Algorithm Template

Kevco

May 12, 2023

Contents

1 DP

1.1 背包 DP

1.1.1 01 背包

```
#include <iostream>

using namespace std;

const int N=1010;

int f[N];
int n,m;

int main()
{
    cin >> n >> m;
    for(int i=1;i<=n;i++)
    {
        int v,w;
        scanf("%d%d",&v,&w);
        for(int j=m;j>=v;j--)
            f[j]=max(f[j],f[j-v]+w);
    }

    cout << f[m];
}
```

1.1.2 完全背包

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N=1010;

int n,V,f[N];

int main()
{
    cin >> n >> V;
    for(int i=1;i<=n;i++)
    {
        int v,w;
        scanf("%d%d",&v,&w);
        for(int j=v;j<=V;j++)
            f[j]=max(f[j],f[j-v]+w);
    }
    cout << f[V];
    return 0;
}
```

1.1.3 多重背包

```cpp
#include <iostream>

using namespace std;

const int N=110;

int f[N][N];
int n,m;

int main()
{
    cin >> n >> m;
    for(int i=1;i<=n;i++)
    {
        int v,w,s;
        scanf("%d%d%d",&v,&w,&s);
        for(int j=1;j<=m;j++)
            for(int k=0;k<=s&&k*v<=j;k++)
                f[i][j]=max(f[i][j],f[i-1][j-k*v]+k*w);
    }
    cout << f[n][m];
}
```

```cpp
//二进制优化  O(n * m * logs)

#include <iostream>
using namespace std;
const int M=2010;

int f[M],n,m;

int main()
{
    cin >> n >> m;
    for(int i=1;i<=n;i++)
    {
        int v,w,s;
        scanf("%d%d%d",&v,&w,&s);
        int k=1;
        while(k<=s)
        {
            int V=v*k,W=w*k;
            for(int j=m;j>=V;j--)
```

```
                f[j]=max(f[j],f[j-V]+W);
            s-=k;
            k*=2;
        }
        if(s)
        {
            int V=v*s,W=w*s;
            for(int j=m;j>=V;j--)
                f[j]=max(f[j],f[j-V]+W);
        }
    }
    cout << f[m];
}


//单调队列优化  O(n * m)

#include <bits/stdc++.h>

using namespace std;

const int N=2e4+10;

int f[2][N],q[N],n,m;

int main()
{
    cin >> n >> m;
    for(int i=1;i<=n;i++)
    {
        int cur=i&1,pre=cur^1;
        int v,w,s;
        scanf("%d%d%d",&v,&w,&s);
        for(int j=0;j<v;j++)
        {
            int ff=0,rr=-1;
            for(int k=0;j+k*v<=m;k++)
            {
                if(ff<=rr&&k-q[ff]>s) ff++;

↪   while(ff<=rr&&f[pre][q[rr]*v+j]-q[rr]*w<=f[pre][k*v+j]-k*w)
↪   rr--;
                q[++rr]=k;
                f[cur][k*v+j]=f[pre][q[ff]*v+j]+(k-q[ff])*w;
```

```
            }
        }
    }

    printf("%d",f[n&1][m]);
    return 0;
}
```

1.1.4 分组背包

```
// o(n * m * s)
#include <bits/stdc++.h>

using namespace std;

const int N=110;

int f[N],n,m,s,v[N],w[N];

int main()
{
    cin >> n >> m;
    for(int i=0;i<n;i++)
    {
        scanf("%d",&s);
        for(int i=0;i<s;i++)
            scanf("%d%d",&v[i],&w[i]);

        for(int j=m;j>=0;j--)
            for(int k=0;k<s;k++)
                if(j>=v[k]) f[j]=max(f[j],f[j-v[k]]+w[k]);
    }
    cout << f[m];

    return 0;
}
```

1.1.5 树上背包

```cpp
//基于 dfs  O(n * m * m)
#include <bits/stdc++.h>

using namespace std;

const int N=110;

int f[N][N],w[N],v[N];
int root,n,m;
vector<int> e[N];


void dfs(int u)
{
    for(int i=v[u];i<=m;i++) f[u][i]=w[u];
    for(auto s : e[u])
    {
        dfs(s);
        for(int j=m;j>=v[u];j--)
            for(int k=0;k<=j-v[u];k++)
                f[u][j]=max(f[u][j],f[u][j-k]+f[s][k]);
    }
}

int main()
{
    cin >> n >> m;
    for(int i=1;i<=n;i++)
    {
        int a,b,p;
        scanf("%d%d%d",&a,&b,&p);
        v[i]=a,w[i]=b;
        if(p==-1) root=i;
        else e[p].push_back(i);
    }

    dfs(root);

    printf("%d\n",f[root][m]);

    return 0;
}
```

```cpp
/*
dfs 序写法    O(n * m)
0 是虚拟源点
*/
#include <bits/stdc++.h>

using namespace std;

const int N=310;

int f[N][N],w[N],v[N],r[N],tot,id[N];
int n,m;
vector<int> e[N];

void dfs(int u)
{
    id[++tot]=u;
    for(auto s : e[u])
        dfs(s);
    r[u]=tot;
}

int main()
{
    cin >> n >> m;
    for(int i=1;i<=n;i++)
    {
        int a,p;
        scanf("%d%d",&p,&a);
        w[i]=a;
        v[i]=1;
        if(!p) e[0].push_back(i);
        else e[p].push_back(i);
    }

    dfs(0);

    for(int i=n;i>=0;i--)
        for(int j=1;j<=m;j++)
            if(j>=v[id[i]])
↪   f[i][j]=max(f[r[id[i]]+1][j],f[i+1][j-v[id[i]]]+w[id[i]]);
            else f[i][j]=f[r[id[i]]+1][j];

    printf("%d\n",f[0][m]);
```

```
    return 0;
}
```

1.1.6 k 优背包

```
/*
求前 k 优解  O(n * m * k)
转移变成两个长为 k 的降序数组合并
*/
#include <bits/stdc++.h>

using namespace std;

const int N=5010,K=60;

int n,m,k,v[N],w[N];
int f[N][K],temp[K];

int main()
{
    cin >> k >> m >> n;
    for(int i=1;i<=n;i++) cin >> v[i] >> w[i];
    memset(f,-0x3f,sizeof f);
    f[0][0]=0;
    for(int i=1;i<=n;i++)
    {
        for(int j=m;j>=v[i];j--)
        {
            int t=0,a=0,b=0;
            while(t<k)
            {
                if(f[j][a]>f[j-v[i]][b]+w[i])
↪   temp[t++]=f[j][a++];
                else temp[t++]=f[j-v[i]][b++]+w[i];
            }
            memcpy(f[j],temp,sizeof temp);
        }
    }

    int res=0;
    for(int i=0;i<k;i++)
        if(f[m][i]) res+=f[m][i];
    printf("%d\n",res);
    return 0;
}
```

1.1.7 前后缀背包

```
/*
给定 n 个物品 m 体积背包  每个物品 有 v w 选取若干物品装背包
n 件物品，求考虑去掉某一件物品的 01 背包
定义 f[i][j] 是只考虑前 i 件物品的 01 背包   g[i][j] 是只考虑后 i 件的 01 背包
去掉第 i 件的背包就是 f[i-1][j] 和 g[n-i][j] 组合起来
可以 O(m) 时间求去掉第 i 件物品，体积不超过 v 的最大值
具体就是 dp[v] = max (f[i-1][j] + g[n-i][v-j])
O(n * m + q * m)
*/
#include <bits/stdc++.h>

using namespace std;

const int mod=1e9+7,N=5010;

int n,m,k,v[N],w[N];
int f[N][N],g[N][N];
int dp[N]; // dp[i] 为去掉第 i 个物品，体积不超过 m 的最大值


void solve()
{
    cin >> n >> m;
    for(int i=1;i<=n;i++) scanf("%d%d",v+i,w+i);

    for(int i=1;i<n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            f[i][j]=f[i-1][j];
            if(j>=v[i])
↪       f[i][j]=max(f[i][j],f[i-1][j-v[i]]+w[i]);
        }
    }

    for(int i=n;i>1;i--)
    {
        for(int j=1;j<=m;j++)
        {
            g[i][j]=g[i+1][j];
```

```
            if(j>=v[i])
↪   g[i][j]=max(g[i][j],g[i+1][j-v[i]]+w[i]);
        }
    }

    for(int i=1;i<=n;i++)
        for(int j=0;j<=m;j++)
            dp[i]=max(dp[i],f[i-1][j]+g[n-i][m-j]);
}
```

1.1.8 方案计数

```
/*
求恰好装满背包的方案数
01 背包  f[i][j]=f[i-1][j]+f[i-1][j-v]
完全背包  f[i][j]=f[i-1][j]+f[i][j-v]
多重背包  f[i][j]=f[i-1][j]+f[i-1][j-v]+...+f[i-1][j-s*v] (预处
↪  理前缀和)
*/

#include <bits/stdc++.h>

using namespace std;

const int N=110,mod=1e6+7;

int n,m;
int f[N],g[N];

int main()
{
    cin >> n >> m;

    f[0]=1,g[0]=1;
    for(int i=0,s;i<n;i++)
    {
        for(int j=1;j<=m;j++)g[j]=g[j-1]+f[j];
        cin >> s;
        for(int j=1;j<=m;j++)
        {
            if(j>s) f[j]=(g[j]-g[j-s-1])%mod;
            else f[j]=g[j]%mod;
        }
    }

    printf("%d",f[m]);
    return 0;
}
//求最优解方案数

#include <bits/stdc++.h>

using namespace std;
```

```
const int mod=1e9+7,N=1010;

int n,m,p;
int f[N],g[N];

void solve()
{
    cin >> n >> m;

    for(int i=0;i<=m;i++) g[i]=1;

    for(int i=1;i<=n;i++)
    {
        int v,w;
        scanf("%d%d",&v,&w);

        for(int j=m;j>=v;j--)
        {
            int t=f[j-v]+w;
            if(f[j]<t) f[j]=t,g[j]=g[j-v];
            else if(f[j]==t) g[j]=(g[j]+g[j-v])%mod;
        }
    }

    printf("%d\n",g[m]);
}
```

1.1.9 输出方案

```
/*
输出具体方案：保留二维，即保留中间状态，从后往前比较状态 反推转移
↪ 的路径
字典序最小 ：逆序选择物体，反推时能选则选
*/
#include <bits/stdc++.h>

using namespace std;

const int N=1010;

int f[N][N],n,m,v[N],w[N];

int main()
{
    cin >> n >> m;
    for(int i=1;i<=n;i++) scanf("%d%d",&v[i],&w[i]);

    for(int i=n;i;i--)
    {
        for(int j=0;j<=m;j++)
        {
            if(j<v[i]) f[i][j]=f[i+1][j];
            else f[i][j]=max(f[i+1][j],f[i+1][j-v[i]]+w[i]);
        }
    }

    for(int i=1;i<=n;i++)
    {
        if(m>=v[i]&&f[i][m]==f[i+1][m-v[i]]+w[i])
        {
            printf("%d ",i);
            m-=v[i];
        }
    }
    return 0;
}
```

1.1.10 其他

```
/*
01 背包 三维属性 a,b,w, 求满足总的 a>=m,b>=p 的最小重量 w
普通背包  f[i][j], 表示二维属性不超过 i,j
本题      f[i][j] 表示二维属性不小于, 因此转移有所区别
*/

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
const int mod=1e9+7,N=110;

int n,m,p;
int f[N][N];

void solve()
{
    cin >> m >> p >> n;

    for(int j=0;j<=m;j++)
        for(int k=0;k<=p;k++)
            f[j][k]=0x3f3f3f3f;

    f[0][0]=0;

    for(int i=1;i<=n;i++)
    {
        int a,b,c;
        scanf("%d%d%d",&a,&b,&c);
        for(int j=m;j>=0;j--)
            for(int k=p;k>=0;k--)

↪   f[j][k]=min(f[j][k],f[max(0,j-a)][max(0,k-b)]+c);
    }

    printf("%d\n",f[m][p]);
}
```

1.2 线性 DP

1.2.1 LIS

```cpp
// O(n * n)
#include <iostream>

using namespace std;

const int N=1010;

int f[N],a[N],n;

int main()
{
    cin >> n;
    for(int i=1;i<=n;i++) scanf("%d",a+i);

    for(int i=1;i<=n;i++)
    {
        f[i]=1;
        for(int j=1;j<i;j++)
            if(a[i]>a[j]) f[i]=max(f[i],f[j]+1);
    }

    int res=0;
    for(int i=1;i<=n;i++) res=max(res,f[i]);
    cout << res;
}

// O(n * logn)
#include <iostream>
#include <algorithm>

using namespace std;

const int N=1e5+10;

int n,g[N];

int main()
{
    cin >> n;
    int len=0;
    for(int i=1;i<=n;i++)
```

```
    {
        int x;
        scanf("%d",&x);
        int l=0,r=len;
        while(l<r)
        {
            int mid=l+r+1>>1;
            if(g[mid]<x) l=mid;
            else r=mid-1;
        }
        g[r+1]=x;
        len=max(len,r+1);
    }
    printf("%d\n",len);
}
```

1.2.2 LCS

```
// O(n * m)
#include <iostream>

using namespace std;

const int N=1010;

char s1[N],s2[N];
int n,m,f[N][N];

int main()
{
    scanf("%d%d%s%s",&n,&m,s1+1,s2+1);

    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            f[i][j]=max(f[i-1][j],f[i][j-1]);
            if(s1[i]==s2[j])
↪   f[i][j]=max(f[i][j],f[i-1][j-1]+1);
        }
    }

    printf("%d\n",f[n][m]);
    return 0;
}
```

1.2.3 LCIS

```
//定义 f[i][j] 表示以 b[j] 结尾的 LCIS
#include <iostream>

using namespace std;

const int N=3010;

int n,a[N],b[N],f[N][N];

int main()
{
    cin >> n;
    for(int i=1;i<=n;i++) scanf("%d",a+i);
    for(int i=1;i<=n;i++) scanf("%d",b+i);

    for(int i=1;i<=n;i++)
    {
        int maxv=0;
        // maxv 表示 f[i-1][1-j] 中满足 b[j]<a[i] 的最大值
        for(int j=1;j<=n;j++)
        {
            f[i][j]=f[i-1][j];
            if(a[i]==b[j]) f[i][j]=max(f[i][j],maxv+1);
            if(b[j]<a[i]) maxv=max(maxv,f[i-1][j]);
        }
    }

    int res=0;
    for(int i=1;i<=n;i++) res=max(res,f[n][i]);

    printf("%d\n",res);

    return 0;
}
```

1.2.4 回文字符串

```
/*
POJ 1159. 将给定字符串插入最少的元素变成回文串
定义 f[i][j] 为将 i-j 变成回文串的最少操作次数
状态计算: f[i][j] = min ( f[i + 1][j] , f[i][j - 1] ) + 1;
if(s[i]==s[j]) f[i][j] = min( f[i + 1][j - 1] , f[i][j] );
*/
#include <iostream>

using namespace std;

const int N=5e3+10;

char s[N];
int f[2][N],n;

int main()
{
    cin >> n;
    cin >> s+1;

    for(int i=n-1;i;i--)
    {
        int cur=i&1,pre=cur^1;
        for(int j=i+1;j<=n;j++)
        {
            f[cur][j]=min(f[cur][j-1],f[pre][j])+1;
            if(s[i]==s[j])
↪   f[cur][j]=min(f[cur][j],f[pre][j-1]);
        }

    }
    printf("%d",f[1][n]);

    return 0;
}
```

1.2.5 方格取数

```
// 两条路同时走
#include <bits/stdc++.h>

using namespace std;

const int N=15;

int n,g[N][N],f[2*N][N][N];

int main()
{
    cin >> n;
    int x,y,w;
    while(scanf("%d%d%d",&x,&y,&w),x) g[x][y]=w;

    for(int i=2;i<=n*2;i++)
        for(int j=1;j<min(n+1,i);j++)
            for(int k=1;k<min(n+1,i);k++)
            {
                if(i-j>n||i-k>n) continue;

