Contents

一切的开始 **3**

宏定义 3
快速读 3
对拍 4

数据结构 **4**

ST 表 4
线段树 5
均摊复杂度线段树 8
持久化线段树 9
K-D Tree 9
树状数组 12
主席树 13
左偏树 15
Treap 16
Treap-序列 17
可回滚并查集 21
舞蹈链 21
CDQ 分治 24
哈希表 25
笛卡尔树 26
Trie 26
pb_ds 27
Link-Cut Tree 28
莫队 29

数学 **30**

矩阵运算 30
筛 30
素数测试 31
线性递推 32
扩展欧几里得 33
逆元 33
组合数 33
simpson 自适应积分 35
快速乘 35
快速幂 36
高斯消元 36
质因数分解 37
原根 38
公式 38
中国剩余定理 38
伯努利数和等幂求和 39

图论 **40**

LCA 40
最短路 40
网络流 41
树上路径交 44
树上点分治 44
树链剖分 46
二分图匹配 51
虚树 52

计算几何 **53**

圆的反演 53

二维 53

字符串 **56**

后缀自动机 56

回文自动机 58

杂项 **58**

STL 58

伪随机数 59

日期 59

子集枚举 60

权值最大上升子序列 60

数位 DP 60

心态崩了 61

一切的开始

宏定义

- 需要 C++11

```
#include <bits/stdc++.h>
using namespace std;
using LL = long long;
#define FOR(i, x, y) for (decay<decltype(y)>::type i = (x), _##i = (y); i < _##i; ++i)
#define FORD(i, x, y) for (decay<decltype(x)>::type i = (x), _##i = (y); i > _##i; --i)
#ifdef zerol
#define dbg(args...) do { cout << "\033[32;1m" << #args << " -> "; err(args); } while (0)
#else
#define dbg(...)
#endif
void err() { cout << "\033[39;0m" << endl; }
template<template<typename...> class T, typename t, typename... Args>
void err(T<t> a, Args... args) { for (auto x: a) cout << x << ' '; err(args...); }
template<typename T, typename... Args>
void err(T a, Args... args) { cout << a << ' '; err(args...); }
// ------------------------------------------------------------------------------
```

- POJ/BZOJ version

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cmath>
#include <string>
#include <vector>
#include <set>
#include <queue>
#include <cstring>
#include <cassert>
using namespace std;
typedef long long LL;
#define FOR(i, x, y) for (LL i = (x), _##i = (y); i < _##i; ++i)
#define FORD(i, x, y) for (LL i = (x), _##i = (y); i > _##i; --i)
#ifdef zerol
#define dbg(args...) do { cout << "\033[32;1m" << #args<< " -> "; err(args); } while (0)
#else
#define dbg(...)
#endif
void err() { cout << "\033[39;0m" << endl; }
template<typename T, typename... Args>
void err(T a, Args... args) {
    cout << a << ' '; err(args...);
}
// ------------------------------------------------------------------------------
```

快速读

```
inline char next_char() {
    static char buf[100000], *p1 = buf, *p2 = buf;
    return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 100000, stdin), p1 == p2) ? EOF : *p1++;
}
inline bool maybe_digit(char c) {
    return c >= '0' && c <= '9';
}
template <typename T>
void rn(T& _v) {
    static char ch;
    static bool negative = false;
    _v = 0;
    while (!maybe_digit(ch)) {
        negative = ch == '-';
        ch = next_char();
    }
    do _v = (_v << 1) + (_v << 3) + ch - '0';
    while (maybe_digit(ch = next_char()));
    if (negative) _v = -_v;
}

template <typename T>
void o(T p) {
    static int stk[70], tp;
    if (p == 0) {
        putchar('0');
        return;
    }
    if (p < 0) { p = -p; putchar('-'); }
    while (p) stk[++tp] = p % 10, p /= 10;
    while (tp) putchar(stk[tp--] + '0');
}
```

- 需要初始化
- 需要一次读入
- 不支持负数

