3800---染色

【题目大意】

给定 **N,w_i,$w_{i,j}$**，求：$Ans = \max_{A\subset[1,N]\cap Z}\left(\sum_{i\in A} w_i + \sum_{i\in A,j\in A} w_{i,j} - \sum_{i\notin A,j\notin A} w_{i,j}\right)$的值。

【方法 1】M=0 时

显然 A 是所有满足 $w_i>0$ 的 i 所组成的集合。

时间复杂度 O(N)。

期望得分 40。

```
#include<cstdio>
#include<algorithm>
#include<cstring>
using namespace std;
int N,M;
long long A[1000005],Ans;
void Init()
{   scanf("%d",&N);
    for(int i=1;i<=N;i++)
    {   scanf("%I64d",&A[i]);
        if(A[i]>=0)Ans+=A[i];
    }
    scanf("%d",&M);
    for(int i=1,U,V,W;i<=M;i++)
    {   scanf("%d%d%d",&U,&V,&W);
        if(A[U]<0&&A[V]<0)   Ans-=W;
        if(A[U]>=0&&A[V]>=0)Ans+=W;
    }
}
int main()
{   Init();
    printf("%I64d\n",Ans);
    return 0;
}
```

【方法 2】N,M≤20 时

暴力搜索。

时间复杂度 $O(2^{\min(2M,N)}M+ N)$。

结合算法一，期望得分 80。

```
#include<cstdio>
#include<algorithm>
#include<iostream>
using namespace std;
int N,M,A[1000005],U[1000005],V[1000005],W[1000005];
long long Ans;
void Init()
{   scanf("%d",&N);
    for(int i=1;i<=N;i++)
    {   scanf("%d",A+i);
        if(A[i]>=0)Ans+=A[i];//统计大于 0 的值
    }
    scanf("%d",&M);
    for(int i=1;i<=M;i++)
        scanf("%d%d%d",&U[i],&V[i],&W[i]);
}
int Choose[1000005];
void DFS(int X,long long Ans1)
{   if(X>N)
```

```
    {   for(int i=1;i<=M;i++)
        {   if(Choose[U[i]]&&Choose[V[i]])Ans1+=W[i];
            if(!Choose[U[i]]&&!Choose[V[i]])Ans1-=W[i];
        }
        if(Ans1>Ans)Ans=Ans1;
        return;
    }
    Choose[X]=0; //该点不染色
    DFS(X+1,Ans1);
    Choose[X]=1; //该点染色
    DFS(X+1,Ans1+A[X]);
}
void Work()
{   Ans=0xc0c0c0c0c0c0c0c0ll;
    if(N<=20)DFS(1,0);//分段，搜索
      else //方便处理 M=0 的情况
      {   Ans=0;
          for(int i=1;i<=N;i++)
              if(A[i]>=0)
              {   Ans+=A[i];
                  Choose[i]=1;
              }
          for(int i=1;i<=M;i++)
          {   if(Choose[U[i]]&&Choose[V[i]])Ans+=W[i];
              if(!Choose[U[i]]&&!Choose[V[i]])Ans-=W[i];
          }
      }
}
int main()
{   Init();
    Work();
    printf("%I64d\n",Ans);
    return 0;
}
```

【方法 3】转化

设法将 M>0的情况转为 M=0的情况。

可以发现，对于固定的 A，利用集合的思想可以得到：

$$\sum_{i\in A, j\in A} w_{i,j} - \sum_{i\notin A, j\notin A} w_{i,j} = -\sum_{i,j} w_{i,j} + \sum_{i\in A} w_{i,j} + \sum_{j\in A} w_{i,j}$$

也即：

$$Ans = \max_{A\subset[1,N]\cap Z} \sum_{i\in A}\left(w_i + \sum_{j} (w_{i,j} + w_{j,i})\right) - \sum_{i,j} w_{i,j}$$

因此，A 应当取所有满足 $w_i+\sum_j(w_{i,j} + w_{j,i}) >0$的 i 所组成的集合。

时间复杂度 O(N+M)。

期望得分100。

```
#include<cstdio>
#include<iostream>
#include<algorithm>
#include<cstring>
using namespace std;
int N,M;
long long A[1000005],Ans;
void Init()
{   scanf("%d",&N);
```

```
    for(int i=1;i<=N;i++)scanf("%I64d",&A[i]);
    scanf("%d",&M);
    for(int i=1,U,V,W;i<=M;i++)
    {   scanf("%d%d%d",&U,&V,&W);
        A[U]+=W;
        A[V]+=W;
        Ans-=W;
    }
}
void Work()
{
    for(int i=1;i<=N;i++)
        if(A[i]>0)Ans+=A[i];
}
int main()
{   Init();
    Work();
    printf("%I64d\n",Ans);
    return 0;
}
```

【最终理解】把边权都分别赋给每个对应的点，如果两个点都选，相当于该边多选了一次，都不选相当于该边未选，一个选一个不选相当于该边多选了一次，故对每条边 Ans-=w[i,j]。

3801---隐藏指令

【题目大意】试求在 d 维空间走 2N 步，每步往 d 个正交的方向或其反方向走一步，最总回到原点的方案数。

【方法 1】N=0 时：

直接输出 1，经济实惠。

时间复杂度 O(1)。

期望得分 5。

```
#include<cstdio>
#include<iostream>
using namespace std;
#define Mod (1000000007)
int N,M,Ans;
int main()
{   cin>>N>>M;
    if(M==0)Ans=1;
    printf("%d\n",Ans);
    return 0;
}
```

【方法 2】d=1 时

显然答案为 C(2N,N)。

时间复杂度 O(N^2)。

结合先前的算法，期望得分 30。

```
#include<cstdio>
#include<iostream>
using namespace std;
long long C[405][405];
#define Mod (1000000007)
int N,M,Ans;
```

```
void Init()
{   C[0][0]=1;
    for(int i=1;i<=400;i++)
    {   C[i][0]=1;
        for(int j=1;j<=400;j++)
            C[i][j]=(C[i-1][j]+C[i-1][j-1])%=Mod;
    }
    cin>>N>>M;
}
void Work()
{   if(M==0)Ans=1;
     else if(N==1)Ans=C[M+M][M];
}
int main()
{   Init();
    Work();
    printf("%d\n",Ans);
    return 0;
}
```

