HRBUST

ACM 模板

哈尔滨理工大学 段学松

JimoHack

2019-10-17

目录

1. 动态规划

(1) zkw 线段树优化 dp

```
#include <bits/stdc++.h>
#include <unordered_map>
#define in32(x) (scanf("%d",&x))
#define in64(x) (scanf("%lld",&x))
#define out32(x) (printf("%d",x))
#define out64(x) (printf("%lld",x))
#define en putchar('\n')
#define sp putchar(' ')
#define O2 __attribute__((optimize("-O2")))
const int MAXN (100005);
const int inf (0x3f3f3f3f) ;
const int64_t INF (0x3f3f3f3f3f3f3f3f) ;
using namespace std ;
int m,n,k,buff,d;
int64_t a[MAXN],b[MAXN],arr[MAXN] ;
int64_t dp[MAXN],pre[MAXN] ;
unordered_map<int64_t,int64_t> id ;
class ZKW_Tree
{
public:
    pair<int64_t,int64_t> Tmax[(MAXN<<2)+10] ; ///first 存 dp 值,second 存下标
    int deep;
    void build(int n)
    {
        deep = 1 ;
        while(deep<n) deep<<=1 ;
    }

    inline void push_up(int pos)
    {
        Tmax[pos] = max(Tmax[pos<<1],Tmax[pos<<1|1]);
    }

    inline void change(int64_t p,int64_t val,int64_t id)
    {
        Tmax[deep+p] = make_pair(val,id) ;
        for(int i(deep+p>>1);i;i>>=1) push_up(i) ;
    }
```

```
    pair<int64_t,int64_t> QMax(int l,int r)
    {
        pair<int64_t,int64_t> ans(0,0);
        l += deep , --l ;
        r += deep , ++r ;
        while(l^r^1) {
            if(~l&1) ans = max(ans,Tmax[l^1]);
            if(r&1) ans = max(ans,Tmax[r^1]);
            l>>=1,r>>=1 ;
        }
        return ans ;
    }
};
ZKW_Tree tree ;

int fen(int l,int r,int64_t val)
{
    while(l<r) {
        int mid(l+r+1>>1) ;
        if(arr[mid]<=val) l = mid ;
        else r = mid - 1 ;
    }
    return l ;
}

int main()
{
    memset(dp,0,sizeof(dp)) ;
    memset(pre,-1,sizeof(pre));
    in32(n),in32(d) ;
    int64_t maxs(-INF),mins(INF) ;
    tree.build(MAXN) ;
    for(int i(1);i<=n;++i) {
        in64(arr[i]) ;b[i] = a[i] = arr[i] ;
        maxs = max((int64_t)maxs,b[i]) ;
        mins = min((int64_t)mins,b[i]) ;
    }
    sort(&arr[1],&arr[n+1]) ;
    int64_t len(unique(&arr[1],&arr[n+1])-&arr[0]) ;
    for(int i(1);i<=n;++i) {
        a[i] = lower_bound(&arr[1],&arr[len],a[i])-&arr[0] ;
        id[b[i]] = a[i];
    }
```

```
    dp[1] = 1 ;
    tree.change(id[b[1]],1,1) ;
    int64_t ans(1),ansp(1) ;
    for(int i(2);i<=n;++i) {
        int64_t pHigh(lower_bound(&arr[1],&arr[n+1],b[i]+d)-&arr[0]) ;
        int64_t pLow(fen(1,n,b[i]-d)) ;
        dp[i] = 1 ;
        pair<int64_t,int64_t> node ;
        if(b[i]-d>=mins) {
            node = tree.QMax(1,id[arr[pLow]]) ;
            if(node.first + 1 > dp[i]) {
                dp[i] = node.first + 1;
                pre[i] = node.second;
            }
        }
        if(b[i]+d<=maxs) {
            node = tree.QMax(id[arr[pHigh]],MAXN) ;
            if(node.first + 1 > dp[i]) {
                dp[i] = node.first + 1;
                pre[i] = node.second;
            }
        }
        tree.change(id[b[i]],dp[i],i);
        if(dp[i] > ans) {
            ans = dp[i];
            ansp = i ;
        }
    }
    out64(ans) ;en ;
    int64_t p(ansp) ;
    stack<int64_t> sta ;
    while(p>0) {
        sta.push(p) ;
        p = pre[p] ;
    }
    while(sta.size()) {
        out64(sta.top()) ;
        sp ;
        sta.pop() ;
    }
    en ;
}
```

(2) 阶段型线性 DP

```
/**
    阶段型线性 DP 一般是以阶段作为基础,不断扩增阶段,得到答案的.
典型的例子:HDU - 1024
 要求从一序列中取出若干段,这些段之间不能交叉,使得和最大并输出.
 用 dp[i][j]表示前 j 个数取出 i 段得到的最大值，那么状态转移方程为:
    dp[i][j]=max(dp[i][j-1]+a[j],dp[i-1][k]+a[j])   i-1<=k<=j-1
以 i 段作为阶段进行 DP 递推,用 i-1 段更新 i 段.
**/
```

(3) 最长公共子序列 nlogn 解法

```
/**
设序列 A 长度为 n，{A(i)}，序列 B 长度为 m，{B(i)}，考虑 A 中所有元素在 B 中的序号，即 A 某元素在 B 的序号为{Pk1,Pk2,..}，将这些序号按照降序排列，然后按照 A 中的顺序得到一个新序列，此新序列的最长严格递增子序列即对应为 A、B 的最长公共子序列。
举例来说，A={a,b,c,d,b}，B={b,c,a,b}，则 a 对应在 B 的序号为 2，b 对应序号为{3,0}，c 对应序号为 1，d 对应为空集,生成的新序列为{2,3, 0, 1, 3, 0},其最长严格递增子序列为{0,1,3},对应的公共子序列为{b, c, b}
时间复杂度:O(nlogn)~O((n^2)logn)
最坏例子: {aaaaaaaaaaaaaa} {aaaaaaaaaa}
*/
#include <bits/stdc++.h>
#include <unordered_map>
using namespace std ;
const int MAXN(100005) ;
unordered_map<int,vector<int> > pos ;
int Tmax[MAXN<<2],deep ;
void build(int n)
{
    deep = 1 ;
    while(deep<n) deep<<=1 ;
}
inline void push_up(int pos)
{
    Tmax[pos] = max(Tmax[pos<<1],Tmax[pos<<1|1]);
}
inline void change(int p,int val)
{
    Tmax[deep+p] = val ;
    for(int i(deep+p>>1);i;i>>=1) push_up(i) ;
}
int QMax(int l,int r)
{
    int ans(0) ;
```

```
        l += deep , --l ;
        r += deep , ++r ;
        while(l^r^1) {
            if(~l&1) ans = max(ans,Tmax[l^1]);
            if(r&1) ans = max(ans,Tmax[r^1]);
            l>>=1,r>>=1 ;
        }
        return ans ;
}
int arr1[MAXN],arr2[MAXN] ;
int len1 ,len2 ;
int LIS()
{
        build(MAXN) ;
        int maxs(0) ;
        bool isok(false) ;
        for(int i(0);i<len1;++i) {
            for(int j(0);j<pos[arr1[i]].size();++j) {
                if(!isok) {
                    change(pos[arr1[i]][j],1) ;
                    isok = true ;
                    continue ;
                }
                int dp(1) ;
                dp = max(dp,QMax(0,pos[arr1[i]][j]-1)+1) ;
                change(pos[arr1[i]][j],dp) ;
                maxs = max(maxs,dp) ;
            }
        }
        return maxs ;
}
int main()
{
        while(~scanf("%d",&len1)) {
            len2 = len1 ;
            pos.clear() ;
            for(int i(0);i<len1;++i) {scanf("%d",&arr1[i]);}
            for(int i(0);i<len2;++i) {scanf("%d",&arr2[i]);}
            for(int i(0);i<len2;++i) {
                pos[arr2[i]].push_back(i+1) ;
            }
            unordered_map<int,vector<int> >::iterator it ;
            for(it = pos.begin();it!=pos.end();++it) {
                vector<int>& v(it->second) ;
```

```
                sort(v.begin(),v.end(),greater<int>()) ;
            }
            printf("%d\n",LIS()) ;
        }
}
/**
5
3 2 1 4 5
1 2 3 4 5
out: 3
*/
```

(4) 单调栈预处理 O(n)

```
///vector 版本
//单调栈得到区间以某一个数为中心最大向左和向右可拓展的范围[L,R]
int L[MAXN],R[MAXN] ;
vector<pair<int64_t,int> > Q;

void QueInit(int64_t* arr,int n)
{
    Q.clear() ;
    for(int i(1);i<=n+1;++i) {
        L[i] = R[i] = i ;
        int64_t x(i == n+1?0x3f3f3f3f:arr[i]);
        ///想要维护最小值的范围的话,要把 inf 变成-inf,下边写成小于等于
        while(Q.size()&&x>=Q.back().first) {
            int64_t val(Q.back().first);
            int id(Q.back().second);
            if(x == val) {L[i] = L[id] ;break ;}
            Q.pop_back(); L[i] = min(L[i],L[id]) ;R[id] = i-1 ;
        }
        Q.push_back({x,i}) ;
    }
}

///数组模拟版本
int L[MAXN],R[MAXN],tot ;
pair<int64_t,int> Q[MAXN];
void QueInit(int64_t* arr,int n)
{
    tot = 0 ;
    for(int i(1);i<=n+1;++i) {
        L[i] = R[i] = i ;
        int64_t x(i == n+1?0x3f3f3f3f3f3f3f3f:arr[i]);
```

```
        ///想要维护最小值的范围的话,要把 inf 变成-inf,下边写成小于等于
        while(tot&&x>=Q[tot-1].first) {
            int64_t val(Q[tot-1].first);int id(Q[tot-1].second);
            if(x == val) {L[i] = L[id] ;break ;}
            --tot; L[i] = min(L[i],L[id]) ;R[id] = i - 1 ;
        }
        Q[tot++] = make_pair(x,i) ;
    }
}
```

(5) 子序列计数 O(nm)

```
/**
题意:
    给出两个只包含大写字母的字符串 s,t,求 s 的子序列中是 t 的个数.
    要求: 输入 26 个整数 cost[i],每个整数表示字母表字母空 cost[i]
    个字符以后才能连接其他字符.
输入:
2 10
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AB
ABBBBABBBB
输出:
10
思路:
    观察子串 AB,我们发现 A 后边必须连着 B,所以说我们可以以
子串匹配的位置作为阶段(dp 数组的第一维),用 dp[i][j]表示子
串 t 匹配到 i 位置,母串 s 匹配到 j 位置的方案数.然后 dp 从后向前跑.
用 sum[i][j]表示 dp[i][j~n]的后缀和,则 dp 转移方程为:
        dp[i][j] = Σ{t[i] == s[j]}(sum[i+1][k+cost[t[i]]+1])
        sum[i] = Σ{j=i,j<=n}(dp[i][j])
    由于 sum[i]和 dp[i]只与 dp[i+1]有关,所以可以滚动数组优化.
*/

#include <bits/stdc++.h>
using namespace std ;
char s[100005],t[100005];
int dp[100005],cost[30] ;
int sum[100005] ;
const int Mod = 1000000007 ;
int main()
{
    int n,m ;
    scanf("%d%d",&m,&n) ;
```

```
        for(int i(1);i<=26;++i) scanf("%d",cost+i) ;
        scanf("%s%s",t+1,s+1) ;
        for(int i(1);i<=n;++i) s[i] -= 'A' - 1 ;
        for(int i(1);i<=m;++i) t[i] -= 'A' - 1 ;
        memset(dp,0,sizeof(dp)) ;
        sum[m+1] = 0 ;
        for(int i(n);i>=1;--i) {
            dp[i] = t[m] == s[i] ;
            sum[i] = (sum[i+1] + dp[i])%Mod ;
        }
        for(int i(m-1);i>=1;--i) {
            fill_n(dp,n+1,0) ;
            for(int j(n);j>=1;--j) {
                int p(j + cost[t[i]] + 1) ;
                if(s[j] == t[i] && p <= n) (dp[j] += sum[p]) %= Mod ;
            }
            for(int j(n);j>=1;--j) sum[j] = (dp[j] + sum[j+1])%Mod ;
        }
        printf("%d\n",sum[1]) ;
    }
```

2. 二分三分

(1) 01 分数规划 O(nlog^2n)

```
#include<bits/stdc++.h>
const int MAXN (100005) ;
const double eps (1e-8) ;
using namespace std ;
/**
01 分数规划:
    求Σ(val[i]*x[i])/Σ(cost[i]*x[i])的最大(小)值问题,其中 x[i]∈{0,1}.
解决问题思想: 二分
时间复杂度: O(nlog^2n)
设答案 ans = Σ(val[i]*x[i])/Σ(cost[i]*x[i]);
设 f(mid) = Σ(val[i]*x[i])-mid*Σ(cost[i]*x[i]);
    ① 如果 f(mid) == 0,说明 mid 就是答案,return mid;
    ② 如果 f(mid) > 0,说明 mid 小了,L = mid ;
    ③ 如果 f(mid) < 0,说明 mid 大了,R = mid ;
关于 check 函数:
    把每个物品的 val[i]-mid*cost[i]排序,选出题目要求的个数加和,
    当成 f(mid)判断即可.
*/
```

```
struct Node
{
    int val/**价值*/,cost/**花费*/;
    double weight;///排序用到的权值,即 val[i]*x[i]-mid*cost[i]*x[i].
    bool operator<(Node& b)
    {
        return weight>b.weight ;/**求最大值的话,就从大到小排序.*/
    }
};
Node node[MAXN+5] ;
double solve(Node* node,int n,int k)
{
    double l(0),r(10000000000000.0),mid ;
    auto check = [&]() -> bool {
        for(int i(1);i<=n;++i)
            node[i].weight = node[i].val-node[i].cost*mid ;
        sort(node+1,node+1+n) ;
        double sum(0) ;
        for(int i(1);i<=k;++i)
            sum += node[i].weight;
        return sum >= 0 ;
    };
    while(r-l>eps) {
        mid = (l + r) / 2 ;
        if(check()) l = mid ;
        else r = mid ;
    }
    return mid ;
}
int main()
{
    int n,k ;
    while(~scanf("%d%d",&n,&k))
    {
        if(!n&&!k) break ;
        for(int i(1);i<=n;++i)
            scanf("%d",&node[i].val) ;
        for(int i(1);i<=n;++i)
            scanf("%d",&node[i].cost) ;
        printf("%.0f\n",100*solve(node,n,n-k)) ;
    }
    return 0 ;
}
/**
```

```
in:
3 1
5 0 2
5 1 6
4 2
1 2 7 9
5 6 7 9
0 0
out: 83 100
*/
```

(2) 二分查找相关

在 a 数列(从 1 开始)中查找第一个>=val(最左边)的元素

```
int pos = lower_bound(&a[1],&a[n+1],val) - &a[0] ;
int fen(int l,int r,int val)
{
	while(l<r) {
		int mid(l+r>>1) ;
		if(arr[mid]>=val) r = mid;
		else l = mid+1;
	}
	return l ;
}
```

在 a 数列(从 1 开始)中查找最后一个<=val(最右边)的元素

```
int pos = upper_bound(&a[1],&a[n+1],val) - &a[0] - 1;

int fen(int l,int r,int val)
{
	while(l<r) {
		int mid(l+r+1 >>1) ;
		if(arr[mid]>val) r = mid - 1;
		else l = mid;
	}
	return l ;
}
```

(3) 二分可行答案的最大(小)值

```
/*
```

有的时候,答案本身不一定是单调的,这时候我们可以尝试二分答案的范围.

题目大意：

给你 n 个数和 m，每个数的范围都是在 0～（m- 1）。然后每一次操作 你可以选择这 n 个数中的任意多个，可以将你选择的数(a[i]+1)%m。然后问你最小操作次数使得整个数列变成非递减的。

具体思路：
次数本身不是单调的，但是次数的范围是单调的，我们可以二分次数的范围，即 mid 代表操作<=mid 的方案是否可行,然后二分即可.
*/

```
#include <iostream>
using namespace std ;
int n,m ;
int a[1000005];
bool check(int mid)
{
    int last = (a[1]+mid>=m?0:a[1]) ;
    for(int i = 2;i<=n; ++i) {
        if(a[i]<last) {
            if(a[i]+mid<last) return false ;
        } else if(a[i] == last) {
            continue ;
        }else {
            if(a[i]+mid>=m&&(a[i]+mid)%m>=last) continue ;
            last = a[i];
        }
    }
    return true ;
}
int main()
{
    scanf("%d%d",&n,&m);
    for(int i = 1;i<=n;++i) {
        scanf("%d",a+i);
    }
    int l(0),r(100000005);
    while(l<r){
        int mid = (l+r)>>1 ;
        if(check(mid)) r = mid ;
        else l=mid+1;
    }
    printf("%d\n",l) ;
}
```

(4) 三分求凸(凹)函数极值

```
while(l<r){ /**整数域*/
        int64_t mid1(l + (r - l)/3),mid2(r - (r - l)/3) ;
        int64_t f1(check(mid1)),f2(check(mid2)) ;
        if(f1 == f2) {l = mid1+1;r = mid2-1;}///这个判断加上会更快一些
        else (f1>f2)?(l = mid1+1):(r = mid2-1);
```

```
}
return l ;


const double eps = 1e-8 ;
bool is0(double x) {return abs(x)<eps;}
double fen(double l,double r) /**实数域*/
{
   while (!is0(r-l)){
      double d=(r-l)/3.0;
      double mid1(l+d),mid2(r-d);
      double f1(check(mid1)),f2(check(mid2)) ;
      if(is0(f1-f2)) {l=mid1;r=mid2;}
      else f1<f2?l=mid1:r=mid2;
   }
   return (l+r)/2.0 ;
}

bool get1(1),get2(1);
while (is0(r-l)) { ///优选法,其实也只是三分的优化
      double mid1(l+(r-l)*(1-GR)),mid2(l+(r-l)*GR);///根据黄金比例分割中间两个点
      if (getl) f1=f(mid2);///精妙之处
      if (getr) f2=f(mid2);
      if (is0(mid1-mid2)) {r=midr,l=midl,getl=1,getr=1;} ///l≈r 时已经得出答案,所以肯定是在异侧,可以同
时向上
      else if (f1>f2) { ///求最小值的话,这里改成<
            r=mid2;f2=f1;get1=1;get2=0;
      } else {
            l=mid1;f1=f2;get1=0;get2=1;
      }
}
return l ;
```

(5) 同一时间做很多操作的二分

```
/*
题意: 有 n 个地点，每个地点有 a[i]个箱子，然后有 m 个人，
      每个人从第 i 个位置移动到第 i+1 的位置花费 1S，每个人
      移动一个箱子花费 1S，要求把所有箱子全部移除花费的最小时间是多少

分析：经过观察我们发现，在同一个时刻有 m 个人在动，直接模拟是不行的。
      所以，我们要把每个人单独考虑，就相当于每个人都有相同的一段时间，
      然后看这些人能不能把箱子都搬完，这样我们就可以二分答案了。
*/
#include <bits/stdc++.h>
```

```
using namespace std ;
int n,m ;
int64_t a[100005],buff[100005] ;
bool check(int64_t mid)
{
    int pos(1) ;
    memcpy(buff,a,sizeof(a)) ;
    for(int i(1);i<=m;++i) {
        int64_t time = mid-pos ;
        if(time < 0) break ;
        while(1) {
            if(buff[pos]>0) {
                if(time == 0) break ;
                if(time>=buff[pos]) {
                    time -= buff[pos],buff[pos] = 0 ;
                    if(pos+1<=n) ++ pos ， -- time ;
                    else break ;
                } else {
                    buff[pos] -= time ;break ;
                }
            } else {
                if(pos+1<=n) ++ pos,-- time ;
                else break ;
            }
        }
    }
    return pos >= n&&buff[pos]==0;
}
int main()
{
    cin >>n>>m ;
    for(int i(1);i<=n;++i) cin >> a[i] ;
    int64_t l(0),r(10000000000000000000LL) ;
    while(l<r) {
        int64_t mid = ((l+r)>>1) ;
        if(check(mid)) r = mid ;
        else l = mid + 1 ;
    }
    cout<<l<<endl;
}
```

(6) 桶排序+二分位置

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

```
vector<int> V[105];
int a[100005];
int main(){
    int n,c;
    scanf("%d%d",&n,&c);
    for(int i = 0;i < n; ++i){
        scanf("%d",&a[i]);
        V[a[i]].push_back(i + 1);
    }
    int ans = 0;
    for(int i = 1;i <= c; ++i){
        if(V[i].empty()) continue ;
        for(int j = 1;j <= c; ++j){
            if(V[j].empty()||i == j) continue ;
            int p = V[i][0];
            int len(1),id(j) ;
            while(1) {
                auto it = lower_bound(V[id].begin(),V[id].end(),p) ;
                if(it == V[id].end()) {if(len == 1) break ;ans = max(ans,len);break ;}
                ++ len ;    id = i+j-id ; p = *it ;
            }
        }
    }
    cout << ans << endl;
    return 0;
}
```

(7) 第 K 大 1 矩阵问题[二分+ST 表]O(knmlog(m))

```
/**
                第 K 大全 1 子矩阵问题
        题意: 给出一个 n*m 的 01 矩阵,让你求全是 1 的面积第 k 大的子矩阵
        时间复杂度:O(k*n*m*log(m)) n<=1000&&m<=1000 没问题(380ms).
        https://ac.nowcoder.com/acm/contest/882/H
*/
#include <bits/stdc++.h>
#include <hash_set>
using namespace std ;
using namespace __gnu_cxx ;
const int MAXN (1005);
int n,m ;
class ST
{
public:
    int mm[MAXN] ;
```

```
    int dp[20][MAXN];
    void Init(int* arr,int n)
    {///下标从 1 开始,传参数的时候 arr 需要注意一下!!!
        for(int i(1);i<=n;++i) {
            mm[i] = __lg(i) ;dp[0][i] = arr[i] ;
        }
        for(int i(1);i<=mm[n];++i) {
            for(int j(1);j+(1<<i)-1<=n;++j) {
                dp[i][j] = min(dp[i-1][j],dp[i-1][j+(1<<(i-1))]) ;
            }
        }
    }
    inline int Q(int l,int r)
    {
        int k(mm[r-l+1]) ;
        return min(dp[k][l],dp[k][r-(1<<k)+1]) ;
    }
};
ST st ;
int cnt[MAXN][MAXN] ;///从(i,j)最多能向下拓展的长度.
char bg[MAXN][MAXN] ;
int fenL(int pos,int val)
{///找最左边第一个大于等于 val 的位置.
    int l(1),r(pos) ;
    while(l<r) {
        int mid((l+r)>>1) ;
        st.Q(mid,pos)<val?l=mid+1:r=mid ;
    }
    return l ;
}

int fenR(int pos,int val)
{///找最右边大于等于 val 的位置.
    int l(pos),r(m) ;
    while(l<r) {
        int mid((l+r+1)>>1) ;
        st.Q(pos,mid)<val?r=mid-1:l=mid ;
    }
    return l ;
}

hash_set<int> s ;
priority_queue<int,vector<int>,greater<int> > Q ;
```

```
int main()
{
    scanf("%d%d",&n,&m) ;
    for(int i(1);i<=n;++i) scanf("%s",&bg[i][1]) ;
    for(int j(1);j<=m;++j) cnt[n][j] = (bg[n][j] == '1') ;
    for(int i(n-1);i>=1;--i) ///预处理 cnt[i][j].
        for(int j(1);j<=m;++j)
            cnt[i][j] = (bg[i][j]=='1')?cnt[i][j]=cnt[i+1][j]+1:0 ;
    for(int i(1);i<=n;++i) {///枚举每一行
        st.Init(cnt[i],m) ; ///建立 ST 表
        s.clear() ;///清空 hash_set
        for(int j(1);j<=m;++j) {
            if(cnt[i][j] == 0) continue ;
            int pL(fenL(j,cnt[i][j])),pR(fenR(j,cnt[i][j])) ;
            ///二分出以 cnt[i][j]为高的最大可向左(右)拓展的范围
            if(s.count((pL-1)*1000+(pR-1))) continue ;///hash_set 去重
            s.insert((pL-1)*1000+(pR-1)) ;
            for(int k(1);k<=pR-pL+1;++k) {///因为要取第二大,所以枚举最大矩形的所有横向小矩形.
                /*取第一大(最大)不用枚举,直接取最大值就行了.*/
                Q.push(k*cnt[i][j]) ;
                if(Q.size()>2) Q.pop() ; ///取第二大
            }
        }
    }
    printf("%d\n",Q.size()<2?0:Q.top()) ;
}
```

3. 分治算法

(1) 点分治[容斥写法]O(nlogn)

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
using namespace std ;
using namespace __gnu_pbds ;
const int MAXN = 200005 ;
bool isok[MAXN] ;
int Size[MAXN],N,maxSon/**子树最大值*/,cPos/**重心节点*/ ; ;
vector<pair<int,int> >Next[MAXN] ;
void Init(int n) { ///初始化.注意!!!!这里的 n 必须严格为点的个数!!!!
    for(int i(0);i<=n;++i) {isok[i] = false ;Next[i].clear() ;N = n;}
}
void addedge(int u,int v,int val = 0) {///加一条双向边
```

```
    Next[u].push_back({v,val}) ;Next[v].push_back({u,val}) ;
}
void dfs(int pos,int fa=-1) {///dfs 求重心
    Size[pos] = 1 ;
    int maxs(0),sz(Next[pos].size()) ;
    for(int i(0);i<sz;++i) {
        int to(Next[pos][i].first);
        if(to == fa||isok[to]) continue ;
        dfs(to,pos) ; Size[pos] += Size[to] ;
        maxs = max(maxs,Size[to]) ;
    }
    maxs = max(maxs,N-Size[pos]) ;
    if(maxs<maxSon) {maxSon=maxs;cPos=pos;}
}
inline int getCore(int pos) {///找到树上以 pos 为根的重心
    maxSon = 0x3f3f3f3f ;dfs(pos) ;return cPos ;
}
void getDis(int pos,gp_hash_table<int,int>& mp,int sum = 0,int fa = -1)
{///统计以 pos 为根子树里的点到 pos 的距离 sum 的数量并且存到 mp 中.
    ++ mp[sum] ;
    int sz(Next[pos].size()) ;
    for(int i(0);i<sz;++i) {
        int to(Next[pos][i].first),val(Next[pos][i].second) ;
        if(isok[to]||to == fa) continue ;
        getDis(to,mp,sum + val,pos) ;
    }
}
vector<gp_hash_table<int,int> > V ;///每一个儿子的桶
vector<int> Q ; ///离线的询问
gp_hash_table<int,int> All ; ///当前的全局桶
int64_t Ans[10000005] ;
void calc(int pos) { ///计算经过 pos 的答案.
    V.clear() ;All.clear() ;
    int sz(Next[pos].size()) ;
    for(int i(0);i<sz;++i) {
        int to(Next[pos][i].first),val(Next[pos][i].second) ;
        if(isok[to]) continue ;
        V.emplace_back(gp_hash_table<int,int>()) ;
        getDis(to,V.back(),val) ;
    }
    for(auto&it:V) {
        for(auto& jt:it) All[jt.first] += jt.second;
        for(auto &jt:Q) Ans[jt] += 2*it[jt];
    }
```

```
        for(auto &q:Q) {
            for(auto& it:All)
                if(All.find(q-it.first)!=All.end())
                    Ans[q] += it.second*All[q-it.first];
            for(auto& it:V)
                for(auto& e:it)
                    if(it.find(q-e.first)!=it.end())
                        Ans[q] -= e.second*it[q-e.first] ;
        }
}

void solve(int pos)
{
        pos = getCore(pos) ;
        isok[pos] = 1 ; calc(pos) ;
        int sz(Next[pos].size()) ;
        for(int i(0);i<sz;++i) {
            int to(Next[pos][i].first) ;
            if(isok[to]) continue ;
            N = Size[to] ;solve(to) ;
        }
}

int main()
{
        int n,m,q ;
        scanf("%d%d",&n,&m) ;
        Init(n) ;
        for(int i(1);i<n;++i) {
            int u,v,val ;
            scanf("%d%d%d",&u,&v,&val) ;
            addedge(u,v,val) ;
        }
        for(int i(1);i<=m;++i) {
            scanf("%d",&q) ;
            Q.push_back(q) ;
        }
        solve(1) ;
        for(auto&it: Q) puts(((Ans[it]>>1)!=0)?"AYE":"NAY") ;
}
/*
3 3
1 2 1
2 3 2
```

```
1 2 3
*/
```

(2) 点分治[直接统计写法]O(nlogn)

```
#include<bits/stdc++.h>
using namespace std ;
using namespace __gnu_cxx ;
const int MAXN = 200005 ;
bool isok[MAXN] ;
int Size[MAXN],N,maxSon/**子树最大值*/,cPos/**重心节点*/ ; ;
vector<pair<int,int> >Next[MAXN] ;///邻接表存边
void Init(int n) {///初始化.注意!!!!这里的 n 必须严格为点的个数!!!!
    for(int i(0);i<=n;++i) {isok[i] = false ;Next[i].clear() ;N = n;}
}
void addedge(int u,int v,int val = 0) { ///加一条双向边
    Next[u].push_back({v,val}) ;Next[v].push_back({u,val}) ;
}
void dfs(int pos,int fa=-1) { ///dfs 找树重心
    Size[pos] = 1 ;
    int maxs(0),sz(Next[pos].size()) ;
    for(int i(0);i<sz;++i) {
        int to(Next[pos][i].first);
        if(to == fa||isok[to]) continue ;
        dfs(to,pos) ; Size[pos] += Size[to] ;
        maxs = max(maxs,Size[to]) ;
    }
    maxs = max(maxs,N-Size[pos]) ;
    if(maxs<maxSon) {maxSon=maxs;cPos=pos;}
}
inline int getCore(int pos)
{///找到树上以 pos 为根的重心
    maxSon = 0x3f3f3f3f ;dfs(pos) ;return cPos ;
}
void getDis(int pos,unordered_map<int,int>& mp,int sum = 0,int fa = -1)
{///统计以 pos 为根子树里的点到 pos 的距离 sum 的数量并且存到 mp 中.
    ++ mp[sum] ;
    int sz(Next[pos].size()) ;
    for(int i(0);i<sz;++i) {
        int to(Next[pos][i].first),val(Next[pos][i].second) ;
        if(isok[to]||to == fa) continue ;
        getDis(to,mp,sum + val,pos) ;
    }
}
```

```
vector<int> Q ;
unordered_map<int,int> Ans/**答案*/,All/**全局桶*/,tmp/**临时桶*/ ;
void calc(int pos) { ///计算过 pos 的节点的答案
    All.clear() ;///每次要清空以 pos 为根的全局桶.
    int sz(Next[pos].size()) ;
    for(int i(0);i<sz;++i) {
        int to(Next[pos][i].first),val(Next[pos][i].second) ;
        if(isok[to]) continue ;
        tmp.clear() ;///清空临时桶
        getDis(to,tmp,val) ;///得到以 pos 为根的每一个点到 pos 的距离并且存到 tmp
里.(点权的话,这里要改成 val[pos])
        for(auto& q:Q) {///遍历每次离线的询问
            for(auto& e:tmp) ///遍历当前桶 tmp 并且统计答案.
                if(All.count(q-e.first))Ans[q] += All[q-e.first]*e.second;
            if(tmp.count(q)) Ans[q] += tmp[q]; ///统计完完全经过 pos 答案,还要统计
单链的答案.
        }
        for(auto& e:tmp) All[e.first] += e.second;///把当前子树的信息添加到全局桶里
边
        ///先算 tmp 与不包含 tmp 的 all 之间的贡献,然后再把 tmp 添加进去,这样的话
就不用容斥了.
    }
}
void solve(int pos) { ///统计以 pos 为根的答案
    pos = getCore(pos) ;///找重心
    isok[pos] = 1 ; calc(pos) ;///计算经过 pos 的答案
    int sz(Next[pos].size()) ;
    for(int i(0);i<sz;++i) {
        int to(Next[pos][i].first) ;
        if(isok[to]) continue ;
        N = Size[to] ;solve(to) ;///分治计算子树的答案
    }
}
int main()
{
    int n,m,q ;
    scanf("%d%d",&n,&m) ; Init(n) ;
    for(int i(1);i<n;++i) {
        int u,v,val ;
        scanf("%d%d%d",&u,&v,&val) ;
        addedge(u,v,val) ;
    }
    for(int i(1);i<=m;++i) {
        scanf("%d",&q) ; Q.push_back(q) ;///离线询问
```

```
    }
    solve(1) ;
    for(auto&it: Q) puts((Ans[it]>0)?"AYE":"NAY") ;
}
/*
3 3
1 2 1
2 3 2
1 2 3
*/
```

4. 数值计算

(1) 中缀转后缀&&后缀计算 O(n)

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

int CMP(char a){///运算符优先级设定，最小一定是"#"
    if(a == '#'){return 0;}
    if(a == '('){return 1;}
    if(a == '+' || a == '-'){return 3;}
    if(a == '*' || a == '/'){return 4;}
    if(a == '^'){return 5;}
    if(a == ')'){return 6;}
    return -1;
}

string Change(string in){///中缀转后缀，一定要先加一个"#"
    /**处理后的后缀表达式中存在多个空格*/
    /**不需要的话请删去所有的 （NPR += " ";） */
    in += "#";
    int L = in.length();
    string t = "";
    string NPR = "";
    stack<char> S;
    for(int i = 0;i < L;i ++){
        if(in[i] >= '0' && in[i] <= '9'){t += in[i];}
        else if(in[i] == '.'){t += in[i];}
        else if(in[i] == '#'){
            if(t.length() != 0){NPR += t;NPR += " ";}
            t = "";
        }
```

```
        else if(in[i] == '+' || in[i] == '-' || in[i] == '*' || in[i] == '/' || in[i] == '^'){
            if(t.length() != 0){NPR += t;NPR += " ";}
            t = "";
            if(S.size() == 0){
                S.push(in[i]);
            }
            else if(CMP(S.top()) < CMP(in[i])){
                S.push(in[i]);
            }
            else if(CMP(S.top()) >= CMP(in[i])){
                while(S.size() && CMP(S.top()) >= CMP(in[i])){
                    NPR += S.top();NPR += " ";
                    S.pop();
                }S.push(in[i]);
            }
        }
        else if(in[i] == '('){
            if(t.length() != 0){NPR += t;NPR += " ";}
            t = "";
            S.push(in[i]);
        }
        else if(in[i] == ')'){
            if(t.length() != 0){NPR += t;NPR += " ";}
            t = "";
            while(S.size() && S.top() != '('){
                NPR += S.top();
                NPR += " ";
                S.pop();
            }
            S.pop();
        }
    }
    while(S.size() != 0){
        NPR += S.top();
        NPR += " ";
        S.pop();
    }
    return NPR;
}

double Cal(double a,double b,char c){
    double out = 0.0;
    if(c == '+'){out = a + b;}
    else if(c == '-'){out = a - b;}
```

```
    else if(c == '*'){out = a * b;}
    else if(c == '/'){out = a / b;}
    else if(c == '^'){out = pow(a,b);}
    return out;
}

double Cal_NPR(string now){
    string Sp[5005];///Sp 的大小取决于表达式中的数字项数
    int cnt = 0;
    string T = "";
    /** 这段是特判最前端有没有前导符号以及对负数的判断*/
    /** 如果表达式一定正确是没有用的*/
    /**
    for(int i = 0;i < L;i ++){
        if(in[i] == '-' ||in[i] == '+'){
            if(i == 0 || in[i - 1] == '('){
                in.insert(i,"0");
            }
        }
    }
    L = in.length();
    */
    for(int i = 0;i < now.size();i ++){
        if(now[i] == ' '){Sp[cnt ++] = T;T = "";}
        else{T += now[i];}
    }
    if(T.size()){Sp[cnt ++] = T;T = "";}
    stack<double> S;
    while(S.size()){S.pop();}
    for(int i = 0;i < cnt;i ++){
        if(Sp[i].size() == 0){continue;}
        if(Sp[i][0] <= '9' && Sp[i][0] >= '0'){
            double num;
            num = atof(Sp[i].c_str()) ;
            S.push(num);
        }
        else{
            double b = S.top();S.pop();
            double a = S.top();S.pop();
            double c = Cal(a,b,Sp[i][0]);
            S.push(c);
        }
    }
    double ans = S.top();
```

```
    return ans;
}

int main()
{
    while(1){
        string in;
        cin >> in;
        cout << Change(in) << endl;///中缀转后缀
        cout << Cal_NPR(Change(in)) << endl;///后缀计算
    }
    return 0;
}
```

(2) 自适应辛普森积分

```
#include <bits/stdc++.h>
using namespace std ;
class Simpson
{
private:
    ///区间[a,b]上的辛普森值,已知区间长度 L 和端点及中点处的函数值 A,B,C.
    inline double simpson(double L,double A,double B,double C){
        return (A+4*C+B)*L/6;
    }
    template<typename Func>
    double asr(Func f,double a,double b,double eps,double A,double B,double C){
        double len(b-a),c(a+len/2),D(f(a+len/4)),E(f(a+3*len/4));
        double L(simpson(len/2,A,C,D)),R(simpson(len/2,C,B,E)),AB(simpson(len,A,B,C));
        if(fabs(L+R-AB)<=15*eps) return L+R+(L+R-AB)/15;
        else return asr(f,a,c,eps/2,A,C,D)+asr(f,c,b,eps/2,C,B,E);
    }
public:
    template<typename Func>
    inline double calc(Func f,double l,double r,double eps = 1e-6)
    {
        return asr(f,l,r,eps,f(l),f(r),f(l+(r-l)/2)) ;
    }
}simpson;

int main(){
    double l,r,a,b,c,d;
    scanf("%lf%lf%lf%lf%lf%lf",&a,&b,&c,&d,&l,&r);
    auto f = [&](double x)->double {return (c*x+d)/(a*x+b);} ;
```

```
    ///f 定义的是被积函数
    printf("%.6f\n",simpson.calc(f,l,r));
    return 0;
}
```

5. 根号算法

(1) 多维偏序[分块+二分]O(k*n*m/32)

```
#define O2 __attribute__((optimize("O2")))
using namespace std;
///时间复杂度 O(k*n*m/32)
class PoSet
{
static const int MAXN = 100005 ;
static const int SQR = sqrt(MAXN) + 7 ;
static const int K = 2 ;
public:
    vector<pair<int,int> > a[K];
    ///用的时候别忘了输入 K 维的点,first 是值 val,second 是编号 id.
    ///输入维度和数据都从 0 开始,[0,n-1].
    bitset<MAXN> bs[K][SQR];
    int bCnt,bSize,n;

    inline void clear() {n = 0 ;for(int i(0);i<K;++i) a[i].clear() ;}

    inline void push_back(int *p,int id)
    {
        ++ n ;
        for(int i(0);i<K;++i) a[i].push_back({p[i],id}) ;
    }

    O2 void Init()
    {
        n = a[0].size();
        for (int i(0); i < K; ++i) sort(a[i].begin(), a[i].end());
        bCnt = sqrt(n + .0);
        bSize = ceil(n / (bCnt + .0));
        for (int i(0); i < K; ++i){
            for (int k(0); k < bCnt; ++k){
                bs[i][k].reset();
            }
        }
```

```
            for (int i(0); i < K; ++i){
                for (int j(0); j < n; ++j){
                    bs[i][j / bSize].set(a[i][j].second);
                }
            }
            for (int i(0); i < K; ++i){
                for (int j(0); j < bCnt; ++j){
                    if (j) bs[i][j] |= bs[i][j - 1];
                }
            }
        }

        O2 int Q(int* p)
        {
            bitset<MAXN> res, tmp;
            res.set();
            for(int k(0);k<K;++k) {
                tmp.reset();
                int id = upper_bound(a[k].begin(),a[k].end(),pair<int,int>{p[k], n + 1}) - a[k].begin() - 1;
                if (id < 0){res.reset(); continue;}
                if (id / bSize) tmp = bs[k][id/bSize-1];
                int st((id / bSize)*bSize),ed(id);
                for (int i(st); i <= ed; ++i) tmp.set(a[k][i].second);
                res &= tmp;
            }
            return res.count();
        }
    }poset[5];
```

(2) 分块带修改区间第 K 小(大)O(nlognsqrtn)

```
///ZOJ 2112
/**n=1e5 的时候限时 3s 用*/
#include<bits/stdc++.h>
#define in(x) (scanf("%d",&x))
#define out(x) printf("%d",x)
#define sp putchar(' ')
#define en putchar('\n')
using namespace std ;
const int MAXN(200005) ;
const int inf(0x3f3f3f3f) ;
int L[MAXN]/**每个块的左端点*/,R[MAXN]/**每个块的右端点*/ ;
int pos[MAXN]/**每个点属于那个块*/,arr[MAXN] ;
pair<int,int> a[MAXN];/**需要维护的单点信息(data,index)*/
int n,m,k,bLen/**每个块大小*/,bCnt/**块的个数*/,maxs(-inf),mins(inf) ;
```

```
/**利用分块和二分实现带单点修改的区间第 K 小(大)*/
/**初始化 O(nlogn)*/
/**单次查询 O((L/sqrt(n))*logn),L 为查找区间长度*/
/**单次修改时间复杂度 O(2*sqrt(n))*/
void Init(int n)
{
    bLen = (int)sqrt(max(n*log(n),1.5)) ;
    ///这里用 sqrt(n*logn)比 sqrt(n)要快
    bCnt = n/bLen ;
    R[0] = 0;
    for(int i(1);i<=bCnt;++i) {
        L[i] = R[i-1]+1;R[i] = i*bLen;
        sort(a+L[i],a+R[i]+1) ;///保证块内有序
        for(int j(L[i]);j<=R[i];++j)
            pos[j] = i ;
    }
    if(R[bCnt]!=n)///不完整的块的处理
        L[++bCnt]=R[bCnt-1]+1,R[bCnt] = n;
    for(int i(L[bCnt]);i<=R[bCnt];++i) pos[i] = bCnt ;
}

inline int check(int l,int r,int mid) ///统计区间比 mid 小的数的个数
{
    /**中间的块二分,两边的块暴力*/
    int pl(pos[l]),pr(pos[r]),cnt(0);
    for(int i(pl+1);i<=pr-1;++i) {///整块直接二分
        cnt += lower_bound(a+L[i],a+R[i]+1,make_pair(mid,0)) - (a+L[i]);
    }
    ///printf("fen(%d) == %d \n",mid,cnt) ;
    for(int i(L[pl]);i<=R[pl];++i) {///左边暴力
        if(a[i].first<mid&&a[i].second>=l&&a[i].second<=r)
            ++cnt ;
    }
    if(pl!=pr) ///同一个块就不需要再次暴力了
    for(int i(L[pr]);i<=R[pr];++i) {///右边暴力
        if(a[i].first<mid&&a[i].second<=r&&a[i].second>=l)
            ++cnt ;
    }
    ///printf("check(%d) == %d\n",mid,cnt) ;
    return cnt ;
}
int Q(int l,int r,int k) ///查询[l,r]内第 k 小
{
    int ll(mins),rr(maxs) ;
```

```
    while(ll<rr) {
        int mid(ll+rr+1>>1) ;
        int chk(check(l,r,mid)) ;
        if(chk<=k-1)///小于 mid 的有 k-1 个,那么 mid 为第 k 小
            ll = mid ;
        else
            rr = mid - 1 ;
    }
    return ll ;
}
int change(int p,int x) ///单点修改
{
    int pl(L[pos[p]]),pr(R[pos[p]]),id(-1) ;
    for(int i(pl);i<=pr;++i) {///暴力修改
        if(a[i].second == p) {
            id = i ; break ;
        }
    }
    arr[p] = a[id].first = x ;
    ///下边冒泡一下,保持块有序.
    for(int i(id);i>pl&&a[i]<a[i-1];--i) swap(a[i],a[i-1]) ;
    for(int i(id);i<pr&&a[i+1]<a[i];++i) swap(a[i],a[i+1]) ;
}
inline void print()///debug 输出
{
    for(int i(1);i<=n;++i) {printf("[%d]=%d,",a[i].second,a[i].first);}en;
}
int main()
{
    int T ;     in(T) ;
    while(T-->0)
    {
        in(n),in(m) ;
        for(int i(1);i<=n;++i)
            in(arr[i]),a[i].first = arr[i],    a[i].second = i,
            maxs=max(maxs,arr[i]),mins=min(mins,arr[i]) ;
        Init(n) ;
        ///print() ;
        char op[105] ;    int l,r ;
        while(m-->0) {
            scanf("%s",op) ;
            in(l),in(r) ;
            if(op[0] == 'Q') {
                in(k) ;    out(Q(l,r,k)),en;
```

```
            } else change(l,r) ;
        }
}
```

(3) 分两类讨论 O(nsqrt(n))

```
/*
    2018 沈阳 ICPC 网络赛 J.Ka Chang
题意: 给你一颗树,树上各点有初始权值,你有两种操作:
    1.给树中深度为 l 的点全部+x,(根节点为 1,深度为 0)
    2.求出以 x 为根的子树权值和.
思路:
    首先,这个题我们想不到任何数据结构直接维护答案,但是我们有两种暴力算法.
        (1)利用 dfs 序列,我们可以 O(logn)查询子树的和,但是修改一层是 O(nlogn)的
        (2)我们对于每一层打标记,这样修改一层是 O(1)的,但是通过二分查询子树是 O(nlogn)的
    所以我们可以折中一下两种暴力,按照每一层节点个数分两类讨论:
        1.该层节点个数大于 bSize:
            利用暴力(2),修改一层时间复杂度为 O(1),通过二分,单次查询 O(bCnt*logn)
        2.该层节点个数小于 bSize:
        利用暴力(1),修改一层时间复杂度为 O(bCnt*logn),查询 O(logn).
    取 bCnt = sqrt(n),总体时间复杂度为最坏 n*logn*sqrt(n).
*/
#include <bits/stdc++.h>
#define LOWBIT(x) (x&(-x))
const int MAXN (100005) ;
const int MAXM (100005) ;
using namespace std ;
struct Edge{int to,Next;}edge[MAXM<<1];
int head[MAXN],tot;
int deep[MAXN],Size[MAXN],id[MAXN],cur ;
int64_t tag[MAXN] ;
vector<int> v[MAXN] ;
void add_edge(int u,int v)
{
    edge[tot].to = v ;
    edge[tot].Next = head[u] ;
    head[u] = tot ++ ;
}

int dfs(int pos=1,int _deep=0,int _fa=-1)
{
    id[pos] = ++ cur ;
    Size[pos] = 1 ;
    deep[pos] = _deep ;
    v[_deep].push_back(id[pos]) ;
```

```
    int ans(_deep) ;
    for(int i(head[pos]);~i;i=edge[i].Next) {
        int to(edge[i].to) ;
        if(to == _fa) continue ;
        ans = max(dfs(to,_deep+1,pos),_deep) ;
        Size[pos] += Size[to] ;
    }
    return ans ;
}
class BIT
{
public:
    int64_t T[MAXN];
    BIT(){clear();}
    void clear(int _n=MAXN)
    {
        for(int i(0);i<MAXN;++i) T[i] = 0 ;
    }
    void add(int pos,int64_t val)
    {
        for(int i(pos);i<MAXN;i+=LOWBIT(i))
            T[i] += val ;
    }
    int64_t Q(int pos)
    {
        if(pos == 0) return 0;
        int64_t ans(0) ;
        for(int i(pos);i;i-=LOWBIT(i))
            ans += T[i] ;
        return ans ;
    }
};
BIT tree ;
vector<int> higher;
void Init(int n)
{
    for(int i(0);i<=n;++i) {
        v[i].clear() ;
        head[i] = -1 ;
        tag[i] = 0 ;
    }
    tree.clear(n) ;
    higher.clear() ;
    cur = tot = 0 ;
```

```
}
int main()
{
    int n,m ;
    while(~scanf("%d%d",&n,&m))
    {
        Init(n) ;
        for(int i(1);i<n;++i) {
            int u,v ;
            scanf("%d%d",&u,&v) ;
            add_edge(u,v) ;
            add_edge(v,u) ;
        }
        int d = dfs() ;
        int bSize = (int)sqrt(n) ;
        for(int i(0);i<=d;++i) {
            if(v[i].size()>bSize) {
                higher.push_back(i) ;
            }
        }
        while(m-->0) {
            int op,pos ;
            int64_t val ;
            scanf("%d",&op) ;
            if(op == 1) {
                scanf("%d%lld",&pos,&val) ;
                if(v[pos].size()>bSize) {
                    tag[pos] += val;
                } else {
                    for(auto &it:v[pos])
                        tree.add(it,val) ;
                }
            } else {
                scanf("%d",&pos) ;
                int64_t ans=tree.Q(id[pos]+Size[pos]-1)-tree.Q(id[pos]-1) ;
                for(auto &it:higher) {
                    auto pl = lower_bound(v[it].begin(),v[it].end(),id[pos]) ;
                    auto pr = upper_bound(v[it].begin(),v[it].end(),id[pos]+Size[pos]-1) - 1;
                    ans += (pr-pl+1)*tag[it] ;
                }
                printf("%lld\n",ans) ;
            }
        }
    }
```

}

(4) 莫队求子区间异或和等于 K 的数目 O(nsqrt(n))

```
/*
记一个前缀 xor 和 s ，那么贡献就转化成两个问题(需分别 O(1) 解决):
（1）移动 R 指针:    有多少个 i∈[l-1,r-1]满足 s[i] xor s[r]=k.
（2）移动 L 指针:    有多少个 i∈[l,r]满足 s[i] xor s[l-1]=k
由 a xor b=c 则 a xor c=b 得到：
（1）的条件：
移动 R 指针： 多少个 i 满足 s[r] xor k =  s[i]
（2）的条件：
移动 L 指针：多少个 i 满足  s[l-1]  xor k = s[i]
因此，需要在莫队移动指针的过程中，维护一个桶 Cnt[]，以 s[i]为关键字，Cnt[s[i]]记录 s[i]出现的次数
条件(1) 里的 i 的个数就变成了 Cnt[s[r]^k]
条件(2) 里的 i 的个数就变成了 Cnt[s[l-1]^k]
这样就能做到 O(1) 查询上面两个问题了。
复杂度 O(n√n) 。
*/
#include <bits/stdc++.h>
using namespace std ;
const int MAXN (2000055) ;
int pos[MAXN] ;
struct Node
{
    int l,r,id ;
    bool operator < (const Node& b) const
    { /// 莫队经典排序
        if(pos[l] == pos[b.l]) {
            return r<b.r ;
        }
        return pos[l]<pos[b.l] ;
        /* ///奇偶优化
        if(pos[l]^pos[b.l]) {
            return pos[l]<pos[b.l] ;
        } else {
            if(pos[l]&1) {
                return r<b.r ;
            } else {
                return r>b.r ;
            }
          }
        */
    }
};
```

```
Node node[MAXN] ;
int arr[MAXN],xorSum[MAXN]/**异或和*/,Cnt[MAXN]/**记录出现的次数*/,k ;
int64_t ans/**指针移动过程中的中间答案*/,ANS[MAXN]/**每个询问最终的答案*/ ;
void insert(int p)
{
	ans += Cnt[xorSum[p]^k]; ///算贡献
	++ Cnt[xorSum[p]] ;
}
void erase(int p)
{
	--Cnt[xorSum[p]] ;
	ans -= Cnt[xorSum[p]^k]; ///算贡献
}
int main()
{
	int n,m;
	xorSum[0] = 0 ;
	scanf("%d%d%d",&n,&m,&k) ;
	int bSize = (int)sqrt(n) ;
	for(int i(1);i<=n;++i) {
		scanf("%d",arr+i) ;
		xorSum[i] = xorSum[i-1] ^ arr[i] ; ///预处理前缀异或和
		pos[i] = i/bSize ; ///分块
	}
	for(int i(1);i<=m;++i) {
		int l,r ;
		scanf("%d%d",&l,&r) ;
		node[i].l = l ;    node[i].r = r ;        node[i].id = i ;
	}
	sort(node+1,node+m+1) ;
	int L(1),R(0) ;    ans = 0 ;      Cnt[0] = 1 ; ///初始化要注意一下哈！
	for(int i(1);i<=m;++i) {
	    /**指针在移动的过程中，要保证 R 始终大于等于 L，所以要先移动 R，后移动 L**/
	    /**有一些题目先移动哪个都无所谓，但是有的时候先移动 L 会 WA(比如训练赛中的.....)**/
		while(R<node[i].r) ++R,insert(R) ;       while(R>node[i].r) erase(R),--R ;
		while(L<node[i].l) erase(L-1),++L ;      while(L>node[i].l) --L,insert(L-1) ;
		ANS[node[i].id] = ans;
	}
	for(int i(1);i<=m;++i) {
		printf("%lld\n",ANS[i]) ;
	}
}
```