```
const int MAXS = 100 * 1024 * 1024;
char buf[MAXS];
template<typename T>
inline bool read(T& x) {
    static char* p = buf;
    x = 0;
    while (*p && !isdigit(*p)) ++p;
    if (!*p) return false;
    while (isdigit(*p)) x = x * 10 + *p++ - 48;
    return true;
}

fread(buf, 1, MAXS, stdin);
```

对拍

```
#!/usr/bin/env bash
g++ -o r main.cpp -O2 -std=c++11
g++ -o std std.cpp -O2 -std=c++11
while true; do
    python gen.py > in
    ./std < in > stdout
    ./r < in > out
    if test $? -ne 0; then
        exit 0
    fi
    if diff stdout out; then
        printf "AC\n"
    else
        printf "GG\n"
        exit 0
    fi
done
```

数据结构

ST 表

- 二维

```
int f[maxn][maxn][10][10];
inline int highbit(int x) { return 31 - __builtin_clz(x); }
inline int calc(int x, int y, int xx, int yy, int p, int q) {
    return max(
        max(f[x][y][p][q], f[xx - (1 << p) + 1][yy - (1 << q) + 1][p][q]),
        max(f[xx - (1 << p) + 1][y][p][q], f[x][yy - (1 << q) + 1][p][q])
    );
}
void init() {
    FOR (x, 0, highbit(n) + 1)
    FOR (y, 0, highbit(m) + 1)
        FOR (i, 0, n - (1 << x) + 1)
        FOR (j, 0, m - (1 << y) + 1) {
            if (!x && !y) { f[i][j][x][y] = a[i][j]; continue; }
            f[i][j][x][y] = calc(
                i, j,
                i + (1 << x) - 1, j + (1 << y) - 1,
                max(x - 1, 0), max(y - 1, 0)
            );
        }
}
inline int get_max(int x, int y, int xx, int yy) {
    return calc(x, y, xx, yy, highbit(xx - x + 1), highbit(yy - y + 1));
}
```

- 一维

```
struct RMQ {
    int f[maxn][20];
    inline int highbit(int x) { return 31 - __builtin_clz(x); }
    void init(int* v, int n) {
        FOR (i, 0, n) f[i][0] = v[i];
        FOR (x, 1, highbit(n) + 1)
            FOR (i, 0, n - (1 << x) + 1)
                f[i][x] = min(f[i][x - 1], f[i + (1 << (x - 1))][x - 1]);
    }
    int get_min(int l, int r) {
        assert(l <= r);
        int t = highbit(r - l + 1);
        return min(f[l][t], f[r - (1 << t) + 1][t]);
    }
} rmq;
```

线段树

- 普适

```
namespace sg {
    struct Q {
        LL setv;
        explicit Q(LL setv = -1): setv(setv) {}
        void operator += (const Q& q) { if (q.setv != -1) setv = q.setv; }
    };
    struct P {
        LL min;
        explicit P(LL min = INF): min(min) {}
        void up(Q& q) { if (q.setv != -1) min = q.setv; }
    };
    template<typename T>
    P operator & (T&& a, T&& b) {
        return P(min(a.min, b.min));
    }
    P p[maxn << 2];
    Q q[maxn << 2];
#define lson o * 2, l, (l + r) / 2
#define rson o * 2 + 1, (l + r) / 2 + 1, r
    void up(int o, int l, int r) {
        if (l == r) p[o] = P();
        else p[o] = p[o * 2] & p[o * 2 + 1];
        p[o].up(q[o]);
    }
    void down(int o, int l, int r) {
        q[o * 2] += q[o]; q[o * 2 + 1] += q[o];
        q[o] = Q();
        up(lson); up(rson);
    }
    template<typename T>
    void build(T&& f, int o = 1, int l = 1, int r = n) {
        if (l == r) q[o] = f(l);
        else { build(f, lson); build(f, rson); q[o] = Q(); }
        up(o, l, r);
    }
    P query(int ql, int qr, int o = 1, int l = 1, int r = n) {
        if (ql > r || l > qr) return P();
        if (ql <= l && r <= qr) return p[o];
        down(o, l, r);
        return query(ql, qr, lson) & query(ql, qr, rson);
    }
    void update(int ql, int qr, const Q& v, int o = 1, int l = 1, int r = n) {
        if (ql > r || l > qr) return;
        if (ql <= l && r <= qr) q[o] += v;
        else {
            down(o, l, r);
            update(ql, qr, v, lson); update(ql, qr, v, rson);
        }
        up(o, l, r);
    }
}
```

- ADD

