杂项

最大化最小值（二分值最小越小越容易满足条件，求最大值）

区间长度为 1 时的写法：
解的范围为

```
// 计算区间为[lb, rb]
while( rb > lb )  // 区间长度为1 时终止循环
{
    // 防止溢出
    int m = lb + (rb - lb + 1) / 2;     // 由于是区间长度为1 时终止循环，所以要加1
    if( ok(m) ) lb = m;
    else rb = m - 1;
}
// 跳出循环时 lb == rb
```

区间长度为 2 时的写法：
解的范围为

```
while( rb - lb > 1 )  // 区间长度为2 时终止循环
{
    // 防止溢出
    int m = lb + (rb - lb) / 2;     // 由于是区间长度为2 时终止循环，所以不用加1（不会死循环）
    if( ok(m) ) lb = m;
    else rb = m;
}
// 跳出循环时 lb + 1 == rb
// 答案为 lb
```

最小化最大值（二分值越大越容易满足条件，求最小值）

区间长度为 1 时的写法：
解的范围为

```
while( rb > lb )
{
    // 防止溢出
    int m = lb + (rb - lb) / 2;      // 这里虽然区间长度为
1，但不需要加1（不会死循环）
    if( ok(m) ) rb = m;
    else lb = m + 1;
}
// 跳出循环时 lb == rb
```

区间长度为 2 时的写法：
解的范围为

```
while( rb - lb > 1 )
{
    // 防止溢出
    int m = lb + (rb - lb) / 2;
    if( ok(m) ) rb = m;
    else lb = m;
}
// 跳出循环时 lb + 1 == rb
// 答案为 rb
```

浮点数的二分，100 次循环可以达到 2^{-100}(约为10^{-30})的精度范围

以最大化最小值为例（即小于该数的解均满足条件）

```
for( int i = 0; i < 100; ++ i )
{
    double m = (lb + rb) / 2;
    if(check(m)) lb=m;
     else rb=m;
}
// 跳出循环时 lb 与 rb 近似相等，所以都可作为答案
```

莫队

```
//动态
struct unit {
        int l, r, id;
  bool operator<(const unit &x) const {
    return l / sq == x.l / sq
                ? (r == x.r ? 0 : ((l / sq) & 1) ^ (r < x.
r))
                : l < x.l;//莫队奇偶优化
  }
};

int totq=0,totxiu=0,sq,ans[N],ans1=0,n,m,a[N],sum[N],l=1,
r=0,now=0;

int main()
{
    scanf("%d%d",&n,&m);
    for (int i=1;i<=n;i++) scanf("%d",&a[i]);
    for (int i=1;i<=m;i++)
    {
        char s[2];
        int x,y;
        scanf("%s%d%d",&s,&x,&y);
        if (s[0]=='Q')
        {
            totq++;
            q[totq].l=x; q[totq].r=y;
            q[totq].x=totxiu;
            q[totq].id=totq;
        }
        else
        {
            totxiu++;
            xiu[totxiu].id=x;
            xiu[totxiu].cl=y;
        }
    }
    sq=sqrt(n*1.0);
    sort(q+1,q+1+totq,cmp);
         for(int i=1;i<=totq;i++)
         {
        while(l>q[i].l)l--,ins(l);
        while(r<q[i].r)r++,ins(r);
                 while(l<q[i].l)del(l),l++;
                 while(r>q[i].r)del(r),r--;

                 while(now<q[i].x)change(now+1),now++;
                 while(now>q[i].x)change(now),now--;
                 ans[q[i].id]=ans1;
         }
         for(int i=1;i<=totq;i++)
         printf("%d\n",ans[i]);
}

//基本
struct node {
    ll l,r,id,A,B;
}Q[maxn];
ll be[maxn],a[maxn],sq,num,l=1,r=0,sum[maxn];
ll ans1=0;
ll gcd(ll a,ll b)
 {
         if(b==0)return a;
         else return gcd(b,a%b);
 }
bool cmp1(const node& A,const node&B) {return be[A.l]==b
e[B.l]?A.r<B.r:A.l<B.l;}
bool cmp2(const node& A,const node&B){return A.id<B.id;}
void ins(int x) {ans1=ans1+2*sum[x];sum[x]++;}
void del(int x) {ans1=ans1-2*sum[x]+2;sum[x]--;}
int main()
{
        scanf("%lld%lld",&n,&m);
        sq=sqrt(n);
        for(ll i=1;i<=n;i++)
        scanf("%d",&a[i]);
        for(ll i=1;i<=m;++i)
                {
            scanf("%d%d",&Q[i].l,&Q[i].r);
            Q[i].id=i;
        }
        for(ll i=1;i<=n;++i){
         be[i]=i/sq+1;
                 }
                 sort(Q+1,Q+1+m,cmp1);
        for(int i=1;i<=m;i++){
                     while(l<Q[i].l)del(a[l]),++l;
            while(l>Q[i].l)--l,ins(a[l]);
            while(r<Q[i].r)++r,ins(a[r]);
            while(r>Q[i].r)del(a[r]),--r;
            if(Q[i].l==Q[i].r){Q[i].A=0;Q[i].B=1;continu
e;}
                                Q[i].B=ans1;Q[i].A=(Q[i].r-Q
[i].l+1)*(Q[i].r-Q[i].l);
        }
        sort(Q+1,Q+1+m,cmp2);
        for(ll i=1;i<=m;i++){
         if(Q[i].A==0)printf("0/1\n");
         else {
                 int g=gcd(Q[i].A,Q[i].B);
             printf("%lld/%lld\n",Q[i].B/g,Q[i].A/g);
                         }
                 }
    return 0;
}
```

三分

```
// 整数
while(l+1<r)
{
    int lm=(l+r)>>1,rm=(lm+r)>>1;
    if(judge(lm)>judge(rm))
        r=rm;
    else
        l=lm;
}
// double

while(l+eps<r)
{
    double lm=(l+r)/2,rm=(lm+r)/2;
    if(judge(lm)>judge(rm))
        r=rm;
    else
        l=lm;
}
```

数位 dp

```
int dfs(int pos, int lim1, int lim2, bool zero)
{
    if (pos== -1) return 1;
    if (dp[pos][lim1][lim2] != -1)
        return dp[pos][lim1][lim2];
    int up1 = lim1 ? a[pos] : 1;
    int up2 = lim2 ? b[pos] : 1;
    int tmp1 = 0, tmp2 = 0;
```

```
    for (int i = 0; i <= up1; i++)
        for (int j = 0; j <= up2; j++)
        {
            if (i & j)
                continue;
            int tmp = dfs(pos - 1, lim1 && (i == up1), l
im2 && (j == up2), zero || (i ^ j));
            if (!zero && (i ^ j))
            {
                tmp1 = (tmp1 + tmp) % mod;
            }
            tmp2 = (tmp2 + tmp) % mod;
        }
    ans = (ans + tmp1 * (pos + 1) % mod) % mod;
    dp[pos][lim1][lim2] = tmp2;
    return tmp2;
}
void cw(int x, int y)//拆位
{
    int pos1 = 0, pos2 = 0;
    while (x)
    {
        a[pos1++] = x & 1;
        x >>= 1;
    }
    while (y)
    {
        b[pos2++] = y & 1;
        y >>= 1;
    }
    for (int i = pos1 - 1; i >= pos2; i--) b[i] = 0;
    dfs(pos1 - 1, 1, 1, 0);
}
```

枚举子集

```
for(int now=S;S!=0;now=(now-1)&S)
{
        tmp=S^now;//now 为子集，tmp 为now 的补集
}
```

SOSDP（高维前缀和）

```
/*
f[i]代表i 所代表的所有子集之和
*/
for(int j=0;j<n;j++)//枚举每一位
        for(int i=0;i<1<<n;i++)//枚举每个可能的集合
                if(i>>j&1)//该位为1
                        f[i]+=f[i^(1<<j)];//代表i 中缺
少了任意一个物品的集合
```

随机数

```
unsigned seed=std::chrono::system_clock::now().time_sinc
e_epoch().count();
        mt19937 rand(seed);
    uniform_int_distribution<int> dis(0,1000000000);
    dis(rand);//最后生成的随机数
```

###

字符串

Manacher

```
vector<int> d1(n);
for (int i = 0, l = 0, r = -1; i < n; i++) {
  int k = (i > r) ? 1 : min(d1[l + r - i], r - i);
  while (0 <= i - k && i + k < n && s[i - k] == s[i + k])
 {
    k++;
  }
  d1[i] = k--;
  if (i + k > r) {
    l = i - k;
    r = i + k;
  }
}

vector<int> d2(n);
for (int i = 0, l = 0, r = -1; i < n; i++) {
  int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i +
1);
  while (0 <= i - k - 1 && i + k < n && s[i - k - 1] ==
s[i + k]) {
    k++;
  }
  d2[i] = k--;
  if (i + k > r) {
    l = i - k - 1;
    r = i + k;
  }
}
```

字典树

```
#include <cstdio>
const int N = 500010;
char s[60];
int n, m, ch[N][26], tag[N], tot = 1;
int main() {
        scanf("%d", &n);
        for (int i = 1; i <= n; ++i) {
                scanf("%s", s + 1);
                int u = 1;
                for (int j = 1; s[j]; ++j) {
                        int c = s[j] - 'a';
                        if (!ch[u][c]) ch[u][c] = ++to
t;
                        u = ch[u][c];
                }
                tag[u] = 1;
        }
        scanf("%d", &m);
        while (m--) {
                scanf("%s", s + 1);
                int u = 1;
                        for (int j = 1; s[j]; ++j) {
                                int c = s[j] - 'a';
                                u = ch[u][c];
                                if (!u) break; // 不
存在对应字符的出边说明名字不存在
                        }
                if (tag[u] == 1) {
                        tag[u] = 2;
                        puts("OK");
                }
                else if (tag[u] == 2)
                        puts("REPEAT");
                else
                        puts("WRONG");
        }
        return 0;
}
```

后缀数组

```
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <iostream>
using namespace std;
const int N = 1000010;
char s[N];
int n, sa[N], rk[N << 1], oldrk[N << 1], id[N], cnt[N];
int main() {
        int i, m, p, w;
        scanf("%s", s + 1);
        n = strlen(s + 1);
        m = max(n, 300);
        for (i = 1; i <= n; ++i) ++cnt[rk[i] = s[i]];
        for (i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
        for (i = n; i >= 1; --i) sa[cnt[rk[i]]--] = i;
        for (w = 1; w < n; w <<= 1) {
                memset(cnt, 0, sizeof(cnt));
                for (i = 1; i <= n; ++i) id[i] = sa[i];
                for (i = 1; i <= n; ++i) ++cnt[rk[id[i]
 + w]];
                for (i = 1; i <= m; ++i) cnt[i] += cnt
[i - 1];
                        for (i = n; i >= 1; --i) sa[cn
t[rk[id[i] + w]]--] = id[i];
                memset(cnt, 0, sizeof(cnt));
                for (i = 1; i <= n; ++i) id[i] = sa[i];
                for (i = 1; i <= n; ++i) ++cnt[rk[id
[i]]];
                for (i = 1; i <= m; ++i) cnt[i] += cnt
[i - 1];
                for (i = n; i >= 1; --i) sa[cnt[rk[id
[i]]]--] = id[i];
                memcpy(oldrk, rk, sizeof(rk));
                for (p = 0, i = 1; i <= n; ++i) {
                        if (oldrk[sa[i]] == oldrk[sa[i
 - 1]] &&
                                oldrk[sa[i] + w] == o
ldrk[sa[i - 1] + w]) {
                                rk[sa[i]] = p;
```

```
                }
                else {
                        rk[sa[i]] = ++p;
                }
        }
}
```

后缀自动机的各种应用

```
struct state {
        int len, link;
        std::map<char, int> next;//可以考虑少一个 log
};
// SAM 本身将会存储在一个 state 结构体数组中。我们记录当前自动机的大小 sz 和变量 last ，当前整个字符串对应的状态。
const int MAXLEN = 100000;
state st[MAXLEN * 2];
int sz, last;
//我们定义一个函数来初始化 SAM（创建一个只有初始状态的 SAM）。
void sam_init() {
        st[0].len = 0;
        st[0].link = -1;
        sz++;
        last = 0;
}
//最终我们给出主函数的实现：给当前行末增加一个字符，对应地在之前的基础上建造自动机。
void sam_extend(char c) {
        int cur = sz++;
        st[cur].len = st[last].len + 1;
        int p = last;
        while (p != -1 && !st[p].next.count(c)) {
                st[p].next[c] = cur;
                p = st[p].link;
        }
        if (p == -1) {
                st[cur].link = 0;
        }
        else {
                int q = st[p].next[c];
                if (st[p].len + 1 == st[q].len) {
                        st[cur].link = q;
                }
                else {
                        int clone = sz++;
                        st[clone].len = st[p].len + 1;
                        st[clone].next = st[q].next;
                        st[clone].link = st[q].link;
                       while (p != -1 && st[p].next[c]
 == q) {
                                st[p].next[c] = clone;
                                p = st[p].link;
                        }
                        st[q].link = st[cur].link = cl
one;
                }
        }
        last = cur;
}
```

应用

下面我们来看一些可以用 SAM 解决的问题。简单起见，假设字符集的大小 k 为常数。这允许我们认为增加一个字符和遍历的复杂度为常数。

检查字符串是否出现

> 给一个文本串 T 和多个模式串 P，我们要检查字符串 P 是否作为 T 的一个子串出现。

我们在 $O(|T|)$ 的时间内对文本串 T 构造后缀自动机。为了检查模式串 P 是否在 T 中出现，我们沿转移（边）从 t_0 开始根据 P 的字符进行转移。如果在某个点无法转移下去，则模式串 P 不是 T 的一个子串。如果我们能够这样处理完整个字符串 P，那么模式串在 T 中出现过。

对于每个字符串 P，算法的时间复杂度为 $O(|P|)$。此外，这个算法还找到了模式串 P 在文本串中出现的最大前缀长度。

不同子串个数

> 给一个字符串 S，计算不同子串的个数。

对字符串 S 构造后缀自动机。

每个 S 的子串都相当于自动机中的一些路径。因此不同子串的个数等于自动机中以 t_0 为起点的不同路径的条数。

考虑到 SAM 为有向无环图，不同路径的条数可以通过动态规划计算。即令 d_v 为从状态 v 开始的路径数量（包括长度为零的路径），则我们有如下递推方程：

$$d_v = 1 + \sum_{w:(v,w,c)\in DAWG} d_w$$

即，d_v 可以表示为所有 v 的转移的末端的和。

所以不同子串的个数为 $d_{t_0} - 1$（因为要去掉空子串）。

总时间复杂度为：$O(|S|)$。

另一种方法是利用上述后缀自动机的树形结构。每个节点对应的子串数量是 $\text{len}(i) - \text{len}(\text{link}(i))$，对自动机所有节点求和即可。

例题：【模板】后缀自动机，SDOI2016 生成魔咒

所有不同子串的总长度

> 给定一个字符串 S，计算所有不同子串的总长度。

本题做法与上一题类似，只是现在我们需要考虑分两部分进行动态规划：不同子串的数量 d_v 和它们的总长度 ans_v。

我们已经在上一题中介绍了如何计算 d_v。ans_v 的值可以通过以下递推式计算：

$$ans_v = \sum_{w:(v,w,c)\in DAWG} d_w + ans_w$$

我们取每个邻接结点 w 的答案，并加上 d_w（因为从状态 v 出发的子串都增加了一个字符）。

算法的时间复杂度仍然是 $O(|S|)$。

同样可以利用上述后缀自动机的树形结构。每个节点对应的所有后缀长度是 $\frac{\text{len}(i)\times(\text{len}(i)+1)}{2}$，减去其 link 节点的对应值就是该节点的净贡献，对自动机所有节点求和即可。

字典序第 k 大子串

> 给定一个字符串 S。多组询问，每组询问给定一个数 K_i，查询 S 的所有子串中字典序第 K_i 大的子串。

解决这个问题的思路可以从解决前两个问题的思路发展而来。字典序第 k 大的子串对应于 SAM 中字典序第 k 大的路径，因此在计算每个状态的路径数后，我们可以很容易地从 SAM 的根开始找到第 k 大的路径。

预处理的时间复杂度为 $O(|S|)$，单次查询的复杂度为 $O(|ans|\cdot|\Sigma|)$（其中 ans 是查询的答案，$|\Sigma|$ 为字符集的大小）。

> 虽然该题是后缀自动机的经典题，但实际上这题由于涉及字典序，用后缀数组做最方便。

例题：SPOJ - SUBLEX，TJOI2015 弦论

最小循环移位

> 给定一个字符串 S。找出字典序最小的循环移位。

容易发现字符串 $S+S$ 包含字符串 S 的所有循环移位作为子串。

所以问题简化为在 $S+S$ 对应的后缀自动机上寻找最小的长度为 $|S|$ 的路径，这可以通过平凡的方法做到：我们从初始状态开始，贪心地访问最小的字符即可。

总的时间复杂度为 $O(|S|)$。

出现次数

> 对于一个给定的文本串 T，有多组询问，每组询问给一个模式串 P，回答模式串 P 在字符串 T 中作为子串出现了多少次。

利用后缀自动机的树形结构，进行 dfs 即可预处理每个节点的终点集合大小。在自动机上查找模式串 P 对应的节点，如果存在，则答案就是该节点的终点集合大小；如果不存在，则答案为 0.

