         b[sz].v=o-phi+2*pi; b[sz++].flg=1;
         b[sz].v=o+phi+2*pi; b[sz++].flg=-1;
      }
      sort(b,b+sz);
      int num=0;
      for(j=0;j<sz;j++) {
         num+=b[j].flg;
         ret=max(ret,num);
      }
   }
   return ret+1;
}
int main() {
   int i,j,t;
   scanf("%d",&t);
   while(t--) {
      scanf("%d",&n);
      for(i=0;i<n;i++) scanf("%lf%lf",&p[i].x,&p[i].y);
      int ret=solve();
      printf("%d\n",ret);
   }
}
```

==========

B.半径不一样的 O(N^3)

==========

```
struct point {
   double x,y;
}p1,p2;
struct circle {
   point p;
   double r;
}c[N];
vector<point>v;
```

```
inline double dis(point p1,point p2) {
   return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
inline bool judge(circle c1,circle c2) {
   double d=dis(c1.p,c2.p);
   if(d<c1.r+c2.r+ep && d>fabs(c1.r-c2.r)-ep) return 1;
   return 0;
}
inline point GetPoint(point u1,point u2,point v1,point v2) {
   point ret=u1;
   double
k=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))/((u1.x-u2.x)*
(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
   ret.x+=(u2.x-u1.x)*k;
   ret.y+=(u2.y-u1.y)*k;
   return ret;
}
inline void cal(point c,double r,point l1,point l2) {
   point p=c;
   double t;
   p.x+=l1.y-l2.y;
   p.y+=l2.x-l1.x;
   p=GetPoint(p,c,l1,l2);
   t=sqrt(r*r-dis(p,c)*dis(p,c))/dis(l1,l2);
   p1.x=p.x+(l2.x-l1.x)*t;
   p1.y=p.y+(l2.y-l1.y)*t;
   p2.x=p.x-(l2.x-l1.x)*t;
   p2.y=p.x-(l2.y-l1.y)*t;
}
inline void Get(point c1,double r1,point c2,double r2) {
   point u,v;
   double t;
   t=(1+(r1*r1-r2*r2)/dis(c1,c2)/dis(c1,c2))/2;
   u.x=c1.x+(c2.x-c1.x)*t;
   u.y=c1.y+(c2.y-c1.y)*t;
   v.x=u.x+c1.y-c2.y;
   v.y=u.y-c1.x+c2.x;
   cal(c1,r1,u,v);
}
int main() {
   int n,t;
   scanf("%d",&t);
   while(t--) {
      scanf("%d",&n);
```

```
      v.clear();
      for(int i=0;i<n;i++) {
         scanf("%lf%lf",&c[i].p.x,&c[i].p.y);
         c[i].r=1.0;
         v.push_back(c[i].p);
      }
      for(int i=0;i<n;i++)
         for(int j=0;j<i;j++)
            if(judge(c[i],c[j])) {
               Get(c[i].p,c[i].r,c[j].p,c[j].r);
               v.push_back(p1); v.push_back(p2);
            }
      int ret=1;
      int sz=v.size();
      for(int i=0;i<sz;i++) {
         int num=0;
         for(int j=0;j<n;j++)
            num+=dis(v[i],c[j].p)<c[j].r+ep;
         ret=max(ret,num);
      }
      printf("%d\n",ret);
   }
   return 0;
}
=================================================================
\*----------------------------------------------------------*/
```

圆的面积并

```
const int maxn = 1005;
typedef double db;
const db EPS = 1e-6;
typedef pair<db, db> PDD;
int x[ maxn ], y[ maxn ], r[ maxn ];
int nx[ maxn ], ny[ maxn ], nr[ maxn ];
int xl[ maxn ], xr[ maxn ];
int s[ maxn ];
inline bool cmp( int a, int b) {
      if( x[ a ] - r [ a ] == x[ b ] - r [ b ] )
          return x[ a ] + r[ a ] < x[ b ] + r [ b ];
      return x[ a ] - r[ a ] < x[ b ] - r [ b ];
}
inline bool cmp0(int a, int b){return r[ a ] > r [ b ];}
int n;
int L, R;
PDD se[ maxn ];
```