```
struct IntervalTree {
#define lson o * 2, l, m
#define rson o * 2 + 1, m + 1, r
    int sum[maxnode], add[maxnode];
    void init() { memset(sum, 0, sizeof sum); memset(add, 0, sizeof add); }
    void maintain(int o, int l, int r) {
        if (l < r) {
            int lc = o * 2, rc = o * 2 + 1;
            sum[o] = sum[lc] + sum[rc];
        } else sum[o] = 0;
        sum[o] += add[o] * (r - l + 1);
    }
    void build(int o, int l, int r) {
        if (l > r) return;
        if (l == r) add[o] = a[l];
        else {
            int m = (l + r) / 2;
            build(lson); build(rson);
        }
        maintain(o, l, r);
    }
    void update(int p, int q, int o, int l, int r, int v) {
        if (p > r || l > q) return;
        if (p <= l && r <= q) add[o] += v;
        else {
            int m = (l + r) / 2;
            update(p, q, lson, v); update(p, q, rson, v);
        }
        maintain(o, l, r);
    }
    LL query(int p, int q, int o, int l, int r, LL addv = 0) {
        if (p > r || l > q) return 0;
        if (p <= l && r <= q) return sum[o] + addv * (r - l + 1);
        int m = (l + r) / 2;
        return query(p, q, lson, addv + add[o]) +
               query(p, q, rson, addv + add[o]);
    }
} IT;
```

- SET

```
struct IntervalTree {
#define lson o * 2, l, m
#define rson o * 2 + 1, m + 1, r
    int setv[maxnode], sumv[maxnode];
    void init() { memset(setv, -1, sizeof setv); memset(sumv, 0, sizeof sumv); }
    void maintain(int o, int l, int r) {
        if (l < r) {
            int lc = o * 2, rc = o * 2 + 1;
            sumv[o] = sumv[lc] + sumv[rc];
        }
        if (setv[o] >= 0) sumv[o] = (r - l + 1) * setv[o];
    }
    void pushdown(int o) {
        if (setv[o] >= 0) {
            int lc = o * 2, rc = o * 2 + 1;
            setv[lc] = setv[rc] = setv[o];
            setv[o] = -1;
        }
    }
    void update(int p, int q, int o, int l, int r, int v) {
        if (p <= r && l <= q)
        if (p <= l && r <= q) setv[o] = v;
        else {
            pushdown(o);
            int m = (l + r) / 2;
            update(p, q, lson, v); update(p, q, rson, v);
        }
        maintain(o, l, r);
    }
    int query(int p, int q, int o, int l, int r) {
        if (p > r || l > q) return 0;
        if (setv[o] >= 0) return setv[o] * (min(r, q) - max(l, p) + 1);
        if (p <= l && r <= q) return sumv[o];
        int m = (l + r) / 2;
        return query(p, q, lson) + query(p, q, rson);
    }
} IT;
```

- SET + ADD