以下为原方法：

> 对文本串 T 构造后缀自动机。
>
> 接下来做预处理：对于自动机中的每个状态 v，预处理 cnt_v，使之等于 $\text{endpos}(v)$ 集合的大小。事实上，对应同一状态 v 的所有子串在文本串 T 中的出现次数相同，这相当于集合 endpos 中的位置数。
>
> 然而我们不能明确的构造集合 endpos，因此我们只考虑它们的大小 cnt。
>
> 为了计算这些值，我们进行以下操作。对于每个状态，如果它不是通过复制创建的（且它不是初始状态 t_0），我们将它的 cnt 初始化为 1。然后我们按它们的长度 len 降序遍历所有状态，并将当前的 cnt_v 的值加到后缀链接指向的状态上，即：
>
> $$cnt_{\text{link}(v)} += cnt_v$$
>
> 这样做每个状态的答案都是正确的。
>
> 为什么这是正确的？不是通过复制获得的状态，恰好有 $|T|$ 个，并且它们中的前 i 个在我们插入前 i 个字符时产生。因此对于每个这样的状态，我们在它被处理时计算它们所对应的位置的数量。因此我们初始将这些状态的 cnt 的值赋为 1，其它状态的 cnt 值赋为 0。
>
> 接下来我们对每一个 v 执行以下操作：$cnt_{\text{link}(v)} += cnt_v$。其背后的含义是，如果有一个字符串 v 出现了 cnt_v 次，那么它的所有后缀也在完全相同的地方结束，即也出现了 cnt_v 次。
>
> 为什么我们在这个过程中不会重复计数（即把某些位置数了两次）呢？因为我们只将一个状态的位置添加到 **一个** 其它的状态上，所以一个状态不可能以两种不同的方式将其位置重复地指向另一个状态。
>
> 因此，我们可以在 $O(|T|)$ 的时间内计算出所有状态的 cnt 的值。
>
> 最后回答询问只需要查找值 cnt_t，其中 t 为模式串对应的状态，如果该模式串不存在答案就为 0。单次查询的时间复杂度为 $O(|P|)$。

第一次出现的位置

> 给定一个文本串 T，多组查询。每次查询字符串 P 在字符串 T 中第一次出现的位置（P 的开头位置）。

我们构造一个后缀自动机。我们对 SAM 中的所有状态预处理位置 firstpos。即，对每个状态 v 我们想要找到第一次出现这个状态的末端的位置 $\text{firstpos}[v]$。换句话说，我们希望先找到每个集合 endpos 中的最小的元素（显然我们不能显式地维护所有 endpos 集合）。

为了维护 firstpos 这些位置，我们将原函数扩展为 `sam_extend()`。当我们创建新状态 cur 时，我们令：

$$\text{firstpos(cur)} = \text{len(cur)} - 1$$

；当我们将结点 q 复制到 clone 时，我们令：

$$\text{firstpos(clone)} = \text{firstpos}(q)$$

（因为值的唯一的其它选项 firstpos(cur) 显然太大了）。

那么查询的答案就是 $\text{firstpos}(t) - |P| + 1$，其中 t 为对应字符串 P 的状态。单次查询只需要 $O(|P|)$ 的时间。

所有出现的位置

> 问题同上，这一次需要查询文本串 T 中模式串出现的所有位置。

利用后缀自动机的树形结构，遍历子树，一旦发现终点节点就输出。

以下为原解法：

> 我们还是对文本串 T 构造后缀自动机。与上一个问题相似，我们为所有状态计算位置 firstpos。
>
> 如果 t 为对应于模式串 T 的状态，显然 $\text{firstpos}(t)$ 为答案的一部分。需要查找的其它位置怎么办？我们使用了含有字符串 P 的自动机，我们还需要将哪些状态纳入自动机呢？所有对应于以 P 为后缀的字符串的状态。换句话说我们要找到所有可以通过后缀链接到达状态 t 的状态。
>
> 因此为了解决这个问题，我们需要为每一个状态保存一个指向它的后缀引用列表。查询的答案就包含了对于每个我们能从状态 t 只使用后缀引用进行 DFS 或 BFS 的所有状态的 firstpos 值。
>
> 这种变通方案的时间复杂度为 $O(\text{answer}(P))$，因为我们不会重复访问一个状态（因为对于仅有一个后缀链接指向一个状态，所以不存在两条不同的路径指向同一状态）。
>
> 我们只需要考虑两个可能有相同 endpos 值的不同状态。如果一个状态是由另一个复制而来的，则这种情况会发生。然而，这并不会对复杂度分析造成影响，因为每个状态至多被复制一次。
>
> 此外，如果我们不从被复制的节点输出位置，我们也可以去除重复的位置。事实上对于一个状态，如果经过被复制状态可以到达，则经过原状态也可以到达。因此，如果我们给每个状态记录标记 `is_clone` 来代表这个状态是不是被复制出来的，我们就可以简单地忽略掉被复制的状态，只输出其它所有状态的 $firstpos$ 的值。
>
> 以下是大致的实现：

```
struct state {
bool is_clone;
int first_pos;
std::vector<int> inv_link;
// some other variables
};

// 在构造 SAM 后
for (int v = 1; v < sz; v++) st[st[v].link].inv_link.pus
h_back(v);

// 输出所有出现位置
void output_all_occurrences(int v, int P_length) {
if (!st[v].is_clone) cout << st[v].first_pos - P_length
+ 1 << endl;
for (int u : st[v].inv_link) output_all_occurrences(u, P
_length);
}
```

最短的没有出现的字符串

> 给定一个字符串 S 和一个特定的字符集，我们要找一个长度最短的没有在 S 中出现过的字符串。

我们在字符串 S 的后缀自动机上做动态规划。

令 d_v 为节点 v 的答案，即，我们已经处理完了子串的一部分，当前在状态 v，想找到不连续的转移需要添加的最小字符数量。计算 d_v 非常简单。如果不存在使用字符集中至少一个字符的转移，则 $d_v = 1$。否则添加一个字符是不够的，我们需要求出所有转移中的最小值：

$$d_v = 1 + \min_{w:(v,w,c)\in SAM} d_w$$

问题的答案就是 d_{t_0}，字符串可以通过计算过的数组 d 逆推回去。

两个字符串的最长公共子串

> 给定两个字符串 S 和 T，求出最长公共子串，公共子串定义为在 S 和 T 中都作为子串出现过的字符串 X。

我们对字符串 S 构造后缀自动机。

我们现在处理字符串 T，对于每一个前缀，都在 S 中寻找这个前缀的最长后缀。换句话说，对于每个字符串 T 中的位置，我们想要找

到这个位置结束的 S 和 T 的最长公共子串的长度。显然问题的答案就是所有 l 的最大值。

为了达到这一目的，我们使用两个变量，**当前状态** v 和 **当前长度** l。这两个变量描述当前匹配的部分：它的长度和它们对应的状态。

一开始 $v = t_0$ 且 $l = 0$，即，匹配为空串。

现在我们来描述如何添加一个字符 T_i 并为其重新计算答案：

- 如果存在一个从 v 到字符 T_i 的转移，我们只需要转移并让 l 自增一。
- 如果不存在这样的转移，我们需要缩短当前匹配的部分，这意味着我们需要按照后缀链接进行转移：

$$v = \text{link}(v)$$

与此同时，需要缩短当前长度。显然我们需要将 l 赋值为 $\text{len}(v)$，因为经过这个后缀链接后我们到达的状态所对应的最长字符串是一个子串。

- 如果仍然没有使用这一字符的转移，我们继续重复经过后缀链接并减小 l，直到我们找到一个转移或到达虚拟状态 -1（这意味着字符 T_i 根本没有在 S 中出现过，所以我们设置 $v = l = 0$）。

这一部分的时间复杂度为 $O(|T|)$，因为每次移动我们要么可以使 l 增加一，要么可以在后缀链接间移动几次，每次都减小 l 的值。

代码实现：

```
string lcs(const string &S, const string &T) {
  sam_init();
  for (int i = 0; i < S.size(); i++) sam_extend(S[i]);

  int v = 0, l = 0, best = 0, bestpos = 0;
  for (int i = 0; i < T.size(); i++) {
    while (v && !st[v].next.count(T[i])) {
      v = st[v].link;
      l = st[v].length;
    }
    if (st[v].next.count(T[i])) {
      v = st[v].next[T[i]];
      l++;
    }
    if (l > best) {
      best = l;
      bestpos = i;
    }
  }
  return t.substr(bestpos - best + 1, best);
}
```

例题：SPOJ Longest Common Substring

多个字符串间的最长公共子串

> 给定 k 个字符串 S_i。我们需要找到它们的最长公共子串，即作为子串出现在每个字符串中的字符串 X。

我们将所有的子串连接成一个较长的字符串 T，以特殊字符 D_i 分开每个字符串（一个字符对应一个字符串）：

$$T = S_1 + D_1 + S_2 + D_2 + \cdots + S_k + D_k.$$

然后对字符串 T 构造后缀自动机。

现在我们需要在自动机中找到存在于所有字符串 S_i 中的一个字符串，这可以通过使用添加的特殊字符完成。注意如果 S_j 包含了一个子串，则 SAM 中存在一条从包含字符 D_j 的子串而不包含以其它字符 D_1, ..., D_{j-1}, D_{j+1}, ..., D_k 开始的路径。

因此我们需要计算可达性，即对于自动机中的每个状态和每个字符 D_i，是否存在这样的一条路径。这可以容易地通过 DFS 或 BFS 及动态规划计算。之后，问题的答案就是状态 v 的字符串 $\text{longest}(v)$ 中存在所有特殊字符的路径。

广义后缀自动机（多个串中不同子串个数）//理解什么的再说

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 2000000;  // 双倍字符串长度
const int CHAR_NUM = 30;   // 字符集个数，注意修改下方的(-'a')
struct exSAM {
  int len[MAXN];              // 节点长度
  int link[MAXN];             // 后缀链接，link
  int next[MAXN][CHAR_NUM];   // 转移
  int tot;                    // 节点总数：[0, tot)
  void init() {               //初始化函数
    tot = 1;
    link[0] = -1;
  }
  int insertSAM(int last, int c) {  // last 为父 c 为子
    int cur = next[last][c];
    if (len[cur]) return cur;
    len[cur] = len[last] + 1;
    int p = link[last];
    while (p != -1) {
      if (!next[p][c])
        next[p][c] = cur;
      else
        break;
      p = link[p];
    }
    if (p == -1) {
      link[cur] = 0;
      return cur;
    }
    int q = next[p][c];
    if (len[p] + 1 == len[q]) {
      link[cur] = q;
      return cur;
    }
    int clone = tot++;
    for (int i = 0; i < CHAR_NUM; ++i)
      next[clone][i] = len[next[q][i]] != 0 ? next[q][i] : 0;
    len[clone] = len[p] + 1;
    while (p != -1 && next[p][c] == q) {
      next[p][c] = clone;
      p = link[p];
    }
    link[clone] = link[q];
    link[cur] = clone;
    link[q] = clone;
    return cur;
  }
  int insertTrie(int cur, int c) {
    if (next[cur][c]) return next[cur][c];  //已有该节点直接返回
    return next[cur][c] = tot++;            //无该节点 建立节点
  }
  void insert(const string &s) {
    int root = 0;
    for (auto ch : s) root = insertTrie(root, ch - 'a');
  }
  void insert(const char *s, int n) {
    int root = 0;
    for (int i = 0; i < n; ++i)
      root =
          insertTrie(root, s[i] - 'a');  //一边插入一边更改所插入新节点的父节点
  }
  void build() {
    queue<pair<int, int>> q;
    for (int i = 0; i < 26; ++i)
      if (next[0][i]) q.push({i, 0});
    while (!q.empty()) {  //广搜遍历
      auto item = q.front();
      q.pop();
      auto last = insertSAM(item.second, item.first);
      for (int i = 0; i < 26; ++i)
        if (next[last][i]) q.push({i, last});
    }
  }
} exSam;
char s[1000100];
int main() {
  int n;
  cin >> n;
  exSam.init();
  for (int i = 0; i < n; ++i) {
```

```
    cin >> s;
    int len = strlen(s);
    exSam.insert(s, len);//有string 版本的多态
  }
  exSam.build();
  long long ans = 0;
  for (int i = 1; i < exSam.tot; ++i) {
    ans += exSam.len[i] - exSam.len[exSam.link[i]];
  }
  cout << ans << endl;
}
```

图论

最小生成树

Boruvka

求最小森林 $O(E\log V)$

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

const int MaxN = 5000 + 5, MaxM = 200000 + 5;

int N, M;
int U[MaxM], V[MaxM], W[MaxM];
bool used[MaxM];
int par[MaxN], Best[MaxN];

void init() {
    //scanf("%d %d", &N, &M);
    cin>>N>>M;
    for (int i = 1; i <= M; ++i)
        cin>>U[i]>>V[i]>>W[i];
        //scanf("%d %d %d", &U[i], &V[i], &W[i]);
}

void init_dsu() {
    for (int i = 1; i <= N; ++i)
        par[i] = i;
}

int get_par(int x) {
    if (x == par[x]) return x;
    else return par[x] = get_par(par[x]);
}

// 比较统一连通块的出边边权
inline bool Better(int x, int y) {
    if (y == 0) return true;
    if (W[x] != W[y]) return W[x] < W[y];
    return x < y;
}

void Boruvka() {
    init_dsu();
    int merged = 0, sum = 0;

    bool update = true;
    while (update) {
        update = false;
        memset(Best, 0, sizeof Best);

        for (int i = 1; i <= M; ++i) {
            if (used[i] == true) continue;
            int p = get_par(U[i]), q = get_par(V[i]);
            if (p == q) continue;

            if (Better(i, Best[p]) == true) Best[p] = i;
            if (Better(i, Best[q]) == true) Best[q] = i;
        }

        for (int i = 1; i <= N; ++i)
            if (Best[i] != 0 && used[Best[i]] == false)
{
                update = true;
                merged++; sum += W[Best[i]];
                used[Best[i]] = true;
                // 合并连通块
                par[get_par(U[Best[i]])] = get_par(V[Bes
t[i]]);
            }
    }

    if (merged == N - 1)  //printf("%d\n", sum);
        cout<<sum<<"\n";
    else cout<<"orz\n";
}

int main() {
    ios::sync_with_stdio(false);
    init();
    Boruvka();
    return 0;
}
```