6. 计算几何

(1) 二维计算几何

```
///#include <bits/stdc++.h>
#include <iostream>
#include <stdio.h>
#include <math.h>
#include <iomanip>
#include <algorithm>
#define PI acos(-1)
const double eps (1e-13) ;
using namespace std ;
/*********************************************************/
/**基本定义**/
inline bool is0(double x) {return (x>0?x:-x)<=eps ;}
class Point
{
    public:
    double x,y;
    Point(double _x=0,double _y=0):x(_x),y(_y) {}
    bool operator == (const Point& b) {return is0(x-b.x)&&is0(y-b.y)<eps ;}
};
const Point O(0.0,0.0) ;
class Vector///向量
{
public:
    double x,y ;
    Vector(Point a,Point b):x(b.x-a.x),y(b.y-a.y){}
    Vector(double _x,double _y):x(_x),y(_y) {};
    double operator*(const Vector &b)///点积
        {return x*b.x+y*b.y ;}
    double operator^(const Vector &b)///叉积:右手定则判断正负号
        {return x*b.y-b.x*y ;}
    Vector operator-(const Vector &b)
        {return Vector(x-b.x,y-b.y);}
    Vector operator+(const Vector &b)
        {return Vector(x+b.x,y+b.y);}
    double len() {return sqrt(x*x+y*y);} ///向量模长
};
class Circle
{
public:
```

```
        Point o;double r ;
        Circle (const Point &_o=O,const double _r=0.0):o(_o),r(_r){} ;
};
inline double len (const Point a,const Point b)
{///点 a 到点 b 距离
        double dx(a.x-b.x),dy(a.y-b.y);
        return sqrt(dx*dx+dy*dy) ;
}
/*****************************************************/

inline double pointToSegment(const Point &a,const Point &b,const Point &p)
{///点 p 到线段<a,b>距离
      double ap_ab(Vector(a,b)*Vector(a,p));
      if (ap_ab<=0.0) return len(a,p);
      double d2(len(a,b));d2*=d2 ;
      if (ap_ab>=d2) return len(b,p);
      double r(ap_ab/d2);
      double px(a.x+(b.x-a.x)*r);
      double py(a.y+(b.y-a.y)*r);
      return len(p,Point(px,py));
}

inline double pointToLine(const Point &a,const Point &b,const Point &p)
{///点 p 到直线<a,b>距离
      double ap_ab(fabs(Vector(a,p)*Vector(a,b))) ;
      double d(ap_ab/len(a,b)),l(len(a,p)) ;
      return    sqrt(l*l-d*d);
}

inline double getS(const Point &a,const Point &b,const Point &c)
{///求<a,b,c>三点组成三角形的面积(有正负号!!!!)
        return 0.5*(Vector(a,b)^Vector(a,c)) ;
}

/**判断三点<a,b,c>是否共线方法:getS(a,b,c)等于 0 就是共线**/

inline bool isIntersect(const Point &a,const Point &b,const Point &c,const Point &d)
{///判断线段[注意是线段不是直线!!!]<a,b>和<c,d>是否相交
        Vector ac(a,c),ad(a,d),bc(b,c),bd(b,d) ;
        Vector ca(c,a),da(d,a),cb(c,b),db(d,b) ;
        return (ac^ad)*(bc^bd)<=0.0&&(ca^cb)*(da^db)<=0.0;
}

inline int isParallel(const Point &a,const Point &b,const Point& c,const Point& d)
```

```
{///判断两向量(直线)[<a,b>,<c,d>]是否平行,共线的话,返回-1,平行返回 1,相交返回 0
    if(is0(getS(a,b,c))&&is0(getS(a,b,d))) return -1 ;
    return is0(Vector(a,b)^Vector(c,d)) ;
}

inline bool isVertical(const Point &a,const Point &b,const Point &c,const Point &d)
{///判断两线段(直线,向量)[<a,b>,<c,d>]是否垂直:点积是不是 0
    return is0((Vector(a,b)*Vector(c,d))) ;
}

/**
直线的一般方程为 F(x)=ax+by+c=0.既然我们已经知道直线的两个点,假设为(x0,y0),(x1,y1)那么可以得到 a=y0–y1,b=x1–x0,c=x0y1–x1y0.
**/

inline Point getCrossPoint(const Point& a,const Point& b,const Point& c,const Point& d)
{///求出<a,b> <c,d>  的交点坐标
    double a0(a.y-b.y),b0(b.x-a.x),c0(a.x*b.y-b.x*a.y) ;
    double a1(c.y-d.y),b1(d.x-c.x),c1(c.x*d.y-d.x*c.y) ;
    double D (a0*b1-a1*b0) ;
    if(is0(D)) return Point(1ull<<63,1ull<<63) ;
    return Point((b0*c1-b1*c0)/D,(a1*c0-a0*c1)/D) ;
}

inline Circle getCircumcircle(const Point& a,const Point& b,const Point& c)
{///求三角形<a,b,c>的外心(外接圆的圆心)[外接圆的半径:a*b*c*0.25/getS(a,b,c)]
    double A1(2*(b.x-a.x)),B1(2*(b.y-a.y)),C1(b.x*b.x-a.x*a.x+b.y*b.y-a.y*a.y);
    double A2(2*(c.x-b.x)),B2(2*(c.y-b.y)),C2(c.x*c.x-b.x*b.x+c.y*c.y-b.y*b.y);
    double D(A1*B2-A2*B1);
    Point ans((C1*B2-C2*B1)/D,(A1*C2-A2*C1)/D) ;
    return Circle(ans,len(ans,a)) ;
}

inline Circle getIncircle(const Point& a,const Point& b,const Point& c)
{///求三角形<a,b,c>内接圆[内切圆半径:2*getS(a,b,c)/(len(a,b)+len(b,c)+len(a,c))]
    double A(len(b,c)),B(len(a,c)),C(len(a,b)),sum(A+B+C) ;
    Point ans((A*a.x+B*b.x+C*c.x)/sum,(A*a.y+B*b.y+C*c.y)/sum) ;
    return Circle(ans,2.0*getS(a,b,c)/(A+B+C)) ;
}

inline Circle MCC(Point *pbegin,int n)
{///最小圆覆盖(Minimum circle cover):求出到包含所有点的最小圆[精度有些问题]
    Point *&p(pbegin) ;
    random_shuffle(p,p+n);
```

```
    Circle ans ;
    for(int i(0);i<n;++i) {
        if(len(ans.o,p[i])>ans.r) {
            ans = Circle(p[i]) ;
            for(int j(0);j<i;++j) {
                if(len(ans.o,p[j])>ans.r) {
                    ans.o.x = (p[i].x + p[j].x)*0.5 ;
                    ans.o.y = (p[i].y + p[j].y)*0.5 ;
                    ans.r = len(p[i],p[j])*0.5 ;
                    for(int k(0);k<j;++k) {
                        if(len(ans.o,p[k])>ans.r) {
                            ans = getCircumcircle(p[i],p[j],p[k]) ;
                        }
                    }
                }
            }
        }
    }
    return ans ;
}

void PA_Sort(Point* pbegin,Point* pend)
{ // 极角排序
    auto cmp = [&](const Point& a,const Point& b) {
        double d (Vector(O,a)^Vector(O,b)) ;
        if(is0(d)<eps) return a.x<b.x ;
        return d>0;
    };
    /*
    for(auto it(pbegin+1);it!=pend;++it) {//以左下角为开始点
        if(it->y<pbegin->y) swap(*it,*pbegin) ;
        if(it->y==pbegin->y&&it->x<pbegin->x) swap(*pbegin,*it) ;
    }
    */
    sort(pbegin+1,pend,cmp) ;
}

Point p[1000005] ;

int main()
{
    int n ,buff;
    while(~scanf("%d%d%d",&buff,&buff,&n))
    {
```

```
            for(int i(1);i<=n;++i) {
                scanf("%lf%lf",&p[i].x,&p[i].y) ;
            }
            Circle ans(MCC(p+1,n)) ;
            printf("(%.1f,%.1f).\n%.1f\n",ans.o.x,ans.o.y,ans.r) ;
        }

    }
```

(2) 三维计算几何

```
#include <bits/stdc++.h>
#define PI acos(-1)
const double eps (1e-13) ;
using namespace std ;
/*************************************************************/
/**基本定义**/
class Point
{
    public:
    double x,y,z;
    Point(double _x=0.0,double _y=0.0,double _z=0.0):x(_x),y(_y),z(_z) {}
};
const Point O(0.0,0.0,0.0) ;
class Vector///向量
{
public:
    double x,y,z ;
    Vector(Point a,Point b):x(b.x-a.x),y(b.y-a.y),z(b.z-a.z){}
    Vector(double _x=0.0,double _y=0.0,double _z=0.0):x(_x),y(_y),z(_z) {};
    double operator*(const Vector &b)///点积
        {return x*b.x+y*b.y+z*b.z ;}
    Vector operator-(const Vector &b)
        {return Vector(x-b.x,y-b.y,z-b.z);}
    Vector operator+(const Vector &b)
        {return Vector(x+b.x,y+b.y+z+b.z);}
    double len() {return sqrt(x*x+y*y+z*z);} ///向量模长
};
class Sphere
{
public:
    Point o ;double r ;
    Sphere(const Point& _o=O,double _r = 0.0):o(_o),r(_r){}
};
```

```
inline double len (const Point a,const Point b)
{///点 a 到点 b 距离
	double dx(a.x-b.x),dy(a.y-b.y),dz(a.z-b.z);
	return sqrt(dx*dx+dy*dy+dz*dz) ;
}
inline bool is0(double x) {return (x>0?x:-x)<=eps ;}
/*****************************************************/

Sphere MSC(Point* pbegin,int n)///[模拟退火法]
{///最小球覆盖(Minimum spherical coverage):包含所有点的最小球
	Point*& p(pbegin) ;
	Sphere ans(O,1e30) ;
	int s(0) ;
	double t(100000.0) ;//初始温度
	while(t>1e-13) {
		double d(len(ans.o,p[s])) ;
		for(int i(0);i<n;++i) {
			double _d(len(ans.o,p[i])) ;
			if(d<_d) s=i,d = _d;
		}
		ans.r = min(ans.r,d) ;
		ans.o.x+=(p[s].x-ans.o.x)/d*t;
		ans.o.y+=(p[s].y-ans.o.y)/d*t;
		ans.o.z+=(p[s].z-ans.o.z)/d*t;
		t *= 0.98 ;//0.98 是降温系数
	}
	return ans;
}

Point p[100005] ;

int main()
{
	int n ;
	while(~scanf("%d",&n)) {
		if(n == 0) return 0 ;
		for(int i(1);i<=n;++i) {
			scanf("%lf%lf",&p[i].x,&p[i].y) ;
		}
		Sphere ans(MSC(p+1,n)) ;
		printf("%.10lf\n%.10lf %.10lf\n",ans.r,ans.o.x,ans.o.y) ;
	}
}
```

(3) 多边形面积并

```
/**
    求简单多边形的面积并
    输入：多边形个数
    然后依次输出每个多边形。输入多边形的格式是：
    先输入多边形的顶点个数，然后按顺时针顺序依次输入每个顶点的坐标
    输出：所有多边形的面积和 所有多边形的面积并
    注意：要求多变形的边可以相交但是不能重叠
    即从所有多边形中任意取两条边，这两个线段的公共点个数只能为 0 或 1，不可以有无穷个公共点

    例：
    输入：
    2
    4  0 0 0 2 2 2 2 0
    4  1 1 1 3 3 3 3 1
    输出：
    8.0000000 7.0000000
*/
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const double inf=1e10;
const double eps=1e-12;
int sign(double x){ return fabs(x)<eps?0:(x<0?-1:1);}
struct point{
    double x,y;
    point(double a=0,double b=0):x(a),y(b){}
    point() = default;
};
point operator +(point A,point B) { return point(A.x+B.x,A.y+B.y);}
point operator -(point A,point B) { return point(A.x-B.x,A.y-B.y);}
point operator *(point A,double p){ return point(A.x*p,A.y*p);}
point operator /(point A,double p){ return point(A.x/p,A.y/p);}
bool operator ==(const point& a,const point& b){
    return fabs(a.x-b.x)<eps&&fabs(a.y-b.y)<eps;
}
double dot(point A,point B){ return A.x*B.x+A.y*B.y;}
double det(point A,point B){ return A.x*B.y-A.y*B.x;}
double det(point O,point A,point B){ return det(A-O,B-O);}///向量 OA 叉乘向量 OB
double length(point A){ return sqrt(dot(A,A));}
double area(vector<point>p){///求任意单个多边形的面积
    double ans=0; int sz=p.size();
```

```
    for(int i=1;i<sz-1;i++) ans+=det(p[i]-p[0],p[i+1]-p[0]);
    return ans/2.0;
}
double seg(point O,point A,point B){///O 点在向量 AB 所在直线上，求向量 AO 的模比上向量 AB 的模
    if(sign(B.x-A.x)==0) return (O.y-A.y)/(B.y-A.y);
    return (O.x-A.x)/(B.x-A.x);
}
vector<point>pp[110];///每一个多边形将顶点按照顺时针顺序存

pair<double,int>s[110*60];///计算每一条边界线段的贡献
double polyunion(vector<point>*p,int N){///N 表示多边形个数，每一个 vector 存放一个多边形的顺时针顶点集
    double res=0;
    for(int i=0;i<N;i++){
        int sz=p[i].size();
        for(int j=0;j<sz;j++){
            int m=0;
            s[m++]=make_pair(0,0);
            s[m++]=make_pair(1,0);
            point a=p[i][j],b=p[i][(j+1)%sz];
            for(int k=0;k<N;k++){
                if(i!=k){
                    int sz2=p[k].size();
                    for(int ii=0;ii<sz2;ii++){
                        point c=p[k][ii],d=p[k][(ii+1)%sz2];
                        int c1=sign(det(b-a,c-a));
                        int c2=sign(det(b-a,d-a));
                        if(c1==0&&c2==0){
                            if(sign(dot(b-a,d-c))){
                                s[m++]=make_pair(seg(c,a,b),1);
                                s[m++]=make_pair(seg(c,a,b),-1);
                            }
                        }
                        else{
                            double s1=det(d-c,a-c);
                            double s2=det(d-c,b-c);
                            if(c1>=0&&c2<0) s[m++]=make_pair(s1/(s1-s2),1);
                            else                                    if(c1<0&&c2>=0)
s[m++]=make_pair(s1/(s1-s2),-1);
                        }
                    }
                }
            }
```

```
                sort(s,s+m);
                double pre=min(max(s[0].first,0.0),1.0),now,sum=0;
                int cov=s[0].second;
                for(int j=1;j<m;j++){
                    now=min(max(s[j].first,0.0),1.0);
                    if(!cov) sum+=now-pre;
                    cov+=s[j].second;
                    pre=now;
                }
                res+=det(a,b)*sum;///用每一个多边形的边界构成面积并之后的多边形的
边界，每条原边界乘以其贡献
            }
        }
        return res/2;
}
int main()
{
        int N,M,i,j; point tp;
        scanf("%d",&N);
        for(i=0;i<N;i++){
            scanf("%d",&M);
            for(j=0;j<M;j++){
                scanf("%lf%lf",&tp.x,&tp.y);
                pp[i].push_back(tp);
            }
        }
        double t1=0,t2=polyunion(pp,N);
        for(i=0;i<N;i++) t1+=area(pp[i]);
        printf("%.7f %.7f\n",-t1,-t2);
        return 0;
}
```

(4) 线段树扫描线 O(logn)

```
class ScanLineSTree
{
#define MAXN (100005<<2)
#define L (pos << 1)
#define R (pos << 1 | 1)
public :
    int len[MAXN]/**至少覆盖一次的区间长度*/,
        cnt[MAXN]/**区间覆盖次数*/,LD[MAXN],RD[MAXN] ;
    ///扫描线的求周长的话,每次  ans += tree.Q()-last ;last = tree.Q() ;
    void build(int l,int r,int pos = 1)
    {
```

```
        LD[pos] = l;RD[pos] = r ;
        len[pos] = cnt[pos] = 0 ;
        if(l == r) return ;
        int mid((l+r) >> 1) ;
        build(l,mid,L) ;
        build(mid+1,r,R) ;
    }

    void push_up(int pos)
    {
        if(cnt[pos]) {
            len[pos] = RD[pos] - LD[pos] + 1;
        } else if(LD[pos] == RD[pos]) {
            len[pos] = 0 ;
        }else {
            len[pos] = len[L] + len[R] ;
        }
    }

    void insert(int l,int r,int type ,int pos = 1)
    {
        if(LD[pos] == l&&RD[pos] == r) {
            cnt[pos] += type ;
            push_up(pos) ;
            return ;
        }
        int mid((LD[pos]+RD[pos]) >> 1) ;
        if(r<=mid) insert(l,r,type,L) ;
        else if(l>mid) insert(l,r,type,R) ;
        else {
            insert(l,mid,type,L) ;insert(mid+1,r,type,R) ;
        }
        push_up(pos) ;
    }
    inline int Q() {return len[1] ;}
#undef MAXN
};
```

(5) 动态开点扫描线 O(logn)

```
#include <bits/stdc++.h>
using namespace std ;
///不用离散化的扫描线.
template<typename T = int>
class ScanLineSTree
```

```
{///动态开点,免离散化扫描线.
private :
    static constexpr int MAXN = 100005 ;
    static constexpr int N = MAXN*80 ;
    int L[N],R[N],cnt[N];T len[N] ; ;
    int tot ;
    T min_val,max_val ;
    ///len: 至少覆盖一次的区间长度 cnt: 区间覆盖次数.
    #define push_up    \
    if(cnt[pos]) {     \
        len[pos] = RD - LD + 1;    \
    } else if(LD == RD) { \
        len[pos] = 0 ; \
    }else { \
        len[pos] = len[L[pos]] + len[R[pos]] ; \
    }
    inline int newNode()
    {
        int ret(++ tot) ;
        L[ret] = R[ret] = cnt[ret] = len[ret] = 0 ;
        return ret ;
    }
    void insert(T l,T r,int type ,T LD,T RD,int pos)
    {
        if(LD == l&&RD == r) {
            cnt[pos] += type ;
            push_up ;
            return ;
        }
        T mid((LD+RD) >> 1) ;
        #define L (L[pos]?L[pos]:L[pos] = newNode())
        #define R (R[pos]?R[pos]:R[pos] = newNode())
        if(r<=mid) insert(l,r,type,LD,mid,L) ;
        else if(l>mid) insert(l,r,type,mid+1,RD,R) ;
        else {insert(l,mid,type,LD,mid,L) ;insert(mid+1,r,type,mid+1,RD,R) ;}
        #undef L
        #undef R
        push_up ;
    }
public:
    void build(T l,T r)
    {
        tot = 0 ; newNode() ; newNode() ;
        min_val = l ; max_val = r ;
```

```
    }
    inline void insert(T l,T r,int type = 1)
        {insert(l,r,type,min_val,max_val,1);}
    int Q() {return len[1] ;}
};
ScanLineSTree <>tree;

struct Node
{
    int l,r,y,type;
    bool operator <(const Node& o) const {
        if(y != o.y) return y<o.y ;
        return type > o.type ;
    }
};
vector<Node> v ;
int main()
{
    tree.build(0,1000000007) ;
    int n ;
    scanf("%d",&n) ;
    for(int i(1);i<=n;++i) {
        int x1,y1,x2,y2 ;
        scanf("%d%d%d%d",&x1,&y1,&x2,&y2) ;
        v.emplace_back(Node{x1,x2,y1,1}) ;
        v.emplace_back(Node{x1,x2,y2,-1}) ;
    }
    stable_sort(v.begin(),v.end()) ;
    int64_t ans(0) ;
    for(int i(0);i<v.size()-1;++i) {
        tree.insert(v[i].l,v[i].r-1,v[i].type) ;
        ans += 1LL * tree.Q() * (v[i+1].y - v[i].y);
    }
    printf("%lld\n",ans) ;
    return 0;
}
```

7. 数据结构

(1) 动态树

① Link_Cut_Tree[O(nlogn)]

```
/**
    Link_Cut_Tree
    类似树链剖分的强大数据结构,LCT 将某一个儿子的连边划分为实边,而连向其他子树的边划分为虚边.
区别在于虚实是可以动态变化的,因此要使用更高级更灵活的 Splay 来维护每一条由若干实边连接而成的实链!!!
    在实链剖分的基础下，LCT 支持更多的操作:

    1. 查询,修改链上的信息(最值,总和等).
    2. 随意指定原树的根(即换根).
    3. 动态连边,删边.
    4. 合并两棵树,分离一棵树.
    5. 动态维护连通性.

    可以用 LCT 维护最小生成树以及直径等信息.
    在维护链信息区间修改的时候,每次下传时注意到一颗 Splay 维护了一条链的信息,所以我只需要 split(u,v)
把 u<->v 这一条链拉出来然后就再在顶部 (v) 处打上一个标记即可.
其他处理都类似的需要 split.

    例题:luogu P1501 [国家集训队]Tree II
    一棵 n 个点的树,每个点的初始权值为 1.对于这棵树有 q 个操作,每个操作为以下四种操作之一:
        + u v c:将 u 到 v 的路径上的点的权值都加上自然数 c
        - u1 v1 u2 v2:将树中原有的边(u1,v1)删除,加入一条新边(u2,v2),保证操作完之后仍然是一棵树
        * u v c:将 u 到 v 的路径上的点的权值都乘上自然数 c
        / u v:询问 u 到 v 的路径上的点的权值和,求出答案对于 51061 的余数
*/

#include <bits/stdc++.h>
using namespace std ;
const int Mod = 51061;
class Link_Cut_Tree
{///从 1 开始.
static const int MAXN = 100005 ;
public:
```

```
    int fa[MAXN],ch[MAXN][2],Size[MAXN]/**Splay 中某一点的子树大小*/,Cnt/**
连通块的个数*/;
    int64_t val[MAXN],Sum[MAXN],lazy_add[MAXN],lazy_mul[MAXN] ;
    bool revLazy[MAXN] ; ///反转标记
    Link_Cut_Tree(int n = 0) {clear(n) ;}
    inline void clear(int n) {
        fa[0] = ch[0][0] = ch[0][1] = Size[0] = 0 ; Cnt = n ;
        for(int i(1);i<=n;++i) {
            val[i] = Sum[i] = lazy_mul[i] = 1 ;lazy_add[i] = 0;
            Size[i] = 1 ;fa[i] = ch[i][0] = ch[i][1] = revLazy[i] = 0 ;
        }
    }
    inline void setVal(int id,int _val) ///输入的时候设置一个点的点权
        {val[id] = Sum[id] = _val ;}
    inline bool isSon(int x,int id = 1) ///判断一个点是不是该点的父亲的左/右儿子
        {return ch[fa[x]][id] == x ;}
    inline bool isRoot(int x) ///判断节点是否为某一个 Splay 的根
        {return !(isSon(x,0)||isSon(x,1));}
    inline void push_up(int x) {///统计 Splay 中子树信息
        int L(ch[x][0]),R(ch[x][1]) ;
        Size[x] = 1 + Size[L] + Size[R] ;
        /// PUSH_UP:
        Sum[x] = (val[x] + Sum[L] + Sum[R])%Mod ;
    }
    inline void push_downRev(int x) { ///翻转 Splay 中节点 x 所在的子树
        if(!x) return ;
        swap(ch[x][0],ch[x][1]) ; revLazy[x] ^= 1 ;
    }
    inline void push_tag(int x,int64_t add,int64_t mul){///单个点的修改
        if(!x) return ;
        val[x] = ((val[x]*mul)%Mod + add)%Mod ;
        Sum[x] = (Sum[x]*mul)%Mod + (add*Size[x])%Mod ;Sum[x] %= Mod ;
        lazy_mul[x] *= mul ;lazy_mul[x] %= Mod ;
        lazy_add[x] = (lazy_add[x]*mul)%Mod + add ;lazy_add[x] %= Mod ;
    }

    inline void push_down(int x) {///下传一个点的所有标记信息(包括翻转)
        if(!x) return ;
        const int &L(ch[x][0]),&R(ch[x][1]) ;
        if(revLazy[x]) {push_downRev(L);push_downRev(R);revLazy[x]=0;}
        ///(PUSH_DOWN)  上边不能改,下边为维护的其他信息,按照题目意思改.
        int64_t &add(lazy_add[x]),&mul(lazy_mul[x]) ;
        push_tag(L,add,mul) ;/**修改左儿子*/ push_tag(R,add,mul) ;/**修改右儿
子*/
```

```
        add = 0 ; mul = 1; ///清除标记
    }
    inline void push_downAll(int x) {///从上往下传一个链的标记
        if(!isRoot(x)) {push_downAll(fa[x]) ;} push_down(x) ;
    }
    inline void rotate(int x) {///一次旋转操作
        bool k(isSon(x,1)) ;
        int y(fa[x]),z(fa[y]),&d(ch[x][k^1]) ;
        if(!isRoot(y)) ch[z][isSon(y,1)] = x ;
        fa[x] =z ; ch[y][k] = d ;
        if(d) fa[d] = y ;
        fa[y] = x ; d = y ;
        push_up(y) ;
    }
    inline void splay(int x) {///把一个点旋转到该点所在的 Splay 的根
        int y ; push_downAll(x) ;
        while(!isRoot(x)) {
            y = fa[x] ;
            if(!isRoot(y)) rotate((isSon(x,0)^isSon(y,0))?x:y) ;
            rotate(x) ;
        }
        push_up(x) ;
    }
    inline void access(int x) { ///把原树中根节点到 x 的链截出来变成一颗 Splay
        for(int y(0);x;x = fa[y = x]) {
            splay(x) ; ch[x][1] = y ; push_up(x) ;
        }
    }
    inline void setRoot(int x) ///把原树的根换成 x
        {access(x) ;splay(x) ;push_downRev(x) ;}
    inline int findRoot(int x) { ///找 x 所在的联通块在原树中的根
        access(x) ;splay(x) ;
        while(ch[x][0]) {push_down(x) ;x = ch[x][0];}
        push_down(x) ; splay(x) ;return x ;
    }
    inline bool isConnect(int x,int y) { /**判断 x 和 y 是否联通*/
        setRoot(x) ; return findRoot(y)==x ;
    }
    inline void split(int x,int y) {/**拉出 x<->y 的路径成为一个 Splay,y 是 Splay 的
根*/
        setRoot(x) ;access(y) ;splay(y) ;
    }
    inline bool link(int x,int y) {/**把 x 和 y 连上*/
        if(isConnect(x,y)) return false ;
```

```
        fa[x] = y; -- Cnt; return true ;
    }
    inline bool cut(int x,int y) { ///把 x 和 y 断开
        if(!isConnect(x,y)||fa[y]!=x||ch[y][0]) return false ;
        fa[y] = ch[x][1] = 0 ; ++ Cnt ; return true ;
    }
    inline void change(int u,int v,int _val,char op) {///区间修改
        split(u,v); _val%=Mod    ;
        if(op == '+') {
            push_tag(v,_val,1) ;
        } else if(op == '*') {
            push_tag(v,0,_val) ;
        }
    }
    inline int64_t Q(int u,int v) { ///区间查询
        split(u,v) ; return Sum[v] ;
    }
}lct;
char op[10] ;
int main()
{
    int n,m,u,v,x,y,val ;
    scanf("%d%d",&n,&m) ;
    lct.clear(n) ;
    for(int i = 1 ;i<n;++i) {
        scanf("%d%d",&u,&v) ;
        lct.link(u,v) ;
    }
    while(m-->0) {
        scanf("%s%d%d",op,&u,&v) ;
        string s = (char*)op ;
        if(s == "+") {
            scanf("%d",&val) ;
            lct.change(u,v,val,'+') ;
        } else if(s == "-") {
            scanf("%d%d",&x,&y) ;
            lct.cut(u,v) ; lct.link(x,y) ;
        } else if(s == "*") {
            scanf("%d",&val) ;
            lct.change(u,v,val,'*') ;
        } else if(s == "/") {
            printf("%lld\n",lct.Q(u,v)) ;
        }
    }
```

```
    return 0;
}
```

② LCT 维护子树信息 O(nlogn)

```
#include <bits/stdc++.h>
using namespace std ;
///LCT 统计子树信息:
/**
https://www.luogu.org/problem/P4219
1. 维护的信息要有可减性,如子树结点数,子树权值和,但不能直接维护子树最大最小值,因为在将一条虚边变成实边时要排除原先虚边的贡献.
2. 新建一个附加值存储虚子树的贡献,在统计时将其加入本结点答案,在改变边的虚实时及时维护.
3. 其余部分同普通 LCT,在统计子树信息时!!!!一定将其作为根节点!!!!!
4. 如果维护的信息没有可减性,如维护区间最值,可以对每个结点开一个平衡树维护结点的虚子树中的最值.
*/
class Link_Cut_Tree
{///LCT 维护连通性并统计子树大小, 下标从 1 开始.
static const int MAXN = 1000005 ;
public:
    int fa[MAXN],ch[MAXN][2],Si[MAXN]/**Splay 中某一点的虚子树大小*/,
        Size[MAXN]/**Splay 中某一点的子树大小*/,Cnt/**连通块的个数*/;
    bool revLazy[MAXN] ; ///反转标记
    Link_Cut_Tree(int n = 0) {clear(n) ;}
    inline void clear(int n) {
        fa[0] = ch[0][0] = ch[0][1] = Size[0] = Si[0] = 0 ; Cnt = n ;
        for(int i(1);i<=n;++i) {
            Size[i] = 1 ; Si[i] = 0 ;fa[i] = ch[i][0] = ch[i][1] = revLazy[i] = 0 ;
        }
    }
    inline bool isSon(int x,int id = 1) ///判断一个点是不是该点的父亲的左/右儿子
        {return ch[fa[x]][id] == x ;}
    inline bool isRoot(int x) ///判断节点是否为某一个 Splay 的根
        {return !(isSon(x,0)||isSon(x,1));}
    inline void push_up(int x) {///统计 Splay 中子树信息
        int L(ch[x][0]),R(ch[x][1]) ;
        Size[x] = 1 + Size[L] + Size[R] + Si[x] ;
        /// PUSH_UP:
    }
    inline void push_downRev(int x) { ///翻转 Splay 中节点 x 所在的子树
        if(!x) return ;
        swap(ch[x][0],ch[x][1]) ; revLazy[x] ^= 1 ;
```

```
}
inline void push_down(int x) {///下传一个点的所有标记信息(包括翻转)
    if(!x) return ;
    const int &L(ch[x][0]),&R(ch[x][1]) ;
    if(revLazy[x]) {push_downRev(L);push_downRev(R);revLazy[x]=0;}
    ///(PUSH_DOWN)  上边不能改,下边为维护的其他信息,按照题目意思改.
}
inline void push_downAll(int x) {///从上往下传一个链的标记
    if(!isRoot(x)) {push_downAll(fa[x]) ;} push_down(x) ;
}
inline void rotate(int x) {///一次旋转操作
    bool k(isSon(x,1)) ;
    int y(fa[x]),z(fa[y]),&d(ch[x][k^1]) ;
    if(!isRoot(y)) ch[z][isSon(y,1)] = x ;
    fa[x] =z ; ch[y][k] = d ;
    if(d) fa[d] = y ;
    fa[y] = x ; d = y ;
    push_up(y) ;
}
inline void splay(int x) {///把一个点旋转到该点所在的 Splay 的根
    int y ; push_downAll(x) ;
    while(!isRoot(x)) {
        y = fa[x] ;
        if(!isRoot(y)) rotate((isSon(x,0)^isSon(y,0))?x:y) ;
        rotate(x) ;
    }
    push_up(x) ;
}
inline void access(int x) { ///把原树中根节点到 x 的链截出来变成一颗 Splay
    for(int y(0);x;x = fa[y = x]) {
        splay(x) ; int& R(ch[x][1]) ;
        if(y) Si[x] -= Size[y] ;
        if(R) Si[x] += Size[R] ;
        R = y ; push_up(x) ;
    }
}
inline void setRoot(int x) ///把原树的根换成 x
    {access(x) ;splay(x) ;push_downRev(x) ;}
inline int findRoot(int x) { ///找 x 所在的联通块在原树中的根
    access(x) ;splay(x) ;
    while(ch[x][0]) {push_down(x) ;x = ch[x][0];}
    push_down(x) ; splay(x) ;return x ;
}
inline bool isConnect(int x,int y) { /**判断 x 和 y 是否联通*/
```

```
        setRoot(x) ; return findRoot(y)==x ;
    }
    inline void split(int x,int y) {/**拉出 x<->y 的路径成为一个 Splay,y 是 Splay 的根*/
        setRoot(x) ;access(y) ;splay(y) ;
    }
    inline bool link(int x,int y) {/**把 x 和 y 连上*/
        if(isConnect(x,y)) return false ;
        setRoot(y) ;///此时 y 不为根,信息得不到统计,因此需要 setRoot(y),这里不写会 WA!!!!!
        fa[x] = y; Si[y] += Size[x] ;-- Cnt; return true ;
    }
    inline bool cut(int x,int y) { ///把 x 和 y 断开
        if(!isConnect(x,y)||fa[y]!=x||ch[y][0]) return false ;
        fa[y] = ch[x][1] = 0 ; ++ Cnt ; return true ;
    }
    inline int Q(int u) { ///查询某一点的子树信息.
        setRoot(u) ; return Size[u] ;
    }
}lct;
char op[5] ;
int u,v ;
int main()
{
    int n,q ;
    scanf("%d%d",&n,&q) ;
    lct.clear(n+5) ;
    while(q-->0) {
        scanf("%s%d%d",op,&u,&v) ;
        if(op[0] == 'A') {
            lct.link(u,v) ;
        } else {
            lct.cut(u,v) ;
            printf("%lld\n",lct.Q(u)*1LL*lct.Q(v)) ;
            lct.link(u,v) ;
        }
    }
}
```

③ LCT 动态维护最小生成树 O(nlogn)

```
///最小生成树满足路径上的最大值最小. 题上如果有删边的话,离线一下就变成了加边.
#include <bits/stdc++.h>
```

```
using namespace std ;
class Link_Cut_Tree
{///从 1 开始.
static const int MAXN = 1000005 ;
public:
    int fa[MAXN],ch[MAXN][2];
    int64_t val[MAXN];
    int MaxId[MAXN] ;
    bool revLazy[MAXN] ; ///反转标记
    Link_Cut_Tree(int n = 0) {clear(n) ;}
    inline void clear(int n) {
        for(int i(0);i<=n;++i) {
            MaxId[i] = i ;
            val[i] = fa[i] = ch[i][0] = ch[i][1] = revLazy[i] = 0 ;
        }
    }
    inline void setVal(int id,int _val) ///输入的时候设置一个点的点权
        {val[id] = _val ;}
    inline bool isSon(int x,int id = 1) ///判断一个点是不是该点的父亲的左/右儿子
        {return ch[fa[x]][id] == x ;}
    inline bool isRoot(int x) ///判断节点是否为某一个 Splay 的根
        {return !(isSon(x,0)||isSon(x,1));}
    inline void push_up(int x) {///统计 Splay 中子树信息
        const int &L(ch[x][0]),&R(ch[x][1]) ;
        MaxId[x] = x ;
        if(L&&val[MaxId[x]]<val[MaxId[L]]) MaxId[x] = MaxId[L] ;
        if(R&&val[MaxId[x]]<val[MaxId[R]]) MaxId[x] = MaxId[R] ;
    }
    inline void push_downRev(int x) { ///翻转 Splay 中节点 x 所在的子树
        if(!x) return ;
        swap(ch[x][0],ch[x][1]) ; revLazy[x] ^= 1 ;
    }
    inline void push_down(int x) {///下传一个点的所有标记信息(包括翻转)
        if(!x) return ;
        const int &L(ch[x][0]),&R(ch[x][1]) ;
        if(revLazy[x]) {push_downRev(L);push_downRev(R);revLazy[x]=0;}
    }
    inline void push_downAll(int x) {///从上往下传一个链的标记
        if(!isRoot(x)) {push_downAll(fa[x]) ;} push_down(x) ;
    }
    inline void rotate(int x) {///一次旋转操作
        bool k(isSon(x,1)) ;
        int y(fa[x]),z(fa[y]),&d(ch[x][k^1]) ;
        if(!isRoot(y)) ch[z][isSon(y,1)] = x ;
```

```
        fa[x] =z ; ch[y][k] = d ;
        if(d) fa[d] = y ;
        fa[y] = x ; d = y ;
        push_up(y) ;
    }
    inline void splay(int x) {///把一个点旋转到该点所在的 Splay 的根
        int y ; push_downAll(x) ;
        while(!isRoot(x)) {
            y = fa[x] ;
            if(!isRoot(y)) rotate((isSon(x,0)^isSon(y,0))?x:y) ;
            rotate(x) ;
        }
        push_up(x) ;
    }
    inline void access(int x) { ///把原树中根节点到 x 的链截出来变成一颗 Splay
        for(int y(0);x;x = fa[y = x]) {
            splay(x) ; ch[x][1] = y ; push_up(x) ;
        }
    }
    inline void setRoot(int x) ///把原树的根换成 x
        {access(x) ;splay(x) ;push_downRev(x) ;}
    inline int findRoot(int x) { ///找 x 所在的联通块在原树中的根
        access(x) ;splay(x) ;
        while(ch[x][0]) {push_down(x) ;x = ch[x][0];}
        push_down(x) ; splay(x) ;return x ;
    }
    inline bool isConnect(int x,int y) { /**判断 x 和 y 是否联通*/
        setRoot(x) ; return findRoot(y)==x ;
    }
    inline void split(int x,int y) {/**拉出 x<->y 的路径成为一个 Splay,y 是 Splay 的
根*/
        setRoot(x) ;access(y) ;splay(y) ;
    }
    inline bool link(int x,int y) {/**把 x 和 y 连上*/
        if(isConnect(x,y)) return false ;
        fa[x] = y; return true ;
    }
    inline bool cut(int x,int y) { ///把 x 和 y 断开
        if(!isConnect(x,y)||fa[y]!=x||ch[y][0]) return false ;
        fa[y] = ch[x][1] = 0 ; return true ;
    }
}lct;
struct pair_hash ///pair
{
```

```
    template<class T1, class T2>
    std::size_t operator() (const std::pair<T1, T2>& p) const {
        auto h1 = std::hash<T1>{}(p.first);
        auto h2 = std::hash<T2>{}(p.second);
        return h1 ^ h2;
    }
};
unordered_map<pair<int,int>,int,pair_hash> hashs ; ///两个哈希表,存边权转点权信息.
unordered_map<int,pair<int,int> > rHashs ;
void Link(int u,int id,int v,int val) {
    hashs[{u,v}] = id; rHashs[id] = {u,v} ;
    lct.setVal(id,val); lct.link(u,id) ; lct.link(id,v) ;
}
void Cut(int id) {
    auto& e(rHashs[id]) ;
    lct.cut(e.first,id) ; lct.cut(id,e.second) ;
    hashs.erase(e) ; rHashs.erase(id) ;
}
void Cut(int u,int v) {Cut(hashs[{u,v}]);}

int main()
{
    int n,m,u,v,val ;
    scanf("%d%d",&n,&m) ;
    lct.clear(m+n+5) ;
    int Cnt = n ; ///记录当前联通快个数
    int64_t Ans = 0 ; ///记录最小生成树
    for(int i = 1;i<= m;++i) {
        scanf("%d%d%d",&u,&v,&val) ;
        if(u == v) continue ;
        if(u>v) swap(u,v) ;
        if(lct.isConnect(u,v)) { ///如果构成环了,那么就删掉环内权值最大的边
            lct.split(u,v) ;
            int id = lct.MaxId[v] ; ///得到(u,v)链当中,边权最大的标号 id
            if(lct.val[id] <= val) continue ; ///如果要加入的边的权值>=(u,v)链中最大边权,那就不用添加了.
            Cut(id) ;Ans -= lct.val[id]; Ans += val ;///否则断掉(u,v)链中的最大边,然后加入要加入的边.
            id=++n ;Link(u,id,v,val) ;
        } else { ///构不成环的话,直接连边就行了.
            int id = ++n ; Ans += val ;
            Link(u,id,v,val) ; -- Cnt ;
        }
```

```
    }
    printf(((Cnt==1)?"%lld\n":"orz\n"),Ans) ;
    return 0;
}
```

④ LCT 维护树的直径以及里某个点最远的点 O(nlogn)

```
#include <bits/stdc++.h>
const int64_t INF = 0x3f3f3f3f3f3f3f3f ;
const int MAXN = 200005 ;
using namespace std ;
/// LCT 动态维护直径模板.
/// Lightning Routing I (https://nanti.jisuanke.com/t/41398)
/**
    单点修改,查询距离某一个点最远的点.(也可以支持查询树的直径.)
*/
template <typename T>
class Heap ///可以获得第一大和第二大的堆
{
    multiset<T> S ;
public:
    inline void insert(T val) {S.insert(val);}
    inline bool erase(T val) {
        auto it(S.find(val)) ; if(it == S.end()) return false ;
        S.erase(it); return true ;
    }
    inline T first()/**求最大值*/{return S.empty()?-INF:*S.rbegin();}
    inline T second()/**求次大值*/{return S.size()<2?-INF:*(++S.rbegin());}
    inline void clear() {S.clear() ;}
};

class Link_Cut_Tree
{
static const int N = MAXN ;
public:
    int fa[N],ch[N][2]/**Splay 中某一点的子树大小*/;
    int64_t MaxL[N],MaxR[N],MaxD[N],Sum[N],val[N] ;
    bool revLazy[N] ; ///反转标记
    Heap<int64_t> H[N],P[N] ;
    void heapInsert(int x,int y) {H[x].insert(MaxL[y]); P[x].insert(MaxD[y]) ;}
    void headErase(int x,int y) {H[x].erase(MaxL[y]); P[x].erase(MaxD[y]) ;}
    Link_Cut_Tree(int n = 0) {clear(n) ;}
    inline void clear(int n) {
        for(int i(0);i<=n;++i) {
```

```
            val[i] = Sum[i] = 0 ;
            MaxL[i] = MaxR[i] = MaxD[i] = 0 ;
            H[i].clear(); P[i].clear() ;
            fa[i] = ch[i][0] = ch[i][1] = revLazy[i] = 0 ;
        }
    }
    inline void setVal(int id,int64_t _val) ///输入的时候设置一个点的点权
        {val[id] = Sum[id] = _val ;}
    inline bool isSon(int x,int id = 1) ///判断一个点是不是该点的父亲的左/右儿子
        {return ch[fa[x]][id] == x ;}
    inline bool isRoot(int x) ///判断节点是否为某一个 Splay 的根
        {return !(isSon(x,0)||isSon(x,1));}
    inline void push_up(int x) {///统计 Splay 中子树信息
        int L(ch[x][0]),R(ch[x][1]) ;
        Sum[x] = val[x] + Sum[L] + Sum[R];
        int64_t t(max((int64_t)0,H[x].first())),
                maxl(max(t,MaxR[L])+val[x]),maxr(max(t,MaxL[R])+val[x]) ;
        MaxL[x] = max(MaxL[L],Sum[L]+maxr) ;MaxR[x] =
max(MaxR[R],Sum[R]+maxl) ;
        MaxD[x] = max({MaxR[L]+maxr,MaxL[R]+maxl,MaxD[L],MaxD[R]}) ;
        MaxD[x] =
max({MaxD[x],P[x].first(),val[x]+t+max((int64_t)0,H[x].second())}) ;
    }
    inline void push_downRev(int x) { ///翻转 Splay 中节点 x 所在的子树
        if(!x) return ;
        swap(ch[x][0],ch[x][1]) ;
        swap(MaxL[x],MaxR[x]) ; revLazy[x] ^= 1 ;
    }
    inline void push_down(int x) {///下传一个点的翻转标记信息
        if(!x||!revLazy[x]) return ;
        const int &L(ch[x][0]),&R(ch[x][1]) ;
        push_downRev(L);push_downRev(R);revLazy[x]=0;
    }
    inline void push_downAll(int x) {///从上往下传一个链的标记
        if(!isRoot(x)) {push_downAll(fa[x]) ;} push_down(x) ;
    }
    inline void rotate(int x) {///一次旋转操作
        bool k(isSon(x,1)) ;
        int y(fa[x]),z(fa[y]),&d(ch[x][k^1]) ;
        if(!isRoot(y)) ch[z][isSon(y,1)] = x ;
        fa[x] =z ; ch[y][k] = d ;
        if(d) fa[d] = y ;
        fa[y] = x ; d = y ;
        push_up(y) ;
```

```
    }
    inline void splay(int x) {///把一个点旋转到该点所在的 Splay 的根
        int y ; push_downAll(x) ;
        while(!isRoot(x)) {
            y = fa[x] ;
            if(!isRoot(y)) rotate((isSon(x,0)^isSon(y,0))?x:y) ;
            rotate(x) ;
        }
        push_up(x) ;
    }
    inline void access(int x) { ///把原树中根节点到 x 的链截出来变成一颗 Splay
        for(int y(0);x;x = fa[y = x]) {
            splay(x) ; int& R(ch[x][1]) ;
            if(y) headErase(x,y) ; ///注意,这里是(x,y)!!!
            if(R) heapInsert(x,R) ; ///注意,这里是(x,R)!!!
            R = y ; push_up(x) ;
        }
    }
    inline void setRoot(int x) ///把原树的根换成 x
        {access(x) ;splay(x) ;push_downRev(x) ;}
    inline int findRoot(int x) { ///找 x 所在的联通块在原树中的根
        access(x) ;splay(x) ;
        while(ch[x][0]) {push_down(x) ;x = ch[x][0];}
        splay(x) ;return x ;
    }
    inline bool isConnect(int x,int y) { /**判断 x 和 y 是否联通*/
        setRoot(x) ; return findRoot(y)==x ;
    }
    inline void split(int x,int y) {/**拉出 x<->y 的路径成为一个 Splay,y 是 Splay 的
根*/
        setRoot(x) ;access(y) ;splay(y) ;
    }
    inline bool link(int x,int y) {/**把 x 和 y 连上*/
        if(isConnect(x,y)) return false ;
        setRoot(y) ;fa[x] = y ;heapInsert(y,x) ;return true ;
    }
    inline bool cut(int x,int y) { ///把 x 和 y 断开
        if(!isConnect(x,y)||fa[y]!=x||ch[y][0]) return false ;
        fa[y] = ch[x][1] = 0 ; return true ;
    }
    inline int64_t Q(int id) { ///查询离 id 最远的距离
        access(id); splay(id); return MaxR[id]/** 最远点返回 MaxR,直径返回
MaxD.*/ ;
    }
```

```
}lct;
int n ;
struct Edge {int u,v;} edge[MAXN] ;
void Link(int id) {Edge &e(edge[id]) ;lct.link(e.u,n+id) ;lct.link(n+id,e.v) ;}
void Cut(int id) {Edge &e(edge[id]) ;lct.cut(e.u,n+id) ;lct.cut(n+id,e.v) ;}

int main()
{
    int m,id ; char op[5] ;
    int64_t val ;
    scanf("%d",&n) ;
    lct.clear(2*n+5) ;
    for(int i(1);i<n;++i) {
        Edge &e(edge[i]) ;
        scanf("%d%d%lld",&e.u,&e.v,&val) ;
        lct.setVal(n+i,val) ;
    }
    for(int i(1);i<n;++i) Link(i) ;
    scanf("%d",&m) ;
    while(m-->0) {
        scanf("%s%d",op,&id) ;
        if(op[0] == 'Q') {
            printf("%lld\n",lct.Q(id)) ;
        } else {
            Edge &e(edge[id]) ;
            scanf("%lld",&val) ;
            Cut(id); lct.setVal(n+id,val) ; Link(id) ;
        }
    }
    return 0;
}
```

(2) 查询所有覆盖某一个点的区间 O(nklogn)

```
/**
使用线段树维护数轴上的点是否被给定区间覆盖,并且输出覆盖它的区间是哪些.
注意:  输出的覆盖给定点的区间  是输出给定区间的编号.
        仅保证点在区间内部时可以输出所有区间,如果点在区间端点上,那么这些区间的编号可能被
    输出,也可能不被输出(这是由于离散化时没有去重的特殊性导致的).所以如果题意要求找出的
    区间包含点恰好在端点上的区间,那么应当在存储区间时将区间左右端点扩大 1.在输出区间后也应
    当判定每个区间是否满足要求.
```

题意:　操作1:给定二维平面上若干个圆形靶子的圆心坐标,这些靶子的底部一定与x轴接触.

操作 2:给定这个二维平面上的一个点,判断这个点是否打在前面已经给出的靶子上,如果是,

则输出靶子的编号.每个被打过的靶子会消失,未来也不会再被打到.

做法:　给定一个射击位置后,根据该射击位置的横坐标,找出所有可以盖住这个横坐标位置的所有靶子

满足题意的靶子的个数 m 不会大于 log4 底(坐标范围).然后 O(m)暴力判断哪个靶子被打中,然后输出靶

子编号并且删除靶子.

使用维护区间上任意区间的最大值,并且支持单点修改的线段树.将每个靶子可以盖住的 x 轴范围

抽象为一个区间.在线段树上每个区间的 left 的位置插入一个 pair(right,id).线段树维护的区间最大

值就是区间上所有点的 right 的最大值.(所以线段树初始化时应当将所有点的 right 值赋为-inf)

寻找合适的区间就是在 1 到 left 的区间内寻找 right 值大于给定点的 x 轴坐标的点.找到这样的点之后这个

点的 id 值就是答案.