【方法 3】d =2,3 时

简单暴力搜索打表。

时间复杂度 $O(4^N)(d=2)$，$O(6^N)(d=3)$。

结合先前的算法，期望得分 45。

```
#include<cstdio>
#include<iostream>
using namespace std;
long long C[405][405];
const int Mod=1000000007;
int N,M,Ans;
void Init()
{   C[0][0]=1;
    for(int i=1;i<=400;i++)
    {   C[i][0]=1;
        for(int j=1;j<=400;j++)
            C[i][j]=(C[i-1][j]+C[i-1][j-1])%Mod;
    }
    scanf("%d%d",&N,&M);
}
void DFS2(int Depth,int X,int Y)
{   if(Depth>=M+M)
    {   if(X==0&&Y==0)Ans=(Ans+1)%Mod;
        return;
    }
    DFS2(Depth+1,X+1,Y);
    DFS2(Depth+1,X-1,Y);
    DFS2(Depth+1,X,Y+1);
    DFS2(Depth+1,X,Y-1);
}
void DFS3(int Depth,int X,int Y,int Z)
{   if(Depth>=M+M)
    {   if(X==0&&Y==0&&Z==0)Ans=(Ans+1)%Mod;
        return;
    }
    DFS3(Depth+1,X+1,Y,Z);
    DFS3(Depth+1,X-1,Y,Z);
    DFS3(Depth+1,X,Y+1,Z);
    DFS3(Depth+1,X,Y-1,Z);
```

```
    DFS3(Depth+1,X,Y,Z+1);
    DFS3(Depth+1,X,Y,Z-1);
}
void Work()
{   if(M==0)Ans=1;
    else if(N==1)Ans=C[M+M][M];
    else if(N==2)DFS2(0,0,0);
    else if(N==3)DFS3(0,0,0,0);
}
int main()
{   freopen("shiawase.in","r",stdin);
    freopen("shiawase.out","w",stdout);
    Init();
    Work();
    printf("%d\n",Ans);
    return 0;
}
```

【方法 4】

如果你继续观察答案常数表，可发现 d=2 时，答案为 $\binom{2N}{N}^2$。

该结论是正确的，稍后将会证明。

时间复杂度 O(N^2)。

结合先前的算法，期望得分 65。

```
#include<cstdio>
#include<iostream>
using namespace std;
long long C[405][405];
const int Mod=1000000007;
int N,M,Ans;
void Init()
{   C[0][0]=1;
    for(int i=1;i<=400;i++)
    {   C[i][0]=1;
        for(int j=1;j<=400;j++)
        C[i][j]=(C[i-1][j]+C[i-1][j-1])%Mod;
    }
    scanf("%d%d",&N,&M);
}
void DFS3(int Depth,int X,int Y,int Z)
{   if(Depth>=M+M)
    {   if(X==0&&Y==0&&Z==0) (Ans+=1)%=Mod;
        return;
    }
    DFS3(Depth+1,X+1,Y,Z);
    DFS3(Depth+1,X-1,Y,Z);
    DFS3(Depth+1,X,Y+1,Z);
    DFS3(Depth+1,X,Y-1,Z);
    DFS3(Depth+1,X,Y,Z+1);
    DFS3(Depth+1,X,Y,Z-1);
}
void Work()
{   if(M==0)Ans=1;
    else if(N==1)Ans=C[M+M][M];
    else if(N==2)Ans=C[M+M][M]*C[M+M][M]%Mod;
    else if(N==3)DFS3(0,0,0,0);
}
```

```
int main()
{   freopen("shiawase.in","r",stdin);
    freopen("shiawase.out","w",stdout);
    Init();
    Work();
    printf("%d\n",Ans);
    return 0;
}
```

【方法 5】：进一步分析

我们先简单的分析 d =2 的情况。

设横向的命令在指令串中出现了 2i 次，那么纵向命令就出现了 2(N−i)次，所以这种情况的方案数有$\binom{2N}{2i}\binom{2i}{i}\binom{2(N-i)}{N-i}$种。

故总答案为：

$$\begin{aligned}Ans &= \sum_{i=0}^{N}\binom{2N}{2i}\binom{2i}{i}\binom{2(N-i)}{N-i}\\ &= \sum_{i=0}^{N}\left(\frac{(2N)!}{(2i)!(2N-2i)!}*\frac{(2i)!}{i!i!}*\frac{(2N-2i)!}{(N-i)!(n-i)!}\right)\\ &= \sum_{i=0}^{N}\frac{(2N)!}{(i!)^2((N-i)!)^2}\\ &= \sum_{i=0}^{N}\frac{(2N)(2N-1)*...*(N+1)*N!}{(i!)^2((N-i)!)^2}\\ &= \frac{(2N)*...*(N+1)}{N!}*\sum_{i=0}^{N}\frac{N!*N!}{(i!)^2((N-i)!)^2}\\ &= \binom{2N}{N}\sum_{i=0}^{N}\binom{N}{i}\binom{N}{N-i}\\ &= \binom{2N}{N}^2\end{aligned}$$

d=3时也可如法炮制。

$$Ans = \sum_{i=0}^{N}\sum_{j=0}^{N-i}\binom{2N}{2i}\binom{2(N-i)}{2j}\binom{2i}{i}\binom{2j}{j}\binom{2(N-i-j)}{N-i-j}$$

时间复杂度为 O(N^2)。

结合先前的算法，期望得分75。

```
#include<cstdio>
#include<iostream>
using namespace std;
long long C[405][405];
const int Mod=1000000007;
int N,M,Ans;
void Init()
{   C[0][0]=1;
    for(int i=1;i<=400;i++)
    {   C[i][0]=1;
        for(int j=1;j<=400;j++)
        C[i][j]=(C[i-1][j]+C[i-1][j-1])%Mod;
    }
    scanf("%d%d",&N,&M);
```

```
}
void Work()
{   if(M==0)Ans=1;
    else if(N==1)Ans=C[M+M][M];
    else if(N==2)
    {   for(int i=0;i<=M;i++)
            Ans=(Ans+((C[M+M][i]*C[M+M-i][M-i])%Mod*C[M][i])%Mod)%Mod;
    }
    else if(N==3)
    {   for(int i=0;i<=M;i++)
            for(int j=0;i+j<=M;j++)
                Ans=(Ans+(((((C[M+M][i]*C[M+M-i][j])%Mod*C[M+M-i-j][M-i-j])
                        %Mod*C[M][i])%Mod*C[M-i][j]))%Mod)%Mod;
    }
}
int main()
{   freopen("shiawase.in","r",stdin);
    freopen("shiawase.out","w",stdout);
    Init();
    Work();
    printf("%d\n",Ans);
    return 0;
}
```