二分图

二分图带权最大匹配-KM

```
template <typename T>
struct hungarian {  // km
  int n;
  vector<int> matchx;  // 左集合对应的匹配点
  vector<int> matchy;  // 右集合对应的匹配点
  vector<int> pre;     // 连接右集合的左点
  vector<bool> visx;   // 拜访数组 左
  vector<bool> visy;   // 拜访数组 右
  vector<T> lx;
  vector<T> ly;
  vector<vector<T> > g;
  vector<T> slack;
  T inf;
  T res;
  queue<int> q;
  int org_n;
  int org_m;

  hungarian(int _n, int _m) {
    org_n = _n;
    org_m = _m;
    n = max(_n, _m);
    inf = numeric_limits<T>::max();
    res = 0;
    g = vector<vector<T> >(n, vector<T>(n));
    matchx = vector<int>(n, -1);
    matchy = vector<int>(n, -1);
    pre = vector<int>(n);
    visx = vector<bool>(n);
    visy = vector<bool>(n);
    lx = vector<T>(n, -inf);
    ly = vector<T>(n);
    slack = vector<T>(n);
  }

  void addEdge(int u, int v, int w) {
    g[u][v] = max(w, 0);  // 负值还不如不匹配 因此设为0 不
影响
    // 最小权匹配改为
        // g[u][v]=w;
  }

  bool check(int v) {
    visy[v] = true;
    if (matchy[v] != -1) {
      q.push(matchy[v]);
      visx[matchy[v]] = true;  // in S
      return false;
    }
    // 找到新的未匹配点 更新匹配点 pre 数组记录着"非匹配边"
上与之相连的点
    while (v != -1) {
      matchy[v] = pre[v];
      swap(v, matchx[pre[
    }
    return true;
  }

  void bfs(int i) {
    while (!q.empty()) {
      q.pop();
    }
    q.push(i);
    visx[i] = true;
    while (true) {
      while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int v = 0; v < n; v++) {
```

```
        if (!visy[v]) {
          T delta = lx[u] + ly[v] - g[u][v];
          if (slack[v] >= delta) {
            pre[v] = u;
            if (delta) {
              slack[v] = delta;
            } else if (check(v)) {  // delta=0 代表有机
会加入相等子图 找增广路
                                    // 找到就return 重
建交错树
              return;
            }
          }
        }
      }
      // 没有增广路 修改顶标
      T a = inf;
      for (int j = 0; j < n; j++) {
        if (!visy[j]) {
          a = min(a, slack[j]);
        }
      }
      for (int j = 0; j < n; j++) {
        if (visx[j]) {  // S
          lx[j] -= a;
        }
        if (visy[j]) {  // T
          ly[j] += a;
        } else {  // T'
          slack[j] -= a;
        }
      }
      for (int j = 0; j < n; j++) {
        if (!visy[j] && slack[j] == 0 && check(j)) {
          return;
        }
      }
    }
  }


  // 入口
  void solve() {
    // 初始顶标
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        lx[i] = max(lx[i], g[i][j]);
      }
    }

    for (int i = 0; i < n; i++) {
      fill(slack.begin(), slack.end(), inf);
      fill(visx.begin(), visx.end(), false);
      fill(visy.begin(), visy.end(), false);
      bfs(i);
    }
```

```
    // custom
    for (int i = 0; i < n; i++) {
      if (g[i][matchx[i]] > 0) {
        res += g[i][matchx[i]];
      } else {
        matchx[i] = -1;
      }
    }
        // 最小权
        /*
        // g 的初始化要改为-inf
        for(int i=0;i<n;++i){
                if(g[i][matchx[i]]!=-inf)
                        res+=g[i][matchx[i]];
                else
                        matchx[i]=-1;
        }
        */
    cout << res << "\n";
    for (int i = 0; i < org_n; i++) {
      cout << matchx[i] + 1 << " ";
    }
    cout << "\n";
  }
};
```

点分治

```
#include <cstdio>
#include <algorithm>
#include <vector>
#include <cstring>
using namespace std;
const int N=1e5+5;
struct node {
    int v, l;
};
vector<node> g[N];
int n, k, Size, s[N], f[N], root, d[N], K, ans;
vector<int> dep;
bool done[N];
void getroot(int now, int fa) {
    int u;
    s[now] = 1; f[now] = 0;
    for (int i=0; i<g[now].size(); i++)
        if ((u = g[now][i].v) != fa && !done[u]) {
            getroot(u, now);
            s[now] += s[u];
            f[now] = max(f[now], s[u]);
        }
    f[now] = max(f[now], Size-s[now]);
    if (f[now] < f[root]) root = now;
}
void getdep(int now, int fa) {
    int u;
    dep.push_back(d[now]);
    s[now] = 1;
    for (int i=0; i<g[now].size(); i++)
        if ((u = g[now][i].v) != fa && !done[u]) {
            d[u] = d[now] + g[now][i].l;
            getdep(u, now);
            s[now] += s[u];
        }
}
int calc(int now, int init) {
    dep.clear(); d[now] = init;
    getdep(now, 0);
    sort(dep.begin(), dep.end());
    int ret = 0;
    for (int l=0, r=dep.size()-1; l<r; )
        if (dep[l] + dep[r] <= K) ret += r-l++;
        else r--;
    return ret;
}
void work(int now) {
    int u;
    ans += calc(now, 0);
    done[now] = true;
    for (int i=0; i<g[now].size(); i++)
        if (!done[u = g[now][i].v]) {
            ans -= calc(u, g[now][i].l);
            f[0] = Size = s[u];
            getroot(u, root=0);
            work(root);
        }
}
signed main() {

    while (scanf("%d%d", &n, &K)) {
        if (n == 0 && K == 0) break;
        for (int i=0; i<=n; i++) g[i].clear();
        memset(done, false, sizeof(done));
        int u, v, l;
        for (int i=1; i<n; i++) {
            scanf("%d%d%d", &u, &v, &l);
            g[u].push_back(node(v, l));
            g[v].push_back(node(u, l));
        }
        f[0] = Size = n;
        getroot(1, root=0);
        ans = 0;
        work(root);
        printf("%d\n", ans);
    }
    return 0;
}
```

Floyd

```
for (k = 1; k <= n; k++) {
        for (i = 1; i <= n; i++) {
                for (j = 1; j <= n; j++) {
                        f[i][j] = min(f[i][j], f[i][k]
 + f[k][j]);
                }
```

```
        }
}
```

Tarjan

强连通分量

```
const int MAXN = 1e5 + 10;
struct Edge{
        int to, next, dis;
}edge[MAXN << 1];
int head[MAXN], cnt, ans;
bool inStack[MAXN];        //判断是否在栈中
//dfn 第一次访问到该节点的时间（时间戳）
//low[i] low[i]能从哪个点（最早时间戳）到达这个点的。
int dfn[MAXN], low[MAXN], tot;
stack<int> stc;
void add_edge(int u, int v, int dis) {
        edge[++cnt].to = v;
        edge[cnt].next = head[u];
        head[u] = cnt;
}
void Tarjan(int x) {
        dfn[x] = low[x] = ++tot;
        stc.push(x);
        inStack[x] = 1;
        for(int i = head[x]; i; i = edge[i].next) {
                int to = edge[i].to;
                if ( !dfn[to] ) {
                        Tarjan(to);
                        low[x] = min(low[x], low[to]);
                } else if (inStack[to]){
                        low[x] = min(low[x], dfn[to]);
                }
        }
        //cout << x << " " << low[x] << " " << dfn[x] <
< endl;
        if(low[x] == dfn[x]) {    //发现是整个强连通分量
子树里 的最小根。
                //int cnt = 0;
                ans++;   //强连通分量计数器
                while(1) {
                        int top = stc.top();
                        stc.pop();
                        //cnt ++;
                        inStack[top] = 0;
                        //cout << top << " "; 每个强连
通分量内的点
                        if(top == x) break;
                }
        }
}
void init() {
        cnt = 1;
        tot = 0;
        ans = 0;
        memset(inStack, 0, sizeof(inStack));
        memset(head, 0, sizeof(head));
        memset(dfn, 0, sizeof(dfn));
        memset(low, 0, sizeof(low));
        while(!stc.empty()) stc.pop();
}
int main () {
        std::ios::sync_with_stdio(false);
        cin.tie(0);
        int n, m;
        while(cin >> n >> m && (n || m)){
                init();
                int x, y;
                for(int i = 1; i <= m; ++i) {
                        cin >> x >> y;
                        add_edge(x, y, 0);        //有
向图求强连通
                }
                for(int i = 1; i <= n; ++i) {
                        if( !dfn[i] )
                                Tarjan(i);
                }
        }
        return 0;
}
```

缩点

```
const int MAXN = 5e3 + 20;
const int MAXM = 1e6 + 10;
int head[MAXN], cnt, tot, dfn[MAXN], low[MAXN], color[MA
XN], col;
bool vis[MAXN];
int degree[MAXN];
stack<int> stc;
int n, m;
struct Edge {
        int to, next, dis;
}edge[MAXM << 1];
void add_edge(int u, int v, int dis) {
        edge[++cnt].to = v;
        edge[cnt].next = head[u];
        head[u] = cnt;
}
void Tarjan(int x) {
        vis[x] = 1;
        dfn[x] = low[x] = ++tot;
        stc.push(x);
        for(int i = head[x]; i; i = edge[i].next) {
                int to = edge[i].to;
                if (!dfn[to]) {
                        Tarjan(to);
                        low[x] = min(low[x], low[to]);
                } else if( vis[to] ) {
                        low[x] = min(low[x], dfn[to]);
                }
        }
        if(dfn[x] == low[x]) {
                col ++;
                while(true) {
                        int top = stc.top();
                        stc.pop();
                        color[top] = col;//颜色相同的
点缩点
                        vis[top] = 0;
                //      cout << top << " ";
                        if(top == x) break;
                }
                //cout << endl;
        }
}
void solve(){
        for(int i = 1; i <= n; ++i) {
                if(!dfn[i])
                        Tarjan(i);
        }

        for(int x = 1; x <= n; ++x) {      //遍历 n 个节
点
                for(int i = head[x]; i; i = edge[i].ne
xt) {    //缩点后  每个点的出度
                        int to = edge[i].to;
                        if(color[x] != color[to]) {
                                degree[color[x]] ++;
                        }
                }
        }
}
void init () {
        cnt = 1;
        tot = 0;
        col = 0;
        memset(vis, 0, sizeof(vis));
        memset(head, 0, sizeof(head));
        memset(dfn, 0, sizeof(dfn));
        memset(low, 0, sizeof(low));
        memset(degree, 0, sizeof(degree));
        memset(color, 0, sizeof(color));
        while(!stc.empty()) stc.pop();
}
int main () {
        std::ios::sync_with_stdio(false);
        cin.tie(0);
        while(cin >> n && n) {
                cin >> m;
                init();
                int x, y;
                for(int i = 1; i <= m; ++i) {
                        cin >> x >> y;
                        add_edge(x, y, 0);
                }
                solve();
        }
        return 0;
}
```

割点

```
#include <bits/stdc++.h>
using namespace std;
int n, m; // n：点数 m：边数
int num[100001], low[100001], inde, res;
// num：记录每个点的时间戳
// low：能不经过父亲到达最小的编号，inde：时间戳，res：答案数量
bool vis[100001], flag[100001]; // flag：答案 vis：标记是否重复
vector<int> edge[100001]; // 存图用的
void Tarjan(int u, int father) { // u 当前点的编号，father 自己爸爸的编号
        vis[u] = true; // 标记
        low[u] = num[u] = ++inde; // 打上时间戳
        int child = 0; // 每一个点儿子数量
        for (auto v : edge[u]) { // 访问这个点的所有邻居（C++11）
                if (!vis[v]) {
                        child++; // 多了一个儿子
                        Tarjan(v, u); // 继续
                        low[u] = min(low[u], low[v]);
// 更新能到的最小节点编号
                        if (father != u && low[v] >= num[u] &&!flag[u]) // 主要代码
                                // 如果不是自己，且不通过父亲返回的最小点符合割点的要求，并且没有被标记过
                                // 要求即为：删了父亲连不上去了，即为最多连到父亲
                        {
                                flag[u] = true;
                                res++; // 记录答案
                        }
                }
                else if (v != father)
                        low[u] =
                        min(low[u], num[v]); // 如果这个点不是自己，更新能到的最小节点编号
        }
        if (father == u && child >= 2 &&
                !flag[u]) { // 主要代码，自己的话需要 2 个儿子才可以
                flag[u] = true;
                res++; // 记录答案
        }
}
int main() {
        cin >> n >> m; // 读入数据
        for (int i = 1; i <= m; i++) { // 注意点是从 1 开始的
                int x, y;
                cin >> x >> y;
                edge[x].push_back(y);
                edge[y].push_back(x);
        } // 使用 vector 存图
        for (int i = 1; i <= n; i++) // 因为 Tarjan 图不一定连通
                if (!vis[i]) {
                        inde = 0; // 时间戳初始为 0
                        Tarjan(i, i); // 从第 i 个点开始，父亲为自己
                }
        cout << res << endl;
        for (int i = 1; i <= n; i++)
                if (flag[i]) cout << i << " "; // 输出结果
        return 0;
}
```

点双连通分量

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll mod=998244353;
const int maxn=1e6+50;
const ll inf=0x3f3f3f3f3f3f3f3fLL;

struct Edge{
    int u,v;
};
///割顶 bccno 无意义
int pre[maxn],iscut[maxn],bccno[maxn],dfs_clock,bcc_cut;
vector<int>G[maxn],bcc[maxn];
stack<Edge>S;
int dfs(int u,int fa){
    int lowu = pre[u] = ++dfs_clock;
    int child = 0;
    for(int i = 0; i < G[u].size(); i++){
        int v =G[u][i];
        Edge e = (Edge){u,v};
        if(!pre[v]){ ///没有访问过
            S.push(e);
            child++;
            int lowv = dfs(v, u);
            lowu=min(lowu, lowv);                    ///用后代更新
            if(lowv >= pre[u]){
                iscut[u]=true;
                bcc_cut++;bcc[bcc_cut].clear();    ///注意 bcc 从 1 开始
                for(;;){
                    Edge x=S.top();S.pop();
                    if(bccno[x.u] != bcc_cut){bcc[bcc_cut].push_back(x.u);bccno[x.u]=bcc_cut;}
                    if(bccno[x.v] != bcc_cut){bcc[bcc_cut].push_back(x.v);bccno[x.v]=bcc_cut;}
                    if(x.u==u&&x.v==v)break;
                }
            }
        }
        else if(pre[v] < pre[u] && v !=fa){
            S.push(e);
            lowu = min(lowu,pre[v]);
        }
    }
    if(fa < 0 && child == 1) iscut[u] = 0;
    return lowu;
}
void find_bcc(int n){
    memset(pre, 0, sizeof(pre));
    memset(iscut, 0, sizeof(iscut));
    memset(bccno, 0, sizeof(bccno));
    dfs_clock = bcc_cut = 0;
    for(int i = 0; i < n;i++)
        if(!pre[i])dfs(i,-1);
}
```

边双连通分量

去除所有桥 dfs 即可

桥：

```
int low[MAXN], dfn[MAXN], iscut[MAXN], dfs_clock;
bool isbridge[MAXN];
vector<int> G[MAXN];
int cnt_bridge;
int father[MAXN];
void tarjan(int u, int fa) {
        father[u] = fa;
        low[u] = dfn[u] = ++dfs_clock;
        for (int i = 0; i < G[u].size(); i++) {
                int v = G[u][i];
                if (!dfn[v]) {
                        tarjan(v, u);
                        low[u] = min(low[u], low[v]);
                        if (low[v] > dfn[u]) {//主要
                                isbridge[v] = true;
                                ++cnt_bridge;
                        }
                }
                else if (dfn[v] < dfn[u] && v != fa) {
                        low[u] = min(low[u], dfn[v]);
                }
        }
}
```

2-SAT

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int maxn=1e6+5;
int a[maxn<<1];
vector<int>g[maxn<<1];
int tot;
int dfn[maxn<<1],low[maxn<<1];
stack<int>sta;
int insta[maxn<<1];
int scccnt;
```

```
int color[maxn<<1];
int n,m;
void tarjan(int u)
{
    dfn[u]=low[u]=++tot;
    sta.push(u);
    insta[u]=1;
    for(auto v:g[u])
    {
        if(!dfn[v])
        {
            tarjan(v);
            low[u]=min(low[u],low[v]);
        }
        else if(insta[v])
        {
            low[u]=min(low[u],dfn[v]);
        }
    }
    if(dfn[u]==low[u])
    {
        ++scccnt;
        do{
            color[u]=scccnt;
            u=sta.top();sta.pop();
            insta[u]=0;
        }while(low[u]!=dfn[u]);
    }

}
signed main()
{
    ios::sync_with_stdio(false);

    cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
        int a,va,b,vb;//a 为va 或者 b 为vb
        cin>>a>>va>>b>>vb;
        if(va&vb)
        {
            g[a+n].push_back(b);
            g[b+n].push_back(a);
        }
        else if(!va&vb)
        {
            g[a].push_back(b);
            g[b+n].push_back(a+n);
        }
        else if(va&!vb)
        {
            g[a+m].push_back(b+n);
            g[b].push_back(a);
        }
        else if(!va&!vb)
        {
            g[a].push_back(b+n);
            g[b].push_back(a+n);
        }
    }
    /*for (int i = 0; i < m; ++i) {
    int a = read(), va = read(), b = read(), vb = read();
    g[a + n * (va & 1)].push_back(b + n * (vb ^ 1));
    g[b + n * (vb & 1)].push_back(a + n * (va ^ 1));*/
    for(int i=1;i<=(n<<1);i++)
    {
        if(!dfn[i])
            tarjan(i);
    }
    for(int i=1;i<=n;i++)
    {
        if(color[i]==color[i+n])
        {
            cout<<"IMPOSSIBLE\n";
            return 0;
        }
    }
    cout<<"POSSIBLE\n";
    for(int i=1;i<=n;i++)
    {
        int tmp=(color[i]<color[i+n])?1:0;
        cout<<tmp<<" ";
    }
}
```