↪   f[i][j][k]=max({f[i-1][j-1][k],f[i-1][j-1][k-1],f[i-1][j][k],f[i-1][j][k-1]});
                f[i][j][k]+=g[j][i-j]+g[k][i-k];
                if(j==k) f[i][j][k]-=g[j][i-j];
            }

    printf("%d\n",f[2*n][n][n]);
}
```

1.3 树形 DP

1.3.1 树的中心

```cpp
// 树的中心
#include <bits/stdc++.h>

using namespace std;

const int N=10010,inf=0x3f3f3f3f;

int n,d1[N],d2[N],pos[N],up[N];
vector<pair<int,int>> e[N];

int dfs_d(int u,int fa)
{
    d1[u]=d2[u]=-inf; // 考虑负权
    for(auto v : e[u])
    {
        int j=v.first,w=v.second;
        if(j==fa) continue;
        int d=dfs_d(j,u)+w;
        if(d>=d1[u]) d2[u]=d1[u],d1[u]=d,pos[u]=j;
        else if(d>d2[u]) d2[u]=d;
    }

    if(d1[u]==-inf) return 0;
    else return d1[u];
}

void dfs_u(int u,int fa)
{
    for(auto v : e[u])
    {
        int j=v.first,w=v.second;
        if(j==fa) continue;
        up[j]=up[u]+w;
        if(j==pos[u]) up[j]=max(up[j],d2[u]+w);
        else up[j]=max(up[j],d1[u]+w);

        dfs_u(j,u);
    }
}
```

```
int main()
{
    cin >> n;
    for(int i=1;i<n;i++)
    {
        int a,b,c;
        scanf("%d%d%d",&a,&b,&c);
        e[a].push_back({b,c});
        e[b].push_back({a,c});
    }

    dfs_d(1,0);
    dfs_u(1,0);

    int res=inf;
    for(int i=1;i<=n;i++) res=min(res,max(d1[i],up[i]));

    printf("%d\n",res);

    return 0;
}
```

1.3.2 最大深度和

```
// 换根 求最大深度和
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int mod=1e9+7,N=1e5+10;

int n,m,k,p[N],siz[N];
ll dp[N];
vector<pair<int,int>> e[N];

void pre(int u,int fa)
{
    siz[u]=p[u];
    for(auto v : e[u])
    {
        int j=v.first,w=v.second;
        if(j==fa) continue;
        pre(j,u);
        siz[u]+=siz[j];
        dp[1]+=(ll)siz[j]*w;
    }
}

void dfs(int u,int fa)
{
    for(auto v : e[u])
    {
        int j=v.first,w=v.second;
        if(j==fa) continue;
        dp[j]=dp[u]+(siz[1]-(ll)2*siz[j])*w;
        dfs(j,u);
    }
}

void solve()
{
    cin >> n;
    for(int i=1;i<=n;i++) scanf("%d",p+i);
    for(int i=1;i<n;i++)
```

```
    {
        int a,b,c;
        scanf("%d%d%d",&a,&b,&c);
        e[a].push_back({b,c});
        e[b].push_back({a,c});
    }

    pre(1,0);
    dfs(1,0);
    ll res=1e18;
    for(int i=1;i<=n;i++) res=min(res,dp[i]);
    printf("%lld\n",res);
}
int main()
{
    solve();
    return 0;
}
```

1.4 状压 DP

1.4.1 例 l

```cpp
#include <bits/stdc++.h>

using namespace std;

const int N=110,M=1<<10;

int n,m,g[N],f[N][70][70];
int cnt[M];
vector<int> st;

bool check(int i)
{
    for(int j=0;j<m;j++)
        if((i>>j&1)+((i>>j+1)&1)+((i>>j+2)&1)>1) return false;
    return true;
}

int main()
{
    cin >> n >> m;

    for(int i=1;i<=n;i++)
    {
        string s;
        cin >> s;
        for(int j=0;j<m;j++)
            if(s[j]=='H') g[i]+=1<<j;
    }
    g[0]=g[n+1]=g[n+2]=(1<<m)-1;

    for(int i=0;i<1<<m;i++)
        if(check(i)) st.push_back(i);

    for(int i=0;i<1<<m;i++)
        for(int j=0;j<m;j++)
            if(i>>j&1) cnt[i]++;

    memset(f,0xc0,sizeof f);
    f[0][0][0]=0;
```

```
    for(int i=1;i<=n+2;i++)
    {
        for(int j=0;j<st.size();j++)
        {
            for(int k=0;k<st.size();k++)
            {

↪   if((st[j]&g[i-1])||(st[k]&g[i])||(st[j]&st[k])) continue;
                for(int l=0;l<st.size();l++)
                    if(!(st[l]&st[k]))

↪   f[i][j][k]=max(f[i-1][l][j]+cnt[st[k]],f[i][j][k]);
            }
        }
    }

    printf("%d\n",f[n+2][0][0]);
}
```

1.5 数位 DP

1.5.1 例 1

```cpp
// Windy 数：不含前导零且相邻两个数字之差至少为 2 的正整数被称为
↪  Windy 数。
#include <bits/stdc++.h>

using namespace std;

int f[11][10],a,b;

int dp(int n)
{
    if(!n) return 0;

    int res=0,last=-2;
    vector<int> a;
    while(n) a.push_back(n%10), n/=10;

    for(int i=a.size()-1;i>=0;i--)
    {
        int t=a[i];
        for(int j=0;j<t;j++)
            if(abs(j-last)>=2) res+=f[i][j];
        if(abs(t-last)>=2) last=t;
        else break;

        if(!i) res++;
    }

    for(int i=0;i<a.size()-2;i++)
        for(int j=1;j<10;j++)
            res+=f[i][j];

    res+=f[a.size()-2][1];

    return res;
}

int main()
{
    cin >> a >> b;
    for(int i=0;i<10;i++) f[0][i]=1;
```

```
    for(int i=1;i<11;i++)
        for(int j=0;j<10;j++)
            for(int k=0;k<10;k++)
                if(abs(j-k)>=2)
                    f[i][j]+=f[i-1][k];

    printf("%d\n",dp(b)-dp(a-1));

    return 0;
}
```

1.6 概率 DP

1.6.1 例 l

```
/*
1. 求到 n 号点的概率:
f[i] 表示从 1 号点到 i 号点的概率
f[1]=1, f[i] = sum(f[j]/deg[j]) j 存在一条指向 i 的出边

2. 求到 n 号点的期望步数
f[i] 表示从 i 号点走到 n 号点的期望步数
f[n]=0  f[i] = sum(f[j]/deg[x]) + 1  j 为 i 的出边指向的点

3. 到达 n 号点的期望步数 图不保证是无环的
f[n]=0 线性方程组 高斯消元 O(n^3)

4. 一个长为 n 的 01 串 每次等概率选择一位翻转 问将 01 串变成全 0
↪  的期望步数
f[i] 表示当前串有 i 个 1 变成 i-1 个 1 的期望步数   有 f[n]=1
考虑 i<n  f[i] = i/n * 1 + (1-i/n) * (1 + f[i+1] + f[i])
整理可得   i * f[i] = n + (n-i) * f[i+1]
假设给定串有 cnt 个 1   res = sum(f[i]) i from 1 to cnt
*/
// 4
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int mod=998244353,N=1e6+10;

int n,m,k;
string x,y;

int qmi(int a,int k,int p)
{
    int res=1;
    while(k)
    {
        if(k&1) res=(ll)res*a%p;
        a=(ll)a*a%p;
        k>>=1;
    }
```

```
    return res;
}

void solve()
{
    cin >> n;
    cin >> x >> y;
    int cnt=0;
    for(int i=0;i<n;i++) cnt+=(x[i]!=y[i]);

    ll res=0;
    vector<int> f(n+1,0);
    f[n]=1;
    if(cnt==n) res+=1;
    for(int i=n-1;i>=1;i--)
    {
        int t=((ll)(n-i)*f[i+1]%mod+n)%mod;
        f[i]=(ll)t*qmi(i,mod-2,mod)%mod;
        if(i<=cnt) res+=f[i],res%=mod;
    }

    printf("%d\n",res);
}
```

1.7 子集和 DP

1.7.1 枚举子集超集

```cpp
#include <bits/stdc++.h>

using namespace std;

int n;

void todo(){}

void solve()
{
    // 枚举 i 的所有非空子集    O(3^n)
    for(int i = 1; i < 1<<n; i++)
        for(int j = i; j; j = (j-1)&i)
            todo();

    // 枚举 i 的所有超集    O(3^n)
    for(int  i = 1; i < 1<<n; i++)
        for(int j = i; j < 1<<n ; j = (j+1)|i)
            todo();
}
```

1.7.2 子集计数

```
/*
a[i] <= m   O(mlogm)
给定 a[i] 解决 f[mask]=sum(a[i]) 的求解   其中 i 是所有 mask 的
↪ 子集

定义 dp[i][mask] 代表只有二进制从低到高到 i 位和 mask 不同的
↪ sum(a[i])
    if( mask>>i &1 == 0) dp[i][mask] = dp[i-1][mask]
    else dp[i][mask] = dp[i-1][mask] + dp[i-1][mask^(1<<i)]
*/
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int mod=1e9+7,N=1e5+10,M=1<<17;

int n,m,k;
ll dp[M];
int a[N];

void solve()
{
    cin >> n;
    int maxv=M-1;
    for(int i=0;i<n;i++)
    {
        int x;
        scanf("%d",&x);
        a[i]=x;
        dp[x]++;
    }

    //i 从 0 开始
    for(int i=0;i<17;i++)
        for(int j=M-1;j>=0;j--)
            if(j>>i&1) dp[j]+=dp[j^(1<<i)];

    ll res=0;
    for(int i=0;i<n;i++)
        res+=dp[a[i]^maxv];
```

```
    cout << res << endl;
}
```

1.7.3 其他例子

```
/*
求 ai | (aj & ak)  i<j<k  的最大值
dp[i][j] 表示 j 的超集中前二大的下标

*/
#include <bits/stdc++.h>

using namespace std;

const int mod=1e9+7,N=1e6+10,M=1<<21;

int n,m,p[N];
pair<int,int> dp[M];

void insert(int mask,int u)
{
    if(u>dp[mask].first)
↪   dp[mask].second=dp[mask].first,dp[mask].first=u;
    else if(u>dp[mask].second&&u!=dp[mask].first)
↪   dp[mask].second=u;
}

void solve()
{
    cin >> n;
    for(int i=1;i<=n;i++) scanf("%d",p+i),insert(p[i],i);
    for(int j=0;j<=20;j++)
        for(int i=0;i<M;i++)
            if((i>>j&1)==0)

↪   insert(i,dp[i^(1<<j)].first),insert(i,dp[i^(1<<j)].second);
    int ans=0;
    for(int i=1;i<n-1;i++)
    {
        int res=0;
        int mask=0;
        for(int j=20;j>=0;j--)
        {
            if(p[i]>>j&1) res+=(1<<j);
            else if(dp[mask|(1<<j)].second>i)
↪   res+=(1<<j),mask|=(1<<j);
        }
```

```
        ans=max(ans,res);
    }

    cout << ans << endl;
}
```

2 Data Structure

2.1 Hash

```cpp
#include <bits/stdc++.h>

using namespace std;

struct custom_hash
{
    static uint64_t splitmix64(uint64_t x)
    {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const
    {
        static const uint64_t FIXED_RANDOM =
↪   chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};

unordered_map<int,int,custom_hash> hh;
```

2.2 单调队列

```cpp
#include <iostream>

using namespace std;

const int N=1e6+10;

int a[N],q[N];
int n,k;

int main()
{
    scanf("%d%d",&n,&k);
    for(int i=0;i<n;i++) scanf("%d",&a[i]);
    int f=0,r=-1;
    for(int i=0;i<n;i++)
    {
        if(f<=r&&i-q[f]>=k) f++;
        while(f<=r&&a[q[r]]>=a[i])r--;
        q[++r]=i;
        if(i>=k-1) printf("%d ",a[q[f]]);
    }
    return 0;
}
```

2.3 DSU

2.3.1 边带权

```cpp
#include <bits/stdc++.h>

using namespace std;

const int N=50010;

int p[N],d[N];

int find(int k)
{
    if(p[k]!=k)
    {
        int u=p[k];
        p[k]=find(p[k]);
        d[k]+=d[u];
    }
    return p[k];
}

int main()
{
    int n,k;
    cin >> n >> k;
    for(int i=1;i<=n;i++) p[i]=i;
    int res=0;

    while(k--)
    {
        int D,x,y;
        cin >> D >> x >> y;

        if(x>n||y>n) res++;
        else if(D==2)
        {
            if(x==y) res++;
            else
            {
                int a=find(x),b=find(y);
                if(a==b)
```

```
                {
                    if((d[x]-d[y]-1)%3) res++;
                }
                else p[a]=b,d[a]=d[y]-d[x]+1;
            }
        }
        else
        {
            int a=find(x),b=find(y);
            if(a==b)
            {
                if((d[x]-d[y])%3)res++;
            }
            else p[a]=b,d[a]=d[y]-d[x];
        }
    }

    cout << res;
    return 0;
}
```

2.3.2 扩展域

```
#include <iostream>

using namespace std;

const int N=50001;

int p[N*3],n,k,res;

int find(int k)
{
    if(p[k]!=k) p[k]=find(p[k]);
    return p[k];
}

void uni(int a,int b)
{
    p[find(a)]=find(b);
}

void check()
{
    int d,x,y;
    cin >> d >> x >> y;

    if(x>n||y>n)
    {
        res++;
        return;
    }

    if(d==1)
    {
        if(find(x)==find(y+n)||find(x+n)==find(y)) res++;
        else
        {
            uni(x,y);
            uni(x+n,y+n);
            uni(x+2*n,y+2*n);
        }
    }
    else
```

```
    {
        if(x==y||find(x)==find(y)||find(y)==find(x+n)) res++;
        else
        {
            uni(x,y+n);
            uni(x+n,y+2*n);
            uni(x+2*n,y);
        }
    }
}

int main()
{
    cin >> n >> k;
    for(int i=1;i<3*n;i++)p[i]=i;

    while(k--) check();

    cout << res;
    return 0;
}
```

2.4 ST 表

```cpp
#include <bits/stdc++.h>

using namespace std;

const int N=1e5+10;

int n,m,f[18][N];

void init()
{
    for(int j=1;j<18;j++)
        for(int i=1;i+(1<<j)-1<=n;i++)
            f[j][i]=max(f[j-1][i],f[j-1][i+(1<<j-1)]);
}

int main()
{
    cin >> n >> m;
    for(int i=1;i<=n;i++)
    {
        int x;
        scanf("%d",&x);
        f[0][i]=x;
    }

    init();

    while(m--)
    {
        int l,r;
        scanf("%d%d",&l,&r);
        int k=31-__builtin_clz(r-l+1);
        printf("%d\n",max(f[k][l],f[k][r-(1<<k)+1]));
    }
}
```

2.5 Trie

2.5.1 普通 trie

```
#include <iostream>

using namespace std;

const int N=1e5+10;

int trie[N][26],cnt[N],n,idx;

char s[N];

void insert()
{
    int p=0;
    for(int i=0;s[i];i++)
    {
        int t=s[i]-'a';
        if(!trie[p][t]) trie[p][t]=++idx;
        p=trie[p][t];
    }
    cnt[p]++;
}

int find()
{
    int p=0;
    for(int i=0;s[i];i++)
    {
        int t=s[i]-'a';
        if(!trie[p][t]) return 0;
        p=trie[p][t];
    }

    return cnt[p];
}

int main()
{
    cin >> n;
    while(n--)
```

```
    {
        char str[2];
        scanf("%s%s",str,s);
        if(str[0]=='I') insert();
        else printf("%d\n",find());
    }
    return 0;
}
```

2.5.2 可持久化 trie

```
#include <bits/stdc++.h>

using namespace std;

const int N=6e5+10;

int n,m,trie[N*25][2],idx,cnt[N*25],root[N];
int s[N];

void insert(int num,int p,int q)
{
    for(int i=24;i>=0;i--)
    {
        if(p) trie[q][0]=trie[p][0],trie[q][1]=trie[p][1];
        int t=num>>i&1;
        trie[q][t]=++idx;
        p=trie[p][t],q=trie[q][t];
        cnt[q]=cnt[p]+1;
    }
}

int query(int num,int p,int q)
{
    int res=0;
    for(int i=24;i>=0;i--)
    {
        int t=num>>i&1;
        if(cnt[trie[q][t^1]]-cnt[trie[p][t^1]])
            res+=1<<i,t^=1;
        p=trie[p][t];
        q=trie[q][t];
    }

    return res;
}

int main()
{
    cin >> n >> m;
    root[0]=++idx,root[1]=++idx;
    insert(0,root[0],root[1]);
```

```
    for(int i=1;i<=n;i++)
    {
        int x;
        scanf("%d",&x);
        s[i]=s[i-1]^x;
        root[i+1]=++idx;
        insert(s[i],root[i],root[i+1]);
    }

    while(m--)
    {
        char str[2];
        scanf("%s",str);
        if(*str=='A')
        {
            int x;
            scanf("%d",&x);
            n++;
            s[n]=s[n-1]^x;
            root[n+1]=++idx;
            insert(s[n],root[n],root[n+1]);
        }
        else
        {
            int l,r,x;
            scanf("%d%d%d",&l,&r,&x);
            printf("%d\n",query(x^s[n],root[l-1],root[r]));
        }
    }
}
```

2.6 树状数组

```cpp
#include <bits/stdc++.h>

using namespace std;

const int N=50010;

int tr[N],n,m,a[N],tree[N][N];
int tr1[N],tr2[N];

// 树状数组
int lowbit(int x)
{
    return x&-x;
}

void init()
{
    for(int i=1;i<=n;i++)
    {
        tr[i]=a[i];
        for(int j=i-1;j>i-lowbit(i);j-=lowbit(j))
        {
            tr[i]+=tr[j];
        }
    }
}

void add(int x,int c,int tr[]=tr)
{
    for(int i=x;i<=n;i+=lowbit(i)) tr[i]+=c;
}

int sum(int x,int tr[]=tr)
{
    int res=0;
    for(int i=x;i;i-=lowbit(i)) res+=tr[i];
    return res;
}