*/

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 600005 ;///区间长度的最大值(离散化后)
const int inf = 0x3f3f3f3f ;
class ZKW_Tree {
public:
    ///单点修改,区间查询的 zkw 线段树
    pair<int,int> Tmax[MAXN<<2] ;
    int deep ;
    void build(int n) {
        deep = 1 ;
        while(deep<n) deep<<=1 ;
        fill_n(Tmax,MAXN<<2,make_pair(-inf,-inf)) ;
    }
    inline void push_up(int pos) {
        Tmax[pos] = max(Tmax[pos<<1],Tmax[pos<<1|1]);
    }
    inline void insert(int p,pair<int,int> val) {
        Tmax[deep+p] = val ;
        for(int i(deep+p>>1); i; i>>=1) push_up(i) ;
    }
    pair<int,int> QMax(int l,int r) {
        pair<int,int> ans(-inf,-inf) ;
        l += deep, --l ;
```

```
            r += deep, ++r ;
            while(l^r^1) {
                if(~l&1) ans = max(ans,Tmax[l^1]);
                if(r&1) ans = max(ans,Tmax[r^1]);
                l>>=1,r>>=1 ;
            }
            return ans ;
        }
} tree ;

struct point {
        int kind,x,y;///类型 1 表示靶子,类型 2 表示射击位置
        void input() {
            scanf("%d%d%d",&kind,&x,&y);
        }
} p[200005],lshp[200005];

vector<int> lsh;
unordered_map<int,int>lshmap;
vector<int>maybe;

void lookfor(int l,int r,int k){///在 left 到 right 区间内寻找 right 值(pair 的 first)大于 k 的叶子节点
        ///并且把这些节点的 id 值(pair 的 second)存入 maybe.
        pair<int,int> now = tree.QMax(l,r);///查询当前区间能到达的最大 right
        if(now.first<=k){///如果不存在能到达 k 右侧的点,那么整个区间不再考虑
            return;
        }
        if(l == r){///如果当前搜到叶子节点并且叶子节点可以到达 k 右侧,那么这个叶子节点的 id 值就是候选答案
            maybe.push_back(now.second);
        }else{///当前区间内存在候选答案但是不知道具体答案的位置
            int mid = (l+r)>>1;
            lookfor(l,mid,k);///用同样的方法将原区间一分为二分别搜索
            lookfor(mid+1,r,k);
        }
}
int check(int x1,int y1,int x2,int y2,int r){///判断找到的区间是否满足题意(根据题意自己写 check)
        return (x1*1LL-x2)*(x1-x2)+(y1*1LL-y2)*(y1-y2)<1LL*r*r;
}
int main() {
        int n;
        scanf("%d",&n);
```

```
    for(int i = 1; i<=n; ++i) {
        p[i].input();
        lshp[i] = p[i];
        if(p[i].kind == 1) {
            lsh.push_back(p[i].x - p[i].y);
            lsh.push_back(p[i].x + p[i].y);
        } else {
            lsh.push_back(p[i].x);
        }
    }
    stable_sort(lsh.begin(),lsh.end());///将所有线段的端点离散化
    tree.build(lsh.size()+2);
    ///离散化不去重,而是将值相同的点在数轴上相邻铺开
    ///这样做是因为存在起点相同的区间,而线段树单个起点只能对应单个终点.
    ///不去重的后果是原先值不相等的点离散化之后值的大小关系保证不变,而原先
///大小关系相同的点离散化之后大小不可能相同.所以无法保证点恰好在区间端点时
///可以找到区间.应当对查找后输出的结果进行检验.
    int tot = 0;
    for(auto it: lsh) {
        lshmap[it] = ++tot;
    }
    for(int i = 1; i<=n; ++i) {
        if(lshp[i].kind == 2){
            lshp[i].x                                                    =
lshmap[lshp[i].x]--

                                                                         ;
        }else{
            int x = lshp[i].x,y = lshp[i].y;
            lshp[i].x = lshmap[x-y]--;
            lshp[i].y = lshmap[x+y]--;
        }
    }
    for(int i = 1;i<=n;++i){
        if(lshp[i].kind == 1){
            tree.insert(lshp[i].x,make_pair(lshp[i].y,i));///插入区间的操作就是在线段树
上
            ///区间左端点的位置插入一个 pair(右端点,区间编号)
        }
        else{
            maybe.clear();
            lookfor(1,lshp[i].x,lshp[i].x);
            ///所有左端点在 1到lshp[i].x之间的区间,只要它的右端点也大于lshp[i].x,
那么就是要找的
```

```
                ///区间,调用 lookfor 函数在起点到待查找点之间分治查找
                bool isok = 0;
                for(auto it:maybe){
                    if(check(p[it].x,p[it].y,p[i].x,p[i].y,p[it].y)){
                        printf("%d\n",it);
                        tree.insert(lshp[it].x,make_pair(-inf,-inf));
                        isok = 1;
                        break;
                    }
                }
                if(isok == 0){
                    printf("-1\n");
                }
            }
        }
    }
```

(3) dfs 序+树状数组解决子树问题

```
#include <bits/stdc++.h>
#define in32(x) (scanf("%d",&x))
#define in64(x) (scanf("%lld",&x))
#define out32(x) (printf("%d",x))
#define out64(x) (printf("%lld",x))
#define en putchar('\n')
#define sp putchar(' ')
#define LOWBIT(x) (x&(-x))
using namespace std ;
const double eps (1e-8) ;
const int inf(0x3f3f3f3f) ;
const int MAXN(100010) ;
vector<vector<int> > Next(MAXN);
int in[MAXN],out[MAXN];
int sum[MAXN] ;
int tot = 0 ;
int n,m ;
bool have[MAXN] ;
/*子树的 dfs 序是连续的*/
inline void add(int p,int d=1)
{
    if(p == 0) return ;
    for(int i(p);i<MAXN;i+=LOWBIT(i))
        sum[i] += d ;
}
inline int Q(int p)
```

```
{
	if(p == 0) return 0 ;
	int ans(0) ;
	for(int i(p);i>0;i-=LOWBIT(i))
		ans += sum[i] ;
	return ans ;
}
void Init(int n)
{
	memset(sum,0,sizeof(sum)) ;
	for(int i(0);i<=n;++i) {
		Next[i].clear() ;
		have[i] = false ;
	}
	tot = 0 ;
}
void dfs(int pos=1,int fa=0)
{
	in[pos] = ++tot ;
	for(int i(0);i<Next[pos].size() ;++i) {
		int p(Next[pos][i]) ;
		if(p != fa)
			dfs(p,pos) ;
	}
	out[pos] = tot ;
}
int main()
{
	in32(n);
	{
		Init(n) ;
		for(int i(0);i<n-1;++i) {
			int a,b ;
			in32(a),in32(b) ;
			Next[a].push_back(b) ;
			Next[b].push_back(a) ;
		}
		dfs() ;
		char op[15] ;
		int o ;
		for(int i(1);i<=n;++i) {
			add(in[i],1) ;
			have[i] = 1;
		}
```

```
            in32(m) ;
            for(int i(0);i<m;++i) {
                scanf("%s%d",op,&o) ;
                if(op[0] == 'C') {
                    if(have[o])
                        add(in[o],-1) ;
                    else
                        add(in[o],1) ;
                    have[o]=!have[o];
                } else {
                    out32(Q(out[o])-Q(in[o]-1)) ;
                    en ;
                }
            }
        }
}
```

(4) FHQ_Treap(权值分裂_平衡树操作)

```
#include <iostream>
#include <stdio.h>
#include <math.h>
#include <algorithm>
#define in(x) (scanf("%d",&x))
#define out(x) printf("%d",x)
#define sp putchar(' ')
#define en putchar('\n')
using namespace std ;

const int inf(0x3f3f3f3f) ;

class FHQ_Treap
{
public:
    #define MAXN (200005)
    int root ;
    int L[MAXN],R[MAXN],Rand[MAXN],Size[MAXN],tot ;
    int64_t T[MAXN], lazy[MAXN] ;
    int memPool[MAXN],pTot ;
    FHQ_Treap () {clear() ;pTot = root = 0 ;}
    inline int random(){///高效随机函数
        static int seed(233); ///seed 可以随便取
        return seed=(int)(seed*48271LL%2147483647);
    }
    inline int newNode(int64_t val)
```

```
{
    int p(pTot?memPool[--pTot]:++tot) ;
    L[p] = R[p] = 0 ;Size[p] = 1 ;
    T[p] = val ;Rand[p] = random();
    return p ;
}
inline void push_up(int pos)
{
    Size[pos] = Size[L[pos]] + Size[R[pos]] + 1 ;
}
inline void push_down(int pos)
{
    ///balabalabala......
}
void split(int pos,int val,int &a,int &b){///注意引用!!
    ///pos 原 Treap，val 判定值,a 左树，b 右树，
    if(pos==0){
        a=b=0;return;///若 pos=0 分割完毕；
    }
    push_down(pos) ; ///和线段树一样,先 push_down;
    if(T[pos]<=val)///因为 pos 左子树中的所有值都小于 pos 的值，所以若 pos 属于左树，那么他们都属于左树递归 pos 的右子树；
        a=pos,split(R[pos],val,R[a],b);//a=pos 已经使 a 的右子树=pos 的右子树，不再递归 a 的右子树；
    else///同上 pos 右子树也都属于左树，递归左子树；
        b=pos,split(L[pos],val,a,L[b]);
    push_up(pos);///因为后面会用到左(右)树的 Size 所以更新维护
}
void merge(int &pos,int a,int b){
    ///pos 新树
    if(a==0||b==0){
        pos=a+b; return;///若某个树已空，则将另一个树整体插入
    }
    ///按照 Rand 值合并(堆性质)
    if(Rand[a]<Rand[b])///若 a 树 Rand 值<b 树，那么 b 树属于 a 树的后代，因为 b 树恒大于 a 树，那么 b 树一定属于 a 树的右后代，a 的左子树不变，直接赋给 pos，递归合并 a 的右子树和 b
        pos=a,push_down(pos),merge(R[pos],R[a],b);
    else
        pos=b,push_down(pos),merge(L[pos],a,L[b]);///同理，a 树一定是 b 树的左儿子，递归合并 b 的左儿子和 a
    push_up(pos);
}
inline void insert(int val)
{
    int a(0),b(0),c(0) ;
```

```
    c = newNode(val) ;
    split(root,val,a,b); ///将树分为两部分,a 树为<=待插入值,b 树大于
    merge(a,a,c); ///合并 a 树和新节点 z(树),赋给 a 树
    merge(root,a,b); ///合并新 a 树和 b 树，赋给根
}
inline void erase(int val)
{
    int a(0),b(0),c(0) ;
    split(root,val,a,b); ///分为 a 树为<=待删除，b 树大于
    split(a,val-1,a,c); ///a 树分为新 a 树<待删除，c 树等于待删除
    memPool[pTot++] = c ;
    merge(c,L[c],R[c]);///合并 c 树左右儿子,赋给 c 树,即丢弃 c 树根节点(实现删除)
    merge(a,a,c);
    merge(root,a,b); ///合并,不在重复
}
int find(int64_t val)
{
    int pos(root) ;
    while(pos&&T[pos] != val) {
        if(T[pos]<val) {
            pos = R[pos] ;
        } else {
            pos = L[pos] ;
        }
    }
    return pos ;
}
int Kth(int k,int pos){///找第 k 小(从 1 开始)的位置编号
    while(Size[L[pos]]+1!=k){
        if(Size[L[pos]]>=k)
            pos = L[pos];///若左子树大小大于 k,找左子树
        else
            k -= (Size[L[pos]]+1),pos=R[pos];///找右子树(k-左子树大小-树根(大小为 1))号的元素
    }
    return pos ;
}
int Rank(int64_t val)///查询排名
{
    int a(0),b(0),ans(0) ;
    split(root,val-1,a,b);///分为小于待查找值的 a 树和大于等于的 b 树
    ans = Size[a] + 1;
    merge(root,a,b) ;
    return ans ;
}
```

```
    int Pre(int64_t val) ///找前驱节点编号
    {
        int a(0),b(0),ans(0) ;
        split(root,val-1,a,b);///a 树为<=val-1 值即小于 val 值
        ans = Kth(Size[a],a);///在小于 val 值中找最大的(编号为 Size 的)就是前驱
        merge(root,a,b) ;
        return ans ;
    }
    int Nxt(int64_t val) ///找后继节点编号
    {
        int a(0),b(0),ans(0) ;
        split(root,val,a,b);///a 树为<=val 值
        ans = Kth(1,b);///在 b 树中找最小的,即为后继
        merge(root,a,b) ;
        return ans ;
    }
    inline void clear()
    {
        tot = pTot = 0 ;
        ///  建虚节点
        /**
        root = newNode(inf) ;
        Size[root] = 0 ;
        **/
    }
    #undef MAXN
};
FHQ_Treap treap ;
int main()
{
    int n,m,j,k ;
    scanf("%d",&m);
    for(int i(1);i<=m;++i){
        in(j),in(k) ;
        if(j==1) treap.insert(k);
        if(j==2) treap.erase(k);
        if(j==3) out(treap.Rank(k)),en ;
        if(j==4) out(treap.T[treap.Kth(k,treap.root)]),en ;
        if(j==5) out(treap.T[treap.Pre(k)]),en ;
        if(j==6) out(treap.T[treap.Nxt(k)]),en ;
    }
}
```

(5) FHQ_Treap(位置分裂_线段树,Splay 区间操作)

```
/**FHQ_Treap :[支持:单点插入,区间插入,单点删除,区间删除,单点更新,区间更新,区间查询,单点更新,单点查询,线段树的所有操作,Splay 的所有操作]**/
#include <bits/stdc++.h>
#define sp putchar(' ')
#define en putchar('\n')
using namespace std ;
const int inf(0x3f3f3f3f) ;
class FHQ_Treap
{
public:
    #define MAXN (100005)
    int root ;
    int L[MAXN],R[MAXN],Rand[MAXN],Size[MAXN],tot ;
    int64_t sum[MAXN],val[MAXN],lazy[MAXN] ;
     int memPool[MAXN],pTot ;
    FHQ_Treap () {clear() ;root = 0 ;}
    inline int random(){///高效随机函数
        static int seed(233); ///seed 可以随便取
        return seed=(int)(seed*48271LL%2147483647);
    }
    inline int newNode(int64_t _val)
    {
        int id(pTot?memPool[--pTot]:++tot) ;
        L[id] = R[id] = 0 ;Size[id] = 1 ;
        sum[id] = val[id] = _val;
        Rand[id] = random() ;
        return id ;
    }
    inline void erase(int l,int r)
    {
        int a(0),b(0),c(0) ;
        split(root,r,a,c) ; ///把 root:[1,n]分成 a:[1,r],:c:[r+1,n]
        split(a,l-1,a,b) ; ///  把 a:[1,r]分成 a:[1,l-1],b:[l,r]
        merge(root,a,c) ; ///  直接合并 a:[1,l-1],c[r+1,n] ;
    }
     inline void erase(int pos) ///删除 pos 位置的元素
     {
          int a(0),b(0),c(0) ;
          split(root,pos,a,c) ;///a:[1,pos],c[pos+1,n]
          split(a,pos-1,a,b) ; ///a:[1,pos-1],b:[pos,pos]
          memPool[pTot++] = b ;
          merge(root,a,c) ;
```

```
 }
inline void push_up(int pos) ///更新 pos 点信息
{
    if(pos == 0) return ;
    int l(L[pos]),r(R[pos]) ;
    Size[pos] = Size[l] + Size[r] + 1 ;
    sum[pos] = sum[l] + sum[r] + val[pos] ;
}
inline void push_down(int pos) ///下推标记
{
    ///balabalabala.....
    int l(L[pos]),r(R[pos]) ;
    int64_t d(lazy[pos]) ;
    if(d&&pos) {
        val[l] += d ; val[r] += d ;
        sum[l] += d*Size[l] ; sum[r] += d*Size[r] ;
        lazy[l] += d ; lazy[r] += d ;
        lazy[pos] = 0 ;
    }
}
void split(int pos,int k,int &a,int &b){///按照树的 size 分裂(注意引用!!)
    ///pos 原 Treap，k 判定值,a 左树，b 右树，
    if(pos==0){
        a=b=0;return;///若 pos=0 分割完毕；
    }
    push_down(pos) ; ///和线段树一样,先 push_down;
    if(k>Size[L[pos]])
        a=pos,split(R[pos],k-Size[L[pos]]-1,R[a],b);
    else
        b=pos,split(L[pos],k,a,L[b]);
    push_up(pos);///和线段树一样,后 push_up
}
void merge(int &pos,int a,int b){///按照 Rand 合并子树
    if(a==0||b==0){///pos 新树
        pos=a+b; return;///若某个树已空，则将另一个树整体插入
    }
    if(Rand[a]<Rand[b])///按照 Rand 值合并(堆性质)
        pos=a,push_down(pos),merge(R[pos],R[a],b);///注意 push_down 位置
    else
        pos=b,push_down(pos),merge(L[pos],a,L[b]);///同理，a 树一定是 b 树的左儿子，递归合并 b 的
左儿子和 a
    push_up(pos);///和线段树一样,后 push_up
}
inline int insert(int val) ///在树的尾部插入一个 val
```

```
{
    int c(0) ;
    c = newNode(val) ;
    merge(root,root,c); ///合并 root 树和新节点 c(树),赋给 root 树
    return c ;
}
 inline int insert(int l,vector<int64_t>&v) ///在[1,l]后边插入一个序列
 {
      int a(0),b(0),c(0),d(0) ;
      split(root,l,a,b) ;///分裂出 a:[1,l]
      for(int i(0);i<v.size();++i) {
            d = newNode(v[i]) ;
            merge(c,c,d) ; ///以 c 为根,建立新树
      }
      merge(a,a,c) ;
      merge(root,a,b) ;
 }
inline void change(int l,int r,int64_t d=1)///改变区间的值
{
    int a(0),b(0),c(0) ;
    split(root,r,a,c) ; ///把 root:[1,n]分成 a:[1,r],:c:[r+1,n]
    split(a,l-1,a,b) ; ///  把 a:[1,r]分成 a:[1,l-1],b:[l,r]
    ///以 b 为根的子树就是要操作的区间
    ///一堆操作,balabalabala...
    val[b] += d;
    sum[b] +=    d*Size[b];
    lazy[b] += d;
    merge(a,a,b) ;
    merge(root,a,c) ;
}
inline int64_t Q(int l,int r)///询问区间的值
{
    int a(0),b(0),c(0) ;
    int64_t ans(0) ;
    split(root,r,a,c) ;///把 root:[1,n]分成 a:[1,r],:c:[r+1,n]
    split(a,l-1,a,b) ;///  把 a:[1,r]分成 a:[1,l-1],b:[l,r]
    ///以 b 为根的子树就是要操作的区间
    ans = sum[b] ;
    merge(a,a,b) ;
    merge(root,a,c) ;
    return ans ;
}
inline void clear()
{tot = 0 ;}
```

```
    inline void dfs(int pos)
    {
     if(pos == 0) return ;
     dfs(L[pos]) ;
     cout<<val[pos]<<endl ;
     dfs(R[pos]) ;
    }
    #undef MAXN
};
FHQ_Treap treap ;
int main()
{
    int n,m,buff ;
    scanf("%d%d",&n,&m);
    for(int i(1);i<=n;++i) {
        scanf("%d",&buff) ;
        treap.insert(buff) ;
    }
    while(m-->0) {
        int op,l,r,d ;
        scanf("%d%d%d",&op,&l,&r) ;
        if(op == 1) {
            scanf("%d",&d) ;
            treap.change(l,r,d) ;
        } else {
            printf("%lld\n",treap.Q(l,r)) ;
        }
    }
}
```

(6) FHQ_Treap 区间翻转(平移)

```
/*区间平移就是分成两部分翻转*/
/**FHQ_Treap :[支持:单点插入,区间插入,单点删除,区间删除,单点更新,区间更新,区间查询,单点更新,单点查询,线段树的所有操作,Splay 的所有操作]**/
#include <bits/stdc++.h>
#define sp putchar(' ')
#define en putchar('\n')
using namespace std ;
const int inf(0x3f3f3f3f) ;
vector<int64_t> ans ;
class FHQ_Treap
{
public:
    #define MAXN (100005)
```

```
int root ;
int L[MAXN],R[MAXN],Rand[MAXN],Size[MAXN],tot ;
int64_t val[MAXN],lazy[MAXN] ;
FHQ_Treap () {clear() ;root = 0 ;}
inline int random(){///高效随机函数
    static int seed(233); ///seed 可以随便取
    return seed=(int)(seed*48271LL%2147483647);
}
inline int newNode(int64_t _val)
{
    int id(++tot) ;
    L[id] = R[id] = 0 ;Size[id] = 1 ;
    val[id] = _val;
    Rand[id] = random() ;
    return id ;
}
inline void push_up(int pos) ///更新 pos 点信息
{
    if(pos == 0) return ;
    int l(L[pos]),r(R[pos]) ;
    Size[pos] = Size[l] + Size[r] + 1 ;
}
inline void push_down(int pos) ///下推标记
{
    ///balabalabala.....
    int l(L[pos]),r(R[pos]) ;
    int64_t d(lazy[pos]) ;
    if(pos&&d) {
        lazy[l] ^= 1 ;
        lazy[r] ^= 1 ;
          swap(L[l],R[l]) ;
          swap(L[r],R[r]) ;
        lazy[pos] = 0 ;
     }
}
void split(int pos,int k,int &a,int &b){///按照树的 size 分裂(注意引用!!)
    ///pos 原 Treap，k 判定值,a 左树，b 右树，
    if(pos==0){
        a=b=0;return;///若 pos=0 分割完毕；
    }
    push_down(pos) ; ///和线段树一样,先 push_down;
    if(k>Size[L[pos]])
        a=pos,split(R[pos],k-Size[L[pos]]-1,R[a],b);
    else
```

```
            b=pos,split(L[pos],k,a,L[b]);
        push_up(pos);///和线段树一样,后 push_up
    }
    void merge(int &pos,int a,int b){///按照 Rand 合并子树
        if(a==0||b==0){///pos 新树
            pos=a+b; return;///若某个树已空，则将另一个树整体插入
        }
        if(Rand[a]<Rand[b])///按照 Rand 值合并(堆性质)
            pos=a,push_down(pos),merge(R[pos],R[a],b);///注意 push_down 位置
        else
            pos=b,push_down(pos),merge(L[pos],a,L[b]);///同理，a 树一定是 b 树的左儿子，递归合并 b 的
左儿子和 a
        push_up(pos);///和线段树一样,后 push_up
    }
    inline int insert(int val) ///在树的尾部插入一个 val
    {
        int c(0) ;
        c = newNode(val) ;
        merge(root,root,c); ///合并 root 树和新节点 c(树),赋给 root 树
        return c ;
    }

    inline void change(int l,int r)///改变区间的值
    {
        int a(0),b(0),c(0) ;
        split(root,r,a,c) ; ///把 root:[1,n]分成 a:[1,r],:c:[r+1,n]
        split(a,l-1,a,b) ; ///  把 a:[1,r]分成 a:[1,l-1],b:[l,r]
        ///以 b 为根的子树就是要操作的区间
        ///一堆操作,balabalabala...
          swap(L[b],R[b]) ;
          lazy[b] = 1 ;
        merge(a,a,b) ;
        merge(root,a,c) ;
    }
    inline void clear()
    {tot = 0 ;}
    inline void dfs(int pos)
    {
     if(pos == 0) return ;
     push_down(pos);
     dfs(L[pos]) ;
     ans.push_back(val[pos]) ;
     dfs(R[pos]) ;
    }
```

```
    #undef MAXN
};
FHQ_Treap treap ;
int main()
{
    int n,m,buff ;
    scanf("%d%d",&n,&m);
    for(int i(1);i<=n;++i) {
        treap.insert(i) ;
    }
    while(m-->0) {
        int l,r ;
        scanf("%d%d",&l,&r) ;
        treap.change(l,r) ;
    }
    treap.dfs(treap.root) ;
    for(int i(0);i<ans.size();++i) {
        printf("%d",ans[i]) ;
        if(i == ans.size()-1) en ; else sp ;
    }
}
```

(7) hash_map(string,int64_t)

```
struct pair_hash ///pair
{
    template<class T1, class T2>
    std::size_t operator() (const std::pair<T1, T2>& p) const
    {
        auto h1 = std::hash<T1>{}(p.first);
        auto h2 = std::hash<T2>{}(p.second);
        return h1 ^ h2;
    }
};
struct string_hash ///字符串
{
    size_t operator()( const std::string& x ) const
        {return std::hash<string>()(x);}
};
struct hash_longlong ///long long
    {size_t operator()(const long long &x) const {return x;}};
struct hash_int64_t ///long long
    {size_t operator()(const int64_t &x) const {return x;}};
```

unordered_map<pair<int,int>,int,pair_hash> mp ;
hash_map<pair<int,int>,int,pair_hash> mp ;

快速清空容器: std::unordered_map<int, int> tmp; t.swap(tmp);

pbds:
cc_hash_table<int, bool> h1; //拉链法处理冲突 /*稍微慢一些,内存消耗小*/
gp_hash_table<int, bool> h2; //探测法处理冲突 /*比较快,内存消耗大*/

(8) K-DTree(HDU4347)

```
#include<bits/stdc++.h>
#define in(x) (scanf("%d",&x))
/**K-D 树寻找离某一点 p 最近的 m 个点(p 的 m 近邻)*/
using namespace std;
const int MAXN=2e5+10;
namespace KDTree
{
    const int MAX_DIM(6);///最大维数
    int K;///维数
    inline double sqr(double x){return x*x;}
    struct Point
    {
        int x[MAX_DIM];
        double distance(const Point &oth)const
        {
            double ans(0);
            for(int i(0);i<K;i++)   ans+=sqr(x[i]-oth.x[i]);
            return ans;
        }
    };
    struct cmpx
    {
        int div;
        cmpx(const int &_div){div=_div;}
        bool operator()(const Point &a,const Point &b)
        {
            for(int i(0);i<K;i++) {
                int k=(div+i)%K;   if(a.x[k]!=b.x[k]) return a.x[k]<b.x[k];
            }
            return true;
        }
    };
    inline bool cmp(const Point &a,const Point &b,int div)
    {
```

```
    cmpx cp =cmpx(div);      return cp(a,b);
}

struct Node ///KDTree 的节点
{
    Point e;    Node *lc,*rc;    int div;
}pool[MAXN],*tail,*root;
void init(int k){tail=pool;K = k;} ///初始化 KDTree
Node* Build(Point *a,int l,int r,int div=0) ///建树
{
    if(l>=r) return NULL;
    Node *p=tail++;
    p->div=div;
    int mid((l+r)>>1);
    nth_element(a+l,a+mid,a+r,cmpx(div));
    p->e=a[mid];
    p->lc=Build(a,l,mid,(div+1)%K);
    p->rc=Build(a,mid+1,r,(div+1)%K);
    return p;
}

struct Qnode
{
    Point p; double dist;
    Qnode(){}
    Qnode(Point _p,double _dist){p=_p; dist=_dist;}
    bool operator <(const Qnode &oth)const{return dist<oth.dist;}
};
priority_queue<Qnode> Q;///注意:这里距离是从远到近存的!!!!!!
void search(const Point &p,Node *now,int div,int m) ///在 now 节点子树下搜索 p 点的 m 近邻
{
    if(now==NULL) return;
    if(cmp(p,now->e,div)){
        search(p,now->lc,(div+1)%K,m);
        if(Q.size()<m){
            Q.push(Qnode(now->e,p.distance(now->e)));
            search(p,now->rc,(div+1)%K,m);
        }else{
            if(p.distance(now->e) < Q.top().dist){
                Q.pop(); Q.push(Qnode(now->e,p.distance(now->e)));
            }
            if(sqr((now->e.x[div])-(p.x[div])) < Q.top().dist)
                search(p,now->rc,(div+1)%K,m);
        }
```

```
        } else {/**下边复制一下上边的,把 lc 与 rc 互换就可以了*/
            search(p,now->rc,(div+1)%K,m);/**这里是 rc*/
            if(Q.size()<m){
                Q.push(Qnode(now->e,p.distance(now->e)));
                search(p,now->lc,(div+1)%K,m);/**这里是 lc*/
            }else{
                if(p.distance(now->e) < Q.top().dist){
                    Q.pop();
                    Q.push(Qnode(now->e,p.distance(now->e)));
                }
                if(sqr((now->e.x[div])-(p.x[div])) < Q.top().dist)
                    search(p,now->lc,(div+1)%K,m);/**这里是 lc*/
            }
        }
    }
    void search(const Point &p,int m) ///搜索 p 点的 m 近邻
    {
        while(!Q.empty()) Q.pop();
        search(p,root,0,m);
    }
};

int n,m;
KDTree::Point p[MAXN];

int main()
{
    while(~in(n))
    {
        in(m) ;
        for(int i=1;i<=n;i++) {
            for(int j(0);j<m;++j)
                in(p[i].x[j]) ;
        }
        KDTree::init(m);
        KDTree::root=KDTree::Build(p,1,n+1);
        KDTree::Point o;    int t ; in(t) ;
        for(int i=1;i<=t;++i){
            for(int j(0);j<m;++j)
                in(o.x[j]) ;
            int cnt ;    in(cnt) ;
            printf("the closest %d points are:\n",cnt) ;
            KDTree::search(o,cnt);
            stack<KDTree::Qnode> sta ; ///优先队列是从远到近的,用一个栈倒过来
```

```
            while(KDTree::Q.size()) {
                sta.push(KDTree::Q.top()) ;
                KDTree::Q.pop() ;
            }
            while(sta.size()) {
                o=sta.top().p;    sta.pop() ;
                for(int j(0);j<m;++j) {
                    printf("%d",o.x[j]) ;    putchar(j == m-1?'\n':' ') ;
                }
            }
        }
    }
}
```

(9) K-DTree_m 动态重构完整模板

```
/**
        完整的动态 KDTree 模板.
    KDTree 实现维护矩形方格里的点的权值.以及查询矩形内有多少个数,以及 K 近邻.
*/
/**利用类似替罪羊树的重构实现,时间复杂度 O(nlogn)~O(nsqrt(n))*/
#include<bits/stdc++.h>
#define int64_t int
using namespace std;
const int MAXN = 500005;

namespace KDTree
{
    const double Alpha = 0.75;///平衡因子
    const int MAX_DIM(6);///最大维数
    int K;///维数
    #define sqr(x) ((x)*(x))
    struct Point
    {
        int x[MAX_DIM]; int64_t val ; ///当前节点的权值
        double getDis(const Point &o)const
        {
            double ans(0);
            for(int i(0);i<K;i++)
                ans+=sqr(x[i]-o.x[i]);
            return ans;
        }
    };
    struct cmpx
    {///完成比较的结构体
```

```
    int div;
    cmpx(const int &_div){div=_div;}
    bool operator()(const Point &a,const Point &b)
    {
        for(int i(0);i<K;i++) {
            int k((div+i)%K);
            if(a.x[k]!=b.x[k]) return a.x[k]<b.x[k];
        }
        return true;
    }
};
struct Qnode
{///找 K 近邻用的,不找 K 近邻可以不用写这个.
    Point p;
    double dis;
    Qnode(){}
    Qnode(Point _p,double d){p=_p; dis=d;}
    bool operator <(const Qnode &o)const{return dis<o.dis;}
};
priority_queue<Qnode> Q;///注意:这里距离是从远到近存的!!!!!!
inline bool cmp(const Point &a,const Point &b,int div)
{///比较某一维度
    cmpx cp =cmpx(div);
    return cp(a,b);
}
struct Node ///KDTree 的节点
{
    Point e; ///点
    int64_t Sum ; ///子树权值之和
    Node *lc,*rc; ///左右孩子
    int Min[MAX_DIM],Max[MAX_DIM];///该点能管理的范围
    int div/**维度*/,Size/**有效的子树大小*/,nodeCnt/**所有节点个数*/;
    int Cnt;/**重复点的个数,等于 0 代表点不存在*/
    inline bool isBad(){ ///判断是否需要重构子树
        return lc->nodeCnt>Alpha*nodeCnt+5||
                rc->nodeCnt>Alpha*nodeCnt+5;
    }
    inline void push_up()
    {///类似线段树统计子树信息
        Size = Cnt + lc->Size + rc->Size ;
        nodeCnt = 1+lc->nodeCnt+rc->nodeCnt ;
        Sum = lc->Sum + rc ->Sum + (Cnt>0)*e.val ;
        for (int i(0);i<K;++i) {///更新边界信息.
            Max[i]=max(lc->Max[i],Max[i]);Min[i]=min(lc->Min[i],Min[i]);
```

```
                Max[i]=max(rc->Max[i],Max[i]);Min[i]=min(rc->Min[i],Min[i]);
            }
        }
}pool[MAXN]/**内存池*/,*tail,*root/**根节点*/,*null;
Node *bc[MAXN] ;/**内存回收池*/int bc_top,tot;/**内存回收池栈顶*/
Point buff[MAXN] ;

inline Node* newNode(Point &e)
{///新建一个节点
    Node *p ;
    p = bc_top?bc[--bc_top]:tail++;
    p->e = e;
    p->lc = p->rc = null ;
    p -> Sum = p->e.val = e.val ;
    p->Cnt = p->Size = p->nodeCnt = 1 ;
    for(int i(0);i<K;++i)
        p->Min[i] = p->Max[i] = e.x[i];
    return p ;
}

Node* Build(Point* e,int l,int r,int div=0)
{///以[l,r)建树.!!!!!!!!!注意!这里右区间是开的!!!
    if(l >= r) return null;
    int mid((l+r)>>1);
    nth_element(e+l,e+mid,e+r,cmpx(div));
    Node *p = newNode(e[mid]) ;
    p->lc = Build(e,l,mid,(div+1)%K) ;
    p->rc = Build(e,mid+1,r,(div+1)%K) ;
    p->div = div ; p->push_up() ;
    return p;
}

void Travel(Node *p)
{///暴力拍扁 p 为根的子树
    if(p == null) return ;
    Travel(p->lc) ;
    if(p->Cnt) buff[tot++] = p->e ;
    bc[bc_top++] = p ;
    Travel(p->rc) ;
    p->Cnt = 0;
}

inline void rebuild(Node*&p)
{///暴力重新建 p 为根的子树
```

```
    tot = 0 ;    Travel(p) ;
    p = Build(buff,0,tot,p->div) ;
}

Node **insert(Node *&p,Point e,int div)
{///在 p 为根的子树下插入一个点 e
    if(p == null) {
        p = newNode(e) ;p -> div = div ;
        return &null ;
    } else if(p->e.x[0] == e.x[0]&&p->e.x[1] == e.x[1]){
        p->e.val += e.val ; ++ p->Cnt ; p -> push_up() ;
        return &null ;
    } else {
        Node **res ;
        if(cmp(e,p->e,div)) res = insert(p->lc,e,(div+1)%K) ;
        else res = insert(p->rc,e,(div+1)%K) ;
        p->push_up() ;
        if(p->isBad()) res = &p ;
        return res ;
    }
}



inline int isIn(int x1,int y1,int x2,int y2,int X1,int Y1,int X2,int Y2)
{///判断矩形{<X1,Y1>,<X2,Y2>}是否包含{<x1,y1>,<x2,y2>}.
    return X1<=x1&&Y1<=y1&&X2>=x2&&Y2>=y2;
}

inline int isOut(int x1,int y1,int x2,int y2,int X1,int Y1,int X2,int Y2)
{///判断矩形{<X1,Y1>,<X2,Y2>}是否与{<x1,y1>,<x2,y2>}相离.
     return X1>x2||X2<x1||Y1>y2||Y2<y1;
}

int64_t search(int X1,int Y1,int X2,int Y2,bool isQSize ,Node *now)
{///查询 now 为根的子树表示的矩形在{<X1,Y1>,<X2,Y2>}内权值或者个数.
    if(now == null) return 0;
    int x1(now->Min[0]),y1(now->Min[1]),x2(now->Max[0]),y2(now->Max[1]);
    ///printf("(%d,%d):{<%d,%d>,<%d,%d>]\n",now->e.x[0],now->e.x[1],x1,y1,x2,y2) ;
    if(isIn(x1,y1,x2,y2,X1,Y1,X2,Y2)) return isQSize?now->Size:now->Sum ;
    if(isOut(x1,y1,x2,y2,X1,Y1,X2,Y2)) return 0 ;
    x1 = now->e.x[0];y1 = now->e.x[1] ;
    int64_t ret (0) ;
    if(isIn(x1,y1,x1,y1,X1,Y1,X2,Y2)) ret = isQSize?now->Cnt:((now->Cnt>0)*now->e.val) ;
```

```
        ret += search(X1,Y1,X2,Y2,isQSize,now->lc)+search(X1,Y1,X2,Y2,isQSize,now->rc)    ;
        return ret ;
    }

    void search(const Point &p,Node *now,int div,int k)
    {///在 now 节点子树下搜索 p 点的 m 近邻
        if(now==null) return;
        Node *lc(now->lc),*rc(now->rc) ;Point& e = now->e ;
        if(cmp(p,e,div)) swap(lc,rc) ;
        search(p,rc,(div+1)%K,k);
        if((int)Q.size()<k){
            Q.push(Qnode(e,p.getDis(e)));
            search(p,lc,(div+1)%K,k);
        }else{
            if(p.getDis(e) < Q.top().dis){
                Q.pop(); Q.push(Qnode(e,p.getDis(e)));
            }
            if(sqr((e.x[div])-(p.x[div])) < Q.top().dis)
                search(p,lc,(div+1)%K,k);
        }
    }
/*************************下边是外部接口*****************************/
    inline void Build(Point* e,int len)
    {///以开始位置为 0 的长度为 len 的数组 e 建立 KDTree
        root = Build(e,0,len) ;
    }

    inline void Init(int k){ ///初始化 KDTree,参数 k 是维度.
        tail=pool; K = k; null = tail++ ;
        null->lc = null->rc = null ;
        null->Size = null->nodeCnt = 0 ;
        root = null ; null->Cnt = bc_top = 0 ;
        for(int i(0);i<K;++i) {
            null -> Min[i] = 0x3f3f3f3f ;
            null -> Max[i] = -0x3f3f3f3f ;
        }
    }

    inline void insert(Point &e)
    {///插入新点以及新点的权值.
        Node **p = insert(root,e,0) ;
        if(*p != null) rebuild(*p) ;
    }
```

```
    void search(const Point &p,int m)
    {///搜索 p 点的 m 近邻
        while(!Q.empty()) Q.pop();
        search(p,root,0,m);
    }

    inline int64_t search(int X1,int Y1,int X2,int Y2,bool isQSize = true)
    {///查询矩形{<X1,Y1>,<X2,Y2>}内点的个数或者权值.
        return search(X1,Y1,X2,Y2,isQSize,root) ;
    }
};

int n,m;
KDTree::Point p[MAXN];

#define getcharEx getchar
template <class T>
inline void read(T &x) {
    static char ch;static bool f;
    for(ch=f=0;ch<'0' || '9'<ch;f|=ch=='-',ch=getcharEx());
    for(x=0;isdigit(ch);x=((x+(x<<2))<<1)+ch-'0',ch=getcharEx());
    x=f?-x:x;
}

template <class T>
inline void write(T x)
{
    if (x < 0) putchar('-'),x = -x;
    if (x >= 10) write (x / 10);
    putchar(x % 10 + '0');
}


int main()
{///BZOJ4066: 矩形内的权值之和
    int64_t last_ans(0);
    KDTree::Init(2);
    read(n) ;
    int op,x1,y1,x2,y2 ;int64_t val ;
    while("Orz")
    {
        read(op) ;
        if(op == 3) break ;
        read(x1); read(y1) ;
```

```
            x1^= last_ans ;y1 ^= last_ans ;
            if(op == 1) {
                read(val) ;    val ^= last_ans ;
                KDTree::Point tmp ; tmp.x[0] = x1;tmp.x[1] = y1 ;
                tmp.val = val ;    KDTree::insert(tmp) ;
            } else if(op == 2){
                read(x2) ;read(y2) ;    x2 ^= last_ans ;y2 ^= last_ans ;
                write(last_ans=KDTree::search(x1,y1,x2,y2,false));putchar('\n') ;
            }
        }
        return 0 ;
}
/**
int main()
{///HDU4347: 一个点的 K 近邻
        while(~scanf("%d",&n))
        {
            read(m) ;
            for(int i=0;i<n;i++) {
                for(int j(0);j<m;++j)
                    read(p[i].x[j]) ;
            }
            KDTree::Init(m);    KDTree::Build(p,n);    KDTree::Point o;
            int t ;
            read(t) ;
            for(int i=1;i<=t;++i){
                for(int j(0);j<m;++j)
                    read(o.x[j]) ;
                int cnt ; read(cnt) ;
                printf("the closest %d points are:\n",cnt) ;
                KDTree::search(o,cnt);
                stack<KDTree::Qnode> sta ; ///优先队列是从远到近的,用一个栈倒过来
                while(KDTree::Q.size()) {
                    sta.push(KDTree::Q.top()) ;
                    KDTree::Q.pop() ;
                }
                while(sta.size()) {
                     o=sta.top().p;      sta.pop() ;
                     for(int j(0);j<m;++j) {
                         write(o.x[j]) ;    putchar(j == m-1?'\n':' ') ;
                     }
                }
            }
        }
```

```
}
*/
```

(10)普通 ST 表 O(nlogn)

```
class ST
{
public:
    int mm[MAXN] ;
    int64_t dp[20][MAXN];
    void Init(int64_t* arr,int n)
    {///下标从 1 开始,传参数的时候 arr 需要注意一下!!!
        ///mm[0] = -1;
        for(int i(1);i<=n;++i) {
            mm[i] = __lg(i) ;
            ///mm[i] = mm[i-1]+((i&(i-1))?0:1);///这样写也行
            dp[0][i] = arr[i] ;
        }
        for(int i(1);i<=mm[n];++i) {
            for(int j(1);j+(1<<i)-1<=n;++j) {
                dp[i][j] = max(dp[i-1][j],dp[i-1][j+(1<<(i-1))]) ;
            }
        }
    }
    int64_t Q(int l,int r)
        {int k(mm[r-l+1]) ;    return max(dp[k][l],dp[k][r-(1<<k)+1]) ;}
};
ST st ;
```

(11)时空优化 ST 表[log 以 3 为底]O(nlogn)

```
class ST
{
public:
    int dpMax[15][MAXN],dpMin[15][MAXN];
    int pw3[20],lg3[MAXN];
    void Init(int* arr,int n)
    {///下标从 1 开始,传参数的时候 arr 需要注意一下!!!
        pw3[0] = 1 ;
        for(int i(1);i<=n;++i) {
            lg3[i] = (int)(log(i)/log(3)) ;
            dpMax[0][i] = dpMin[0][i] = arr[i] ;
            if(i<=15) pw3[i] = pw3[i-1]*3 ;
        }
        for(int i(1);i<=lg3[n];++i) {
```

```
                for(int j(1);j+pw3[i]-1<=n;++j) {
                    dpMax[i][j] =
max({dpMax[i-1][j],dpMax[i-1][j+(pw3[i-1])],dpMax[i-1][j+2*(pw3[i-1])]}) ;
                    dpMin[i][j] =
min({dpMin[i-1][j],dpMin[i-1][j+(pw3[i-1])],dpMin[i-1][j+2*(pw3[i-1])]}) ;
                }
            }
        }
        int QMax(int l,int r)
        {
            int k(lg3[r-l+1]),d(r-2*pw3[k]+1)   ;
            int ans(max(dpMax[k][l],dpMax[k][r-pw3[k]+1])) ;
            return d<l?ans:max(ans,dpMax[k][d]);
        }
        int QMin(int l,int r)
        {
            int k(lg3[r-l+1]),d(r-2*pw3[k]+1) ;
            int ans(min(dpMin[k][l],dpMin[k][r-pw3[k]+1])) ;
            return d<l?ans:min(ans,dpMin[k][d]);
        }
};
ST st ;
```

(12)zkw 极值线段树 O(logn)

```
///单点修改,区间查询的 zkw 线段树
class ZKW_Tree
{
public:
    ///单点修改,区间查询的 zkw 线段树
    int64_t Tmax[MAXN<<2],deep ;

    void build(int n)
    {
        deep = 1 ;
        while(deep<n) deep<<=1 ;
        memset(Tmax,0,sizeof(Tmax)) ;
    }

    inline void push_up(int pos)
    {
        Tmax[pos] = max(Tmax[pos<<1],Tmax[pos<<1|1]);
    }

    inline void change(int p,int val)
```

```
    {
        Tmax[deep+p] = val ;
        for(int i(deep+p>>1);i;i>>=1) push_up(i) ;
    }

    int64_t QMax(int l,int r)
    {
        int ans(0) ;
        l += deep , --l ;
        r += deep , ++r ;
        while(l^r^1) {
            if(~l&1) ans = max(ans,Tmax[l^1]);
            if(r&1) ans = max(ans,Tmax[r^1]);
            l>>=1,r>>=1 ;
        }
        return ans ;
    }
};
ZKW_Tree tree ;
```

(13)包含某一点的最大(小)连续和

对于序列 arr[1~n],求前缀和 Sum[0~n],初始化 Sum[0]=0,把 Sum[0~n]插入到线段树中(包含 Sum[0],或者开一个 ST 表).
则对于 i 点来说,在区间[l,r]中能够包含 i 点的最大子区间和求法为:

```
int64_t sL,sR ;
sL = tree.QMin(l-1,i-1) ;
sR = tree.QMax(i,r) ;
```

sR-sL 就是包含 i 点的在[l,r]范围内的最大子区间和.

```
sL = tree.QMax(l-1,i-1) ;
sR = tree.QMin(i,r) ;
```

sR-sL 就是包含 i 点的在[l,r]范围内的最小子区间和.
正负都通用.

(14)普通并查集

```
class Union_Find_Set
{
static const int N = 2000005;
public:
    int tot/**现有的集合数量*/,cnt[N]/**每个集合有多少人*/,fa[N] ;
    void clear(int n)
    {
```

```
        tot = n; ///开始没有合并的时候,每个都是单独的集合.
        iota(fa,fa+n+1,0) ; fill_n(cnt,n+1,1) ;
    }
    int find(int x) ///找 x 的根
        {return x == fa[x]?x:fa[x] = find(fa[x]) ;}
    int count(int x)///得到元素 x 所在的集合的人数
        {return cnt[find(x)] ;}
    int size() {return tot ;}
    bool is_same(int a,int b)///判断是不是在同一个集合
        {return find(a)==find(b);}
    int merge(int a,int b)
    {///合并两个集合
        a = find(a);b = find(b) ;
        if(a != b) {
            fa[b] = fa[a] ; cnt[a] += cnt[b];
            -- tot ;
        }
        return tot ;
    }
};
```

(15) 笛卡尔树解决区间最值查询 O(n)

```
/**注意!!!!!只能在数据随机的情况下用!!!!!!!!*/
class Cartesian_Tree
{///笛卡尔树求区间最值 O(n)建树,O(1)查询.
static const int N = 2e7+10;
public :
    int val[N],lc[N],rc[N],Stack[N],root,top,*cur;
    void build(int* arr,int n,bool minTree = false)
    {/// arr 输入下标从 1 开始,查询的时候也从 1 开始.当 minTree 为真的时候,
     ///  笛卡尔树是以最小值建树的,为假时以最大值查询建树的.
        top = 0 ;cur = Stack ;
        for(register int i=1;i<=n;i++){
            if(arr) val[i] = arr[i] ;
            ///下边改成 val[i]<=val[*cur]维护的就是最小值,反之是最大值.
            while(cur!=Stack&&(minTree?val[i]<=val[*cur]:val[i]>=val[*cur]))
lc[i]=*cur--;
            rc[*cur]=i;*++cur=i;
        }
        root=Stack[1];
    }
    inline int Q(int l,int r){
        for(int x(root);;x=(x>r?lc:rc)[x]) if(l<=x&&x<=r) return val[x];
```

```
    }
}tree;
```

(16) 动态开点线段树

```
class DSTree
{
static const int MAX = 1000000005;
static const int MIN = 0;
#define push_down() \
    int mid((LD+RD)>>1),L(T[pos].L),R(T[pos].R),lazy(T[pos].lazy),id ;\
    if(!L) {id = newNode();L = T[pos].L = id ;}\
    if(!R) {id = newNode() ;R = T[pos].R = id;}\
    if(lazy) {\
        T[L].lazy += lazy ;\
        T[R].lazy += lazy ;\
        T[L].Sum += 1LL*lazy*(mid-LD+1) ;\
        T[R].Sum += 1LL*lazy*(RD-(mid+1)+1) ;\
        T[pos].lazy = 0 ;\
    }
public:
    struct Node {int L,R,lazy;int64_t Sum;};
    vector<Node> T ;int tot;

    int newNode()
        {T.push_back(Node{0,0,0,0}) ;return T.size()-1;}

    void clear()
        {vector<Node>buff;T.swap(buff);newNode() ;newNode() ;}

    void push_up(int pos)
    {
        int L(T[pos].L),R(T[pos].R) ;
        T[pos].Sum = T[L].Sum + T[R].Sum ;
    }

    void change(int l,int r,int64_t val,int LD=MIN,int RD=MAX,int pos = 1)
    {
        if(LD == l&&RD ==r) {
            T[pos].lazy += val;
            T[pos].Sum += 1LL*(r-l+1)*val ;
            return ;
        }
```

```
        push_down() ;
        if(r<=mid) change(l,r,val,LD,mid,L) ;
        else if(l>mid) change(l,r,val,mid+1,RD,R) ;
        else {
            change(l,mid,val,LD,mid,L);
            change(mid+1,r,val,mid+1,RD,R);
        }
        push_up(pos) ;
    }

    int64_t Q(int l,int r,int LD=MIN,int RD=MAX,int pos = 1)
    {
        if(pos == 0) return 0 ;
        if(LD == l&&RD ==r) return T[pos].Sum;
        push_down() ;
        if(r<=mid) return Q(l,r,LD,mid,L) ;
        else if(l>mid) return Q(l,r,mid+1,RD,R) ;
        else return Q(l,mid,LD,mid,L) + Q(mid+1,r,mid+1,RD,R);
    }

    int Kth(int64_t k,int LD=MIN,int RD=MAX,int pos = 1)
    {
        if(LD == RD) return LD ;
        push_down() ;
        int64_t LSz(T[L].Sum) ;
        if(LSz<k) return Kth(k-LSz,mid+1,RD,R) ;
        else return Kth(k,LD,mid,L) ;
    }
}tree;
```

(17) 珂朵莉树 O(nlog^2n)

```
/**
    珂朵莉树的核心操作在于推平一个区间,它的高效建立在数据随机的前提下.
    珂朵莉树使用条件是数据随机!!!!!!!!!构造的数据很容易卡死珂朵莉数
    1.能够快速解决询问区间 N 次方和的问题
    2.能够完成线段树的绝大部分操作(比如区间和啥的)
    3.在数据随机并带有区间赋值的情况下用它来 A 题
```

当发现题目里面有一个操作是推平一整段区间,并且数据是随机的话,我们可以用珂朵莉树.

对于推平操作,我们让每一个节点维护一个区间,然后对于 2 号操作清空区间[l,r]里的所有区间,用一个大区间[l,r]取代他们.

对于其他操作操作(比如求和啥的),我们暴力地找到每一个[l,r]里面的区间,然后对它们各个进行操作.

珂朵莉树的另一个应用就是用在对区间操作的次数比较小的时候,
由于每次操作最多增加两个区间,那么如果有 m 次操作,set 里最多有 2*(m+1)个节点
所以访问的时间复杂度为 O(m^2log(m)).
*/
```
struct Node
{
    int l,r;mutable int64_t val ;
    Node(int _l,int _r=-1,int64_t _val=0):l(_l),r(_r),val(_val){}
    bool operator < (const Node& b) const{return l<b.l ;}
};

class Chtholly_Tree
{
public:
    int64_t seed ; ///随机数种子
    Chtholly_Tree (int n=0){if(n) clear(n);}
    set<Node> s ;
    inline void clear(int n) {s.clear() ;s.insert(Node(n+1,n+1,0));/*!!别忘了插入!!!!!*/}
    inline void insert(int l,int r,int64_t val=0)///插入一段区间[l,r] -> val
        {s.insert(Node(l,r,val)) ;}

    inline auto spilt(int mid) -> decltype (s.begin())
    { ///找出左端点为 mid 的区间.如果不存在,
      ///则把包含 mid 的区间拆分成[l,mid-1]和[mid,r].O(logn)
        auto it = s.lower_bound(Node(mid)) ;
        if(it!=s.end()&&it->l == mid) return it ;
        -- it ;
        int l(it->l),r(it->r);
        int64_t val(it->val);
        s.erase(it) ;s.insert(Node(l,mid-1,val)) ;
        return s.insert(Node(mid,r,val)).first ;
    }
    inline void assign(int l,int r,int64_t val=0)
    {///推平一个区间,即[l,r] = val .O(logn)
        auto itR(spilt(r+1)),itL(spilt(l)) ;///注意!!!先 R 后 L!!!不然会 RE!!!
        s.erase(itL,itR) ;s.insert(Node(l,r,val)) ;
    }

    inline int64_t Operate(int l,int r,int id,int64_t val=0)
    {///对区间的操作[包括询问,修改等等,直接暴力] O(logn)
        int64_t ans(0) ;
        auto itR(spilt(r+1)),itL(spilt(l)) ;///注意!!!先 R 后 L!!!不然会 RE!!!
        for(;itL!=itR;++itL) {///直接暴力每一个值相等的区间
```