```
struct IntervalTree {
#define ls o * 2, l, m
#define rs o * 2 + 1, m + 1, r
    static const LL M = maxn * 4, RS = 1E18 - 1;
    LL addv[M], setv[M], minv[M], maxv[M], sumv[M];
    void init() {
        memset(addv, 0, sizeof addv);
        fill(setv, setv + M, RS);
        memset(minv, 0, sizeof minv);
        memset(maxv, 0, sizeof maxv);
        memset(sumv, 0, sizeof sumv);
    }
    void maintain(LL o, LL l, LL r) {
        if (l < r) {
            LL lc = o * 2, rc = o * 2 + 1;
            sumv[o] = sumv[lc] + sumv[rc];
            minv[o] = min(minv[lc], minv[rc]);
            maxv[o] = max(maxv[lc], maxv[rc]);
        } else sumv[o] = minv[o] = maxv[o] = 0;
        if (setv[o] != RS) { minv[o] = maxv[o] = setv[o]; sumv[o] = setv[o] * (r - l + 1); }
        if (addv[o]) { minv[o] += addv[o]; maxv[o] += addv[o]; sumv[o] += addv[o] * (r - l + 1); }
    }
    void build(LL o, LL l, LL r) {
        if (l == r) addv[o] = a[l];
        else {
            LL m = (l + r) / 2;
            build(ls); build(rs);
        }
        maintain(o, l, r);
    }
    void pushdown(LL o) {
        LL lc = o * 2, rc = o * 2 + 1;
        if (setv[o] != RS) {
            setv[lc] = setv[rc] = setv[o];
            addv[lc] = addv[rc] = 0;
            setv[o] = RS;
        }
        if (addv[o]) {
            addv[lc] += addv[o]; addv[rc] += addv[o];
            addv[o] = 0;
        }
    }
    void update(LL p, LL q, LL o, LL l, LL r, LL v, LL op) {
        if (p <= r && l <= q)
        if (p <= l && r <= q) {
            if (op == 2) { setv[o] = v; addv[o] = 0; }
            else addv[o] += v;
        } else {
            pushdown(o);
            LL m = (l + r) / 2;
            update(p, q, ls, v, op); update(p, q, rs, v, op);
        }
        maintain(o, l, r);
    }
    void query(LL p, LL q, LL o, LL l, LL r, LL add, LL& ssum, LL& smin, LL& smax) {
        if (p > r || l > q) return;
        if (setv[o] != RS) {
            LL v = setv[o] + add + addv[o];
            ssum += v * (min(r, q) - max(l, p) + 1);
            smin = min(smin, v);
            smax = max(smax, v);
        } else if (p <= l && r <= q) {
            ssum += sumv[o] + add * (r - l + 1);
            smin = min(smin, minv[o] + add);
            smax = max(smax, maxv[o] + add);
        } else {
            LL m = (l + r) / 2;
            query(p, q, ls, add + addv[o], ssum, smin, smax);
            query(p, q, rs, add + addv[o], ssum, smin, smax);
        }
    }
} IT;
```

均摊复杂度线段树

- 区间取 max，区间求和。

```
namespace R {
#define lson o * 2, l, (l + r) / 2
#define rson o * 2 + 1, (l + r) / 2 + 1, r
    int m1[N], m2[N], cm1[N];
    LL sum[N];
    void up(int o) {
        int lc = o * 2, rc = lc + 1;
        m1[o] = max(m1[lc], m1[rc]);
        sum[o] = sum[lc] + sum[rc];
        if (m1[lc] == m1[rc]) {
            cm1[o] = cm1[lc] + cm1[rc];
            m2[o] = max(m2[lc], m2[rc]);
        } else {
            cm1[o] = m1[lc] > m1[rc] ? cm1[lc] : cm1[rc];
            m2[o] = max(min(m1[lc], m1[rc]), max(m2[lc], m2[rc]));
        }
    }
    void mod(int o, int x) {
        if (x >= m1[o]) return;
        assert(x > m2[o]);
        sum[o] -= 1LL * (m1[o] - x) * cm1[o];
        m1[o] = x;
    }
    void down(int o) {
        int lc = o * 2, rc = lc + 1;
        mod(lc, m1[o]); mod(rc, m1[o]);
    }
    void build(int o, int l, int r) {
        if (l == r) { int t; read(t); sum[o] = m1[o] = t; m2[o] = -1; cm1[o] = 1; }
        else { build(lson); build(rson); up(o); }
    }
    void update(int ql, int qr, int x, int o, int l, int r) {
        if (r < ql || qr < l || m1[o] <= x) return;
        if (ql <= l && r <= qr && m2[o] < x) { mod(o, x); return; }
        down(o);
        update(ql, qr, x, lson); update(ql, qr, x, rson);
        up(o);
    }
    int qmax(int ql, int qr, int o, int l, int r) {
        if (r < ql || qr < l) return -INF;
        if (ql <= l && r <= qr) return m1[o];
        down(o);
        return max(qmax(ql, qr, lson), qmax(ql, qr, rson));
    }
    LL qsum(int ql, int qr, int o, int l, int r) {
        if (r < ql || qr < l) return 0;
        if (ql <= l && r <= qr) return sum[o];
        down(o);
        return qsum(ql, qr, lson) + qsum(ql, qr, rson);
    }
}
```

持久化线段树

- ADD