// 二维树状数组
```

```
void add(int x,int y,int c)
{
    for(int i=x;i<=n;i+=lowbit(i))
        for(int j=y;j<=m;j+=lowbit(j))
            tree[i][j]+=c;
}

int sum(int x,int y)
{
    int res=0;
    for(int i=x;i;i-=lowbit(i))
        for(int j=y;j;j-=lowbit(j))
            res+=tree[i][j];
}


// 区间加, 区间求和 ：tr1 维护差分数组 d[i], tr2 维护 i*d[i]
// 求 sum 1-x
int pre(int x)
{
    return (x+1)*sum(x,tr1)-sum(x,tr2);
}


// 维护区间最值
void update(int x,int c)
{
    //剪枝优化
    if(c==a[x]) return;
    if(c>a[x])
    {
        for(int i=x;i<=n;i+=lowbit(i))
            tr[i]=max(tr[i],c);
        a[x]=c;
        return;
    }

    a[x]=c;
    for(int i=x;i<=n;i+=lowbit(i))
    {
        tr[i]=a[i];
        for(int j=i-1;j>i-lowbit(i);j-=lowbit(j))
            tr[i]=max(tr[i],tr[j]);
    }
```

```
}

//如果是前缀最值可以直接 O(logn) 求
int query(int l,int r)
{
    if(l>r) return 0;
    int res=0;
    if(r-lowbit(r)+1<l) res=max(a[r],query(l,r-1));
    else res=max(tr[r],query(l,r-lowbit(r)));
    return res;
}


// 倍增求序列第 k 小
int k_min(int k)
{
    int pos=0;
    for(int i=20;i>=0;i--)
    {
        if(pos+(1<<i)<=n && tr[pos+(1<<i)]<k)
            pos+=1<<i,k-=tr[pos];
    }
    return pos+1;
}
```

2.7 线段树

2.7.1 普通线段树

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef pair<int,int> pii;

const int mod=1e9+7,N=2e5+10;

int n,m,k;
int p[N];
struct node
{

}tr[N*4];

void pushup(node &u,node &l,node &r)
{

}

void pushup(int u)
{
    pushup(tr[u],tr[u<<1],tr[u<<1|1]);
}

void pushdown(int u,int L,int R)
{

}

void build(int u,int L,int R)
{
    if(L==R)
    {

        return;
    }
    int mid=L+R>>1;
```

```
    build(u<<1,L,mid),build(u<<1|1,mid+1,R);
    pushup(u);
}

//区间修改
void modify(int u,int L,int R,int l,int r,int d)
{
    if(L>=l&&R<=r)
    {

        return;
    }
    pushdown(u,L,R);
    int mid=L+R>>1;
    if(l<=mid) modify(u<<1,L,mid,l,r,d);
    if(r>mid) modify(u<<1|1,mid+1,R,l,r,d);
    pushup(u);
}

//单点修改
void modify(int u,int L,int R,int x,int d)
{
    if(L==R)
    {

        return;
    }
    int mid=L+R>>1;
    if(x<=mid) modify(u<<1,L,mid,x,d);
    else modify(u<<1|1,mid+1,R,x,d);
    pushup(u);
}

node query(int u,int L,int R,int l,int r)
{
    if(L>=l&&R<=r) return tr[u];
    pushdown(u,L,R);
    int mid=L+R>>1;
    if(l>mid) return query(u<<1|1,mid+1,R,l,r);
    else if(r<=mid) return query(u<<1,L,mid,l,r);
    node res;
    node ls=query(u<<1,L,mid,l,r);
    node rs=query(u<<1|1,mid+1,R,l,r);
    pushup(res,ls,rs);
```

```
    return res;
}
```

2.7.2 动态开点

```
//动态开点 root=1

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef pair<int,int> pii;

const int mod=1e9+7,N=2e5+10;

int n,m,k;
int p[N];
struct node
{
    int ls,rs;

}tr[N];//O(mlogn) 尽量开大
int root=1,id=1;

void pushup(node &u,node &left,node &right)
{

}

void pushup(int u)
{
    pushup(tr[u],tr[tr[u].ls],tr[tr[u].rs]);
}

void pushdown(int u,int L,int R)
{
    if(!tr[u].tag) return;
    if(!tr[u].ls) tr[u].ls=++id;
    if(!tr[u).rs) tr[u].rs=++id;

}

//区间修改
void modify(int &u,int L,int R,int l,int r,int d)
{
    if(!u) u=++id;
```

```
    if(L>=l&&R<=r)
    {

        return;
    }
    pushdown(u,L,R);
    int mid=L+R-1>>1;
    if(l<=mid) modify(tr[u].ls,L,mid,l,r,d);
    if(r>mid) modify(tr[u].rs,mid+1,R,l,r,d);
    pushup(u);
}

//单点修改
void modify(int &u,int L,int R,int x,int d)
{
    if(!u) u=++id;
    if(L==R)
    {

        return;
    }
    int mid=L+R-1>>1;
    if(x>mid) modify(tr[u].rs,mid+1,R,x,d);
    else modify(tr[u].ls,L,mid,x,d);
    pushup(u);
}

//区间查询 要保证 l <= r
node query(int u,int L,int R,int l,int r)
{
    if(!u) return node();
    if(L>=l&&R<=r) return tr[u];
    pushdown(u,L,R);
    int mid=L+R-1>>1;
    if(l>mid) return query(tr[u].rs,mid+1,R,l,r);
    else if(r<=mid) return query(tr[u].ls,L,mid,l,r);
    node res;
    node left=query(tr[u].ls,L,mid,l,r);
    node right=query(tr[u].rs,mid+1,R,l,r);
    pushup(res,left,right);
    return res;
}
```

2.7.3 势能线段树

```
//势能线段树 区间开根

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int mod=1e9+7,N=2e5+10;

int n,m,k;
ll p[N];
struct node
{
    ll sum,v;
}tr[N*4];

void pushup(node &u,node &l,node &r)
{
    u.sum=l.sum+r.sum;
    u.v=max(l.v,r.v);
}

void pushup(int u)
{
    pushup(tr[u],tr[u<<1],tr[u<<1|1]);
}

void build(int u,int L,int R)
{
    if(L==R)
    {
        tr[u].sum=p[L];
        tr[u].v=p[L];
        return;
    }
    int mid=L+R>>1;
    build(u<<1,L,mid),build(u<<1|1,mid+1,R);
    pushup(u);
}

void modify(int u,int L,int R,int l,int r)
```

```
{
    if(tr[u].v<=1) return;
    if(L==R)
    {
        tr[u].sum=sqrt(tr[u].sum);
        tr[u].v=tr[u].sum;
        return;
    }
    int mid=L+R>>1;
    if(l<=mid) modify(u<<1,L,mid,l,r);
    if(r>mid) modify(u<<1|1,mid+1,R,l,r);
    pushup(u);
}

ll query(int u,int L,int R,int l,int r)
{
    if(L>=l&&R<=r) return tr[u].sum;
    int mid=L+R>>1;
    ll res=0;
    if(l<=mid) res+=query(u<<1,L,mid,l,r);
    if(r>mid) res+=query(u<<1|1,mid+1,R,l,r);
    return res;
}

void solve()
{
    cin >> n >> m;
    for(int i=1;i<=n;i++) scanf("%lld",p+i);
    build(1,1,n);
    while(m--)
    {
        int k,l,r;
        scanf("%d%d%d",&k,&l,&r);
        if(l>r) swap(l,r);
        if(k==2) printf("%lld\n",query(1,1,n,l,r));
        else modify(1,1,n,l,r);
    }
}

int main()
{
    solve();
    return 0;
}
```

2.7.4 可持久化线段树

```
// 区间第 k 小
#include <bits/stdc++.h>

using namespace std;

const int N=1e5+10,INF=1e9+5;

int n,m,idx;
int root[N],a[N];

struct node
{
    int l,r;
    int cnt;
}tr[N*40];

void pushup(int u)
{
    tr[u].cnt=tr[tr[u].l].cnt+tr[tr[u].r].cnt;
}

void modify(int p,int q,int L,int R,int x)
{
    tr[q]=tr[p];
    if(L==R)
    {
        tr[q].cnt++;
        return;
    }

    int mid=L+R-1>>1;
    if(x>mid)
    {
        tr[q].r=++idx;
        modify(tr[p].r,tr[q].r,mid+1,R,x);
    }
    else
    {
        tr[q].l=++idx;
        modify(tr[p].l,tr[q].l,L,mid,x);
    }
    pushup(q);
```

```
}

int query(int p,int q,int L,int R,int k)
{
    if(L==R) return L;
    int mid=L+R-1>>1;
    int cnt=tr[tr[q].l].cnt-tr[tr[p].l].cnt;
    if(cnt>=k) return query(tr[p].l,tr[q].l,L,mid,k);
    else return query(tr[p].r,tr[q].r,mid+1,R,k-cnt);

}

int main()
{
    cin >> n >> m;
    root[0]=++idx;
    for(int i=1;i<=n;i++) scanf("%d",a+i);
    for(int i=1;i<=n;i++)
    {
        root[i]=++idx;
        modify(root[i-1],root[i],-INF,INF,a[i]);
    }

    while(m--)
    {
        int l,r,k;
        scanf("%d%d%d",&l,&r,&k);
        printf("%d\n",query(root[l-1],root[r],-INF,INF,k));
    }
}
```

2.8 平衡树

2.8.1 有旋 treap

```
#include <bits/stdc++.h>

using namespace std;

const int N=1e5+10,INF=1e9+10;

int n,root,idx;
struct node
{
    int l,r;
    int key,val;
    int cnt,size;
}tr[N];

int get_node(int key)
{
    tr[++idx].key=key;
    tr[idx].val=rand();
    tr[idx].cnt=tr[idx].size=1;
    return idx;
}

void pushup(int p)
{
    tr[p].size=tr[tr[p].l].size+tr[p].cnt+tr[tr[p].r].size;
}

void zig(int &p)
{
    int q=tr[p].l;
    tr[p].l=tr[q].r,tr[q].r=p,p=q;
    pushup(tr[p].r),pushup(p);
}

void zag(int &p)
{
    int q=tr[p].r;
    tr[p].r=tr[q].l,tr[q].l=p,p=q;
    pushup(tr[p].l),pushup(p);
```

```
}

void insert(int &p,int key)
{
    if(!p) p=get_node(key);
    else if(tr[p].key==key) tr[p].cnt++;
    else if(tr[p].key>key)
    {
        insert(tr[p].l,key);
        if(tr[tr[p].l].val>tr[p].val) zig(p);
    }
    else
    {
        insert(tr[p].r,key);
        if(tr[tr[p].r].val>tr[p].val) zag(p);
    }

    pushup(p);

}

void dele(int &p,int key)
{
    if(!p) return;
    if(tr[p].key==key)
    {
        if(tr[p].cnt>1) tr[p].cnt--;
        else if(tr[p].l||tr[p].r)
        {
            if(!tr[p].r||tr[tr[p].l].val>tr[tr[p].r].val)
            {
                zig(p);
                dele(tr[p].r,key);
            }
            else
            {
                zag(p);
                dele(tr[p].l,key);
            }
        }
        else p=0;
    }
    else if(tr[p].key>key) dele(tr[p].l,key);
    else dele(tr[p].r,key);
```

```
    pushup(p);
}

int get_rank(int p,int key)
{
    if(!p) return 1;
    if(tr[p].key==key) return tr[tr[p].l].size+1;
    if(tr[p].key>key) return get_rank(tr[p].l,key);
    return get_rank(tr[p].r,key)+tr[p].cnt+tr[tr[p].l].size;
}

int get_key(int p,int rank)
{
    if(!p) return INF;
    if(tr[tr[p].l].size>=rank) return get_key(tr[p].l,rank);
    if(tr[tr[p].l].size+tr[p].cnt>=rank) return tr[p].key;
    return get_key(tr[p].r,rank-tr[p].cnt-tr[tr[p].l].size);
}

int get_pre(int p,int key)
{
    if(!p) return -INF;
    if(tr[p].key>=key) return get_pre(tr[p].l,key);
    return max(tr[p].key,get_pre(tr[p].r,key));
}

int get_next(int p,int key)
{
    if(!p) return INF;
    if(tr[p].key<=key) return get_next(tr[p].r,key);
    return min(tr[p].key,get_next(tr[p].l,key));
}

int main()
{
    cin >> n;
    while(n--)
    {
        int opt,x;
        scanf("%d%d",&opt,&x);
        if(opt==1) insert(root,x);
        else if(opt==2) dele(root,x);
        else if(opt==3) printf("%d\n",get_rank(root,x));
```

```
        else if(opt==4) printf("%d\n",get_key(root,x));
        else if(opt==5) printf("%d\n",get_pre(root,x));
        else printf("%d\n",get_next(root,x));
    }

    return 0;
}
```

2.8.2 FHQtreap

```cpp
#include <bits/stdc++.h>

using namespace std;

const int N=1e5+10,INF=1e8+10;

int n,m,idx,root;
int a[N],st[N];
//相同权值的点不会去重 同时存在
struct node
{
    int l,r;
    int key,val;
    int size;
}tr[N];

int get_node(int key)
{
    tr[++idx].key=key;
    tr[idx].val=rand();
    tr[idx].size=1;
    return idx;
}

void pushup(int p)
{
    tr[p].size=tr[tr[p].l].size+tr[tr[p].r].size+1;
}

//按照 key 将树分成 <=key 和 >key 的两棵子树
void split_by_key(int p,int key,int &x,int &y)
{
    if(!p) x=y=0;
    else
    {
        if(tr[p].key<=key)
 ↪  x=p,split_by_key(tr[p].r,key,tr[x].r,y);
        else y=p,split_by_key(tr[p].l,key,x,tr[y].l);
        pushup(p);
    }
}
```

```
//按前 k 个分配 将树分成前 k 个点组成的树和剩余点组成的树
void split_by_size(int p,int k,int &x,int &y)
{
    if(!p) x=y=0;
    else
    {
        if(tr[tr[p].l].size<k)
↪   x=p,split_by_size(tr[p].r,k-tr[tr[p].l].size-1,tr[x].r,y);
        else y=p,split_by_size(tr[p].l,k,x,tr[y].l);
        pushup(p);
    }
}

//合并 x 和 y 两棵树 其中 x.key<y.key 返回合并后的根
int merge(int x,int y)
{
    if(!x||!y) return x+y;
    int res;
    if(tr[x].val>tr[y].val) res=x,tr[x].r=merge(tr[x].r,y);
    else res=y,tr[y].l=merge(x,tr[y].l);
    pushup(res);
    return res;
}

//插入：按照 key 分裂 开新点 合并左和新点 再合并右
void insert(int &p,int key)
{
    int x,y;
    split_by_key(p,key,x,y);
    int new_node=get_node(key);
    p=merge(merge(x,new_node),y);
}

//删除：按照 key 分裂成三部分 删除中间部分的根 合并三次
void dele(int &p,int key)
{
    int x,y,z;
    split_by_key(p,key,x,z);
    split_by_key(x,key-1,x,y);
    y=merge(tr[y].l,tr[y].r);
    p=merge(merge(x,y),z);
}
```

```
//获得 key 的排名：按 key-1 分裂 排名就是左子树的大小 +1
int get_rank(int &p,int key)
{
    int x,y,res;
    split_by_key(p,key-1,x,y);
    res=tr[x].size+1;
    p=merge(x,y);
    return res;
}

//rank 对应的 key rank<1 返回最小数 rank>sum_node 返回 INF
int get_key(int &p,int rank)
{
    int x,y,z,res;
    split_by_size(p,rank-1,x,y);
    split_by_size(y,1,y,z);
    if(y) res=tr[y].key;
    else res=INF;
    p=merge(merge(x,y),z);
    return res;
}

//前驱 不存在返回-INF
int get_pre(int &p,int key)
{
    int x,y,z,res;
    split_by_key(p,key-1,y,z);
    split_by_size(y,tr[y].size-1,x,y);
    if(y) res=tr[y].key;
    else res=-INF;
    p=merge(merge(x,y),z);
    return res;
}

//后继 不存在返回 INF
int get_next(int &p,int key)
{
    int x,y,z,res;
    split_by_key(p,key,x,y);
    split_by_size(y,1,y,z);
    if(y) res=tr[y].key;
    else res=INF;
```

```
    p=merge(merge(x,y),z);
    return res;
}

void dfs(int p)
{
    if(!p) return;
    dfs(tr[p].l),dfs(tr[p].r);
    pushup(p);
}

//a[] 是有序序列 O(n) 建树建成 treap 别忘记最后 dfs 更新信息
void build(int &root,int len)
{
    int top=0;
    for(int i=1;i<=len;i++)
    {
        int k=top;
        int new_id=get_node(a[i]);
        while(k&&tr[st[k]].val<tr[new_id].val) k--;
        if(k) tr[st[k]].r=new_id;
        if(k<top) tr[new_id].l=st[k+1];
        st[++k]=new_id;
        top=k;
    }
    root=st[1];
    dfs(root);
}


int main()
{
    cin >> n >> m;
    for(int i=0;i<n;i++)
    {
        int x;
        scanf("%d",&x);
        insert(root,x);
    }
    int last=0;
    int ans=0;
    while(m--)
    {
```

```
        int opt,x;
        scanf("%d%d",&opt,&x);
        x^=last;
        if(opt==1) insert(root,x);
        else if(opt==2) dele(root,x);
        else if(opt==3) last=get_rank(root,x);
        else if(opt==4) last=get_key(root,x);
        else if(opt==5) last=get_pre(root,x);
        else last=get_next(root,x);
        if(opt>2) ans^=last;
    }

    printf("%d\n",ans);

    return 0;
}
```