找环

```
void tarjan(int u) {
    low[u] = dfn[u] = ++dfsClock;
    stk.push(u); ins[u] = true;
    for (const auto &v : g[u]) {
        if (!dfn[v]) tarjan(v), low[u] = std::min(low[u],
 low[v]);
        else if (ins[v]) low[u] = std::min(low[u], dfn
[v]);
    }
    if (low[u] == dfn[u]) {
        ++sccCnt;
        do {
            color[u] = sccCnt;
            u = stk.top(); stk.pop(); ins[u] = false;
        } while (low[u] != dfn[u]);
    }
}
// 笔者使用了 Tarjan 找环，得到的 color[x] 是 x 所在的 scc
的拓扑逆序。
for (int i = 1; i <= (n << 1); ++i) if (!dfn[i]) tarjan
(i);
for (int i = 1; i <= n; ++i)
    if (color[i] == color[i + n]) { // x 与 -x 在同一强连
通分量内，一定无解
        puts("IMPOSSIBLE");
        exit(0);
    }
puts("POSSIBLE");
for (int i = 1; i <= n; ++i)
    print((color[i] < color[i + n])), putchar(' '); //
如果不使用 Tarjan 找环，请改成大于号
```

网络流

```
最大流-Dinic
template <class T>
struct Dinic
{
    struct Edge
    {
        int v, next;
        T flow;
        Edge() {}
        Edge(int v, int next, T flow) : v(v), next(next),
 flow(flow) {}
    } e[N * 30];
    int head[N], tot;
    int cur[N]; // 当前弧优化
    int dep[N];
    void init(int siz)
    {
        memset(head, -1, sizeof(head)*(siz+2));
        tot = 0;
    }
    void adde(int u, int v, T w, T rw = 0)
    {
        e[tot] = Edge(v, head[u], w);
        head[u] = tot++;
        cur[u]=head[u];
        e[tot] = Edge(u, head[v], rw);
        head[v] = tot++;
        cur[v]=head[v];
    }
    bool BFS(int _S, int _T)
    {
        memset(dep, 0, sizeof(dep));
        queue<int> q;
        q.push(_S);
        dep[_S] = 1;
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            for (int i = head[u]; ~i; i = e[i].next)
            {
                int v = e[i].v;
                if (!dep[v] && e[i].flow > 0)
                {
                    dep[v] = dep[u] + 1;
                    q.push(v);
                }
            }
        }
        return dep[_T] != 0;
    }
    T dfs(int _S, int _T, T a)
    {
```

```
        T flow = 0, f;
        if (_S == _T || a == 0)
            return a;
        for (int i = cur[_S]; ~i; i = e[i].next)
        {
            cur[_S]=i;
            int v = e[i].v;
            if (dep[v] != dep[_S] + 1)
                continue;
            f = dfs(v, _T, min(a, e[i].flow));
            if (f)
            {
                e[i].flow -= f;
                e[i ^ 1].flow += f;
                flow += f;
                a -= f;
                if (a == 0)
                    break;
            }
        }
        if (!flow)
            dep[_S] = -1;
        return flow;
    }
    T dinic(int _S, int _T)
    {
        T max_flow = 0;
        while (BFS(_S, _T))
            max_flow += dfs(_S, _T, INF);
        return max_flow;
    }
};
```

树形 DP 求树的最小支配集,最小点覆盖,最大独立集

一:最小支配集

考虑最小支配集,每个点有两种状态,即属于支配集合或者不属于支配集合,其中不属于支配集合时此点还需要被覆盖,被覆盖也有两种状态,即被子节点覆盖或者被父节点覆盖.总结起来就是三种状态,现对这三种状态定义如下:

1):**dp[i] [0]**,表示点 i 属于支配集合,并且以点 i 为根的子树都被覆盖了的情况下支配集中所包含最少点的个数.

2):**dp[i] [1]**,表示点 i 不属于支配集合,且以 i 为根的子树都被覆盖,且 i 被其中不少于一个子节点覆盖的情况下支配集所包含最少点的个数.

3):**dp[i] [2]**,表示点 i 不属于支配集合,且以 i 为根的子树都被覆盖,且 i 没被子节点覆盖的情况下支配集中所包含最少点的个数.即 i 将被父节点覆盖.

对于第一种状态,dp[i] [0]含义为点 i 属于支配集合,那么依次取每个儿子节点三种状态中的最小值,再把取得的最小值全部加起来再加 1,就是 dp[i] [0]的值了.即只要每个以 i 的儿子为根的子树都被覆盖,再加上当前点 i,所需要的最少的点的个数,DP 转移方程如下:

dp[i] [0] = 1 + ∑(u 取 i 的子节点)min(dp[u] [0], dp[u] [1], dp[u] [2])

对于第三种状态,dp[i] [2]含义为点 i 不属于支配集合,且 i 被其父节点覆盖.那么说明点 i 和点 i 的儿子节点都不属于支配集合,所以点 i 的第三种状态之和其儿子节点的第二种状态有关,方程为:

dp[i] [2] = ∑(u 取 i 的子节点)dp[u] [1]

对于第二种状态,略有些复杂.首先如果点 i 没有子节点那么 dp[i] [1]应该初始化为 INF;否则为了保证它的每个以 i 的儿子为根的子树被覆盖,那么要取每个儿子节点的前两种状态的最小值之和,因为此时点 i 不属于支配集,不能支配其子节点,所以子节点必须已经被支配,与子节点的第三种状态无关.如果当前所选状态中每个儿子都没被选择进入支配集,即在每个儿子的前两种状态中,第一种状态都不是所需点最小的,那么为了满足第二种状态的定义(因为点 i 的第二种状态必须被其子节点覆盖,即其子节点必须有一个属于支配集,如果此时没有,就必须强制使一个子节点的状态为状态一),需要重新选择点 i 的一个儿子节点为第一种状态,这时取花费最少的一个点,即取 min(dp[u] [0] - dp[u] [1])的儿子节点 u,强制取其第一种状态,其他的儿子节点取第二种状态,DP 转移方程为:

if(i 没有子节点) dp[i] [1] = INF

else dp[i] [1] = ∑(u 取 i 的子节点)min(dp[u] [0], dp[u] [1]) + inc

其中对于 inc 有:

if(上面式子中的 ∑(u 取 i 的子节点)min(dp[u] [0], dp[u] [1]) 中包含某个 dp[u] [0], 即存在一个所选的最小值为状态一的儿子节点) inc = 0

else inc = min(dp[u] [0] - dp[u] [1]) (其中 u 取点 i 的儿子节点)

代码:

```
1 void DP(int u, int p) {// p 为 u 的父节点
2     dp[u][2] = 0;
3     dp[u][0] = 1;
4     bool s = false;
5     int sum = 0, inc = INF;
6     for(int k = head[u]; k != -1; k = edge[k].next) {
7         int to = edge[k].to;
8         if(to == p) continue;
9         DP(to, u);
10        dp[u][0] += min(dp[to][0], min(dp[to][1], dp
[to][2]));
11        if(dp[to][0] <= dp[to][1]) {
12            sum += dp[to][0];
13            s = true;
14        }
15        else {
16            sum += dp[to][1];
17            inc = min(inc, dp[to][0] - dp[to][1]);
18        }
19        if(dp[to][1] != INF && dp[u][2] != INF) dp[u]
[2] += dp[to][1];
20        else dp[u][2] = INF;
21    }
22    if(inc == INF && !s) dp[u][1] = INF;
23    else {
24        dp[u][1] = sum;
25        if(!s) dp[u][1] += inc;
26    }
27 }
```

二:最小点覆盖

对于最小点覆盖,每个点只有两种状态,即属于点覆盖或者不属于点覆盖:

1):**dp[i] [0]**表示点 i 属于点覆盖,并且以点 i 为根的子树中所连接的边都被覆盖的情况下点覆盖集中所包含最少点的个数.

2):**dp[i] [1]**表示点 i 不属于点覆盖,且以点 i 为根的子树中所连接的边都被覆盖的情况下点覆盖集中所包含最少点的个数.

对于第一种状态 dp[i] [0],等于每个儿子节点的两种状态的最小值之和加 1,DP 转移方程如下:

dp[i] [0] = 1 + ∑(u 取 i 的子节点)min(dp[u] [0], dp[u] [1])

对于第二种状态 dp[i] [1],要求所有与 i 连接的边都被覆盖,但是点 i 不属于点覆盖,那么点 i 的所有子节点就必须属于点覆盖,即对于点 i 的第二种状态与所有子节点的第一种状态有关,在数值上等于所有子节点第一种状态的和.DP 转移方程如下:

dp[i] [1] = ∑(u 取 i 的子节点)dp[u] [0]

代码:

```
1 void DP(int u, int p) {// p 为 u 的父节点
2     dp[u][0] = 1;
3     dp[u][1] = 0;
4     for(int k = head[u]; k != -1; k = edge[k].next) {
5         int to = edge[k].to;
6         if(to == p) continue;
7         DP(to, u);
8         dp[u][0] += min(dp[to][0], dp[to][1]);
9         dp[u][1] += dp[to][0];
10    }
11 }
```

三:最大独立集

对于最大独立集,每个点也只有两种状态,即属于点 i 属于独立集或者不属于独立集两种情况:

1):**dp[i] [0]**表示点 i 属于独立集的情况下,最大独立集中点的个数.

2):**dp[i] [1]**表示点 i 不属于独立集的情况下.最大独立集中点的个数.

对于第一种状态 dp[i] [0],由于 i 点属于独立集,所以它的子节点都不能属于独立集,所以对于点 i 的第一种状态,只和它的子节点的第二种状态有关.等于其所有子节点的第二种状态的值加 1,DP 转移方程如下:

dp[i] [0] = 1 + ∑(u 取 i 的子节点) dp[u] [1]

对于第二种状态 dp[i] [1],由于点 i 不属于独立集,所以子节点可以属于独立解,也可以不属于独立集,所取得子节点状态应该为数值较大的那个状态,DP 转移方程:

dp[i] [1] = ∑(u 取 i 的子节点)max(dp[u] [0], dp[u] [1])

代码:

```
void DP(int u, int p) {// p 为 u 的父节点
    dp[u][0] = 1;
    dp[u][1] = 0;
    for(int k = head[u]; k != -1; k = edge[k].next) {
        int to = edge[k].to;
        if(to == p) continue;
        DP(to, u);
        dp[u][0] += dp[to][1];
        dp[u][1] += max(dp[to][0], dp[to][1]);
    }
}
```

数据结构

并查集

```
int parent[maxn],rk[maxn];
void init(int n)
{
    for(int i=0;i<n;i++)
    {
        parent[i]=i;
        rk[i]=0;   // 初始树的高度为0
    }
}
// 合并x 和y 所属的集合
int fid(int x)       //查找 x 元素所在的集合,回溯时压缩路径
{
    if (x != parent[x])
    {
        parent[x] = fid(parent[x]);      //回溯时的压缩路径
    }         //从 x 结点搜索到祖先结点所经过的结点都指向该祖先结点
    return parent[x];
}
void unite(int x,int y)
{
    x=fid(x);
    y=fid(y);
    if(x==y) return ;
    if(rk[x]<rk[y])
        parent[x]=y;  // 合并是从 rank 小的向 rank 大的连边
    else
    {
        parent[y]=x;
        if(rk[x]==rk[y]) rk[x]++;
    }
}
```

树状数组（区间查询区间修改）

```
#include<iostream>
#include<stdio.h>
#include<string.h>
#include<algorithm>
using namespace std;
const int maxn=1e5+10;
int sum1[maxn];
int sum2[maxn];
int a[maxn];
int n,m;
int lowbit(int x){
  return x&(-x);
}
void update(int x,int w){//更新效果：把 x 位置后面所有的数的值+w
   for (int i=x;i<=n;i+=lowbit(i)){
      sum1[i]+=w;//维护前缀和 c[i]
      sum2[i]+=w*(x-1);//维护前缀和 c[i]*(n-1)
   }
}
void range_update(int l,int r,int val)//更新效果：把 l 位置到 r 位置所有的数的值+w
{
    update(l,val);
    update(r+1,-val);
}
int sum(int x){//求 1-x 的和
  int ans=0;
  for (int i=x;i>0;i-=lowbit(i)){
    ans+=x*sum1[i]-sum2[i];
  }
  return ans;
}
int range_ask(int l,int r){//求 l-r 的和
   return sum(r)-sum(l-1);
}
int main(){
  while(~scanf("%d%d",&n,&m)){
    for (int i=1;i<=n;i++){
        scanf("%d",&a[i]);
        update(i,a[i]-a[i-1]);//维护差分数组
    }
   }
  return 0;
}
```

主席树（非工程版）

```
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const int maxn = 1e5;  // 数据范围
int tot, n, m;
int sum[(maxn << 5) + 10], rt[maxn + 10], ls[(maxn << 5) + 10],
    rs[(maxn << 5) + 10];
int a[maxn + 10], ind[maxn + 10], len;
inline int getid(const int &val) {  // 离散化
  return lower_bound(ind + 1, ind + len + 1, val) - ind;
}
int build(int l, int r) {  // 建树
  int root = ++tot;
  if (l == r) return root;
  int mid = l + r >> 1;
  ls[root] = build(l, mid);
  rs[root] = build(mid + 1, r);
  return root;  // 返回该子树的根节点
}
int update(int k, int l, int r, int root) {  // 插入操作
  int dir = ++tot;
  ls[dir] = ls[root], rs[dir] = rs[root], sum[dir] = sum[root] + 1;
  if (l == r) return dir;
  int mid = l + r >> 1;
  if (k <= mid)
    ls[dir] = update(k, l, mid, ls[dir]);
  else
    rs[dir] = update(k, mid + 1, r, rs[dir]);
  return dir;
}
int query(int u, int v, int l, int r, int k) {  // 查询操作
  int mid = l + r >> 1,
      x = sum[ls[v]] - sum[ls[u]];  // 通过区间减法得到左儿子的信息
  if (l == r) return l;
  if (k <= x)  // 说明在左儿子中
    return query(ls[u], ls[v], l, mid, k);
  else  // 说明在右儿子中
    return query(rs[u], rs[v], mid + 1, r, k - x);
}
inline void init() {
  scanf("%d%d", &n, &m);
  for (int i = 1; i <= n; ++i) scanf("%d", a + i);
  memcpy(ind, a, sizeof ind);
  sort(ind + 1, ind + n + 1);
```

```
  len = unique(ind + 1, ind + n + 1) - ind - 1;
  rt[0] = build(1, len);
  for (int i = 1; i <= n; ++i) rt[i] = update(getid(a
[i]), 1, len, rt[i - 1]);
}
int l, r, k;
inline void work() {
  while (m--) {
    scanf("%d%d%d", &l, &r, &k);
    printf("%d\n", ind[query(rt[l - 1], rt[r], 1, len,
k)]);  // 回答询问
  }
}
int main() {
  init();
  work();
  return 0;
}
```

线段树

- simple 线段树

```
/*
初始化可以传 vector
vector 从 1-n
区间修改 modify(l,r,k) 默认区间加
单点修改 modify(p,k)
区间查询 modify(l,r)
单点查询 modify(p)

*/


using i64=long long ;
#pragma GCC optimize(2)
#define lson rt<<1
#define rson rt<<1|1
template<class Info,
    class Merge = plus<Info>>
struct SegmentTree{
    const int n;
    const Merge merge;
    vector<Info> info;
    SegmentTree(int n) :n(n) ,merge(Merge()),info(4 << _
_lg(n)) {}
    SegmentTree(vector<Info> init) : SegmentTree(init.si
ze()-1) {
        function<void(int,int,int)> build = [&](int rt,i
nt l,int r) {
            if (r == l) {
                info[rt] = init[l];
                return;
            }
            int mid = (l + r) / 2;
            build(lson,l,mid);
            build(rson,mid+1,r);
            pull(rt);
        };
        build(1,1,n);
    }
    void pull(int rt) { info[rt] = merge(info[lson],info
[rson]); }
    void push(int rt)
    {
        pushDown(info[rt],info[lson]);
        pushDown(info[rt],info[rson]);
        cleanLazy(info[rt]);
    }
    void pushDown(Info &a,Info &b){
        if(a.lazy){
            b.s += a.lazy*(b.r-b.l+1);
            b.lazy += a.lazy;
        }
    }
    void cleanLazy(Info &a){ a.lazy=0; }
    void modify(int rt,int l,int r,int L,int R,const int
 &v) {
        if (L <= l && r <= R) {
            info[rt] = info[rt] + v;
            return;
        }
        push(rt);
        int mid = (l + r) / 2;
        if(L > mid)
            modify(rson,mid+1,r,L,R,v);
        else if (R <= mid)
            modify(lson, l,mid,L,R,v);
        else
            modify(lson,l,mid,L,mid,v),modify(rson,mid+1,
r,mid+1,R,v);
        pull(rt);
    }
    void modify(int l,int r,const int &v) { modify(1,1,n,
l,r,v); }
    void modify(int p,const int & v) {modify(1,1,n,p,p,
v);}
    Info rangeQuery(int rt,int l,int r,int L,int R){
        if(R < l || r < L )
            return Info(l,r,0);
        if(L <= l && r <= R)
            return info[rt];
        push(rt);
        int mid = (l + r) / 2;
        return merge(rangeQuery(lson,l,mid,L,R),rangeQue
ry(rson,mid+1,r,L,R));
    }
    Info rangeQuery(int l,int r)
    {
        return rangeQuery(1,1,n,l,r);
    }

};

struct Info {
    int l,r;
    i64 s,lazy;
    Info() : l(0),r(0),s(0),lazy(0) {}
    Info(int x,i64 val) : l(x),r(x),s(val),lazy(0) {}
    Info(i64 val) : l(0),r(0),s(val),lazy(0) {}
    Info(int L,int R,i64 val) : l(L),r(R),s(val),lazy(0)
 {}
    Info(int L,int R,i64 val,i64 lz) : l(L),r(R),s(val),
lazy(lz) {}
};

Info operator+ (const Info &a,const int& b) // Lazy 下标
的运算符重载
{
    return Info(a.l,a.r,a.s+b*(a.r-a.l+1),a.lazy+b);
}

Info operator+ (const Info &a,const Info &b) { // 区间合
并的运算符重载
    return Info(a.l,b.r,a.s+b.s);
}
```