```
namespace tree {
#define mid ((l + r) >> 1)
#define lson ql, qr, l, mid
#define rson ql, qr, mid + 1, r
    struct P {
        LL add, sum;
        int ls, rs;
    } tr[maxn * 45 * 2];
    int sz = 1;
    int N(LL add, int l, int r, int ls, int rs) {
        tr[sz] = {add, tr[ls].sum + tr[rs].sum + add * (len[r] - len[l - 1]), ls, rs};
        return sz++;
    }
    int update(int o, int ql, int qr, int l, int r, LL add) {
        if (ql > r || l > qr) return o;
        const P& t = tr[o];
        if (ql <= l && r <= qr) return N(add + t.add, l, r, t.ls, t.rs);
        return N(t.add, l, r, update(t.ls, lson, add), update(t.rs, rson, add));
    }
    LL query(int o, int ql, int qr, int l, int r, LL add = 0) {
        if (ql > r || l > qr) return 0;
        const P& t = tr[o];
        if (ql <= l && r <= qr) return add * (len[r] - len[l - 1]) + t.sum;
        return query(t.ls, lson, add + t.add) + query(t.rs, rson, add + t.add);
    }
}
```

K-D Tree

- 维护信息
- 带重构（适合在线）
- 插入时左右儿子要标记为 null。

```
namespace kd {
    const int K = 2, inf = 1E9, M = N;
    const double lim = 0.7;
    struct P {
        int d[K], l[K], r[K], sz, val;
        LL sum;
        P *ls, *rs;
        P* up() {
            sz = ls->sz + rs->sz + 1;
            sum = ls->sum + rs->sum + val;
            FOR (i, 0, K) {
                l[i] = min(d[i], min(ls->l[i], rs->l[i]));
                r[i] = max(d[i], max(ls->r[i], rs->r[i]));
            }
            return this;
        }
    } pool[M], *null = new P, *pit = pool;
    static P *tmp[M], **pt;
    void init() {
        null->ls = null->rs = null;
        FOR (i, 0, K) null->l[i] = inf, null->r[i] = -inf;
        null->sum = null->val = 0;
        null->sz = 0;
    }

    P* build(P** l, P** r, int d = 0) { // [l, r)
        if (d == K) d = 0;
        if (l >= r) return null;
        P** m = l + (r - l) / 2; assert(l <= m && m < r);
        nth_element(l, m, r, [&](const P* a, const P* b){
            return a->d[d] < b->d[d];
        });
        P* o = *m;
        o->ls = build(l, m, d + 1); o->rs = build(m + 1, r, d + 1);
        return o->up();
    }
    P* Build() {
        pt = tmp; FOR (it, pool, pit) *pt++ = it;
        return build(tmp, pt);
    }
    inline bool inside(int p[], int q[], int l[], int r[]) {
        FOR (i, 0, K) if (r[i] < q[i] || p[i] < l[i]) return false;
        return true;
    }
    LL query(P* o, int l[], int r[]) {
        if (o == null) return 0;
        FOR (i, 0, K) if (o->r[i] < l[i] || r[i] < o->l[i]) return 0;
        if (inside(o->l, o->r, l, r)) return o->sum;
        return query(o->ls, l, r) + query(o->rs, l, r) +
               (inside(o->d, o->d, l, r) ? o->val : 0);
    }
    void dfs(P* o) {
        if (o == null) return;
        *pt++ = o; dfs(o->ls); dfs(o->rs);
    }
    P* ins(P* o, P* x, int d = 0) {
        if (d == K) d = 0;
        if (o == null) return x->up();
        P*& oo = x->d[d] <= o->d[d] ? o->ls : o->rs;
        if (oo->sz > o->sz * lim) {
            pt = tmp; dfs(o); *pt++ = x;
            return build(tmp, pt, d);
        }
        oo = ins(oo, x, d + 1);
        return o->up();
    }
}
```

- 维护信息
- 带修改（适合离线）