```
inline db f( db v){
    int sz = 0, i, j ;
    db ret = 0.0;
    for(i = L; i < R; ++ i){
        if( v <= xl[ i ] || v >= xr[ i ] ) continue;
        j = s[ i ];
        db d = sqrt(r[ j ]- (v - x [ j ]) * (v - x[ j ]));
        se[ sz ].first = y[ j ] - d;
        se[ sz ].second = y[ j ] +    d;
        ++ sz;
    }
    sort( se, se + sz);
    for(i = 0; i < sz; ++ i){
        db nowl , nowr;
        nowl = se[ i ].first;
        nowr = se[ i ].second;
        for( j = i + 1; j < sz; ++ j) if(se[ j ].first > nowr) break;
        else nowr = max( nowr, se[ j ].second);
        ret += nowr - nowl;
        i = j - 1;
    }
    return ret;
}
#define fs(x) ((x) < 0 ? (-(x)) : (x))
inline db rsimp( db l,db m, db r, db sl, db sm, db sr,db tot){
    db m1 = (l + m) * 0.5, m2 = (m + r) * 0.5;
    db s0 = f( m1), s2 = f( m2);
    db gl = (sl + sm + s0 + s0 + s0 + s0)*(m-l), gr = (sm + sr + s2 + s2 + s2 + s2)*(r-m);
    if( fs(gl + gr - tot) < EPS) return gl + gr;
    return rsimp( l, m1, m, sl, s0, sm, gl) + rsimp( m, m2,r, sm, s2, sr, gr);
}
bool get(){
    if(1 != scanf("%d", &n)) return 0;
    int i, j = 0, k;
    for(i = 0; i < n; ++ i) scanf("%d%d%d", x + i, y + i, r + i), s[ i ] = i;
    sort( s, s + n, cmp0);
    for(i = 0; i < n; ++ i){
        for(k = 0; k < j; ++ k)
            if( (nx[k]-x[s[i]])*(nx[k]-x [s[i]]) + (ny [ k ] - y [s[i]]) *(ny [ k ] - y [s[i]])
                <= (nr[ k ] - r[ s[ i ] ]) * (nr[ k ] - r[ s[ i ] ]) ) break;
        if( k == j) {
            nx[ j ] = x[ s[ i ] ];
            ny[ j ] = y[ s[ i ] ];
            nr[ j ] = r[ s[ i ] ];
```

```
                s[ j ] = j;
                j ++;
            }
        }
        n = j;
        for(i = 0; i < n; ++ i) x[ i ] = nx[ i ], y[ i ] = ny[ i ], r[ i ] = nr[ i ];
        return 1;
}
void work(){
        sort( s, s + n, cmp) ;
        db lo, hi, ans = 0.0;
        int i, j;
        for(i=0;i<n;++i) xl[i]=x[s[i]]-r[s[i]],xr[i]=x[s[i]]+r[ s[ i ] ], r[ s[i] ] *= r[ s[i] ];
        for(i = 0; i < n; ++ i) {
            int ilo, ihi;
            ilo = xl[ i ];
            ihi = xr[ i ];
            for(j = i + 1; j < n; ++ j) {
                if( xl[ j ] > ihi ) break;
                ihi = max( ihi, xr[ j ]);
            }
            db lo = ilo;
            db hi = ihi;
            L = i;
            R = j;
            db mid = (lo + hi) * 0.5;
            db sl = f(lo), sm = f(mid), sr = f(hi);
            db tot = sl + sr + sm + sm + sm + sm;
            ans += rsimp( lo, mid , hi, sl, sm , sr, tot );
            i = j - 1;
        }
        printf("%.3f\n", ans / 6.0);
}
int main(){
      while( get() ) work();
      return 0;
}
\*-------------------------------------------------------*/
```

平行四边形覆盖所有点

```
struct point {
   double x,y;
   friend bool operator<(const point &p1,const point &p2) {
      return p1.y<p2.y||(p1.y==p2.y&&p1.x<p2.x);
   }
```

```
}pt[N],res[N];
int n,m;
double height[N];
double dot(point p1,point p2) {
   return p1.x*p2.x+p1.y*p2.y;
}
double xmult(point p1,point p2,point p0){
   return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
double dist(point p1,point p2) {
   return
sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y)+0.0);
}
double point_line_dist(point p,point l1,point l2) {
   return fabs(xmult(p,l1,l2))/dist(l1,l2);
}
bool ral(point p1,point p2,point p3) {
   return (p1.x-p2.x)*(p3.y-p2.y)-(p3.x-p2.x)*(p1.y-p2.y)<0;
}
int convexhull() {
   res[0]=pt[0];
   res[1]=pt[1];
   int top=1;
   for(int i=2;i<n;i++) {
      while(top&&ral(res[top-1],res[top],pt[i]))top--;
      res[++top]=pt[i];
   }
   int len=top;
   res[++top]=pt[n-2];
   for(int i=n-3;i>=0;i--) {
      while(top!=len&&ral(res[top-1],res[top],pt[i]))top--;
      res[++top]=pt[i];
   }
   return top;
}
int main() {
   int i,j,t,cas=0;
   point p1,p2,q1,q2,e1,e2;
   scanf("%d",&t);
   while(t--) {
      scanf("%d",&n);
      for(i=0;i<n;i++) scanf("%lf%lf",&pt[i].x,&pt[i].y);
      sort(pt,pt+n);
      int top=convexhull();
```

```
      for(i=0;i<top;i++) {
        height[i]=-1.0;
        for(j=0;j<top;j++)

height[i]=max(height[i],point_line_dist(res[j],res[i],res[(i+1)%t
op]));
      }
      double ret=9999999999.9;
      for(i=0;i<top;i++)
        for(j=i+1;j<top;j++) {
          p1=res[i]; q1=res[(i+1)%top];
          p2=res[j]; q2=res[(j+1)%top];
          e1.x=q1.x-p1.x; e1.y=q1.y-p1.y;
          e2.x=q2.x-p2.x; e2.y=q2.y-p2.y;
       double jd=sin(acos(dot(e1,e2)/sqrt(dot(e1,e1)*dot(e2,e2))));
          if(jd<ep) continue;
          double s=height[i]*height[j]/jd;
          ret=min(ret,s);
        }
      printf("Swarm %d Parallelogram Area: %.4f\n",++cas,ret);
   }
}
\*-------------------------------------------------------*/
```

叉乘