```
                switch (id)
                {
                    case 1:
                        itL->val += val;break; /*区间加上一个数*/
                    case 4:
                        {}///balabalabala.....
                }
            }
            return ans ;
        }

        inline int64_t rand()
        {///  生成一个随机数
            int64_t ret(seed) ;
            seed = (1LL*seed * 7 + 13) % 1000000007 ;
            return ret ;
        }

        int64_t Kth(int l,int r,int k)
        {///区间第 K 大
            vector<pair<int64_t,int> > v ;
            auto itR(spilt(r+1)),itL(spilt(l)) ;///注意!!!先 R 后 L!!!不然会 RE!!!
            for(;itL!=itR;++itL)
                v.push_back({itL->val,itL->r-itL->l+1}) ;
            sort(v.begin(),v.end()) ;
            if(v.size())
            for(auto& it:v) {
                k -= it.second ;
                if(k<=0) return it.first ;
            }
            return -1LL ;
        }
};
Chtholly_Tree tree ;
```

(18) 树链合并 O(nlogn + k)

```
/**
    O(nlogn)预处理,O(1)求 LCA + O(1)合并两条树链.
*/
#include <bits/stdc++.h>
using namespace std ;
const int MAXN = 500005 ;
vector<pair<int,int> > Next[MAXN] ;
```

```
int id[MAXN<<1],st[MAXN],pos[MAXN],deep[MAXN],tot ;
int dfs_st[MAXN],dfs_ed[MAXN],tot_dfs ;

inline void addedge(int u,int v,int val = 0)
{
	Next[u].push_back({v,val}) ;Next[v].push_back({u,val}) ;
}

void dfs(int pos = 1,int _deep = 1,int _fa = -1)
{
	id[++tot] = pos ; st[pos] = tot ;
	dfs_st[pos] = ++tot_dfs ;
	deep[pos] = _deep ;
	int sz(Next[pos].size()) ;
	for(register int i(0);i<sz;++i) {
		pair<int,int>&e = Next[pos][i];
		if(e.first == _fa) continue ;
		dfs(e.first,_deep+1,pos) ;
		id[++tot] = pos ;
	}
	dfs_ed[pos] = ++tot_dfs ;
}

template<typename T>
inline T min_d(const initializer_list<T>& x)
{
	T ret = *x.begin() ;
	for(auto& it:x) if(deep[it]<deep[ret]) ret = it ;
	return ret ;
}

template<typename T>
inline T max_d(const initializer_list<T>& x)
{
	T ret = *x.begin() ;
	for(auto& it:x) if(deep[it]>deep[ret]) ret = it ;
	return ret ;
}

class ST
{
public:
	int dpMin[15][MAXN<<1];
	int pw3[15],lg3[MAXN<<1];
```

```
    void Init(int* arr,int n)
    {///下标从 1 开始,传参数的时候 arr 需要注意一下!!!
        pw3[0] = 1 ;
        for(register int i(1);i<=n;++i) {
            lg3[i] = (int)(log(i)/log(3)) ;
            dpMin[0][i] = arr[i] ;
            if(i<=15) pw3[i] = pw3[i-1]*3 ;
        }
        for(register int i(1);i<=lg3[n];++i)
            for(register int j(1);j+pw3[i]-1<=n;++j)
                dpMin[i][j]                                                    =
min_d({dpMin[i-1][j],dpMin[i-1][j+(pw3[i-1])],dpMin[i-1][j+2*(pw3[i-1])]}) ;
    }
    int QMin(int l,int r)
    {
        int k(lg3[r-l+1]),d(r-2*pw3[k]+1) ;
        int ans(min_d({dpMin[k][l],dpMin[k][r-pw3[k]+1]})) ;
        return d<l?ans:min_d({ans,dpMin[k][d]});
    }
}St;

void InitLCA(int n,int root = 1) {tot = 0 ;dfs(root) ;St.Init(id,tot) ;}
void Init(int n) {tot_dfs = 0 ;for(int i(0);i<=n;++i) Next[i].clear() ;}
inline int LCA(int u,int v) {///得到以初始为根的 LCA
    int a(min(st[u],st[v])),b(max(st[u],st[v])) ;
    return St.QMin(a,b) ;
}
bool have(int u,int v) {
    return dfs_st[u]<=dfs_st[v]&&dfs_ed[v]<=dfs_ed[u] ;
}
class Tree_Chain
{
public :
    int u,v,lca ;
    Tree_Chain(int u=0,int v=0,int lca=-1):u(u),v(v),lca(lca) {}
    bool check(int e) {
        if(!lca) return 1 ;
        return (lca<0)?0:(have(lca,e)&&(have(e,u)||have(e,v))) ;
    }
    void read() {scanf("%d%d",&u,&v) ; lca = LCA(u,v) ;}
    int solve() { ///计算当前链的结果
        if(lca<=0) return 0 ;
        return deep[u] + deep[v] - 2*deep[LCA(u,v)] +1 ;
    }
```

```
};
Tree_Chain val[MAXN] ;
Tree_Chain merge(Tree_Chain a,Tree_Chain b) {///合并两条链,返回两条链的交
    int u,v,&x(a.lca),&y(b.lca) ;
    if(!x) return b ;   if(!y) return a ;
    if(x<0||y<0) return Tree_Chain (0,0,-1) ;
    if(deep[x]<deep[y]) swap(a,b);
    if(!b.check(x)) return Tree_Chain(0,0,-1) ;
    int uu(LCA(a.u,b.u)),uv(LCA(a.u,b.v)) ,
        vu(LCA(a.v,b.u)),vv(LCA(a.v,b.v)) ;
    u = max_d({uu,uv}) ; v = max_d({vu,vv}) ;
    return Tree_Chain(u,v,LCA(x,y)) ;
}

int main()
{
    int n,m,q,u,v,T,t=0   ;
    scanf("%d",&T) ;
    while(T-->0)
    {
        printf("Case %d:\n",++t) ;
        scanf("%d",&n) ; Init(n) ;
        for(int i(1);i<n;++i) {
            scanf("%d%d",&u,&v) ; addedge(u,v) ;
        }
        InitLCA(n,1) ;
        scanf("%d",&q) ;
        while(q-->0) {
            scanf("%d",&m) ;
            for(int i(1);i<=m;++i) {
                val[i].read() ;
                if(i!=1) val[1] = merge(val[1],val[i]) ;
            }
            printf("%d\n",val[1].solve()) ;
        }
    }
    return 0;
}
```

(19) 树链剖分+线段树 O(nlog^2n)

```
/**洛谷 P2590 [ZJOI2008]树的统计*/
/*
一棵树上有 n 个节点，编号分别为 1 到 n，每个节点都有一个权值 w。
我们将以下面的形式来要求你对这棵树完成一些操作：
```

```
        I. CHANGE u t : 把结点 u 的权值改为 t
        II. QMAX u v: 询问从点 u 到点 v 的路径上的节点的最大权值
        III. QSUM u v: 询问从点 u 到点 v 的路径上的节点的权值和
注意：从点 u 到点 v 的路径上的节点包括 u 和 v 本身
*/
#include <bits/stdc++.h>
#define in32(x) (scanf("%d",&x))
#define in64(x) (scanf("%d",&x))
#define out32(x) (printf("%d",x))
#define out64(x) (printf("%lld",x))
#define en putchar('\n')
#define sp putchar(' ')
const int MAXN (30005);
const int inf (0x3f3f3f3f) ;
const int64_t INF (0x3f3f3f3f3f3f3f3f) ;
using namespace std ;
int m,n,k,buff;
int size[MAXN],deep[MAXN],son[MAXN],fa[MAXN],id[MAXN],top[MAXN] ;
struct Edge
{
        int to,Next ; int64_t val ;
};
Edge edge[MAXN<<2] ;
int head[MAXN],tot ,tot2 ;
int64_t Tmax[MAXN<<2],Tsum[MAXN<<2];
int LD[MAXN<<2],RD[MAXN<<2] ;
void add_edge(int from,int to,int val=0)
{
        edge[tot].Next = head[from] ;
        edge[tot].to = to ;
        edge[tot].val = val ;
        head[from] = tot++ ;
}
void Init(int n)
{
        for(int i(0);i<=n;++i) head[i] = -1 ;
        tot = tot2 = 1 ;
}
void dfs1(int pos = 1,int f = -1,int _deep = 1)
{
        size[pos] = 1 ;
        deep[pos] = _deep ;
        fa[pos] = f ;
        son[pos] = -1 ;
```

```
        int son_size(0) ;
        for(int i(head[pos]);~i;i=edge[i].Next) {
                int p(edge[i].to);
                if(p == f) continue ;
                dfs1(p,pos,_deep+1) ;
                size[pos] += size[p] ;
                if(size[p]>son_size) son_size = size[p],son[pos] = p ;
        }
}
void dfs2(int pos=1,int _top=1)
{
        id[pos] = tot2++ ;
        top[pos] = _top ;
        if(~son[pos]) dfs2(son[pos],_top) ;
        for(int i(head[pos]);~i;i = edge[i].Next) {
                int p(edge[i].to) ;
                if(p == fa[pos]||p == son[pos]) continue ;
                dfs2(p,p) ;
        }
}
void build(int l,int r,int pos = 1 )
{
        LD[pos] = l,RD[pos] = r ;
        Tmax[pos] = Tsum[pos] = 0 ;
        if(l == r) return ;
        int mid(l+r>>1) ;
        build(l,mid,pos<<1) ;
        build(mid+1,r,pos<<1|1) ;
}
void push_up(int pos)
{
        Tsum[pos] = Tsum[pos<<1] + Tsum[pos<<1|1] ;
        Tmax[pos] = max(Tmax[pos<<1],Tmax[pos<<1|1]) ;
}
void change(int p,int d,int pos=1)
{
        if(LD[pos] == RD[pos]) {
                Tmax[pos] = Tsum[pos] = d ;
                return ;
        }
        int mid(LD[pos]+RD[pos]>>1) ;
        if(p<=mid) change(p,d,pos<<1) ;
        else change(p,d,pos<<1|1) ;
        push_up(pos) ;
```

```
}
int64_t QMax(int l,int r,int pos = 1)
{
	if(l==LD[pos]&&r == RD[pos]) {return Tmax[pos];}
	int mid(LD[pos]+RD[pos]>>1) ;
	if(r<=mid) return QMax(l,r,pos<<1) ;
	else if(l>mid) return QMax(l,r,pos<<1|1) ;
	else return max(QMax(l,mid,pos<<1),QMax(mid+1,r,pos<<1|1));
}
int64_t QSum(int l,int r,int pos = 1)
{
	if(l==LD[pos]&&r == RD[pos]) {return Tsum[pos];}
	int mid(LD[pos]+RD[pos]>>1) ;
	if(r<=mid) return QSum(l,r,pos<<1) ;
	else if(l>mid) return QSum(l,r,pos<<1|1) ;
	else return QSum(l,mid,pos<<1)+QSum(mid+1,r,pos<<1|1);
}
int64_t Tree_QMax(int u,int v)
{
	int64_t ans(-INF) ;
	while(top[u]!=top[v]) {
		if(deep[top[u]]<deep[top[v]]) swap(u,v) ;
		ans = max(ans,QMax(id[top[u]],id[u])) ;
		u = fa[top[u]] ;
	}
	if(deep[u]>deep[v]) swap(u,v) ;
	ans = max(ans,QMax(id[u],id[v])) ;
	return ans ;
}
int64_t Tree_QSum(int u,int v)
{
	int64_t ans(0) ;
	while(top[u]!=top[v]) {
		if(deep[top[u]]<deep[top[v]]) swap(u,v) ;
		ans += QSum(id[top[u]],id[u]) ;
		u = fa[top[u]] ;
	}
	if(deep[u]>deep[v]) swap(u,v) ;
	ans += QSum(id[u],id[v]) ;
	return ans ;
}
int main()
{
	in32(n) ;
```

```
        Init(n) ;
        for(int i(0);i<n-1;++i) {
            int a,b;
            in32(a),in32(b) ;
            add_edge(a,b) ;
            add_edge(b,a) ;
        }
        dfs1() ; dfs2() ; build(1,n) ;
        for(int i(1);i<=n;++i) {in32(buff);change(id[i],buff);}
        in32(m) ;   char op[15] ;
        while(m-->0) {
            scanf("%s",op) ;
            int u,v;   in32(u),in32(v) ;
            switch (op[1])
            {
            case 'M':   { out64(Tree_QMax(u,v)) ; en ; break ;}
            case 'S':   {out64(Tree_QSum(u,v)) ; en ; break ;}
            case 'H':   {change(id[u],v); break ;}
            }
        }
}
```

(20) 树链剖分边权转点权

直接把边权下推即可转成点权。
转成点权需要注意的是，计算路径的时候要去除 LCA；
先 dfs 一遍，即可。
树链剖分的最后一条链上 top[]最小的就是 LCA.

(21) 树链剖分换根

改变根对路径的形态显然没有影响。
考虑怎么计算出根改变后不同节点的当前子树。
考虑我们现在的树根是 root，当前询问的子树根节点是 u。
那么会有以下情况（以下 LCA 均指原树中的 LCA）
1.u==root，那么 u uu 的子树就是整棵树。
2.LCA(root,u)≠u,即 root 不在 u 的子树中。那么 u 现在的子树就是原来的子树
3.LCA(root,u)=u,即 u 在原来的树中是 root 的祖先。那么我们找到 u 到 root 路径上的第一个儿子。这个儿子对应的原树中的子树,就是现在 u 的子树的补集。

(22) 树状数组实现平衡树 O(logn)

///洛谷普通平衡树 P3369

```
#include <bits/stdc++.h>
#include <hash_map>
#define LOWBIT(x) (x&(-x))
#define O3 __attribute__((optimeze("-O3")))
using namespace std ;
class BIT
{
public:
    int BITS ;
    __gnu_cxx::hash_map<int,int> T; ///这里 hash_map 比 unorder_map 快大约 2.2 倍.
    ///int T[33554432] ;///数组最快,unorder_map 是数组的 20 倍
    inline void insert(int val,int d=1)///d 是插入的数目,d<0 就是删除
    {
        if(val == 0) return ;
        for(int64_t i(val);i<=1<<BITS;i += LOWBIT(i)) {T[i] += d ;}
    }
    inline int Rank(int val)
    {///小于等于 val 的数的个数
        if(val == 0) return 0;
        int ans(0) ;    for(int64_t i(val);i;i-=LOWBIT(i)) ans += T[i] ;
        return ans ;
    }
    inline int Kth(int k)
    {///查询第 k 小的数
        int ans(1<<BITS) ;
        for(int i(1<<BITS);i>>=1;)    if(T[ans-i]>=k) ans -= i ;    else k -= T[ans-i];
        return ans;
    }
    inline int Pre(int val)///查询前驱(前驱定义为小于 val，且最大的数)
        {return Kth(Rank(val-1));}
    inline int Nxt(int val)///查询后继(后继定义为大于 val，且最小的数)
        {return Kth(Rank(val)+1);}
    inline void clear() ///清空 T 数组
        {T.clear();}
    BIT(int bits=30):BITS(bits){clear() ;}
};
BIT b(25) ;
O3 int main()
{
    const int BASE (10000005) ;
    int m,op,d ;
    scanf("%d",&m) ;
    while(m-->0)
    {
```

```
            scanf("%d%d",&op,&d) ;
            switch (op)
            {
            case 1: b.insert(d+BASE);break ;
            case 2: b.insert(d+BASE,-1) ;break ;
            case 3: printf("%d\n",b.Rank(d+BASE-1)+1) ;break ;
            case 4: printf("%d\n",b.Kth(d)-BASE) ;break ;
            case 5: printf("%d\n",b.Pre(d+BASE)-BASE) ;break ;
            case 6: printf("%d\n",b.Nxt(d+BASE)-BASE) ;break ;
            }
        }
}
```

(23) 树状数组实现维护极值 O(nlog^2n)

```
int a[100005] ;

class BIT
{
#define MAXN 100005
#define INF 0x3f3f3f3f
public:
    int h[MAXN+5],l[MAXN+5], n;    // a[] 数列的值
    //h[] 区间最大值
    //l[] 区间最小值
    BIT(int _n=MAXN) {clear(_n) ;}
    void clear(int _n=MAXN)
    {
        n = _n ; for(int i(0); i<=n; ++i) {h[i] = -INF; l[i] = INF ;}
    }
    void insert (int pos, int val)
    {
        while (pos <= n) {h[pos] = max (h[pos], val);    l[pos] = min (l[pos], val) ; pos += pos&(-pos);}
    }
    int Q(int L, int R,bool isMax = true )
    {//isMax==true 时,返回最大值
        int ret = isMax?-INF:INF, len = R-L+1;
        while (len && R)    {     // len 是当前还需要判断的范围长度，R 是对应区间最大值的下标
            if (len < (R&(-R))) {
                ret = isMax?max(ret, a[R]):min(ret,a[R]); R--; len--;
            } else {
                ret = isMax?max (ret, h[R]):min(ret,l[R]);
                len -= (R&(-R)); //  不断的缩短要判断的区间长度  R -= (R&(-R));
            }
        }
```

```
        return ret;
    }
#undef MAXN
#undef INF
};
BIT bit ;
```

(24) 线段树加法乘法同时更新

```
/**
    线段树加法和乘法同时更新,包括推平一个区间.
    单次操作时间复杂度为 O(logn).
    如果维护其他东西的话,要先更新乘法,再更新加法!!!!!!
*/
#include <bits/stdc++.h>
const int MAXN(100005) ;
using namespace std ;
int64_t Mod ;
class STree
{
#define L (pos<<1)
#define R (pos<<1|1)
static const int N = MAXN<<2 ;
public :
    int64_t Sum[N],lazy_mul[N],lazy_add[N] ;
    int LD[N],RD[N] ;
    inline void push_up(int pos) {
        Sum[pos] = (Sum[L] + Sum[R])%Mod ;
    }
    void build(int* arr,int l,int r,int pos = 1) {
        LD[pos] = l ; RD[pos] = r ;
        Sum[pos] = lazy_add[pos] =    0;lazy_mul[pos] = 1;
        if(l == r) {Sum[pos] = arr[l]; return    ;}
        int mid((l+r)>>1) ;
        build(arr,l,mid,L) ; build(arr,mid+1,r,R) ;
        push_up(pos) ;
    }
    inline void push_tag(int pos,int64_t add,int64_t mul) {
        Sum[pos] = (((Sum[pos]*mul)%Mod) + (((RD[pos]-LD[pos]+1)*add)%Mod))%Mod ;
        lazy_add[pos] = ((lazy_add[pos]*mul)%Mod + add)%Mod ;
        lazy_mul[pos] = (lazy_mul[pos]*mul)%Mod ;
    }
    inline void push_down(int pos) {
        int64_t &add(lazy_add[pos]),&mul(lazy_mul[pos]) ;
        push_tag(L,add,mul) ; push_tag(R,add,mul) ;
```

```
            add = 0 ; mul = 1;
        }
        void change(int l,int r,char op,int64_t d,int pos=1) {
            if(LD[pos] == l&&r == RD[pos]) {
                switch(op)
                {
                    case '+':/**区间加 d*/ push_tag(pos,d,1) ; break ;
                    case '*':/**区间乘 d*/ push_tag(pos,0,d) ; break ;
                    case '_':/**区间推平为 d*/ push_tag(pos,d,0) ; break ;
                }
                return ;
            }
            push_down(pos) ;
            int mid((LD[pos]+RD[pos])>>1) ;
            if(r<=mid) change(l,r,op,d,L) ;
            else if(l>=mid+1) change(l,r,op,d,R) ;
            else change(l,mid,op,d,L),change(mid+1,r,op,d,R) ;
            push_up(pos) ;
        }
        int64_t Q(int l,int r,int pos=1) {
            if(LD[pos] == l&&RD[pos] == r) return Sum[pos] ;
            push_down(pos) ;
            int mid((LD[pos]+RD[pos])>>1) ;
            if(r<=mid) return Q(l,r,L) ;
            else if(l>=mid+1) return Q(l,r,R) ;
            else return (Q(l,mid,L)+Q(mid+1,r,R))%Mod    ;
        }
#undef L
#undef R
}tree;
int n,m ;
int arr[MAXN] ;
int main()
{
        scanf("%d%d%lld",&n,&m,&Mod) ;
        for(int i(1);i<=n;++i) scanf("%d",&arr[i]) ;
        tree.build(arr,1,n) ;
        int op,l,r,d ;
        while(m-->0) {
            scanf("%d%d%d",&op,&l,&r) ;
            switch (op)
            {
            case 1: scanf("%d",&d) ; tree.change(l,r,'*',d%Mod) ; break ;
            case 2: scanf("%d",&d) ; tree.change(l,r,'+',d%Mod) ; break ;
```

```
                case 3: printf("%lld\n",tree.Q(l,r)) ;
                }
            }
        }
```

(25) 线段树染色问题

```
#include <bits/stdc++.h>
#define in32(x) (scanf("%d",&x))
#define in64(x) (scanf("%lld",&x))
#define out32(x) (printf("%d",x))
#define out64(x) (printf("%lld",x))
#define en putchar('\n')
#define sp putchar(' ')
#define O2 __attribute__((optimize("-o2")))
using namespace std ;
const int MAXN(200005) ; const int inf (0x3f3f3f3f) ; const int64_t INF (0x3f3f3f3f3f3f3f3f) ;
int n,m,k,buff,t ;    bool isok[35] ;
int LD[MAXN<<2],RD[MAXN<<2],lazy[MAXN<<2],data[MAXN<<2];
O2 void build(int l,int r,int pos=1)
{
    lazy[pos] = 0 ;data[pos] = 1 ;
    LD[pos] = l,RD[pos] = r ;
    if(l == r) return ;
    int mid((l+r)>>1) ;    build(l,mid,pos<<1) ;    build(mid+1,r,pos<<1|1) ;
}
inline void push_up(int &pos)
{
    if(data[pos<<1]!=data[pos<<1|1]) data[pos] = -1 ;    else data[pos] = data[pos<<1] ;
}
inline void push_down(int &pos)
{
    if(lazy[pos])
    {data[pos<<1] = data[pos<<1|1] = lazy[pos] ; lazy[pos<<1] = lazy[pos<<1|1] = lazy[pos] ; lazy[pos] = 0 ;}
}
O2 void change(int l,int r,int d,int pos=1)
{
    if(LD[pos] == l&&RD[pos] == r) {
        lazy[pos] = d ;    data[pos] = d ;    return ;
    }
    push_down(pos) ;
    int mid((LD[pos]+RD[pos])>>1) ;
    if(r<=mid) change(l,r,d,pos<<1) ;
    else if(l>mid) change(l,r,d,pos<<1|1) ;
    else { change(l,mid,d,pos<<1) ; change(mid+1,r,d,pos<<1|1) ;}
```

```
        push_up(pos) ;
}
O2 int Q(int l,int r,int pos=1)
{
        if(LD[pos] == l&&RD[pos] == r&&data[pos] != -1) {
                if(isok[data[pos]]) { return 0 ;} else { isok[data[pos]] = true ; return 1 ;}
        }
        push_down(pos) ;
        int mid((LD[pos]+RD[pos])>>1) ;
        if(r<=mid)    return Q(l,r,pos<<1) ;
        else if(l>mid)    return Q(l,r,pos<<1|1) ;
        else {return Q(l,mid,pos<<1) + Q(mid+1,r,pos<<1|1) ;}
}
int main()
{
        while(~in32(n))
        {
                in32(t) ; in32(m) ; build(1,n) ;
                while(m-->0) {
                        char op[15] ;
                        scanf("%s",op) ;
                        int l,r,c ; in32(l),in32(r); if(l>r) swap(l,r) ;
                        if(op[0] == 'C') {
                                in32(c) ;    change(l,r,c) ;
                        } else {
                                memset(isok,0,sizeof(isok)) ; out32(Q(l,r)) ;en ;
                        }
                }
        }
}
/*
8 10 1000000
C 3 3 2
C 4 4 1
C 5 5 1
C 6 6 3
Q 3 7
*/
```

(26) 支持区间操作的树状数组(logn)

```
class BIT
{
static const int MAXN = 1000005 ;
```

```
public:
    /**树状数组区间更新+区间查询*/
    /**设 c[i] = a[i] - a[i-1]*/
    /**a[1]+...+a[n]=(n+1)(c[1]+...+c[n])- (1*c[1]+2*c[2]+...+n(c[n])) */
    int64_t sum1[MAXN];///差分数组:(c[1]+...+c[n])
    int64_t sum2[MAXN];///(1*c[1]+2*c[2]+...+n(c[n]))
    void clear(int n) {for(int i(0);i<=n;++i) sum1[i] = sum2[i] = 0 ;}
    inline void range_add(int l, int r,int64_t val=1) ///区间修改
        {if(l>r)return ;add(l,val);add(r+1,-val);}
    inline int64_t range_Q(int l, int r) ///查询区间[l,r]的和
        {if(l>r)return 0 ;return Q(r) - Q(l - 1);}
private:
    inline void add(int pos, int64_t val=1) {///单点更新[单点修改用区间修改代替,这个函数不能用!!!!!!!]
        for (int i(pos);i<=MAXN;i+=LOWBIT(i))
            sum1[i] += val, sum2[i] += val*(pos-1) ;
    }
    inline int64_t Q(int pos) {///查询[1,pos]前缀和
        int64_t ans(0);
        for (int i(pos);i>=1;i-=LOWBIT(i))
            ans += pos * sum1[i] - sum2[i];
        return ans;
    }
}bit;
```

8. 可持久化数据结构

(1) 动态开点静态主席树

```
class PTree
{
#define INF 0x3f3f3f3f
#define MAXN 200005
#define N MAXN*40
public:
    struct Node {int L,R;int64_t data;}T[N] ;
    int root[MAXN+5],tot;
    int64_t max_val,min_val ;
    PTree(int _n=MAXN,int64_t min_val = 0,int64_t max_val=INF) {clear(_n,min_val,max_val);}
    inline void clear(int _n=MAXN,int64_t _min_val=0,int64_t _max_val=INF)
    {
        tot=0;min_val = _min_val;max_val=_max_val ;
```

```
	T[0] = {0,0,0} ; for(int i(0);i<=n;++i) root[i] = 0 ;
}

inline int newNode()
		{T[++tot] = {0,0,0} ; return tot ;}

void insert(int rlast,int& rnow,int64_t l,int64_t r,int64_t p,int val = 1)
{
	rnow = newNode();T[rnow]=T[rlast];T[rnow].data += val ;
	if(l == r) return ;
	int64_t mid((l+r)>>1) ;
	if(p<=mid) insert(T[rlast].L,T[rnow].L,l,mid,p,val) ;
	else insert(T[rlast].R,T[rnow].R,mid+1,r,p,val) ;
}

int _Rank(int pl,int pr,int64_t l,int64_t r,int64_t val)
{
	int64_t ans(0) ;
	while(l!=r) {
			int64_t mid((l+r)>>1) ;
			int dson = T[T[pr].L].data - T[T[pl].L].data ;
			if(val<=mid) {pl = T[pl].L;pr = T[pr].L;;r = mid;}
			else {ans +=    dson ;pl = T[pl].R;pr = T[pr].R;l = mid+1;}
		}
	return ans; ///(<val:0) ;(<=val:T[pr].data-T[pl].data)
}

int64_t _Kth(int pl,int pr,int64_t l,int64_t r,int64_t k)
{
		while(l!=r) {
			int d = T[T[pr].L].data-T[T[pl].L].data ;
			int64_t mid((l+r)>>1) ;
			if(d<k) {pl = T[pl].R; pr = T[pr].R; l = mid+1;k-=d;}
			else {pl = T[pl].L; pr = T[pr].L; r = mid;}
	    }
		return l;
}

void insert(int i,int64_t d,int64_t val = 1)
{
	root[i] = 0 ;
	if(val == 0) {root[i] = root[i-1] ; return ;}
	insert(root[i-1],root[i],min_val,max_val,d,val) ;
}
```

```
    int Rank(int l,int r,int64_t val)/**在区间[l,r]查询比 val 小的数有多少个*/
        {return _Rank(root[l-1],root[r],min_val,max_val,val) ;}

    int64_t Kth(int l,int r,int64_t k)/**在区间[l,r]查询第 k 小,例如 5 1 3 2 中 Kth(3,4,1) =
2*/
        {return _Kth(root[l-1],root[r],min_val,max_val,k) ;}
#undef N
#undef MAXN
#undef INF
};
PTree tree;
```

主席树找区间 K 近邻:

我们可以直接二分 k 近邻与点 p 的距离 mid;

然后利用主席树得 Rank 查询[p-mid.p+mid]之间的数的个数.

如果个数 cnt>=k,那么说明合法,r = mid;否则 l=mid+1;

Q: 为什么 cnt>=k 合法呢？因为可能有两个相等的，比如说|3-4| = 1,|5-4| = 1;

如果我们二分得到的 mid 是 1 的话 k=1,r=mid-1 的话,直接下一次 r 就=0 了,就错了.

主席树找区间 K 近邻:

我们可以直接二分 k 近邻与点 p 的距离 mid;

然后利用主席树得 Rank 查询[p-mid.p+mid]之间的数的个数.

如果个数 cnt>=k,那么说明合法,r = mid;否则 l=mid+1;

Q: 为什么 cnt>=k 合法呢？因为可能有两个相等的，比如说|3-4| = 1,|5-4| = 1;

如果我们二分得到的 mid 是 1 的话 k=1,r=mid-1 的话,直接下一次 r 就=0 了,就错了.

(2) 可持久化数组

```
/// 实现 1:rope 实现
///优点:最简单,缺点:空间复杂度和时间复杂度都很高.
template<typename T>
class PersistenceArray
{///rope 实现可持久化数组,1e5 以下可以用,1e6 会 MLE 和 TLE
public:
    vector<rope<T>*> tree ;
    void change(int id,int pos,T data)
    {
        tree.push_back(new rope<T>(*tree[id])) ;
        tree.back()->replace(pos,data) ;
    }
    T Q(int id,int pos,bool isAdd = false)
    {///这里注意,如果 isAdd = true 的话,查询也会产生新的数组.
        if(isAdd) tree.push_back(tree[id]) ;
```

```
        return tree.back()->at(pos) ;
    }
    void clear()
    {
        for(int i(0);i<tree.size();++i) delete(tree[i]) ;
        tree.clear() ;
        tree.push_back(new rope<T>);
        tree[0] -> push_back(0) ;
    }

    void insert(T x) ///插入初始值
        {tree[0].push_back(x);}
};
PersistenceArray<int> arr ;

///实现 2:链表模拟
///优点:比较简单,很快  缺点:用了指针,打错容易 RE

template<typename T>
class PersistenceArray
{
public:
    static const int MAXM = 1000005 ;///操作的次数.
    struct Node{int pos;T val;Node* pre;};
    Node tree[MAXM+5];
    vector<T> arr ;int tot;
    void clear(int m)
    {
        arr.clear();arr.push_back(0) ;
        tree[tot++] = {0,0,0} ;
    }
    //vector<int>::iterator
    void push_back(T x) ///插入初始值
        {arr.push_back(x);} ;

    void change(int id,int pos,T val)///把 id 版本 pos 位置的数改成 val
        {tree[tot++] = {pos,val,tree+id};}

    T Q(int id,int pos,bool isAdd = false)
    {///询问 id 版本的 pos 位置的数
        if(isAdd) tree[tot++] = tree[id] ;
        Node* p = tree + id ;
        while(p->pre != NULL){
            if(p->pos==pos) return p -> val ;
```

```
            p = p->pre;
        }
        return arr[pos] ;
    }
};
PersistenceArray<int> arr ;
```

(3) 树链剖分套主席树 O(nlog^2n)

```
/*树上路径上权值<=k 的边的个数*/
#include <bits/stdc++.h>
using namespace std;
class PTree
{
#define INF 0x3f3f3f3f3f3f3f3f
#define MAXN 100005
#define N MAXN*40
public:
    struct Node {int L,R,data;}T[N] ;
    int root[MAXN+5],tot,n;
    int64_t max_val ;
    PTree(int _n=MAXN,int64_t max_val=INF) {clear(_n,max_val);}
    inline void clear(int _n=MAXN,int64_t _max_val=INF)
    {
        tot=0;n=_n;max_val=_max_val ;
        T[0] = {0,0,0} ;
        for(int i(0);i<=n;++i) root[i] = 0 ;
    }

    int newNode(int L=0,int R=0,int data=0)
    {
        int ret = ++ tot   ;
        T[ret].L = L; T[ret].R = R ;
        T[ret].data = data ;
        return ret ;
    }

    void insert(int rlast,int& rnow,int64_t l,int64_t r,int64_t p,int val = 1)
    {
        rnow = newNode(),T[rnow]=T[rlast],T[rnow].data += val ;
        if(l == r) return ;
        int64_t mid((l+r)>>1) ;
        if(p<=mid) insert(T[rlast].L,T[rnow].L,l,mid,p,val) ;
        else insert(T[rlast].R,T[rnow].R,mid+1,r,p,val) ;
```

```
    }

    int _Rank(int pl,int pr,int64_t l,int64_t r,int64_t val)
    {
        if(l==r) return T[pr].data-T[pl].data;///(<val:0) ;(<=val:T[pr].data-T[pl].data)
        int64_t mid((l+r)>>1) ;
        int dson = T[T[pr].L].data - T[T[pl].L].data ;
        if(val<=mid) return _Rank(T[pl].L,T[pr].L,l,mid,val) ;
        else return _Rank(T[pl].R,T[pr].R,mid+1,r,val) + dson;
    }

    int64_t _Kth(int pl,int pr,int64_t l,int64_t r,int k)
    {
        if(l==r) return l ;
        int d = T[T[pr].L].data-T[T[pl].L].data ;
        int64_t mid((l+r)>>1) ;
        if(d<k) return _Kth(T[pl].R,T[pr].R,mid+1,r,k-d) ;
        else return _Kth(T[pl].L,T[pr].L,l,mid,k) ;
    }

    void insert(int i,int64_t d)
    {
        root[i] = 0 ;
        insert(root[i-1],root[i],-max_val,max_val,d) ;
    }

    int Rank(int l,int r,int64_t val)/**在区间[l,r]查询比 val 小的数有多少个*/
    {
        return _Rank(root[l-1],root[r],-max_val,max_val,val) ;
    }

    int64_t Kth(int l,int r,int k)/**在区间[l,r]查询第 k 小,例如 5 1 3 2 中 Kth(3,4,1) = 2*/
    {
        return _Kth(root[l-1],root[r],-max_val,max_val,k) ;
    }
#undef N
#undef MAXN
#undef INF
};
PTree tree;
const int MAXN = 100005;
const int MAXM = 100005;
int arr[MAXN] ;
struct Edge {int to,val,Next;}edge[MAXM<<1];
```

```
int head[MAXN],tot;

void add_edge(int u,int v,int val)
{
    edge[tot].to = v ;
    edge[tot].val = val ;
    edge[tot].Next = head[u];
    head[u] = tot ++ ;
}

void addedge(int u,int v,int val)
{
    add_edge(u,v,val) ;
    add_edge(v,u,val) ;
}

int Size[MAXN],top[MAXN],fa[MAXN],deep[MAXN],Son[MAXN],id[MAXN],cur ;

void dfs1(int pos=1,int _deep = 1,int _fa=0)
{
    deep[pos] = _deep ;
    fa[pos] = _fa ;
    Son[pos] = 0 ;
    Size[pos] = 1 ;
    for(int i=head[pos];i!=-1;i = edge[i].Next) {
        int p = edge[i].to,val = edge[i].val ;
        if(p == _fa) continue ;
        arr[p] = val; //边权下推成点权
        dfs1(p,_deep+1,pos) ;
        Size[pos] += Size[p] ;
        if(Size[p] > Size[Son[pos]]) Son[pos] = p ;
    }
}

void dfs2(int pos=1,int _top=1)
{
    id[pos] = ++cur ;
    tree.insert(cur,arr[pos]) ;
    top[pos] = _top ;
    if(Son[pos]) dfs2(Son[pos],_top) ;
    for(int i = head[pos];i!=-1;i = edge[i].Next) {
        int to = edge[i].to;
        if(to == fa[pos]||to == Son[pos]) continue ;
        dfs2(to,to) ;
```

```
    }
}

void Init(int n)
{
    tree.clear(n,0x3f3f3f3f) ;
    tot = cur = 0 ;
    for(int i = 0 ; i <= n; ++i) {
        head[i] = -1 ;
    }
}

int64_t Q(int u,int v,int val)
{
    int64_t ans (0) ;
    while(top[u]!=top[v]) {
        if(deep[top[u]]<deep[top[v]]) swap(u,v) ;
        ans += tree.Rank(id[top[u]],id[u],val) ;
        u = fa[top[u]];
    }
    if(deep[u]>deep[v]) swap(u,v);
    if(id[u]!=id[v]) //去掉 lca
        ans += tree.Rank(id[u]+1,id[v],val);
    return ans ;
}

int main()
{
    int n,m ;
    scanf("%d%d",&n,&m);
    {
        Init(n) ;
        for(int i=1;i<n; ++ i) {
            int u,v,val;
            scanf("%d%d%d",&u,&v,&val) ;
            addedge(u,v,val) ;
        }
        dfs1() ;dfs2() ;
        while (m-->0) {
            int u,v,val;
            scanf("%d%d%d",&u,&v,&val) ;
            printf("%lld\n",Q(u,v,val)) ;
        }
    }
```

```
    return 0 ;
}
```

(4) 主席树得到区间不同的数的个数

last[i] 表示上一个颜色为 a[i]的位置，没有则为 0。
pre[i] 表示上一个 i 颜色的位置
那么一个区间里的答案就是
r
∑ (last[i]<l)
i=l
我们可以维护一个 pre[]和 last[],把 last[i]插入到主席树中,用主席树维护一下就可以了.

```
int a[100005],last[100005],pre[100005] ;
for(int i(1);i<=n;++i) {//插入
                scanf("%d",&buff) ;
                a[i] = buff ;
                last[i] = pre[buff] ;
                pre[buff] = i ;
                tree.insert(i,last[i]) ;
}
//询问[l,r]不同的数的个数:
tree.Rank(l,r,l) ;
```

9. 数论&&组合数学

(1) FFT&&NTT&&多项式

① 常数优化 FFTO(nlogn)

```
#include <bits/stdc++.h>
using namespace std;
const double PI = acos(-1.0);
const int MAXN = 1<<21|1 ;
struct Complex
{
    double x,y;
    Complex operator +(Complex o) {return {x+o.x,y+o.y};}
    Complex operator -(Complex o) {return {x-o.x,y-o.y};}
    Complex operator *(Complex o) {return {x*o.x-y*o.y,y*o.x+x*o.y};}
    Complex operator *(double o) {return {x*o,y*o};}
    Complex operator !(){return{x,-y};}
} x[MAXN],y[MAXN],z[MAXN],w[MAXN];
```

```
void FFT(Complex x[],int n,int v = 1)
{
	for(int i(0),j(0);i < n; ++i) {
		if(i>j) swap(x[i],x[j]);
		for(int l(n>>1);(j^=l) < l;l >>= 1) ;
	}
	w[0] = {1,0};
	for(int i(2); i<=n; i<<=1) {
		Complex g={cos(2*PI/i),v*sin(2*PI/i)};
		for(int j(i>>1);j>=0;j -= 2) w[j] = w[j>>1];
		for(int j(1);j < i>>1;j += 2) w[j] = w[j-1]*g;
		for(int j(0);j < n; j+=i) {
			Complex *a(x+j),*b(a+(i>>1));
			for(int l(0);l < (i>>1); ++l) {
				Complex o(b[l]*w[l]);
				b[l] = a[l]-o;
				a[l] = a[l]+o;
			}
		}
	}
	if(v<0) for(int i(0);i < n; ++i) x[i] = {x[i].x/n,x[i].y/n};
}

int n,m,a;
int main()
{
	scanf("%d%d",&n,&m) ;
	for(int i(0); i<=n; ++i) {
		scanf("%d",&a) ;
		(i&1?x[i>>1].y:x[i>>1].x) = a;
	}
	for(int i(0); i<=m; ++i) {
		scanf("%d",&a) ;
		(i&1?y[i>>1].y:y[i>>1].x) = a;
	}
	int len(1) ;
	while(len < n+m) len <<= 1;
	FFT(x,len);
	FFT(y,len);
	for(int i=0; i<len; ++i) {
		int j=(len-1)&(len-i);
		z[i]=(x[i]*y[i]*4-(x[i]-!x[j])*(y[i]-!y[j])*(((i&len>>1)?(Complex)
		{1,0}-w[i^len>>1]:w[i]+(Complex){1,0})))*0.25;
	}
```

```
    FFT(z,len,-1);
    for(int i(0); i<=n+m; ++i) {
        a = (int)((i&1)?z[i>>1].y+0.1:z[i>>1].x+0.1) ;
        printf("%d",a); putchar(i==n+m?'\n':' ') ;
    }
    return 0;
}
```

② 任意模 FFT[O(nlogn)]

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1<<18;
const double PI = acos(-1) ;
class FFT_Mod
{
public:
    struct Complex{
        double x, y;
        inline void operator +=(const Complex &b){ x+=b.x;y+=b.y;}
        inline Complex operator +(const Complex &b)const{ return {x+b.x,y+b.y};}
        inline Complex operator -(const Complex &b)const{ return {x-b.x,y-b.y};}
        inline Complex operator *(const Complex &b)const{ return
{x*b.x-y*b.y,x*b.y+y*b.x};}
        inline Complex operator *(const double b)const{ return {x*b, y*b};}
        inline Complex operator ~()const{return {x,-y};}
    } X[N], Y[N],A[N],B[N], w[N];
    static inline int Align(int n) {
        int ret(1);
        while(ret<n) ret<<=1;
        return ret;
    }
    inline void DFT(Complex *X, int n){
        for(int i(0),j(0); i<n; ++i){
            if(i>j) swap(X[i], X[j]);
            for(int k(n>>1);(j ^= k)<k;k >>= 1);
        }
        for(int i(1); i<n; i<<=1)
            for(int j(0); j<n; j+=i<<1)
                for(int k(j); k<j+i; ++k){
                    Complex t(w[i+k-j]*X[k+i]);
                    X[k+i]=X[k]-t; X[k]+=t;
                }
    }
```

```
    inline void IDFT(Complex *X, int n) {reverse(X+1, X+n), DFT(X, n);}
    void FFT(int64_t* x,int n,int64_t* y,int m,int64_t* z,int64_t Mod)
    {
        int len = Align(n+m) ;
        for(int i(0);i<=n;++i) {X[i].x=x[i]>>16; X[i].y=x[i]&65535;}
        for(int i(0);i<=m;++i) {Y[i].x=y[i]>>16; Y[i].y=y[i]&65535;}

        for(int i=1; i<len; i<<=1){
            w[i] = (Complex){1, 0};
            for(int j(1); j<i; ++j)
                w[i+j]=((j&31)==1?(Complex){cos(PI*j/i),
sin(PI*j/i)}:w[i+j-1]*w[i+1]);
        }
        DFT(X, len); DFT(Y, len);
        for(int i(0); i<len; ++i){
            Complex q, f0, f1, g0, g1;
            int id(i?len-i:0);
            q=~X[id]; f0=(X[i]-q)*(Complex){0, -0.5}; f1=(X[i]+q)*0.5;
            q=~Y[id]; g0=(Y[i]-q)*(Complex){0, -0.5}; g1=(Y[i]+q)*0.5;
            A[i]=f1*g1; B[i]=f1*g0+f0*g1+f0*g0*(Complex){0, 1};
        }
        IDFT(A, len); IDFT(B, len);
        double k(1.0/len);
        for(int i=0; i<=n+m; ++i)
        {
            int64_t a = (int64_t)(A[i].x*k+0.5)%Mod<<32 ;
            int64_t b = (int64_t)(B[i].x*k+0.5)%Mod<<16 ;
            int64_t c = (int64_t)(B[i].y*k+0.5)%Mod ;
            z[i] = (a+b+c)%Mod ;
        }
    }
}FFT;
int64_t X[N],Y[N],Z[N] ;

int main() {

    int n,m;
    int64_t Mod ;
    scanf("%d%d%lld",&n,&m,&Mod) ;
    for(int i(0);i<=n;++i) scanf("%lld",X+i) ;
    for(int i(0);i<=m;++i) scanf("%lld",Y+i) ;
    FFT.FFT(X,n,Y,m,Z,Mod) ;
    for(int i(0);i<=n+m;++i) {
        printf("%lld ",Z[i]) ;
```

```
        }
        return 0;
    }
```

(2) Cantor 表分数形式枚举有理数.O(logn)

```
/**
    现代数学的著名证明之一是 Georg Cantor 证明了有理数是可枚举的.他是用下面这一张表来证明这一命题的:
1/1,1/2,1/3,1/4,1/5,...
2/1,2/2,2/3,2/4,...
3/1,3/2,3/3,...
4/1,4/2,...
5/1,...
...
我们以 Z 字形给上表的每一项编号.第 1 项是 1/1,然后是 1/2,2/1,3/1,2/2,...
求表上的第 K 个分数并输出
*/
#include<bits/stdc++.h>
using namespace std;
void getKthFract(int64_t k,int64_t& x,int64_t& y)
{///二分的到 Cantor 表中的第 K 个 O(logk).
    int64_t l(1),r(k);
    while(l<r) {
        int64_t mid((l+r)>>1) ;
        ((mid*(mid+1))>>1)<k?(l = mid+1):(r = mid);
    }
    x = k - ((l*(l-1))>>1) ;
    y = l + 1 - x ;
    if(l&1) swap(x,y) ;
}
int main()
{
    int64_t k ;
    while(~scanf("%lld",&k))
    {
        int64_t x,y ;
        getKthFract(k,x,y) ;
        cout<<x<<"/"<<y<<endl;
    }
    return 0;
}
```

(3) Fibonacci 数高效记忆化倍增算法

```
#include <iostream>
#include <inttypes.h>
#include <map>
using namespace std;
map<int64_t,int64_t> fibo ;
const int64_t Mod(1000000007) ;
int64_t F(int64_t n)
{
    if(n == 1||n==2) return 1 ;
    if(n <=0) return 0 ;
    if(fibo.find(n)!=fibo.end()) return fibo[n] ;
    if(n&1) {
        int64_t l(F((n+1)>>1)),r(F((n-1)>>1)) ;
        return fibo[n] = (((l*l)%Mod)+((r*r)%Mod)%Mod)%Mod;
    } else {
        return fibo[n] = (F(n>>1)*((((F((n>>1)+1))%Mod)+F((n>>1)-1))%Mod)%Mod)%Mod;
    }
}
```

(4) gcd 和 xor 的特殊性质

a - b <= a xor b （a >= b)
如果 a xor b = gcd(a, b) = c 则 c = a - b
gcd(2^a-1,2^b-1)=2^(gcd(a,b))-1
gcd(fib[a],fib[b])=fib[gcd(a,b)]
gcd((2^x1)-1,(2^x2)-1,...,(2^xk)-1) = 2^(gcd(x1,x2,...,xk))-1

(5) gcd 预处理 O(n+q)

```
/**
    gcd 预处理
题意: 给两个长度为 n 的数组 a[n],b[n],求Σ{j=1,j<=n}(pow(i,j)*gcd(a[i],b[j]))%Mod
数据范围: n <= 5000, a[i],b[i] <= 1e6.
思路:
    O(1e6)预处理出 gcd,然后 O(1)查询即可.
*/
#include <bits/stdc++.h>
using namespace std ;
class Gcd
{///gcd O(值域)预处理,Q(1)查询. 1e7 的预处理半秒可以完成.
 ///大于值域的 gcd 用普通方法优化到值域以内,减小常数.
static const int N = 1000005 ; ///这里的 N 越大,查询的常数越小.
static const int SQR = sqrt(N) + 5;
```

```
public:
    int g[SQR+5][SQR+5],fac[N+5][3];
    int pre[N+5],prime[N+5],tot;
    inline int Q(int a, int b)/**这里的参数可以改成 long long 的*/
    {///最小值在[0,SQR]的是 O(1)查询的,
     ///最小值在[SQR+1~N]的是 O(3)查询的,其余最坏是 O(logn)的.
        if(a > b) swap(a,b) ;
        if(a == 1) return 1 ;
        if(a == 0) return a + b ;
        if(a == b) return a ;
        if(b <= SQR) return g[a][b] ;
        if(a <= SQR) return g[a][b%a] ;
        if(b >= N) return Q(b%a,a) ;
        int ret(1);
        #define fun(i) /**循环展开优化*/ \
        if (fac[a][i] != 1) {                   \
            int& t(fac[a][i]);                  \
            if (pre[t]) {                       \
                int& d(g[t][b%t]);              \
                ret *= d ; b /= d;              \
            } else if (b % t == 0) {            \
                ret *= t ; b /= t;              \
            }                                       \
        }
        fun(0) ;fun(1) ;fun(2) ;
        #undef fun
        return ret;
    }
    void Init() { ///预处理 gcd
        register int a,b ;
        memset(pre,0,sizeof(pre)) ;
        tot = 0 ; memset(g,0,sizeof(g)) ;
        for (register int i = 0; i <= SQR; ++i)
            for (register int j = 0; j <= SQR; ++j) {
                if (i == 0 || j == 0) {g[i][j] = i + j;continue ;}
                a = i ; b = j ;
                if(a>b) swap(a,b) ;
                g[i][j] = g[a][b%a] ;
            }
        fac[1][0] = fac[1][1] = fac[1][2] = 1;
        for (register int i(2); i <= N; ++i) {
            if (!pre[i]) {
                prime[++tot] = i ;
                fac[i][0] = fac[i][1] = 1; fac[i][2] = i;
```

```
                }
                for (register int j(1);j <= tot&& prime[j]*i <= N; ++j) {
                    int num(prime[j]*i),p(prime[j]),id;
                    pre[num] = p;
                    fac[num][0] = fac[i][0];
                    fac[num][1] = fac[i][1];
                    fac[num][2] = fac[i][2];
                    if (fac[num][0]*p <= SQR) id = 0 ;
                    else if (fac[num][1]*p <= SQR) id = 1 ;
                    else id = 2 ;
                    fac[num][id] *= p;
                }
            }
        }
    }gcd;
    int a[50005],b[50005] ;
    int main()
    {
        gcd.Init() ;
        int n ;
        int Mod = 998244353;
        scanf("%d",&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 ans(0),pwi(i) ;
            for(int j(1);j<=n;++j,pwi = (1LL*pwi*i) % Mod) {
                ans = (ans + (1LL*pwi*gcd.Q(a[i],b[j]))%Mod)%Mod ;
            }
            printf("%d\n",ans) ;
        }
    }
```

(6) i 的 j 次方等于 k 的倍数的(i,j)对数

求满足 $d|i^j$的{ < i,j > |i ∈ n,j ∈ m}的数量

解: 我们把 d 分解质因数: $d = \sum_{p\in prime} p^k$之后我们会发现当 j ≥ 64 时,j 的值都是一样的,我们设为 u.所以我们直接暴力 j,计算 $1 \le j \le \log_2 m$的每一个 j. 我们设 $g = \sum_{p\in prime} p^{\left[\frac{k}{j}\right]}$,此时满足条件的 i 一定满足 g|$i$. 所以满足条件的 i 的个数就是: $\frac{n}{g}$. j > 64 时,运算结果直接加上 u ∗ $(m - \log_2 m)$即可. 计算代码如下:

```
int64_t calc(int64_t n,int64_t m,int64_t d)
{
    getFactor(d) ; ///分解质因数
```

```
    int limit = 60 ;
    int64_t ans(0),ansLimit(0) ;
    for(int j(1);j<=min(1LL*limit,1LL*m);++j) {
        int64_t g(1) ;
        for(auto&it:factMap) {
            g *= powEx(it.first,(it.second+j-1)/j,INF) ;
        }
        ans += n/g;
        if(j == limit) ansLimit = n/g ;
    }
    if(m>limit) ans += (m-limit) * ansLimit ;
    return ans ;
}
```