树链剖分

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define pii pair<int,int>
using namespace std;
const int maxn=1e5+10;
struct Node{
    int sum,lazy,l,r,ls,rs;
}node[2*maxn];
int rt,n,m,r,p,a[maxn],cnt,f[maxn],d[maxn],siz[maxn],son
[maxn],rk[maxn],top[maxn],id[maxn];

vector<int>g[maxn];
int mod(int a,int b)
{
    return (a+b)%p;
}
inline void add_edge(int x,int y)
{
    g[x].push_back(y);
}
void dfs1(int u,int fa,int depth)
{
    f[u]=fa;
    d[u]=depth;
    siz[u]=1;
    for(auto &v:g[u])
    {
        if(v==fa)
            continue;
        dfs1(v,u,depth+1);
        siz[u]+=siz[v];
        if(siz[v]>siz[son[u]])
            son[u]=v;
    }
}
```

```
void dfs2(int u,int t)
{
    top[u]=t;
    id[u]=++cnt;
    rk[cnt]=u;
    if(!son[u])
        return;
    dfs2(son[u],t);
    for(auto &v:g[u])
    {
        if(v!=son[u]&&v!=f[u])
            dfs2(v,v);
    }
}
void pushup(int x)
{
    node[x].sum=(node[node[x].ls].sum+node[node[x].rs].s
um+node[x].lazy*(node[x].r-node[x].l+1))%p;
}
void build(int li,int ri,int cur)
{
    if(li==ri)
    {
        node[cur].l=node[cur].r=li;
        node[cur].sum=a[rk[li]];
        return;
    }
    int mid=(li+ri)>>1;
    node[cur].ls=cnt++;
    node[cur].rs=cnt++;
    build(li,mid,node[cur].ls);
    build(mid+1,ri,node[cur].rs);
    node[cur].l=node[node[cur].ls].l;
    node[cur].r=node[node[cur].rs].r;
    pushup(cur);
}
void update(int li,int ri,int c,int cur)
{
    if(li<=node[cur].l&&node[cur].r<=ri)
    {
        node[cur].sum=mod(node[cur].sum,c*(node[cur].r-n
ode[cur].l+1));
        node[cur].lazy=mod(node[cur].lazy,c);
        return;
    }
    int mid=(node[cur].l+node[cur].r)>>1;
    if(li<=mid)
        update(li,ri,c,node[cur].ls);
    if(mid<ri)
        update(li,ri,c,node[cur].rs);
    pushup(cur);
}
int query(int li,int ri,int cur)
{
    if(li<=node[cur].l&&node[cur].r<=ri)
        return node[cur].sum;
    int tot=node[cur].lazy*(min(node[cur].r,ri)-max(node
[cur].l,li)+1)%p;
    int mid=(node[cur].l+node[cur].r)>>1;
    if(li<=mid)
        tot=mod(tot,query(li,ri,node[cur].ls));
    if(mid<ri)
        tot=mod(tot,query(li,ri,node[cur].rs));
    return tot%p;
}
int sum(int x,int y)
{
    int ans=0;
    int fx=top[x],fy=top[y];
    while(fx!=fy)
    {
        if(d[fx]>=d[fy])
        {
            ans=mod(ans,query(id[fx],id[x],rt));
            x=f[fx],fx=top[x];
        }
        else
        {
            ans=mod(ans,query(id[fy],id[y],rt));
            y=f[fy],fy=top[y];
        }
    }
    if(id[x]<=id[y])
        ans=mod(ans,query(id[x],id[y],rt));
    else
        ans=mod(ans,query(id[y],id[x],rt));
    return ans%p;
}
void updates(int x,int y,int c)
{
    int fx=top[x],fy=top[y];
    while(fx!=fy)
    {
        if(d[fx]>=d[fy])
        {
            update(id[fx],id[x],c,rt);
            x=f[fx],fx=top[x];
        }
        else
        {
            update(id[fy],id[y],c,rt);
            y=f[fy],fy=top[y];
        }
    }
    if(id[x]<=id[y])
        update(id[x],id[y],c,rt);
    else
        update(id[y],id[x],c,rt);
}
signed main()
{
    ios::sync_with_stdio(false);
    cin>>n>>m>>r>>p;
    for(int i=1;i<=n;i++)
        cin>>a[i];
    for(int i=1;i<n;i++)
    {
        int x,y;
        cin>>x>>y;
        add_edge(x,y);
        add_edge(y,x);
    }
    cnt=0;
    dfs1(r,0,1);
    dfs2(r,r);
    cnt=0;
    rt=cnt++;
    build(1,n,rt);
    for(int i=1;i<=m;i++)
    {
        int op,x,y,z;
        cin>>op;
        if(op==1)
        {
            cin>>x>>y>>z;
            updates(x,y,z);
        }
        else if(op==2)
        {
            cin>>x>>y;
            cout<<sum(x,y)<<'\n';
        }
        else if(op==3)
        {
            cin>>x>>z;
            //子树也有连续区间的性质
            update(id[x],id[x]+siz[x]-1,z,rt);
        }
        else if(op==4)
        {
            cin>>x;
            cout<<query(id[x],id[x]+siz[x]-1,rt)<<'\n';
        }
    }
    return 0;
}
```

FHQ-Treap

FHQ-Treap-ptr

```
#include<random>
using namespace std;

uniform_int_distribution<unsigned> u(1, 100000000);
random_device rd;
mt19937 e(rd());
struct fhqTreap {
    struct Node {
        int val, rnd, siz, num;
        Node* rs, * ls;
        Node() {}
        Node(int x) :val(x), ls(nullptr), rs(nullptr), r
nd(u(e)), siz(1), num(1) {}
        Node(int x, int y) :val(0), ls(nullptr), rs(null
ptr), rnd(0), siz(0), num(0) {}
    };
```

```
    Node* nullnode = new Node(0, 0);
    Node* root = nullnode;
    int n;
    fhqTreap() :n(0) {};
    Node* newNode(int x) {
        Node* ret = new Node(x);
        ret->ls = nullnode;
        ret->rs = nullnode;
        return ret;
    }
    void pushup(Node* x) {
        if (x == nullnode) return;
        x->siz = x->ls->siz + x->rs->siz;
        x->siz += x->num;
    }
    void split(Node* rt, int k, Node** x, Node** y) {
        if ((rt) == nullnode) { *x = *y = nullnode; retu
rn; }
        if ((rt)->val <= k) *x = rt, split(rt->rs, k, &
(rt->rs), y);
        else *y = rt, split(rt->ls, k, x, &(rt->ls));
        pushup(rt);
    }
    Node* merge(Node* x, Node* y) {
        if (x == nullnode || y == nullnode) {
            return x == nullnode ? y : x;
        }
        if (x->rnd < y->rnd) {
            x->rs = merge(x->rs, y);
            pushup(x);
            return x;
        }
        else {
            y->ls = merge(x, y->ls);
            pushup(y);
            return y;
        }
    }
    void Delete(int k)
    {
        Node* x, * y, * z;
        split(root, k, &x, &y);
        split(x, k - 1, &x, &z);
        if (z->num == 1) {
            delete z;
            root = merge(x, y);
        }
        else {
            z->num--, z->siz--;
            root = merge(merge(x, z), y);
        }
    }
    void insert(int k)
    {
        Node* x, * y, * z;
        x = y = z = nullnode;
        split(root, k, &x, &y);
        split(x, k - 1, &x, &z);
        // z=new Node(k);
        if (z != nullnode) z->num++, z->siz++;
        else z = newNode(k);
        root = merge(merge(x, z), y);
    }
    Node* find_kth(Node* rt, int k)
    {
        while (1) {
            if (rt->ls->siz >= k) rt = rt->ls;
            else if (rt->ls->siz < k && k <= rt->ls->siz
 + rt->num) return rt;
            else k -= rt->ls->siz + rt->num, rt = rt->rs;
        }
    }
    Node* kth(int k)
    {
        return find_kth(root, k);
    }
    int kth_val(int k)
    {
        return kth(k)->val;
    }
    Node* max(Node* rt) {
        if (rt == nullnode) return rt;
        while (rt->rs != nullnode) {
            rt = rt->rs;
        }
        return rt;
    }
    Node* max() { return max(root); }
    Node* min(Node* rt) {
        if (rt == nullnode) return rt;
        while (rt->ls != nullnode) rt = rt->ls;
        return rt;
    }
    Node* min() { return min(root); }
    Node* pre(Node** rt, int k)
    {
        Node* x, * y;
        split(*rt, k - 1, &x, &y);
        Node* ret = max(x);
        *rt = merge(x, y);
        return ret;
    }
    Node* pre(int k) { return pre(&root, k); }
    Node* sub(Node** rt, int k)
    {
        Node* x, * y;
        split(*rt, k, &x, &y);
        Node* ret = min(y);
        *rt = merge(x, y);
        return ret;
    }
    Node* sub(int k) { return sub(&root, k); }
    int rank(Node** rt, int k)
    {
        Node* x, * y;
        split(*rt, k - 1, &x, &y);
        int ret = x->siz;
        *rt = merge(x, y);
        return ret + 1;
    }
    int rank(int k) { return rank(&root, k); }
};
```

区间翻转(可以部分替代 splay)

```
# include<iostream>
# include<cstdio>
# include<cstring>
# include<cstdlib>
using namespace std;
const int MAX=1e5+1;
int n,m,tot,rt;
struct Treap{
    int pos[MAX],siz[MAX],w[MAX];
    int son[MAX][2];
    bool fl[MAX];
    void pus(int x)
    {
        siz[x]=siz[son[x][0]]+siz[son[x][1]]+1;
    }
    int build(int x)
    {
        w[++tot]=x,siz[tot]=1,pos[tot]=rand();
        return tot;
    }
    void down(int x)
    {
        swap(son[x][0],son[x][1]);
        if(son[x][0]) fl[son[x][0]]^=1;
        if(son[x][1]) fl[son[x][1]]^=1;
        fl[x]=0;
    }
    int merge(int x,int y)
    {
        if(!x||!y) return x+y;
        if(pos[x]<pos[y])
        {
            if(fl[x]) down(x);
            son[x][1]=merge(son[x][1],y);
            pus(x);
            return x;
        }
        if(fl[y]) down(y);
        son[y][0]=merge(x,son[y][0]);
        pus(y);
        return y;
    }
    void split(int i,int k,int &x,int &y)
    {
        if(!i)
        {
            x=y=0;
            return;
        }
        if(fl[i]) down(i);
```

```
        if(siz[son[i][0]]<k)
        x=i,split(son[i][1],k-siz[son[i][0]]-1,son[i][1],
y);
        else
        y=i,split(son[i][0],k,x,son[i][0]);
        pus(i);
    }
    void coutt(int i)
    {
        if(!i) return;
        if(fl[i]) down(i);
        coutt(son[i][0]);
        printf("%d ",w[i]);
        coutt(son[i][1]);
    }
}Tree;
int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)
      rt=Tree.merge(rt,Tree.build(i));
    for(int i=1;i<=m;i++)
      {
          int l,r,a,b,c;
          scanf("%d%d",&l,&r);
          Tree.split(rt,l-1,a,b);
        Tree.split(b,r-l+1,b,c);
        Tree.fl[b]^=1;
        rt=Tree.merge(a,Tree.merge(b,c));
      }
    Tree.coutt(rt);
    return 0;
}
```

可持久化

```
#include<cstdio>
#include<cctype>
#include<cstring>
#include<cstdlib>
#include<ctime>
#include<utility>
#include<algorithm>
using namespace std;
typedef pair<int,int> Pair;
int read() {
    int x=0,f=1;
    char c=getchar();
    for (;!isdigit(c);c=getchar()) if (c=='-') f=-1;
    for (;isdigit(c);c=getchar()) x=x*10+c-'0';
    return x*f;
}
const int maxn=5e4+5;
const int nlogn=1.3e7+5;
struct node {
    int x,hp,l,r,sum,size;
    bool rev;
    void clear() {
        x=hp=l=r=sum=size=rev=0;
    }
};
struct TREAP {
    int pool[nlogn];
    int pooler;
    node t[nlogn];
    int now,all;
    int root[maxn];
    TREAP ():now(0),pooler(1) {
        for (int i=1;i<nlogn;++i) pool[i]=i;
        root[now]=pool[pooler++];
    }
    int newroot() {
        int ret=pool[pooler++];
        return ret;
    }
    int newnode(int x) {
        int ret=pool[pooler++];
        t[ret].hp=rand();
        t[ret].size=1;
        t[ret].x=t[ret].sum=x;
        return ret;
    }
    void delnode(int x) {
        t[x].clear();
        pool[--pooler]=x;
    }
    void next() {
        root[++all]=newroot();
        t[root[all]]=t[root[now]];
        now=all;
    }
    void back(int x) {
        now=x;
    }
    void update(int x) {
        t[x].sum=t[x].x+t[t[x].l].sum+t[t[x].r].sum;
        t[x].size=t[t[x].l].size+t[t[x].r].size+1;
    }
    void pushdown(int x) {
        if (!t[x].rev) return;
        if (t[x].l) {
            int tx=newnode(t[t[x].l].x);
            t[tx]=t[t[x].l];
            t[tx].rev^=true;
            t[x].l=tx;
        }
        if (t[x].r) {
            int tx=newnode(t[t[x].r].x);
            t[tx]=t[t[x].r];
            t[tx].rev^=true;
            t[x].r=tx;
        }
        swap(t[x].l,t[x].r);
        t[x].rev=false;
    }
    int merge(int x,int y) {
        if (!x) return y;
        if (!y) return x;
        int now;
        if (t[x].hp<=t[y].hp) {
            now=newnode(t[x].x);
            t[now]=t[x];
            pushdown(now);
            t[now].r=merge(t[now].r,y);
        } else {
            now=newnode(t[y].x);
            t[now]=t[y];
            pushdown(now);
            t[now].l=merge(x,t[now].l);
        }
        update(now);
        return now;
    }
    Pair split(int x,int p) {
        if (t[x].size==p) return make_pair(x,0);
        int now=newnode(t[x].x);
        t[now]=t[x];
        pushdown(now);
        int l=t[now].l,r=t[now].r;
        if (t[l].size>=p) {
            t[now].l=0;
            update(now);
            Pair g=split(l,p);
            now=merge(g.second,now);
            return make_pair(g.first,now);
        } else if (t[l].size+1==p) {
            t[now].r=0;
            update(now);
            return make_pair(now,r);
        } else {
            t[now].r=0;
            update(now);
            Pair g=split(r,p-t[l].size-1);
            now=merge(now,g.first);
            pushdown(now);
            return make_pair(now,g.second);
        }
    }
    void rever(int l,int r) {
        ++l,++r;
        Pair g=split(root[now],l-1);
        Pair h=split(g.second,r-l+1);
        int want=h.first;
        int here=newnode(t[want].x);
        t[here]=t[want];
        t[here].rev^=true;
        int fi=merge(g.first,here);
        int se=merge(fi,h.second);
        root[now]=se;
    }
    int query(int l,int r) {
        ++l,++r;
        Pair g=split(root[now],l-1);
        Pair h=split(g.second,r-l+1);
        int want=h.first;
        int ret=t[want].sum;
        int fi=merge(g.first,want);
```