2.8.3 文艺平衡树

```cpp
#include <bits/stdc++.h>

using namespace std;

const int mod=1e9+7,N=1e5+10;

int n,m,idx,root;
struct node
{
    int l,r;
    int tag;
    int key,val;
    int size;
}tr[N];

int get_node(int key)
{
    tr[++idx].key=key;
    tr[idx].val=rand();
    tr[idx].size=1;
    return idx;
}

void pushup(int p)
{
    tr[p].size=tr[tr[p].l].size+tr[tr[p].r].size+1;
}

void pushdown(int p)
{
    if(!tr[p].tag) return;
    swap(tr[p].l,tr[p].r);
    if(tr[p].l) tr[tr[p].l].tag^=1;
    if(tr[p].r) tr[tr[p].r].tag^=1;
    tr[p].tag=0;
}

void split_by_key(int p,int key,int &x,int &y)
{
    if(!p) x=y=0;
    else
```

```
    {
        if(tr[p].key<=key)
↪   x=p,split_by_key(tr[p].r,key,tr[x].r,y);
        else y=p,split_by_key(tr[p].l,key,x,tr[y].l);
        pushup(p);
    }
}

void split(int p,int k,int &x,int &y)
{
    if(!p) x=y=0;
    else
    {
        pushdown(p);
        if(tr[tr[p].l].size+1<=k)
        {
            x=p;
            split(tr[p].r,k-tr[tr[p].l].size-1,tr[x].r,y);
        }
        else
        {
            y=p;
            split(tr[p].l,k,x,tr[y].l);
        }
        pushup(p);
    }
}

int merge(int x,int y)
{
    if(!x||!y) return x+y;
    pushdown(x),pushdown(y);
    int res;
    if(tr[x].val>tr[y].val)
    {
        res=x;
        tr[x].r=merge(tr[x].r,y);
    }
    else
    {
        res=y;
        tr[y].l=merge(x,tr[y].l);
    }
    pushup(res);
```

```
    return res;
}

// 区间翻转
void rever(int l,int r)
{
    int x,y,z;
    split(root,r,y,z);
    split(y,l-1,x,y);
    tr[y].tag^=1;
    root=merge(merge(x,y),z);
}

void insert(int &p,int key)
{
    int x,y;
    split_by_key(p,key,x,y);
    int new_node=get_node(key);
    p=merge(merge(x,new_node),y);
}

void print(int p)
{
    if(!p) return;
    pushdown(p);
    print(tr[p].l);
    printf("%d ",tr[p].key);
    print(tr[p].r);
}


void solve()
{
    cin >> n >> m;
    for(int i=1;i<=n;i++) insert(root,i);
    while(m--)
    {
        int l,r;
        scanf("%d%d",&l,&r);
        rever(l,r);
    }
    print(root);
}
```

```
int main()
{
    solve();
    return 0;
}
```

2.8.4 可持久化 treap

```
#include <bits/stdc++.h>

using namespace std;

const int N=5e5+10,INF=(1ll<<31)-1;

int n,m,root[N],idx;
struct node
{
    int l,r;
    int key,val;
    int size;
}tr[N*52];

int get_node(int key)
{
    tr[++idx].key=key;
    tr[idx].val=rand();
    tr[idx].size=1;
    return idx;
}

void pushup(int p)
{
    tr[p].size=1+tr[tr[p].l].size+tr[tr[p].r].size;
}

void split_by_key(int p,int key,int &x,int &y)
{
    if(!p) x=y=0;
    else
    {
        int res;
        if(tr[p].key<=key)
 ↪  x=get_node(0),res=x,tr[x]=tr[p],split_by_key(tr[p].r,key,tr[x].r,y);
        else
 ↪  y=get_node(0),res=y,tr[y]=tr[p],split_by_key(tr[p].l,key,x,tr[y].l);
        pushup(res);
    }
}
```

```
void split_by_size(int p,int k,int &x,int &y)
{
    if(!p) x=y=0;
    else
    {
        int res;
        if(tr[tr[p].l].size<k)
↪   x=get_node(0),res=x,tr[x]=tr[p],split_by_size(tr[p].r,k-tr[tr[p].l].size-1,tr[x]
        else
↪   y=get_node(0),res=y,tr[y]=tr[p],split_by_size(tr[p].l,k,x,tr[y].l);
        pushup(res);
    }
}

int merge(int x,int y)
{
    if(!x||!y) return x+y;
    int root;
    if(tr[x].val>tr[y].val)
↪   root=get_node(0),tr[root]=tr[x],tr[root].r=merge(tr[x].r,y);
    else
↪   root=get_node(0),tr[root]=tr[y],tr[root].l=merge(x,tr[y].l);
    pushup(root);
    return root;
}

void insert(int &root,int odd,int key)
{
    int x,y,z;
    y=get_node(key);
    split_by_key(odd,key-1,x,z);
    root=merge(merge(x,y),z);
}

void dele(int &root,int odd,int key)
{
    int x,y,z;
    split_by_key(odd,key-1,x,y);
    split_by_key(y,key,y,z);
    y=merge(tr[y].l,tr[y].r);
    root=merge(merge(x,y),z);
}

int get_rank(int &root,int odd,int key)
```

```
{
    int x,y,res;
    split_by_key(odd,key-1,x,y);
    res=tr[x].size+1;
    root=merge(x,y);
    return res;
}

int get_key(int &root,int odd,int rank)
{
    int x,y,z;
    split_by_size(odd,rank-1,x,y);
    split_by_size(y,1,y,z);
    int res=tr[y].key;
    root=merge(merge(x,y),z);
    return res;
}

int get_pre(int &root,int odd,int key)
{
    int x,y,z;
    split_by_key(odd,key-1,y,z);
    split_by_size(y,tr[y].size-1,x,y);
    int res;
    if(y) res=tr[y].key;
    else res=-INF;
    root=merge(merge(x,y),z);
    return res;
}

int get_next(int &root,int odd,int key)
{
    int x,y,z;
    split_by_key(odd,key,x,y);
    split_by_size(y,1,y,z);
    int res;
    if(y) res=tr[y].key;
    else res=-INF;
    root=merge(merge(x,y),z);
    return res;
}


int main()
```

```
{
    cin >> n;
    for(int i=1;i<=n;i++)
    {
        int v,opt,x;
        scanf("%d%d%d",&v,&opt,&x);
        if(opt==1) insert(root[i],root[v],x);
        else if(opt==2) dele(root[i],root[v],x);
        else if(opt==3)
↪   printf("%d\n",get_rank(root[i],root[v],x));
        else if(opt==4)
↪   printf("%d\n",get_key(root[i],root[v],x));
        else if(opt==5)
↪   printf("%d\n",get_pre(root[i],root[v],x));
        else printf("%d\n",get_next(root[i],root[v],x));
    }

    return 0;
}
```

2.9 莫队

2.9.1 普通莫队

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int N=5e4+10;

int n,m,k,cnt[N],a[N];
int len;
ll ans[N],res;
struct q
{
    int id,l,r;
}Q[N];

bool cmp(q &a,q &b)
{
    if(a.l/len!=b.l/len) return a.l<b.l;
    if(a.l/len&1) return a.r>b.r;
    return a.r<b.r;
}

void add(int x)
{
    res-=(ll)cnt[x]*cnt[x];
    cnt[x]++;
    res+=(ll)cnt[x]*cnt[x];
}

void dele(int x)
{
    res-=(ll)cnt[x]*cnt[x];
    cnt[x]--;
    res+=(ll)cnt[x]*cnt[x];
}

int main()
{
```

```
    cin >> n >> m >> k;
    for(int i=1;i<=n;i++) scanf("%d",a+i);
    for(int i=0;i<m;i++)
    {
        int l,r;
        scanf("%d%d",&l,&r);
        Q[i]={i,l,r};
    }

    len=max(1,(int)sqrt((double)n*n/m));

    sort(Q,Q+m,cmp);

    int i=0,j=1;
    for(int k=0;k<m;k++)
    {
        int l=Q[k].l,r=Q[k].r;
        while(i<r) add(a[++i]);
        while(j>l) add(a[--j]);
        while(i>r) dele(a[i--]);
        while(j<l) dele(a[j++]);
        ans[Q[k].id]=res;
    }

    for(int i=0;i<m;i++) printf("%lld\n",ans[i]);

    return 0;
}
```

2.9.2 莫队套分块

```
// 莫队 + 分块 ：统计区间中在 [a,b] 值域内数的个数和不重复的数的个数

#include <bits/stdc++.h>

using namespace std;

typedef pair<int,int> pii;

const int mod=1e9+7,N=1e5+10;

int n,m,len;
int a[N],cnt[N],sum1[N],sum2[N];
pii ans[N];
struct q
{
    int id,l,r,a,b;
}Q[N];

bool cmp(q &A,q &B)
{
    if(A.l/len!=B.l/len) return A.l<B.l;
    if(A.l/len&1) return A.r>B.r;
    return A.r<B.r;
}

void add(int x)
{
    sum1[x/len]++;
    if(!cnt[x]) sum2[x/len]++;
    cnt[x]++;
}

void del(int x)
{
    sum1[x/len]--;
    cnt[x]--;
    if(!cnt[x]) sum2[x/len]--;
}

void query(int l,int r,int id)
```

```
{
    int res1=0,res2=0;
    if(l/len==r/len)
    {
        for(int i=l;i<=r;i++)
            res1+=cnt[i],res2+=(cnt[i]>0);
    }
    else
    {
        int i=l,j=r;
        while(i/len==l/len) res1+=cnt[i],res2+=(cnt[i]>0),i++;
        while(j/len==r/len) res1+=cnt[j],res2+=(cnt[j]>0),j--;
        for(int k=i/len;k<=j/len;k++)
↪   res1+=sum1[k],res2+=sum2[k];
    }
    ans[id].first=res1,ans[id].second=res2;
}

void solve()
{
    cin >> n >> m;
    len=(int)sqrt((double)n*n/m/2)+1;
    for(int i=1;i<=n;i++) scanf("%d",a+i);
    for(int i=1;i<=m;i++)
    {
        int l,r,a,b;
        scanf("%d%d%d%d",&l,&r,&a,&b);
        Q[i]={i,l,r,a,b};
    }

    sort(Q+1,Q+m+1,cmp);

    for(int k=1,i=0,j=1;k<=m;k++)
    {
        int l=Q[k].l,r=Q[k].r;
        while(i<r) add(a[++i]);
        while(j>l) add(a[--j]);
        while(i>r) del(a[i--]);
        while(j<l) del(a[j++]);
        query(Q[k].a,Q[k].b,Q[k].id);
    }

    for(int i=1;i<=m;i++) printf("%d
↪   %d\n",ans[i].first,ans[i].second);
```

```
}

int main()
{
    solve();
    return 0;
}
```

2.9.3 带修莫队

```
//数颜色：带修 求区间不重复的数的个数
#include <bits/stdc++.h>

using namespace std;

const int N=14e4+10,s=1e6+10;

int n,m,tot,qot,len;
int cnt[s],a[N],res,ans[N];
struct q
{
    int id,l,r,t;
}Q[N];

struct modify
{
    int x,y;
}M[N];

bool cmp(q &a,q &b)
{
    if(a.l/len!=b.l/len) return a.l<b.l;
    if(a.r/len!=b.r/len) return a.r<b.r;
    if(a.r/len&1)return a.t>b.t;
    return a.t<b.t;
}

inline void add(int x)
{
    if(!cnt[x]) res++;
    cnt[x]++;
}

inline void del(int x)
{
    cnt[x]--;
    if(!cnt[x]) res--;
}

inline void changet(int l,int r,int t)
{
```

```
    int x=M[t].x,y=M[t].y;
    if(x>=l&&x<=r) add(y),del(a[x]);
    M[t].y=a[x],a[x]=y;
}

int main()
{
    cin >> n >> m;
    for(int i=1;i<=n;i++) scanf("%d",a+i);
    for(int i=1;i<=m;i++)
    {
        char str[2];
        int x,y;
        scanf("%s%d%d",str,&x,&y);
        if(*str=='R')
        {
            ++tot;
            M[tot]={x,y};
        }
        else qot++,Q[qot]={qot,x,y,tot};
    }

    len=cbrt((double)n*n*max(1,tot)/qot)+1;
    sort(Q+1,Q+qot+1,cmp);

    for(int k=1,i=1,j=0,tt=0;k<=qot;k++)
    {
        int l=Q[k].l,r=Q[k].r,t=Q[k].t;
        while(j<r) add(a[++j]);
        while(i>l) add(a[--i]);
        while(j>r) del(a[j--]);
        while(i<l) del(a[i++]);
        while(tt<t) changet(i,j,++tt);
        while(tt>t) changet(i,j,tt--);
        ans[Q[k].id]=res;
    }

    for(int i=1;i<=qot;i++) printf("%d\n",ans[i]);

    return 0;
}
```

2.9.4 回滚莫队 1(只加不删)

```cpp
// 只加不删的莫队 ：求区间 a[i]*cnt[a[i]] 的最大值
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int N=1e5+10;

int n,m,len,a[N],cnt[N];
ll ans[N];
struct q
{
    int id,l,r;
}Q[N];
vector<int> num;

bool cmp(q &a,q &b)
{
    if(a.l/len!=b.l/len) return a.l<b.l;
    return a.r<b.r;
}

int main()
{
    cin >> n >> m;
    len=(int)sqrt((double)n*n/m)+1;
    for(int i=1;i<=n;i++)
    {
        scanf("%d",a+i);
        num.push_back(a[i]);
    }

    sort(num.begin(),num.end());
    num.erase(unique(num.begin(),num.end()),num.end());
    for(int i=1;i<=n;i++)
↪   a[i]=lower_bound(num.begin(),num.end(),a[i])-num.begin();

    for(int i=1;i<=m;i++)
    {
        int l,r;
```

```
        scanf("%d%d",&l,&r);
        Q[i]={i,l,r};
    }

    sort(Q+1,Q+m+1,cmp);

    for(int k=1;k<=m;)
    {
        int p=k;
        while(p<=m&&Q[p].l/len==Q[k].l/len) p++;
        int right=Q[k].l/len*len+len-1;

        while(k<p&&Q[k].r<=right)
        {
            ll res=0;
            int l=Q[k].l,r=Q[k].r;
            for(int i=l;i<=r;i++)
            {
                cnt[a[i]]++;
                res=max(res,(ll)cnt[a[i]]*num[a[i]]);
            }
            ans[Q[k].id]=res;
            for(int i=l;i<=r;i++) cnt[a[i]]--;
            k++;
        }

        ll res=0;
        int j=right;
        while(k<p)
        {
            int l=Q[k].l,r=Q[k].r;
            while(j<r)
            {
                ++j;
                cnt[a[j]]++;
                res=max(res,(ll)cnt[a[j]]*num[a[j]]);
            }
            ll backup=res;
            for(int i=right;i>=l;i--)
            {
                cnt[a[i]]++;
                res=max(res,(ll)cnt[a[i]]*num[a[i]]);
            }
            ans[Q[k].id]=res;
```

```
            res=backup;
            for(int i=l;i<=right;i++) cnt[a[i]]--;
            k++;
        }

        memset(cnt,0,sizeof cnt);
    }

    for(int i=1;i<=m;i++) printf("%lld\n",ans[i]);

    return 0;
}
```

2.9.5 回滚莫队 2(只删不加)