```
   bool xc(ll x1,ll y1,ll x2,ll y2,ll x3,ll y3,ll x4,ll y4) {
     ll a=(x2-x1)*(y3-y1)-(y2-y1)*(x3-x1);
     ll b=(x2-x1)*(y4-y1)-(y2-y1)*(x4-x1);
     ll c=(x4-x3)*(y1-y3)-(y4-y3)*(x1-x3);
     ll d=(x4-x3)*(y2-y3)-(y4-y3)*(x2-x3);
     if(a*b<=0&&c*d<=0) return 1;
     else return 0;
  }
\*-------------------------------------------------------*/
```

整点多边形

是指顶点的横纵坐标均为整数,由外积导出的面积计算公式可以
看出,整点多边形的面积或为整数,或为整数/2.
Pick公式:整点多边形的面积=内部整点个数+边上的整点个数/2-1.

```
\*-------------------------------------------------------*/
```

```
/*************************************************************\
```

数学

```
=============================================================
```

矩阵乘法

```
struct node {
    ll mat[4][4];
    void init() {
        memset(mat, 0, sizeof(mat));
    }
}I,ret,O;

node matri(node m1, node m2) {
    int i,j,k;
    node buf;
    buf.init();
    for(k=1;k<=3;k++)
        for(i=1;i<=3;i++) if(m1.mat[i][k])
            for(j=1;j<=3;j++) if(m2.mat[k][j])
                buf.mat[i][j]+=m1.mat[i][k]*m2.mat[k][j];
    return buf;
}

node mul(node a,ll k) {
    ret=I;
    for(;k;k>>=1) {
        if(k&1) ret=matri(ret,a);
        a=matri(a,a);
    }
    return ret;
}
```

***--*/**

欧拉函数

--

写法一

```
    LL phi[N+1];

    for(i=1;i<=N;i++)  phi[i]=i;
    for(i=2;i<=N;i+=2) phi[i]/=2;
    for(i=3;i<=N;i+=2) if(phi[i]==i)
        for(j=i;j<=N;j+=i) phi[j]=phi[j]/i*(i-1);
```

--

写法二

```
    ------------------------------------------
    欧拉函数： P(N) 表示的是小于N且与N互质的数的个数；
    互质的和： 小于N与N互质的数和 N*P(N)/2;
    ------------------------------------------
    求 gcd(n,X) >=m 的个数  (X<n)
```

因为对于x的倍数和n取最大公约数，则其值完全可能大于x。
那么可以证明出，必然在m<=k*x<=n之间
一定存在eular(n/x)个数与n的最大公约数是x
GCD(n,X) == x --> GCD(n/x , X/x) == 1
互质的个数，就是和n约数为x的X个数 !!!

```
-----------------------------------------
int eular(int n) {
    int i,ret=1;
    for(i=2;i*i<=n;i++) {
        if(n%i==0){
            n/=i;
            ret*=i-1;
            while(n%i==0) {
                n/=i;
                ret*=i;
            }
        }
    }
    if(n>1) ret=ret*(n-1);
    return ret;
}
-----------------------------------------------------------
```

写法三

```
//按照公式p=a(1-1/p1)(1-1/p2)...(1-1/pn)
//先生成质数表
for(int i=2;i<32768;i++) if(!a[i])
    for(int j=i;i*j<32768;j++) a[i*j]=1;
double p=n;
//如果i是质数并且可以被队长的编号整除的话，
//说明i和队长的编号互质，即为新朋友
//按照公式p=a(1-1/p1)(1-1/p2)...(1-1/pn)
for(int i=2;i<=n;i++)
    if(!a[i]&&n%i==0) p=p*(1.0-1.0/i);
printf("%.0lf\n",p);
\*-----------------------------------------------------*/
```

Fibo 数列通向公式

```
fibo[n]=(1/√5)*[((1+√5)/2)^n-((1-√5)/2)^n](n=1,2,3.....)
证明: log10(an)=-0.5*log10(5.0)+((double)n)*log(f)/
     log(10.0)+log10(1-((1-√5)/(1+√5))^n)
     其中f=(sqrt(5.0)+1.0)/2.0;
     (忽略最后一项) log10(1-((1-√5)/(1+√5))^n)->0
\*-----------------------------------------------------*/
```

求一个数阶乘的位数:

斯特林公式: log10(N!)+1,

```
            log(N!)=log(1)+....+log(N)
        或者  log(n!)= log(sqrt(2*pi*n))+ n*log(n/e);
            pi=acos(-1)  e = exp(1)
\*-------------------------------------------------------*/
```

求 N!中某个因子出现的个数

```
      int x=n,y=p[i];
      int sum=0;
      while(x) sum+=x/y,x/=y;
\*-------------------------------------------------------*/
```

两个数字大的进行比较[HDU1141]

用对数比较好 比如 2^x >= n! 可以取对数

```
   log2(2^x)>=log2(n!) ==> x>=log2(n!)
   ——> x>=log2(1)+log2(2)+...log2(n)
\*-------------------------------------------------------*/
```

卡特兰数

```
       for(i=1;i<=35;i++) f[i][0]=1;
       for(i=1;i<=35;i++)
          for(j=i;j<=35;j++)
             f[j][i]=f[j][i-1]+f[j-1][i];
\*-------------------------------------------------------*/
```

生成树个数

无度数限制的生成树的个数: n^(n-2) 个

有度数限制的生成树的个数:

```
        dp[i][j]=(dp[i][j]+dp[i-1][j-k]*C(k,n-2-(j-k)));
        dp[i][j]表示放完第i个数,用掉j个位置总共有多少种方法.
//THE code
const ll mod=20090829;
ll dp[N][N],c[N][N];
ll a[N],b[N];
ll C(int k,int n) {
   int i,j;
   if (c[k][n]!=0) return c[k][n];
   ll tmp,ans;
   for(i=n-k+1;i<=n;i++) a[i]=i;
   for(i=1;i<=k;i++) b[i]=i;
   for(i=1;i<=k;i++)
      for(j=n-k+1;j<=n;j++) {
         tmp=gcd(b[i],a[j]);
         b[i]/=tmp;
         a[j]/=tmp;
      }
   ans=1;
   for(i=n-k+1;i<=n;i++) ans=ans*a[i]%mod;
   c[k][n]=ans;
```

```
   return ans;
}
int main() {
   int n,d,i,j,k;
   ll ans;
   memset(c,0,sizeof(c));
   while(scanf("%d%d",&n,&d)!=EOF) {
      memset(dp,0,sizeof(dp));
      dp[0][0]=1;
      for(i=1;i<=n;i++)
         for(j=0;j<=n-2;j++)
            for(k=0;k<=min(j,d-1);k++)
               dp[i][j]=(dp[i][j]+dp[i-1][j-k]*C(k,n-2-(j-k)))%mod;

      ans=dp[n][n-2];
      printf("%I64d\n",ans);
   }
}
\*-------------------------------------------------------*/
```

分数 GCD

```
      while(scanf("%d/%d %d/%d",&a1,&a2,&b1,&b2)!=EOF) {
         int x=gcd(a2,b2);
         int d=a2/x*b2;
         a1*=b2/x; b1*=a2/x;
         int y=gcd(a1,b1);
         int z=gcd(y,d);
         y/=z; d/=z;
         printf("%d/%d\n",y,d);
      }
\*-------------------------------------------------------*/
```

因子的因子个数立方和

每个素因子个数cnt对应的2^cnt 的立方和 乘积

比如12

素因子 2 3

2有2个 ->36

3有1个 ->9

所以12 的因子的因子个数立方和就是 9*36 = 324

```
\*-------------------------------------------------------*/
```

容斥原理

```
   ll dfs(int x,int b,int now) {
      ll res=0;
      for(int i=x;i<num[now];i++)
         res+=b/p[now][i]-dfs(i+1,b/p[now][i],now);
      return res;
```

```
}
\*------------------------------------------------------*/
```

平面分割

n个平面可以将空间分割成多少块区域

```
sum=(pow(n,3)+5*n+6)/6;
\*------------------------------------------------------*/
```

计算 a^n+b^n

```
p=a+b , q=a*b
1: a+b    = a+b
2: a^2+b^2 = (a+b)*(a+b) - (a*b)*(1+1)
3: a^3+b^3 = (a^2+b^2)*(a+b) - (a*b)*(a+b)
4: a^4+b^4 = (a^3+b^3)*(a+b) - (a*b)*(a^2+b^2)
5: a^5+b^5 = (a^4+b^4)*(a+b) - (a*b)*(a^3+b^3)
...
a^n+b^n = ( a^(n-1) + b^(n-1) )*(a+b) - (a^(n-2)+b^(n-2))
|a+b,a*b| |a^(n-1)+b^(n-1) |
| 1 , 0 | |a^(n-2)+b^(n-2) |
|p -q| | a+b |
|1 0|  | 2  |
\*------------------------------------------------------*/
```

计算 1^k+2^k+...n^k

```
a^k ==> (a+1)^k
|1 0     0     0     1         |   |sum|
|0 C[0,0] 0     0     0         |   |a0 |
|0 c[1,0] c[1,1] 0     0       |   |a1 |
|0 c[2,0] c[2,1] c[2,2] 0      |   |a2 |
|0 c[3,0] c[3,1] c[3,2] c[3,3] |   |a3 |
\*------------------------------------------------------*/
```

因子和的和

```
//当gcd(a,b)=1时 s(a*b)=s(a)*s(b);
//素数因子: s(p^n)=1+p+p^2+...+p^n = (p^(n+1)-1)/(p-1)
   inline ll sum(ll a,ll b) { return (a+b)*(b-a+1)/2; }
   ll csod(ll n) {
      ll i,ret=0,m=sqrt(n+0.0);
      for(i=2;i<=m;i++) ret+=(n/i-1)*i; //O(sqrt(n))
      for(;m>=1;i=n/m+1,m--) ret+=((m-1)*sum(i,n/m));// O(sqrt(n))
      return ret;
   }
   int main() {
      int t,cas=0; ll n;
      while(scan_d(n)!=EOF) {
         ll ret=csod(n);
         printf("%lld\n",ret);
      }
```