```
namespace kd {
    const int K = 3, inf = 1E9, M = N << 3;
    extern struct P* null;
    struct P {
        int d[K], l[K], r[K], val;
        int Max;
        P *ls, *rs, *fa;
        P* up() {
            Max = max(val, max(ls->Max, rs->Max));
            FOR (i, 0, K) {
                l[i] = min(d[i], min(ls->l[i], rs->l[i]));
                r[i] = max(d[i], max(ls->r[i], rs->r[i]));
            }
            return ls->fa = rs->fa = this;
        }
    } pool[M], *null = new P, *pit = pool;
    void upd(P* o, int val) {
        o->val = val;
        for (; o != null; o = o->fa)
            o->Max = max(o->Max, val);
    }
    static P *tmp[M], **pt;
    void init() {
        null->ls = null->rs = null;
        FOR (i, 0, K) null->l[i] = inf, null->r[i] = -inf;
        null->Max = null->val = 0;
    }
    P* build(P** l, P** r, int d = 0) { // [l, r)
        if (d == K) d = 0;
        if (l >= r) return null;
        P** m = l + (r - l) / 2; assert(l <= m && m < r);
        nth_element(l, m, r, [&](const P* a, const P* b){
            return a->d[d] < b->d[d];
        });
        P* o = *m;
        o->ls = build(l, m, d + 1); o->rs = build(m + 1, r, d + 1);
        return o->up();
    }
    P* Build() {
        pt = tmp; FOR (it, pool, pit) *pt++ = it;
        P* ret = build(tmp, pt); ret->fa = null;
        return ret;
    }
    inline bool inside(int p[], int q[], int l[], int r[]) {
        FOR (i, 0, K) if (r[i] < q[i] || p[i] < l[i]) return false;
        return true;
    }
    int query(P* o, int l[], int r[]) {
        if (o == null) return 0;
        FOR (i, 0, K) if (o->r[i] < l[i] || r[i] < o->l[i]) return 0;
        if (inside(o->l, o->r, l, r)) return o->Max;
        int ret = 0;
        if (o->val > ret && inside(o->d, o->d, l, r)) ret = max(ret, o->val);
        if (o->ls->Max > ret) ret = max(ret, query(o->ls, l, r));
        if (o->rs->Max > ret) ret = max(ret, query(o->rs, l, r));
        return ret;
    }
}
```

- 最近点对
- 要用全局变量大力剪枝