【方法6】DP

d>3时算法3仍然正确，但效率低下。

容易发现，算法3的本质是，枚举分配求和。

这部分枚举可用动态规划加快效率。

设所有可行的合法串中，前 i 个维度的命令只出现了2j 次的串的数量为 F[i,j]，则：

$$F[i,j]=\sum_{k=0}^{j}\binom{2j}{2k}F[i-1,j-k]\binom{2k}{k},\qquad i>0$$

$$F[0,0]=1$$

$$F[0,j]=0,\qquad j>0$$

时间复杂度 $O(dN^2)$。

期望得分100。

```
#include<cstdio>
#include<iostream>
using namespace std;
long long C[405][405],F[205];
const int Mod=1000000007;
int N,M,Ans;
void Init()
{   C[0][0]=1;
    for(int i=1;i<=400;i++)
    {   C[i][0]=1;
        for(int j=1;j<=400;j++)
            (C[i][j]=C[i-1][j]+C[i-1][j-1])%=Mod;
    }
    scanf("%d%d",&N,&M);
}
void Work()
{   F[0]=1;
    for(int i=1;i<=N;i++)//阶段：前 i 个维度
    {   for(int j=M;j>=0;j--)//状态：分的资源 j
            for(int k=1;k<=j;k++)//决策：第 i 个维度分的资源 k
```

```
            F[j]=(F[j]+F[j-k]*C[j+j][k+k]%Mod*C[k+k][k]%Mod)%Mod;
    }
    Ans=F[M];
}
int main()
{   freopen("shiawase.in","r",stdin);
    freopen("shiawase.out","w",stdout);
    Init();
    Work();
    printf("%d\n",Ans);
    return 0;
}
```

3802---图形变换

【题目分析】

看到本题的第一个思路自然是模拟，平移和缩放都是很简单的操作，只需要介绍下旋转。我们知道点绕原点逆时针旋转θ度的公式为：

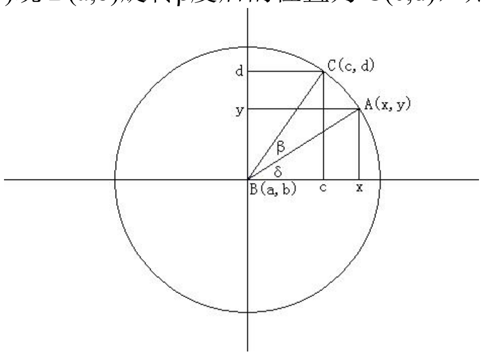
$$\begin{cases} x' = x\cos\theta - y\sin\theta \\ y' = y\cos\theta + x\sin\theta \end{cases}$$

在网上看到有一个平面内坐标点的旋转公式：

$$\begin{cases} x' = x\cos\theta - y\sin\theta \\ y' = y\cos\theta + x\sin\theta \end{cases}$$

其中 x，y 表示物体相对于旋转点旋转 angle 的角度之前的坐标，x1，y1表示物体旋转θ后相对于旋转点的坐标。

从数学上来说，此公式可以用来计算某个点绕另外一点旋转一定角度后的坐标，例如：A(x,y)绕 B(a,b)旋转β度后的位置为 C(c,d)，则 x,y,a,b,β,c,d 有如下关系式：



1.设 A 点旋转前的角度为δ，则旋转(逆时针)到 C 点后角度为δ+β

2.求 A，B 两点的距离：dist1=|AB|=y/sin(δ)=x/cos(δ)

3.求 C，B 两点的距离：dist2=|CB|=d/sin(δ+β)=c/cos(δ+β)

4.显然 dist1=dist2，设 dist1=r 所以：

r=x/cos(δ)=y/sin(δ)=d/sin(δ+β)=c/cos(δ+β)

5.由三角函数两角和差公式知：

sin(δ+β)=sin(δ)cos(β)+cos(δ)sin(β)

cos(δ+β)=cos(δ)cos(β)-sin(δ)sin(β)

所以得出：

c=r*cos(δ+β)=r*cos(δ)cos(β)-r*sin(δ)sin(β)=xcos(β)-ysin(β)

d=r*sin(δ+β)=r*sin(δ)cos(β)+r*cos(δ)sin(β)=ycos(β)+xsin(β)

即旋转后的坐标 c，d 只与旋转前的坐标 x，y 及旋转的角度β有关

从图中可以很容易理解出 A 点旋转后的 C 点总是在圆周上运动，圆周的半径为|AB|，利用这点就可以使物体绕圆周运动，即旋转物体。

而题目要求绕给定点旋转，只需要把操作变为 3 步，平移所有点使得给定的点与原点重合，套用公式旋转，再把所有的点平移回原位。

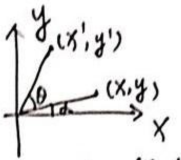
还需要注意题目中要求顺时针旋转，套用公式时把角度取负。

使用模拟的方法应该能得到 30%的分。由于循环嵌套的原因，直接模拟对于其余的数据就无能为力了。我们要思考怎样避免每次都模拟循环，一个直接的想法就是把循环作为一个整体保存下来，避免重复运行。

如何保存中间过程，我们注意到题目中的变换其实都是对于两个变量的线性递推，完全可以用矩阵来实现，这样根据矩阵的结合律，就可以把一堆变换作为整体保存。而循环多次就相当于把这个代表变换的矩阵乘几次，这样就有 60%的分数了，而且程序会更简洁（因为读入一遍指令就可以构造完矩阵，不需要考虑重复执行）。

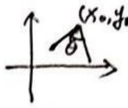
而最后 40%的数据在此基础上就容易解决了，只需要用快速幂优化一下连续矩阵乘法即可。

对于如何构造矩阵还有疑问的，请自行 google 仿射变换。



令 $r=\sqrt{x^2+y^2}$ $\begin{cases} r\cos\alpha = x \\ r\sin\alpha = y \end{cases}$

$$\begin{cases} x' = r\cos(\theta+\alpha) = r\cos\theta\sin\alpha - r\sin\theta\cos\alpha = x\cos\theta - y\sin\theta \\ y' = r\sin(\theta+\alpha) = r\sin\theta\cos\alpha + r\sin\alpha\cos\theta = x\sin\theta + y\cos\theta \end{cases}$$