(7) Lucas 组合数大数取模

```
inline int64_t C(int n,int m)
{
    if(m>n-m) m=n-m;
    int64_t x(1),y(1);
    for(int i=1;i<=m;i++){
        x = (x*(n-i+1))%Mod;
        y =(y*i)%Mod;
    }
    return x*Inv(y)%Mod;
}
inline int64_t lucas(int64_t n,int64_t m){
    if(m==0) return 1;
    return C(n%Mod,m%Mod)*lucas(n/Mod,m/Mod);
}
```

(8) Min_25 筛

```
/*
    Min_25 筛 O(n^0.75/ln(n))[1e10 左右]
    用于计算积性函数的前缀和.
积性函数 f(x)应该满足以下条件:
当 p∈prime 时
1. f(p)  是一个关于 p 的多项式.比如μ(p)=-1,φ(p)=p-1.
2. f(p^k)  能够很快的求出来.一般是 O(1).

例题: 洛谷 P5325
定义积性函数 f(x),且 f(p^k)=p^k(p^k-1),求前 n 项和%1e9+7.
*/
```

```
#include <bits/stdc++.h>
const int MAXN (1000005) ;
using namespace std;
const int64_t Mod(1000000007),inv2(500000004),inv3(333333336);
int64_t pCnt,prime[MAXN],sumP[5][MAXN];
bool flag[MAXN];
int64_t n,Sqr,tot,g[5][MAXN],w[MAXN],ind1[MAXN],ind2[MAXN];
/*
w[i]: 记录每一个 n/i 的值,只记录一次,最多记录 sqrt(n)个数.
      比如当 n=10 的时候,w[]={0,10,5,3,2,1}
g[k][i]: 记录Σ{2<=i<=w[i]&&i∈primee}(i^k) 注意!不算 1!
*/
void getP(int n)//预处理，线性筛
{
    flag[1]=1;pCnt = 0 ;
    for(int i(1);i<=n;++ i) {
        if(!flag[i]) {
            prime[++pCnt]=i;
            sumP[1][pCnt]=(sumP[1][pCnt-1]+i)%Mod;
            sumP[2][pCnt]=(sumP[2][pCnt-1]+1LL*i*i)%Mod;
        }
        for(int j(1);j<=pCnt&&prime[j]*i<=n;++ j) {
            flag[i*prime[j]]=1;
            if(i%prime[j]==0)break;
        }
    }
}

void get_g(int64_t n)
{
    Sqr=sqrt(n);  getP(Sqr);  tot = 0 ;
    for(int64_t l(1),r;l<=n;l=r+1) {//整除分块
    //先算Σ{i=2,i<=w[i]}(i^k),然后利用埃氏筛的思想,筛去和数.
        r=n/(n/l);
        w[++tot]=n/l;
        g[1][tot]=w[tot]%Mod;
        g[2][tot]=g[1][tot]*(g[1][tot]+1)/2%Mod*(2*g[1][tot]+1)%Mod*inv3%Mod;
        //g[2][i] = w[i]*(w[i]+1)*(2*w[i]+1)/6 = g[1][i]*(2*w[i]+1)/3
        -- g[2][tot] ;//不能算 1
        g[1][tot]=g[1][tot]*(g[1][tot]+1)/2%Mod;
        //g[1][i] = w[i]*(w[i]+1)/2
        -- g[1][tot] ;//不能算 1
        if(n/l<=Sqr) ind1[n/l]=tot;
        else ind2[r]=tot;
```

```
    }//g[1],g[2]分别表示一次项和二次项的和,ind1 和 ind2 用来记录这个数在数组中的位置
    for(int i=1;i<=pCnt;i++) {//由于 g 数组可以滚动,所以就只开了一维
        for(int j=1;j<=tot&&prime[i]*prime[i]<=w[j];j++) {
            //固定写法,筛去和数的部分.
            int64_t k=w[j]/prime[i]<=Sqr?ind1[w[j]/prime[i]]:ind2[n/(w[j]/prime[i])];
            g[1][j] -= prime[i]*(g[1][k]-sumP[1][i-1]+Mod)%Mod;
            g[2][j] -= prime[i]*prime[i]%Mod*(g[2][k]-sumP[2][i-1]+Mod)%Mod;
            //这里 g[u][j] -= (prime[i]^u)*(g[u][k']-sumP[u][i-1]) ;
            g[1][j] %= Mod,g[2][j] %= Mod;
            if(g[1][j]<0)g[1][j]+=Mod;
            if(g[2][j]<0)g[2][j]+=Mod;
        }
    }
}
int64_t S(int64_t x,int y)//第二部分
{
    if(prime[y]>=x)return 0;
    int64_t k=(x<=Sqr)?ind1[x]:ind2[n/x];
    int64_t ans=((g[2][k]-g[1][k])+Mod-(sumP[2][y]-sumP[1][y])+Mod)%Mod;
    //上边是固定写法,ans 初始值是 g(n)-sumP[y]
    //由于这题是 f(p)=p^2-p,所以是(g[2][k]-g[1][k])-(sumP[2][y]-sum[1][y])
    for(int i(y+1);i<=pCnt&&prime[i]*prime[i]<=x;++i) {
        int64_t pe(prime[i]);
        for(int e(1);pe<=x;pe=pe*prime[i],++e) {//枚举 P 的 e 次方
            int64_t xx(pe%Mod);
            int64_t fx(xx*(xx-1)%Mod) ;
            //这里的 xx=p[i]^e,f(p^k) = p^k*(p^k-1),所以上边 fx 是 xx*(xx-1) ;
            ans=(ans+fx*(S(x/pe,i)+(e!=1)))%Mod;
        }
    }
    return ans%Mod;
}
int main()
{
    scanf("%lld",&n);
    get_g(n) ;
    printf("%lld\n",(S(n,0)+1)%Mod);//1 单算.
    return 0;
}
```

(9) O(4e5)预处理后 O(4)求递推数列第 n 项

```
/**
    F[0] = x,F[1] = y
```

```
    O(4)  求 F[n] = a*F[n-1]+b*F[n-2] (n∈[0,1e18])
*/
#include <bits/stdc++.h>
using namespace std ;
const int Mod = 998244353;
struct Matrix {int a[2][2] ;};
const Matrix E = {1,0,0,1} ; ///单位矩阵
inline Matrix operator*(const Matrix& a,const Matrix& b)
{///循环展开优化矩阵乘法
    Matrix ans ;
    ans.a[0][0] = ((1LL*a.a[0][0]*b.a[0][0])%Mod+(1LL*a.a[0][1]*b.a[1][0])%Mod)%Mod ;
    ans.a[0][1] = ((1LL*a.a[0][0]*b.a[0][1])%Mod+(1LL*a.a[0][1]*b.a[1][1])%Mod)%Mod ;
    ans.a[1][0] = ((1LL*a.a[1][0]*b.a[0][0])%Mod+(1LL*a.a[1][1]*b.a[1][0])%Mod)%Mod ;
    ans.a[1][1] = ((1LL*a.a[1][0]*b.a[0][1])%Mod+(1LL*a.a[1][1]*b.a[1][1])%Mod)%Mod ;
    return ans ;
}
class Pow
{
static const int N = 100000;
public:
    Matrix biao[4][N+5] ;
    void Init(Matrix x) {///O(4e5)  预处理.
        for(register int i(0);i<4;++i) biao[i][0] = E;
        for(register int i(0);i<4;++i) {
            Matrix last(i?biao[i-1][N]:x) ;
            for(register int j(1);j<=N;++j) {
                biao[i][j] = biao[i][j-1]*last ;
            }
        }
    }
    inline Matrix Q(int64_t n) {///O(1)  查询答案.
        if(n<0) return E ;
        int64_t b(n/N),c(b/N),d(c/N) ;///循环展开优化.
        return biao[0][n%N]*biao[1][b%N]*biao[2][c%N]*biao[3][d%N];
    }
}powEx;
int main()
{
    int x(0),y(1),a(3),b(2) ;
    Matrix base = {a,b,1,0},Ans = {y,0,x,0} ;
    powEx.Init(base) ;
    int64_t q,n,last(0),ans(0),ret(0) ;
    scanf("%lld%lld",&n,&q) ;
    last = q ;
```

```
        for (int i = 1; i <= n; i++) {
            last ^= (ans * ans);
            ans = (Ans*powEx.Q(last-1)).a[0][0];
            ret ^= ans;
        }
        printf("%lld\n",ret) ;
    }
```

(10) O(n)预处理逆元和阶乘逆元

```
const int MAXN(100005) ;
const int Mod (1000000007);
int64_t inv[MAXN] ;
int64_t fact[MAXN] ;
int64_t factInv[MAXN] ;
int n,m ;

void getInv(int n,int Mod=::Mod)
{
    inv[1] = fact[0] = fact[1] = 1 ;
    for(int i(2);i<=n;++i) {
        fact[i] = fact[i-1]*i % Mod ;
        inv[i] = (Mod-(Mod / i)) * inv[Mod % i]%Mod ;
    }
    factInv[n] = Inv(fact[n], Mod);
    for(int i(n-1);i>=0;--i) factInv[i] = factInv[i+1]*(i+1) % Mod ;
}
```

(11) O(N^0.25)预处理 O(1)查询快速幂

```
#include <bits/stdc++.h>
using namespace std ;
/**
    注意: 本程序还可以循环展开优化,见矩阵版本.
    数/矩阵  的幂次 O(N^0.25)预处理,Q(4)查询.
求: a^n % Mod
思路:
    设 MAXN 为 n 的最大值,设 N = MAXN^(1/4),那么
    n = id[3]*N^3 + id[2]*N^2 + id[1]*N^1 + id[0]*N^0 ;
    这里 id[0~4]的范围都是[0,N],所以我们只需要预处理出
    a 的[0,N]次方,a^N 的[0,N]次方,
    a^(N^2)的[0,N]次方,a^(N^3)的[0,N]次方即可.
    预处理时间复杂度 O(4*N^0.25),查询时间复杂度 O(4).

    应用: 矩阵快速幂的 O(400000)预处理,O(4)查询,所以
```

```
        可以 1e7 次在线询问矩阵快速幂了.
*/
class Pow
{
static const int N = 100000;
public:
    int64_t biao[4][N+5],Mod ;
    void Init(int64_t x,int64_t _Mod)
    {
        Mod = _Mod ; x %= Mod ;
        for(int i(0);i<4;++i) biao[i][0] = 1;
        for(int i(0);i<4;++i) {
            int64_t last(i?biao[i-1][N]:x) ;
            for(int j(1);j<=N;++j) {
                biao[i][j] = (biao[i][j-1]*last)%Mod ;
            }
        }
    }
    inline int64_t Q(int64_t n)
    {///O(1)  查询答案.
        int64_t ret(1),id ;
        for(int i(0);i<4&&n;++i,n/=N) {
            id = n%N ; ///一点小小的优化,不加也行.
            if(id) ret = (ret * biao[i][id]) % Mod ;
        }
        return ret%Mod ;
    }
}powEx;

int main()
{
    long long   a,n,p ;
    cin >> a >> n >> p ;
    powEx.Init(a,p) ;
    printf("%lld^%lld mod %lld=%lld",a,n,p,powEx.Q(n)) ;
}
```

(12)不定方程解的个数 O(logn)

```
/**
    O(logn)求解 x∈[l1,r1],y 属于[l2,r2]的不定方程:
        ax+by+c=0
    的整数解的个数.
*/
```

```
#define mod(x,Mod) ((((x)%Mod)+Mod)%Mod)
struct Limit{int l,r;};
int64_t gcdEx(int64_t a,int64_t b,int64_t &x,int64_t &y)
{
	if(!b) return x = 1,y = 0,a ;
	int64_t d(gcdEx(b,a%b,y,x)) ;
	return y-=(a/b)*x,d ;
}
int64_t solve(int64_t a,int64_t b,int64_t c,Limit x,Limit y)
{
	if(x.l>x.r||y.l>y.r) return 0 ;
	if(!a&&!b) return c?0:(x.r-x.l+1)*(y.r-y.l+1) ;
	if(!b) swap(a,b),swap(x.l,x.r),swap(y.l,y.r) ;
	if(!a) {
		if(c%b) return 0 ;
		int64_t t(-c/b) ;
		return (t<y.l||y.r<t)?0:(x.r-x.l+1) ;
	}
	int64_t xx,yy,bb(abs(b)),d(gcdEx(mod(a,bb),bb,xx,yy)) ;
	if(c%d) return 0 ;
	xx = (mod(xx,bb)*mod(-c,bb)/d)%abs(b/d) ;
	d = abs(b/d) ;
	int64_t kl((x.l-xx)/d-3),kr((x.r-xx)/d+3) ;
	while(xx+kl*d<x.l) ++ kl ;
	while(xx+kr*d>x.r) -- kr ;
	int64_t A((-y.l*b-a*xx-c)/(a*d)),B((-y.r*b-a*xx-c)/(a*d)) ;
	if(A>B) swap(A,B) ;
	kl = max(kl,A-3) ; kr = min(kr,B+3) ;
	while(kl<=kr) {
		int64_t Y((-c-a*xx-a*d*kl)/b);
		if(y.l<=Y&&Y<=y.r) break ;
		++ kl ;
	}
	while(kl<=kr) {
		int64_t Y((-c-a*xx-a*d*kl)/b);
		if(y.l<=Y&&Y<=y.r) break ;
		-- kr ;
	}
	return (kl>kr)?0:(kr-kl+1) ;
}
```

(13)超级线性筛+莫比乌斯反演 O(Tsqrt(n))

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

```
/*洛谷 P2257 YY 的 GCD*/
const int MAXN(10000005) ;
int prime[MAXN+10],Mobius[MAXN+10],phi[MAXN+10];
bool isP[MAXN+10] ;
int64_t Sum[MAXN+10] ;

void getPEx()
{///超级线性筛
    for(int i(0);i<=MAXN;++i) {isP[i] = true;Sum[i] = 0 ;}
    phi[1] = Mobius[1] = 1;
    for(int i(2);i<=MAXN;++i) {
        if(isP[i]) {prime[++prime[0]] = i ;Mobius[i] = -1;phi[i]=i-1;}
        for(int j(1);j<=prime[0]&&prime[j]<=MAXN/i;++j) {
            int p(prime[j]) ;
            isP[p*i] = 0 ;
            if(i%p == 0) {
                Mobius[i*p] = 0;
                phi[i*p]=phi[i]*p;
                break ;
            } else {
                 Mobius[i*p] = -Mobius[i] ;
                 phi[i*p]=phi[i]*(p-1) ;
            }
        }
    }
    for(int i(1);i<=prime[0];++i) {///算每一个质数的贡献 nlogn
        int p(prime[i]) ;
        for(int j(1);j*p<=MAXN;++j) {
            Sum[j*p] += Mobius[j];
        }
    }
    for(int i(1);i<=MAXN;++i) {//求贡献的前缀和
        Sum[i] += Sum[i-1] ;
    }
}

int64_t calc(int N,int M)
{
    int64_t ans(0) ;
    int mins(min(N,M)) ;
    for(register int l(1),r;l<=mins;l = r+1) {
        r = min(N/(N/l),M/(M/l)) ;
        ans += (Sum[r]-Sum[l-1])*(N/l)*(M/l);
    }
```

```
    return ans;
}
int main()
{
    getPEx() ;
    int T ;
    scanf("%d",&T) ;
    while(T--)
    {
        int n,m ;
        scanf("%d%d",&n,&m) ;
        printf("%lld\n",calc(n,m)) ;
    }
}
```

(14)大素数判断+大数分解质因数+大数阶乘分解

```
#include <bits/stdc++.h>
#define in32(x) (scanf("%d",&x))
#define in64(x) (scanf("%lld",&x))
#define out32(x) (printf("%d",x))
#define out64(x) (printf("%lld",x))
#define en putchar('\n')
#define sp putchar(' ')
using namespace std ;
const int inf(0x3f3f3f3f) ;
const int64_t INF(0x7FFFFFFFFFFFFFFF) ;
const int MAXN(100005) ;

int64_t Alpha(int64_t p,int64_t n) ///阶乘分解质因数
{
    int64_t ans(0),b(1) ;///这个 b 变量是防止结果溢出的
    for(int64_t base(p);base/p == b&&base<=n;base*=p,b*=p) {
        ans += n/base ;
    }
    return ans ;
}

int64_t mulEx(int64_t a,int64_t b,int64_t Mod) {///logn 快速乘
    if(!a) return 0;
    int64_t ans(0) ;
    while(b) {
        if(b&1) ans = (ans+a)%Mod;
        a <<= 1 ; a%=Mod ;
        b >>= 1;
```

```
    }
    return ans ;
}

int64_t powEx(int64_t base,int64_t n,int64_t Mod) {///快速幂
    int64_t ans(1);
    while(n) {
        if(n&1) ans = mulEx(ans,base,Mod);
        base = mulEx(base,base,Mod);
        n >>= 1;
    }
    return ans;
}

bool check(int64_t a,int64_t d,int64_t n) {
    if(n == a) return true;
    while(~d&1) d >>= 1;
    int64_t t = powEx(a,d,n);
    while(d < n-1 && t != 1 && t != n-1) {
        t = mulEx(t,t,n);
        d <<= 1;
    }
    return (d&1) || t == n-1;
}

bool isP(int64_t n) { ///判断大数是否是质数
    if(n <= 1) return false ;
    if(n == 2) return true;
    if(n < 2 || 0 == (n&1)) return false;
    static int p[5] = {2,3,7,61,24251};
    for(int i = 0; i < 5; ++i)
        if(!check(p[i],n-1,n)) return false;
    return true;
}

int64_t gcd(int64_t a,int64_t b) {
    if(a < 0) return gcd(-a,b);
    return b?gcd(b,a-b*(a/b)):a;
}

int64_t Pollard_rho(int64_t n,int64_t c) {///大数分解质因数

     int64_t i = 1,k = 2,x = rand()%n,y = x;
        while(true) {
```

```
            x = (mulEx(x,x,n) + c)%n;
            int64_t d = gcd(y - x,n);
            if(d != 1 && d != n) return d;
            if(y == x) return n;
            if(++i == k) {
                y = x;
                k <<= 1;
            }
        }
}
int64_t Fac[MAXN],factCnt;
///Fac 存的是质因子,大小不一定按照顺序,有重复
void factorization(int64_t n) {
        if(n <= 1)return ;
        if(isP(n)) {
            Fac[factCnt++] = n;
            return;
        }
        int64_t p (n);
        while(p >= n) p = Pollard_rho(p,rand()%(n-1)+1);
        factorization(p);
        factorization(n/p);
}

map<int64_t,int64_t>factMap;
///遍历 map 的 first 表示因子,second 表示次数

void getFactor(int64_t x)
{///不用判断是否是质数,但是比较费时间
 /**因此最好先判断一下是否是质数**/
        srand(time(0));
        factCnt = 0 ;
        factMap.clear();
        factorization(x);
        for(int i = 0; i < factCnt; ++i) ++ factMap[Fac[i]];
}

int main()
{
        int64_t n,m ;
        in64(n),in64(m) ;
        if(isP(m)) {/**这里特判一下省时间**/
            out64(Alpha(m,n)) ;en ;
            return 0;
```

```
        }
        getFactor(m) ;
        int64_t mins(INF) ;
        for(int i(0);i<factCnt;++i)
            mins = min(mins,Alpha(Fac[i],n)/factMap[Fac[i]]) ;
        out64(mins);en ;
    }
```

(15)多项式求值 O(N)

```
#define mod(x) ((((x)%Mod)+Mod)%Mod)
//多项式求值,系数数组 COEF[i]表示 x^i 的系数,n 代表最高次数,x 代表要求的值.
int64_t Horner(int64_t* COEF,int n,int64_t x,int64_t Mod=::Mod)
{
    int64_t ans(mod(COEF[n])) ;
    for(int i(n-1);i>=0;--i) {
        ans = mod(ans*x) ;
        ans = mod(ans+COEF[i]) ;
    }
    return mod(ans) ;
}
```

(16)二次剩余(模数为质数)O(log^2n)

```
int64_t sqrtMod(int64_t a,int64_t Mod=::Mod)
{
    int64_t b,k,i,x ;
    if(a < 0||a >= Mod) a = ((a%Mod) +Mod) %Mod ;
    if(a == 0) return 0 ;
    if(Mod == 2) return a%Mod ;
    if(powEx(a,(Mod-1)>>1,Mod) != 1) return -1 ;//欧拉判别
    if(Mod%4 == 3) x = powEx(a,(Mod+1)>>2,Mod) ;
    else {
        for(b = 1;powEx(b,(Mod-1)>>1,Mod) == 1;++b) ;
        i = (Mod-1)>>1 ; k = 0 ;
        do {
            i >>= 1;    k >>= 1 ;
            if((powEx(a,i,Mod)*powEx(b,k,Mod)+1)%Mod) continue ;
            k += (Mod-1)>>1    ;
        }while((i&1) == 0) ;
        x = powEx(a,(i+1)>>1,Mod)*powEx(b,k>>1,Mod) % Mod ;
    }
    if((x<<1)>Mod) x = Mod - x ; //最多两个,返回最小的二次剩余.
    return x ;
}
```

(17)二阶递推的解析表达式

对于已知$F(1),F(2)$的二阶线性数列$F(n)=a*F(n-1)+b*F(n-2),(n\geq 3)$
我们先计算方程$x^2-ax-b=0$的解,
如果该方程有二实根p,q，则
$F(n)=Ap^n+Bq^n, A=\frac{F(2)-qF(1)}{p(p-q)}, B=\frac{pF(1)-F(2)}{q(p-q)}$ ----------①
如果该方程只有一实根p，则
$F(n)=(A+bn)p^n, A=\frac{pF(1)-F(2)}{p^2}, B=\frac{F(2)-pF(1)}{p^2}$ ----------②

(18)反素数(dfs)

```
///反素数: 1 到 n 之内因子数目最多的数
int64_t p[20]= {0,2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53};
int64_t maxn=-1,ans=-1;
int64_t n=0;

void get(int64_t m,int64_t f,int64_t t,int64_t pr) {
	if(t>maxn || (t==maxn && m<ans)) {
		ans=m,maxn=t;
	}
	int64_t i=m,j=0;
	int64_t nt=0;
	while(j<pr) {
		j++;
		if(n/i<p[f]) {
			break;
		}
		nt=t*(j+1),i*=p[f];
		if(i<=n) {
			get(i,f+1,nt,j);
		}
	}
}

int main()
{
	while(~in64(n))
	{
		get(1,1,1,30);
		out64(ans) ;
		en ;
	}
	return 0 ;
```

```
}
```

(19)分数类

```
template <typename T>
class Fract
{
typedef long double LD ;
public:
    static const int BIT_MAX = sizeof(T)*8;
    static const T MAX = numeric_limits<T>::max() ;
    static const T SQR = ((T)sqrt(MAX))>>1 ;
    T zi,mu ;
    inline int get_sign() {return ((zi>>(BIT_MAX-1))^(mu>>(BIT_MAX-1)))?-1:1;}
    Fract(T _zi=0,T _mu=1):zi(_zi),mu(_mu){check();}
    inline void gcd() {T g(abs(__gcd(zi,mu)));zi /= g;mu /= g ;}
    inline void check()
    {
        zi = abs(zi)*get_sign(); mu = abs(mu) ;
        if(mu == 0||zi == 0) {zi = 0;mu = 1;}
        if(zi>=SQR||mu=>SQR) gcd() ;
    }
    inline Fract operator!() {return {-zi,mu};}
    inline Fract operator~() {return {mu,zi};}
    inline Fract operator+(Fract b) {return {zi*b.mu+b.zi*mu,mu*b.mu} ;}
    inline Fract operator-(Fract b) {return *this+!b;}
    inline Fract operator*(Fract b) {return {zi*b.zi,mu*b.mu};}
    inline Fract operator/(Fract b) {return *this*~b;}
    inline friend ostream& operator<<(ostream &out,Fract& f)
        {f.gcd();return out<<f.zi<<'/'<<f.mu;}
    inline friend istream& operator>>(istream& in,Fract& f)
        {in>>f.zi>>f.mu;f.check();return in;}
    inline bool operator<(const Fract b) const {return (LD)zi/mu<(LD)b.zi/b.mu;}
    inline bool operator==(Fract b){return (LD)zi*b.mu==(LD)mu*b.zi;}
    inline bool operator<=(Fract b) {return (*this)<b||(*this)==b;}
    inline bool operator>(Fract b) const {return !((*this)<=b);}
    inline bool operator>=(Fract b) {return !((*this)<b);}
};
typedef Fract<int64_t> Fen ;
```

(20)根据分数取模的结果和模数还原分数

```
void f(int64_t la,int64_t lb,int64_t ra,int64_t rb,int64_t& x,int64_t& y)
{///求 la/lb < x/y < ra/rb  的 x,y 中最小的 x 和 y
    int64_t mins(la/lb+1);
```

```
    if(mins<=ra/rb) {
        x=mins; y=1; return ;
    }
    -- mins;///意为 floor(la/lb)
    la -= mins*lb; ra -= mins*rb;
    f(rb,ra,lb,la,y,x);
    x += mins*y;
    return ;
}
void getFraction(int64_t a,int64_t Mod,int64_t& x,int64_t& y)
{///给出 x/y%Mod = a 的结果 a 和 Mod,反求出最小的 x 和 y.
    int64_t b ;
    f(Mod,a,Mod,a-1,y,b);
    x=y*a-b*Mod;
}
```

(21)广义斐波那契循取模环节

```
/**
题意:
    F[n] = a*F[n-1]+b*F[n-2], F[0] = x,F[1] = y,求 F[n]%Mod,n 是一个 1000000 长的字符串.
思路:
    求出 F[n]的循环节,F[n]的循环节只与 a,b 有关,然后矩阵快速幂直接算即可.

有关 F[n]循环节的相关结论:
    1. 如果 F[n]%Mod 有循环节为 m,那么(F[n]^k)%Mod 也有循环节 m.
    2. 如果 F[n]%Mod 有循环节为 m,那么(F[n]^k)%Mod 的前 n 项和也有循环节 m.
    3. 如果知道 Mod 的因子的循环节,那么 Mod 的循环节就是因子的循环节的 LCM.
    4. 如果模数不是质数,那么我们可以把循环节分解质因数成质因数的幂次形式,然后求每个质因数幂次的循环节,
        得到结果以后用中国剩余定理合并.
*/
#include <bits/stdc++.h>
using namespace std ;
constexpr int N = 2 ; ///矩阵的长
int64_t MOD;
struct Matrix {int64_t a[N][N] ;};
/// 矩阵快速幂不是求循环节用的,可以不抄.
const Matrix E = {1,0,0,1};
Matrix operator*(const Matrix& a,const Matrix& b)
{
    Matrix ret ;
    for(int i(0);i<N;++i)
```

```
            for(int j(0);j<N;++j) {
                ret.a[i][j] = 0 ;
                for(int k(0);k<N;++k)
                    ret.a[i][j] = (ret.a[i][j]+(1LL*a.a[i][k]*b.a[k][j])%MOD) % MOD ;
            }
        return ret ;
}
Matrix powEx(Matrix base,int64_t n)
{
        Matrix ret = E ;
        while(n) {
            if(n&1) ret = ret * base ;
            base = base * base ;
            n >>= 1 ;
        }
        return ret ;
}
/****************矩阵快速幂结束*******************/
inline int64_t mulEx(int64_t a,int64_t b,int64_t Mod) {
        int64_t ret(0) ;
        while(a) {
            if(a&1) {
                ret = (ret + b) ;
                if(ret >= Mod) ret %= Mod ;
            }
            b <<= 1;    a >>= 1 ;
            if(b>=Mod) b %= Mod ;
        }
        return ret ;
}
int64_t powEx(int64_t base,int64_t n,int64_t Mod)
{
        int64_t ret(1) ;
        while(n) {
            if(n&1) ret = mulEx(ret,base,Mod) ;
            base = mulEx(base,base,Mod);
            n >>= 1 ;
        }
        return ret ;
}
inline int64_t Legendre(int64_t a,int64_t p,int64_t Mod)
{/**判断是否是二次剩余*/
        return (powEx(a,(p-1)>>1,Mod)==1)?1:-1;
}
```

```
unordered_map<int64_t,int> factMap ;
inline void getFact(int64_t x)
{///暴力 O(sqrt(n))分解质因数,这里也可以用更快的素数筛.
	factMap.clear() ;
	for(int64_t i(2);i*i<=x; ++i) {
		while(x%i==0) {
			++factMap[i] ; x/=i;
		}
	}
	if(x>1) ++ factMap[x];
}
inline int64_t lcm(int64_t a,int64_t b) {return a/__gcd(a,b)*b ;}
int64_t getLoopLen(int64_t a,int64_t b,int64_t Mod) {
	getFact(Mod);
	int64_t ans(1),c(a*a+4*b);
	for(auto&it:factMap) {
		int64_t p(1),fact(it.first),cnt(it.second),flag(-1) ;
		if(fact == 2) {
			flag = 1 ; p *= 3 ;
		} else if(c%fact == 0) {
			p *= fact * (fact-1) ;
		} else if(Legendre(c,fact,fact) == 1) {
			p *= (fact-1) ;
		} else {
			p *= (fact-1) * (fact+1);
		}
		for(int j(1);j <= cnt+flag;++j) p *= fact ;
		ans = lcm(ans,p) ;
	}
	return ans;
}
char str[1000005] ;
int main()
{
	int64_t x,y,a,b ;
	scanf("%lld%lld%lld%lld%s%lld",&x,&y,&a,&b,str,&MOD) ;
	int64_t loopLen(getLoopLen(a,b,MOD)) ; ///找到取模循环节
	int len(strlen(str));
	int64_t n = 0 ;
	for (int i = 0; i < len; ++i) {///这里直接乘 10 会爆,用快速乘.
		n = (mulEx(n,10,loopLen) + str[i] - '0')%loopLen;
	}
	Matrix base = {static_cast<int64_t>(a),static_cast<int64_t>(b),1,0} ,
		Ans = {static_cast<int64_t>(y),0,static_cast<int64_t>(x),0} ;
```

```
    printf("%lld\n",(powEx(base,n)*Ans).a[1][0]) ;///矩阵快速幂求结果
}
```

(22)卡特兰数(前 n 项和)取模(O(n))

```
/**
        卡特兰数(前 n 项和)对非质数取模
    模数非质数时,我们可以把模数分解质因数,然后得到质因数的幂次乘积的形式.
然后我们求每个幂次乘积作为模数得到的结果,然后中国剩余定理合并即可.
*/
#include <bits/stdc++.h>
using namespace std;
int64_t n,Mod ;
int64_t factor[20][2],m[20] ;
int fCnt ;
void getFact(int64_t x)
{
    fCnt = 0 ;
    for(int i(2);i*i<=x;++i) {
        if(x%i) continue ;
        factor[fCnt][0] = i ;
        factor[fCnt][1] = 0 ;
        m[fCnt] = 1 ;
        for(;!(x%i);x /= i,m[fCnt] *= i) ++ factor[fCnt][1] ;
        ++ fCnt ;
    }
    if(x != 1) m[fCnt] = factor[fCnt][0] = x,factor[fCnt++][1] = 1 ;
}
int64_t a[20] ;
int64_t gcdEx(int64_t a, int64_t b, int64_t&x, int64_t& y)
{
    if(!b) return x = 1,y = 0,a ;
    int64_t d(gcdEx(b,a%b,y,x));
    return y -= (a/b)*x,d;
}
int64_t Inv(int64_t _x,int64_t Mod)
{
    int64_t x,y ;
    return gcdEx(_x,Mod,x,y),(x<0)?(x+Mod):x ;
}
int64_t China_Reminder(int len, int64_t* a, int64_t* m)
{
    int64_t M(m[0]),ans(a[0]);
    for(int i(1); i < len; ++i) {
```

```
            int64_t m_i(m[i]),x,y,d(gcdEx(M,m_i,x,y));
            int64_t z(((a[i]-ans)%m_i+m_i)%m_i) ;
            if(z%d) return -1 ;
            z /= d ; m_i /= d ;
            x = ((x*z)%m_i+m_i)%m_i;
            ans += M*x ; M *= m_i;
        }
        return ans;
}
int64_t powEx(int64_t base,int64_t n,int64_t Mod)
{
        int64_t ret(1) ;
        while(n>0) {
            if(n&1) ret = ret * base % Mod ;
            base = base * base %Mod ;
            n >>= 1 ;
        }
        return ret ;
}
int64_t calc(int n,int64_t u,int64_t M)
{
        int64_t ret(1),Cntu(0)/**这里记录质数 u 出现的次数*/,Sum(1) ;
        for(int i(2);i <= n;++i) {
            int64_t zi(4*i - 2),mu(i + 1) ;
            while(!(zi % u)) ++ Cntu , zi /= u ;
            while(!(mu % u)) -- Cntu , mu /= u ;
            ///不互质不能求逆元,所以要把 u 作为因子的情况分离出来单算.
            ret = (ret * zi)%M ; ret = (ret * Inv(mu,M))%M ;
            Sum += ret*powEx(u,Cntu,M) % M;
            Sum %= M ;
        }
        return Sum ;///如果算的不是前 n 项和的话,直接返回 ret*powEx(u,Cntu,M)%M 即可.
}
int main()
{
        while(~scanf("%d%d",&n,&Mod))
        {
            if(!n&&!Mod) break ;
            getFact(Mod) ;
            for(int i(0);i<fCnt;++i) {
                a[i] = calc(n,factor[i][0],m[i]) ;
            }
            printf("%lld\n",China_Reminder(fCnt,a,m)) ;
        }
```

```
}
```

(23)开根号向下取整 O(logn)

```
int64_t sqrtInt(int64_t x)
{
    if(x == 1) return 1 ;
    int64_t i(x>>1),last(-1),j ;
    for(;(j=(i + x/i)>>1) != last;) {last = i;i = j;}
    return j ;
}
```

(24)拉格朗日插值[点不连续]O(n^2)

```
//拉格朗日插值：给出 n-1 次多项式 f(x)的 n 个点(X[i],Y[i]){下标从 1 开始},以及 x,求 f(x)%Mod
/*时间复杂度:O(n^2),n 是多项式的最高次数,一般用于多项式打表之后丢进去算结果*/
int64_t Lagrange(int64_t* X,int64_t* Y,int n,int64_t x,int64_t Mod = :: Mod)
{
    int64_t ans(0) ;
    for(int i(1);i<=n;++i) {
        int64_t s1(Y[i]%Mod),s2(1LL);
        for(int j(1);j<=n;++j) {
            if(i!=j) {
                int64_t dx1(mod(x-X[j])),mod(X[i]-X[j]);
                s1 = (s1*dx1)%Mod ; s2 = (s2*dx2)%Mod;
            }
        }
        ans += (s1*Inv(s2))%Mod; ans %= Mod ;//Inv 是逆元
    }
    return ans ;
}
```

(25)类欧几里得算法 O(logn)

类欧几里得算法是一种时间复杂度为 O(logn)的求解某种求和式子的算法,仅仅是时间复杂度和欧几里得算法相同,和欧几里得算法其实并没有任何关系.

类欧几里得算法主要是求解下列三种式子:

① $f(a,b,c,n) = \sum_{i=1}^{n} \left\lfloor \frac{a*\mathrm{i}+\mathrm{b}}{c} \right\rfloor$

② $g(a,b,c,n) = \sum_{i=1}^{n} \left\lfloor \frac{a*\mathrm{i}+\mathrm{b}}{c} \right\rfloor^2$

③ $h(a,b,c,n) = \sum_{i=1}^{n} i * \left\lfloor \frac{a*\mathrm{i}+\mathrm{b}}{c} \right\rfloor$

如果是单独求 f,则可以用下边的代码来求:

```
/*
///在模 1e9+7 下的 Inv2 = 500000004,Inv6 = 166666668
///在模下的 Inv2 = 499122177,Inv6 = 166374059
*/
/**基础类欧几里得算法：用来求 likegcd(a,b,c,n) = ∑(i=0,n){floor((a*i+b)/c)}*/
int64_t likegcd(int64_t a, int64_t b, int64_t c, int64_t n) {
    if(a<0) a = -a ;
     if (a == 0)   return (n+1)*(b/c);
    if (a >= c || b >= c) {
        int64_t tmp = n&1?(n+1)/2*n:n/2*(n+1) ;//防止溢出,膜东西的话需要改成*inv(2)
        int64_t x = ((a/c)*tmp),y = ((b/c)*(n+1)) ;
        return likegcd(a%c,b%c,c,n)+x+y;
    }
    int64_t m = (a*n+b)/c;
    return (n*m-likegcd(c,c-b-1,a,m-1));
}
```

f,g,h 其实也可以放在一起 O(logn)一下子求出来,用下边的代码:

```
///下边是完全类欧几里得算法
struct Node{int64_t f,g,h;};
class Like_Euclid
{
public:
    int64_t Mod,Inv2,Inv6;
    int64_t Sqr(int64_t x) {x%=Mod;return x*x%Mod;}
    void setMod(int64_t Mod)
    {
        this->Mod = Mod ;
        Inv2 = 500000004;Inv6 = 166666668;
    }
    Node solve(int64_t a,int64_t b,int64_t c,int64_t N)
    {
        Node ret;
        if(!a) {
            ret.f=b/c*(N+1)%Mod;
            ret.g=Sqr(b/c)*(N+1)%Mod;
            ret.h=b/c*N%Mod*(N+1)%Mod*Inv2%Mod;
        } else if(a>=c||b>=c) {
            Node d=solve(a%c,b%c,c,N);
            ret.f=(a/c*N%Mod*(N+1)%Mod*Inv2+b/c*(N+1)+d.f)%Mod;
            ret.g=(d.g+N*(N+1)%Mod*(N*2ll+1)%Mod*Inv6%Mod*Sqr(a/c)
                +(N+1)*Sqr(b/c)+N*(N+1)%Mod*(a/c)%Mod*(b/c)
                +b/c*2LL*d.f+a/c*2LL%Mod*d.h)%Mod;
            ret.h=(d.h+N*(N+1)%Mod*(N*2ll+1)%Mod*Inv6%Mod*(a/c)
                +N*(N+1)%Mod*Inv2%Mod*(b/c))%Mod;
```

```
        } else {
            int64_t M=(a*N+b)/c;
            Node d=solve(c,c-b-1,a,M-1);
            ret.f=(N*M-d.f)%Mod;
            ret.g=(M*N%Mod*(M+1)-ret.f+2*(Mod*2ll-d.h-d.f)+Mod)%Mod;
            ret.h=(M*N%Mod*(N+1)-d.g-d.f+Mod+Mod)%Mod*Inv2%Mod;
        }
        return ret;
    }
}likeGcd;
```

类欧几里得的经典问题:

$$\sum_{i=1}^{N} A*i+B \text{ 的二进制第 i 位的贡献.}$$

我们***先拆位***考虑每一位的贡献，原式子可以表示成

$$\sum_{i=1}^{n}\left(\sum_{j=1}^{BIT\_MAX}(((Ai+B)\gg j)\%2)\ll j\right)$$

由于 $x\%2=\left\lfloor\frac{x}{2}\right\rfloor*2$,所以中间处理一下:

$$\sum_{i=1}^{n}\left(\sum_{j=1}^{BIT\_MAX}\left(\left(\left\lfloor\frac{Ai+B}{2^j}\right\rfloor\right)*2^j-\left(\left\lfloor\frac{Ai+B}{2^{j+1}}\right\rfloor\right)*2^{j+1}\right)\right)$$

然后交换求和顺序:

$$\sum_{j=1}^{BIT\_MAX}\left(\sum_{i=1}^{n}\left(\left(\left\lfloor\frac{Ai+B}{2^j}\right\rfloor\right)*2^j-\left(\left\lfloor\frac{Ai+B}{2^{j+1}}\right\rfloor\right)*2^{j+1}\right)\right)$$

然后合并同类项:

$$\sum_{j=1}^{BIT\_MAX}\left(\sum_{i=1}^{n}\left(\left(\left\lfloor\frac{Ai+B}{2^j}\right\rfloor\right)-\left(\left\lfloor\frac{Ai+B}{2^{j+1}}\right\rfloor\right)*2\right)*2^j\right)$$

例题 1: (https://ac.nowcoder.com/acm/contest/889/I)

$$\sum_{i=1}^{N}((i*M)\&M)\%1000000007$$

思路: 暴力枚举 M 的每一位,如果该位为 0,那么直接 continue,如果该位为 1,那么按照上边说过的办法做即可. 最后算一下该位存在的贡献.

主函数代码(类欧几里得里边要取模):

```
int main()
{
    int64_t n,m,ans(0) ;
    read(n);read(m) ;
    for(int i=0;i<=40;++i)
    if((1LL<<i)&m)
(ans+=(((likegcd(m,0,1LL<<i,n)+2*Mod-(likegcd(m,0,1ll<<(i+1),n))<<1)%Mod)<<i)%Mod)%=Mod;
    write(ans) ;puts("") ;
    return 0;
}
```

例题 2:

有 t≤10000 组询问，每组询问给出 n,a,b≤1e9，求:

$$a \oplus (a+b) \oplus (a+2b) \ldots\ldots \oplus (a+(n-1)b)$$

思路:

异或和就是不进位的加法,所以式子中算每一位的贡献的时候要加上一个%2,变成:

$$\sum_{j=1}^{BIT\_MAX}\left(\sum_{i=1}^{n}\left(\left(\left\lfloor\frac{Ai+B}{2^j}\right\rfloor\right)-\left(\left\lfloor\frac{Ai+B}{2^{j+1}}\right\rfloor\right)*2\right)\right)\%2*2^j$$
$$=\sum_{j=1}^{BIT\_MAX}\left(\sum_{i=1}^{n}\left(\left(\left\lfloor\frac{Ai+B}{2^j}\right\rfloor\right)\right)\%2*2^j\right)$$

(26)离散对数 O(ksqrt(n))

```
/// 离散对数: 用来求解 a^x = b (% Mod) 的解 x.
/// 取模不要求是质数,但是要求 gcd(a,Mod) == 1.
/// 离散对数的时间复杂度为 sqrt(Mod)*k,k 是 hash_map 的常数.
unordered_map<int64_t,int64_t> h ;
int64_t logMod(int64_t a,int64_t b,int64_t Mod)
{
    a %= Mod; b %= Mod ; h.clear() ;
    int64_t m(ceil(sqrt((long double)Mod))),x,y,d,t(1),v(1) ;
    for(int64_t i(0);i<m;++i,t = (t*a)%Mod)
        if(h.count(t)) h[t] = min(h[t],i); else h[t] = i ;
    for(int64_t i(0);i<m;++i,v = (v*t)%Mod) {
        d = gcdEx(v,Mod,x,y) ;
        x = (x*b/d%Mod+Mod)%Mod ;
        if(h.count(x)) return i*m+h[x] ;
    }
    return -1 ;
}

/// 拓展离散对数: 用来求解 a^x = b (% Mod) 的解 x.
/// 拓展离散对数参数可以任意.
/// 离散对数的时间复杂度为 sqrt(Mod)*k,k 是 hash_map 的常数.
int64_t logModEx(int64_t a,int64_t b,int64_t Mod)
{
    int64_t d(__gcd(a,Mod)),k(0),e(1) ;
    while(d>1) {
        if(b%d) return -1 ;
        ++k ; b/=d ; Mod/=d ; e = e*(a/d) %Mod ;
        if(e == b) return k ;
        d = __gcd(a,Mod) ;
    }
    int64_t ret(logMod(a,b*Inv(e,Mod)%Mod,Mod)) ;
    return (ret==-1)?-1:(ret+k);
}
```

(27)连续拉格朗日插值[点连续]O(n)

```
#define mod(x) ((((x)%Mod)+Mod)%Mod)
//已知 F(0),F(1),F(2)...F(n)的值为 Y[0],Y[1],Y[2]...Y[n]求任意项
/*时间复杂度:一次 O(n),n 是多项式的最高次数*/
int64_t Lagrange(int64_t* Y,int n,int64_t x,int64_t Mod = :: Mod)
{
    static int64_t fact,pre[MAXN],suf[MAXN],factInv[MAXN] ;
    fact = 1,pre[0] = x,suf[n] = mod(x-n);
    for(int i(1);i<=n;++i) {
        fact = fact*i %Mod ;
        pre[i] = (pre[i-1]*mod(x-i))%Mod ;
    }
    factInv[n] = Inv(fact,Mod) ;
    for(int i(n-1);i>=0;--i) {
        factInv[i] = factInv[i+1]*(i+1) % Mod ;
        suf[i] = (suf[i+1] * mod(x-i))%Mod ;
    }
    int64_t ans(0) ;
    for(int i(0);i<=n;++i) {
        int64_t s1 =   (i?pre[i-1]:1)*(i<n?suf[i+1]:1)%Mod;
        int64_t s2 =   (factInv[i]*factInv[n-i]) %Mod;
        int s3 = (Y[i]*((s1*s2)%Mod))%Mod ;
        ans += (n-i)&1?Mod-s3:s3;
        ans %= Mod ;
    }
    return ans ;
}
```

(28)莫比乌斯反演常用推导

(luoguP2257 YY 的 GCD)求

$$\sum_{p\in prime}\sum_{i=1}^{n}\sum_{j=1}^{m}[\gcd(i,j)==p]$$

解: $\sum_{p\in prime}\sum_{i=1}^{n}\sum_{j=1}^{m}[\gcd(i,j)==p]$

由于$\gcd(i,j)==p <=> p|i$ 且 p|j,所以我们可以直接枚举 p 的倍数 k*p 中的 k.

$=\sum_{p\in prime}\sum_{i=1}^{\lfloor n/p\rfloor}\sum_{j=1}^{\lfloor m/p\rfloor}[\gcd(i,j)==1]$

$=\sum_{p\in prime}\sum_{i=1}^{\lfloor n/p\rfloor}\sum_{j=1}^{\lfloor m/p\rfloor}\varepsilon(\gcd(i,j))$

; 利用$[f(x)==1]=\varepsilon(f(x))$以及反演公式: $\varepsilon=\mu*1$

$=\sum_{p\in prime}\sum_{i=1}^{\lfloor n/p\rfloor}\sum_{j=1}^{\lfloor m/p\rfloor}(\mu*1)(\gcd(i,j))$; 利用狄利克雷卷积的定义展开

= $\sum_{p\in prime}\sum_{i=1}^{\lfloor \frac{n}{p} \rfloor}\sum_{j=1}^{\lfloor \frac{m}{p} \rfloor}\sum_{d|(\gcd(i,j))}\mu(d)$

;由于 d 的取值范围是[1,min $\left(\left\lfloor \frac{n}{p} \right\rfloor, \left\lfloor \frac{m}{p} \right\rfloor\right)$]所以我们枚举 d,看看每一个$\mu$(d)有多少贡献.由于 d|gcd (i,j),所以 i,j 都是 d 的倍数,这样中间的两个和式相当于问[1, $\left\lfloor \frac{n}{p} \right\rfloor$]和[1, $\left\lfloor \frac{m}{p} \right\rfloor$]中 d 的倍数有多少个, 所以上限都除以 d.