```
namespace kd {
    const int K = 3;
    const int M = N;
    const int inf = 1E9 + 100;
    struct P {
        int d[K];
        int l[K], r[K];
        P *ls, *rs;
        P* up() {
            FOR (i, 0, K) {
                l[i] = min(d[i], min(ls->l[i], rs->l[i]));
                r[i] = max(d[i], max(ls->r[i], rs->r[i]));
            }
            return this;
        }
    } pool[M], *null = new P, *pit = pool;
    static P *tmp[M], **pt;
    void init() {
        null->ls = null->rs = null;
        FOR (i, 0, K) null->l[i] = inf, null->r[i] = -inf;
    }
    P* build(P** l, P** r, int d = 0) { // [l, r)
        if (d == K) d = 0;
        if (l >= r) return null;
        P** m = l + (r - l) / 2;
        nth_element(l, m, r, [&](const P* a, const P* b){
            return a->d[d] < b->d[d];
        });
        P* o = *m;
        o->ls = build(l, m, d + 1); o->rs = build(m + 1, r, d + 1);
        return o->up();
    }
    LL eval(P* o, int d[]) {
        // ...
    }
    LL dist(int d1[], int d2[]) {
        // ...
    }
    LL S;
    LL query(P* o, int d[]) {
        if (o == null) return 0;
        S = max(S, dist(o->d, d));
        LL mdl = eval(o->ls, d), mdr = eval(o->rs, d);
        if (mdl < mdr) {
            if (S > mdl) S = max(S, query(o->ls, d));
            if (S > mdr) S = max(S, query(o->rs, d));
        } else {
            if (S > mdr) S = max(S, query(o->rs, d));
            if (S > mdl) S = max(S, query(o->ls, d));
        }
        return S;
    }
    P* Build() {
        pt = tmp; FOR (it, pool, pit) *pt++ = it;
        return build(tmp, pt);
    }
}
```

树状数组

- 注意：0 是无效下标

```
inline LL lowbit(LL x) { return x & -x; }
void add(LL x, LL v) {
    for (; x < N; x += lowbit(x))
        c[x] += v;
}
LL sum(LL x) {
    LL ret = 0;
    for (; x > 0; x -= lowbit(x))
        ret += c[x];
    return ret;
}
int kth(LL k) {
    int ret = 0;
    LL cnt = 0;
    FORD (i, 20, -1) {
        ret += 1 << i;
        if (ret >= M || cnt + c[ret] >= k)
            ret -= 1 << i;
        else cnt += c[ret];
    }
    return ret + 1;
}
```

- 区间修改 & 区间查询

```cpp
namespace bit {
    int c[maxn], cc[maxn];
    inline int lowbit(int x) { return x & -x; }
    void add(int x, int v) {
        for (int i = x; i <= n; i += lowbit(i)) {
            c[i] += v; cc[i] += x * v;
        }
    }
    void add(int l, int r, int v) { add(l, v); add(r + 1, -v); }
    int sum(int x) {
        int ret = 0;
        for (int i = x; i > 0; i -= lowbit(i))
            ret += (x + 1) * c[i] - cc[i];
        return ret;
    }
    int sum(int l, int r) { return sum(r) - sum(l - 1); }
}
```

- 三维

```cpp
inline int lowbit(int x) { return x & -x; }
void update(int x, int y, int z, int d) {
    for (int i = x; i <= n; i += lowbit(i))
        for (int j = y; j <= n; j += lowbit(j))
            for (int k = z; k <= n; k += lowbit(k))
                c[i][j][k] += d;
}
LL query(int x, int y, int z) {
    LL ret = 0;
    for (int i = x; i > 0; i -= lowbit(i))
        for (int j = y; j > 0; j -= lowbit(j))
            for (int k = z; k > 0; k -= lowbit(k))
                ret += c[i][j][k];
    return ret;
}
LL solve(int x, int y, int z, int xx, int yy, int zz) {
    return    query(xx, yy, zz)
            - query(xx, yy, z - 1)
            - query(xx, y - 1, zz)
            - query(x - 1, yy, zz)
            + query(xx, y - 1, z - 1)
            + query(x - 1, yy, z - 1)
            + query(x - 1, y - 1, zz)
            - query(x - 1, y - 1, z - 1);
```

主席树

- 正常主席树

```cpp
namespace tree {
#define mid ((l + r) >> 1)
#define lson l, mid
#define rson mid + 1, r
    const int MAGIC = M * 30;
    struct P {
        int sum, ls, rs;
    } tr[MAGIC] = {{0, 0, 0}};
    int sz;
    int N(int sum, int ls, int rs) {
        if (sz == MAGIC) while(1);
        tr[sz] = {sum, ls, rs};
        return sz++;
    }
    int ins(int o, int x, int v, int l = 1, int r = ls) {
        if (x < l || x > r) return o;
        const P& t = tr[o];
        if (l == r) return N(t.sum + v, 0, 0);
        return N(t.sum + v, ins(t.ls, x, v, lson), ins(t.rs, x, v, rson));
    }
    int query(int o, int ql, int qr, int l = 1, int r = ls) {
        if (ql > r || l > qr) return 0;
        const P& t = tr[o];
        if (ql <= l && r <= qr) return t.sum;
        return query(t.ls, ql, qr, lson) + query(t.rs, ql, qr, rson);
    }
}
```

- 第k大