对于操作 Rotate(θ, x_0, y_0)

$$\begin{cases} x' = (x-x_0)\cos\theta - (y-y_0)\sin\theta + x_0 \\ \quad = \cos\theta\, x - \sin\theta\, y + x_0 - x_0\cos\theta + y_0\sin\theta \\ y' = (y-y_0)\cos\theta + (x-x_0)\sin\theta + y_0 \\ \quad = \sin\theta\, x + \cancel{y}\cos\theta\, y + y_0 - x_0\sin\theta - y_0\cos\theta \end{cases}$$

$$[x', y', 1] = [x, y, 1] * \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ x_0 - x_0\cos\theta + y_0\sin\theta & y_0 - x_0\sin\theta - y_0\cos\theta & 1 \end{bmatrix}$$

△ Trans(x_0, y_0)

$$[x', y', 1] = [x, y, 1] * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_0 & y_0 & 1 \end{bmatrix}$$

△ Scale(x_0, y_0)

$$[x', y', 1] = [x, y, 1] * \begin{bmatrix} x_0 & 0 & 0 \\ 0 & y_0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
#include <iostream>
#include <iomanip>
#include <cstring>
#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;
const double PI=M_PI;
struct FM{double x,y,c;int z,d;}op[1020];
struct PO{double x,y;}po[120];
double rec[5][5]={0};
int n,m=0;
void ToBe_Initial()//读入数据
{    char CHAR;
     cin>>n;
     for(int i=1;i<=n;i++)cin>>po[i].x>>po[i].y;
     for(int i=1;i<=3;i++)rec[i][i]=1;
```

```
    while(cin>>CHAR)
    {   if(CHAR=='T')//平移
        {   op[++m].z=1;
            scanf("rans(%lf,%lf)",&op[m].x,&op[m].y);
        }
        if(CHAR=='S')//缩放
        {   op[++m].z=2;
            scanf("cale(%lf,%lf)",&op[m].x,&op[m].y);
        }
        if(CHAR=='R')//旋转
        {   op[++m].z=3;
            scanf("otate(%lf,%lf,%lf)",&op[m].c,&op[m].x,&op[m].y);
            op[m].c=-op[m].c/180*M_PI;//注意顺时针应该为负
        }
        if(CHAR=='L')//循环语句
        {   op[++m].z=4;
            scanf("oop(%d)",&op[m].d);
        }
        if(CHAR=='E')//结束语句
        {   op[++m].z=5;
            scanf("nd");
        }
    }
}
void ToBe_Mul(double a[][5],double b[][5])
{   double c[5][5]={0};
    for(int i=1;i<=3;i++)
        for(int j=1;j<=3;j++)
           for(int k=1;k<=3;k++)
                c[i][j]+=a[i][k]*b[k][j];
    for(int i=1;i<=3;i++)
        for(int j=1;j<=3;j++)
             a[i][j]=c[i][j];
}
void ToBe_Trans(double x,double y,double a[][5])//平移
{   double arr[5][5]={0};
    for(int i=1;i<=3;i++)arr[i][i]=1;
    arr[3][1]=x;
    arr[3][2]=y;
    ToBe_Mul(a,arr);
}
void ToBe_Scale(double x,double y,double a[][5])//缩放
{   double arr[5][5]={0};
    for(int i=1;i<=3;i++)arr[i][i]=1;
    arr[1][1]=x;
    arr[2][2]=y;
    ToBe_Mul(a,arr);
}
void ToBe_Rotat(double c,double x,double y,double a[][5])
{   double arr[5][5]={0};
    for(int i=1;i<=3;i++)arr[i][i]=1;
    arr[1][1]=cos(c);
    arr[1][2]=sin(c);
    arr[2][1]=-sin(c);
    arr[2][2]=cos(c);
    arr[3][1]=x+y*sin(c)-x*cos(c);
```

```
    arr[3][2]=y-y*cos(c)-x*sin(c);
    ToBe_Mul(a,arr);
}
void ToBe_Pow(double a[][5],int b)
{   double c[5][5]={0};
    for(int i=1;i<=3;i++)
       for(int j=1;j<=3;j++)
            c[i][j]=a[i][j];
    int bin[1020]={0};
    while(b)
    {   bin[++bin[0]]=b%2;
        b>>=1;
    }
    for(int i=bin[0]-1;i;i--)
    {   ToBe_Mul(a,a);
        if(bin[i])ToBe_Mul(a,c);
    }
}
void ToBe_Solve(int L,int R,int t,double a[][5])
{   double arr[5][5]={0};
    for(int i=1;i<=3;i++)arr[i][i]=1;
    for(int i=L;i<=R;i++)
    {   if(op[i].z==1)ToBe_Trans(op[i].x,op[i].y,arr);
        if(op[i].z==2)ToBe_Scale(op[i].x,op[i].y,arr);
        if(op[i].z==3)ToBe_Rotat(op[i].c,op[i].x,op[i].y,arr);
        if(op[i].z==4)//循环
        {   int cnt=1,k;
            for(int j=i+1;j<=m;j++)//寻找退出这个循环的位置
            {   if(op[j].z==4)cnt++;
                if(op[j].z==5)cnt--;
                if(cnt==0){k=j;break;}
            }
            ToBe_Solve(i+1,k-1,op[i].d,arr);
            i=k;
        }
    }
    ToBe_Pow(arr,t);//矩阵快速幂求 arr^t
    ToBe_Mul(a,arr);
}
void ToBe_Print()
{   for(int i=1;i<=n;i++)
    {   double x=po[i].x*rec[1][1]+po[i].y*rec[2][1]+rec[3][1];
        double y=po[i].x*rec[1][2]+po[i].y*rec[2][2]+rec[3][2];
        cout<<fixed<<setprecision(4)<<x<<" ";
        cout<<fixed<<setprecision(4)<<y<<endl;
    }
}
int main()
{   ToBe_Initial();
    ToBe_Solve(1,m,1,rec);//对于 m 条语句分别处理
    ToBe_Print();
    return 0;
}
```

【方法 2】类似于表达式求值来编写