```
        int se=merge(fi,h.second);
        root[now]=se;
        return ret;
    }
    void insert(int x) {
        int k=newnode(x);
        root[now]=merge(root[now],k);
    }
} Treap;
int main() {
#ifndef ONLINE_JUDGE
    freopen("test.in","r",stdin);
    freopen("my.out","w",stdout);
#endif
    srand(time(0));
    int n=read(),m=read();
    for (int i=1;i<=n;++i) {
        int x=read();
        Treap.insert(x);
    }
    while (m--) {
        int op=read();
        if (op==1) {
            Treap.next();
            int l=read(),r=read();
            Treap.rever(l,r);
        } else if (op==2) {
            int l=read(),r=read();
            int ans=Treap.query(l,r);
            printf("%d\n",ans);
        } else if (op==3) {
            Treap.back(read());
        }
    }
    return 0;
}
```

Splay

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

const int N = 100010;
int m,n;
struct Splay
{
    int rt, tot, fa[N], ch[N][2], val[N], cnt[N], sz[N];
 // cnt 权值出现次数
    int maxVal=-0x3f3f3f3f;
    bool rev[N];
    void pushdown(int x)
    {
        if (rev[x])
        {
            swap(ch[x][0], ch[x][1]);
            rev[ch[x][0]] ^= 1;
            rev[ch[x][1]] ^= 1;
            rev[x] = 0;
        }
    }

    void maintain(int x)
    {
        sz[x] = sz[ch[x][0]] + sz[ch[x][1]] + cnt[x];
    }
    // 右儿子返回1 左儿子返回0
    bool get(int x) { return x == ch[fa[x]][1]; }
    void clear(int x)
    {
        ch[x][0] = ch[x][1] = fa[x] = val[x] = sz[x] = c
nt[x] = 0;
    }
    // 旋转操作
    void rotate(int x)
    {
        int y = fa[x], z = fa[y], chk = get(x);
        // x 是左儿子右旋，右儿子左旋
        ch[y][chk] = ch[x][chk ^ 1];
        if (ch[x][chk ^ 1])
            fa[ch[x][chk ^ 1]] = y;
        ch[x][chk ^ 1] = y;
        fa[y] = x;
        fa[x] = z;
        if (z)
            ch[z][y == ch[z][1]] = x;
        maintain(x);
        maintain(y);
    }

    // Slpay 操作
    void splay(int x, int goal = 0)
    {
        for (int f = fa[x]; f = fa[x], f != goal; rotate
(x))
            if (fa[f] != goal)
                rotate(get(x) == get(f) ? f : x);
        if (!goal)
            rt = x;
    }

    // 插入
    void ins(int k)
    {
        maxVal=max(maxVal,k);
        // 树空
        if (!rt)
        {
            val[++tot] = k;
            cnt[tot]++;
            rt = tot;
            maintain(rt);
            return;
        }
        int cur = rt, f = 0;
        while (1)
        {
            if (val[cur] == k)
            {
                cnt[cur]++;
                maintain(cur);
                maintain(f);
                splay(cur);
                break;
            }
            f = cur;
            cur = ch[cur][val[cur] < k];
            if (!cur)
            {
                val[++tot] = k;
                cnt[tot]++;
                fa[tot] = f;
                ch[f][val[f] < k] = tot;
                maintain(tot);
                maintain(f);
                splay(tot);
                break;
            }
        }
    }
    // 查询排名  等价于find
    int rk(int k)
    {
        int res = 0, cur = rt;
        while (1)
        {
            if (k < val[cur])
            {
                cur = ch[cur][0];
            }
            else
            {
                res += sz[ch[cur][0]];
                if (k == val[cur])
                {
                    splay(cur);
                    return res + 1;
                }
                res += cnt[cur];
                cur = ch[cur][1];
            }
        }
    }
    // 查询第k大 索引
    int kth_idx(int k)
    {
        int cur = rt;
        while (1)
        {
            pushdown(cur);
            if (ch[cur][0] && k <= sz[ch[cur][0]])
            {
                cur = ch[cur][0];
            }
            else
```

```
        {
            k -= cnt[cur] + sz[ch[cur][0]];
            if (k <= 0)
            {
                splay(cur);
                return cur;
            }
            cur = ch[cur][1];
        }
    }
}
// 查询第k大 值
int kth_val(int k) { return val[kth_idx(k)]; }
// 查询前驱
int pre()
{
    int cur = ch[rt][0];
    if (!cur)
        return cur;
    while (ch[cur][1])
        cur = ch[cur][1];
    splay(cur);
    return cur;
}
// 查询后继
int nxt()
{
    int cur = ch[rt][1];
    if (!cur)
        return cur;
    while (ch[cur][0])
        cur = ch[cur][0];
    splay(cur);
    return cur;
}
// pre 封装
int q_pre(int x)
{
    ins(x);
    int ret = val[pre()];
    del(x);
    return ret;
}
// nxt 封装
int q_nxt(int x)
{
    ins(x);
    int ret = val[nxt()];
    del(x);
    return ret;
}
// 删除
void del(int k)
{
    rk(k);
    if (cnt[rt] > 1)
    {
        cnt[rt]--;
        maintain(rt);
        return;
    }
    if (!ch[rt][0] && !ch[rt][1])
    {
        clear(rt);
        rt = 0;
        return;
    }
    if (!ch[rt][0])
    {
        int cur = rt;
        rt = ch[rt][1];
        fa[rt] = 0;
        clear(cur);
        return;
    }
    if (!ch[rt][1])
    {
        int cur = rt;
        rt = ch[rt][0];
        fa[rt] = 0;
        clear(cur);
        return;
    }
    int cur = rt, x = pre();
    fa[ch[cur][1]] = x;
    ch[x][1] = ch[cur][1];
    clear(cur);
    maintain(rt);
}

void reverse(int l, int r)
{
    int x = kth_idx(l), y = kth_idx(r + 2);
    splay(x);
    splay(y, x);
    rev[ch[y][0]] ^= 1;
}
// 打印索引为x 的节点及其子树
void output(int x)
{
    pushdown(x);
    if (ch[x][0])
        output(ch[x][0]);
    if (val[x] && val[x] <= n)
        cout << val[x] <<" ";
    if (ch[x][1])
        output(ch[x][1]);
}
//打印整颗树
void print_tree()
{
    output(rt);
}
// 已知序列建树
int build(int l,int r)
{
    int x=++tot;
    int mid=(l+r)/2;
    cnt[tot]++;
    // val[tot]=xxxx;
    if(l==r){
        maintain(x);
        return x;
    }
    ch[x][0]=build(l,mid-1);
    ch[x][1]=build(mid+1,r);
    maintain(x);
    return x;
  }
} tree;
```

LCA

```
// 倍增方法:
#include<bits/stdc++.h>
#define MXN 50007
using namespace std;
std::vector<int> v[MXN];
std::vector<int> w[MXN];
int fa[MXN][31], cost[MXN][31], dep[MXN];
int n, m;
int a, b, c;
void dfs(int root, int fno) {
        fa[root][0] = fno;
        dep[root] = dep[fa[root][0]] + 1;
        for (int i = 1; i < 31; ++i) {
                fa[root][i] = fa[fa[root][i - 1]][i -
1];
                cost[root][i] = cost[fa[root][i - 1]]
[i - 1] + cost[root][i - 1];
        }
        int sz = v[root].size();
        for (int i = 0; i < sz; ++i) {
                if (v[root][i] == fno) continue;
                cost[v[root][i]][0] = w[root][i];
                dfs(v[root][i], root);
        }
}
int lca(int x, int y) {
        if (dep[x] > dep[y]) swap(x, y);
        int tmp = dep[y] - dep[x], ans = 0;
        for (int j = 0; tmp; ++j, tmp >>= 1)
                if (tmp & 1) ans += cost[y][j], y = fa
[y][j];
        if (y == x) return ans;
        for (int j = 30; j >= 0 && y != x; --j) {
                if (fa[x][j] != fa[y][j]) {
                        ans += cost[x][j] + cost[y][j];
                        x = fa[x][j];
                        y = fa[y][j];
                }
        }
        ans += cost[x][0] + cost[y][0];
        return ans;
}
```

```
int main() {
    ios::sync_with_stdio(false);
        memset(fa, 0, sizeof(fa));
        memset(cost, 0, sizeof(cost));
                memset(dep, 0, sizeof(dep));
        // scanf("%d", &n);
    cin>>n;
        for (int i = 1; i < n; ++i) {
                // scanf("%d %d %d", &a, &b, &c);
        cin>>a>>b>>c;
                ++a, ++b;
                v[a].push_back(b);
                v[b].push_back(a);
                w[a].push_back(c);
                w[b].push_back(c);
        }
        dfs(1, 0);
        // scanf("%d", &m);
    cin>>m;
        for (int i = 0; i < m; ++i) {
                // scanf("%d %d", &a, &b);
                cin>>a>>b;
        ++a, ++b;
                // printf("%d\n", lca(a, b));
        cout<<lca(a,b)<<"\n";
        }
        return 0;
}

// Tarjan 方法:
#include <algorithm>
#include <iostream>
using namespace std;
class Edge {
public:
        int toVertex, fromVertex;
        int next;
        int LCA;
       Edge() : toVertex(-1), fromVertex(-1), next(-1),
 LCA(-1) {};
        Edge(int u, int v, int n) : fromVertex(u), toVe
rtex(v), next(n), LCA(-1) {};
};
const int MAX = 100;
int head[MAX], queryHead[MAX];
Edge edge[MAX], queryEdge[MAX];
int parent[MAX], visited[MAX];
int vertexCount, edgeCount, queryCount;
void init() {
        for (int i = 0; i <= vertexCount; i++) {
                parent[i] = i;
        }
}
int find(int x) {
        if (parent[x] == x) {
                return x;
        }
        else {
                return find(parent[x]);
        }
}
void tarjan(int u) {
        parent[u] = u;
        visited[u] = 1;
       for (int i = head[u]; i != -1; i = edge[i].next)
 {
                Edge& e = edge[i];
                if (!visited[e.toVertex]) {
                        tarjan(e.toVertex);
                        parent[e.toVertex] = u;
                }
        }
        for (int i = queryHead[u]; i != -1; i = queryEd
ge[i].next) {
                Edge& e = queryEdge[i];
                if (visited[e.toVertex]) {
                        queryEdge[i ^ 1].LCA = e.LCA =
 find(e.toVertex);
                }
        }
}
int main() {
        memset(head, 0xff, sizeof(head));
        memset(queryHead, 0xff, sizeof(queryHead));
        cin >> vertexCount >> edgeCount >> queryCount;
        int count = 0;
        for (int i = 0; i < edgeCount; i++) {
                int start = 0, end = 0;
                cin >> start >> end;
                        edge[count] = Edge(start, end,
 head[start]);
                head[start] = count;
                count++;
                edge[count] = Edge(end, start, head[en
d]);
                head[end] = count;
                count++;
        }
        count = 0;
        for (int i = 0; i < queryCount; i++) {
                int start = 0, end = 0;
                cin >> start >> end;
                queryEdge[count] = Edge(start, end, qu
eryHead[start]);
                queryHead[start] = count;
                count++;
                queryEdge[count] = Edge(end, start, qu
eryHead[end]);
                queryHead[end] = count;
                count++;
        }
        init();
        tarjan(1);
        for (int i = 0; i < queryCount; i++) {
                Edge& e = queryEdge[i * 2];
                cout << "(" << e.fromVertex << "," <<
e.toVertex << ") " << e.LCA << endl;
        }
        return 0;
}
```

大数

```
//注：可以直接把BigInt 和一样用cin cout 都行，就是高精乘为了
速度才用了FFT 降低了精度，有需要可以自行更改。
#include <cstdio>
#include <iostream>
#include <cmath>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
using namespace std;
const double PI = acos(-1.0);
struct Complex{
    double x,y;
    Complex(double _x = 0.0,double _y = 0.0){
        x = _x;
        y = _y;
    }
    Complex operator-(const Complex &b)const{
        return Complex(x - b.x,y - b.y);
    }
    Complex operator+(const Complex &b)const{
        return Complex(x + b.x,y + b.y);
    }
    Complex operator*(const Complex &b)const{
        return Complex(x*b.x - y*b.y,x*b.y + y*b.x);
    }
};
void change(Complex y[],int len){
    int i,j,k;
    for(int i = 1,j = len/2;i<len-1;i++){
        if(i < j)    swap(y[i],y[j]);
        k = len/2;
        while(j >= k){
            j = j - k;
            k = k/2;
        }
        if(j < k)    j+=k;
    }
}
void fft(Complex y[],int len,int on){
    change(y,len);
    for(int h = 2;h <= len;h<<=1){
        Complex wn(cos(on*2*PI/h),sin(on*2*PI/h));
        for(int j = 0;j < len;j += h){
            Complex w(1,0);
            for(int k = j;k < j + h/2;k++){
                Complex u = y[k];
                Complex t = w*y[k + h/2];
                y[k] = u + t;
                y[k + h/2] = u - t;
                w = w*wn;
            }
        }
```

```
    }
    if(on == -1){
        for(int i = 0;i < len;i++){
            y[i].x /= len;
        }
    }
}
class BigInt
{
#define Value(x, nega) ((nega) ? -(x) : (x))
#define At(vec, index) ((index) < vec.size() ? vec[(inde
x)] : 0)
    static int absComp(const BigInt &lhs, const BigInt &
rhs)
    {
        if (lhs.size() != rhs.size())
            return lhs.size() < rhs.size() ? -1 : 1;
        for (int i = lhs.size() - 1; i >= 0; --i)
            if (lhs[i] != rhs[i])
                return lhs[i] < rhs[i] ? -1 : 1;
        return 0;
    }
    using Long = long long;
    const static int Exp = 9;
    const static Long Mod = 1000000000;
    mutable std::vector<Long> val;
    mutable bool nega = false;
    void trim() const
    {
        while (val.size() && val.back() == 0)
            val.pop_back();
        if (val.empty())
            nega = false;
    }
    int size() const { return val.size(); }
    Long &operator[](int index) const { return val[inde
x]; }
    Long &back() const { return val.back(); }
    BigInt(int size, bool nega) : val(size), nega(nega)
{}
    BigInt(const std::vector<Long> &val, bool nega) : va
l(val), nega(nega) {}

public:
    friend std::ostream &operator<<(std::ostream &os, co
nst BigInt &n)
    {
        if (n.size())
        {
            if (n.nega)
                putchar('-');
            for (int i = n.size() - 1; i >= 0; --i)
            {
                if (i == n.size() - 1)
                    printf("%lld", n[i]);
                else
                    printf("%0*lld", n.Exp, n[i]);
            }
        }
        else
            putchar('0');
        return os;
    }
    friend BigInt operator+(const BigInt &lhs, const Big
Int &rhs)
    {
        BigInt ret(lhs);
        return ret += rhs;
    }
    friend BigInt operator-(const BigInt &lhs, const Big
Int &rhs)
    {
        BigInt ret(lhs);
        return ret -= rhs;
    }
    BigInt(Long x = 0)
    {
        if (x < 0)
            x = -x, nega = true;
        while (x >= Mod)
            val.push_back(x % Mod), x /= Mod;
        if (x)
            val.push_back(x);
    }
    BigInt(const char *s)
    {
        int bound = 0, pos;
        if (s[0] == '-')
            nega = true, bound = 1;
        Long cur = 0, pow = 1;
        for (pos = strlen(s) - 1; pos >= Exp + bound - 1;
 pos -= Exp, val.push_back(cur), cur = 0, pow = 1)
            for (int i = pos; i > pos - Exp; --i)
                cur += (s[i] - '0') * pow, pow *= 10;
        for (cur = 0, pow = 1; pos >= bound; --pos)
            cur += (s[pos] - '0') * pow, pow *= 10;
        if (cur)
            val.push_back(cur);
    }
    BigInt &operator=(const char *s){
        BigInt n(s);
        *this = n;
        return n;
    }
    BigInt &operator=(const Long x){
        BigInt n(x);
        *this = n;
        return n;
    }
    friend std::istream &operator>>(std::istream &is, Bi
gInt &n){
        string s;
        is >> s;
        n=(char*)s.data();
        return is;
    }
    BigInt &operator+=(const BigInt &rhs)
    {
        const int cap = std::max(size(), rhs.size()) + 1;
        val.resize(cap);
        int carry = 0;
        for (int i = 0; i < cap - 1; ++i)
        {
            val[i] = Value(val[i], nega) + Value(At(rhs,
 i), rhs.nega) + carry, carry = 0;
            if (val[i] >= Mod)
                val[i] -= Mod, carry = 1;
            else if (val[i] < 0)
                val[i] += Mod, carry = -1;
        }
        if ((val.back() = carry) == -1) //assert(val.bac
k() == 1 or 0 or -1)
        {
            nega = true, val.pop_back();
            bool tailZero = true;
            for (int i = 0; i < cap - 1; ++i)
            {
                if (tailZero && val[i])
                    val[i] = Mod - val[i], tailZero = fa
lse;
                else
                    val[i] = Mod - 1 - val[i];
            }
        }
        trim();
        return *this;
    }
    friend BigInt operator-(const BigInt &rhs)
    {
        BigInt ret(rhs);
        ret.nega ^= 1;
        return ret;
    }
    BigInt &operator-=(const BigInt &rhs)
    {
        rhs.nega ^= 1;
        *this += rhs;
        rhs.nega ^= 1;
        return *this;
    }
    friend BigInt operator*(const BigInt &lhs, const Big
Int &rhs)
    {
        int len=1;
        BigInt ll=lhs,rr=rhs;
        ll.nega = lhs.nega ^ rhs.nega;
        while(len<2*lhs.size()||len<2*rhs.size())len<<=1;
        ll.val.resize(len),rr.val.resize(len);
        Complex x1[len],x2[len];
        for(int i=0;i<len;i++){
            Complex nx(ll[i],0.0),ny(rr[i],0.0);
            x1[i]=nx;
            x2[i]=ny;
        }
        fft(x1,len,1);
        fft(x2,len,1);
```

```
        for(int i = 0 ; i < len; i++)
            x1[i] = x1[i] * x2[i];
        fft( x1 , len , -1 );
        for(int i = 0 ; i < len; i++)
            ll[i] = int( x1[i].x + 0.5 );
        for(int i = 0 ; i < len; i++){
            ll[i+1]+=ll[i]/Mod;
            ll[i]%=Mod;
        }
        ll.trim();
        return ll;
    }
    friend BigInt operator*(const BigInt &lhs, const Lon
g &x){
        BigInt ret=lhs;
        bool negat = ( x < 0 );
        Long xx = (negat) ? -x : x;
        ret.nega ^= negat;
        ret.val.push_back(0);
        ret.val.push_back(0);
        for(int i = 0; i < ret.size(); i++)
            ret[i]*=xx;
        for(int i = 0; i < ret.size(); i++){
            ret[i+1]+=ret[i]/Mod;
            ret[i] %= Mod;
        }
        ret.trim();
        return ret;
    }
    BigInt &operator*=(const BigInt &rhs) { return *this
 = *this * rhs; }
    BigInt &operator*=(const Long &x) { return *this = *
this * x; }
    friend BigInt operator/(const BigInt &lhs, const Big
Int &rhs)
    {
        static std::vector<BigInt> powTwo{BigInt(1)};
        static std::vector<BigInt> estimate;
        estimate.clear();
        if (absComp(lhs, rhs) < 0)
            return BigInt();
        BigInt cur = rhs;
        int cmp;
        while ((cmp = absComp(cur, lhs)) <= 0)
        {
            estimate.push_back(cur), cur += cur;
            if (estimate.size() >= powTwo.size())
                powTwo.push_back(powTwo.back() + powTwo.
back());
        }
        if (cmp == 0)
            return BigInt(powTwo.back().val, lhs.nega ^
rhs.nega);
        BigInt ret = powTwo[estimate.size() - 1];
        cur = estimate[estimate.size() - 1];
        for (int i = estimate.size() - 1; i >= 0 && cmp !
= 0; --i)
            if ((cmp = absComp(cur + estimate[i], lhs))
<= 0)
                cur += estimate[i], ret += powTwo[i];
        ret.nega = lhs.nega ^ rhs.nega;
        return ret;
    }
    friend BigInt operator/(const BigInt &num,const Long
 &x){
        bool negat = ( x < 0 );
        Long xx = (negat) ? -x : x;
        BigInt ret;
        Long k = 0;
        ret.val.resize( num.size() );
        ret.nega = (num.nega ^ negat);
        for(int i = num.size() - 1 ;i >= 0; i--){
            ret[i] = ( k * Mod + num[i]) / xx;
            k = ( k * Mod + num[i]) % xx;
        }
        ret.trim();
        return ret;
    }
    bool operator==(const BigInt &rhs) const
    {
        return nega == rhs.nega && val == rhs.val;
    }
    bool operator!=(const BigInt &rhs) const { return ne
ga != rhs.nega || val != rhs.val; }
    bool operator>=(const BigInt &rhs) const { return !
(*this < rhs); }
    bool operator>(const BigInt &rhs) const { return !(*
this <= rhs); }
    bool operator<=(const BigInt &rhs) const
    {
        if (nega && !rhs.nega)
            return true;
        if (!nega && rhs.nega)
            return false;
        int cmp = absComp(*this, rhs);
        return nega ? cmp >= 0 : cmp <= 0;
    }
    bool operator<(const BigInt &rhs) const
    {
        if (nega && !rhs.nega)
            return true;
        if (!nega && rhs.nega)
            return false;
        return (absComp(*this, rhs) < 0) ^ nega;
    }
    void swap(const BigInt &rhs) const
    {
        std::swap(val, rhs.val);
        std::swap(nega, rhs.nega);
    }
};
BigInt ba,bb;
int main(){
    cin>>ba>>bb;
    cout << ba + bb << '\n';//和
    cout << ba - bb << '\n';//差
    cout << ba * bb << '\n';//积
    BigInt d;
    cout << (d = ba / bb) << '\n';//商
    cout << ba - d * bb << '\n';//余
    return 0;
}
```