= $\sum_{p\epsilon \text{prime}}\sum_{d=1}^{\min\left(\lfloor \frac{n}{p} \rfloor, \lfloor \frac{m}{p} \rfloor\right)}\mu(d)\sum_{i=1}^{\lfloor \frac{n}{dp} \rfloor}\sum_{j=1}^{\lfloor \frac{m}{dp} \rfloor}1 = \sum_{p\epsilon \text{prime}}\sum_{d=1}^{\min\left(\lfloor \frac{n}{p} \rfloor, \lfloor \frac{m}{p} \rfloor\right)}\mu(d)\left\lfloor \frac{n}{dp} \right\rfloor\left\lfloor \frac{m}{dp} \right\rfloor$

;化简到这里,我们发现时间复杂度还是不够,如果我们要是知道每一个素数的贡献就好了,所以我们尝试把$\mu(d)$提出来,把它和枚举素数放到一起.设 T=dp

= $\sum_{p\epsilon \text{prime}}\sum_{d=1}^{\min\left(\lfloor \frac{n}{p} \rfloor, \lfloor \frac{m}{p} \rfloor\right)}\mu(d)\left\lfloor \frac{n}{T} \right\rfloor\left\lfloor \frac{m}{T} \right\rfloor$; 然后枚举 T

= $\sum_{T=1}^{\min(n,m)}\left\lfloor \frac{n}{T} \right\rfloor\left\lfloor \frac{m}{T} \right\rfloor\sum_{p\epsilon prime,p|T}\mu(\frac{T}{p})$

整除分块

```
int64_t calc(int64_t N,int64_t M)
{
    int64_t ans(0) ;
    int mins(min(N,M)) ;
    for(int64_t l(1),r;l<=mins;l = r+1) {
        r = min(a/(a/l),b/(b/l)) ;///有几个写几个.
        ans += ((int64_t)Sum[r]-Sum[l-1])*(a/(l))*(b/(l));
    }
    return ans;
}
```

(Uva11426)给定 N,M 求

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\gcd(i,j)$$

解: $\sum_{i=1}^{n}\sum_{j=1}^{m}\gcd(i,j)$

= $\sum_{d=1}^{\min(n,m)}d\sum_{i=1}^{n}\sum_{j=1}^{m}[\gcd(i,j)==d]$

= $\sum_{d=1}^{\min(n,m)}d\sum_{i=1}^{\lfloor \frac{n}{d} \rfloor}\sum_{j=1}^{\lfloor \frac{m}{d} \rfloor}[\gcd(i,j)==1]$

= $\sum_{d=1}^{\min(n,m)}d\sum_{i=1}^{\lfloor \frac{n}{d} \rfloor}\sum_{j=1}^{\lfloor \frac{m}{d} \rfloor}\varepsilon(gcd(i,j))$

= $\sum_{d=1}^{\min(n,m)}d\sum_{i=1}^{\lfloor \frac{n}{d} \rfloor}\sum_{j=1}^{\lfloor \frac{m}{d} \rfloor}(\mu*1)(gcd(i,j))$

= $\sum_{d=1}^{\min(n,m)}d\sum_{i=1}^{\lfloor \frac{n}{d} \rfloor}\sum_{j=1}^{\lfloor \frac{m}{d} \rfloor}\sum_{g|gcd(i,j)}\mu(g)$

= $\sum_{d=1}^{\min(n,m)}d\sum_{g=1}^{\min\left(\lfloor \frac{n}{d} \rfloor, \lfloor \frac{m}{d} \rfloor\right)}\mu(g)\left\lfloor \frac{n}{dg} \right\rfloor\left\lfloor \frac{m}{dg} \right\rfloor$;设 dg = T;

$= \sum_{d=1}^{\min(n,m)} d \sum_{g=1}^{\min(\lfloor \frac{n}{d} \rfloor, \lfloor \frac{m}{d} \rfloor)} \mu(g) \lfloor \frac{n}{T} \rfloor \lfloor \frac{m}{T} \rfloor$

$= \sum_{T=1}^{\min(n,m)} \sum_{d|T} d\mu(\frac{T}{d}) \lfloor \frac{n}{T} \rfloor \lfloor \frac{m}{T} \rfloor$

$= \sum_{T=1}^{\min(n,m)} \lfloor \frac{n}{T} \rfloor \lfloor \frac{m}{T} \rfloor \sum_{d|T} d\mu(\frac{T}{d})$ ；利用 Id*μ = φ继续化简

$$= \sum_{T=1}^{\min(n,m)} \left\lfloor \frac{n}{T} \right\rfloor \left\lfloor \frac{m}{T} \right\rfloor \varphi(T)$$

(29)欧拉降幂 O(TlognSqrt(n))

```
/**
    广义欧拉降幂 O(T*logn*sqrt(n))
    求 a^a^a^a^a^a^.....^a[b 个 a] % Mod
    广义欧拉降幂公式(a 和 Mod 互质或者不互质都可以):
        a^c % Mod =  a ^ ((c%phi[Mod])+phi[Mod])  [c >= phi[Mod]]
    当 c<phi[Mod]时候,只能暴力.
*/
#include<bits/stdc++.h>
#define sp putchar(' ')
#define en putchar('\n')
using namespace std;
int64_t powEx(int64_t base,int64_t n,int64_t Mod=0)
{
    int64_t ret(1) ;
    while(n) {
        if(n&1) {ret *= base ;if(Mod) ret %= Mod ;}
        base *= base ;
        if(Mod) base %= Mod ;
        n >>= 1 ;
    }
    return Mod?ret%Mod:ret;
}

int64_t euler(int64_t n)
{
        int64_t ret(n),x(n);
        for(int64_t i(2);i*i <= x;++i){
            if(x%i==0){
                ret=ret/i*(i-1);//先进行除法是为了防止中间数据的溢出
                while(x%i==0) x/=i;
            }
```

```
        }
        if(x>1) ret = ret / x * (x-1);
        return ret;
}

int64_t a,b,Mod;

int64_t calc(int deep)
    {return (deep == 1)?a:powEx(a,calc(deep-1)) ;}

long double calcD(int deep)
    {return (deep == 1)?a:pow(a,calcD(deep-1)) ;}

bool check(int deep,int64_t x)
    {return (deep >= 5)?true:calcD(deep) > x ;}

int64_t dfs(int deep,int64_t Mod)
{
    if(Mod == 1) return 0 ;
    if(deep == 1) return a % Mod;
    int64_t phi(euler(Mod)) ;
    if(deep == 2)
        return a>phi?powEx(a,a%phi+phi,Mod):powEx(a,a,Mod) ;
    return check(deep-1,phi)
        ?powEx(a,dfs(deep-1,phi)+phi,Mod)
        :powEx(a,calc(deep-1),Mod) ;
}

int main()
{
    int T ;
    scanf("%d",&T) ;
    while(T-->0)
    {
        scanf("%lld%lld%lld",&a,&b,&Mod) ;
        if(a == 1||b == 0)
            {printf("%lld\n",1%Mod) ; continue ;}
        if(b == 1)
            {printf("%lld\n",a%Mod) ; continue ;}
        printf("%lld\n",dfs(b,Mod)) ;
    }
}
```

(30)普通高斯消元.O(n^3)

```
/*
    普通高斯消元: 用来解方程,时间复杂度 O(n^3)
*/
#include <bits/stdc++.h>
#define IS_ZERO(x) (abs(x)<=eps)
const double eps (1e-8) ;
using namespace std ;

//传参接口: A:二维数组,NxN 的系数矩阵,从 0 开始算,
//最后一列(第 N 列存增广条件矩阵)
/*
N = 2
A[0] A[1] A[2]
  1    1   | 2
  3    3   | 6
*/
bool Gauss(auto A,int n) {
    int rMatrix(0),rBound(0);//矩阵的秩
    for(int i = 0; i < n; ++i) {
        int r = i;
        for(int j = i + 1; j < n; ++j)
            if(abs(A[j][i]) > abs(A[r][i])) r = j;
        if(r != i)
            for(int j = 0; j <= n; ++ j)
                swap(A[r][j], A[i][j]);
        bool flag = 0 ;
        for(int j = n; j >= i; -- j) {
            for(int k = i + 1; k < n; ++k)
                A[k][j] -= A[k][i] / A[i][i] * A[i][j];
            if(!IS_ZERO(A[i][j])) {
                if(j == n) ++rBound ;
                else flag = 1 ;
            }
        }
        if(flag) ++ rMatrix ;
    }
    for(int i = n - 1; i >= 0; --i) {
        for(int j = i + 1; j < n; ++j)
            A[i][n] -= A[j][n] * A[i][j] ;
        A[i][n] /= A[i][i];
    }
    //方程有唯一解的条件是系数矩阵的秩等于增广矩阵的秩
```

```
    //并且都等于 n.
    return rMatrix == rBound&&rBound == n;
}
double A[1005][1005] ;
int main()
{
    int n ;
    scanf("%d",&n) ;
    for(int i(0);i<n;++i) {
        for(int j(0);j<=n;++j) {
            scanf("%lf",&A[i][j]) ;
        }
    }
    bool flag = Gauss(A,n) ;
    if(!flag) {
        puts("No Solution") ;
        return 0 ;
    }
    for(int i(0);i<n;++i) {
        printf("%.2f\n",A[i][n]) ;
    }
}
```

(31) 普通矩阵求逆 O(n^3)

```
#include<bits/stdc++.h>
using namespace std;
const int MAXN = 505 ;
const int64_t Mod = 1e9+7;
int64_t a[MAXN][MAXN];
int n,is[MAXN],js[MAXN];
int64_t gcdEx(int64_t a,int64_t b,int64_t &x,int64_t &y){
    if(!b)return x=1,y=0,a;
    int64_t d(gcdEx(b,a%b,y,x));
    return y-=x*(a/b),d;
}
int64_t Inv(int64_t _x){
    int64_t x,y; gcdEx(_x,Mod,x,y);
    return (x+Mod)%Mod;
}

bool MatrixInv(int64_t A[][MAXN],int n){
    for(int k(1);k<=n;++k){
        for(int i(k);i<=n;++i)
```

```
            for(int j(k);j<=n;++j)
                if(A[i][j]) {is[k]=i;js[k]=j;break;}
        for(int i(1);i<=n;++i) {
            swap(A[k][i],A[is[k]][i]) ;swap(A[i][k],A[i][js[k]]) ;
        }
        if(!A[k][k]) return false ;
        A[k][k]=Inv(A[k][k]);
        for(int i(1);i<=n;++i) if(i!=k) (A[k][i]*=A[k][k])%=Mod;
        for(int i(1);i<=n;++i) if(i!=k)
            for(int j(1);j<=n;++j) if(j!=k)
                (A[i][j]+=Mod-A[i][k]*A[k][j]%Mod)%=Mod;
        for(int i(1);i<=n;++i) if(i!=k)
            A[i][k]=(Mod-A[i][k]*A[k][k]%Mod)%Mod;
    }
    for(int k(n);k;--k){
        for(int i(1);i<=n;++i) {
            swap(A[js[k]][i],A[k][i]) ;swap(A[i][is[k]],a[i][k]) ;
        }
    }
    return true ;
}
int main(){
    scanf("%d",&n);
    for(int i(1);i<=n;++i)
        for(int j(1);j<=n;++j)
            scanf("%lld",a[i]+j);
    bool f = MatrixInv(a,n);
    if(!f) {puts("No Solution") ;return 0;}
    for(int i(1);i<=n;++i)
        for(int j(1);j<=n;++j)
            printf("%lld%c",a[i][j],j==n?'\n':' ');
    return 0;
}
```

(32) 普通拓展 BM[模数可以不为质数](可以输出递推式子)

```
///BM: 解决递推式.请保证模数的平方不会爆 long long!!!
///不用全抄,需要那一部分,就抄哪一部分.
using VI = vector<int64_t> ;
class Linear_Seq
{
public:
    static const int N = 50010;///多项式系数最大值
    int64_t res[N],c[N],md[N],COEF[N]/**COEF 是多项式系数*/,Mod;
    vector<int> Md;
```

```
inline static int64_t gcdEx(int64_t a, int64_t b, int64_t&x, int64_t& y)
{
    if(!b) {x=1;y=0;return a;}
    int64_t d = gcdEx(b,a%b,y,x);
    y -= (a/b)*x;
    return d;
}

inline static int64_t Inv(int64_t a, int64_t Mod) {
    int64_t x, y;
    return gcdEx(a, Mod, x, y)==1?(x%Mod+Mod)%Mod:-1;
};

inline void mul(int64_t *a,int64_t *b,int k) {///下边的线性齐次递推用的.
    fill(c,c+2*k,0) ;
    for(int i(0);i<k;++i)if(a[i])for(int j(0);j<k;++j)
        c[i+j]=(c[i+j]+a[i]*b[j])%Mod;
    for (int i(2*k-1);i>=k;--i) if (c[i])for(size_t j(0);j<Md.size();++j)
        c[i-k+Md[j]]=(c[i-k+Md[j]]-c[i]*md[Md[j]])%Mod;
    copy(c,c+k,a) ;
}

int64_t solve(int64_t n,VI A,VI B) { //线性齐次递推:A 系数,B 初值 B[n]=A[0]*B[n-1]+...
    ///这里可以可以单独用,给出递推系数和前几项代替矩阵快速幂求递推式第 n 项.
    int64_t ans(0),cnt(0);
    int k(A.size());
    for(int i(0);i<k;++i) md[k-i-1]=-A[i];
    md[k]=1 ;   Md.clear() ;
    for(int i(0);i<k;++i) {
        res[i] = 0 ;
        if (md[i]) Md.push_back(i);
    }
    res[0]=1;
    while ((1LL<<cnt)<=n) ++ cnt;
    for (int p(cnt);~p;-- p) {
        mul(res,res,k);
        if ((n>>p)&1) {
            copy(res,res+k,res+1) ; res[0]=0;
            for(size_t j(0);j<Md.size();++j)
                res[Md[j]]=(res[Md[j]]-res[k]*md[Md[j]])%Mod;
        }
    }
    for(int i(0);i<k;++i) ans=(ans+res[i]*B[i])%Mod;
```

```
    return ans+(ans<0?Mod:0);
}

///1-st**********模数是质数用这里*****************/
VI BM(VI s) {///BM 算法求模数是质数的递推式子的通项公式,可以单独用
    VI C(1,1),B(1,1);
    int L(0),m(1),b(1);
    for(size_t n(0);n<s.size();++n) {
        int64_t d(0);
        for(int i(0);i<=L;++i) d=(d+(int64_t)C[i]*s[n-i])%Mod;
        if (!d) ++m;
        else {
            VI T(C);
            int64_t c(Mod-d*Inv(b,Mod)%Mod);
            while (C.size()<B.size()+m) C.push_back(0);
            for (size_t i(0);i<B.size();++i)
                C[i+m]=(C[i+m]+c*B[i])%Mod;
            if (2*L<=(int)n) {L=n+1-L; B=T; b=d; m=1;}
            else ++m ;
        }
    }
    /** ///下边这样写能够输出递推式的系数.
    printf("F[n] = ") ;
    for(size_t i(0);i<C.size();++i) {
        COEF[i+1] = min(C[i],Mod-C[i]) ;
        if(i>0) {
            if(i != 1) printf(" + ") ;
            printf("%lld*F[n-%d]",COEF[i+1],i) ;
            putchar(i+1==C.size()?'\n':' ') ;
        }
    }
    */
    return C;
}
///1-ed**********模数是质数用这里*****************/

///2-st**********模数非质数用这里*****************/
inline static void extand(VI &a, size_t d, int64_t value = 0) {
    if (d <= a.size()) return; a.resize(d, value);
}
static int64_t CRT(const VI &c, const VI &m) {///中国剩余定理合并
    int n(c.size());
    int64_t M(1), ans(0);
    for (int i = 0; i < n; ++i) M *= m[i];
```

```
    for (int i = 0; i < n; ++i) {
        int64_t x,y,tM(M / m[i]);
        gcdEx(tM, m[i], x, y);
        ans = (ans + tM * x * c[i] % M) % M;
    }
    return (ans + M) % M;
}

static VI ReedsSloane(const VI &s, int64_t Mod) {///求模数不是质数的递推式系数
    auto L = [](const VI &a, const VI &b) {
        int da = (a.size()>1||(a.size()== 1&&a[0]))?a.size()-1:-1000;
        int db = (b.size()>1||(b.size()== 1&&b[0]))?b.size()-1:-1000;
        return max(da, db + 1);
    };
    auto prime_power = [&](const VI &s, int64_t Mod, int64_t p, int64_t e) {
        // linear feedback shift register Mod p^e, p is prime
        vector<VI> a(e), b(e), an(e), bn(e), ao(e), bo(e);
        VI t(e), u(e), r(e), to(e, 1), uo(e), pw(e + 1);;
        pw[0] = 1;
        for (int i(pw[0] = 1); i <= e; ++i) pw[i] = pw[i - 1] * p;
        for (int64_t i(0); i < e; ++i) {
            a[i] = {pw[i]}; an[i] = {pw[i]};
            b[i] = {0}; bn[i] = {s[0] * pw[i] % Mod};
            t[i] = s[0] * pw[i] % Mod;
            if (!t[i]) {t[i] = 1; u[i] = e;}
            else for (u[i] = 0; t[i] % p == 0; t[i] /= p, ++u[i]);
        }
        for (size_t k(1);k < s.size(); ++k) {
            for (int g(0); g < e; ++g) {
                if (L(an[g], bn[g]) > L(a[g], b[g])) {
                    int id (e-1-u[g]);
                    ao[g] = a[id]; bo[g] = b[id];
                    to[g] = t[id]; uo[g] = u[id];
                    r[g] = k - 1;
                }
            }
            a = an; b = bn;
            for (int o(0); o < e; ++o) {
                int64_t d(0);
                for (size_t i(0); i < a[o].size() && i <= k; ++i)
                    d = (d + a[o][i] * s[k - i]) % Mod;
                if (d == 0) {t[o] = 1;u[o] = e;}
                else {
                    for (u[o]=0,t[o]=d;!(t[o]%p);t[o]/=p,++u[o]);
```

```
                int g (e-1-u[o]);
                if (!L(a[g], b[g])) {
                    extand(bn[o], k + 1);
                    bn[o][k] = (bn[o][k] + d) % Mod;
                } else {
                    int64_t coef = t[o]*Inv(to[g],Mod)%Mod*pw[u[o]-uo[g]]%Mod;
                    int m(k-r[g]);
                    extand(an[o],ao[g].size()+m); extand(bn[o],bo[g].size()+m);
                    auto fun = [&](vector<VI> &vn,vector<VI> &vo,bool f) {
                        for (size_t i(0);i < vo[g].size(); ++i) {
                            vn[o][i+m] -= coef*vo[g][i]%Mod;
                            if (vn[o][i + m]<0) vn[o][i+m] += Mod*(f?1:-1);
                        }
                        while (vn[o].size() && !vn[o].back()) vn[o].pop_back();
                    } ;
                    fun(an,ao,1) ;fun(bn,bo,-1) ;
                }
            }
        }
    }
    return make_pair(an[0], bn[0]);
};
vector<tuple<int64_t, int64_t, int> > fac;
for (int64_t i(2); i*i <= Mod; ++i)
    if (!(Mod % i)) {
        int64_t cnt(0),pw(1);
        while (!(Mod % i)) {Mod /= i; ++cnt; pw *= i;}
        fac.emplace_back(pw, i, cnt);
    }
if (Mod > 1) fac.emplace_back(Mod, Mod, 1);
vector<VI> as;
size_t n = 0;
for (auto &&x: fac) {
    int64_t Mod, p, e;
    VI a, b;
    std::tie(Mod, p, e) = x;
    auto ss = s;
    for (auto &&x: ss) x %= Mod;
    std::tie(a, b) = prime_power(ss, Mod, p, e);
    as.emplace_back(a);
    n = max(n, a.size());
}
VI a(n),c(as.size()),m(as.size());
for (size_t i(0); i < n; ++i) {
```

```
            for (size_t j(0); j < as.size(); ++j) {
                m[j] = std::get<0>(fac[j]);
                c[j] = i < as[j].size() ? as[j][i] : 0;
            }
            a[i] = CRT(c, m);
        }
        return a;
    }
    ///2-ed**********模数非质数用这里******************/

    int64_t solve(VI a,int64_t n,int64_t Mod,bool prime=true) {
        VI c; this->Mod = Mod ;
        if(prime) c = BM(a);///如果已经知道系数了,直接输入到 c 就行了,不用调用 BM().
        else c = ReedsSloane(a,Mod);
        c.erase(c.begin()) ;
        for(size_t i(0);i<c.size();++i) c[i] = (Mod-c[i])%Mod;
        return solve(n,c,VI(a.begin(),a.begin()+c.size()));
    }
}BMEX;
///BMEX.slove(初始值 vector[从 0 开始],要得到的项数,模数,模数是不是质数)
```

(33)求 Pi E 以及大数开根号

```
class Sqrt
{///大数开根号
    static BigDecimal ONE = BigDecimal.ONE ;
    static BigDecimal TWO = ONE.add(ONE) ;
    static BigDecimal ZERO = BigDecimal.ZERO ;
    static BigDecimal calc(BigDecimal x,int scale)
    {
        BigInteger pows = BigInteger.TEN.pow(scale) ;
        BigDecimal EPS = ONE.divide(new
BigDecimal(pows.toString()),scale+10,RoundingMode.DOWN) ;
        BigDecimal mid=ZERO,l = ZERO,r = x ;
        while(r.subtract(l).compareTo(EPS)>0) {
            mid = l.add(r).divide(TWO,scale+1,RoundingMode.DOWN) ;
            if(mid.multiply(mid).compareTo(x)>0) r = mid ;
            else l = mid ;
        }
        return mid ;
    }
};

class Pi
```

```
{//10000 位左右
    static BigDecimal ONE = BigDecimal.ONE ;
    static BigDecimal TWO = ONE.add(ONE) ;
    static BigDecimal ZERO = BigDecimal.ZERO ;
    static BigDecimal calc(int scale)
    {
        BigInteger pows = BigInteger.TEN.pow(scale+2) ;
        BigDecimal EPS = ONE.divide(new
BigDecimal(pows.toString()),scale+2,RoundingMode.DOWN) ;
        BigDecimal ans = TWO,eps = TWO ;
        int n=1,m=3;
        while(eps.compareTo(EPS) > 0){
            eps = eps.multiply(BigDecimal.valueOf(n));
            eps = eps.setScale(scale+2, RoundingMode.DOWN) ;
            eps = eps.divide(BigDecimal.valueOf(m),scale+5,RoundingMode.DOWN) ;
            ans = ans.add(eps) ;
            ans = ans.setScale(scale+2, RoundingMode.DOWN) ;
            ++ n; m += 2 ;
        }
        return ans;
    }
};

class E
{///300 位左右
    static BigDecimal ONE = BigDecimal.ONE ;
    static BigDecimal TWO = ONE.add(ONE) ;
    static BigDecimal ZERO = BigDecimal.ZERO ;

    static BigDecimal calc(int scale)
    {
        BigInteger pows = BigInteger.TEN.pow(scale+2) ;
        BigDecimal EPS = ONE.divide(new
BigDecimal(pows.toString()),scale+2,RoundingMode.DOWN) ;
        BigDecimal ans = ONE,eps = ONE ;
        int i = 1 ;
        while(eps.compareTo(EPS)>0) {
            ans = ans.add(eps) ;
            ans = ans.setScale(scale+2, RoundingMode.DOWN) ;
            eps =
eps.multiply(ONE.divide(BigDecimal.valueOf(++i),scale+10,RoundingMode.DOWN)) ;
        }
        return ans ;
    }
```

```
};
```

(34)求单个欧拉函数 O(sqrt(n))

```
int64_t euler(int64_t n)
{
        int64_t ret(n),x(n);
        for(int64_t i(2);i*i<=x;i++){
            if(x%i==0){
                ret=ret/i*(i-1);//先进行除法是为了防止中间数据的溢出
                while(x%i==0) x/=i;
            }
        }
       if(x>1) ret=ret/x*(x-1);
       return ret;
}
```

(35)十进制快速幂

```
const int N = 100 ; ///矩阵的长
const int Mod = 1000000007;
struct Matrix {int a[N][N] ;};
Matrix E ;
Matrix operator*(const Matrix& a,const Matrix& b)
{ ///这里可以循环展开优化
    Matrix ret ;
    for(int i(0);i<N;++i)
        for(int j(0);j<N;++j) {
            ret.a[i][j] = 0 ;
            for(int k(0);k<N;++k)
                (ret.a[i][j] += (1LL*a.a[i][k]*b.a[k][j])%Mod)%= Mod ;
        }
    return ret ;
}
Matrix powEx(Matrix base,int n)
{
    Matrix ret = E ;
    while(n) {
        if(n&1) ret = ret * base ;
        base = base * base ;
        n >>= 1 ;
    }
    return ret ;
}
Matrix powEx10(Matrix base,char* n)
```

```
{
    Matrix ret = E;
    int len(strlen(n)) ;
    for(int i(len-1);~i;--i) {
        ret = ret*powEx(base,n[i]-'0') ;
        base = powEx(base,10) ;
    }
    return ret ;
}
```

(36)拓展中国剩余定理(Java 大数版本)

```
class CRT
{
    static BigInteger x,y,buff ;
    static BigInteger[] M,A;//M[]为模数,A[]为余数,下标从 0 开始.
    static BigInteger ZERO = BigInteger.ZERO ;
    static BigInteger ONE = BigInteger.ONE ;
    static BigInteger NULLANS = new BigInteger("-1"); //不存在答案
    static BigInteger OVERFLOW = new BigInteger("-2"); //答案>limit
    static BigInteger gcdEx(BigInteger a,BigInteger b)
    {//拓展欧几里得
        buff = x ;x = y;y = buff ;
        if(b.compareTo(ZERO) == 0) {
            x = ONE ; y = ZERO ;
            return a ;
        }
        BigInteger gcd = gcdEx(b,a.remainder(b)) ;
        buff = x ;x = y;y = buff ;
        y = y.subtract((a.divide(b)).multiply(x)) ;
        return gcd ;
    }

    static BigInteger China_Reminder(int len,BigInteger limit)
    {///拓展中国剩余定理
        BigInteger _M = M[0],ans = A[0] ;
        BigInteger m_i,gcd,z ;
        if(limit!=null&&ans.compareTo(limit)>0) return OVERFLOW ;
        for(int i = 1;i<len;++i) {
            m_i = M[i] ;
            z = (A[i].subtract(ans.remainder(m_i)).add(m_i)).remainder(m_i) ;
            gcd = gcdEx(_M,M[i]) ;
            if(z.remainder(gcd).compareTo(ZERO)!=0) {
                return  NULLANS;
            }
```

```
                m_i=m_i.divide(gcd) ;
                x = (((x.multiply(z)).divide(gcd)).remainder(m_i).add(m_i)).remainder(m_i) ;
                ans = ans.add(_M.multiply(x)) ;
                _M = _M.multiply(m_i) ;
                if(limit!=null&&ans.compareTo(limit)>0) return OVERFLOW ;
            }
            return ans ;
        }
};
```

(37)拓展中国剩余定理(C++普通版本)

```
#include <bits/stdc++.h>
using namespace std;
int64_t a[15],b[15] ;
int64_t gcdEx(int64_t a, int64_t b, int64_t&x, int64_t& y)
{
    if(!b) return x = 1,y = 0,a ;
    int64_t d(gcdEx(b,a%b,y,x));
    return y -= (a/b)*x,d;
}
int64_t China_Reminder(int len, int64_t* a, int64_t* m)
{///求解模线性方程组 x=a[i] (%m[i]) (下标从 0 开始!!!!)
    int64_t M(m[0]),ans(a[0]);
    for(int i(1); i < len; ++i) {
        int64_t m_i(m[i]),x,y,d(gcdEx(M,m_i,x,y));
        int64_t z(((a[i]-ans)%m_i+m_i)%m_i) ;
        if(z%d) return -1 ;
        z /= d ; m_i /= d ;
        x = ((x*z)%m_i+m_i)%m_i;
        ans += M*x ; M *= m_i;
    }
    return ans;
}
int main()
{
    ios::sync_with_stdio(false) ;
    int T ;
    cin >>T;
    while(T--) {
        for(int i(0);i<3;++i) cin >>b[i] ;
        for(int i(0);i<3;++i) cin >>a[i] ;
        int64_t x_3 = China_Reminder(3,a,b);
        if(x_3<0) {
            puts("-1") ;
```

```
                continue ;
            }
            for(int i(0);1LL*i*i*i<=x_3;++i) {
                if(1LL*i*i*i == x_3) {
                    cout<<i<<endl;
                    break ;
                }
            }
        }
    return 0;
}
/*
7
1328881 1307377 1989929
1064198 685339 595270
746321 753377 1624751
497700 685946 639383
1385141 1237729 1037843
484277 69853 272320
1328881 1307377 1989929
1064198 685339 595270
746321 753377 1624751
497700 685946 639383
1385141 1237729 1037843
484277 69853 272320
1328881 1307377 1989929
1064198 685339 595270

749417
54623
1011236
749417
54623
1011236
749417
*/
```

(38)线段树套线性基 O(nlogn+log^3n)

```
#include<bits/stdc++.h>
#define int64_t int
using namespace std ;
/**
    线性基一般用来解决区间子集异或问题.
    线性基一般使用普通递归式线段树来维护,输入数据在线段树 build 内.
```

这样保证了合并线性基只用合并 logn 次.
保证了建树的时间复杂度为 O(nlogn+log^3n)查询的时间复杂度为 O(mlog^2n).
*/
```
class Node
{
#define N 32
public:
    int64_t base[N] ;
    int tot,Size ;/**tot:线性基内数的个数(线性基的秩);Size:线性基的个数*/
    Node() {clear() ;}
    inline void clear()/**清空线性基*/ {tot = Size = 0;memset(base,0,sizeof(base));}
    inline void set() {///初始化为满秩
        tot = 1<<N;Size = N;
        for(int i(0);i<N;++i) base[i] = 1ULL << i;
    }
    inline bool insert(int64_t x)
    {///插入一个数到线性基
        ++ tot ;
        for(int i(N-1);~i;--i) {
            if((x>>i)&1) {
                if(base[i]) x ^= base[i] ;
                else {base[i] = x ;++Size;return true;}
            }
        }
        return false ;
    }
    inline void insert(Node b)
    {///插入另一个线性基到当前线性基
        for(int i(N-1);~i;--i)
            if(b[i]) insert(b[i]) ;
        tot -= b.Size ;tot += b.tot;
    }
    void intersect(Node& a,Node& b)
    {///线性基求交
        //clear() ;
        unsigned e[N],d[N];
        for(int i(0);i<N;++i) e[i] = d[i] = 0 ;
        for(int i(N-1);~i;--i) {
            e[i] = a[i];d[i] = (1ULL<<i);
        }
        for(int i(N-1);~i;--i) {
            int64_t v (b[i]),k(0);
            if(v) {
                bool isok(true);
```

```
                for(int j(N-1);~j;--j) {
                    if((v>>j)&1) {
                        if(e[j]) {
                            v ^= e[j]; k ^= d[j];
                        } else {
                            isok = false; e[j] = v; d[j] = k; break;
                        }
                    }
                }
                if(isok) {
                    int64_t v(0);
                    for(int j(N-1);~j;--j)    if((k>>j)&1)    v ^= a[j];
                    insert(v);
                }
            }
        }
}
inline void gauss()
{///高斯消元,化成上三角形式.(查询第 K 小前必需先高斯消元)
    for(int i(N-1);~i;--i)
        for(int j(i-1);~j;--j)
            if ((base[i]>>j)&1) base[i]^=base[j];
}
inline int64_t Kth(int64_t k)
{///查询第 K 小(别忘了先高斯消元)
    /*
    ///包括 0;
    if(tot>Size) {
        if(k == 1) return 0;
        -- k ;
    }
    */
    if (k>=(1LL<<Size)) return -1 ;
    int64_t ret(0);
    for (int i(N-1);~i;--i)
        if ((k>>i)&1) ret ^= base[i];
    return ret;
}
inline bool find(int64_t x)
{///判断线性基是否能够表示出 x
    if(Size == N) return 1 ;
    if(!x) return 1 ;
    /*
    if(!x) return tot>Size ;  包括 0
```

```
            */
            for(int i(N-1);~i;--i) {
                if((x>>i)&1) x ^= base[i];
                if(!x) return true ;
            }
            return false ;
        }
        inline int64_t QMax()
        {///查询最大值
            int64_t ans(0) ;
            for(int i(N-1);~i;--i) {///注意括号!!!
                if((ans^base[i])>ans) ans ^= base[i] ;
            }
            return ans ;
        }
        inline int64_t QMin()
        {///查询最小值
            for(int i(0);i<N;++i) if(base[0]) return base[0] ;
            return 0 ;
        }
        inline int64_t& operator[](int i) {return base[i];}
        inline void operator ^=(Node& b) {insert(b);}///线性基求并
        inline void operator()(Node &a,Node& b) {intersect(a,b) ;}///线性基求交
#undef N
};

template <class T>
void read(T &x) {
    static char ch;static bool f;
    for(ch=f=0;ch<'0' || '9'<ch;f|=ch=='-',ch=getchar());
    for(x=0;'0'<=ch && ch<='9';(x*=10)+=ch-'0',ch=getchar());
    x=f?-x:x;
}

const int MAXN (50005) ;
class STree
{
#define L (pos<<1)
#define R (pos<<1|1)
public:
    Node T[MAXN<<2] ;
    int LD[MAXN<<2],RD[MAXN<<2] ;

    void build(int l,int r,int pos = 1)
```

```
    {
        LD[pos] = l;RD[pos] = r ;
        if(l == r) {
            T[pos].clear() ;
            int sz;int64_t x ;
            read(sz) ;
            for(int i(1);i<=sz;++i) {
                read(x) ;   T[pos].insert(x) ;
            }
            T[pos].gauss() ;   return ;
        }
        int mid((l+r)>>1) ;
        build(l,mid,L) ;   build(mid+1,r,R) ;
        T[pos](T[L],T[R]) ;   T[pos].gauss() ;
    }

    bool Q(int l,int r,int64_t k,int pos = 1)
    {
        if(l == LD[pos] && r == RD[pos]) return T[pos].find(k) ;
        int mid((LD[pos]+RD[pos])>>1) ;
        if(r<=mid) return Q(l,r,k,L) ;   else if(l>mid) return Q(l,r,k,R) ;
        else return Q(l,mid,k,L)&&Q(mid+1,r,k,R) ;
    }
};
STree tree ;
int main()
{
    int n,m,sz ;    read(n);read(m) ;
    int64_t x ;   Node tmp ;   tree.build(1,n) ;    int l,r ;
    while(m--) {read(l);read(r);read(x) ;   puts(tree.Q(l,r,x)?"YES":"NO") ;}
}
```

(39)一般分解质因数 O(ksqrtn)

```
int64_t factor[20][2] ;
int factCnt ;
void getFact(int64_t x)
{
    factCnt = 0 ;
    for(int i(2);i*i<=x;++i) {
        if(x%i) continue ;
        factor[factCnt][0] = i ;
        factor[factCnt][1] = 0 ;
        for(;!(x%i);x /= i) ++ factor[factCnt][1] ;
```

```
            ++ factCnt ;
        }
        if(x != 1) factor[factCnt][0] = x,factor[factCnt++][1] = 1 ;
}
```

10. 图论和网络流

(1) Astar 第 K 短路

```
#include <bits/stdc++.h>
#pragma comment(linker, "/STACK:102400000,102400000")
using namespace std;
typedef long long LL;
typedef pair<int, int> PII;
const int maxn = 1000 + 5;
const int INF = 0x3f3f3f3f;
int s, t, k;
bool vis[maxn];
int dist[maxn];
struct Node {
    int v, c;
    Node (int _v = 0, int _c = 0) : v(_v), c(_c) {}
    bool operator < (const Node &rhs) const {
        return c + dist[v] > rhs.c + dist[rhs.v];
    }
};
struct Edge {
    int v, cost;
    Edge (int _v = 0, int _cost = 0) : v(_v), cost(_cost) {}
};
vector<Edge>E[maxn], revE[maxn];
void Dijkstra(int n, int s) {
    memset(vis, false, sizeof(vis));
    for (int i = 1; i <= n; i++) dist[i] = INF;
    priority_queue<Node>que;
    dist[s] = 0;
    que.push(Node(s, 0));
    while (!que.empty()) {
        Node tep = que.top(); que.pop();
        int u = tep.v;
        if (vis[u]) continue;
        vis[u] = true;
        for (int i = 0; i < (int)E[u].size(); i++) {
```

```
                int v = E[u][i].v;
                int cost = E[u][i].cost;
                if (!vis[v] && dist[v] > dist[u] + cost) {
                    dist[v] = dist[u] + cost;
                    que.push(Node(v, dist[v]));
                }
            }
        }
}
int astar(int s) {
        priority_queue<Node> que;
        que.push(Node(s, 0)); k--;
        while (!que.empty()) {
            Node pre = que.top(); que.pop();
            int u = pre.v;
            if (u == t) {
                if (k) k--;
                else return pre.c;
            }
            for (int i = 0; i < (int)revE[u].size(); i++) {
                int v = revE[u][i].v;
                int c = revE[u][i].cost;
                que.push(Node(v, pre.c + c));
            }
        }
        return -1;
}
void addedge(int u, int v, int w) {
        revE[u].push_back(Edge(v, w));
        E[v].push_back(Edge(u, w));
}
int main() {
        int n, m, u, v, w;
        while (scanf("%d%d", &n, &m) != EOF) {
            for (int i = 0; i <= n; i++) {
                E[i].clear();
                revE[i].clear();
            }
            for (int i = 0; i < m; i++) {
                scanf("%d%d%d", &u, &v, &w);
                addedge(u, v, w);
            }
            scanf("%d%d%d", &s, &t, &k);
            Dijkstra(n, t);
```

```
            if (dist[s] == INF) {
                puts("-1");
                continue;
            }
            if (s == t) k++;
            printf("%d\n", astar(s));
        }
        return 0;
}
```

(2) dfs 求树的直径 O(n)

```
#include <bits/stdc++.h>
#define en putchar('\n')
#define sp putchar(' ')
using namespace std ;
const int inf(0x3f3f3f3f) ;
const int MAXN(100010) ;
int n,w,m,buff,arr[MAXN],tot=0;

struct Edge {int to,val,Next ;};

Edge edge[MAXN<<2] ;
int head[MAXN],dis[MAXN]/**dis[i]表示离 i 最远的距离大小*/ ;

void add_edge(int from,int to,int val=0)
{
    edge[tot].to = to;edge[tot].val =val ;
    edge[tot].Next = head[from];
    head[from] = tot ++;
}

void Init(int n=MAXN)
{
    tot = 0 ;
    for(int i(0);i<=n;++i) head[i] = -1 ;
}

int dfs(int pos=1,int fa=-1)
{
    dis[pos] = -inf ;
    int ans_p(-1),isok(0);
    for(int i(head[pos]);i!=-1;i=edge[i].Next) {
        int p(edge[i].to),val(edge[i].val) ;
        if(p == fa) continue ;
```

```
            isok = 1 ;
            int u(dfs(p,pos)) ;
            if(dis[p]+val>=dis[pos]) {
                dis[pos] = dis[p] + val ;
                ans_p = u ;
            }
        }
        if(isok) {return    ans_p ;}
        else {
            dis[pos] = 0 ;
            return pos ;
        }
}

int main()
{
        Init() ;
        int a,b,c;
        while(~scanf("%d%d%d",&a,&b,&c)) {
            add_edge(a,b,c) ;
            add_edge(b,a,c) ;
        }
        int u(dfs()) ;
        int v(dfs(u)) ;
        printf("%d\n",dis[u]) ;en ;
}
```

(3) dfs 求树的重心 O(n)

```
/*
```

树的重心也叫树的质心

定义：找到一个点,其所有的子树中最大的子树节点数最少,那么这个点就是这棵树的重心,删去重心后，生成的多棵树尽可能平衡。

换句话说，删除这个点后最大连通块（一定是树）的结点数最小。

树的重心的性质:

1. 树中所有点到某个点的距离和中，到重心的距离和是最小的，如果有两个距离和，他们的距离和一样。
2. 把两棵树通过一条边相连，新的树的重心在原来两棵树重心的连线上。
3. 一棵树添加或者删除一个节点，树的重心最多只移动一条边的位置。
4. 一棵树最多有两个重心，且相邻。
5. 以树的重心为根，所有的子树（不算整个树自身）的大小都不超过整个树大小的一半。
6. 把两个树通过一条边相连得到一个新的树，那么新的树的重心在连接原来两个树的重心的路径上。
7. 把一个树添加或删除一个叶子，那么它的重心最多只移动一条边的距离。

目前遇到的题型有:

1.路径和等于或小于等于 k 的点对（路径条数）。

2.路径和为某个数的倍数。

3.路径和为 k 且路径的边数最少。

4.路径和 mod M 后为某个值。

5.路径上经过不允许点的个数不超过某个值，且路径和最大。

一般点分治的做法是:

对于某个重心 u，先进入子树 v1，求解出 u 到子树 v1 所有节点的路径，然后进入子树 v2，进入时先统计答案，然后再统计相关值。每次进入新的子树时，先统计答案，这样每次计算的路径一定是和之前统计过子树的相连而成的，没有不合法的答案，所以不用删除。最后，依次递归进子树，找出重心递归求解。

*/

```
//POJ 1655
#include <stdio.h>
#include <iostream>
#include <vector>
using namespace std ;
const int MAXN = 100005 ;
const int inf = 0x3f3f3f3f ;
vector<int> Next[MAXN] ;
bool isok[MAXN] ;
int Size[MAXN],N,maxSon/**子树最大值*/,cPos/**重心节点*/ ;

void dfs(int pos,int fa=-1)
{
    Size[pos] = 1 ;
    int maxs(0),sz(Next[pos].size()) ;
    for(int i(0);i<sz;++i) {
        int to(Next[pos][i]);
        if(to == fa) continue ;
        dfs(to,pos) ; Size[pos] += Size[to] ;
        maxs = max(maxs,Size[to]) ;
    }
    maxs = max(maxs,N-Size[pos]) ;
    if(maxs<maxSon) {maxSon=maxs;cPos=pos;}
}

inline int getCore(int pos) ///pos:树上的一个点
{
    maxSon = inf ;dfs(pos) ;return cPos ;
}

inline void add_edge(int u,int v)
    {Next[u].push_back(v) ;}
```

```
inline void addedge(int u,int v)
    {add_edge(u,v);add_edge(v,u);}

inline void Init(int n)
{///n 是点的最大标号,必须一个不差.
    N = n ;
    for(int i(0);i<=n;++i)
        {Next[i].clear() ;isok[i] = 0 ;}
}

int main()
{
    int T,n,u,v ;
    scanf("%d",&T) ;
    while(T-->0)
    {
        scanf("%d",&n) ;Init(n) ;
        for(int i(0);i<n-1;++i) {
            scanf("%d%d",&u,&v) ;
            addedge(u,v) ;
        }
        int id = getCore(1) ;
        printf("%d %d\n",id,maxSon) ;
    }
}
```

(4) dfs 求图上环的个数 O(n)

```
#include <bits/stdc++.h>
using namespace std ;
const int MAXM (3000005) ;
const int MAXN (3000005) ;
struct Edge {int to,Next ;};
Edge edge[MAXM<<2] ;
int head[MAXN],tot ;
bool InStack[MAXN] ;
bool isHave[MAXN] ;
void add_edge(int u,int v)
{
    edge[tot].to = v; ;
    edge[tot].Next = head[u];
    head[u] = tot ++;
}
void addedge(int u,int v)
{
```

```
    add_edge(u,v) ;add_edge(v,u) ;
}
void Init(int n)
{
     tot = 0 ;
     for(int i(0);i<=n;++i)
         head[i] = -1,isHave[i] = InStack[i] = 0 ;
}
int dfs(int pos,int last=-1)
{
    InStack[pos] = 1 ;
    int ans(0) ;
    for(int i(head[pos]);~i;i=edge[i].Next) {
        int to(edge[i].to) ;
        if(i == (last^1)) continue ;
        if(InStack[to]) {
            //printf("%d -> %d\n",pos,to) ;
            ++ ans ;
            continue ;
        }
        ans += dfs(to,i) ;
    }
    return ans ;
}
int main()
{
    int u,v,n,m ;
    scanf("%d%d",&n,&m) ;
    Init(n+5) ;
    for(int i(1);i<=m;++i) {
        scanf("%d%d",&u,&v) ;
        addedge(u,v) ;
        isHave[u] = isHave[v] = 1 ;
    }
    int ans(0) ;
    for(int i(1);i<=n;++i)
    {
        if(InStack[i]) continue ;
        if(!isHave[i]) {puts("NO");/*不连通*/return 0;}
        ans += dfs(i)>>1 ;//无向图需要除以 2
    }
    ///cout<<ans<<endl ;
    puts(ans == 1?"FHTAGN!":"NO") ;
}
```

```
/*
9 11
1 2
1 3
2 3
2 4
4 5
5 4
4 6
4 8
1 8
8 7
8 9
*/
```

(5) Dijkstra[可输出路径+链式前向星](nlogn)

```
#include <bits/stdc++.h>
/*
    https://www.luogu.org/problemnew/show/P2384
*/
const int MAXN (100005) ;
const int MAXM (100005) ;
using namespace std ;
const int inf (0x3f3f3f3f) ;
const int64_t INF (0x3f3f3f3f3f3f3f3f) ;
int n ;

class Edge
{
public :
    int to,Next ;
    int64_t val ;
    Edge(int _to=0,int64_t _val=0):to(_to),val(_val),Next(-1){}
    bool operator <(const Edge &b) const   {return val>b.val;}
};

class Dijkstra
{
public:
    int isok[MAXN],pre[MAXN]/**pre 记录前驱节点*/,head[MAXN],tot,N/*点的总数*/;
    Edge edge[MAXM<<1] ;

    void add_edge(int from,int to,int64_t val = 0)
```

```
    {
        edge[++tot].Next = head[from] ;
        edge[tot].to = to ;edge[tot].val = val ;
        head[from] = tot ;
    }

    void clear(int n)
    {
        N = n ;
        for(int i(0);i<=N;++i) head[i] = -1 ;
        tot = 0 ;
    }

    int64_t dijkstra(int from,int to,int64_t* dis)
    {
        for(int i(0);i<=N;++i) {
            isok[i] = false ;
            dis[i] = INF ;
        }
        dis[from] = 0 ;
        priority_queue<Edge> que ;
        que.push(Edge(from,0)) ;
        while(que.size()) {
            int to(que.top().to) ;que.pop() ;
            if(isok[to]) continue ;
            isok[to] = true ;
            for(int j(head[to]);~j;j=edge[j].Next) {
                int x(edge[j].to) ;
                int64_t val(edge[j].val) ;
                if(dis[to] + val < dis[x]) {
                    dis[x] =    dis[to] + val ;
                    pre[x] = to ;
                    que.push(Edge(x,dis[x])) ;
                }
            }
        }
        return dis[to] ;
    }
}dijkstra ;

int64_t dis[MAXN] ;
int main()
{
    int T,m ;
```

```
    while(~scanf("%d%d",&n,&m))
    {
        if(!n||!m) break ;
        dijkstra.clear(n) ;
        for(int i(0);i<m;++i) {
            int from,to;
            int64_t val ;
            scanf("%d%d%lld",&from,&to,&val) ;
            dijkstra.add_edge(from,to,val) ;
            dijkstra.add_edge(to,from,val) ;
        }
        int64_t len(dijkstra.dijkstra(1,n,dis));
        ///printf("%lld\n",len) ;
        /**
        ///以下代码为输出路径:
        */
        if(len == INF) {puts("-1");continue ;}
        vector<int> v ;
        for(int p(n);p!=1;p=dijkstra.pre[p]) {
            v.push_back(p) ;
            //printf("%d <- %d\n",p,dijkstra.pre[p]);
        }
        v.push_back(1) ;
        for(int i(v.size()-1);i>=0;--i) {
            printf("%d",v[i]) ;
            putchar(i?' ':'\n') ;
        }
    }
}
```

(6) Dijkstra 优化费用流 O(FElogV)[邻接表]

```
#include <bits/stdc++.h>
#define double float
#define sp putchar(' ')
#define en putchar('\n')
using namespace std;
const int inf(0x3f3f3f3f);
const int64_t INF(((1LL*inf)<<32)+inf);
const int MAXN (30010) ;
template<typename T = int64_t>
class DinicEx//时间复杂度是 O(F*ElogV)(F 是流量,E 是边数,V 是顶点数)
{
public:
```

```
struct Edge{
	int to,rev,flow;
	T cost;
	Edge(int _to=0,T _val=0):to(_to),cost(_val){}
	Edge(int t,int r,int f,T c):to(t),rev(r),flow(f),cost(c){}
	bool operator < (const Edge& b) const {return cost>b.cost;}
} ;
vector<Edge> edge[MAXN] ;
int N/*点的总个数*/;
int pre[MAXN]/**前驱点编号*/,last[MAXN]/**前驱边编号*/ ;
T dis[MAXN]/**花费*/,H[MAXN]/**势能数组*/ ;

bool dijkstra(int s,int t)
{
	for(int i(0);i<=N+1;++i)
		dis[i] = INF,last[i] = -1 ;
	dis[s] = 0 ;pre[t] = -1 ;
	priority_queue<Edge> que ;
	que.push(Edge(s,0)) ;
	while(que.size()) {
		Edge tp = que.top() ; que.pop() ;
		int pos(tp.to),sz(edge[pos].size()) ;
		if(dis[pos]<tp.cost) continue ;
		for(int i(0);i<sz;++i) {
			Edge& e = edge[pos][i] ;
			int p(e.to),_flow(e.flow) ;
			T cost(e.cost) ;
			if(_flow&&dis[p]>dis[pos]+cost+H[pos]-H[p]) {
				dis[p] = dis[pos]+cost+H[pos]-H[p] ;
				last[p] = i; pre[p] = pos ;
				que.push(Edge(p,dis[p])) ;
			}
		}
	}
	return pre[t]!=-1 ;
}

inline void dinic(int s,int t,int& maxflow,T& mincost)
{
	mincost =  maxflow = 0 ;
	while (dijkstra(s,t)) {
		int f(inf) ;
		for (int i = 0; i <= N; ++i) H[i] += dis[i] ;
		for(int i(t);i!=s;i = pre[i]) {
```

```
                Edge&e = edge[pre[i]][last[i]] ;
                f = min(f,e.flow) ;
            }
            maxflow += f ; mincost += f*H[t] ;
            for(int i(t);i!=s;i = pre[i]) {
                Edge&e = edge[pre[i]][last[i]] ;
                e.flow -= f ;
                edge[i][e.rev].flow += f ;
            }
        }
    }

    inline void clear(int n)
    {///用之前别忘了 clear 一下
        for(int i(0);i<=n+1;++i) {edge[i].clear();H[i] = 0;}
        N = n ;
    }

    inline void addedge(int from,int to,int flow=1,T cost=0)
    {///加正向边和反向边(默认流量为 1,费用为 0)
        edge[from].push_back(Edge(to,edge[to].size(),flow,cost)) ;
        edge[to].push_back(Edge(from,edge[from].size()-1,0,-cost)) ;
    }
};
DinicEx<double> dinic ;
double arr[105][105] ;
int main()
{
    int n,buff ;
    scanf("%d",&n) ;
    int s(0),t(2*n+5) ;
    for(int i(1);i<=n;++i) {
        for(int j(1);j<=n;++j) {
            scanf("%d",&buff) ;
            arr[i][j] = -log(buff)/log(2) ;
        }
    }
    dinic.clear(1003) ;
    for(int i(1);i<=n;++i) {
        for(int j(1);j<=n;++j) {
            dinic.addedge(i,j+n,1,arr[i][j]) ;
        }
        dinic.addedge(s,i,1,0) ;
        dinic.addedge(i+n,t,1,0) ;
```

```
    }
    int maxflow;
    double mincost ;
    dinic.dinic(s,t,maxflow,mincost) ;
    for(int i(n+1);i<=n+n;++i) {
        for(int j(0);j<dinic.edge[i].size();++j) {
            if(dinic.edge[dinic.edge[i][j].to][dinic.edge[i][j].rev].flow == 0) {
                printf("%d%c",dinic.edge[i][j].to,(i == n+n)?'\n':' ') ;
            }
        }
    }
    return 0;
}
```

(7) Dinic+SPFA 费用流

```
#include <bits/stdc++.h>
using namespace std;
const int64_t INF(0x3f3f3f3f3f3f3f3f);
const int MAXN (100010) ;
const int MAXM (100010) ;
struct Edge{
    int Next,to;
    int64_t flow,cost;
} ;

class DinicEx
{
    public:
    Edge edge[MAXM<<1];///这里注意,一定是 2 倍,因为正边反边
    int tot,head[MAXN],N/*点的总个数*/;
    int pre[MAXN],last[MAXN]/**前驱边编号*/ ;
    int64_t dis[MAXN]/**花费*/,flow[MAXN] ;
    bool isok[MAXN] ;

    inline void add_edge(int from,int to,int64_t flow,int64_t cost = 0)
    {
        edge[tot].Next=head[from];
        edge[tot].to=to; edge[tot].flow=flow;edge[tot].cost = cost ;
        head[from]=tot++;
    }

    bool SPFA(int s,int t)
    {
```

```
    for(int i(0);i<=N;++i) {
        isok[i] = false ;
        dis[i] = flow[i] = INF ;
    }
    queue<int> que ;
    que.push(s) ;
    isok[s] = true ;dis[s] = 0;pre[t] =  -1 ;
    while(que.size()) {
        int pos = que.front() ;
        que.pop() ;
        isok[pos] = false ;
        for(int i(head[pos]);i!=-1;i = edge[i].Next) {
            int p(edge[i].to);
            int64_t _flow(edge[i].flow),cost(edge[i].cost) ;
            if(_flow&&dis[p]>dis[pos]+cost) {
                dis[p] = dis[pos]+cost ;
                last[p] = i;
                pre[p] = pos ;
                flow[p] = min(flow[pos],_flow) ;
                if (!isok[p]) {
                    isok[p] = true ;
                    que.push(p);
                }
            }
        }
    }
    return pre[t]!=-1;
}

inline void dinic(int s,int t,int64_t& maxflow,int64_t& mincost)
{
    mincost =  maxflow = 0 ;
    while (SPFA(s,t)) {
        maxflow += flow[t] ;
        mincost += flow[t]*dis[t] ;
        for(int i(t);i!=s;i = pre[i]) {
            edge[last[i]].flow -= flow[t] ;
            edge[last[i]^1].flow += flow[t] ;
        }
    }
}

inline void clear(int n)
{
```

```
        for(int i(0);i<=n;++i) head[i] = -1;
        N = n ;tot = 0 ;
    }

    inline void addedge(int from,int to,int64_t flow=1,int64_t cost=0)
    {
        add_edge(from,to,flow,cost); ///加正向边
        add_edge(to,from,0,-cost)   ;///加容量为 0 的反向边
    }
}dinic;


int main()
{
    int s/**源点**/,t/**汇点**/,n,m ;
    scanf("%d%d%d%d",&n,&m,&s,&t);
    dinic.clear(n) ;///!!!!这里一定要注意,不一定是 n,这里应该是点的总个数!!!!!!
    for (int i(1);i<=m;++i)
    {
        int u,v;
        int64_t flow,cost ;
        scanf("%d%d%lld%lld",&u,&v,&flow,&cost);
        dinic.addedge(u,v,flow,cost) ; ///添加一条有向边 u->v
    }
    int64_t maxflow/**存最大流**/,mincost/**存最小费用**/ ;
    dinic.dinic(s,t,maxflow,mincost) ;
    printf("%lld %lld\n",maxflow,mincost);
}
```