```
#include<iostream>
#include<iomanip>
#include<cstdio>
```

```
#include<cmath>
#include<cstring>
#include<algorithm>
using namespace std;
const int Maxn=10000+7;
const double Pie=M_PI;
const int LT=3;
struct Matrix {Matrix(){memset(M,0,sizeof(M));}double M[7][7];};
double X[Maxn]={0},Y[Maxn]={0};
int N,M=0;
struct ReQuest{ double A,B,C;int t;} R[10000+7];
void Init()
{    char c;
     string s;
     cin>>N;
     for(int i=1;i<=N;i++)cin>>X[i]>>Y[i];
     while(cin>>c)
     {    M++;
          if(c=='T')//平移操作
          {    R[M].t=1;
               while(c!='s')cin>>c;
               cin>>c;
               cin>>R[M].A;
               cin>>c;
               cin>>R[M].B;
               cin>>c;
          }
          if(c=='S')//缩放操作
          {    R[M].t=2;
               while(c!='e')cin>>c;
               cin>>c;
               cin>>R[M].A;
               cin>>c;
               cin>>R[M].B;
               cin>>c;
          }
          if(c=='R')//旋转操作
          {    R[M].t=3;
               while(c!='e') cin>>c;
               cin>>c;
               cin>>R[M].A;
               cin>>c;
               cin>>R[M].B;
               cin>>c;
               cin>>R[M].C;
               cin>>c;
          }
          if(c=='L')//循环操作
          {    R[M].t=4;
               while(c!='p') cin>>c;
               cin>>c;
               cin>>R[M].A;
               cin>>c;
          }
          if(c=='E')//结束操作
          {    R[M].t=5;
               while(c!='d') cin>>c;
```

```
        }
    }
}
Matrix Mul(Matrix A,Matrix B)//矩阵乘法
{   Matrix C;
    for(int i=1;i<=LT;i++)
        for(int j=1;j<=LT;j++)
            for(int k=1;k<=LT;k++)
                C.M[i][j]+=A.M[i][k]*B.M[k][j];
    return C;
}
Matrix Power(Matrix A,int T)//快速幂
{   Matrix Ret;
    int BB[Maxn]={0};
    Ret=A;
    while(T>0)
    {   BB[++BB[0]]=T&1;
        T>>=1;
    }
    for(int i=BB[0]-1;i>0;i--)
    {   Ret=Mul(Ret,Ret);
        if(BB[i]) Ret=Mul(Ret,A);
    }
    return Ret;
}
int Find(int L,int r)
{   int t=0,i;
    for(i=r;i>=L;i--)
    {   if(R[i].t==5) t++;
        if(R[i].t==4) t--;
        if(!t) break ;
    }
    return i;
}
void Print(Matrix AAA)
{   for(int i=1;i<=LT;i++)
    {   for(int j=1;j<=LT;j++)cout<<AAA.M[i][j]<<" ";
        cout<<endl;
    }
    cout<<endl;
}
void P(double A)
{
    cout<<fixed<<setprecision(4)<<A;
}
Matrix Get(int L,int r)
{   if(L>r)
    {   Matrix ret;
        ret.M[1][1]=1;
        ret.M[2][2]=1;
        ret.M[3][3]=1;
        return ret;
    }
    if(L==r)
    {   Matrix ret;
        if(R[L].t==1)//平移操作
        {   ret.M[1][1]=ret.M[2][2]=1;
```

```
            ret.M[3][3]=1;
            ret.M[3][1]=R[L].A;
            ret.M[3][2]=R[L].B;
        }
        if(R[L].t==2)//缩放操作
        {   ret.M[1][1]=R[L].A;
            ret.M[2][2]=R[L].B;
            ret.M[3][3]=1;
        }
        if(R[L].t==3)//旋转操作
        {   double Jiao=R[L].A;
            Jiao=(360.0-Jiao);
            Jiao=Jiao/180*Pie;
            double Cos=cos(Jiao);
            double Sin=sin(Jiao);
            double x0=R[L].B;
            double y0=R[L].C;
            ret.M[1][1]=Cos;ret.M[1][2]=Sin;
            ret.M[2][1]=-Sin;ret.M[2][2]=Cos;
            ret.M[3][1]=x0-Cos*x0+Sin*y0;ret.M[3][2]=y0-Cos*y0-Sin*x0;
            ret.M[3][3]=1;
        }
        return ret;
    }
    if(R[r].t==5)//类似于表达式求值得思想
    {   int Place=Find(L,r);
        Matrix AA=Get(L,Place-1);
        Matrix BB=Get(Place+1,r-1);
        BB=Power(BB,R[Place].A);
        return Mul(AA,BB);
    }
    return Mul(Get(L,r-1),Get(r,r));
}
void Solve()
{   Matrix Fin=Get(1,M);
    for(int i=1;i<=N;i++)
    {   P(X[i]*Fin.M[1][1]+Y[i]*Fin.M[2][1]+Fin.M[3][1]);
        cout<<" ";
        P(X[i]*Fin.M[1][2]+Y[i]*Fin.M[2][2]+Fin.M[3][2]);
        cout<<endl;
    }
}
int main()
{   Init();
    Solve();
    return 0;
}
```

3803---人类基因组

【方法1】

把给出的序列复制两份构成：

A[0]、A[1]、…、A[n-1]、A[n]、A[n+1]、…、A[n+n-1](其中 A[n+i]=A[i]，i=0,1,2,…,n-1)

则循环移动 K 位后的序列就是上面序列的：A[k],A[k+1],…,A[k+n-1]。

设 sum[i]=A[0]+A[1]+...+A[i]，则循环 K 位后序列的任意前 p 项的和为：sum[k+p]-sum[k-1]；

由此可知：判断环移动 K 位后的序列任意前 p 项和都不小于0，只判断最小的 sum[k+p]

与 sum[k-1]的差大于等于0即可。

至于找最小的 sum[k+p]，可以用下列数据结构之一进行优化：

（1）线段树：时间复杂度 O(2nlog2n);

（2）优先队列：时间复杂度 O(2nlog2n);

（3）单调队列：时间复杂度 O(2n);

【单调队列】

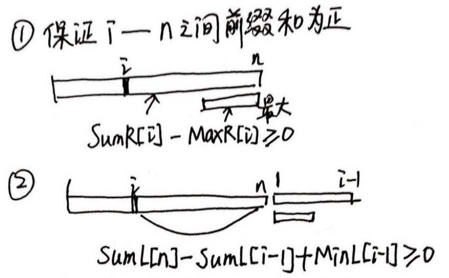
①变成两倍，求出 Sum[i]的值；

②利用单调队列求出 q[i]：表示区间[i-n+1,i]的最小值；

③for(i=n;i<=2*n;i++)if(q[i]-Sum[i-n]>=0)Ans++；统计符合要求的情况。