带修改整体二分

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

const int maxn = 1e5 + 5;
struct query1
{
    int type;
    int l, r, k;
    int id;
    // 对于询问, type = 1, l, r 表示区间左右边界, k 表示询
问第 k 小
    // 对于修改, type = 0, l 表示修改位置, r 表示修改后的值,
    // k 表示当前操作是插入(1)还是擦除(-1), 更新树状数组时使
用.
    // id 记录每个操作原先的编号, 因二分过程中操作顺序会被打
散
};
struct num1
{
    int p, x;
};
vector<int> ans(100000);
int n, m;
int t[maxn];
int a[maxn];
int sum(int p)
{
    int ans = 0;
    while (p)
    {
        ans += t[p];
        p -= p & (-p);
    }
    return ans;
}
void add(int p, int x)
{
    while (p <= n)
    {
        t[p] += x;
        p += p & (-p);
    }
}
void solve(int l, int r, vector<query1> &q)
{
    if (q.size() == 0)
        return;
    if (l == r)
    {
        for (auto i : q)
```

```
        {
            if (i.type == 1)
                ans[i.id] = l;
        }
        return;
    }
    int mid = (l + r) >> 1;
    vector<query1> q1;
    vector<query1> q2;
    for (auto i : q)
    {
        if (i.type == 1)
        {
            int t = sum(i.r) - sum(i.l - 1);
            if (i.k <= t)
                q1.push_back(i);
            else
            {
                i.k -= t;
                q2.push_back(i);
            }
        }
        else
        {
            if (i.r <= mid)
            {
                add(i.l, i.k);
                q1.push_back(i);
            }
            else
            {
                q2.push_back(i);
            }
        }
    }
    for (auto i : q)
        if(i.type==0)
            if (i.r <= mid)
            {
                add(i.l, -i.k);
            }
    solve(l, mid, q1);
    solve(mid + 1, r, q2);
}
signed main()
{
    ios::sync_with_stdio(false);
    int tmp1, tmp2, tmp3;
    vector<query1> query;
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
    {
        cin >> a[i];
        query.push_back({0, i, a[i], 1, 0});
    }
    char op;
    int l, r, k;
    int cnt = 0;
    int x, y;
    for (int i = 1; i <= m; i++)
    {
        cin >> op;
        if (op == 'Q')
        {
            cin >> l >> r >> k;
            query.push_back({1, l, r, k, ++cnt});
        }
        else if (op == 'C')
        {
            cin >> x >> y;
            query.push_back({0, x, a[x], -1, 0});
            a[x]=y;
            query.push_back({0, x, y, 1, 0});

        }
    }
    solve(0, 1e9, query);
    for (int i = 1; i <= cnt; i++)
        cout << ans[i] << "\n";
}
```

数学

常系数齐次线性递推数列的特征方程

对于 k 阶**常系数齐次线性递推数列**：

$$a_{n+k} = c_1a_{n+k-1} + c_2a_{n+k-2} + \cdots + c_{k-1}a_{n+1} + c_ka_n$$

其中 $n \in \mathbb{N}, k \in \mathbb{N}^*$，$c_1, c_2, \cdots, c_{k-1}, c_k$ 是常数，$c_k \neq 0$ 来说,代数方程 $x^k - c_1x^{k-1} - c_2x^{k-2} - \cdots - c_{k-1}x - c_k = 0$

叫做该线性递推数列的**特征方程**

以二阶线性递推数列为例，设两个特征根分别为 ϕ_1 和 ϕ_2

则 $f_n = C_1\phi_1 + C_2\phi_2$，利用数列前几项求解系数即可

线形基

```
#include<bits/stdc++.h>
using namespace std;
// #pragma GCC optimize(2)
#define fi first
#define se second
typedef pair<int, int> pii;
typedef long long ll;
typedef unsigned long long ull;
typedef long double ld;
void io() { ios::sync_with_stdio(false); cin.tie(0); cou
t.tie(0); }
template<typename T>
inline void debug(T const& x) { cout << x << "\n"; }
```

```
struct LinearBase {
    const int siz=64;
    int MN;
    vector<ll>p, tmp;
    bool flag = false;
    LinearBase(){
        p.resize(siz);
        tmp.resize(siz);
        MN=siz-1;
    }

    void clear()
    {
        // siz = MN = 0;
        p.clear(); tmp.clear();
        flag = false;
    }

    void resize(int size)
    {
        p.resize(siz);
        tmp.resize(siz);
        MN = siz - 1;
        flag = false;
    }

    void insert(ll x)
    {
        for (int i = MN; ~i; --i) {
            if (x & (1ll << i)) {
                if (!p[i]) {
                    p[i] = x;
                    return;
                }
                else
                    x ^= p[i];
            }
        }
        flag = true;
    }

    bool check(ll x)
    {
        for (int i = MN; ~i; i--) {
            if (x & (1ll << i)) {
                if (!p[i]) return false;
                else x ^= p[i];
            }
        }
        return true;
    }

    ll Qmax()
    {
        ll res = 0ll;
        for (int i = MN; ~i; --i) {
            res = max(res, res ^ p[i]);
        }
```

```
        return res;
    }

    ll Qmin()
    {
        if (flag) return 0;
        for (int i = 0; i <= MN; ++i)
            if (p[i]) p[i];
    }

    // void rebuild()
    // {
    //     int cnt=0,top=0;
    //     for(int i=MN;~i)
    // }

    ll Qnth_element(ll k)
    {
        ll res = 0;
        int cnt = 0;
        k -= flag;
        if (!k) return 0;
        for (int i = 0; i <= MN; ++i) {
            for (int j = i - 1; ~j; j--) {
                if (p[i] & (1ll << j)) p[i] ^= p[j];
            }
            if (p[i]) tmp[cnt++] = p[i];
        }
        if (k >= (1ll << cnt)) return -1;
        for (int i = 0; i < cnt; ++i)
            if (k & (1ll << i))
                res ^= tmp[i];
        return res;
    }


};



int main()
{
    io();
    int n;
    cin >> n;
    LinearBase lb;
    for (int i = 0; i < n; ++i) {
        ll _;
        cin >> _;
        lb.insert(_);
    }
    cout << lb.Qmax() << "\n";
    return 0;
}
```

pollard's rho

```
#include <bits/stdc++.h>
#define sz(x) int((x).size())
#define all(x) begin(x), end(x)

using namespace std;
template<class T>
using vc = vector<T>;
using ull = unsigned long long;
using ll = long long;

ull modmul(ull a, ull b, ull M) {
        ll ret = a * b - M * ull(1.L / M * a * b);
       return ret + M * (ret < 0) - M * (ret >= (ll)M);
}

ull modpow(ull b, ull e, ull mod) {
        ull ans = 1;
        for (; e; b = modmul(b, b, mod), e /= 2)
                if (e & 1) ans = modmul(ans, b, mod);
        return ans;
}

ull Qpow(ull b, int e) {
  ull res = 1;
  for (; e; b *= b, e /= 2) if (e & 1) res *= b;
  return res;
}

bool isPrime(ull p) {
        if (p == 2) return true;
        if (p == 1 || p % 2 == 0) return false;
        ull s = p - 1;
        while (s % 2 == 0) s /= 2;
  for (int i = 0; i < 15; ++i) {
                ull a = rand() % (p - 1) + 1, tmp = s;
                ull mod = modpow(a, tmp, p);
                while (tmp != p - 1 && mod != 1 && mod
 != p - 1) {
                        mod = modmul(mod, mod, p);
                        tmp *= 2;
                }
                if (mod != p - 1 && tmp % 2 == 0) retu
rn false;
        }
        return true;
}

ull pollard(ull n) {
        auto f = [n](ull x) { return modmul(x, x, n) +
1; };
        ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
        while (t++ % 40 || __gcd(prd, n) == 1) {
                if (x == y) x = ++i, y = f(x);
                if ((q = modmul(prd, max(x,y) - min(x,
y), n))) prd = q;
                x = f(x), y = f(f(y));
        }
        return __gcd(prd, n);
}
```



```
vector<ull> factor(ull n) {
        if (n == 1) return {};
        if (isPrime(n)) return {n};
        ull x = pollard(n);
        auto l = factor(x), r = factor(n / x);
        l.insert(l.end(), all(r));
        return l;
}

int main() {
#ifdef LOCAL
  freopen("in.txt", "r", stdin);
#endif
        cin.tie(nullptr)->sync_with_stdio(false);
  int n; cin >> n;
  while (n--) {
    ull x; cin >> x;
    auto fac = factor(x);
    map<ull, int> mp;
    for (auto e: fac) {
      ++mp[e];
    }
    ull ans = 1;
    for (auto p: mp) {
      ans *= Qpow(p.first, p.second / 3);
    }
    cout << ans << '\n';
  }
}
```

数论和杂项

扩欧求逆元

```
inline void exgcd(LL a, LL b, LL &x, LL &y)
{
    if (!b)
    {
        x = 1;
        y = 0;
        return;
    }
    exgcd(b, a % b, y, x);
    y -= a / b * x;
}
inline LL inv(LL a, LL mo)
{
    LL x, y;
    exgcd(a, mo, x, y);
    return x >= 0 ? x : x + mo;
}
```

NTT

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

inline int read() {
  int x = 0, f = 1;
```

```
  char ch = getchar();
  while (ch < '0' || ch > '9') {
    if (ch == '-') f = -1;
    ch = getchar();
  }
  while (ch <= '9' && ch >= '0') {
    x = 10 * x + ch - '0';
    ch = getchar();
  }
  return x * f;
}
void print(int x) {
  if (x < 0) putchar('-'), x = -x;
  if (x >= 10) print(x / 10);
  putchar(x % 10 + '0');
}

const int N = 300100, P = 998244353;

inline int qpow(int x, int y) {
  int res(1);
  while (y) {
    if (y & 1) res = 1ll * res * x % P;
    x = 1ll * x * x % P;
    y >>= 1;
  }
  return res;
}

int r[N];

void ntt(int *x, int lim, int opt) {
  register int i, j, k, m, gn, g, tmp;
  for (i = 0; i < lim; ++i)
    if (r[i] < i) swap(x[i], x[r[i]]);
  for (m = 2; m <= lim; m <<= 1) {
    k = m >> 1;
    gn = qpow(3, (P - 1) / m);
    for (i = 0; i < lim; i += m) {
      g = 1;
      for (j = 0; j < k; ++j, g = 1ll * g * gn % P) {
        tmp = 1ll * x[i + j + k] * g % P;
        x[i + j + k] = (x[i + j] - tmp + P) % P;
        x[i + j] = (x[i + j] + tmp) % P;
      }
    }
  }
  if (opt == -1) {
    reverse(x + 1, x + lim);
    register int inv = qpow(lim, P - 2);
    for (i = 0; i < lim; ++i) x[i] = 1ll * x[i] * inv %
P;
  }
}

int A[N], B[N], C[N];

char a[N], b[N];
```

```
int main() {
  register int i, lim(1), n;
  scanf("%s", &a);
  n = strlen(a);
  for (i = 0; i < n; ++i) A[i] = a[n - i - 1] - '0';
  while (lim < (n << 1)) lim <<= 1;
  scanf("%s", &b);
  n = strlen(b);
  for (i = 0; i < n; ++i) B[i] = b[n - i - 1] - '0';
  while (lim < (n << 1)) lim <<= 1;
  for (i = 0; i < lim; ++i) r[i] = (i & 1) * (lim >> 1)
+ (r[i >> 1] >> 1);
  ntt(A, lim, 1);
  ntt(B, lim, 1);
  for (i = 0; i < lim; ++i) C[i] = 1ll * A[i] * B[i] % P;
  ntt(C, lim, -1);
  int len(0);
  for (i = 0; i < lim; ++i) {
    if (C[i] >= 10) len = i + 1, C[i + 1] += C[i] / 10,
C[i] %= 10;
    if (C[i]) len = max(len, i);
  }
  while (C[len] >= 10) C[len + 1] += C[len] / 10, C[len]
 %= 10, len++;
  for (i = len; ~i; --i) putchar(C[i] + '0');
  puts("");
  return 0;
}
```