```
    }
\*-------------------------------------------------------*/
```

乘法逆元

```
   //y为关于mod的n的逆元
   //注意mod和n要互质才有逆元!!!
      int x,y,q;
      void exgcd(int a,int b) {
          if(b==0) { x=1; y=0;q=a; return; }
          exgcd(b,a%b);
          int t=x;
          x=y; y=t-a/b*y;
      }
      exgcd(mod,n);
      while(y<0) y+=mod;
\*-------------------------------------------------------*/
```

组合数取模

```
| 求C(n+m,n) % p的值。保证p是素数
| Lucas定理:
| Lu(n,m,p)=cm(n%p,m%p)* Lu(n/p,m/p,p),Lu(x,0,p)=1;
| 对每个i求i关于p的逆元，/i改成 *那个逆元 a^(p-1) = 1 (%p),
   所以a的逆就是a^(p-2)
| 所以有: (a/b) mod p = a * b^(p-2) mod p
=======================================================
```

① n<=10^9 m<=10^6 p<=10^9 (m很小,p很大!)

```
ll mul(ll a,ll k,ll mod) {
 ll ret=1;
 for(;k;k>>=1) {
   if(k&1) ret=(ret*a)%mod;
   a=(a*a)%mod;
 }
 return ret;
}
ll cm(ll n,ll m,ll mod) {
 ll i,ans=1,a,b;
  //(a/b) mod p = a * b^(p-2) mod c
 for(i=0;i<m;i++) {
   a=(n-i)%mod; b=(m-i)%mod;
   ans=ans*( a*mul(b,mod-2,mod)%mod )%mod;
 } return ans;
}
ll lucas(ll n,ll m,ll p) {
  if(m==0) return 1;
  return ( cm(n%p,m%p,p)*lucas(n/p,m/p,p) )%p;
}
```

```
int main()
{
  ll n,m,p; int t;
  scanf("%d",&t);
  while(t--) {
    scanf("%I64d%I64d%I64d",&n,&m,&p);
    ll ans=lucas(n,m,p);
    printf("%I64d\n",ans);
  }
}
```

==========

② n<=10^9 m<=10^9 p<=10^6 (p很小,n,m很大!)

```
ll fac[100086];
ll n,m,mod;
void init () {
  fac[0]=1;
  for(ll i=1;i<=mod;i++)
   fac[i]=fac[i-1]*i%mod;
}
ll mul(ll a,ll k) {
 ll ret=1;
 for(;k;k>>=1) {
   if(k&1) ret=(ret*a)%mod;
   a=(a*a)%mod;
 }
 return ret;
}
ll cm(ll n,ll m) {
 if(m>n) return 0;
  //(a/b) mod p = a * b^(p-2) mod c
 ll res=fac[n]*mul(fac[n-m]*fac[m]%mod,mod-2)%mod;
 return res;
}
ll lucas(ll n,ll m) {
  ll ans=1,a,b;
  while(n&&m) {
     a=n%mod; b=m%mod;
     ans=ans*cm(a,b)%mod;
     n/=mod; m/=mod;
  }
  return ans;
}
int main()
{
```

```
  while(scanf("%I64d%I64d%I64d",&n,&m,&mod)!=EOF) {
    init();
    ll ans=lucas(n,m);
    printf("%I64d\n",ans);
  }
}
\*-------------------------------------------------------*/
```

Farey 序列

Fn = {a/b | gcd(a,b)=1 && 0<=a,b<=n};

即由小于或等于n的整数所组成的不可再约分数的递增序列，并满足分子分母互质。

如：

F1 = {0/1, 1/1}

F2 = {0/1, 1/2, 1/1}

F3 = {0/1, 1/3, 1/2, 2/3, 1/1}

性质

除了F1，其余Farey序列都有奇数个元素，并且中间值是1/2。

Farey序列是一个对称序列，头尾之和为1。

假如序列中有三个连续元素x1/y1,x2/y2,x3/y3,则有x2=x1+x3;y2=y1+y3;并且有x1*y2 - x2*y1 = 1。这条性质保证构造出来的分式肯定是不可约分式。

构造

从第三个性质我们可以得出它的构造方法。求N阶Farey序列：

```
Procedure make_farey(x1,y1,x2,y2 : integer)
      If x1+x2>N or y1+y2>N then Return
      make_farey(x1,y1,x1+x2,y1+y2)
      inc(total)
      farey [total] = {x1+x2,y1+y2}
        make_farey(x1+x2,y1+y2,x2,y2)
End Procedure.
```

就这么简单，当时竟然没想到，唉。

这个方法适合随机给定N，求Farey序列。

如果求出连续的F1,F2,F3,F4…Fn的话，朴素构造方法更好。