(8) Dinic+当前弧优化最大流

```
#include <bits/stdc++.h>
using namespace std;
const int inf(0x3f3f3f3f);
const int MAXN (100005) ;
const int MAXM (250000) ;
int n,m,k;
struct Edge{
    int Next,to; int64_t val;
} ;

class Dinic
{
public:
```

```
int tot,N/*总点数*/,deep[MAXN],head[MAXN],cur[MAXN];///cur 用于复制 head
Edge edge[MAXM<<1];///这里注意,一定是 2 倍,因为正边反边
inline void add_edge(int from,int to,int64_t val=1)
{
    edge[tot].Next=head[from];
    edge[tot].to=to; edge[tot].val=val;
    head[from]=tot++;
}

///bfs 用来分层
bool bfs(int s,int t)
{
    queue <int> que;
    for (int i=0; i<=N; ++i) {deep[i] = 0;cur[i]=head[i];}
    deep[s]=1;
    que.push(s);
    while (que.size()) {
        int pos(que.front()); que.pop();
        for (int i(head[pos]);~i; i=edge[i].Next) {
            int p(edge[i].to);
            int64_t val(edge[i].val) ;
            if (deep[p]||!val) continue ;
            deep[p]=deep[pos]+1;
            que.push(p);
        }
    }
    return deep[t] ;
}

///dfs 找增加的流的量
int64_t dfs(int pos,int t,int64_t limit)///limit 为源点到这个点的路径上的最小边权
{
    if (!limit || pos==t) return limit;
    int64_t flow(0),f;
    for (int i(cur[pos]); i!=-1; i=edge[i].Next) {
        int p(edge[i].to);
        int64_t val(edge[i].val) ;
        cur[pos]=i;
        if (deep[p]==deep[pos]+1 &&
            (f=dfs(p,t,min(limit,val)))) {
            flow+=f;
            limit-=f;
            edge[i].val-=f;
            edge[i^1].val+=f;
```

```
                    if (!limit) break;
                }
            }
            return flow;
        }

        inline int64_t dinic(int s,int t)
        {
            int64_t maxflow(0) ;
            while (bfs(s,t)) maxflow+=dfs(s,t,inf);
            return maxflow ;
        }
        inline void clear(int n)
        {
            for(int i(0);i<=n;++i) head[i] = -1;
            N = n ;tot = 0 ;
        }
        inline void addedge(int from,int to,int64_t flow=1)
        {
            add_edge(from,to,flow); ///加正向边
            add_edge(to,from,0)   ;///加容量为 0 的反向边
        }

}dinic;
int main()
{
        int s/**源点**/,t/**汇点**/ ;
        scanf("%d%d%d%d",&n,&m,&s,&t);
        dinic.clear(n) ;///这里要注意,不一定是 n,应该是点的总数!!!!!!!!
        for (int i(1);i<=m;++i)
        {
            int u,v,flow ;
            scanf("%d%d%d",&u,&v,&flow);
            dinic.addedge(u,v,flow) ;///添加一条有向边 u -> v
        }
        printf("%lld\n",dinic.dinic(s,t));
}
```

(9) HLPP 高效最大流[Dinic 被卡用这个]

```
template <class T = int>
struct HLPP {
        static const int MAXN = 10005 ;
        const T INF = numeric_limits<T>::max();
```

```
struct Edge {int to,rev;T f;};
vector<Edge> adj[MAXN];
deque<int> lst[MAXN];
vector<int> gap[MAXN];
int ptr[MAXN];
T excess[MAXN];
int highest, height[MAXN], cnt[MAXN], work,N;
void addedge(int from, int to, int f=1, bool isDirected = true) {
	adj[from].push_back({to,adj[to].size(),f});
	adj[to].push_back({from,adj[from].size()-1,isDirected?0:f});
}
void clear(int n) {
	N = n ; for(int i(0);i<=n;++i) adj[i].clear() ;
}
void updHeight(int v, int nh) {
	++ work;
	if (height[v] != N) --cnt[height[v]];
	height[v] = nh;
	if (nh == N) return;
	cnt[nh]++, highest = nh;
	gap[nh].push_back(v);
	if (excess[v] > 0)
		{lst[nh].push_back(v); ++ptr[nh];}
}
void globalRelabel(int s,int t) {
	work = 0;
	fill(height,height+N+1, N);
	fill(cnt,cnt+N+1, 0);
	for (int i = 0; i <= highest; ++i)
		{lst[i].clear();gap[i].clear();ptr[i] = 0;}
	height[t] = 0;
	queue<int> q({t});
	while (!q.empty()) {
		int v = q.front(); q.pop();
		for (auto &e : adj[v])
			if (height[e.to] == N && adj[e.to][e.rev].f > 0)
				{q.push(e.to);updHeight(e.to, height[v] + 1);}
		highest = height[v];
	}
}
void push(int v, Edge &e) {
	if (excess[e.to] == 0)
		{lst[height[e.to]].push_back(e.to);++ptr[height[e.to]];}
	T df = min(excess[v], e.f);
```

```
        e.f -= df; adj[e.to][e.rev].f += df;
        excess[v] -= df; excess[e.to] += df;
    }
    void discharge(int v) {
        int nh = N;
        for (auto &e : adj[v]) {
            if (e.f > 0) {
                if (height[v] == height[e.to] + 1) {
                    push(v, e);
                    if (excess[v] <= 0) return;
                } else
                    nh = min(nh, height[e.to] + 1);
            }
        }
        if (cnt[height[v]] > 1) {
            updHeight(v, nh);
        } else {
            for (int i = height[v]; i < N; i++) {
                for (auto j : gap[i]) updHeight(j, N);
                gap[i].clear(); ptr[i] = 0;
            }
        }
    }
    T hlpp(int s,int t) {
        fill(excess,excess+N+1, 0);
        excess[s] = INF, excess[t] = -INF;
        globalRelabel(s,t);
        for (auto &e : adj[s]) push(s, e);
        for (; highest >= 0; --highest) {
            while (!lst[highest].empty()) {
                int v = lst[highest].back();
                lst[highest].pop_back();
                discharge(v);
                if (work > 4 * N) globalRelabel(s,t);
            }
        }
        return excess[t] + INF;
    }
};
HLPP<int> hlpp;
```

(10) kruskal(nlogn)

```
///HDU 1683
#include <bits/stdc++.h>
```

```
using namespace std ;
int n ,m;
const int MAXN(1000005) ;
int fa[MAXN] ;

class Node
{
public:
	int from,to,val;
	Node(int _from,int _to,int _val):from(_from),to(_to),val(_val){}
	bool operator < (Node&b) {return val<b.val ;}
};

vector<Node> Edge ;

void Init(int n = ::n)
{
	Edge.clear() ;
	for(int i(0);i<=n;++i) fa[i] = i ;
}

int find(int x)
{
	int f(x) ;
	while(fa[f]!=f) {f = fa[f] ;}
	while(fa[x]!=x) {int t(x) ;x = fa[x] ;fa[t] = f;}
	return f ;
}

int join(int a,int b)
{
	int Fa(find(a)),Fb(find(b)) ;
	return fa[Fb] = Fa ;
}

int kruskal()
{
	sort(Edge.begin(),Edge.end()) ;
	int cnt(0),val(0) ;
	for(int i(0);cnt<n-1&&i<Edge.size();++i) {
		if(find(Edge[i].from)!=find(Edge[i].to)) {
			val += Edge[i].val ;
			join(Edge[i].from,Edge[i].to) ;
			++cnt ;
```

```
            }
        }
        return cnt == n-1?val:-1 ;
}

int main()
{
        while(~scanf("%d%d",&m,&n))
        {
                if(m == 0) break ;
                Init() ;
                for(int i(0);i<m;++i) {
                        int from,to,val;
                        scanf("%d%d%d",&from,&to,&val) ;
                        Edge.push_back(Node(from,to,val)) ;
                }
                int ans(kruskal()) ;
                if(ans == -1) {
                        puts("?") ;
                } else {
                        printf("%d\n",ans);
                }
        }
        return 0;
}
```

(11) LCA 倍增法(nlogn)

```
/*(HDU2586)
        给出一颗树,问你任意两点的路径上的所有权值的和.
*/
#include <bits/stdc++.h>
const int MAXN (100005) ;
const int MAXM (100005) ;
using namespace std ;
struct Edge{int to,Next,val;};
Edge edge[MAXM<<1] ;
int head[MAXN+5],tot ;
int fa[MAXN][30],h,deep[MAXN] ;
int64_t disSum[MAXN] ;//从根到 i 点的权值和

inline void add_edge(int u,int v,int val = 1)
{
        edge[tot].to = v ;
        edge[tot].Next = head[u] ;
```

```
    edge[tot].val = val ;
    head[u] = tot ++ ;
}

void dfs(int pos=1,int _fa=0)
{
    deep[pos] = deep[_fa] + 1 ;
    fa[pos][0] = _fa ;
    for(int i(1);i<=h;++i) {
        fa[pos][i] = fa[fa[pos][i-1]][i-1] ;
    }
    for(int i(head[pos]);~i;i = edge[i].Next) {
        int p(edge[i].to),val(edge[i].val) ;
        if(p == _fa) continue ;
        disSum[p] = disSum[pos] + val ;
        dfs(p,pos) ;
    }
}

inline void Init(int n)
{
    for(int i(0);i<=n;++i) {
        head[i] = -1 ;
        disSum[i] = 0 ;
    }
    memset(fa[0],0,sizeof(fa[0])) ;///!!!!!!!!!!!!这里一定别忘记写啊!!!!
    deep[0] = tot = 0 ;
    h = (int)ceil(log(n)/log(2)) ;///!!!!!!!!!!!!这里一定别忘记写啊!!!!
}

inline int lca(int x,int y)
{
    if(deep[x]>deep[y]) swap(x,y) ;
    for(int i(h);i>=0;--i)
        if(deep[fa[y][i]]>=deep[x]) y = fa[y][i] ;
    if(x == y) return x ;
    for(int i(h);i>=0;--i)
        if(fa[x][i] != fa[y][i]) x = fa[x][i],y=fa[y][i] ;
    return fa[x][0] ;
}

int main()
{
    int T ;
```

```
    scanf("%d",&T) ;
    while(T--)
    {
        int n,m ;
        scanf("%d%d",&n,&m) ;
        Init(n+5) ;
        for(int i(1);i<n;++i) {
            int u,v,val ;
            scanf("%d%d%d",&u,&v,&val) ;
            add_edge(u,v,val) ;
            add_edge(v,u,val) ;
        }
        dfs() ;
        while(m--) {
            int u,v ;
            scanf("%d%d",&u,&v) ;
            printf("%lld\n",disSum[v]+disSum[u]-2*disSum[lca(u,v)]) ;
        }
    }
}
```

(12)LCA 可 O(1)换根[欧拉序+ST 表]O(nlogn+q)

```
#include <bits/stdc++.h>
using namespace std ;
const int MAXN = 500005 ;
vector<pair<int,int> > Next[MAXN] ;
int id[MAXN<<1],st[MAXN],pos[MAXN],deep[MAXN],tot ;

inline void addedge(int u,int v,int val = 0)
    {Next[u].push_back({v,val}) ;Next[v].push_back({u,val}) ;}

void dfs(int pos = 1,int _deep = 1,int _fa = -1)
{
    id[++tot] = pos ; st[pos] = tot ;
    deep[pos] = _deep ;
    int sz(Next[pos].size()) ;
    for(register int i(0);i<sz;++i) {
        pair<int,int>&e = Next[pos][i];
        if(e.first == _fa) continue ;
        dfs(e.first,_deep+1,pos) ;
        id[++tot] = pos ;
    }
}
```

```
template<typename T>
inline T deep_min(const initializer_list<T>& x)
{
	T ret = *x.begin() ;
	for(auto& it:x) if(deep[it]<deep[ret]) ret = it ;
	return ret ;
}

template<typename T>
inline T deep_max(const initializer_list<T>& x)
{
	T ret = *x.begin() ;
	for(auto& it:x) if(deep[it]>deep[ret]) ret = it ;
	return ret ;
}

class ST
{
public:
	int dpMin[15][MAXN<<1];
	int pw3[15],lg3[MAXN<<1];
	void Init(int* arr,int n)
	{///下标从 1 开始,传参数的时候 arr 需要注意一下!!!
		pw3[0] = 1 ;
		for(register int i(1);i<=n;++i) {
			lg3[i] = (int)(log(i)/log(3)) ;
			dpMin[0][i] = arr[i] ;
			if(i<=15) pw3[i] = pw3[i-1]*3 ;
		}
		for(register int i(1);i<=lg3[n];++i)
			for(register int j(1);j+pw3[i]-1<=n;++j)
				dpMin[i][j]                                                                    =
deep_min({dpMin[i-1][j],dpMin[i-1][j+(pw3[i-1])],dpMin[i-1][j+2*(pw3[i-1])]}) ;
	}
	int QMin(int l,int r)
	{
		int k(lg3[r-l+1]),d(r-2*pw3[k]+1) ;
		int ans(deep_min({dpMin[k][l],dpMin[k][r-pw3[k]+1]})) ;
		return d<l?ans:deep_min({ans,dpMin[k][d]});
	}
}St;

void InitLCA(int n,int root = 1)
```

```
    {tot = 0 ; dfs(root) ; St.Init(id,tot) ;}
void Init(int n) {for(int i(0);i<=n;++i) Next[i].clear() ;}

inline int LCA(int u,int v)
{///得到以初始为根的 LCA
    int a(min(st[u],st[v])),b(max(st[u],st[v])) ;
    return St.QMin(a,b) ;
}
inline int LCA(int u,int v,int root)
{///得到以参数 root 为根的 LCA
    return deep_max({LCA(u,v),LCA(u,root),LCA(v,root)}) ;
}

int main()
{
    int n,m,s,u,v   ;
    scanf("%d%d%d",&n,&m,&s) ;
    Init(n) ;
    for(int i(1);i<n;++i) {
        scanf("%d%d",&u,&v) ;
        addedge(u,v) ;
    }
    InitLCA(n,s) ;
    for(int i(1);i<=m;++i) {
        scanf("%d%d",&u,&v) ;
        printf("%d\n",LCA(u,v,s)) ;
    }
    return 0;
}
```

(13)spfa(klogn)

```
#include <bits/stdc++.h>
#define MAXN 100005
using namespace std ;
const int inf (0x3f3f3f3f) ;
int n ;
class Node
{
public :
    int to,val ;
    Node(int _to,int _val):to(_to),val(_val){}
    bool operator <(const Node &b) const    {return val>b.val;}
};
int dis[MAXN] ,isok[MAXN] ;
```

```
vector<Node> Next[MAXN] ;
void Init()
{
	for(int i(0);i<=n;++i) {
		Next[i].clear() ;
	}
}
int spfa(int from,int to)
{
	memset(dis,0x3f,sizeof(dis)) ;
	queue<int> que ;
	dis[from] = 0;
	que.push_front(from);
	while(que.size())
	{
		int x(que.front()) ;
		que.pop_front() ;
		for(int i(0);i<Next[x].size();++i) {
			int xx(Next[x][i].to),val(Next[x][i].val) ;
			if(dis[x] + val < dis[xx]) {
				dis[xx] =    dis[x] + val ;
				que.push_back(xx) ;
			}
		}
	}
	return dis[to] ;
}
int main()
{
	int T ;
	//scanf("%d",&T) ;
	while(1){
		int m;
		scanf("%d%d",&n,&m) ;
		if(n == 0 && m == 0) return 0;
		Init() ;
		for(int i(0);i<m;++i) {
			int from,to,val ;
			scanf("%d%d%d",&from,&to,&val) ;
			Next[from].push_back(Node(to,val)) ;
			Next[to].push_back(Node(from,val)) ;
		}
		int len(spfa(1,n));
		if(len == inf)
```

```
                puts("Bad luck!") ;
            else
                printf("%d\n",len) ;
        }
}
```

(14) spfa 最长路(klogn)

```
///HDU 6201
#include <bits/stdc++.h>
using namespace std ;
int n ,m;
const int MAXN(100005) ;

int pri[MAXN] ;

class Node
{
public :
    int to,val ;
    Node(int _to,int _val):to(_to),val(_val){}
};

vector<Node> Next[MAXN] ;

void Init(int n = ::n+1)
{
    for(int i(0);i<=n;++i) {
        Next[i].clear() ;
    }
}

int dis[MAXN] ;

int spfa(int from,int to)
{
    memset(dis,0x3f,sizeof(dis)) ;
    dis[from] = 0 ;
    queue<int> que ;
    que.push(from) ;
    while(que.size())
    {
        int x(que.front()) ;
        que.pop() ;
```

```
            for(int i(0);i<Next[x].size();++i) {
                int xx(Next[x][i].to),val(Next[x][i].val) ;
                if(dis[x] + val < dis[xx]) {
                    dis[xx] = dis[x] + val ;
                    que.push(xx) ;
                }
            }
        }
        return -dis[to] ;
}

int main()
{
        int T ,from,to,val;
        scanf("%d",&T);
        while(T-->0)
        {
            scanf("%d",&n);
            Init(n+1) ;
            for(int i (1) ; i<=n;++i) {
                scanf("%d",&pri[i]) ;
            }
            for(int i(0);i<n-1;++i) {///求最长路,存负边权
                scanf("%d%d%d",&from,&to,&val) ;
                Next[from].push_back(Node(to,-(pri[to]-pri[from]-val))) ;
                Next[to].push_back(Node(from,-(pri[from]-pri[to]-val))) ;
            }
            for(int i(1);i<=n;++i) {///建立源点汇点(虚点)
                Next[0].push_back(Node(i,0)) ;
                Next[i].push_back(Node(n+1,0)) ;
            }
            printf("%d\n",spfa(0,n+1)) ;
        }
}
```

(15)tarjan 缩点强连通 O(n)

```
/*
POJ1236
题意:n 个学校,学校可以发送物品到另一个学校(单向)求:
    1.最少需要几个学校初始有物品,才能使物品传递到所有学校.
    2.最少需要加几条边,才可以实现强连通分量.
思路:很明显的强连通分量,经过强连通分量缩点之后,第一问是所有入度为 0 的点(因为没有谁能传递给他),第二问是 max(出度为 0 的点数,入度为 0 的点数).
```

易错点:当强连通缩点后只有一个点时作为特殊情况,这个时候不需要加边,这个地方坑了好久~~~
*/
```
#include <set>
#include <stdio.h>
#include <bitset>
#include <vector>
const int MAXN (100005) ;
const int MAXM (100005) ;
using namespace std ;

struct Edge
{
	int to,Next ;
};
Edge edge[MAXM+5] ;
int head[MAXN+5],tot ;

inline void add_edge(int u,int v)
{
	edge[tot].to = v ;
	edge[tot].Next = head[u] ;
	head[u] = tot ++ ;
}

int DFN[MAXN+5],LOW[MAXN+5],cur/*时间戳*/,Cnt/*强连通分量个数*/ ;
int belong[MAXN+5];/*每个点属于哪个强连通分量*/
//DFN[i]: i 点的进入时间
//LOW[i]: 从 i 点出发,所能访问到的最早的进入时间.
int sta[MAXN+5],top ;
bitset<MAXN+5> isInStack ;

void tarjan(int pos)///dfs 找以 pos 为树根的所有强连通分量
{
	sta[top++] = pos ;
	isInStack[pos] = 1 ;
	DFN[pos] = LOW[pos] = ++ cur ;
	for(int i(head[pos]);~i;i=edge[i].Next) {
		int p(edge[i].to) ;
		if(!DFN[p]) {
			tarjan(p) ;
			LOW[pos] = min(LOW[pos],LOW[p]) ;
		} else if(isInStack[p]) {
			LOW[pos] = min(LOW[pos],DFN[p]) ;
```

```
            }
        }
        if(DFN[pos] == LOW[pos]) {///得到强连通分量
            ++Cnt ;
            int p ;
            //putchar('\t') ;
            do
            {
                p = sta[--top];     //退栈
                isInStack[p] = 0;     //标记不在栈中
                belong[p] = Cnt;     //出栈结点 p 属于 Cnt 标号的强连通分量
                // printf("%d,",p) ;
            }while(p != pos);    //直到将 v 从栈中退出
            //putchar('\n') ;
        }
}

void tarjanAll(int n)///找所有的强连通分量
{
    for(int i(1);i<=n;++i)
        if(!DFN[i])
            tarjan(i) ;
}

vector<int> Next[MAXN+5] ;///得到缩点以后的有向无环图

set<int> s ; //重建图之后存在的点集
int deg_in[MAXN] ;//重建图之后点的入度
int deg_out[MAXN] ;//重建图之后点的出度

inline void reBuild(int n)///缩点建图
{
    tarjanAll(n) ;
    s.clear() ;
    for(int i(0);i<=n+5;++i) deg_in[i]^=deg_in[i] ,deg_out[i]^=deg_out[i] ;
    for(int i(1);i<=n;++i) {
        s.insert(belong[i]) ;
        for(int j(head[i]);~j;j=edge[j].Next) {
            int p(edge[j].to) ;
            if(belong[p]!=belong[i]) {
                Next[belong[i]].push_back(belong[p]) ;
                s.insert(belong[p]) ;
                ++ deg_in[belong[p]] ;
                ++ deg_out[belong[i]] ;
```

```
                }
            }
        }
}

inline void Init(int n)///初始化
{
        isInStack.reset() ;
        Cnt = cur = top = tot =    0 ;
        for(int i(0);i<=n;++i) {
                Next[i].clear() ;
                head[i] = -1 ;
                belong[i] = -1;
                DFN[i] = LOW[i] = 0 ;
        }
}

void print(int n)//DEBUG 用的,输出构建后的新图
{
        for(int i(1);i<=n;++i) {
                printf("son[%d]:\n\t",i) ;
                for(int j(0);j<Next[i].size();++j) {
                        printf("%d",Next[i][j]) ;
                        if(j != Next[i].size()-1) {
                                printf(" , ") ;
                        }
                }
                putchar('\n') ;
        }
}

int main()
{
        int n, m ;
        while(~scanf("%d",&n)) {
                Init(n+5) ;
                for(int i(1);i<=n;++i) {
                        while(scanf("%d",&m)) {
                                if(m == 0) break ;
                                add_edge(i,m) ;
                        }
                }
                reBuild(n) ;
                if(Cnt == 1) {
```

```
            puts("1\n0") ;
            continue ;
        }
        int _in0(0),_out0(0) ;
        for(set<int>::iterator it = s.begin();it!=s.end();++it) {
            if(!deg_in[*it]) ++ _in0 ;
            if(!deg_out[*it]) ++ _out0 ;
        }
        printf("%d\n%d\n",_in0,max(_in0,_out0)) ;
    }
}
```

(16)tarjan 缩点双连通 O(n)

```
/*
hdu4612
题意:
给定一个联通图，问加入一条边后，最少还余下多少个割边
分析:先求强连通分量个数 num,然后缩点形成一棵树,再求树的直径 cnt,答案就是 num-1-cnt;
*/
#pragma comment(linker, "/STACK:10240000000000,10240000000000")
#include <bits/stdc++.h>
#define O2 __attribute__((optimize("O2")))
const int MAXN (200005) ;
const int MAXM (1000005) ;
const int inf (0x3f3f3f3f) ;
using namespace std ;
struct Edge
{
    int to,Next ;
};
Edge edge[MAXM<<1+5] ;
int head[MAXN],tot ;

inline void add_edge(int u,int v)
{
    edge[tot].to = v ;
    edge[tot].Next = head[u] ;
    head[u] = tot ++ ;
}

int DFN[MAXN+5],LOW[MAXN+5],cur/*时间戳*/,Cnt/*强连通分量个数*/ ;
int belong[MAXN+5];/*每个点属于哪个强连通分量*/
//DFN[i]: i 点的进入时间
//LOW[i]: 从 i 点出发,所能访问到的最早的进入时间.
```

```
int sta[MAXN+5],top ;
bitset<MAXN+5> isInStack ;
O2 void tarjan(int pos,int last=-1)///dfs 找以 pos 为树根的所有双连通分量
{
	sta[top++] = pos ;
	isInStack[pos] = 1 ;
	DFN[pos] = LOW[pos] = ++ cur ;
	for(int i(head[pos]);~i;i=edge[i].Next) {
		if(i == (last^1)) continue ;
		int p(edge[i].to) ;
		if(!DFN[p]) {
			tarjan(p,i) ;
			LOW[pos] = min(LOW[pos],LOW[p]) ;
		} else if(isInStack[p]) {
			LOW[pos] = min(LOW[pos],DFN[p]) ;
		}
	}
	if(DFN[pos] == LOW[pos]) {///得到双连通分量
		++Cnt ;
		int p ;
		//putchar('\t') ;
		do
		{
			p = sta[--top];	//退栈
			isInStack[p] = 0;	//标记不在栈中
			belong[p] = Cnt;	//出栈结点 p 属于 Cnt 标号的强连通分量
			// printf("%d,",p) ;
		}while(p != pos);	//直到将 v 从栈中退出
		//putchar('\n') ;
	}
}

inline void tarjanAll(int n)///找所有的双连通分量
{
	for(int i(1);i<=n;++i)
		if(!DFN[i])
			tarjan(i) ;
}

Edge edge2[MAXM<<1+5] ;
int head2[MAXN],tot2 ;

inline void add_edge2(int u,int v)
{
```

```
    edge2[tot2].to = v ;
    edge2[tot2].Next = head2[u] ;
    head2[u] = tot2 ++ ;
}


O2 inline void reBuild(int n)///缩点建树
{
    tarjanAll(n) ;
    tot2 = 0 ;
    for(int i(0);i<=n;++i) head2[i] = -1 ;
    for(int i(1);i<=n;++i) {
        for(int j(head[i]);~j;j=edge[j].Next) {
            int p(edge[j].to) ;
            //printf("%d -> %d\n",i,p) ;
            if(belong[p]!=belong[i]) {
                add_edge2(belong[i],belong[p]) ;
            }
        }
    }
}

inline void Init(int n)///初始化
{
    isInStack.reset() ;
    Cnt = cur = top = tot =    0 ;
    for(int i(0);i<=n;++i) {
        head[i] = -1 ;
        belong[i] = -1;
        DFN[i] = LOW[i] = 0 ;
    }
}

void print(int n)//DEBUG 用的,输出构建后的新图
{
    for(int i(1);i<=n;++i) {
        printf("Son[%d]:\n\t",i) ;
        for(int j(head2[i]);~j;j=edge2[j].Next) {
            printf("%d , ",edge2[j].to) ;
        }
        putchar('\n') ;
    }
}
```

```
int dis[MAXN] ;

int dfs(int pos=1,int _fa=-1)
{
	dis[pos] = -inf ;
	int ans_p(-1),isok(0);
	for(int i(head2[pos]);~i;i=edge2[i].Next) {
		int p(edge2[i].to),val(1) ;
		if(p == _fa) continue ;
		isok = 1 ;
		int u(dfs(p,pos)) ;
		if(dis[p]+val>=dis[pos]) {
			dis[pos] = dis[p] + val ;
			ans_p = u ;
		}
	}
	if(isok) {return    ans_p ;}
	else {
		dis[pos] = 0 ;
		return pos ;
	}
}

O2 int main()
{
	int n, m ;
	while(~scanf("%d%d",&n,&m)) {
		if(n == 0||m == 0) break ;
		Init(n+5) ;
		while(m--) {
			int u,v ;
			scanf("%d%d",&u,&v) ;
			add_edge(u,v) ;
			add_edge(v,u) ;
		}
		reBuild(n) ;//缩点建树
		int u(dfs()) ;//两遍 dfs 找出树的直径.
		int v(dfs(u)) ;
		printf("%d\n",Cnt-1-dis[u]) ;
	}
	return 0 ;
}
```

(17) topSort[链式前向星]O(v+e)

```
#include <bits/stdc++.h>
using namespace std ;
int n ,m;
const int MAXN(1000005) ;
int deg_in[MAXN]   ;

struct Edge {int to,val,Next ;};
Edge edge[MAXN<<2] ;
int head[MAXN],tot ;
vector<int> topQue ;

void Init(int n)
{
    tot = 0 ;
    for(int i(0);i<=n;++i) {
        head[i] = -1 ;
        deg_in[i] = 0 ;
    }
}

void add_edge(int from,int to,int val = 0)
{
    edge[tot].to = to;edge[tot].val =val ;
    edge[tot].Next = head[from];
    head[from] = tot ++;
}

bool topSort(int n)///返回图中是否有环
{
    topQue.clear() ;
    queue<int> que ;
    for(int i(1);i<=n;++i) {
        if(deg_in[i] == 0) que.push(i) ;
    }
    if(que.empty()) return false ;
    while(que.size())
    {
        int p(que.front()) ;que.pop() ;
        topQue.push_back(p) ;
        for(int i(head[p]);~i;i = edge[i].Next) {
            int pp(edge[i].to) ;
            if(--deg_in[pp] == 0) que.push(pp) ;
```

```
            }
        }
        return topQue.size() == n ;
}

int main()
{
        int T ,from,to;
        scanf("%d",&T);
        while(T-->0)
        {
                scanf("%d%d",&n,&m);
                Init(n+1) ;
                for(int i(1);i<=m;++i) {
                        scanf("%d%d",&from,&to) ;
                        ++deg_in[to] ;
                        add_edge(from,to) ;
                }
                topSort(n);
                for(int i(0);i<topQue.size();++i) {
                        printf("%d%c",topQue[i],i == topQue.size()-1?'\n':' ') ;
                }
        }
}
```

(18)二分图相关

定义：给定一个二分图 G，在 G 的一个子图 M 中，M 的边集{E}中的任意两条边都不依附于同一个顶点，则称 M 是一个匹配。
匹配点：匹配边上的两点

极大匹配(Maximal Matching)：是指在当前已完成的匹配下,无法再通过增加未完成匹配的边的方式来增加匹配的边数。

最大匹配(maximum matching)：是所有极大匹配当中边数最大的一个匹配,设为 M。选择这样的边数最大的子集称为图的最大匹配问题。

完美匹配（完备匹配）：一个图中所有的顶点都是匹配点的匹配，即 2|M| = |V|。完美匹配一定是最大匹配，但并非每个图都存在完美匹配。

最优匹配：最优匹配又称为带权最大匹配，是指在带有权值边的二分图中，求一个匹配使得匹配边上的权值和最大。一般 X 和 Y 集合顶点个数相同，最优匹配也是一个完备匹配，即每个顶点都被匹配。如果个数不相等，可以通过补点加 0 边实现转化。一般使用 KM 算法解决该问题。（KM（Kuhn and Munkres）算法，是对匈牙利算法的一种贪心扩展。）

最小覆盖
二分图的最小覆盖分为最小顶点覆盖和最小路径覆盖：

①最小顶点覆盖是指最少的顶点数使得二分图 G 中的每条边都至少与其中一个点相关联
注：二分图的最小顶点覆盖数=二分图的最大匹配数

②最小路径覆盖也称为最小边覆盖，是指用尽量少的不相交简单路径覆盖二分图中的所有顶点。
注：二分图的最小路径覆盖数=|V|-二分图的最大匹配数

最大独立集
最大独立集是指寻找一个点集，使得其中任意两点在图中无对应边。对于一般图来说，最大独立集是一个 NP 完全问题，对于二分图来说最大独立集=|V|-二分图的最大匹配数。最大独立集 S 与 最小覆盖集 T 互补

如果在图 G 左右两边加上源汇点后，图 G 等价于一个网络流，二分图最大匹配问题可以转为最大流的问题。解决此问题的匈牙利算法的本质就是寻找最大流的增广路径。

在二分图最大匹配中，每个点（不管是 X 方点还是 Y 方点）最多只能和一条匹配边相关联，然而，我们经常遇到这种问题，即二分图匹配中一个点可以和多条匹配边相关联，但有上限，或者说，Li 表示点 i 最多可以和多少条匹配边相关联。

二分图多重匹配分为二分图多重最大匹配与二分图多重最优匹配两种，分别可以用最大流与最大费用最大流解决。

（1）二分图多重最大匹配：
在原图上建立源点 S 和汇点 T，S 向每个 X 方点连一条容量为该 X 方点 L 值的边，每个 Y 方点向 T 连一条容量为该 Y 方点 L 值的边，原来二分图中各边在新的网络中仍存在，容量为 1（若该边可以使用多次则容量大于 1），求该网络的最大流，就是该二分图多重最大匹配的值。

（2）二分图多重最优匹配：
在原图上建立源点 S 和汇点 T，S 向每个 X 方点连一条容量为该 X 方点 L 值、费用为 0 的边，每个 Y 方点向 T 连一条容量为该 Y 方点 L 值、费用为 0 的边，原来二分图中各边在新的网络中仍存在，容量为 1（若该边可以使用多次则容量大于 1），费用为该边的权值。求该网络的最大费用最大流，就是该二分图多重最优匹配的值。

(19)二分图最大独立集(最大团)问题

二分图的最大独立集定义: 满足任意两个点都没有边相连的点集
二分图的最大团定义: 满足任意两个点都有边相连的点集

二分图的最大独立集合 = 该图的补图的最大团
二分图的最大团问题可以直接转换为最大独立及问题

所以我们可以不拆点,直接分成两堆建立二分图,然后跑最大流
n - maxflow 就是答案.

输出路径的时候,跑完最大流的残余网络中大左边 deep[i] = 0 的点一定是最大流中的关键点,是不可以被替代的,

所以左边 deep[i] = 0 的话,与它相连的右边的点一定只有一个,并且它的 deep[j]一定也是 0.所以这两个点只能拿一个

所以输出最大独立集的时候,只需要按照如下规则判断就可以了:

```
for(int i(1);i<=n;++i) {
        if(isLeft[i]&&dinic.deep[i]) {
            v.push_back(a[i]) ;
        } else if(!isLeft[i]&&dinic.deep[i] == 0){
            v.push_back(a[i]) ;
        }
    }
```

(20)网络流有关问题

(1)DAG [有向无环图] 的最小不相交路径覆盖

①最小路径覆盖的定义: 在一个有向图中，找出最少的路径，使得这些路径经过了所有的点。

②二分图定理：最小路径覆盖数=顶点数-最大匹配

简单证明：一开始每个点都独立的为一条路径，总共有 n 条不相交路径。我们每次在二分图里加一条边就相当于把两条路径合成了一条路径，因为路径之间不能有公共点，所以加的边之间也不能有公共点，这就是匹配的定义。所以有：最小路径覆盖数=顶点数-最大匹配。

有这个定理就简单了，求一遍最大匹配就好了。

把原图的每个点 V 拆成 Vx 和 Vy 两个点(Vx 和 Vy 不相连)，如果有一条有向边 A->B，那么就加边 Ax->By。这样就得到了一个二分图。那么最小路径覆盖=原图的结点数-新图的最大匹配数(即:n-maxflow)。最后的方案可以利用残量网络用并查集维护。即从 1 到 n 枚举，从每个点向外扫一圈，如果有流从这条边经过，并流向 y+n，则合并 x 与 y。然后 n^2 输出方案即可。

(2)无向图的最大团 == 该无向图补图的最大独立集

(3)最大独立集 == 点的总数 - 最小点覆盖

(4)最小点覆盖 == 最大匹配

(21) 原始对偶费用流

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

class PrimalDual
{
public:
    const static int V=50005, E=500005,inf=0x3F3F3F3F;
    int N ;
    void clear(int n) {
        tot = 0 ; N = n+1 ; fill_n(head,N+1,-1) ;
    }
    void addedge(int u, int v,int flow=1,int cost=0) {
        edge[tot] = Edge(v, +cost, flow, head[u]); head[u] = tot ++ ;
```

```
        edge[tot] = Edge(u, -cost, 0, head[v]); head[v] = tot ++ ;
    }
    void MCMF(int s,int t,int &maxFlow,int &minCost) {
        S = s ;T = t ; flow = buffcost = cost = 0;
        int buff(0) ;
        while(modlabel())
            do {fill_n(v,N+5,0) ;flow += buff ;} while((buff = aug(S, inf)));
        maxFlow = flow;minCost = cost ;
    }
private:
    struct Edge {
        int t,c,u,Next ;
        Edge() {}
        Edge(int T, int C, int U, int N): t(T), c(C), u(U), Next(N) {}
    } edge[E<<1];
    int head[V],tot ;
    int S, T,buffcost, cost,flow;
    bool v[V];
    int aug(int no, int m) {
        if (no == T) return cost += buffcost*m,m;
        v[no] = true;
        int l(m);
        for (int i(head[no]); ~i; i = edge[i].Next)
            if (edge[i].u && !edge[i].c && !v[edge[i].t]) {
                int d = aug(edge[i].t, l < edge[i].u ? l : edge[i].u);
                edge[i].u -= d; edge[i^1].u += d; l -= d;
                if (!l) return m;
            }
        return m - l;
    }
    bool modlabel() {
        static int d[V]; memset(d, 0x3F, sizeof(int)*N+5); d[T] = 0;
        static deque<int> Q; Q.push_back(T);
        while(Q.size()) {
            int dt, no = Q.front(); Q.pop_front();
            for(int i(head[no]);~i; i = edge[i].Next)
                if(edge[i^1].u && (dt = d[no] - edge[i].c) < d[edge[i].t])
                    (d[edge[i].t] = dt) <= d[Q.size() ? Q.front() : 0]
                        ? Q.push_front(edge[i].t) : Q.push_back(edge[i].t);
        }
        for(int i(0); i < N; ++i)
            for(int j(head[i]); ~j; j = edge[j].Next)
                edge[j].c += d[edge[j].t] - d[i];
        buffcost += d[S];
```

```
            return d[S] < inf;
        }
}pd ;
int main()
{
        int n,m,s,t ;
        while(~scanf("%d%d%d%d",&n,&m,&s,&t))
        {
                pd.clear(n) ;
                while(m-->0)
                {
                        int u,v,f,c ;
                        scanf("%d%d%d%d",&u,&v,&f,&c) ;
                        pd.addedge(u,v,f,c) ;
                }
                int f,c ;
                pd.MCMF(s,t,f,c) ;
                printf("%d %d\n",f,c) ;
        }
        return 0;
}
```

(22) 原始对偶邻接表

```
#include <bits/stdc++.h>
using namespace std;
class PrimalDual
{
public:
        const static int V=500005,inf=0x3F3F3F3F;
        int N ;
        void clear(int n) {
                N = n+1 ; for(int i(0);i<=N;++i) edge[i].clear();
        }
        void addedge(int u, int v,int flow=1,int cost=0) {
                edge[u].push_back({v,cost,flow,edge[v].size()}) ;
                edge[v].push_back({u,-cost,0,edge[u].size()-1}) ;
        }
        void MCMF(int s,int t,int &maxFlow,int &minCost) {
                S = s,T = t ;
                flow = buffcost = cost = 0;
                int buff(0) ;
                while(modlabel())
                        do {fill_n(v,N+5,0) ;flow += buff ;}while((buff = aug(S, inf)));
```

```
            maxFlow = flow;minCost = cost ;
        }
    private:
        struct Edge
        {
            int t, c, u,rev ;
            Edge() {}
            Edge(int T, int C, int U, int R): t(T), c(C), u(U), rev(R) {}
        };
        vector<Edge> edge[V];
        int S, T,buffcost, cost,flow;
        bool v[V];
        int aug(int no, int m) {
            if (no == T) return cost += buffcost*m,m;
            v[no] = true;
            int l = m,sz = edge[no].size();
            for (int i(0);i<sz;++i) {
                Edge&e = edge[no][i] ;
                if (e.u && !e.c && !v[e.t]) {
                    int d = aug(e.t, l < e.u ? l : e.u);
                    e.u -= d; edge[e.t][e.rev].u += d; l -= d;
                    if (!l) return m;
                }
            }
            return m - l;
        }
        bool modlabel() {
            static int d[V]; memset(d, 0x3F, sizeof(int)*N+5); d[T] = 0;
            static deque<int> Q; Q.push_back(T);
            while(Q.size()) {
                int dt, no = Q.front(),sz(edge[no].size()); Q.pop_front();
                for(int i(0);i<sz;++i) {
                    Edge &e = edge[no][i],&re = edge[e.t][e.rev] ;
                    if(re.u && (dt = d[no] - e.c) < d[e.t])
                        (d[e.t] = dt) <= d[Q.size() ? Q.front() : 0]
                            ? Q.push_front(e.t) : Q.push_back(e.t);
                }
            }
            for(int i(0); i < N; ++i) {
                int sz (edge[i].size()) ;
                for(int j(0); j < sz ; ++j) {
                    Edge& e = edge[i][j] ;
                    e.c += d[e.t] - d[i];
                }
```

```
            }
            buffcost += d[S];
            return d[S] < inf;
        }
}pd;
int main()
{
        int n,m,s,t ;
        while(~scanf("%d%d%d%d",&n,&m,&s,&t))
        {
                pd.clear(n) ;
                while(m-->0)
                {
                        int u,v,f,c ;
                        scanf("%d%d%d%d",&u,&v,&f,&c) ;
                        pd.addedge(u,v,f,c) ;
                }
                int f,c ;
                pd.MCMF(s,t,f,c) ;
                printf("%d %d\n",f,c) ;
        }
        return 0;
}
```

(23)原始对偶小数费用流

```
/**用来跑费用为小数的费用流**/
#include <bits/stdc++.h>
#define double float ///这里也可以改成 int64_t
#define O2 __attribute__((optimize("O2")))
const double eps (1e-8);
using namespace std;
class PrimalDual
{
public:
        const static int V=1005, E=50005,inf=0x3F3F3F3F;
        int N ;
        inline void clear(int n) {
                tot = 0 ; N = n+1 ; fill_n(head,N+1,-1) ;
        }
        inline void addedge(int u, int v,int flow,double cost) {
                edge[tot] = Edge(v, +cost, flow, head[u]); head[u] = tot ++ ;
                edge[tot] = Edge(u, -cost, 0, head[v]); head[v] = tot ++ ;
        }
```

```
    O2 void MCMF(int s,int t,int &maxFlow,double &minCost) {
        S = s ;T = t ;flow = buffcost = cost = 0 ;
        int buff(0) ;
        while(modlable())
            do {v.reset() ;flow += buff ;}while((buff = aug(S, inf)));
        maxFlow = flow; minCost = cost ;
    }
private:
    struct Edge
    {
        int t, u,Next;
        double c ;
        Edge() {}
        Edge(int T, double C, int U, int N): t(T), c(C), u(U), Next(N) {}
    } edge[E<<1];
    int head[V],tot ;
    int S, T,flow;
    double buffcost, cost ;
    bitset<V> v ;
    O2 int aug(int no, int m) {
        if (no == T) return cost += buffcost*m,m;
        v[no] = true;
        int l = m;
        for (int i(head[no]); ~i; i = edge[i].Next)
            if (edge[i].u && edge[i].c<eps && !v[edge[i].t]) {
                int d = aug(edge[i].t, l < edge[i].u ? l : edge[i].u);
                edge[i].u -= d, edge[i^1].u += d, l -= d;
                if (!l) return m;
            }
        return m - l;
    }
    O2 bool modlable(){
        static double d[V];
        for(int i(0);i<=N;++i) d[i] = inf ;
        d[T] = 0;
        static deque<int> Q; Q.push_back(T);
        while(Q.size()) {
            int no = Q.front(); Q.pop_front();
            double dt ;
            for(int i(head[no]);~i; i = edge[i].Next)
                if(edge[i^1].u && (dt = d[no] - edge[i].c) < d[edge[i].t])
                    (d[edge[i].t] = dt) <= d[Q.size() ? Q.front() : 0]
                        ? Q.push_front(edge[i].t) : Q.push_back(edge[i].t);
        }
```

```
            for(int i(0); i < N; ++i)
                for(int j(head[i]); ~j; j = edge[j].Next)
                    edge[j].c += d[edge[j].t] - d[i];
            buffcost += d[S];
            return d[S] < inf;
        }
}pd ;
O2 int main()
{
        int T ;
        scanf("%d",&T) ;
        while(T--)
        {
            int n,m,s,t ;
            scanf("%d%d",&n,&m) ;
            pd.clear(n+5) ;
            s = 0,t = n+1;
            for(int i(1);i<=n;++i)
            {
                int u,v ;
                scanf("%d%d",&u,&v) ;
                if(u > v) pd.addedge(s,i,u-v,0);
                if(v > u) pd.addedge(i,t,v-u,0);
            }
            for(int i(1);i<=m;++i) {
                int u,v,cnt;
                double cost ;
                scanf("%d%d%d%f",&u,&v,&cnt,&cost) ;
                pd.addedge(u,v,cnt-1,-log2(1-cost)) ;
                pd.addedge(u,v,1,0) ;
            }
            int f; double c ;
            pd.MCMF(s,t,f,c) ;
            printf("%.2f\n",1-pow(2,-c)) ;
        }
        return 0;
}
```

(24)原始对偶小数费用流临接表(内存充足的时候用)

```
/**用来跑费用为小数的费用流**/
///#include <bits/stdc++.h>
#include <bits/stdc++.h>
#define double float///这里也可以改成 int64_t!
```

```
const double eps (1e-8);
using namespace std;
class PrimalDual
{
public:
    const static int V=5005,inf=0x3F3F3F3F;
    int N ;
    void clear(int n) {
        N = n+1 ; for(int i(0);i<=N;++i) {edge[i].clear() ;};
    }
    void addedge(int u, int v,int flow=1,double cost=0) {
        edge[u].push_back(Edge(v,cost,flow,edge[v].size())) ;
        edge[v].push_back(Edge(u,-cost,0,edge[u].size()-1)) ;
    }
    void MCMF(int s,int t,int &maxFlow,double &minCost) {
        S = s ;T = t ; flow = buffcost = cost = 0 ;
        int buff(0) ;
        while(modlable())
            do {v.reset() ;flow += buff ;}while((buff = aug(S, inf)));
        maxFlow = flow; minCost = cost ;
    }
    struct Edge
    {
        int t, u , rev ; double c ;
        Edge() {}
        Edge(int T, double C, int U, int R): t(T), c(C), u(U), rev(R) {}
    };
    vector<Edge> edge[V];
private:
    int S, T,flow;
    double buffcost, cost ;
    bitset<V> v ;
    int aug(int no, int m) {
        if (no == T) return cost += buffcost*m,m;
        v[no] = true;
        int l(m),sz(edge[no].size());
        for (int i(0);i<sz;++i) {
            Edge&e = edge[no][i] ;
            if (e.u && e.c<eps && !v[e.t]) {
                int d = aug(e.t, l < e.u ? l : e.u);
                e.u -= d; edge[e.t][e.rev].u += d; l -= d;
                if (!l) return m;
            }
        }
```

```
            return m - l;
        }
        bool modlable(){
            static double d[V]; for(int i(0);i<=N;++i) d[i] = inf ;d[T] = 0;
            static deque<int> Q; while(Q.size()) Q.pop_back() ; Q.push_back(T);
            while(Q.size()) {
                int no(Q.front()),sz(edge[no].size()); Q.pop_front();
                double dt ;
                for(int i(0);i<sz;++i) {
                    Edge &e(edge[no][i]),&re(edge[e.t][e.rev]);
                    if(re.u && (dt = d[no] - e.c) < d[e.t])
                        (d[e.t] = dt) <= d[Q.size() ? Q.front() : 0]
                            ? Q.push_front(e.t) : Q.push_back(e.t);
                }
            }
            for(int i(0); i < N; ++i) {
                int sz (edge[i].size()) ;
                for(int j(0);j<sz;++j) {
                    Edge& e(edge[i][j]);
                    e.c += d[e.t] - d[i];
                }
            }
            buffcost += d[S];
            return d[S] < inf;
        }
}pd ;
double arr[105][105] ;
int main()
{
        int n,buff ;
        scanf("%d",&n) ;
        int s(0),t(2*n+5) ;
        for(int i(1);i<=n;++i) {
            for(int j(1);j<=n;++j) {
                scanf("%d",&buff) ;
                arr[i][j] = -log(buff)/log(2) ;
            }
        }
        pd.clear(1003) ;
        for(int i(1);i<=n;++i) {
            for(int j(1);j<=n;++j) {
                pd.addedge(i,j+n,1,arr[i][j]) ;
            }
```

```
            pd.addedge(s,i,1,0) ;
            pd.addedge(i+n,t,1,0) ;
        }
        int maxflow;
        double mincost ;
        pd.MCMF(s,t,maxflow,mincost) ;
        for(int i(n+1);i<=n+n;++i) {
            for(int j(0);j<pd.edge[i].size();++j) {
                if(pd.edge[pd.edge[i][j].t][pd.edge[i][j].rev].u == 0) {
                    printf("%d%c",pd.edge[i][j].t,(i == n+n)?'\n':' ') ;
                }
            }
        }
        return 0;
}
```

(25)支配树 O(nlogn)

```
#include<bits/stdc++.h>
using namespace std ;
/**
支配树(灭绝树)的定义:
    对于一棵可以存在环的有向图给定一个起点 r 对于一个终点 x 如果 r 到达 x 的所有路径都会经过一个点 y 那么我们就称之为 y 支配 x.
另一种定义:
    在一张捕食图上(从捕食者向被捕食者有连边),若某生物的所有食物都灭绝了,则该生物灭绝.灭绝树便是此图的一种生成树,使得满足灭绝树上某点灭绝,该点子树内所有点都灭绝.
支配树的性质:
    支配树中的任意一个点都支配它的子树,自己可以支配自己.
例题:洛谷 P2597[https://www.luogu.org/problem/P2597]
题意:
    一个食物网有 N 个点,代表 N 种生物,如果生物 x 可以吃生物 y,那么从 y 向 x 连一个有向边.
我们定义一个生物在食物网中的"灾难值"为,如果它突然灭绝,那么会跟着一起灭绝的生物的种数.
求每个生物的灾难值.
思路:
    由于出度为 0 的(生产者)可以支配所有点,那么我们建立一个超级源点作为所有的生产者的父节点,
然后求出以超级源点为根的支配树,然后支配树每个节点的 Size-1 就是答案(同类不能吃同类).
*/
const int N = 2000005 ; ///点的个数
```

```
const int M = 4000005 ; ///边的个数
class Graph {
public:
	int head[N],to[M],Next[M],tot ;
	int add_edge(int u,int v) {
		to[tot] = v ; Next[tot] = head[u];
		return head[u] = tot ++ ;
	}
	void clear(int n) {
		tot = 0;
		memset(head,-1,sizeof(head[0])*(n+2));
	}
};
class Dominator_Tree
{///用之前别忘了 clear()清空一下.
private:
	Graph G,R,S ;
	int g[N],sdom[N],mn[N],idom[N],DFN[N],id[N],fa[N],tot,n;
	int Fa[N][30],h ;/**倍增求支配树 LCA 用到的,不求 LCA 可以不写.*/
	inline int find(int x) {
		if(g[x] == x) return x;
		int f(g[x]); g[x] = find(g[x]);
		if(DFN[sdom[mn[f]]] < DFN[sdom[mn[x]]])
			mn[x] = mn[f];
		return g[x];
	}
	void dfs(int u) {
		DFN[u] = ++tot ; id[tot] = u;
		for(register int i(G.head[u]),v;~i;i = G.Next[i])
			if(!DFN[v = G.to[i]])
				fa[v] = u , dfs(v);
	}
	void Tarjan() {
		for(register int k(tot); k; --k) {
			const int& u(id[k]) ;
			for(register int i(S.head[u]),v;~i;i = S.Next[i]) {
				find(v = S.to[i]);
				if(sdom[mn[v]] == u) idom[v] = u;
				else idom[v] = mn[v];
			}
			for(register int i(R.head[u]),v;~i;i = R.Next[i])
				if(DFN[v = R.to[i]]) {
					find(v);
					if(DFN[sdom[mn[v]]] < DFN[sdom[u]])
```

```
                    sdom[u] = sdom[mn[v]];
                }
            S.add_edge(sdom[u], u); g[u] = fa[u];
        }
        for(register int i(2);i <= tot; ++i) {
            const int& u(id[i]);
            if(idom[u] != sdom[u])
                idom[u] = idom[idom[u]];
            T.add_edge(idom[u], u);
        }
    }
public:
    Graph T ; ///build()之后的 T 就是仅有单向边的支配树.
    int Size[N] ;///Size[i]表示 i 能支配多少个点(支配树中 i 的儿子的个数).
    int deep[N] ;///deep[i]表示 i 能被多少个点支配(支配树的深度).
    void clear(int _n) {
        tot = 0 ; n = _n ;
        h = __lg(_n) + 1 ;
        G.clear(n) ; R.clear(n) ;
        S.clear(n) ; T.clear(n) ;
        memset(Fa[0],0,sizeof(Fa[0])) ;
        for(register int i(0);i <= _n+2; ++i) {
            DFN[i] = 0 ; sdom[i] = mn[i] = g[i] = i;
        }
    }
    inline void add_edge(int u,int v)///加有向边
        {G.add_edge(u,v); R.add_edge(v,u);}
    Graph& build(int pos = 1){///以 pos 为根建支配树
        deep[pos] = 1;/**这里默认根的深度为 1 了,也可以改成 0*/
        dfs(pos); Tarjan(); return T;
    }
    void count(int pos = 1,int _fa = 0) {///统计支配信息.
        Size[pos] = 1 ;
        Fa[pos][0] = _fa ;
        for(int i(1);i<=h;++i) {
            Fa[pos][i] = Fa[Fa[pos][i-1]][i-1] ;
        }
        for(register int i(T.head[pos]),to;~i;i = T.Next[i]) {
            deep[to = T.to[i]] = deep[pos] + 1;
            count(to,pos) ;
            Size[pos] += Size[to] ;
        }
    }
    inline int LCA(int x,int y)///支配树上求两个点的 LCA
```