```cpp
// 只删不加 ：求区间 mex
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int N=2e5+10;

int n,m,len,a[N],cnt[N];
int ans[N];
struct q
{
    int id,l,r;
}Q[N];
vector<int> num;

bool cmp(q &a,q &b)
{
    if(a.l/len!=b.l/len) return a.l<b.l;
    return a.r>b.r;
}

int main()
{
    cin >> n >> m;
    len=(int)sqrt(n)+1;
    for(int i=1;i<=n;i++) scanf("%d",a+i);

    for(int i=1;i<=m;i++)
    {
        int l,r;
        scanf("%d%d",&l,&r);
        Q[i]={i,l,r};
    }

    sort(Q+1,Q+m+1,cmp);

    for(int k=1;k<=m;)
    {
        int p=k;
```

```
    while(p<=m&&Q[p].l/len==Q[k].l/len) p++;
    int left=Q[k].l/len*len;
    int right=left+len-1;

    while(k<p&&Q[k].r<=right)
    {
        int res=0;
        int l=Q[k].l,r=Q[k].r;
        for(int i=l;i<=r;i++) cnt[a[i]]++;
        while(cnt[res]) res++;
        ans[Q[k].id]=res;
        for(int i=l;i<=r;i++) cnt[a[i]]--;
        k++;
    }
    if(k>=p) continue;
    int res=0;
    int j=Q[k].r;
    for(int i=left;i<=j;i++) cnt[a[i]]++;
    while(cnt[res]) res++;
    while(k<p)
    {
        int l=Q[k].l,r=Q[k].r;
        while(j>r)
        {
            cnt[a[j]]--;
            if(!cnt[a[j]]) res=min(res,a[j]);
            --j;
        }
        int backup=res;
        for(int i=left;i<l;i++)
        {
            cnt[a[i]]--;
            if(!cnt[a[i]]) res=min(res,a[i]);
        }
        ans[Q[k].id]=res;
        res=backup;
        for(int i=left;i<l;i++) cnt[a[i]]++;
        k++;
    }

    memset(cnt,0,sizeof cnt);
}

for(int i=1;i<=m;i++) printf("%d\n",ans[i]);
```

```
    return 0;
}
```

2.9.6 树上莫队

```cpp
/*
利用括号序，将两个点之间的路径变为连续的区间
具体而言 设 l[u] < l[v]
(1) 若 lca(u,v) = u 则 u,v 之间的路径就是括号序列中 [l[u],
↪ l[v]] 这一段里只出现一次的节点
(2) 否则，u,v 之间的路径就是 [r[u], l[v]] 这一段里只出现一次的节
↪ 点 + lca 这个节点
因为只计算只出现一次的点 开一个 cnt[] 记录点的权值信息，同时开
↪ st[u] 记录编号为 u 的这个点的出现次数
一旦 st[u] 为偶数，就要删除这个点的权值信息
求路径上出现的不重复数的个数
*/
#include <bits/stdc++.h>

using namespace std;

const int N=1e5+10;

int n,m,a[N],cnt[N],st[N],l[N],r[N],id[N],tot,ans[N],res;
int p[20][N],dep[N];
vector<int> nums,e[N];
array<int,4> query[N];
int len;

bool cmp(array<int,4> &a,array<int,4> &b)
{
    if(a[0]/len!=b[0]/len) return a[0]<b[0];
    if(a[0]/len&1) return a[1]>b[1];
    return a[1]<b[1];
}

void dfs(int u,int fa)
{
    p[0][u]=fa,dep[u]=dep[fa]+1;
    l[u]=++tot;
    id[tot]=u;
    for(auto v : e[u])
    {
        if(v==fa) continue;
        dfs(v,u);
    }
```

```
    r[u]=++tot;
    id[tot]=u;
}

int lca(int u,int v)
{
    if(dep[v]>dep[u]) swap(u,v);
    int d=dep[u]-dep[v];
    for(int i=19;i>=0;i--)
        if(d>>i&1) u=p[i][u];
    if(u==v) return u;
    for(int i=19;i>=0;i--)
        if(p[i][u]!=p[i][v])
            u=p[i][u],v=p[i][v];
    return p[0][u];
}

void add(int x)
{
    st[x]^=1;
    if(st[x])
    {
        if(!cnt[a[x]]) res++;
        cnt[a[x]]++;
    }
    else
    {
        cnt[a[x]]--;
        if(!cnt[a[x]]) res--;
    }
}

int main()
{
    cin >> n >> m;
    for(int i=1;i<=n;i++)
↪   scanf("%d",a+i),nums.push_back(a[i]);
    sort(nums.begin(),nums.end());
    nums.erase(unique(nums.begin(),nums.end()),nums.end());
    for(int i=1;i<=n;i++)
↪   a[i]=lower_bound(nums.begin(),nums.end(),a[i])-nums.end()+1;
    for(int i=1;i<n;i++)
    {
        int a,b;
```

```
    scanf("%d%d",&a,&b);
    e[a].push_back(b);
    e[b].push_back(a);
}

dfs(1,0);

len=(int)sqrt((double)tot*tot/m)+1;

for(int j=1;j<=19;j++)
    for(int i=1;i<=n;i++)
        p[j][i]=p[j-1][p[j-1][i]];

for(int i=1;i<=m;i++)
{
    int x,y;
    scanf("%d%d",&x,&y);
    if(l[x]>l[y]) swap(x,y);
    int w=lca(x,y);
    if(x==w) query[i]={l[x],l[y],i,0};
    else query[i]={r[x],l[y],i,w};
}

sort(query+1,query+m+1,cmp);

for(int i=1,j=0,k=1;k<=m;k++)
{
    int l=query[k][0],r=query[k][1],w=query[k][3];
    while(j<r) add(id[++j]);
    while(i>l) add(id[--i]);
    while(j>r) add(id[j--]);
    while(i<l) add(id[i++]);
    if(w)
    {
        add(w);
        ans[query[k][2]]=res;
        add(w);
    }
    else ans[query[k][2]]=res;
}

for(int i=1;i<=m;i++) printf("%d\n",ans[i]);

return 0;
```

```
}
```

2.9.7 树上带修莫队

```cpp
// 树上带修
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int N=1e5+10;

int n,m,q,a[N],l[N],r[N],id[N*2],tot,tt,qq;
ll ans[N],res;
int p[20][N],dep[N],v[N],w[N];
vector<int> e[N];
int cnt[N],st[N];
struct node
{
    int l,r,w,t,id;
}query[N];
int len;
struct
{
    int x,y;
}modify[N];

bool cmp(node &a,node &b)
{
    if(a.l/len!=b.l/len) return a.l<b.l;
    if(a.r/len!=b.r/len) return a.r<b.r;
    return a.t<b.t;
}

void dfs(int u,int fa)
{
    p[0][u]=fa,dep[u]=dep[fa]+1;
    l[u]=++tot;
    id[tot]=u;
    for(auto v : e[u])
    {
        if(v==fa) continue;
        dfs(v,u);
    }
```

```
    r[u]=++tot;
    id[tot]=u;
}

int lca(int u,int v)
{
    if(dep[v]>dep[u]) swap(u,v);
    int d=dep[u]-dep[v];
    for(int i=19;i>=0;i--)
        if(d>>i&1) u=p[i][u];
    if(u==v) return u;
    for(int i=19;i>=0;i--)
        if(p[i][u]!=p[i][v])
            u=p[i][u],v=p[i][v];
    return p[0][u];
}

void add(int x)
{
    int t=a[x];
    st[x]^=1;
    if(st[x])
    {
        cnt[t]++;
        res+=(ll)v[t]*w[cnt[t]];
    }
    else
    {
        res-=(ll)v[t]*w[cnt[t]];
        cnt[t]--;
    }
}

void swapp(int m1,int m2)
{
    res-=(ll)v[m1]*w[cnt[m1]];
    cnt[m1]--;
    cnt[m2]++;
    res+=(ll)v[m2]*w[cnt[m2]];
}

void change(int t,int L,int R)
{
    int x=modify[t].x,y=modify[t].y;
```

```
    // 判断是否在区间内
    if(st[x]) swapp(a[x],y);
    swap(a[x],modify[t].y);
}

int main()
{
    cin >> n >> m >> q;
    for(int i=1;i<=m;i++) scanf("%d",v+i);
    for(int i=1;i<=n;i++) scanf("%d",w+i);
    for(int i=1;i<n;i++)
    {
        int a,b;
        scanf("%d%d",&a,&b);
        e[a].push_back(b);
        e[b].push_back(a);
    }
    dfs(1,0);
    for(int j=1;j<=19;j++)
        for(int i=1;i<=n;i++)
            p[j][i]=p[j-1][p[j-1][i]];

    for(int i=1;i<=n;i++) scanf("%d",a+i);
    for(int i=1;i<=q;i++)
    {
        int opt,x,y;
        scanf("%d%d%d",&opt,&x,&y);
        if(opt)
        {
            ++qq;
            if(l[x]>l[y]) swap(x,y);
            int w=lca(x,y);
            if(x==w) query[qq]={l[x],l[y],0,tt,qq};
            else query[qq]={r[x],l[y],w,tt,qq};
        }
        else
        {
            ++tt;
            modify[tt]={x,y};
        }
    }

    len=(int)cbrt((double)tot*tot*max(1,tt)/qq)+1;
```

```
    sort(query+1,query+qq+1,cmp);

    for(int i=1,j=0,T=0,k=1;k<=qq;k++)
    {
        int
↪   l=query[k].l,r=query[k].r,w=query[k].w,t=query[k].t;
        while(j<r) add(id[++j]);
        while(i>l) add(id[--i]);
        while(j>r) add(id[j--]);
        while(i<l) add(id[i++]);
        while(T<t) change(++T,i,j);
        while(T>t) change(T--,i,j);
        if(w)
        {
            add(w);
            ans[query[k].id]=res;
            add(w);
        }
        else ans[query[k].id]=res;
    }

    for(int i=1;i<=qq;i++) printf("%lld\n",ans[i]);

    return 0;
}
```

2.9.8 二次离线莫队

```cpp
// 二次离线
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int N=1e5+10;

int n,m,k,a[N],len;
vector<int> nums;
ll f[N],g[N],ans[N];
struct q
{
    int l,r,id;
    ll res;
}query[N];
struct node
{
    int l,r,id,t;
};
vector<node> range[N];

bool cmp(q &a,q &b)
{
    if(a.l/len!=b.l/len) return a.l<b.l;
    if(a.l/len&1) return a.r>b.r;
    return a.r<b.r;
}

int main()
{
    cin >> n >> m >> k;
    len=(int)sqrt((double)n*n/m)+1;
    for(int i=1;i<=n;i++) scanf("%d",a+i);
    for(int i=0;i<(1<<14);i++)
        if(__builtin_popcount(i)==k)
            nums.push_back(i);

    for(int i=1;i<=n;i++)
    {
```

```
    for(auto x : nums) g[a[i]^x]++;
    f[i]=g[a[i+1]];
}

for(int i=1;i<=m;i++)
{
    int l,r;
    scanf("%d%d",&l,&r);
    query[i]={l,r,i};
}

sort(query+1,query+m+1,cmp);

for(int i=1,L=1,R=0;i<=m;i++)
{
    int l=query[i].l,r=query[i].r;
    if(R<r) range[L-1].push_back({R+1,r,i,-1});
    while(R<r) query[i].res+=f[R++];
    if(R>r) range[L-1].push_back({r+1,R,i,1});
    while(R>r) query[i].res-=f[--R];
    if(L<l) range[R].push_back({L,l-1,i,-1});
    while(L<l) query[i].res+=f[L-1]+!k,L++;
    if(L>l) range[R].push_back({l,L-1,i,1});
    while(L>l) query[i].res-=f[L-2]+!k,L--;
}


memset(g,0,sizeof g);
for(int i=1;i<=n;i++)
{
    for(auto x : nums) g[a[i]^x]++;
    for(auto &q : range[i])
    {
        int l=q.l,r=q.r,id=q.id,t=q.t;
        for(int x=l;x<=r;x++)
            query[id].res+=g[a[x]]*t;
    }
}

for(int i=2;i<=m;i++) query[i].res+=query[i-1].res;
for(int i=1;i<=m;i++) ans[query[i].id]=query[i].res;
for(int i=1;i<=m;i++) printf("%lld\n",ans[i]);

return 0;
```

```
}
```

3 Tree

3.1 树 Hash

3.1.1 判断同构

```cpp
// 求无根树是否同构 : hash + 换根
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef unsigned long long ull;

const int mod=1e9+7,N=110;

const ull mask =
↪  std::chrono::steady_clock::now().time_since_epoch().count();

ull shift(ull x) {
  x ^= mask;
  x ^= x << 13;
  x ^= x >> 7;
  x ^= x << 17;
  x ^= mask;
  return x;
}

ull hh[N],res;
int n,m,k;
map<ull,int> mp;
vector<int> e[N];

// 求以 u 为根的树的 hash 值
void dfs(int u)
{
    hh[u]=1;
    for(auto v : e[u])
    {
        dfs(v);
        hh[u]+=shift(hh[v]);
    }
}
```

```
//换根
void dp(int u)
{
    res=min(res,hh[u]);
    for(auto v : e[u])
    {
        hh[v]+=shift(hh[u]-shift(hh[v]));
        dp(v);
    }
}

void solve()
{
    cin >> m;
    for(int t=1;t<=m;t++)
    {
        cin >> n;
        for(int i=1;i<=n;i++) e[i].clear();
        int root=-1;
        for(int i=1;i<=n;i++)
        {
            int b;
            scanf("%d",&b);
            if(b) e[b].push_back(i);
            else root=i;
        }
        res=18446744073709551615ull;
        dfs(root);
        dp(root);
        if(!mp.count(res)) mp[res]=t;
        printf("%d\n",mp[res]);
    }
}

int main()
{
    solve();
    return 0;
}
```

3.1.2 判断对称

```cpp
/*
给出一棵以 1 为根的树 判断这棵树是否对称
对称 ：若有偶数个子树 则要求两两同构
        若有奇数个 可以有一个在中间 其他的两两同构 在中间的必须自己也对称
判断是否对称：将所有子树的 hash 值异或起来 如果为 0 一定对称
             如果不为 0 枚举放在中间那棵子树
             如果 sum ^ 子树 hash 值 ==0 且子树本身也对称 那么整棵树就可以对称
*/
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef unsigned long long ull;
typedef pair<int,int> pii;

const int mod=1e9+7,N=2e5+10;

const ull mask =
    std::chrono::steady_clock::now().time_since_epoch().count();

ull shift(ull x) {
  x ^= mask;
  x ^= x << 13;
  x ^= x >> 7;
  x ^= x << 17;
  x ^= mask;
  return x;
}

ull hh[N];
int n;
vector<int> e[N];
bool f[N];

void dfs(int u,int fa)
{
    hh[u]=1;
    ull sum=0;
```

```
    for(auto v : e[u])
    {
        if(v==fa) continue;
        dfs(v,u);
        hh[u]+=shift(hh[v]);
        sum^=shift(hh[v]);
    }

    if(!sum) f[u]=1;
    else
    {
        for(auto v : e[u])
        {
            if(v==fa) continue;
            if(sum==shift(hh[v])&&f[v])
            {
                f[u]=1;
                break;
            }
        }
    }
}

void solve()
{
    cin >> n;
    for(int i=1;i<=n;i++) e[i].clear(),f[i]=0;
    for(int i=0;i<n-1;i++)
    {
        int a,b;
        scanf("%d%d",&a,&b);
        e[a].push_back(b);
        e[b].push_back(a);
    }

    dfs(1,0);

    if(f[1]) puts("YES");
    else puts("NO");
}

int main()
{
    int _;
```

```
    cin >> _;
    while(_--) solve();
    return 0;
}
```

3.2 树上启发式合并

```cpp
/*
询问所有子树中，出现最多次数的颜色，有多少种
O(nlogn)
*/
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int N=1e5+10;

int n,c[N];
vector<int> e[N];
int son[N],siz[N],cnt[N];
int l[N],r[N],id[N],tot;
ll ans[N],sum,maxv;

void dfs(int u,int fa)
{
    l[u]=++tot;
    id[tot]=u;
    siz[u]=1;
    for(auto j : e[u])
    {
        if(j==fa) continue;
        dfs(j,u);
        siz[u]+=siz[j];
        if(siz[j]>siz[son[u]]) son[u]=j;
    }
    r[u]=tot;
}

void dfs2(int u,int fa,bool keep)
{
    for(auto j : e[u])
    {
        if(j==fa||j==son[u]) continue;
        dfs2(j,u,0);
    }

    if(son[u]) dfs2(son[u],u,1);
```

```
    for(auto j: e[u])
    {
        if(j==fa||j==son[u]) continue;
        for(int k=l[j];k<=r[j];k++)
        {
            int x=c[id[k]];
            cnt[x]++;
            if(cnt[x]>maxv) maxv=cnt[x],sum=0;
            if(cnt[x]==maxv) sum+=x;
        }
    }

    int x=c[u];
    cnt[x]++;
    if(cnt[x]>maxv) maxv=cnt[x],sum=0;
    if(cnt[x]==maxv) sum+=x;

    ans[u]=sum;
    if(!keep)
    {
        for(int i=l[u];i<=r[u];i++) cnt[c[id[i]]]--;
        maxv=0,sum=0;
    }
}

int main()
{
    cin >> n;
    for(int i=1;i<=n;i++) scanf("%d",c+i);
    for(int i=1;i<n;i++)
    {
        int a,b;
        scanf("%d%d",&a,&b);
        e[a].push_back(b);
        e[b].push_back(a);
    }

    dfs(1,0);
    dfs2(1,0,1);

    for(int i=1;i<=n;i++) printf("%lld ",ans[i]);

    return 0;
```

```
}
```

3.3 LCA

3.3.1 倍增

```cpp
// 预处理 O(nlogn) 单次询问 O(logn)
#include <bits/stdc++.h>

using namespace std;

const int mod=1e9+7,N=5e5+10;

int n,m,root;
int p[N][20],d[N];
vector<int> e[N];

void dfs(int u,int fa)
{
    p[u][0]=fa;
    for(auto v : e[u])
    {
        if(v==fa) continue;
        d[v]=d[u]+1;
        dfs(v,u);
    }
}

void solve()
{
    cin >> n >> m >> root;
    for(int i=1;i<n;i++)
    {
        int x,y;
        scanf("%d%d",&x,&y);
        e[x].push_back(y);
        e[y].push_back(x);
    }

    dfs(root,0);
    for(int j=1;j<=19;j++)
        for(int i=1;i<=n;i++)
            p[i][j]=p[p[i][j-1]][j-1];