```
Procedure make_farey(n : integer)
      farey[1] = {0/1, 1/1}
    total = 2
For i=2 to n
   farey [i] = {0/1}
   For j=2 to total
      If farey [i-1][j].denominator + farey [i][j].
              denominator = N then
            farey[i] += { farey [i-1][j] + farey [i][j] }
End if
      farey [i] += {farey [i-1][j]}
End for
End for
```

End Procedure.
当时思维局限在这个构造算法上，导致超时。
所谓的Stern-Brocot树，其实已经在第一个构造算法里面隐含了。
Stern-Brocot扩展了Farey序列，它能构造出小于某个分式的所有分式，当然这些分式是无穷的。所以它能从另一方面证明素数是无穷的。
查找第K大元素
今天pku 3374 Cake Share有较多查询，得预先构造出Fn序列。
如果查询对不可预知Fn的时候，我们可以简单的改造算法1，当total达到k时输出后，立即跳出。时间复杂度O(K)。K最大就是|Fn|。
Fn的序列大小是可以递推出来的，有一个近似公式，可以让我们大致了解下Fn的大小程度。
|Fn| = 0.304*N^2。如 |F5000| ≈ 7600000，实际有7600459个。
那可以看出这个算法复杂度是O(N^2)的。
黑书上的解决方案是二分加上统计。没有详细描述。
今天比赛时，想了一下，可以如下操作：
规定另一个操作计算出小于给定分式的不可约分式数目，要求<=O(NlogN)。
再对X/N的X进行二分枚举，找出区间X/N ~ (X+1)/N。
在枚举出改区间的所有不可约分式，最多也就N个。
统计再输出答案。
总过程时间复杂度是O(N (logN)^2)的。关键就在于实现O(NlogN)的操作。

```
//PKU 3374 Accepted 30120K 248MS
const int MAX = 4000000;
int total;
int n,k;
int farey[2][MAX];
void make_farey_seq(int x1,int y1,int x2, int y2) {
    if(x1+x2 > n || y1+y2 > n) return;
    make_farey_seq(x1, y1,x1+x2, y1+y2);
    total ++;
    farey[0][total] = x1+x2;
    farey[1][total] = y1+y2;
    make_farey_seq(x1+x2, y1+y2,x2,y2);
}
int main() {
    int t;
    scanf("%d %d", &n, &t);
    if(n == 1) {
        while(t --) {
            scanf("%d", &k);
            if(k == 1) puts("0/1");
            else if(k == 2) puts("1/1");
            else puts("No Solution");
        }
        return 0;
    }
```

```
    total = 1;
    farey[0][1] = 0;
    farey[1][1] = 1;
    make_farey_seq(0,1,1,2);
    farey[0][total+1] = 1;
    farey[1][total+1] = 2;
    total ++;
    int all = 2*total;
    while(t --) {
        scanf("%d", &k);
        if(k >= all) puts("No Solution");
        else if(k <= total)
         printf("%d/%d ", farey[0][k], farey[1][k]);
        else if(k > total)
         printf("%d/%d                                                    ",
farey[1][all-k]-farey[0][all-k],farey[1][all-k]);
    }
}
\*-------------------------------------------------------*/
```

A^n%c(n 很大!)

```
| A^n ==  A^( n%Phi(c) + Phi(c) )  % c   (  n>=phi  )
| PS：一定要满足 n>=phi 等式才成立...
\*-------------------------------------------------------*/
```

最远曼哈顿距离

```
int onemaxdistance() { //一维点集最大曼哈顿距离
   int max1=MIN,min1=MAX,x,ans;
   for(int i=1;i<=n;i++) {
      scanf("%d",&x);
      if(x>max1) max1=x;
      if(x<min1) min1=x;
   }
   ans=max1-min1;
   return ans;
}
int twomaxdistance() { //二维点集最大曼哈顿距离
   int max1=MIN,min1=MAX,max2=MIN,min2=MAX,x,y,ans;
   for(int i=1;i<=n;i++) {
      scanf("%d%d",&x,&y);
      if(x+y>max1) max1=x+y;
      if(x+y<min1) min1=x+y;
      if(x-y>max2) max2=x-y;
      if(x-y<min2) min2=x-y;
   }
   ans=max1-min1;
```

```
   if(max2-min2>ans) ans=max2-min2;
   return ans;
}
int threemaxdistance() { //三维点集最大曼哈顿距离
   int
max1=MIN,min1=MAX,max2=MIN,min2=MAX,max3=MIN,min3=MAX,max4=MIN,mi
n4=MAX,x,y,z,ans;
   for(int i=1;i<=n;i++) {
      scanf("%d%d%d",&x,&y,&z);
      if(x+y+z>max1) max1=x+y+z;
      if(x+y+z<min1) min1=x+y+z;
      if(x-y+z>max2) max2=x-y+z;
      if(x-y+z<min2) min2=x-y+z;
      if(x+y-z>max3) max3=x+y-z;
      if(x+y-z<min3) min3=x+y-z;
      if(x-y-z>max4) max4=x-y-z;
      if(x-y-z<min4) min4=x-y-z;
   }
   ans=max1-min1;
   if(max2-min2>ans) ans=max2-min2;
   if(max3-min3>ans) ans=max3-min3;
   if(max4-min4>ans) ans=max4-min4;
   return ans;
}
\*-------------------------------------------------------*/
```

高效素数筛选

```
typedef vector<int> IV;
#define IsComp(n)    (_c[n>>6]&(1<<((n>>1)&31)))
#define SetComp(n) _c[n>>6]|=(1<<((n>>1)&31))
namespace Num
{
      const int MAX=100000000;
      const int LMT=10000;
      int _c[(MAX>>6)+1];
      IV prime;
      void init_prime() {
            for(int i=3;i<=LMT;i+=2) if(!IsComp(i))
                  for(int j=i*i;j<=MAX;j+=i+i) SetComp(j);
            prime.push_back(2);
            for(int i=3;i<=MAX;i+=2)
                  if(!IsComp(i)) prime.push_back(i);
      }
}
using namespace Num;
```

----------/

波兰表达式