```
    {
        if(deep[x] > deep[y]) swap(x,y) ;
        for(register int i(h);i>=0;--i)
            if(deep[Fa[y][i]] >= deep[x]) y = Fa[y][i] ;
        if(x == y) return x ;
        for(register int i(h);i>=0;--i)
            if(Fa[x][i] != Fa[y][i]) x = Fa[x][i] , y = Fa[y][i] ;
        return Fa[x][0] ;
    }
}tree;
int deg_in[N] ;

int main()
{
    int n,m,u,v ;
    scanf("%d",&n) ;
    memset(deg_in,0,sizeof(deg_in[0])*(n+2)) ;
    tree.clear(n+1) ;
    for(int i = 1;i <= n; ++i)
        while(~scanf("%d",&u)) {
            if(!u) break ;
            tree.add_edge(u,i) ;
            ++ deg_in[i] ;
        }
    int root = n + 1 ;
    for(int i(1);i<=n;++i)
        if(deg_in[i] == 0)
            tree.add_edge(root,i) ;
    tree.build(root) ; tree.count(root) ;///注意!!!count 这里一定是 root!!!
    for(int i = 1;i <= n; ++i)
        printf("%d\n",tree.Size[i] - 1) ;
    return 0;
}
```

(26)求树上一个点的 K 级祖先 O((n+q)logn)

```
#include <bits/stdc++.h>
const int MAXN (300005) ;
const int MAXM (600005) ;
using namespace std ;
struct Edge{int to,Next,val;};
Edge edge[MAXM<<1] ;
int head[MAXN+5],tot ;
int fa[MAXN][30],h,deep[MAXN] ;
```

```
int64_t disSum[MAXN] ;//从根到 i 点的权值和

inline void add_edge(int u,int v,int val = 1)
{
	edge[tot].to = v ;
	edge[tot].Next = head[u] ;
	edge[tot].val = val ;
	head[u] = tot ++ ;
}

void dfs(int pos=1,int _fa=0)
{
	deep[pos] = deep[_fa] + 1 ;
	fa[pos][0] = _fa ;
	for(int i(1);i<=h;++i) {
		fa[pos][i] = fa[fa[pos][i-1]][i-1] ;
	}
	for(int i(head[pos]);~i;i = edge[i].Next) {
		int p(edge[i].to),val(edge[i].val) ;
		if(p == _fa) continue ;
		disSum[p] = disSum[pos] + val ;
		dfs(p,pos) ;
	}
}

inline void Init(int n)
{
	memset(fa[0],0,sizeof(fa[0])) ;
	for(int i(0);i<=n;++i)
		head[i] = -1 ;
	deep[0] = tot = 0 ;
	h = (int)ceil(log(n)/log(2)) ;
}

inline int KthFather(int pos,int k)
{
	int cnt(0) ;
	while(k) {
		if(k & 1) {
			pos = fa[pos][cnt] ;
		}
		k >>= 1 ; ++ cnt ;
	}
	return pos ;
```

```
}
#define en *++now_w = '\n' ///换行
#define sp *++now_w = 0x20 ///空格
const int ch_top=4e7;
char ch[ch_top],*now_r=ch-1,*now_w=ch-1;
template<typename T>
inline void read(T& x)
{
	while(*++now_r<'0');
	x = (*now_r-'0');
	while(*++now_r>='0')x=x*10+*now_r-'0';
}
template<typename T>
inline void write(T x)
{
	static char st[20];static int top;
	while(st[++top]='0'+x%10,x/=10);
	while(*++now_w=st[top],--top);
}
int main()
{
	fread(ch,1,ch_top,stdin);
	int T,n,m,k,u,v ;
	read(n);
	{
		Init(n+5) ;
		for(int i(1);i<n;++i) {
			///scanf("%d%d",&u,&v) ;
			read(u) ; read(v) ;
			add_edge(u,v) ; add_edge(v,u) ;
		}
		dfs() ;
		///scanf("%d",&m) ;
		read(m) ;
		int lastans(0) ;
		while(m-->0) {
			///scanf("%d%d",&u,&k) ;
			read(u) ; read(k) ;
			u ^= lastans ; k ^= lastans ;
			///printf("%d\n",lastans = KthFather(u,k)) ;
			write(lastans = KthFather(u,k)) ; en ;
		}
	}
	fwrite(ch,1,now_w-ch,stdout);
```

}

11. 字符串

(1) 后缀数组

⑤ 求后缀数组[倍增法]O(nlogn)

```
class SA
{
static const int MAXN = 2000005 ;
static const int LOG_MAXN = log(MAXN)/log(3)+2 ;
public:
    int Sa[MAXN],Rank[MAXN] ;/*Sa 和 Rank 互为反函数*/
    /// Sa:排名为 i 的后缀的位置(第 i 个后缀的位置),Rank:第 i 个后缀的 Rank
    /*Sa 表示排名为 i 的是啥,Rank 表示第 i 个的排名是啥*/
    int tp[MAXN], tax[MAXN];
    /// tp:第二关键字的排名为 i 的后缀的位置, 还被用作 rank 的暂存
    /// tax:每个排名对应的后缀数量，用作桶
    int Height[MAXN],len[MAXN] ;/*Height:第 i 个后缀与第 i-1 个后缀的最长公共前缀长度,即:lcp(Sa[i],Sa[i-1])[排名相邻的两个后缀的最长公共前缀的长度]*/
    int64_t kind[MAXN] ;///排名为[1,i]内的本质不同的子串的数量.
    ///len[i]: 排名为 i 的后缀的长度 len[i] = N+1-Sa[i]
    ///用 Suffix[i]表示从 i 开始的后缀(也叫 i 号后缀,从 i 开始的后缀).
    ///lcp(x,y):第 i 个后缀和第 j 个的后缀(也就是 Suffix(Sa[i])和 Suffix(Sa[j]))的最长公共前缀的长度.
    int dp[LOG_MAXN][MAXN],pw3[LOG_MAXN],mm[MAXN],N; ///ST 表维护任意区间的 height 的最小值,方便求 LCP
    /**
    性质:
        1. Suffix[j] 和 Suffix[k] 的最长公共前缀为 min{height[Rank[j]+1],height[Rank[j]+2]...height[Rank[k]]}.
        2. 对任意 i<=k<=j,有 LCP(i,j) = min(LCP(i,k),LCP(k,j))
        3. LCP(i,j)=min(i<k<=j)(LCP(k-1,k))
        第 2 个性质是显然的,它的意义在于可以用来证明第 3 个性质.
        第 3 个性质的意义在于提供了一个将 LCP 问题转换为 RMQ 问题的方法：
        令 height[i]＝LCP(i-1,i),即 height[i]代表第 i 小的后缀与第 i-1 小的后缀的 LCP,则求 LCP(i,j)就等于求 height[i+1]~height[j]之间的 RMQ,
        套用 RMQ 算法就可以了,复杂度是预处理 O(nlogn),查询 O(1).
    */
    /*
    字符串          height[i]   Sa[i](索引)   Rank[i]
    123232323          0            1           1
    23                 0            8           2
```

```
2323            2           6           3
232323          4           4           4
23232323        6           2           5
3               0           9           6
323             1           7           7
32323           3           5           8
3232323         5           3           9
*/
void rSort (int n,int m) {
        fill_n(tax,m+1,0) ;
        for (int i(1); i <= n; ++i) ++tax[Rank[i]];
        for (int i(1); i <= m; ++i) tax[i] += tax[i - 1];
        for (int i(n); i ; --i) Sa[tax[Rank[tp[i]]]--] = tp[i];
    } ;

inline void insert(int* str,int n,int m=128)//n 是字符串长度,m 是字符集大小
{/*注意!使用后缀数组时,字符串要从 0 开始输入!!!!*/
    N = n ; --str ;
    for (int i(1); i <= n; ++i) Rank[i] = str[i] , tp[i] = i;
    rSort(n,m) ;
    for (int k(1),p(0);p < n;m = p,k <<= 1) {
        p = 0;
        for (int i(n-k+1);i <= n;++i) tp[++p] = i;
        for (int i(1);i <= n; ++i)
            if (Sa[i] > k) tp[++p] = Sa[i] - k;
        rSort(n,m) ;
        swap(Rank, tp);
        Rank[Sa[1]] = p = 1;
        for (int i(2);i <= n;++i)
            Rank[Sa[i]] = (tp[Sa[i]]==tp[Sa[i-1]]&&tp[Sa[i]+k]==tp[Sa[i-1]+k])
                        ? p : ++p;
    }
    pw3[0] = 1 ;
    for(int i(1),j(0),k(0); i <= n; ++i) {///线性求出 Height 数组
        if(k) --k;
        j = Sa[Rank[i] - 1];mm[i] = (int)(log(i)/log(3)) ;
        if(i<LOG_MAXN) pw3[i] = pw3[i-1]*3 ;
        while(str[i + k] == str[j + k]) ++k;
        Height[Rank[i]] = k; ///Height 下标从 2 开始,要想从 1 开始,改成 Rank[i-1]
        dp[0][Rank[i]] = k;///预处理 ST 表
        len[i] = N+1-Sa[i] ;///得到排名为 i 的后缀的长度.
    }
    for(int i(1);i<=mm[n];++i) ///预处理 ST 表
        for(int j(1);j+mm[i]-1<=n;++j)
```

```
                dp[i][j] = min({dp[i-1][j],dp[i-1][j+pw3[i-1]],dp[i-1][j+2*pw3[i-1]]}) ;
    }

    int LCP(int i,int j,bool isByRank = false)
    {///任意两个后缀(i,j)的最长公共前缀长度
        if(isByRank) i = Sa[i],j = Sa[j] ;
        if(i==j) return len[Rank[i]];
        i = Rank[i];j = Rank[j] ;
        if(i>j) swap(i,j) ;++i;
        int k(mm[j-i+1]),d(j-2*pw3[k]+1),ans(min(dp[k][i],dp[k][j-pw3[k]+1]));
        return d<i?ans:min(ans,dp[k][d]);
    }

    int64_t getKind()
    {
        ///预处理出不同的子串的个数以及其前缀和
        /*对于每一次新加进来第 K 个后缀,将产生 len[k]个新的前缀。
        但是其中有 height[k]个是和前面的字符串的前缀是相同的.
        所以第 K 个后缀将"贡献"出  len[k] - height[k]个不同的子串.*/
        kind[0] = 0 ;
        for(int i(1);i <= N;++i)    kind [i] = kind[i-1] + len[i] - Height[i];
        return kind[N];
    }
}sa;

//注意,多字符串输入别忘了转成数字数组.两个字符串用一个|分开就行了.
```

② 出现大于等于 K 次的最长子串长度[Height 数组分组+二分]O(nlogn)

```
/**
    后缀数组求重复次数>=k 次的字串的最长长度.
    首先答案肯定是单调的,所以我们可以考虑二分.二分把答案转换为判定.假设二分长度的值为 mid.
由于 Height 数组在局部是单调增的,所以我们 for i=2->N,动态更新长度大于 mid 的连续区间的长度,记为 cnt.
因为每个后缀都位置都是不一样的,所以 cnt 也就是长度为 mid 的可重叠字串出现的最多次数.如果 cnt>=k,
那么 mid 就可行,返回 true,否则返回 false.
*/
template <class T>
void read(T &x) {
    static char ch;static bool f;
    for(ch=f=0;ch<'0' || '9'<ch;f|=ch=='-',ch=getchar());
    for(x=0;'0'<=ch && ch<='9';(x*=10)+=ch-'0',ch=getchar());
    x=f?-x:x;
```

```
}
int arr[MAXN] ;
int n,m ;
bool check(int mid)
{
    int cnt(1) ;
    for(int i(2);i<=sa.N;++i) {
        if(sa.Height[i]>=mid) {
            ++ cnt ;
            if(cnt>=m) return true ;
        }
        else cnt = 1 ;
    }
    return false ;
}

int fen(int l,int r)
{
    while(l<r) {
        int mid((l+r+1)>>1) ;
        if(check(mid)) {
            l = mid ;
        } else {
            r = mid - 1 ;
        }
    }
    return l;
}

int main()
{
    read(n) ;read(m) ;
    ///n 是字符串大小,m 是字符串的长度(这题用的 int 数组代表字符串)
    for(int i(0);i<n;++i) read(arr[i]) ;
    sa.insert(arr,n) ;
    printf("%d\n",fen(0,n)) ;
}
```

③ 多字符串问题处理方法例子

```
/**
Uva12338
```

多字符串拼接的时候,要保证拼接的特殊字符不能相等,所以我们可以把字符串-'a'转换成数字然后用传入参数需要改变成 inline void insert(int* str,int n,int m),其中 m 是指字符集大小

(包含 26 个英文字母和分割符).分隔符可以用一个变量 sepChr,每次加入一个分隔符后,++sepChr,保证分隔符不重复.

下个题是求多个字符串 LCP 的例子.

题意:

本题题意是给你 n 个字符串,m 次查询,查询两个字符串的最长公共前缀.

我们只需要把 n 个字符串记录下标之后进行拼接,放进后缀数组之后,然后利用 st 表预处理求任意两个后缀的 lcp 就可以了.

*/

```
int arr[MAXN] ;
char str1[MAXN] ;
int pos[MAXN],len[MAXN] ;
int main()
{
    int T,t(1) ;
    int n,m,sepChr(30) ;
    scanf("%d",&T) ;
    while(T-->0)
    {
        scanf("%d",&n) ;
        int lenSum(0),len2(0) ;
        for(int i(1);i<=n;++i) {
            scanf("%s",str1) ;
            len2 = strlen(str1) ;
            for(int i(0);i<len2;++i) {
                arr[lenSum + i] = str1[i] - 'a' + 1;
            }
            len[i] = len2 ;
            pos[i] = lenSum + 1;
            lenSum += len2 ;
            if(i != n) arr[lenSum++] = ++ sepChr;
        }
        sa.insert(arr,lenSum,sepChr) ;
        scanf("%d",&m) ;
        printf("Case %d:\n",t++) ;
        while(m--) {
            int i,j ;
            scanf("%d%d",&i,&j) ;
            if(i == j) printf("%d\n",len[i]) ;
            else printf("%d\n",sa.LCP(pos[i],pos[j])) ;
        }
    }
}
```

④ 两个串的最长公共子串 O(nlogn)

```
/*
    我们用一个特殊字符|把两个串隔开,然后合并成一个串.
我们通过后缀数组排序得到的 Sa[i]数组能够知道每一个后缀.
我们从 i=2->N 枚举每一个后缀,然后我们就会发现,Height 具有局部单调递增的性质,
所以我们可以只考虑相邻的两个,也就是 Height[i]的最大值就行了.因为 i 前边只能
是和 i-1 的公共前缀最大.所以我们可以直接暴力后缀 i 和后缀 i-1 只能有一个包含|的
Height 的最大值即可.
*/
int ans(0) ;
for(int i(2);i<=sa.N;++i) {
      if(sa.Height[i]>ans&&sa.len[i]>len2+1&&sa.len[i-1]<=len2) {
            ans = sa.Height[i] ;
      }
      if(sa.Height[i]>ans&&sa.len[i-1]>len2+1&&sa.len[i]<=len2) {
            ans = sa.Height[i] ;
      }
}
printf("%d\n",ans) ;
```

⑤ 子串第 k 次出现的位置[后缀数组+2 次二分+主席树](O(nlogn))

```
/**
        HDU6704
    题意: 给一个字符串,m 次询问,每次询问一个字串出现 k 次的位置.
    思路: 用后缀数组处理一下,那么所有相同子串最后的 sa 都会靠在一起.
          所以找到对应的 Height 位置,然后向左向右延伸直到 LCP 长度不足 r?l+1
          (RMQ 然后二分左右第一个长度比 r-l+1 小的位置),那么就找到了所有和
          S[l]...S[r]长得一模一样的开始位置,然后在主席树上找第 k 大即可.
*/
int fenR(int pos,int n,int len) {
    int l(pos),r(n) ;
    while(l<r) {
        int mid((l+r+1)>>1) ;
        if(sa.LCP(pos,mid,1)>=len) l = mid ;
        else r = mid - 1;
    }
    return l ;
}
int fenL(int pos,int n,int len) {
    int l(1),r(pos) ;
    while(l<r) {
```

```
            int mid((l+r)>>1) ;
            if(sa.LCP(mid,pos,1)>=len) r = mid ;
            else l = mid + 1;
        }
        return l ;
}
int main()
{
        int T ; scanf("%d",&T) ;
        while(T-->0) {
            int n,m ;
            scanf("%d%d%s",&n,&m,str) ;
            tree.clear() ;
            sa.insert(str,n) ;
            for(int i(1);i<=n;++i) tree.insert(i,sa.Sa[i]) ;
            while(m-->0) {
                int l,r,k ;
                scanf("%d%d%d",&l,&r,&k) ;
                int L = fenL(sa.Rank[l],n,r-l+1) ;
                int R = fenR(sa.Rank[l],n,r-l+1) ;
                int ans = R-L+1<k?-1:tree.Kth(L,R,k) ;
                printf("%d\n",ans) ;
            }
        }
}
```

(2) 后缀自动机 O(nlog26)

```
#include <bits/stdc++.h>
using namespace std ;
class SAM
{
static const int MAXN = 1000001;
static const int N = (MAXN << 1) + 5 ;
public:
        map<char,int> Next[N]/**下一个状态*/;
        int link[N]/**上一个状态*/,
            len[N],/**当前状态表示的字串的长度*/
            Size[N]/**当前状态表示的字串的初选次数*/ ;
        vector<int> treeNxt [N];///ParentTree
        int tax[N]/**存放排序的中间值*/,tp[N]/**存放 parent 树拓扑排序的结果*/;
        int last/**上一的状态的位置*/,tot/**状态的总数*/ ;

        inline int newNode(int l = 0) {///新建节点
```

```
        int id(tot++) ; link[id] = -1 ; len[id] = l;
        Next[id].clear() ;
        return id ;
}
inline void clear()///清空树
        {link[0] = -1;tot = 0 ; last = newNode() ;}
inline int insert(char c) {///插入一个字符
        int p(last),cur(newNode(len[p]+1)) ;
        Size[cur] = 1 ;
        while(p!=-1&&!Next[p].count(c))
                {Next[p][c] = cur ; p = link[p] ;}
        if(p == -1) link[cur] = 0 ;
        else {
                int q = Next[p][c] ;
                if(len[p] + 1 == len[q]) link[cur] = q ;
                else {
                        int id = newNode(len[p] + 1) ;
                        int fa = link[id] = link[q] ;
                        Next[id] = Next[q] ;
                        while(p != -1 && Next[p].count(c) && Next[p][c] == q)
                                {Next[p][c] = id ; p = link[p] ;}
                        link[q] = link[cur] = id ;
                }
        }
        return last = cur ;
}
int insert(char* str,int n = 0) {
        n = n?n:strlen(str) ;
        for(int i(0);i<n;++i) insert(str[i]) ;
        return last ;
}
inline void rSort() {///对 len[]进行拓扑排序
        fill(tax,tax+tot,0) ;
        for(int i(0);i<tot;++i) ++tax[len[i]] ;
        for(int i(0);i<tot;++i) tax[i] += tax[i-1] ;
        for(int i(0);i<tot;++i) tp[--tax[len[i]]] = i ;
}
int64_t count() {///按照拓扑序自低向上统计每个子串出现次数.
        int64_t ans(0) ;rSort() ;Size[0] = 0 ;
        for(int i(tot-1),j;i>=1;--i) {
                j = tp[i];    Size[link[j]] += Size[j] ;
                if(Size[j]>1) ans = max(1LL*ans,1LL*len[j]*Size[j]) ;
        }
        return ans ;
```

```
    }
    void buildTree() {///建立 ParentTree(把字符串倒着建立 SAM 得到的是后缀树).
        for(int i(0);i<tot;++i) treeNxt[i].clear() ;
        for(int i(1);i<tot;++i) treeNxt[link[i]].push_back(i) ;
    }
    int64_t dfs(int pos = 0) {///遍历 ParentTree.
        int64_t maxs(0) ;
        for(auto& it: treeNxt[pos]) {
            maxs = max(maxs,dfs(it)) ;
            Size[pos] += Size[it] ;
        }
        if(Size[pos] <= 1) return 0 ;
        return max(1LL*maxs,1LL*Size[pos]*len[pos]) ;
    }
}sam;
char str[1000005] ;
int main()
{

    scanf("%s",str) ;
    int len = strlen(str) ;
    sam.clear() ;
    sam.insert(str,len) ;
    sam.buildTree() ;
    cout<<sam.dfs()<<endl;
}
```

(3) KMP(O(m+n))

```
class Kmp
{///输入字符串从 0 开始,得到的数组从 1 开始
static const int MAXN = 1000005 ;
public:
    int Next[MAXN];
    int64_t f[MAXN] ; ///最大循环节用到的数组
    inline int getNext(char *str,int len) ///求出 Next 数组
    {    ///Next 数组是从 S[0 到 i-1]前子串  的前缀后缀最大值
        int i(0),j(Next[0]=-1);
        while(i<len) {
            if(!~j|| str[i]==str[j]) ///类似于 KMP 的匹配
                Next[++i]=++j;
            else j=Next[j];///失配
        }
        int ret(len-Next[len]) ;///ret:最小循环节
```

```
        ///循环次数: len/ret
        ///一个不完整的循环串至少要补充(ret-(len%ret))个字符才能使得其完整
        return    Next[len]&&!(len%ret)?ret:0;
    }
    inline int KMP(char* s,int lens,char* t,int lent) ///KMP
    {
        getNext(t,lent) ;
        int i(0),j(0),cnt(0);///从 0 位开始匹配
        while(i<lens) { ///临界值
            if(!~j|| s[i]==t[j]) {++i;++j;}///匹配成功,继续.
            else j = Next[j]; ///失配
            if(j == lent) {
                ++ cnt ;
                printf("%d\n",i-j+1);///j==len2 时,匹配成功;i-j+1 即为第一个字母的位
置.
                j = Next[j];///匹配成功后，t2 置为 next[t2]
            }
        }
        return cnt ;///返回成功匹配的次数.
    }
    inline int64_t getMaxLoop(int len)
    {///求最大循环节,别忘了先调用一下 getNext().
        int64_t ret(0) ; fill_n(f,len+1,0) ;
        for(int i(1);i<=len;++i) {
            if(!Next[i]) continue ;
            f[i] = f[Next[i]] + 1LL*i-Next[i] ;
            ret += f[i] ;
        }
        return ret ;
    }
}kmp;
```

(4) Manacher O(2*n)

```
#include <bits/stdc++.h>
const int MAXN (11000005) ;
using namespace std ;
char str[MAXN];
int p[MAXN*2+10] ; ///注意!这里要开二倍+10!!!

int Manacher(char* str,int n)//求出最长回文串的长度
{
    int id(0),mx(0),ans(0) ;
    char* buff = new char[2*n+10] ;
```

```
    buff[0] = '$' ;buff[1] = '#' ;
    int len(2*n+2) ;
    for(int i(0);i<n;++i) {
        buff [2*i+2] = str[i] ;
        buff [2*i+3] = '#' ;
    }buff[len] = 0 ;
    for(int i(1);i<len;++i) {
        if(i<mx) p[i] = min(p[(id<<1)-i],mx-i) ;
        else p[i] = 1 ;
        while(buff[i-p[i]] == buff[i+p[i]]) ++ p[i] ;
        if(p[i] + i>mx) mx = p[i] + i, id = i ;
        ans = max(ans,p[i]) ;
    }
    delete(buff) ;
    return ans - 1 ;
}

int main()
{
    cin >>str ;
    cout<<Manacher(str,strlen(str))<<endl;
}
```

(5) 回文自动机[回文树]O(nlog26)

```
/**
```

类似 AC 自动机的一种回文串匹配自动机,也就是一棵字符树.准确的说,是两颗字符树,0 号表示回文串长度为偶数的树,1 号表示回文串长度为奇数的树.每一个节点都代表一个字符串,并且类似 AC 自动机那样,有字符基个儿子,它的第 i 个儿子就表示将字符基的第 i 个字符接到它表示的字符串两边形成的字符串.

同样类似 AC 自动机的是,每一个节点都有一个 fail 指针,fail 指针指向的点表示当前串后缀中的最长回文串.特殊地,0 号点的 fail 指针指向 1,非 0,1 号点并且后缀中不存在回文串的节点不指向它本身,而是指向 0.

功能:

1.求串 S 本质不同回文串的个数(tot-2)
2.求串 S 内每一个本质不同回文串出现的次数(cnt[i])
3.求串 S 内回文串的个数(其实就是 1 和 2 结合起来)
4.求以下标 i 结尾的回文串的个数(num[i])
5.求以 i 结尾的最长回文串的长度(len[i])
6.本质不同回文串的位置

```
**/
//回文字动机每次新建点就相当于增加了一种回文串,指针记录的是回文串的结尾标号.
class Palindromic_Tree {
public:
```

```
#define DEBUG(x) cout<<#x<<" == "<<x<<","
#define MAXN 300010   ///字符串的长度
#define N 26 ///字符集大小
int Next[MAXN][N] ;//Next 指针,Next 指针和字典树类似,表示编号为 i 的节点表示的回文串在两边添加字符 c 以后变成的回文串的状态编号
int fail[MAXN] ;//fail 指针,表示该状态的最长回文后缀对应的状态编号,失配后跳转到 fail 指针指向的节点
int cnt[MAXN] ;/*调用 count()之后它表示节点状态 i 的出现次数*/
int num[MAXN]/*表示以 i 状态的回文后缀的个数,也表示 i 状态 fail 指针的深度*/ ;
int len[MAXN]/*表示状态 i 表示的回文串的长度*/,S[MAXN] ;/*存放添加的字符*/
int last ;/*指向上一个字符所在的节点,方便下一次 insert*/
int id[MAXN]/*i 字符在树中的状态编号*/,no[MAXN]/*状态 i 的最右边的字符在原串中的位置,从 1 开始*/;
int n /*添加的字符个数*/,tot/*添加的节点个数*/ ;
deque<char> buff ;///DEBUG 输出用的
Palindromic_Tree(){clear() ;}
inline int newNode (int l) {//新建节点
    fill_n(Next[tot],N,0) ;
    num[tot] = cnt[tot] = 0 ;
    len[tot] = l ;
    return tot ++ ;
}
inline void clear () {//初始化
    buff.clear() ;
    tot = 0 ;
    newNode (0) ;/*0 表示偶数长度串的根*/
    newNode ( -1 )/*1 表示奇数长度串的根*/ ;
    last = n = 0 ;fail[0] = 1 ;
    S[n] = -1 ;//开头放一个字符集中没有的字符,减少特判
}
inline int get_fail (int x) {//和 KMP 一样,失配后找一个尽量最长的
    while (S[n-len[x]- 1]!= S[n]) x = fail[x] ;
    return x ;
}
inline void insert(char* str,int n=0)
{
    n = n?n:strlen(str);
   for(int i(0);i<n;++i) id[i] = insert(str[i]) ;
}
inline int insert (int c) {///插入的时候返回节点的编号.
    S[++n] = c -= 'a'; ///!!!要是大写的话,注意这里!!!
    int cur(get_fail(last)) ;//通过上一个回文串找这个回文串的匹配位置
    if (!Next[cur][c]) {//如果这个回文串没有出现过,说明出现了一个新的本质不同的回文串
        int now (newNode(len[cur] + 2)) ;//新建节点
```

```
                fail[now] = Next[get_fail(fail[cur])][c] ;//和 AC 自动机一样建立 fail 指针,以便失配后跳转
                Next[cur][c] = now ;
                num[now] = num[fail[now]] + 1 ;
            }
            ++ cnt[last = Next[cur][c]] ;no[last]=n;
            return last ;
        }
        inline int getNum(int i) ///返回以 i 结尾的不同回文串个数
            {return num[id[i]] ; }
        inline int getLongest(int i)///返回以 i 为结尾的最长回文串长度
            {return len[id[i]] ;}
        inline int size() ///返回本质不同的回文字串数目
            {return tot-2;}

        inline int64_t count ()
        {///  统计 S 中所有节点 i 表示的本质不同的串的个数
            int64_t ans (0) ;
            for (int i(tot-1);i>=0;--i) {///逆序相加,每个点都会比它的父亲节点先算完,于是父亲节点能加到所有子孙.
                cnt[fail[i]] += cnt[i] ;
                if(i>1) ans = max(ans,(int64_t)1*cnt[i]*len[i]);
            ///这里可以算贡献,比如上边求的是所有回文子串在 s 中出现的次数乘以这个回文子串的长度的最大值.
            }
            return ans ;
        }

        void debug()
        {
            for(int i(2);i<tot;++i) {
            ///输出每个本质不同的字符串在原串中的位置,个数,和长度.
                DEBUG(no[i]) ;DEBUG(cnt[i]) ;DEBUG(len[i]) ;
                cout<<endl;
            }
        }

        inline void print0(int x = 0)
        {///输出偶数的回文串
            if(x) {for(auto it:buff) cout<<it; cout<<endl;}
            for(int i(0);i<N;++i) {
                int xx(Next[x][i]) ;
                if(xx) {
                    buff.push_front('a'+i),buff.push_back('a'+i) ;
```

```
                print0(xx) ;
                buff.pop_front(),buff.pop_back() ;
            }
        }

    }

    inline void print1(int x = 1)
    { ///输出奇数的回文串
        if(x-1) {for(auto it:buff) cout<<it; cout<<endl;}
        for(int i(0);i<N;++i) {
            int xx(Next[x][i]) ;
            if(xx) {
                buff.push_front('a'+i) ;
                if(x-1) buff.push_back('a'+i) ;
                print1(xx) ;
                buff.pop_front() ;
                if(x-1) buff.pop_back() ;
            }

        }
    }
    inline void print()
    {///输出所有不相同的回文串
        print1() ; print0() ;
    }
    #undef N
    #undef MAXN
} tree;
```

(6) 字符串哈希+判断是否回文 O(n)

```
///O(1)判断字串是否回文: getHash(l,r)==getHash(len+1-r,len+1-l)
const int MAXN (100005) ;
class StrHash
{
public:
    #define P (131) ///注意!!!这里是 uint64_t!!!!
    uint64_t Hash[MAXN],p[MAXN] ;///注意!!!这里是 uint64_t!!!!

    inline void insert(char* str,int len=0)
    {//正常输入字符串,str 首指针
        len = len?len:strlen(str) ;
        p[0] = 1;//131^0
        for(int i(1);i<=len;++i) {
```

```
            Hash[i] = Hash[i-1]*P +(str[i-1]-'a'+1) ;
            p[i] = p[i-1]*P ;
        }
    }

    inline uint64_t getHash(int l,int r)//l 从 1 开始
        {return Hash[r] - Hash[l-1]*p[r-l+1] ;}
    uint64_t operator ()(int l,int r)////l 从 1 开始
        {return getHash(l,r) ;}
}hashs;
```

(7) 子序列自动机 O(nlog26)

```
#include <bits/stdc++.h>
using namespace std ;
class Subsequence_Automaton
{
static const int MAXN = 100005 ;
public:
    int tot,link[MAXN];
    int64_t Cnt[MAXN] ;
    map<char,int> Next[MAXN],last,root ;
    void clear() {tot = 0; last.clear() ;root.clear() ;}

    int newNode()
    {
        int id(++tot) ;
        Next[id].clear() ; link[id] = -1 ;
        return id ;
    }

    int insert(char c)
    {
        int id(newNode()) ;
        if(!last.count(c)) Next[0][c] = root[c] = id ;
        else link[id] = last[c] ;
        for(char i('a');i<='z';++i) {
            if(last.count(i))
            for(int j(last[i]);~j&&!Next[j].count(c);j=link[j]) Next[j][c] = id ;
        }
        return last[c] = id ;
    }
```

```
    void insert(char* str,int len = 0)
    {
        if(!len) len = strlen(str) ;
        for(int i(0);i<len;++i) insert(str[i]) ;
    }

    bool find(char* str,int len = 0)
    {///判断一个串是不是另一个串的子序列
        int nowpos;
        if(!len) len = strlen(str);
        for(int i(0);i<len;++i) {
            char c(str[i]);
            if(!i) {
                if(!root.count(c)) return false ;
                nowpos = root[c];
            } else {
                if(!Next[nowpos].count(c)) return false ;
                nowpos = Next[nowpos][c];
            }
        }
        return true ;
    }

    int64_t count()
    {///返回本质不同子序列的个数
        int64_t ans(0) ;
        for(char i('a');i<='z';++i) {
            if(!root.count(i)) continue ;
            dfs(root[i]); ans += Cnt[root[i]];
        }
        return ans ;
    }
    void dfs(int pos = 0)
    {
        Cnt[pos] = 0 ;
        for(auto& it:Next[pos]) {
            dfs(it.second) ; Cnt[pos] += Cnt[it.second] ;
        }
        ++ Cnt[pos] ;
    }
}subam;

int main()
{
```

```
    subam.clear() ;
    subam.insert("abc") ;
    cout<<subam.count()<<endl;
}
```

(8) 拓展 kmpO(n+m)

```
class  KmpEx
{///输入和得到的数组下标都从 0 开始!!!!!!!!!!
public:
    static const int MAXN = 100005 ;///字符串最大长度
    int Next[MAXN],Extend[MAXN];
    ///Next[i]:    T[i,lent)与 T 的最长公共前缀长度.
    ///Extend[i]: 从 i 开始,S 和 T 一直相等的最长可拓展长度(后缀的最长公共前缀).
    void KMPEX(char* s,int lens,char* t,int lent)///s 为母串,t 为子串
    {
        int i,j(0),k(1),p,L;
        Next[0]=lent;
        while(j+1<lent && t[j]==t[j+1]) ++ j;
        Next[1]=j;
        for(i = 2;i < lent; ++i) {
            p=Next[k]+k-1; L=Next[i-k];
            if(i+L < p+1) Next[i] = L ;
            else {
                j = max(0,p-i+1);
                while(i+j<lent && t[i+j]==t[j]) ++j ;
                Next[k = i] = j ;
            }
        }
        j = 0 ;
        while(j<lens && j<lent && s[j]==t[j]) ++ j ;
        Extend[0] = j ;
        for(i=1,k=0;i<lens; ++i) {
            p = Extend[k]+k-1; L = Next[i-k];
            if(i+L < p+1) Extend[i] = L;
            else {
                j = max(0,p-i+1);
                while(i+j<lens && j<lent && s[i+j]==t[j])++ j;
                Extend[k = i] = j ;
            }
        }
    }
}kmpex;
```

12. 其他

(1) Bitset

bitset<n> b;//b 有 n 位，每位都为 0
bitset<n> b(u);//b 是 unsigned long 型 u 的一个副本
bitset<n> b(s);//b 是 string 对象 s 中含有的位串的副本
bitset<n> b(s, pos, n);//b 是 s 中从位置 pos 开始的 n 个位的副本
b.any()//b 中是否存在置为 1 的二进制位?
b.none()//b 中不存在置为 1 的二进制位吗?
b.count()//b 中置为 1 的二进制位的个数
b.size()//b 中二进制位的个数
b[pos]//访问 b 中在 pos 处的二进制位
b.test(pos)//b 中在 pos 处的二进制位是否为 1?
b.set()//把 b 中所有二进制位都置为 1
b.set(pos)//把 b 中在 pos 处的二进制位置为 1
b.reset()//把 b 中所有二进制位都置为 0
b.reset(pos)//把 b 中在 pos 处的二进制位置为 0
b.flip()//把 b 中所有二进制位逐位取反
b.flip(pos)//把 b 中在 pos 处的二进制位取反
b.to_ulong() //用 b 中同样的二进制位返回一个 unsigned long 值
b.to_ullong() //用 b 中同样的二进制位返回一个 unsigned long long 值
b.to_string() //用 b 中同样的二进制位返回一个 string 值
os << b ;//把 b 中的位集输出到 os 流
b._Find_first();// 找到从低位到高位第一个 1 的位置
b._Find_next(pos) ;//找到当前位置的下一个 1 的位置,如果某个元素之后没有元素的话会返回 bitset 的大小

(2) C++浮点数

float 的 示数精度是 6-7 位(包括整数和小数总共)
double 示数精度是 15-16 位(包括整数和小数总共)
long double 示数精度是 18-19 位(包括整数和小数总共)
__float128 示数精度是 32-33 位(包括整数和小数总共) [目前没有办法输入输出]
#include <decimal/decimal> using namespace decimal ;可以用 decimal128 完成运算.

int isfinite(x) :判断 x 是否有限,是返回 1,其它返回 0.
int isnormal(x) :判断 x 是否为一个数(非 inf 或 nan),是返回 1,其它返回 0.
int isnan(x) :当 x 时 nan 返回 1,其它返回 0.
int isinf(x) :当 x 是正无穷是返回 1,当 x 是负无穷时返回-1,其它返回 0.有些编译器不区分.

fix(x) :保留整数.

double modf(double x,double *integer)

modf()是分解 x，以得到 x 的整数和小数部分。
返回 x 的小数部分,符号与 x 相同.
*integer 是 x 的整数部分,是浮点值,integer 是指向一个对象的指针.

double fmod (double x,double Mod); 浮点数取模,返回浮点数下的 x%Mod.

(3) Java 重定向输入输出到文件

```
FileInputStream instream = null;
FileInputStream instream = null;
PrintStream outstream = null;
try {
    instream = new FileInputStream("D:/in.txt");
    outstream = new PrintStream(new FileOutputStream("D:/out.txt"));
    System.setIn(instream);
    System.setOut(outstream);
} catch (Exception e) {
    System.err.println("Error Occurred.");
}
```

(4) 其他 STL

fill(start,end,x): [start,end) 这段区间被初始化为: x
fill_n(start,cnt,x)[start,start+cnt] 这段区间被初始化为: x
iota(start,end,x): [start,end) 这段区间被初始化为: x,x+1,x+2,x+3.....
vector<int> v;
fill_n(back_inserter(v),10,1); ///插入 10 个 1 到 vector
back_inserter 函数是一种增量迭代插入的方式，这种方法可以动态的填装数据，避免了容量不足的情况。

copy(sStart,sEnd,tStart): [sStart,sEnd) 这段区间复制到以 tStart 开始的位置.
copy_n(sStart,n,tStart) ; [sStart,sStart+n) 这段区间复制到以 tStart 开始的位置.
copy 还可以用来向前移动数组,例如: copy_n(arr,n,arr-m),把[arr,arr+n)整体向前移动到以 arr-m 开头的地方.
注意!!只能向前移!!不能向后移!!!!

accumulate(start,end,init) 计算 init 和[start,end)内所有元素的总和,需要提供初始值 init
partial_sum(start,end,tStart,op) 预处理[start,end)的前缀和并且存放到以 tStart 开始的位置.op 默认是加法，还可以自定义。
op:
std::and: 二元谓词对象类,x& y;

std::or: 二元谓词对象类,x |y;

std::xor: 二元谓词对象类， x ^y;

std::plus: 二元谓词对象类,x +y;

std::minus: 二元谓词对象类,x -y;

std::multiplies: 二元谓词对象类,x *y;

std::divides: 二元谓词对象类,x /y;

std::modulus: 二元谓词对象类,x %y;

std::negate: 一元谓词对象类, -x;

std::equal_to: 二元谓词对象类,x ==y;

std::not_equal_to:二元谓词对象类,x != y;

std::greater: 二元谓词对象类,x >y;

std::less: 二元谓词对象类,x <y;

std::greater_equal:二元谓词对象类,x >= y;

std::less_equal:二元谓词对象类,x <= y;

std::logical_and:二元谓词对象类,x && y;

std::logical_or:二元谓词对象类,x || y;

std::logical_not:一元谓词对象类, !x;

比如: int op(int a,int b) {return a^b} ;
adjacent_difference(start,end,tStart,op) 预处理[start,end)的差分数组并且存放到以 tStart 开始的位置.op 默认是减法，还可以自定义。
inner_product(start1,end1,start2,op1,op2) 计算内积 op1 和 op2 支持自定义,默认 op1 是加,op2 是乘,算法先计算第二个操作符.

partition(first,last,p)
对[first,last)元素进行处理,使得满足 p 的元素移到[first,last)前部,不满足的移到后部,返回第一个不满足 p 元素所在的迭代器,如果都满足的话返回 last.
stable_partition(first,last,p) 和上边的函数功能一样,只不过保证元素位置不变.
inplace_merge(first,middle,last); 把[first,middle)和[middle,last)两个已经有序的序列合并成一个有序的序列(本身就是 stable 的).

set_intersection(st1,ed1,st2,ed2,st3) ; [A∩B] 对[st1,ed1) 和[st2,ed2)进行集合交集运算,并存到 st3 开始的位置.要求两个集合有序.
如果 st3 是 set 或者其他容器的话,可以用 back_inserter(s) ;
set_union 并集 [A∪B],用法同上
set_difference 差集[A-B] 用法同上
set_symmetric_difference 对称差[A∪B-(A∩B)] 用法同上
如果使用比较函数,末尾再追加一个参数.

(5) 黑科技

```
inline int LOG2(unsigned int x){   ///x=2^k
      static                                                                                    int
biao[32]={31,0,27,1,28,18,23,2,29,21,19,12,24,9,14,3,30,26,17,22,20,11,8,13,25,16,10,7,15,6,5,4};
      return biao[x*263572066>>27];
}

inline long long mulEx(long long a,long long b,long long Mod){   ///a*b%p
     long long ans(0);
     __asm__ __volatile__ ("\tmovq %1, %%rax \n\t imulq %2 \n\t idivq %3\n" : "=d"(ans): "m"(a), "m"(b),
"m"(Mod) : "%rax");
     return ans;
}///可能编译会出问题,不过交上去能 AC

inline int64_t mulEx(int64_t a,int64_t b,int64_t Mod){ ///O(1)快速乘
      return (a*b-(int64_t)((__float128)a/Mod*b)*Mod+Mod)%Mod;
}
```

```
inline int random(){///高效随机函数
    static int seed(233); //seed 可以随便取
    return seed=int(seed*48271LL%2147483647);
}

///随机打乱一个数组(或者是容器)
int a[10] ;
random_shuffle(a,a+10) ;

///O(n)得到第 k 小
//arr  从 0  开始
nth_element(arr, arr+k-1,arr+len)
cout<<arr[k-1]
```

可以使第 n 大元素处于第 n 位置(从 0 开始,其位置是下标为 n 的元素),并且比这个元素小的元素都排在这个元素之前(不保证有序)

一个 10 进制数转换为二进制的位数为：

```
ceil(lgN/lg2) + 1

///红黑树声明:
template <typename T1,typename T2 = null_type>
class pbTree: public tree<T1,T2,less<T1>,rb_tree_tag,tree_order_statistics_node_update>{};
```

(6) 离散化

```
class Dispersed
{
static const int MAXN = 1000005 ;
public:
    int a[MAXN];
    int original[MAXN],len ;
    ///原来的 arr[]     100 2 5 8
    ///离散化后的 arr[]          4       1 2 3
    ///离散化后 original[]   2       5 8 100
    void dispersed(int *arr,int n)
    {///离散化:arr 下标从 1 开始离散化
     ///arr 存的是离散化之后的值,原来的对应值存在 original 里.
        for(int i(1);i<=n;++i) a[i] = arr[i] ;
        sort(a+1,a+n+1) ;
        len = unique(a+1,a+n+1)-(a+1) ;
        for(int i(1);i<=n;++i) {
            int buff = arr[i] ;
            arr[i] = getDispersed(arr[i]);
            original[arr[i]] = buff ;
```

```
        }
    }
    int getDispersed(int x)///得到离散化之后的值
        {return lower_bound(a+1,a+1+len,x) - a;}
};
```

(7) 神奇的 O(1)算法

O(1) 求 1~n 的异或和：inline int calc(int n){int t(n&3);return t&1?(t/2^1):(t/2^n);}

(8) 输入输出外挂

```
//普通输入输出挂:
///getchar 可以换成这个究极快读!!!!!
///  按下 ctril + z 结束输入才能看到结果!!!!
const int SIZ =100000 +3;
char buf1[SIZ];
char *p1=buf1,*p2=buf1;
inline int getcharEx(){
    if(p1==p2)p1=buf1,p2=buf1+fread(buf1,1,SIZ,stdin);
    return p1==p2?EOF:*p1++;
}
#define getcharEx getchar
template <class T>
inline void read(T &x) {
    static int ch;static bool f;
    for(ch=f=0;ch<'0' || '9'<ch;f|=ch=='-',ch=getcharEx());
    for(x=0;'0'<=ch && ch<='9';x=((x+(x<<2))<<1)+ch-'0',ch=getcharEx());
    x=f?-x:x;
}

template <class T>
inline void write(T x)
{
    if (x < 0) putchar('-'),x = -x;
    if (x >= 10) write (x / 10);
    putchar(x % 10 + '0');
}


///文件流究极输入输出挂!:
#define en *++now_w = '\n' ///换行
#define sp *++now_w = 0x20 ///空格
const int ch_top=4e7;
char ch[ch_top],*now_r=ch-1,*now_w=ch-1;
```

```
template<typename T>
inline void read(T& x)
{
	while(*++now_r<'0');
	x = (*now_r-'0');
	while(*++now_r>='0')x=x*10+*now_r-'0';
}
template<typename T>
inline void write(T x)
{
	static char st[20];static int top;
	while(st[++top]='0'+x%10,x/=10);
	while(*++now_w=st[top],--top);
}

int main()
{
	fread(ch,1,ch_top,stdin); ///第一句话一定是这个!
	int n=read(),m=read() ;
	///balabalabala...
	write(n+m) ; en ;
	fwrite(ch,1,now_w-ch,stdout); ///最后一句话一定是这个!

}
```

(9) 位运算__builtin_

枚举一个二进制状态的子集:

```
int S = 0b1101 ;
for(int state(S);state;state=(state-1)&S)
state^S ///补集
```

O(1) 求 1~n 的异或和：inline int calc(int n){int t(n&3);return t&1?(t/2^1):(t/2^n);}

1.__builtin_popcount(unsigned int n)

该函数时判断 n 的二进制中有多少个 1

```
int n = 15; //二进制为 1111
cout<<__builtin_popcount(n)<<endl;//输出 4
```

2.__builtin_parity(unsigned int n)

该函数是判断 n 的二进制中 1 的个数的奇偶性

```
int n = 15;//二进制为 1111
int m = 7;//111
```

```
cout<<__builtin_parity(n)<<endl;//偶数个，输出 0
cout<<__builtin_parity(m)<<endl;//奇数个，输出 1
```

3.__builtin_ffs(unsigned int n)
该函数判断 n 的二进制末尾最后一个 1 的位置，从一开始

```
int n = 1;//1
int m = 8;//1000
cout<<__builtin_ffs(n)<<endl;//输出 1
cout<<__builtin_ffs(m)<<endl;//输出 4
```

4.__builtin_ctz(unsigned int n)
该函数判断 n 的二进制末尾后面 0 的个数，当 n 为 0 时，和 n 的类型有关

```
int n = 1;//1
int m = 8;//1000
cout<<__builtin_ctzll(n)<<endl;//输出 0
cout<<__builtin_ctz(m)<<endl;//输出 3
```

5. __builtin_clz (unsigned int x)
返回前导的 0 的个数。

6. uint16_t __builtin_bswap16 (uint16_t x)
uint32_t __builtin_bswap32 (uint32_t x)
按字节翻转 x，返回翻转后的结果。

7. __builtin_constant_p (exp)
判断 exp 是否在编译时就可以确定其为常量，如果 exp 为常量，该函数返回 1，否则返回 0。如果 exp 为常量，可以在代码中做一些优化来减少处理 exp 的复杂度。

8. __builtin_prefetch (const void *addr, ...)
它通过对数据手工预取的方法，在使用地址 addr 的值之前就将其放到 cache 中，减少了读取延迟，从而提高了性能，但该函数也需要 CPU 的支持。该函数可接受三个参数，第一个参数 addr 是要预取的数据的地址，第二个参数可设置为 0 或 1（1 表示我对地址 addr 要进行写操作，0 表示要进行读操作），第三个参数可取 0-3（0 表示不用关心时间局部性，取完 addr 的值之后便不用留在 cache 中，而 1、2、3 表示时间局部性逐渐增强）。

(10)优先队列+搜索解决第 K 小团

```
//priority_queue<int,vector<int>,greater<int> > Q ; ///堆顶最小
//priority_queue<int,vector<int>,less<int> > Q ; ///堆顶最大

#include <bits/stdc++.h>
using namespace std ;
int64_t val[105] ;
```

```
char str[105] ;
bitset<105> bg[105] ;
struct Node
{
    int64_t sum;
    int pMax ;
    bitset<105> isok ;
    bool operator < (const Node &b)const {return sum>b.sum;}
};

int64_t bfs(int n,int k)
{
    std::priority_queue<Node> que ;
    que.push(Node{0,-1,0}) ;
    while(que.size()) {
        Node tp = que.top(); que.pop() ; --k ;
        if(k == 0) return tp.sum ;
        for(int i(tp.pMax+1);i<n;++i) {///记录上次搜索的下标,防止搜重复
            if(!tp.isok[i]&&(tp.isok&bg[i]) == tp.isok) {
                tp.isok[i] = 1 ;
                que.push(Node{tp.sum + val[i],i,tp.isok}) ;
                tp.isok[i] = 0 ;
            }
        }
    }
    return    -1;
}
int main()
{
    int n,k ;
    scanf("%d%d",&n,&k) ;
    for(int i(0);i<n;++i) scanf("%lld",val+i) ;
    for(int i(0);i<n;++i) {
        scanf("%s",str) ;
        for(int j(0);j<n;++j) bg[i][j] = str[j] == '1' ;
    }
    printf("%lld\n",bfs(n,k)) ;
    return 0;
}
```

(11) 智能指针

一般用 shared_ptr<T>,智能指针支持自动内存回收.
unique_ptr<T> 可以开动态数组

定义: shared_ptr<int> sp;
sp = make_shared<int>() ;///括号里可以初始化.
清空: sp = nullptr 或者 sp = NULL 或者 sp = 0
智能指针内部有一个计数器,当一个地址没人用的时候,会被自动回收.
查看被使用的次数: sp.use_count()
赋值新的指针: sp.reset(new int) ;
得到真实地址: sp.get() /*可以修改指针指向的内容*/

(12)获取 CPU 型号以及 ID

```
#include <stdint.h>
#include <iostream>
#include <cpuid.h>
static void cpuid(uint32_t func, uint32_t sub, uint32_t data[4]) {
	__cpuid_count(func, sub, data[0], data[1], data[2], data[3]);
}
int main() {
	uint32_t data[4];
	char str[48];
	for(int i = 0; i < 3; ++i) {
		cpuid(0x80000002 + i, 0, data);
		for(int j = 0; j < 4; ++j)
			reinterpret_cast<uint32_t*>(str)[i * 4 + j] = data[j];
	}
	std::cout << str;
}
```

(13)最小出栈顺序 O(n)

```
/*字典序最小的出栈顺序*/
#include <bits/stdc++.h>
using namespace std ;
const int MAXN (3000005) ;
char str[MAXN] ;
char minChr[MAXN] ;
int main()
{
	scanf("%s",str) ;
	int len(strlen(str)) ;
	minChr[len-1] = str[len-1] ;
	for(int i(len-2);i>=0;--i)	{minChr[i] = min(minChr[i+1],str[i]) ;}
	vector<char> v ;	stack<char> sta ;
	for(int i(0);i<len;++i) {
		while(sta.size()) {
			int chr = sta.top() ;
```

```
                if(chr<=minChr[i]) {
                    v.push_back(chr) ;
                    sta.pop() ;
                } else break ;
            }
            sta.push(str[i]) ;
        }
        while(sta.size()) {v.push_back(sta.top());sta.pop();}
        for(auto& it:v) {
            putchar(it) ;
        }
        putchar('\n') ;
}
```