    while(m--)
```

```
    {
        int a,b;
        scanf("%d%d",&a,&b);
        if(d[a]<d[b]) swap(a,b);
        int h=d[a]-d[b];
        for(int j=19;j>=0;j--)
            if(h>>j&1) a=p[a][j];
        if(a==b) {
            printf("%d\n",a);
            continue;
        }
        for(int j=19;j>=0;j--)
            if(p[a][j]!=p[b][j]) a=p[a][j],b=p[b][j];

        printf("%d\n",p[a][0]);
    }
}

int main()
{
    solve();
    return 0;
}
```

```cpp
/*
给定 n 点 m 边的无向边带权图  n,m <= 2e5 对于每一条边 分别求出包含这条边的最小生成树

先求出全局最小生成树，对于任意一条边而言
(1) 如果这条边在最小生成树里 那么答案就是全局最小生成树
(2) 如果不是 那么 对于这条边 (u,v) 在全局最小生成树中 找到 u 到 v 路径上边权最大的边
    去掉那条边 把这条边 (u,v) 加入生成树 就是权重最小的生成树
    求路径:LCA 维护倍增的父亲节点 同时维护路径上边权的最大值 预处理 O(nlogn)  查询 O(logn)
*/
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef pair<int,int> pii;

const int mod=1e9+7,N=2e5+10;

int n,m,p[N],pmax[19][N],f[19][N],depth[N];
vector<pii> e[N];
ll res,ans2[N];
struct edge
{
    int v,l,r,id;
}E[N];

void dfs(int u,int fa)
{
    for(auto v: e[u])
    {
        int j=v.first,w=v.second;
        if(j==fa) continue;
        depth[j]=depth[u]+1;
        pmax[0][j]=w;
        f[0][j]=u;
        dfs(j,u);
    }
}

bool cmp(edge &a,edge &b)
```

```
{
    return a.v<b.v;
}

int find(int x)
{
    if(p[x]!=x) p[x]=find(p[x]);
    return p[x];
}

void solve()
{
    cin >> n >> m;
    for(int i=1;i<=n;i++) p[i]=i;
    for(int i=1;i<=m;i++)
    {
        int u,v,x;
        scanf("%d%d%d",&u,&v,&x);
        E[i]={x,u,v,i};
    }

    sort(E+1,E+m+1,cmp);

    for(int i=1;i<=m;i++)
    {
        int u=E[i].l,v=E[i].r;
        int a=find(u),b=find(v);
        if(a==b) continue;
        e[u].push_back({v,E[i].v});
        e[v].push_back({u,E[i].v});
        res+=E[i].v;
        p[a]=b;
    }

    dfs(1,0);

    for(int j=1;j<=18;j++)
        for(int i=1;i<=n;i++)
            f[j][i]=f[j-1][f[j-1][i]],
            pmax[j][i]=max(pmax[j-1][i],pmax[j-1][f[j-1][i]]);

    for(int i=1;i<=m;i++)
    {
        ll ans=0;
```

```
        int u=E[i].l,v=E[i].r,id=E[i].id;
        if(depth[u]<depth[v]) swap(u,v);
        int d=depth[u]-depth[v];
        for(int j=18;j>=0;j--)
            if(d>>j&1) ans=max(ans,(ll)pmax[j][u]),u=f[j][u];

        if(u!=v)
        {
            for(int j=18;j>=0;j--)
            {
                if(f[j][u]!=f[j][v])
                {
                    ans=max(ans,(ll)pmax[j][u]);
                    ans=max(ans,(ll)pmax[j][v]);
                    u=f[j][u];
                    v=f[j][v];
                }
            }
            ans=max(ans,(ll)pmax[0][u]);
            ans=max(ans,(ll)pmax[0][v]);
        }
        ans2[id]=res-ans+E[i].v;
    }

    for(int i=1;i<=m;i++) printf("%lld\n",ans2[i]);

}

int main()
{
    solve();
    return 0;
}
```

3.3.2 欧拉序

```cpp
/*
欧拉序转为 RMQ 问题
对两个点 u,v 他们的 lca 一定是
区间 pos[u] - pos[v] 中深度最小的那个点
*/
#include <bits/stdc++.h>

using namespace std;

typedef pair<int,int> pii;

const int mod=1e9+7,N=5e5+10;

int n,m,root;
vector<int> e[N];
// pos[u] 记录 u 第一次出现的位置
int depth[N],pos[N],tot;
pii id[N*2],f[21][N*2];

void dfs(int u,int fa)
{
    depth[u]=depth[fa]+1;
    pos[u]=++tot;
    f[0][tot]={depth[u],u};
    for(auto v: e[u])
    {
        if(v==fa) continue;
        dfs(v,u);
        f[0][++tot]={depth[u],u};
    }
}

void solve()
{
    cin >> n >> m >> root;
    for(int i=1;i<n;i++)
    {
        int a,b;
        scanf("%d%d",&a,&b);
        e[a].push_back(b);
        e[b].push_back(a);
```

```
    }

    dfs(root,0);
    for(int j=1;j<=20;j++)
        for(int i=1;i+(1<<j)-1<=tot;i++)
            f[j][i]=min(f[j-1][i],f[j-1][i+(1<<j-1)]);

    while(m--)
    {
        int a,b;
        scanf("%d%d",&a,&b);
        if(pos[a]>pos[b]) swap(a,b);
        int k=31-__builtin_clz(pos[b]-pos[a]+1);

↪   printf("%d\n",min(f[k][pos[a]],f[k][pos[b]-(1<<k)+1]).second);
    }

}

int main()
{
    solve();
    return 0;
}
```

3.3.3 Tarjan

```cpp
// O(n + m)
#include <bits/stdc++.h>

using namespace std;

typedef pair<int,int> pii;

const int mod=1e9+7,N=5e5+10;

int n,m,root,p[N];
int ans[N];
bool st[N];
vector<int> e[N];
vector<pii> query[N];


int find(int x)
{
    if(x!=p[x]) p[x]=find(p[x]);
    return p[x];
}

void tarjan(int u,int fa)
{
    st[u]=1;
    for(auto v : e[u])
    {
        if(v==fa) continue;
        tarjan(v,u);
        p[v]=u;
    }
    for(auto x : query[u])
    {
        int v=x.first,id=x.second;
        if(st[v]) ans[id]=find(p[v]);
    }
}

void solve()
{
    cin >> n >> m >> root;
```

```
    for(int i=1;i<=n;i++) p[i]=i;
    for(int i=1;i<n;i++)
    {
        int a,b;
        scanf("%d%d",&a,&b);
        e[a].push_back(b);
        e[b].push_back(a);
    }
    for(int i=1;i<=m;i++)
    {
        int a,b;
        scanf("%d%d",&a,&b);
        query[a].push_back({b,i});
        query[b].push_back({a,i});
    }

    tarjan(root,0);
    for(int i=1;i<=m;i++) printf("%d\n",ans[i]);

}

int main()
{
    solve();
    return 0;
}
```

```cpp
// tarjan 求路径上边权的最值
#include <bits/stdc++.h>

using namespace std;

typedef pair<int,int> pii;

const int mod=1e9+7,N=1e5+10,INF=1e9;

int n,m,root,p[N];
int maxv[N],minv[N];
pii ans[N];
bool st[N];
vector<pii> query[N],e[N];
vector<array<int,3>> deal[N];

int find(int x)
{
    if(x!=p[x])
    {
        int t=p[x];
        p[x]=find(p[x]);
        minv[x]=min(minv[x],minv[t]);
        maxv[x]=max(maxv[x],maxv[t]);
    }
    return p[x];
}

void tarjan(int u,int fa)
{
    minv[u]=INF,maxv[u]=0;
    st[u]=1;
    for(auto v : e[u])
    {
        int j=v.first,w=v.second;
        if(j==fa) continue;
        tarjan(j,u);
        maxv[j]=minv[j]=w;
        p[j]=u;
    }
    for(auto x : query[u])
    {
```

```
        int v=x.first,id=x.second;
        if(st[v]) deal[find(v)].push_back({u,v,id});
    }

    for(auto x : deal[u])
    {
        find(x[0]),find(x[1]);
        int t1=min(minv[x[0]],minv[x[1]]);
        int t2=max(maxv[x[0]],maxv[x[1]]);
        ans[x[2]]={t1,t2};
    }
}

void solve()
{
    cin >> n;
    for(int i=1;i<=n;i++) p[i]=i;
    for(int i=1;i<n;i++)
    {
        int a,b,c;
        scanf("%d%d%d",&a,&b,&c);
        e[a].push_back({b,c});
        e[b].push_back({a,c});
    }
    cin >> m;
    for(int i=1;i<=m;i++)
    {
        int a,b;
        scanf("%d%d",&a,&b);
        query[a].push_back({b,i});
        query[b].push_back({a,i});
    }

    tarjan(1,0);
    for(int i=1;i<=m;i++) printf("%d
↪   %d\n",ans[i].first,ans[i].second);

}

int main()
{
    solve();
    return 0;
}
```

3.4 DFS 序/树上差分

3.4.1 子树修改

```
/*
(1) 单点修改 子树查询 : dfs 序相当于单点修改 区间查询 树状数组即
↪ 可
(2) 子树修改 单点查询 : dfs 序相当于区间修改 单点查询 维护序列的
↪ 差分数组即可
(3) 子树修改  子树查询： dfs 序 相当于区间修改 区间查询 维护两个
↪ dfs 序差分树状数组即可
*/
// (3)
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int mod=1e9+7,N=1e6+10;

int n,m,root,p[N];
ll tr1[N],tr2[N],res[N];
vector<int> e[N];
int l[N],r[N],tot;


int lowbit(int x)
{
    return x&-x;
}

void add(int x,ll c,ll tr[])
{
    for(int i=x;i<=n;i+=lowbit(i)) tr[i]+=c;
}

ll sum(int x,ll tr[])
{
    ll res=0;
    for(int i=x;i;i-=lowbit(i)) res+=tr[i];
    return res;
}
```

```
ll prefix(int x)
{
    return (ll)(x+1)*sum(x,tr1)-sum(x,tr2);
}

void dfs(int u,int fa)
{
    l[u]=++tot;
    for(auto v : e[u])
    {
        if(v==fa) continue;
        dfs(v,u);
    }
    r[u]=tot;
}

void solve()
{
    cin >> n >> m >> root;
    for(int i=1;i<=n;i++) scanf("%d",p+i);
    for(int i=1;i<n;i++)
    {
        int a,b;
        scanf("%d%d",&a,&b);
        e[a].push_back(b);
        e[b].push_back(a);
    }

    dfs(root,0);

    for(int i=1;i<=n;i++) res[l[i]]=p[i];
    for(int i=n;i>0;i--) res[i]-=res[i-1];
    for(int i=1;i<=n;i++)
↪   add(i,res[i],tr1),add(i,res[i]*i,tr2);

    while(m--)
    {
        int opt,a,x;
        scanf("%d%d",&opt,&a);
        if(opt==1)
        {
            scanf("%d",&x);
            add(l[a],x,tr1),add(r[a]+1,-x,tr1);
```

```
↪   add(l[a],(ll)l[a]*x,tr2),add(r[a]+1,-(ll)(r[a]+1)*x,tr2);
        }
        else printf("%lld\n",prefix(r[a])-prefix(l[a]-1));
    }
}

int main()
{
    solve();
    return 0;
}
```

3.4.2 树链修改

```
/*
树上差分：真正的节点值是整棵子树的差分数组值之和
(1) 修改路径上的所有点：变成修改单点的信息，修改一点，
   那么这一点到根的路径上的所有点都会修改
(2) 求单点点权 ：子树和
(3) 求子树和 ：推公式 与点的深度有关
具体来说：
(1) 点差分 ：修改 u -> v 的路径上的所有点  --> 修改 u -> lca 和
↪  v -> lca 的下面一个节点 两条链
            也就是 d[u]++,d[v]++,d[lca.father]--,d[lca]--;
(2) 边差分 ：定义某一点代表这个点到其父亲节点的边 那么修改 u -> v
↪  路径上的所有边
            也就是 d[u]++,d[v]--,d[lca]-=2;

维护两个树状数组，一个 d[u]
另一个 depth[u]*d[u] 的值
子树 u 的和 = sum( d[v] * (depth[v] - depth[u] + 1) )
= sum_tr2(l[u],r[u]) - sum_tr1(l[u],r[u]) * (depth[u] - 1)
*/

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef pair<int,int> pii;

const int mod=1e9+7,N=1e6+10;

int n,m,root,tot,idx;
int depth[N],a[N],pos[N],l[N],r[N],par[N];
int p[22][2*N];
vector<int> e[N];
ll tr1[N],tr2[N];

int lowbit(int x)
{
    return x&-x;
}

void add(int x,ll c,ll tr[])
{
```

```
    if(!x) return;
    for(int i=x;i<=n;i+=lowbit(i)) tr[i]+=c;
}

ll sum(int x,ll tr[])
{
    ll res=0;
    for(int i=x;i;i-=lowbit(i)) res+=tr[i];
    return res;
}


void dfs(int u,int fa)
{
    par[u]=fa;
    l[u]=++idx;
    depth[u]=depth[fa]+1;
    pos[u]=++tot;
    p[0][tot]=u;
    for(auto v : e[u])
    {
        if(v==fa) continue;
        dfs(v,u);
        p[0][++tot]=u;
    }
    r[u]=idx;
}

int lca(int u,int v)
{
    int a=pos[u],b=pos[v];
    if(a>b) swap(a,b);
    int k=31-__builtin_clz(b-a+1);
    if(depth[p[k][a]]<depth[p[k][b-(1<<k)+1]]) return p[k][a];
    else return p[k][b-(1<<k)+1];
}

void add_list(int u,int v,int x)
{
    int w=lca(u,v);
    add(l[u],x,tr1);
    add(l[u],(ll)depth[u]*x,tr2);
    add(l[v],x,tr1);
    add(l[v],(ll)depth[v]*x,tr2);
```

```
    add(l[w],-x,tr1);
    add(l[w],(ll)depth[w]*(-x),tr2);
    add(l[par[w]],-x,tr1);
    add(l[par[w]],(ll)depth[par[w]]*(-x),tr2);
}

ll query(int u)
{
   ll sum1=sum(r[u],tr2)-sum(l[u]-1,tr2);
   ll sum2=sum(r[u],tr1)-sum(l[u]-1,tr1);
   return sum1-sum2*(depth[u]-1);
}

void solve()
{
    cin >> n >> m >> root;
    for(int i=1;i<=n;i++) scanf("%d",a+i);
    for(int i=1;i<n;i++)
    {
        int a,b;
        scanf("%d%d",&a,&b);
        e[a].push_back(b);
        e[b].push_back(a);
    }
    dfs(root,0);

    for(int j=1;j<=21;j++)
        for(int i=1;i+(1<<j)-1<=tot;i++)
            if(depth[p[j-1][i]]<depth[p[j-1][i+(1<<j-1)]])
↪   p[j][i]=p[j-1][i];
            else p[j][i]=p[j-1][i+(1<<j-1)];

    for(int i=1;i<=n;i++) add_list(i,i,a[i]);

    while(m--)
    {
        int opt,a,b,x;
        scanf("%d%d",&opt,&a);
        if(opt==1)
        {
            scanf("%d%d",&b,&x);
            add_list(a,b,x);
        }
```

```
        else if(opt==2)
↪   printf("%lld\n",sum(r[a],tr1)-sum(l[a]-1,tr1));
        else printf("%lld\n",query(a));
    }
}

int main()
{
    solve();
    return 0;
}
```

3.4.3 树链查询

```
/*
(7) 单点修改 树链查询
维护所有树链 变成 子树修改 单点查询 再差分 变成 dfs 序上的单点修
↪ 改 区间查询
(8) 子树修改 树链查询 :
维护树链的差分 推公式得 维护 d[u] 和 d[u] * depth[u]
子树修改 = [l[u],r[u]] 区间
所有点 v += x * (depth[v]-depth[u]+1)
= x * depth[v] + x * (1 - depth[u])
后一项变成差分数组 d[i] 的单点修改 前一项变成 tr2 的单点修改
树链值 = sum_tr1[l[u],r[u]] + sum_tr2[l[u],r[u]] * depth[u]

*/
// 1. 单点修改 2. 子树修改 3. 树链查询
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef pair<int,int> pii;

const int mod=1e9+7,N=1e6+10;

int n,m,root,tot,idx;
int depth[N],a[N],pos[N],l[N],r[N],par[N];
int p[22][2*N];
vector<int> e[N];
ll tr1[N],tr2[N];

int lowbit(int x)
{
    return x&-x;
}

void add(int x,ll c,ll tr[])
{
    if(!x) return;
    for(int i=x;i<=n;i+=lowbit(i)) tr[i]+=c;
}

ll sum(int x,ll tr[])
```

```
{
    ll res=0;
    for(int i=x;i;i-=lowbit(i)) res+=tr[i];
    return res;
}


void dfs(int u,int fa)
{
    par[u]=fa;
    l[u]=++idx;
    depth[u]=depth[fa]+1;
    pos[u]=++tot;
    p[0][tot]=u;
    for(auto v : e[u])
    {
        if(v==fa) continue;
        dfs(v,u);
        p[0][++tot]=u;
    }
    r[u]=idx;
}

int lca(int u,int v)
{
    int a=pos[u],b=pos[v];
    if(a>b) swap(a,b);
    int k=31-__builtin_clz(b-a+1);
    if(depth[p[k][a]]<depth[p[k][b-(1<<k)+1]]) return p[k][a];
    else return p[k][b-(1<<k)+1];
}

void query_one(int u,ll x)
{
    add(l[u],x,tr1);
    add(r[u]+1,-x,tr1);
}

void query_tree(int u,ll x)
{
    ll t=(1-depth[u])*x;
    add(l[u],t,tr1);
    add(r[u]+1,-t,tr1);
    add(l[u],x,tr2);
```