```
/*
    也叫:前缀表达式
    波兰表达式模板
    (1) 首先构造一个运算符栈(也可放置括号)，运算符(以括号分界点)在栈内遵循越往栈顶优先级不降低的原则进行排列。
    (2)从右至左扫描中缀表达式，从右边第一个字符开始判断：
    如果当前字符是数字，则分析到数字串的结尾并将数字串直接输出。
    如果是运算符，则比较优先级。
    如果当前运算符的优先级大于等于栈顶运算符的优先级(当栈顶是括号时，直接入栈)，则将运算符直接入栈；
    否则将栈顶运算符出栈并输出，
    直到当前运算符的优先级大于等于栈顶运算符的优先级(当栈顶是括号时，直接入栈)，再将当前运算符入栈。
    如果是括号，则根据括号的方向进行处理。
    如果是右括号，则直接入栈；
    否则，遇右括号前将所有的运算符全部出栈并输出，遇右括号后将左右的两括号一起删除。
    (3) 重复上述操作(2)直至扫描结束，将栈内剩余运算符全部出栈并输出，再逆缀输出字符串。中缀表达式也就转换为前缀表达式了。
*/
bool operator>(const string a,const string b) {
    if(b=="=") return 1;
    if(b==")") return 0; //直接入栈
    if(a==")") return 0; //当前栈顶是括号,直接入栈
    if(a=="*"||a=="/") {
        if(b=="*"||b=="/") return 0; //栈顶不大于b
        else return 1; //栈顶大于b，继续弹
    }
    return 0;
}
string que[9999];  //保存后缀表达式
string stak[9999]; //保存运算符
ll stk[9999];  //保存运算结果
int qtop,stop;
void mid_to_next(string s) { //中缀转前缀表达式
    string a="",tmp;
    stop=qtop=-1;
    for(int i=(int)s.size()-1;i>=0;i--) {
        if(s[i]>='0'&&s[i]<='9') {
            int j=i,jj;
            a="";
            while(j>=0&&s[j]>='0'&&s[j]<='9') j--;
            jj=j+1;
```

```
            for(j=j+1;j<=i;j++) a+=s[j];
            que[++qtop]=a;
            i=jj;
        }else {
            a=s[i];
            if(a=="(") {
                while(stak[stop--]!=")") {
                que[++qtop]=stak[stop+1];
                }
            }else {
                while(stop!=-1) {
                    tmp=stak[stop];
                    if(tmp>a) {
                        que[++qtop]=tmp;
                        stop--;
                    }else break;
                }
                stak[++stop]=a;
            }
            a="";
        }
    }
}
ll cal() { //计算后缀表达式
    stop=-1;
    for(int i=0;i<=qtop;i++) {
        if(que[i]=="+")     stk[stop-1]=stk[stop]+stk[stop-1],stop--;
        else if(que[i]=="-") stk[stop-1]=stk[stop]-stk[stop-1],stop--;
        else if(que[i]=="*") stk[stop-1]=stk[stop]*stk[stop-1],stop--;
        else if(que[i]=="/") stk[stop-1]=stk[stop]/stk[stop-1],stop--;
        else {
            ll num=0;
            for(int j=0;j<que[i].size();j++)
                num=num*10+(que[i][j]-'0');
            stk[++stop]=num;
        }
    }
    return stk[0];
}
int main() {
    string s,ss;
    while(getline(cin,ss)) {
        if(ss=="0") break;
        s="=";
```

```
        for(i=0;i<ss.size();i++) {
            if(ss[i]==' ') continue;
            s+=ss[i];
        }
        mid_to_next(s);
        printf("%I64d\n",cal());
    }
}
```

***--*/**

GSS

```
struct node {
    int l,r,mid;
    int sum,ls,rs,ss;
}tree[N<<2];

void up(int id) {
    tree[id].sum=tree[id*2].sum+tree[id*2+1].sum;
    tree[id].ls=max(tree[id*2].ls,tree[id*2].sum+tree[id*2+1].ls);
    tree[id].rs=max(tree[id*2+1].rs,tree[id*2+1].sum+tree[id*2].rs);
    tree[id].ss=max(tree[id].ls,tree[id].rs);
    tree[id].ss=max(tree[id].ss,tree[id*2].ss);
    tree[id].ss=max(tree[id].ss,tree[id*2+1].ss);
    tree[id].ss=max(tree[id].ss,tree[id*2].rs+tree[id*2+1].ls);
}
void down(int id) {

}
node merge(node f1,node f2) {
    node f3;
    f3.sum=f1.sum+f2.sum;
    f3.ls=max(f1.ls,f1.sum+f2.ls);
    f3.rs=max(f2.rs,f2.sum+f1.rs);
    f3.ss=max(max(f3.ls,f3.rs),max(f1.ss,f2.ss));
    f3.ss=max(f3.ss,f1.rs+f2.ls);
    return f3;
}
void build(int l,int r,int id) {
    tree[id].l=l;
    tree[id].r=r;
    tree[id].mid=(l+r)>>1;
    if(l==r) {
        tree[id].ls=tree[id].rs=tree[id].ss=num[l];
        tree[id].sum=num[l];
        return;
```