拉格朗日插值

```
#include <algorithm>
#include <cstdio>
#include <cstring>
const int maxn = 2010;
using ll = long long;
ll mod = 998244353;
ll n, k, x[maxn], y[maxn], ans, s1, s2;
ll powmod(ll a, ll x) {
        ll ret = 1ll, nww = a;
        while (x) {
                if (x & 1) ret = ret * nww % mod;
                        nww = nww * nww % mod;
                x >>= 1;
        }
        return ret;
}
ll inv(ll x) { return powmod(x, mod - 2); }
int main() {
        scanf("%lld%lld", &n, &k);
        for (int i = 1; i <= n; i++) scanf("%lld%lld",
x + i, y + i);
        for (int i = 1; i <= n; i++) {
                s1 = y[i] % mod;
                s2 = 1ll;
                for (int j = 1; j <= n; j++)
                        if (i != j)
                                s1 = s1 * (k - x[j]) %
 mod, s2 = s2 * ((x[i] - x[j] % mod) % mod) % mod;
                ans += s1 * inv(s2) % mod;
                ans = (ans + mod) % mod;
        }
        printf("%lld\n", ans);
        return 0;
}
```

高斯消元

```
void Gauss() {
        for(int i = 0; i < n; i ++) {
                r = i;
                for(int j = i + 1; j < n; j ++)
                        if(fabs(A[j][i]) > fabs(A[r]
[i])) r = j;
                if(r != i) for(int j = 0; j <= n; j ++)
 std :: swap(A[r][j], A[i][j]);

                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];
                }
        }

        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];
        }
}
```

组合数学

卡特兰数

$$K_n = \frac{C_{2n}^n}{n+1}$$

$$K_0 = K_1 = 1$$

$$K_n = \sum_{i=1}^{n} K_{i-1} K_{n-i} \ (n >= 2)$$

$$K_n = \frac{K_{n-1}(4n-2)}{n+1}$$

$$K_n = C_{2n}^n - C_{2n}^{n-1}$$

斯特林数

第二类斯特林数

将 n 个两两不同的元素划分为 k 个互不区分的非空子集方案数

$$S(n,k)=S(n-1,k-1)+k*S(n-1,k)$$

$$S(n,0)=[n=0]$$

$$S(n,m)=\sum_{i=0}^{m}\frac{(-1)^{m-i}i^n}{i!\,(m-i)!}$$

```
S[0][0] = 1;
        FOR (i, 1, N)
                FOR (j, 1, i + 1) S[i][j] = (S[i - 1]
[j - 1] + j * S[i - 1][j]) % MOD;
```

反演

二项式反演

fn 为至多，gn 为恰好的方案数

$$f_n=\sum_{i=0}^{n}C_n^i\,g_i$$

$$g_n=\sum_{i=0}^{n}(-1)^{n-i}\,C_n^i f_i$$

或者：

fn 为恰好，gi 为至少

$$g_k=\sum_{i=k}^{n}C_n^i\,f_i$$

$$f_k=\sum_{i=0}^{n}(-1)^{i-k}\,C_i^k g_i$$

min_max 反演

min{S}为集合最小值，max{S}为集合最大值

$$max\{S\}=\sum_{T\subseteq S,T!=0}(-1)^{|T|-1}\,min\{T\}$$

生成函数

普通生成函数

常用生成函数的开放，收敛转化：

$$1+px+p^2x^2+\ldots=\frac{1}{1-px}$$

$$\sum_{n=0}C_m^n\,x^n=(1+x)^m$$

$$\sum_{n=0}C_{n+m}^n\,x^n=\frac{1}{(1-x)^{m+1}}$$

自然数幂和表

```
MP = {
    0:"1 1 0",
    1:"2 1 1 0",
    2:"6 2 3 1 0",
    3:"4 1 2 1 0 0",
    4:"30 6 15 10 0 -1 0",
    5:"12 2 6 5 0 -1 0 0",
    6:"42 6 21 21 0 -7 0 1 0",
    7:"24 3 12 14 0 -7 0 2 0 0",
    8:"90 10 45 60 0 -42 0 20 0 -3 0",
    9:"20 2 10 15 0 -14 0 10 0 -3 0 0",
    10:"66 6 33 55 0 -66 0 66 0 -33 0 5 0",
    11:"24 2 12 22 0 -33 0 44 0 -33 0 10 0 0",
    12:"2730 210 1365 2730 0 -5005 0 8580 0 -9009 0 4550
 0 -691 0",
    13:"420 30 210 455 0 -1001 0 2145 0 -3003 0 2275 0 -
691 0 0",
    14:"90 6 45 105 0 -273 0 715 0 -1287 0 1365 0 -691 0
 105 0",
    15:"48 3 24 60 0 -182 0 572 0 -1287 0 1820 0 -1382 0
 420 0 0",
    16:"510 30 255 680 0 -2380 0 8840 0 -24310 0 44200 0
 -46988 0 23800 0 -3617 0",
    17:"180 10 90 255 0 -1020 0 4420 0 -14586 0 33150 0
-46988 0 35700 0 -10851 0 0",
    18:"3990 210 1995 5985 0 -27132 0 135660 0 -529074 0
 1469650 0 -2678316 0 2848860 0 -1443183 0 219335 0",
    19:"840 42 420 1330 0 -6783 0 38760 0 -176358 0 5878
60 0 -1339158 0 1899240 0 -1443183 0 438670 0 0",
    20:"6930 330 3465 11550 0 -65835 0 426360 0 -2238390
 0 8817900 0 -24551230 0 44767800 0 -47625039 0 24126850
 0 -3666831 0"
}
```

预处理组合数

```
LL C[M][M];
void init_C(int n) {
        FOR (i, 0, n) {
                C[i][0] = C[i][i] = 1;
                FOR (j, 1, i)
                        C[i][j] = (C[i - 1][j] + C[i -
 1][j - 1]) % MOD;
        }
}
```

OTHER

BM 线性递推

```
#include<bits/stdc++.h>
using namespace std;
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(),(x).end()
#define fi first
#define se second
#define SZ(x) ((int)(x).size())
typedef vector<int> VI;
typedef long long ll;
typedef pair<int,int> PII;
const ll mod=1000000007;
ll powmod(ll a,ll b) {ll res=1;a%=mod; assert(b>=0); for
(;b;b>>=1){if(b&1)res=res*a%mod;a=a*a%mod;}return res;}
// head

ll n;
namespace linear_seq {
    const int N=10010;
    ll res[N],base[N],_c[N],_md[N];

    vector<int> Md;
    void mul(ll *a,ll *b,int k) {
        rep(i,0,k+k) _c[i]=0;
        rep(i,0,k) if (a[i]) rep(j,0,k) _c[i+j]=(_c[i+j]
+a[i]*b[j])%mod;
        for (int i=k+k-1;i>=k;i--) if (_c[i])
            rep(j,0,SZ(Md)) _c[i-k+Md[j]]=(_c[i-k+Md[j]]
-_c[i]*_md[Md[j]])%mod;
        rep(i,0,k) a[i]=_c[i];
    }
    int solve(ll n,VI a,VI b) { // a 系数 b 初值 b[n+1]=a
[0]*b[n]+...
        ll ans=0,pnt=0;
        int k=SZ(a);
        assert(SZ(a)==SZ(b));
        rep(i,0,k) _md[k-1-i]=-a[i];_md[k]=1;
        Md.clear();
        rep(i,0,k) if (_md[i]!=0) Md.push_back(i);
        rep(i,0,k) res[i]=base[i]=0;
        res[0]=1;
```

```
        while ((1ll<<pnt)<=n) pnt++;
        for (int p=pnt;p>=0;p--) {
            mul(res,res,k);
            if ((n>>p)&1) {
                for (int i=k-1;i>=0;i--) res[i+1]=res[i];
res[0]=0;
                rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-r
es[k]*_md[Md[j]])%mod;
            }
        }
        rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
        if (ans<0) ans+=mod;
        return ans;
    }
    VI BM(VI s) {
        VI C(1,1),B(1,1);
        int L=0,m=1,b=1;
        rep(n,0,SZ(s)) {
            ll d=0;
            rep(i,0,L+1) d=(d+(ll)C[i]*s[n-i])%mod;
            if (d==0) ++m;
            else if (2*L<=n) {
                VI T=C;
                ll c=mod-d*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mo
d;
                L=n+1-L; B=T; b=d; m=1;
            } else {
                ll c=mod-d*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mo
d;
                ++m;
            }
        }
        return C;
    }
    int gao(VI a,ll n) {
        VI c=BM(a);
        c.erase(c.begin());
        rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
        return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));
    }
};

int main() {
    /*push_back 进去前 8~10 项左右、最后调用 gao 得第 n 项
*/
    vector<int>v;
    v.push_back(3);
    v.push_back(9);
    v.push_back(20);
    v.push_back(46);
    v.push_back(106);
    v.push_back(244);
    v.push_back(560);
    v.push_back(1286);
    v.push_back(2956);
    v.push_back(6794);
    int nCase;
    scanf("%d", &nCase);
    while(nCase--){
        scanf("%lld", &n);
        printf("%lld\n",1LL * linear_seq::gao(v,n-1) % m
od); ///求第 n 项
    }
}
```

常用宏及函数与快读

```
// v2021.5.22 主席树更面向对象


#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
__gnu_pbds::tree<int, __gnu_pbds::null_type, std::less<i
nt>, __gnu_pbds::rb_tree_tag, __gnu_pbds::tree_order_sta
tistics_node_update> TTT;

// 函数不返回值可能会 RE
// 少码大数据结构，想想复杂度更优的做法
// 小数 二分/三分 注意 break 条件
// 浮点运算 sqrt(a^2-b^2) 可用 sqrt(a+b)*sqrt(a-b) 代替，
避免精度问题
// long double -> %Lf 别用C11 (C14/16)
// 控制位数 cout << setprecision(10) << ans;
// reverse vector 注意判空 不然会re
// 分块注意维护块上标记 来更新块内数组 a[]
// vector+lower_bound 常数 < map/set/(unordered_map)
// map.find 不会创建新元素 map[]会 注意空间
// 别对指针用memset
// 用位运算表示 2^n 注意加 LL 1LL<<20
// 注意递归爆栈
// 注意边界
// 注意 memset 多组会T

// lambda

// sort(p + 1, p + 1 + n,
//              [](const point &x, const point &y) -> bo
ol { return x.x < y.x; });

// append l1 to l2 (l1 unchanged)

// l2.insert(l2.end(),l1.begin(),l1.end());

// append l1 to l2 (elements appended to l2 are removed
from l1)
// (general form ... TG gave form that is actually bette
r suited
//  for your needs)

// l2.splice(l2.end(),l1,l1.begin(),l1.end());
```

```
//位运算函数
//int __builtin_ffs (unsigned int x 最后一位 1 的是从后向前
第几位，1110011001000 返回4
//int __builtin_clz (unsigned int x)前导0个数
//int __builtin_ctz (unsigned int x)末尾0个数
//int __builtin_popcount (unsigned int x) 1的个数
//此外，这些函数都有相应的usigned long 和usigned long long
版本，只需要在函数名后面加上 l 或 ll 就可以了，比如int __built
in_clzll。

//java 大数
//import java.io.*;
//import java.math.BigInteger;
//import java.util.*;
//public class Main {
//    public static void main(String args[]) throws Exce
ption {
//        Scanner cin=new Scanner(System.in);
//        BigInteger a;
//        BigInteger b;
//        a = cin.nextBigInteger();
//       b = cin.nextBigInteger();
//
//        System.out.println(a.add(b));
//    }
//}
//
//生成超过32767的随机数
//unsigned seed = std::chrono::system_clock::now().time_
since_epoch().count();
//       mt19937 rand_num(seed);  // 大随机数
//       uniform_int_distribution<long long> dist(0, 100
0000000);  // 给定范围
//       cout << dist(rand_num) << endl;
```

常见博弈

巴什博弈

只有一堆 n 个物品，两个人轮流从这堆物品中取物，规定每次至少取一个，最多取 m 个。最后取光者得胜。

n%（m+1）==0 必败，否则必胜

威佐夫博弈

有两堆各若干个物品，两个人轮流从任意一堆中取出至少一个或者同时从两堆中取出同样多的物品，规定每次至少取一个，至多不限，最后取光者胜利。设两堆分别为 n 和 m

较小堆*两堆之差为（黄金分割比+1）时必败，否则比胜

```
int a=min(n,m);
int b=max(n,m);
double r=(sqrt(5.0)+1)/2;
```

```
double c=(double)b-a;
int temp=(int)(r*c);
if(temp==a)
    败
else
    胜
```

nim 博弈

有若干堆各若干个物品，两个人轮流从某一堆取任意多的物品，规定每次至少取一个，多者不限，最后取光者得胜。

各堆物品异或为 0 时必败，否则必胜

anti-nim 博弈

有若干堆各若干个物品，两个人轮流从某一堆取任意多的物品，规定每次至少取一个，多者不限，最后取光者得败。

先手胜当且仅当 ①所有堆石子数都为 1 且游戏的 SG 值为 0（即有偶数个孤单堆-每堆只有 1 个石子数）；②存在某堆石子数大于 1 且游戏的 SG 值不为 0.

阶梯博弈

地面表示第 0 号阶梯。每次都可以将一个阶梯上的石子向其左侧移动任意个石子，没有可以移动的空间时（及所有石子都位于地面时）输。

阶梯博弈等效为奇数号阶梯的尼姆博弈

对拍

- check.cpp (Windows)

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

using namespace std;
int main()
{
    system("g++ data.cpp -o data --std=c++17");
    system("g++ std.cpp -o std --std=c++17");
    system("g++ test.cpp -o test --std=c++17");
    int t = 10000;
    while (t--)
    {
        system("data.exe > data.txt");
        clock_t st=clock();
        system("test.exe < P3372_8.in > test.out");
        clock_t end=clock();
        system("std.exe < data.txt > std.txt");
        if (system("fc P3372_8.out test.out"))
            t=-1;
            break;
        cout<<"TIME: "<<end-st<<" ms\n\n";
    }
    if (t == 0)
        cout << "Accepted!" << endl;
    else
        cout << "Wrong Answer!" << endl;
    return 0;
}
```

- check.cpp (Linux)

重点！数据比较器

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    system("g++ ./data.cpp -o data --std=c++17");
    system("g++ ./std.cpp -o std --std=c++17");
    system("g++ ./test.cpp -o test --std=c++17");
    int t = 10000;
    while (t--)
    {
        system("./data.exe > ./data.txt");
        clock_t st = clock();
        system("./test.exe < ./data.txt > ./test.txt");
        clock_t end = clock();
        system("./std.exe < ./data.txt > ./std.txt");
        if (system("diff ./std.txt ./test.txt"))
            t=-1;
            break;
        cout << "TIME: " << end - st << " ms\n\n";
    }
    if (t == 0)
        cout << "Accepted!" << endl;
    else
        cout << "Wrong Answer!" << endl;
    return 0;
}
```

- data.cpp 生成数据
- std.cpp 暴力程序
- test.cpp 需确认正确性

质数表

1e2	1e3	1e4	1e5	1e6
101	1009	10007	100003	1000003
103	1013	10009	100019	1000033
107	1019	10037	100043	1000037
109	1021	10039	100049	1000039
113	1031	10061	100057	1000081
127	1033	10067	100069	1000099
131	1039	10069	100103	1000117
137	1049	10079	100109	1000121
139	1051	10091	100129	1000133
149	1061	10093	100151	1000151
151	1063	10099	100153	1000159

1e7	1e8	1e10	1e11	1e12
10000019	100000007	1000000007	10000000019	100000000003
10000079	100000037	1000000009	10000000033	100000000019
10000103	100000039	1000000021	10000000061	100000000057

1e13	1e14	1e15	1e16
10000000000037	100000000000031	1000000000000037	10000000000000061
10000000000051	100000000000067	1000000000000091	10000000000000069
10000000000099	100000000000097	1000000000000159	10000000000000079
10000000000129	100000000000099	1000000000000187	10000000000000099
10000000000183	100000000000133	1000000000000223	10000000000000453

1e17	1e18
100000000000000003	1000000000000000003
100000000000000013	1000000000000000009
100000000000000019	1000000000000000031
100000000000000021	1000000000000000079
100000000000000049	1000000000000000177