```
    add(r[u]+1,-x,tr2);
}

ll query(int u)
{
    ll t=sum(l[u],tr2)*depth[u];
    t+=sum(l[u],tr1);
    return t;
}

ll query_list(int u,int v)
{
    int w=lca(u,v);
    ll res=query(u)+query(v)-query(w)-query(par[w]);
    return res;
}

void solve()
{
    cin >> n >> m >> root;
    for(int i=1;i<=n;i++) scanf("%d",a+i);
    for(int i=1;i<n;i++)
    {
        int a,b;
        scanf("%d%d",&a,&b);
        e[a].push_back(b);
        e[b].push_back(a);
    }
    dfs(root,0);

    for(int j=1;j<=21;j++)
        for(int i=1;i+(1<<j)-1<=tot;i++)
            if(depth[p[j-1][i]]<depth[p[j-1][i+(1<<j-1)]])
↪   p[j][i]=p[j-1][i];
            else p[j][i]=p[j-1][i+(1<<j-1)];

    for(int i=1;i<=n;i++) query_one(i,a[i]);

    while(m--)
    {
        int opt,a,x;
        scanf("%d%d%d",&opt,&a,&x);
        if(opt==1) query_one(a,x);
        else if(opt==2) query_tree(a,x);
```

```
        else printf("%lld\n",query_list(a,x));
    }
}

int main()
{
    solve();
    return 0;
}
```

4 String

4.1 hash

```cpp
/*
base 要大于字符串元素种类
字符串元素不要映射成 0

*/
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int mod=1e9+7,N=2e5+10,base=131;
const int mod2=998244353,base2=13331;

int n,m,k;
int ha[N],b[N];
char s[N];

int hash_value(int l,int r)
{
    return (ha[r]-(ll)ha[l-1]*b[r-l+1]%mod+mod)%mod;
}

void solve()
{
    scanf("%d%s",&n,s+1);
    b[0]=1;
    for(int i=1;i<=n;i++)
    {
        b[i]=(ll)b[i-1]*base%mod;
        ha[i]=((ll)ha[i-1]*base+s[i])%mod;
    }
}

int main()
{
    solve();
    return 0;
}
```

4.2 kmp

```
/*
t = n - ne[n] 是 s 的最小循环覆盖 并且
其他的循环覆盖长度是 t 的倍数
假设 t | n , 那么 t 就是 s 的最小周期 否则 不存在周期
求 s 的 所有 border : ne[n] 是最长 border
ne[ne[n]] 是次长 以此类推
while(ne[n]) n=ne[n];
*/

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int mod=1e9+7,N=1e6+10,M=1e5+10;

int n,m;
char s[N],p[M];
int ne[M],f[N];

void kmp()
{
    ne[1]=0;
    for(int i=2,j=0;i<=m;i++)
    {
        while(j&&p[j+1]!=p[i]) j=ne[j];
        if(p[j+1]==p[i]) j++;
        ne[i]=j;
    }
    for(int i=1,j=0;i<=n;i++)
    {
        while((j==m)||(j&&p[j+1]!=s[i])) j=ne[j];
        if(p[j+1]==s[i]) j++;
        f[i]=j;
    }
}
```

4.3 exkmp

```
/*
求所有的 i s[i - n] 与 p 的最大前缀匹配
*/
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int mod=1e9+7,N=2e5+10;

int n,m,k;
int z[N],Z[N];
char s[N],p[N];

void exkmp()
{
    p[m+1]='@',s[n+1]='#';
    z[1]=0;
    int L=1,R=0;
    for(int i=2;i<=m;i++)
    {
        if(i>R) z[i]=0;
        else
        {
            int k=i-L+1;
            z[i]=min(z[k],R-i+1);
        }
        while(p[i+z[i]]==p[1+z[i]]) z[i]++;
        if(i+z[i]-1>R) L=i,R=i+z[i]-1;
    }

    L=0,R=0;
    for(int i=1;i<=n;i++)
    {
        if(i>R) Z[i]=0;
        else
        {
            int k=i-L+1;
            Z[i]=min(z[k],R-i+1);
        }
        while(s[i+Z[i]]==p[1+Z[i]]) Z[i]++;
```

```
        if(i+Z[i]-1>R) L=i,R=i+Z[i]-1;
    }
}
```

4.4 manacher

```
/*
最长回文子串问题
(1) 枚举中点 二分 + 字符串哈希  O(nlogn)
(2) manacher O(n)

mamacher
1. 首先插入无关字符 避免奇偶分类讨论 全部变成奇数情况 同时在首尾
↪  插入一个不可能和其他位置相等的哨兵 避免判边界
2. 记录一个 p[i] 表示以 i 为中点的最长回文半径 (包括中点)
3. 从前往后枚举中点 i 维护一个 [L,R] 区间 表示 < i 里 回文区间的
↪  R 最大的区间 也就是 [m-p[m]+1,m+p[m]-1]
4. 考虑求 p[i]
  (1) 如果 R < i 直接暴力求 p[i]
  (2) 如果 R >= i  求点 i 在区间 [L,R] 里的对称点 k = 2m-i
      若 Lk > L 那么 p[i] = p[k]
      否则 Lk <= L 那么 p[i] 至少 = p[k] 然后暴力扩展
  求完 p[i] 别忘记更新 [L,R]
  最后 res = max(p[i]) - 1

每次暴力扩展时 R 一定会增大 而 R 最多增大 O(n)  因此总时间 O(n)
*/
#include <bits/stdc++.h>

using namespace std;

const int mod=1e9+7,N=11e6+10;

int n,m,k;
int p[N*2];
char s[N],t[N*2];

void manacher()
{
    t[0]='@';
    t[++m]='$';
    for(int i=1;i<=n;i++) t[++m]=s[i],t[++m]='$';
    t[m+1]='~';
    int M=0,R=0;
    for(int i=1;i<=m;i++)
    {
        if(i>R) p[i]=1;
        else p[i]=min(p[2*M-i],R-i+1);
```

```
        while(t[i-p[i]]==t[i+p[i]]) p[i]++;
        if(i+p[i]-1>R) M=i,R=i+p[i]-1;
    }
    int res=0;
    for(int i=1;i<=m;i++) res=max(res,p[i]);
    printf("%d\n",res-1);
}
```

4.5 getmin

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int mod=1e9+7,N=2e5+10;

int n;
char s[N*2];

void getmin()
{
    for(int i=1;i<=n;i++) s[i+n]=s[i];
    int i=1,j=2;
    while(j<=n)
    {
        int k=0;
        while(k<n&&s[i+k]==s[j+k]) k++;
        if(s[i+k]>s[j+k]) i+=k+1;
        else j+=k+1;
        if(i==j) j++;
        if(i>j) swap(i,j);
    }

    for(int l=i;l<=i+n-1;l++) printf("%c",s[l]);
}
```

5 Math

5.1 数论

5.1.1 线性筛

```cpp
// 线性筛
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int mod=1e9+7,N=2e5+10;

int n,cnt;
int p[N];//p[i]:i 的最小质因子
int pr[N];//所有质数
bool st[N];

//线性筛
void shai()
{
    for(int i=2;i<=n;i++)
    {
        if(!p[i]) p[i]=i,pr[cnt++]=i;
        for(int j=0;pr[j]*i<=n;j++)
        {
            p[pr[j]*i]=pr[j];
            if(p[i]==pr[j]) break;
        }
    }
}

// long long 取模  O(1)
ll longlong_mod(ll a,ll b,ll m)
{
    a%=m,b%=m;
    ll d=(long double)a*b/m;
    d=a*b-d*m;
    if(d>=m) d-=m;
    if(d<0) d+=m;
    return d;
}
```

```cpp
// 快速乘方法 O(logn)
ll longlong_mul(ll a,ll b,ll p)
{
    a%=p,b%=p;
    ll res=0;
    while(b)
    {
        if(b&1) res=(res+a)%p;
        a=(a+a)%p;
    }
    return res;
}


// 区间筛：求 [l,r] 之间的素数个数   l,r <= 1e14    r-l <= 1e7
// 我们知道一个合数一定有不超过其根号的质因子
//因此我们可以预处理 1e7 以内的质数 然后利用埃氏筛的思想
// 将 [l,r] 筛一遍  时间复杂度 O(n*loglogn)
int interval()
{
    ll l,r;
    cin >> l >> r;
    if(l==1) l=2;
    for(int i=0;i<cnt;i++)
    {
        int t=pr[i];
        for(ll j=r/t*t;j>=l&&j>t;j-=t) st[j-l]=1;
    }
    int sum=0;
    for(int i=0;i<=r-l;i++) sum+=(!st[i]);
    return sum;
}
```

5.1.2 逆元

```
/*
逆元
(1) p 是质数
x = b^p-2 mod p

(2) p 不是质数
扩展欧几里得算法
b * x mod p = 1  ->  b * x + p * y = 1
求得一个特殊解后 变成 0 - p-1 范围内即可

(3) 求 1 - n 逆元   inv[i] = (p - p / i) * inv[p % i] % p;

(4) 求 1! - n! 逆元
先求出来 n! 逆元 然后从后往前递推  inv[i-1] = inv[i] * i % p;

(5) 求 a1 a2 ... an 的逆元
前缀乘积 si 先求出 sn 的逆元 然后 invs[i-1]= invs[i] * a[i] %
↪  p;
而 inv[i] = s[i-1] * invs[i] % p;  (s[0]=1)

(6) a / b mod p 如果 b p 不互质
   a / b mod p = a mod (p * b) / b

*/
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int N=3e6+10;

ll res,m1,a1,m2,a2;
int n,a[N],b[N];

int qmi(int a,int k,int p)
{
    int res=1;
    while(k)
    {
        if(k&1) res=(ll)res*a%p;
        a=(ll)a*a%p;
```

```
        k>>=1;
    }
    return res;
}

ll exgcd(ll a,ll b,ll &x,ll &y)
{
    if(!b)
    {
        x=1,y=0;
        return a;
    }
    ll d=exgcd(b,a%b,y,x);
    y=y-a/b*x;
    return d;
}

// (1)
int inv1(int b,int p)
{
    return qmi(b,p-2,p);
}

// (2)
int inv2(int b,int p)
{
    ll x,y;
    ll d=exgcd(b,p,x,y);
    if(d!=1) return -1;
    return (x%p+p)%p;     //x 通解  x = x0 + b/d * k 同余 b/d,
↪  同余 b
}


//求 1 - n 的逆元
int inv[N];
void inv3(int n,int p)
{
    inv[1]=1;
    for(int i=2;i<=n;i++)
        inv[i]=(ll)(p-p/i)*inv[p%i]%p;
}

//求 1! - n! 逆元
```

```
int infact[N];
void inv4(int n,int p)
{
    int res=1;
    for(int i=2;i<=n;i++) res=(ll)res*i%p;
    infact[n]=inv1(res,p);
    for(int i=n-1;i>0;i--)
        infact[i]=(ll)infact[i+1]*(i+1)%p;
}
```

5.1.3 康托展开

```
/*
康托展开: 将 1 - n 的任一排列 映射到 1 - n! 某个唯一的数上 (其实是字典序的排名)
逆康托展开: 逆映射 数 -> 排列

应用:(1) 将某一排列映射到唯一的数字上, 便于记录, 如八数码
     (2) 单纯求 某一排列 有多少个排列 字典序比他小
     (3) 求排在第 x 位的排列是什么 (逆康托展开)

康托展开: 数位 dp 的思考方式 先考虑第一位不同的
找 2 - n 中比 a1 小的数的个数 b1 那么 res+=b1*(n-1)!
现在将第一位固定为 a1 考虑第二位 .. 类推
到 b[] 数组 表示数组右边比这一位的元素小的元素个数  res = sum ( bi * (n-i)! )
(1) O(n^2) 朴素求法
(2) O(nlogn) 树状数组优化

逆康托展开: 类似进制转换 每次 bi = x / (n-i)!  x %= (n-i)!
            得到 b[] 数组 再推出 a[] 暴力 O(n^2)
                    平衡树优化 O(nlogn)

*/
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef unsigned long long ull;
typedef pair<int,int> pii;

const int mod=998244353,N=1e6+10;

int n,m,k;
int a[N],tr[N];
int fact[N];

void init()
{
    fact[0]=fact[1]=1;
    for(int i=2;i<=n;i++) fact[i]=(ll)fact[i-1]*i%mod;
}
```

```
// 暴力做法
ll cantor()
{
    ll ans=1;
    for(int i=1;i<n;i++)
    {
        int cnt=0;
        for(int j=i+1;j<=n;j++)
            if(a[j]<a[i]) cnt++;
        ans+=cnt*fact[n-i];
    }
    return ans;
}

int lowbit(int x)
{
    return x&-x;
}

void add(int x,int c)
{
    for(int i=x;i<=n;i+=lowbit(i)) tr[i]+=c;
}

int sum(int x)
{
    int res=0;
    for(int i=x;i;i-=lowbit(i)) res+=tr[i];
    return res;
}

//树状数组
int cantor2()
{
    ll ans=1;
    for(int i=n;i>0;i--)
    {
        int cnt=sum(a[i]);
        ans+=(ll)cnt*fact[n-i]%mod;
        add(a[i],1);
        ans%=mod;
    }

    return ans;
```

```
}

vector<int> decantor(ll x)
{
    x--;
    vector<int> b(n+1,0),a(n+1,0);
    for(int i=1;i<=n;i++)
    {
        b[i]=x/fact[n-i];
        x%=fact[n-i];
    }

    bool st[N]={0};
    for(int i=1;i<=n;i++)
    {
        int cnt=0;
        int j;
        for(j=1;j<=n;j++)
        {
            if(st[j]) continue;
            if(cnt==b[i]) break;
            cnt++;
        }
        a[i]=j,st[j]=1;
    }

    return a;
}

int main()
{
    cin >> n;
    init();
    return 0;
}
```

5.1.4 整除分块

```
/*
整除分块：对于 n/i (下取整  1 <= i <= n)
其取值只有 O(sqrt n) 个

1. 计算 sum (n/i)  i 属于 [1,n] 下取整
根据整除分块的性质 每次计算一整块 那么时间复杂度 O(sqrt n)
如何 O(1) 计算一整块：对于每一块的左端点 l 块内的值 d=n/l,
块内的右端点就是满足 i*d<=n 的最大的 i
故 r = n/d 下取整

2. 计算 sum(n/i) 上取整
  上取整 n/i = (n-1) / i + 1

3. 计算 sum(k mod i)  i[1,n]
  k mod i = k - k/i * i 每一块都是等差数列
  注意边界 if(!d) r=n; else r = min(n,k/d);

4. 计算 sum(f(i)*(n/i))
  一类题 可以预处理 f(i) 的前缀和 s[i] 然后整除分块
*/
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int mod=1e9+7,N=2e5+10;

int n,m,k;
int p[N];

// sum (n/i)
ll block()
{
    ll sum=0;
    for(int l=1;l<=n;l++)
    {
        ll d=n/l,r=n/d;
        sum+=(r-l+1)*d;
        l=r;
    }
    return sum;
```

```
}

int main()
{

    return 0;
}
```

5.1.5 dfs 求因子

```
/*
因子个数上界
n <= 1e8 d(n) <= 1e3
n <= 1e18 d(n) <= 1e5

n = a * b  a,b <= 1e9
分解质因数后 dfs 出 n 的因子
*/
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef pair<int,int> pii;

// dfs 枚举因子
vector<pii> prime;
vector<pair<ll,ll>> factor;
void dfs(int id,ll k1,ll k2)
{
    if(id==prime.size())
    {
        factor.push_back({k1,k2});
        return;
    }
    int p=prime[id].first,cnt=prime[id].second;
    ll pow=1;
    for(int i=1;i<=cnt;i++) pow*=p;
    ll res=1;
    for(int i=0; i<=cnt;i++)
    {
        dfs(id+1,k1*res,k2*(pow/res));
        res*=p;
    }
}
```

5.2 线性代数

5.2.1 矩阵乘法

```
/*
dp[n][m] 直接 dp 时间复杂度 O(nm)
矩阵乘法  O(m^3*logn)
*/
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int mod=1e9+7,N=102;

ll n,m;
ll a[N][N],f[N];

void fa()
{
    ll t[N]={0};
    for(int i=1;i<=m;i++)
        if(f[i])
            for(int j=1;j<=m;j++)
                if(a[i][j])
                    t[j]+=f[i]*a[i][j],t[j]%=mod;
    memcpy(f,t,sizeof t);
}

void aa()
{
    ll t[N][N]={0};
    for(int i=1;i<=m;i++)
        for(int k=1;k<=m;k++)
            if(a[i][k])
                for(int j=1;j<=m;j++)
                    if(a[k][j])
                        t[i][j]+=a[i][k]*a[k][j],t[i][j]%=mod;
    memcpy(a,t,sizeof t);
}

void qmi(ll x)
{
```

```
    while(x)
    {
        if(x&1) fa();
        aa();
        x>>=1;
    }
}

void solve()
{
    cin >> n >> m;
    for(int i=2;i<=m;i++) a[i][i-1]=1;
    a[1][m]=a[m][m]=1;
    f[m]=1;
    qmi(n);
    printf("%d\n",f[m]);
}

int main()
{
    solve();
    return 0;
}
```