```
#include<iostream>
#include<cstdio>
#include<iomanip>
#include<cstring>
#include<algorithm>
#include<cmath>
using namespace std;
int n,Ans=0,a[2000005],q[2000005];
long long sum[2000005];
int main()
{   scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {   scanf("%d",&a[i]);
        a[i+n]=a[i];//变为两倍
    }
    for(int i=1;i<=2*n;i++)sum[i]=sum[i-1]+a[i];//求前缀和
    int top=1,h=0;
    q[top]=sum[1];
    for(int i=2;i<=n;i++)
    {   while(top&&sum[q[top]]>=sum[i])top--;
        q[++top]=i;
    }
    for(int i=n+1;i<=2*n;i++)
    {   while(h<=top&&(q[h]<=i-n))h++;
        while(h<=top&&sum[q[top]]>=sum[i])top--;
        q[++top]=i;
        if(sum[q[h]]-sum[i-n]>=0)Ans++;
    }
    printf("%d\n",Ans);
    return 0;
}
```

【方法2】递推




```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<cstdlib>
using namespace std;
int SumL[1000005],SumR[1000005],MinL[1000005],MaxR[1000005];
```

```
int a[1000005],n;
int main()
{   scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {   scanf("%d",&a[i]);
        SumL[i]=SumL[i-1]+a[i];//前缀和
        MinL[i]=min(MinL[i-1],SumL[i]);//前缀和最小
    }
    for(int i=n;i>=1;i--)
    {   SumR[i]=SumR[i+1]+a[i];//后缀和
        MaxR[i]=max(SumR[i],MaxR[i+1]);//后缀和最大
    }
    int Ans=0;
    for(int i=1;i<=n;i++)//枚举开始位置
      if(SumR[i]-MaxR[i]>=0&&SumL[n]-SumL[i-1]+MinL[i-1]>=0)
            Ans++;
    printf("%d\n",Ans);
    return 0;
}
```

3804---物语

【题目大意】给定图 G=(V,E)，边有边权 W。M 次操作，每次对同一条边 e, e∈E 的权值 W(e) 重新赋值，其他边不变。输出每次操作后固定的两个点 S,T 之间的最短路。

【方法 1】暴力

Floyed 时间复杂度 $O(N^3M)$，期望得分 40。

Dijkstra 时间复杂度 $O(N^2M)$，期望得分 50。

SPFA(优化的 BellMan-Ford)时间复杂度 O(k |E|M)，期望得分 60。

堆优化 Dijkstra 时间复杂度 O(NMlogN)，期望得分 75。

【方法 2】

显然，答案有三种情况。设每次修改边(u,v)

①.不经过(u,v)

②.S→u→v→T

③.S→v→u→T

如果当前要修改的边的权值为 x，d(u,v)表示 u,v 在图中的最短路，那么答案

Ans=min{d(S,T), d(S,u)+x+d(v,T), d(S,v)+x+d(u,T)}

结合堆优化 Dijkstra，时间复杂度 O(NlogN +M)。

期望得分 100。

方法 1

```
Dijkstra(1);
long long Ans1=Dis[N];
long long Ans2=Dis[U1];
long long Ans3=Dis[V1];
Dijkstra(N);
Ans2+=Dis[V1];
Ans3+=Dis[U1];
if(Ans2>Ans3)Ans2=Ans3;
long long Ans;
for(int i=1;i<=K;i++)
{   cin>>W;
    Ans=min(Ans1,Ans2+W);
    if(Ans<0x2f2f2f2f2f2f2f2fll)cout<<Ans<<endl;
          else cout<<"+Inf"<<endl;
}
```

方法 2

```
ToBe_Dijkstra();
```

```
long long Dis=dis[n];
scanf("%d%d",&x,&y);
ToBe_Edge(x,y,0);
ToBe_Edge(y,x,0);
ToBe_Dijkstra();
long long Len=dis[n];
for(int i=1;i<=k;i++)
{   scanf("%d",&D);
    if(Len==INF){cout<<"+Inf"<<endl;continue;}
    long long ans=Dis;
    if(ans>Len+D)ans=Len+D;
    printf("%lld\n",ans);
}
```

【方法 3】分层图

构建新图 G'=(V',E')以及新的边权函数 W'。

对于每个点 i∈V，将其拆分成两个点 i,i'。

对于每一条边 e∈E，若 e=(u,v)是每次都被修改的边，那么在新图 G'中加边 e'=(u,v'), e''=(u',v)，W'(e')=W'(e'')=0 否则加边 e'=(u,v)，e''=(u',v')，W'(e')=W'(e'')=W(e)。

设当前要修改的边的权值为 x，d(u,v)表示 u,v 在新图中的最短路，则答案

Ans=min{d(S,T), d(S,T')+x}

结合堆优化 Dijkstra，时间复杂度 O(NlogN+M)。

期望得分 100。

3129--消息传递

【题目分析】

【20 分算法】

对于 20%的部分数据，我们可以枚举每一个居民为根，进行一次简单 DP 即可。设 F[i]表示从 i 节点开始传遍其子树所需要花费的最少时间。贪心可得，将儿子花费时间升序排序，花费时间最多的儿子 j 肯定要最先传，次大的紧接着，以此类推。即 $F[i]=max(F[E_j]+j)$。由方程知可以在 O(N^2)的时间解决。

```
#include<iostream>
#include<cstring>
#include<cstdio>
#include<cmath>
#include<algorithm>
using namespace std;
struct Edge{int to,next;}w[6040];
int n,cnt=0,h[3020]={0};
int f[3020],g[3020]={0},vst[3020];
void AddEdge(int x,int y)
{   w[++cnt].to=y;w[cnt].next=h[x];h[x]=cnt;}
void Read()
{    int i,x;
     cin>>n;
     for(i=2;i<=n;i++)
     {   cin>>x;
         AddEdge(i,x);
         AddEdge(x,i);
     }
}
int DFS(int x)
{   int i,j,k,y,maxx=0,q[3002]={0};
    if(f[x]!=-1)return f[x];
    if(n==1){f[x]=0;return f[x];}
```