```
        }
        int mid=tree[id].mid;
        build(l,mid,id*2);
        build(mid+1,r,id*2+1);
        up(id);
    }
    node query(int l,int r,int id) {
        if(tree[id].l==l&&tree[id].r==r) {
            return tree[id];
        }
        int mid=tree[id].mid;
        if(r<=mid) return query(l,r,id*2);
        else if(l>mid) return query(l,r,id*2+1);
        else {
            node fl=query(l,mid,id*2);
            node fr=query(mid+1,r,id*2+1);
            return merge(fl,fr);
        }
    }
\*----------------------------------------------------------*/
```

大数判素+最小素因子分解

```
ll mi;
ll gcd(ll x,ll y){
  if(y==0) return x;
  return gcd(y,x%y);
}
// a*a % n 农民算法
ll multi(ll a,ll k,ll n){
    ll ret=0;
    a%=n;
    for(;k;k>>=1){
      if(k&1) ret=(ret+a)%n;
        a=(a+a)%n;
    }
    return ret;
}
// a^u % n
ll mul(ll a,ll k,ll n){
    ll ret=1;
    a%=n;
    for(;k;k>>=1) {
        if(k&1) ret=multi(ret,a,n);
        a=multi(a,a,n);
    }
```

```
    return ret;
}
bool witness(ll a,ll n){
    ll u=n-1,t=0;
    ll i,x,y;
    while(!(u&1)) u>>=1,t++;
    x=mul(a,u,n);
    for(i=1;i<=t;i++) {
        y=multi(x,x,n);
        if(y==1&&x!=1&&x!=n-1) return 1;
        x=y;
    }
    if(x!=1) return 1;
    return 0;
}
// 素数判断
int test(ll n){
    ll a;
    int i;
    if(n==2) return 1;
    if(n==1 || n%2==0 ) return 0;
    srand((ll)time(0));
    for(i=1;i<=10;i++) {
        a=((ll)rand())%(n-1) + 1;
        if(witness(a, n)) return 0;
    }
    return 1;
}
//质因子分解
ll pollard_rho(ll n,ll c){
    ll x,y,d;
    ll i=1,k=2;
    srand((ll)time(0));
    x=((ll) rand())%(n-1) + 1;
    y=x;
    while(1) {
      i++;
      x=( multi(x,x,n) + c) % n;
      d=gcd(y-x+n,n);
      if(d!=1 && d!=n) return d;
      if(y==x)        return n;
      if(i==k) y=x,k<<=1;
    }
}
```

```
// dfs 查找最小素因子
void find(ll n,ll c) {
    ll r;
    if(n==1) return;
    if(test(n)) {
      if(mi>n) mi=n;
      return;
    }
    r=pollard_rho(n,c--);
    find(n/r,c);
    find(r,c);
}
int main()
{
  ll n;
  while(scanf("%I64d",&n)!=EOF){
    if(n==1||test(n)) {
       puts("is not a D_num");
       continue;
    }
    mi=(ll)1<<62;
    find(n,10007);
    ll d=n/mi;
    if( d!=1 && d!=mi && (test(d)||(d==mi*mi)) ) {
       printf("%I64d %I64d %I64d\n",mi,d,n);
       continue;
    }
    puts("is not a D_num");
  }
}
\*-------------------------------------------------------*/
```

数论结论

::::::分解质因子::::::

如果正整数n分解质因子的结果为n = p1^e1 * p2^e2 ... pr^er。

则n的约数个数为: (e1+1)*(e2+1)*...*(er+1)。

所有约数之和为:

(1 + p1 + p1^2 +...+ p1^e1)*(1 + p2 + p2^2 +...+ p2^e2)
...(1 + pr + pr^2 +...+ pr^er)

---

n 如果不能被 1-sqrt(n)的数整除的话， 肯定 n 是一个很大的素数

---

n = p1^e1 * p2^e2 * ......*pr^er

其中 p<n 的素数，那么：

n 的素因子个数 (e1 + 1) * (e2 + 1) * (e3 + 1)*......
n^2 的素因子数是 (2*e1+1) * (2*e2+1)* (2*e3+1)......

::::::因子和::::::

6的因子是1,2,3,6; 6的因子和是 s(6)=1+2+3+6=12;
20的因子是1,2,4,5,10,20; 20的因子和是 s(20)=1+2+4+5+10+20=42;
2的因子是1,2; 2的因子和是 s(2)=1+2=3;
3的因子是1,3; 3的因子和是 s(3)=1+3=4;
4的因子和是 s(4)=1+2+4=7;
5的因子和是 s(5)=1+5=6;
s(6)=s(2)*s(3)=3*4=12;
s(20)=s(4)*s(5)=7*6=42;
这是巧合吗?
再看 s(50)= 1+2+5+10+25+50=93=3*31=s(2)*s(25),s(25)=1+5+25=31.
这在数论中叫积性函数，当gcd(a,b)=1时 s(a*b)=s(a)*s(b);
如果p是素数
s(p^n)=1+p+p^2+...+p^n= (p^(n+1) - 1) / (p-1) (1)

----------/

Trick 录

是否有两个点在同一个位置
二分思想
反面思考,逆向思维(最小最大?正向逆向?合并分解?删边加边?)
单调性？对偶性？逆序数？前缀与后缀(前缀和)
利用线段树维护
sqrt(n)分块
分而治之

----------/