5.2.2 gauss

```
// 高斯消元  O(n^3)
#include <bits/stdc++.h>

using namespace std;

const int mod=1e9+7, N=110;
const double eps=1e-10;

int n,m,k;
double a[N][N], b[N];

void gauss()
{
    int l=1;
    for(int i=1;i<=n;i++)
    {
        for(int j=l;j<=n;j++)
        {
            if(abs(a[j][i])>abs(a[l][i]))
            {
                for(int k=i;k<=n;k++) swap(a[j][k],a[l][k]);
                swap(b[j],b[l]);
            }
        }

        if(abs(a[l][i])<eps) continue;

        for(int j=1;j<=n;j++)
        {
            if(j!=l&&abs(a[j][i])>eps)
            {
                double d=a[j][i]/a[l][i];
                for(int k=i;k<=n;k++) a[j][k]-=d*a[l][k];
                b[j]-=d*b[l];
            }
        }
        l++;
    }
    for(int i=l;i<=n;i++)
    {
        if(abs(b[i])>eps)
```

```
        {
            //无解
            printf("-1\n");
            return;
        }
    }
    if(l<=n)
    {
        //无穷多解
        printf("-2\n");
        return;
    }
    for(int i=1;i<=n;i++) printf("%.10lf ",b[i]/a[i][i]);

}
```

5.3 博弈论

5.3.1 nim

```
/*
一.nim 游戏
n 堆石子，每一堆 xi 个，轮流取石子，每次从某堆中取石子，
至少取一个，无法取石子的玩家输

结论： x1 ^ x2 ^ ... ^ xn = 0 P 态
       x1 ^ x2 ^ ... ^ xn != 0 N 态
证明:
终态 0 ^ 0 ^ ... ^ 0 = 0 是 P 态
小范围内结论成立，下证任意范围都成立
对于任意 x1 ^ x2 ^ ... ^ xn = 0 无论如何取，x1 ^ x2 ^ ... ^ xi'
↪  ^ .. ^ xn != 0
反证，如果 = 0 ，那么有 xi = xi' ，但是 对于任意 != 0 的情况 ，
↪  一定 有取法使得 = 0 ，因此是 N 态
下面构造一种取法：设 x1 ^ ... ^ xn = b1b2...bn 其中 bi 为二进制
↪  位
假设二进制中最高位 1 是 bi ，那么在 x 中寻找一个 xj，使得 xj 的
↪  bi 位为 1，这样的 xj 一定存在
然后对于 bi+1 - bn ，如果 某一位是 0，那么 xj 的这一位不变，否则，
↪  xj 的这一位取反
这样，就得到了一个 xj'，并且满足 xj' < xj ，x1 ^ ... ^ xj' ^ ...
↪  ^ xn = 0 是合法构造


二．反常 nim 游戏
无法取石子的玩家赢

结论：如果 x1 = x2 = ... = xn = 1，
那么结论与 nim 相反 (异或和为 0，N；非 0，P)
否则，结论与 nim 相同


三．类 nim
每次操作可以选一堆石子 放或取 石子
结论仍然与 nim 一样，因为不管加或减，异或和 = 0 的，操作后一定 !=
↪  0


四．分裂 nim 游戏
n 堆石子，每次可以选某一堆取石子，或者将其分裂为两堆  正常规则
```

分析：拆成 n 个相同的组合游戏，只看一堆来分析，
↪ SG(0)=0,SG(1)=1,SG(2) = mex {0,1,1^1} = 2, ...
打表得到结论 SG(0) = 0, SG(4k+1) = 4k+1, SG(4k+2) = 4k+2,
↪ SG(4k+3) = 4k+4, SG(4k+4) = 4k+3

如果是反常规则
仍然按照正常规则 来求得所有点的 SG 值，然后和反常 nim 一样，特判
↪ 全 1，如果全 1，则结论相反

五.Fibonacci nim
初始有 n 个石子 先手第一次可取 1 - n-1 个石子，之后每一次可取的数
↪ 量最多为上一个人取的两倍 不能取的输

结论：n = Fibonacci 数 <=> n 是 P 态，n != Fibonacci 数 <=>
↪ n 是 N 态

Zeckendorf 定理：每个正整数都可以唯一的写成若干个不相邻的
↪ Fibonacci 数（不包括第一个 Fib 数,1,2,3,5,8）的和
如何得到这种表示：每次贪心地选取小于等于 n 的最大的 Fibonacci 数

必胜策略 ：1. 若 n != Fib 数，假设 n = Fx1 + Fx2 + ... + Fxk ,
↪ 其中 xi 递减，则取 Fxk，即最小项
n 变成 Fx1 + Fx2 + ... + Fxk-1 ，由于 Fxk-1 > 2Fxk
↪ ，因此后手不可能取完最后一项
故最终只会是先手将 n 取完，先手胜
2. 若 n = Fib = Fx，假设先手取 m 个，m < Fx
(1) 若 m >= Fx-2 ，那么有 Fx - m <= Fx - Fx-2 = Fx-1
↪ < 2 * Fx-2 ，因此后手直接取 Fx - m
(2) 若 m < Fx-2 && m >= Fx-4 ，取 Fx-2 - m，剩下
↪ Fx-1，此时下一轮先手并不能一次取完
(3) 若 m < Fx-4 && m >= Fx-6 ，取 Fx-4 - m，剩下
↪ Fx-1 + Fx-3，且下一轮先手并不能取完 Fx-3
...
实际操作时，n - m = Fx1 + Fx2 + ... + Fxk，后手取 Fxk
↪ 且一定可以取到（由上面的 (1)(2)(3) 可证）

七.k 倍动态减法
有 n 个石子 先手第一次可取 1 - n-1 个石子，之后每一次可取的数量最
↪ 多为上一个人取的 k 倍 不能取的输

1.k=1
结论：n = 2^i <=> n 是 P 态 n != 2^i <=> n 是 N 态

```
必胜策略 (证明)
 (1) n != 2^i , 先手取 n 的二进制表示的最后一位 1, 故后手并不能拿走倒数第二位 1, 因此并不能一次取完
     故最终一定是 n 只剩 1 个 1, 然后先手取走
 (2) n = 2^i , 由于先手不能取完, 因此先手取完后, 后手一定可以取到最后一位 1

2.k=2
Fibonacci nim

3.k>2
需要构造某个数列, 这个数列满足的性质是, 任何正整数都能被数列中若干项表示出来, 且表示的这几项满足
ax1 > k * ax2 , ax2 > k * ax3, ...
假设已经构造出了前 i-1 个数, 第 i 个数 ai, 前 i-1 项所能构造出来的最大数是 bi-1, 那么有 ai = bi-1 + 1
如何更新 bi : 找到 满足 ai > k * aj 最大的 j, 那么 bi = ai + bj
构造出以后, 如果 n = 数列中的某一项 , 那么是 P 态, 否则是 N 态;
最优策略仍然是 将 n 用数列若干项表示, 然后取最小的那一项
*/
// hdu 2486
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef pair<int,int> pii;

const int mod=1e9+7,N=1e6+10;

int n,m,k;
int a[N],b[N],tot;

void print()
{
    int res=0;
    for(int i=tot;i>=1;i--)
        if(n>=a[i]) n-=a[i],res=a[i];
    printf("%d\n",res);
}

void solve()
{
```

```
    a[1]=1,b[1]=1,tot=1;
    cin >> n >> k;
    for(int i=2,j=1;a[i-1]<n;i++)
    {
        a[i]=b[i-1]+1,tot++;
        while((ll)k*a[j+1]<a[i]) j++;
        if(a[j]*k<a[i]) b[i]=a[i]+b[j];
        else b[i]=a[i];
    }
    if(a[tot]==n) printf("lose\n");
    else print();
}

int main()
{
    int _;
    cin >> _;
    for(int i=1;i<=_;i++)
    {
        printf("Case %d: ",i);
        solve();
    }
    return 0;
}
```

5.3.2 其他模型

```
/*
一. 普通 Bash
n 个物品, 每次取 1 - m 个, 最后无法取的人输

归纳知, SG(x) = x % (m+1), 故若 n % (m+1) == 0, P 态, 否则是 N
↪  态
最优策略就是, 如果当前 x % (m+1) != 0 , 则取 x % (m+1) 个, 否则
↪  任取


二.chomp ! 游戏
n*m 矩阵, 每个格子放饼干, 每次可以选则一块饼干 (x,y), 并且将以这
↪  块饼干为左上角的矩形全部吃掉,
吃到 (1,1) 这块饼干的人输

结论: 如果 n = m = 1 ,P 态, 否则是 N 态
证明 (存在性证明) : 现在假设先手取 (n,m), 如果后手必输, 那么得证,
↪  否则如果后手有必胜策略 (a,b),
                    那么先手就可以取 (a,b) 而不是 (n,m), 因此一定
↪  是先手必胜
由于是存在性证明, 因此并不能给出必胜策略, 目前还没有构造出具体的必
↪  胜策略

扩展: 可以扩展到任意一个存在最大元的有限偏序集 S, 每次选择 x 并且
↪  移走 x 和所有比 x 大的元素, 取走最后一个元素的人输
结论: 如果 S 中元素个数 > 1 那么先手必胜
当然也可以是有最小元, 每次移走比 x 小的元素, 那么先手必胜
具体例子
(1) 三维 chomp 游戏
(2) 由 n 的因子构成的集合, x<=y 当且仅当 x|y
    n 是最大元, 故 n>1, N 态,  n=1, P 态
(3)1 - n 个数, 每次选择一个集合中的数 x, 并删除集合中所有 x 的约
↪  数, 取走最后一个元素的人输
    1 是最小元, 故 n>1 先手必胜
(4) 给定有根树, 初始节点为白色, 每次可以选择一个白色节点并将此节点
↪  到根的路径上所有的节点全部染黑, 无法染色的人赢
    root 相当于最大元, 因此如果节点个数 > 1, 先手必胜


三.Wythoff 游戏
两堆石子 n,m, 每次可以从一堆中取出若干个, 或者从两堆石子同时取走若
↪  干个
```

(等价的游戏：在棋盘上移动，每次可以向左/上移动若干步，或者向左上
↪ 45 度移动若干步)

打表观察得到结论：假设 n <= m ，得到 P 态的一堆数对 (a,b),
规律是 ai = [i*C] , bi = [i*C*C] , 其中 C = (1 + 根号 5) / 2,
↪ [] 代表下取整
且 ai = mex(aj,bj) j<i, bi = ai + i

Beatty 序列：由一个正无理数 r 产生的序列 Br = [r*1], [r*2],
↪ [r*3], ...
Rayleigh 定理：如果正无理数 r,s 满足 1/r + 1/s = 1，那么 Br 与
↪ Bs 是全体正整数的一个分割
　　即任意一个正整数存在且仅存在于 Br 或 Bs 中
如何判断： n,m, n <= m, i = m-n, if(n==i*C) P 态, else N 态
最优策略：同取 if(n>i*C) n,m -> i*C,i*C+i
　　取 n i = (m-1)/(C*C) + 1 , t = i * C ,if(n>t) n -> t
　　取 m i = (n-1)/C + 1 , t = i * C*C ,if(m>t) n -> t

四．树上删边
给定一棵树，每次删一条边，删除之后如果某个支不与地面/根相连，那么
↪ 也一并删去

1. 如果树是一条链，那么 = nim 游戏 ， sg(x) = x
2.Colon 原则：假设 G1,G2 是两个树上删边游戏，如果 sg(G1) =
↪ sg(G2) ，那么将 G1 的根新加一条边连到更低的节点作为新的根
↪ (地面)
变成一个新的树上删边游戏 H1，同理 G2 变成 H2，那么有结论 sg(H1)
↪ = sg(H2)
证明：已知 sg(G1) = sg(G2)，证明 sg(H1) = sg(H2)，等价于证明 H1
↪ + H2 是 P 态
　　假设先手删 H1 新加的那条边，那么后山删 H2 新加的边 ，后手胜
　　否则，先手删 G1 内部的边，那么有两种情况
　　(1) sg(G1') < sg(G1) ，那么后手一定可以操作使得 sg(G2') =
↪ sg(G1') ，因为 sg(G1) = sg(G2),
　　(2) sg(G1') > sg(G1) ，那么后手也操作 G1 ，使得 sg(G1')
↪ 变回 sg(G1)，由 sg 的定义知这是一定可以的
　　故后手能一直保持 sg(G1) = sg(G2)，直到 sg(G1) = sg(G2) =
↪ 0 ，这时先手只能选择新加的边，因此是 P 态
　　故 sg(H1) ^ sg(H2) = 0 ，sg(H1) = sg(H2)
3. 结论 由 1, 2, 假设 G2 是链，那么 sg(H2) = sg(G2) + 1，故
↪ sg(H1) = sg(G1) + 1
　　故得出结论 sg(u) = xorsum (sg(v) + 1) ，v 是 u 所有的儿
↪ 子

```
*/

// Wythoff
#include <bits/stdc++.h>

using namespace std;

int main()
{
        int a, b, cha, c;
        double k=(sqrt(5.0)+1)/2;
        while(scanf("%d%d", &a, &b) && (a+b)!=0)
        {
                if(a==b) //如果相同，可以一次取完
                {
                        printf("1\n0 0\n");
                        continue;
                }
                if(a>b) swap(a, b);//a 代表小的，代表大的
                cha=b-a;//求差
                c=int(cha*k);//计算如果 b 固定的话，a 应该是多少
                if(c==a)//如果相等，那么先手必输
                        printf("0\n");
                else
                {
                        printf("1\n");
                        int bb=c+cha;//计算如果 c 当做小的，大
↪  的应该是多少
                        if(b-bb==a-c)//判断这个可不可以两堆同时
↪  拿
                                printf("%d %d\n", c, bb);
                        for(int i=a-1, j=1; i>=1; i--,
↪  j++)//固定 a，求另一个数
                        {
                                if(int(j*k)==i)
                                {
                                        printf("%d %d\n",
↪  int(j*k), a);
                                        break;
                                }
                        }
                        for(int i=b-1, j=1; i>=1; i--,
↪  j++)//固定 b，求另一个数
                        {
```

```
                                if(int(j*k)==i)
                                {
                                        printf("%d %d\n",
↪  int(j*k), b);
                                }
                        }
                }
        }
        return 0;
}
```

6 杂项

6.1 根号分治

```
/*
1. 给一个 n 点 m 边的图, 每个点初始为白色 q 个询问 有两种类型 n <= 2e5 m,q <= 4e5
1. 将一个点变色 即白变黑 黑变白
2. 查询一个点邻接有多少个黑点

n 个点 按照度的不同 分成两类 deg > sqrt(n) 的称为大点 个数不超过 sqrt(n) 个
deg < sqrt(n) 的称为小点
考虑查询 : 如果是 小点 那么可以直接暴力统计
如果是大点 那么不能暴力统计了 可以修改时维护大点的答案 O(sqrt(n))
考虑修改 : 要维护大点的答案 所以将这个点所有邻接的大点的答案都修改一下 O(sqrt(n))
当然每个点周围的大点需要先预处理 O(m)


2. 给定无向图 n 个点 m 条边 求三元环的个数
根号分治
按 点的度 重新建边 任意一条边 点度小的指向点度大的 点度相同的, 编号小的指向编号大的 变成新图
不难知道新图是有向无环的
点度小的指向点度大的 保证了 新图 任意一个点的出度 <= sqrt(m)

而原图中的任意三元环 i-j-k i<j<k 在新图中一定会以 i->j j->k i->k 出现
因此第一层循环点 i 第二层循环出边 对应的点 j 第三层循环点 j 的出边 对应点 k 判断 i 是否指向 k (预处理)
*/
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int mod=1e9+7,N=1e5+10,M=2*N;

int n,m,k;
int h[N],e[M],ne[M],d[N],in[N],idx;
pair<int,int> edge[M];
```

```
void add(int a,int b)
{
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}

void solve()
{
    cin >> n >> m;;
    for(int i=0;i<m;i++)
    {
        int u,v;
        scanf("%d%d",&u,&v);
        d[u]++,d[v]++;
        edge[i]={u,v};
    }

    for(int i=1;i<=n;i++) h[i]=-1;
    for(int i=0;i<m;i++)
    {
        int u=edge[i].first,v=edge[i].second;
        if(d[u]>d[v]) swap(u,v);
        else if(d[u]==d[v])
            if(u>v) swap(u,v);
        add(u,v);
    }

    ll res=0;
    for(int i=1;i<=n;i++)
    {
        for(int j=h[i];~j;j=ne[j]) in[e[j]]=i;
        for(int j=h[i];~j;j=ne[j])
        {
            int u=e[j];
            for(int k=h[u];~k;k=ne[k])
                if(in[e[k]]==i) res++;
        }
    }

    cout << res;
}

int main()
{
```

```
    solve();
    return 0;
}
```