```
    for(i=h[x];i;i=w[i].next)
    {   y=w[i].to;
        if(!vst[y])
        {   vst[y]=1;
            k=DFS(y);
            q[++q[0]]=k;
        }
    }
    sort(q+1,q+q[0]+1);//从小到大排序
    for(i=1;i<=q[0];i++)//依次安排
        maxx=max(maxx,q[i]+q[0]-i+1);
    f[x]=maxx;
    return f[x];
}
void Solve()
{   int i,k,ans=0x7ffffff/2;
    for(i=1;i<=n;i++)
    {   memset(f,-1,sizeof(f));
        memset(vst,0,sizeof(vst));
        vst[i]=1;
        k=DFS(i);
        k++;
        if(ans>k){g[0]=1;g[1]=i;ans=k;}
            else if(ans==k)g[++g[0]]=i;
    }
    cout<<ans<<endl;
    for(i=1;i<=g[0];i++)cout<<g[i]<<" ";
}
int main()
{   freopen("news.in","r",stdin);
    freopen("news.out","w",stdout);
    Read();
    Solve();
    return 0;
}
```

【50 分算法】

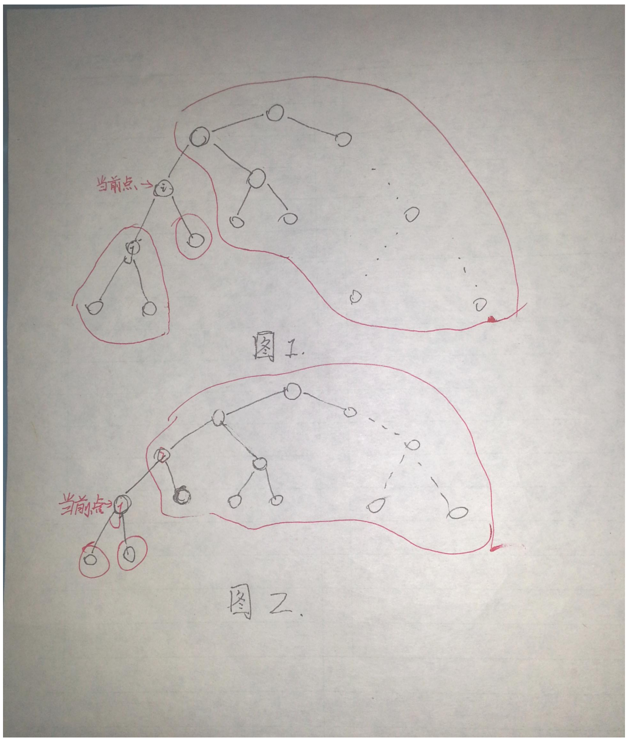
20 分算法的瓶颈在于每次需要枚举根才开始求解。这之中有着许多的冗余计算，为了提升效率和优化时间复杂度，我们不妨考虑通过按边动规的思想，设状态 F[i,j]表示（i,j）这条有向边，即从 i 到 j 要遍历完当前以 j 为根的子树需要花费的最少时间，转移方法同 20 分类似。这样每个状态可以在(DlogD)的时间内解决，D 为该点的度数。

在随机数据的前提下，该算法的时间复杂度约为 O(NlogN)。可以通过 50 分的数据。

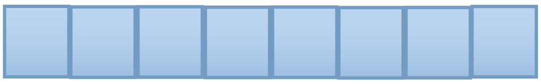
【满分算法】

继续分析得知，上述情况最坏可达到 O(N^2logN)，在人工构造的猥琐数据下明显不能出解。我们应该更换一种思维方式来解决问题了。

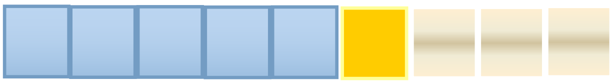
我们是否可以用对数级别的时间从一个点转移到另一个点呢？答案是显然的。观察下图。



我们发现从 i 转移到 j 时，i 的儿子又多了一个，他的值我们是可以快速计算出来的。问题是如何合并这个新的儿子与之前的儿子，这样就能快速转移答案了。



很自然的想到可以二分这个儿子所在的位置。



按照原来的顺序，蓝色方块中，第一个元素应该最先传递，第二个儿子第二个单位时间内传递，以此类推。若令新儿子所在位置为 k，可以发现第 1 到 k-1 个元素所花费的时间并未改变，而新儿子的花费时间就是 k+F[newchild]。而白色方块的时间都在原来的基础上增加了一，最大值也就是后面这部分的最大值加 1，比较这三者中的较大值就可以应付这个子问题了。

但是，我们还有一个问题没有解决，如何将当前节点的值转化为下一个当前点的儿子所对应的值呢？这等价于从该节点的儿子中删除。其处理方式类似于增加。

实现细节就交给读者。

以上问题可以通过将边表连成一张线性大表，使用 RMQ+二分进行优化处理。

至此，我们已经可以在任意极端数据下 O(nlogn)解决该题。

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
using namespace std;
struct ddm{int to,next;} e[800005];
int n,h[400005]={0},sum=0;
int v[400005]={0};
int s[400005]={0};
int road[400005],ans=0x7fffffff/2,tot=0;
int l[400005]={0},r[400005]={0};
int rmq[400005][25]={0};
void add(int x,int y)
```

```
{   sum++;e[sum].to=y;e[sum].next=h[x];h[x]=sum;}
void chushihua(int k,int fa)
//第一次任意一个点 DP 一次,用去 O(n)的时间,顺便初始化之后需要询问的 RMQ 数组;
{   int i,j,maxx=0;
    for(i=h[k];i;i=e[i].next)
        if(e[i].to!=fa)chushihua(e[i].to,k);
    l[k]=tot+1;//l,r 数组记录了每个节点的儿子所在于大数组的哪个区间:[l,r];
    for(i=h[k];i;i=e[i].next)
        if(e[i].to!=fa)s[++tot]=v[e[i].to];
    r[k]=tot;
    sort(s+l[k],s+r[k]+1);//排序以便贪心;
    for(i=l[k];i<=r[k];i++)//贪心:大的尽量先分配,小的尽量后分配,所以小的时间需要加上之前分配到几个大的的时间;
    {   maxx=max(maxx,s[i]+r[k]-i);
        rmq[i][0]=s[i]+r[k]-i;
    }
    maxx=maxx+1;
    v[k]=maxx;//记忆一下以便后用;
    return;
}
void prepare()
{   int i,j;
    for(j=1;j<=log(n)/log(2)+1;j++)
      for(i=1;i+(1<<j)-1<=n;i++)
            rmq[i][j]=max(rmq[i][j-1],rmq[i+(1<<(j-1))][j-1]);
}
int ef(int L,int r,int x)
{   int mid;
    if(s[L]>x)return L-1;
    while(L<r)
    {   mid=(L+r+1)/2;
        if(x>=s[mid])L=mid;
          else r=mid-1;
    }
    return L;
}
int find(int L,int t)
{   int j=L,ret=0,i;
    while(j<=t)
    {   i=0;
        while(j+(1<<i)-1<=t)i++;
        i--;
        ret=max(ret,rmq[j][i]);
        j=j+(1<<i);
    }
    return ret;
}
int trans(int i,int cost)
{   int x,y;
    x=ef(l[i],r[i],cost);//可插入的位置的前一个数的位置(第一个比它大的数字之前);
    y=max(find(l[i],x)+1,max(cost+r[i]-x,find(x+1,r[i])))+1;
     //x 之前的数字+1;x 之后的数字不变;x 本身为父辈传过来的权值加上后面的元素个数;
    return y;
}
int shan(int i,int cost,int dele)//预处理删除掉到下个节点的权值;
{   int x,y,s=0;
```

```
    x=ef(l[i],r[i],cost);
    y=ef(l[i],r[i],dele);
    if(x==y)
    {   s=max(find(l[i],x),find(x+1,r[i]));
        if(cost!=0)s=max(s,cost+r[i]-x);//加上被改变的值;
        return s+1;
    }
    if(x>y)
      return max(max(find(l[i],y-1),find(y+1,x)+1),max(find(x+1,r[i]),cost+r[i]-x))+1;
    if(x<y)
    {   s=max(max(find(l[i],x),find(x+1,y-1)-1),find(y+1,r[i]));
        if(cost!=0)s=max(s,cost+r[i]-x-1);
        return s+1;
    }
}
void kill(int k,int fa,int cost)//第 k 个点,父亲为 fa,处理之前需要消耗 cost 时间
{   int i,j,s;
    if(fa!=0)
    {   if(l[k]>r[k])s=cost+1;//叶子节点;
            else s=trans(k,cost);//当前节点为根的权值;
        if(s<ans)//更新答案;
        {   ans=s;
            road[0]=1;
            road[1]=k;
        }
        else if(s==ans)road[++road[0]]=k;
    }
    if(l[k]<=r[k])
      for(i=h[k];i;i=e[i].next)
        if(e[i].to!=fa)
          kill(e[i].to,k,shan(k,cost,v[e[i].to]));//递归儿子继续求解;
}
void deal()
{   int i,j;
    chushihua(1,0);//第一次任意点 DP;
    prepare();//初始化 RMQ 数组;
    ans=v[1];//初始化答案;
    road[1]=1;
    road[0]=1;//保存方案数组;
    kill(1,0,0);//开始一层一层往下处理;DOCTOR 算法;
    cout<<ans<<endl;
    sort(road+1,road+road[0]+1);//字典序排序;
    for(i=1;i<=road[0];i++)cout<<road[i]<<" ";
}
int main()
{   int x,i,j;
    scanf("%d",&n);
    for(i=2;i<=n;i++)
    {   scanf("%d",&x);
        add(i,x);
        add(x,i);
    }
    deal();
    return 0;
}
```

【优化方法】

```
#include<cstdio>
#include<cstdlib>
#include<algorithm>
#include<cstring>
#define MAXN 200005
#define INF 0x7fffffff
using namespace std;
struct son{int x,next;}r[MAXN];
int N,M;
int dp[MAXN],ans,have[MAXN],get;
int fa[MAXN],up[MAXN],down[MAXN];
int cnt[MAXN],num[MAXN],tot,tmd[MAXN];
int lmx[MAXN],rmx[MAXN];
int st[MAXN],w;
int cmp(int a, int b){return a>b;}
void add(int x,int y)
{
	r[++w].x=y;r[w].next=st[x];st[x]=w;
}
int find(int x)
{	int L=1,r=tot,mid;
	while(L<=r)
	{
		mid=(L+r)/2;
		if(num[mid]<=x)r=mid-1;
			else L=mid+1;
	}
	return L;
}
void Make_down(int x)
{   int i,tmp;
    for(i=st[x];i;i=r[i].next)
    {   tmp=r[i].x;
        Make_down(tmp);
    }
    for(i=st[x],tot=0;i;i=r[i].next)
    {   tmp=r[i].x;
        num[++tot]=down[tmp];
	}
	sort(num+1, num+1+tot, cmp);
	for(i=1,tmp=0;i<=tot;i++)
		tmp=max(tmp,num[i]+i);
	down[x]=tmp;
}
void Work(int x)
{
	int i,j,tmp,t;
	for(i=st[x], tot=0;i;i=r[i].next)
		num[++tot]=down[r[i].x];
	if(fa[x]) num[++tot]=up[x];
	sort(num+1, num+1+tot, cmp);//从大到小排序
	for(i=1,j=0;i<=tot;i++)//去掉相等的值
		if(i==1||num[i]!=num[i-1]){num[++j]=num[i];cnt[j]=1;}
			else cnt[j]++;
	for(i=1,tot=j,tmp=0;i<=tot;i++)
		{tmp+=cnt[i];tmd[i]=num[i]+tmp;}//tmd[i]第 i 个儿子的时间
```

```
    for(i=1,lmx[0]=0;i<=tot;i++)lmx[i]=max(lmx[i-1],tmd[i]);
    //lmx[i]从左到右 tmd[i]中的最大值
    for(i=tot,rmx[tot+1]=0;i>=1;i--)rmx[i]=max(rmx[i+1],tmd[i]);
    //lmx[i]从右到左 tmd[i]中的最大值
    for(i=st[x];i;i=r[i].next)//计算若 tmp 作为根时父亲传过来的时间
    {   tmp=r[i].x;
        t=find(down[tmp]);
        if(cnt[t]==1)up[tmp]=max(lmx[t-1],rmx[t+1]-1);
            else up[tmp]=max(lmx[t-1],rmx[t]-1);
    }
    dp[x]=lmx[tot];
    for(i=st[x];i;i=r[i].next)//递归处理儿子作为根的情况
        Work(r[i].x);
}
int main()
{
    int i, j;
    freopen("news.in", "r", stdin);
    freopen("news.out", "w", stdout);
    scanf("%d", &N);
    for(i=2;i<=N;i++)
    {
        scanf("%d",&fa[i]);
        add(fa[i],i);
    }
    Make_down(1);
    Work(1);
    for(i=1,ans=INF;i<=N;i++)
        if(ans>dp[i]){ans=dp[i];have[get=1]=i;}
            else if(ans==dp[i])have[++get]=i;
    printf("%d\n",ans+1);
    for(i=1;i<=get;i++)printf("%d ",have[i]);
    printf("\n");
    return 0;
}
```