```
struct TREE {
#define mid ((l + r) >> 1)
#define lson l, mid
#define rson mid + 1, r
    struct P {
        int w, ls, rs;
    } tr[maxn * 20];
    int sz = 1;
    TREE() { tr[0] = {0, 0, 0}; }
    int N(int w, int ls, int rs) {
        tr[sz] = {w, ls, rs};
        return sz++;
    }
    int ins(int tt, int l, int r, int x) {
        if (x < l || r < x) return tt;
        const P& t = tr[tt];
        if (l == r) return N(t.w + 1, 0, 0);
        return N(t.w + 1, ins(t.ls, lson, x), ins(t.rs, rson, x));
    }
    int query(int pp, int qq, int l, int r, int k) {
        if (l == r) return l;
        const P &p = tr[pp], &q = tr[qq];
        int w = tr[q.ls].w - tr[p.ls].w;
        if (k <= w) return query(p.ls, q.ls, lson, k);
        else return query(p.rs, q.rs, rson, k - w);
    }
} tree;
```

- 树状数组套主席树

```
typedef vector<int> VI;
struct TREE {
#define mid ((l + r) >> 1)
#define lson l, mid
#define rson mid + 1, r
    struct P {
        int w, ls, rs;
    } tr[maxn * 20 * 20];
    int sz = 1;
    TREE() { tr[0] = {0, 0, 0}; }
    int N(int w, int ls, int rs) {
        tr[sz] = {w, ls, rs};
        return sz++;
    }
    int add(int tt, int l, int r, int x, int d) {
        if (x < l || r < x) return tt;
        const P& t = tr[tt];
        if (l == r) return N(t.w + d, 0, 0);
        return N(t.w + d, add(t.ls, lson, x, d), add(t.rs, rson, x, d));
    }
    int ls_sum(const VI& rt) {
        int ret = 0;
        FOR (i, 0, rt.size())
            ret += tr[tr[rt[i]].ls].w;
        return ret;
    }
    inline void ls(VI& rt) { transform(rt.begin(), rt.end(), rt.begin(), [&](int x)->int{ return tr[x].ls; }); }
    inline void rs(VI& rt) { transform(rt.begin(), rt.end(), rt.begin(), [&](int x)->int{ return tr[x].rs; }); }
    int query(VI& p, VI& q, int l, int r, int k) {
        if (l == r) return l;
        int w = ls_sum(q) - ls_sum(p);
        if (k <= w) {
            ls(p); ls(q);
            return query(p, q, lson, k);
        }
        else {
            rs(p); rs(q);
            return query(p, q, rson, k - w);
        }
    }
} tree;
struct BIT {
    int root[maxn];
    void init() { memset(root, 0, sizeof root); }
    inline int lowbit(int x) { return x & -x; }
    void update(int p, int x, int d) {
        for (int i = p; i <= m; i += lowbit(i))
            root[i] = tree.add(root[i], 1, m, x, d);
    }
    int query(int l, int r, int k) {
        VI p, q;
        for (int i = l - 1; i > 0; i -= lowbit(i)) p.push_back(root[i]);
        for (int i = r; i > 0; i -= lowbit(i)) q.push_back(root[i]);
        return tree.query(p, q, 1, m, k);
    }
} bit;

void init() {
    m = 10000;
    tree.sz = 1;
    bit.init();
    FOR (i, 1, m + 1)
        bit.update(i, a[i], 1);
}
```

左偏树

```
namespace LTree {
    extern struct P* null, *pit;
    queue<P*> trash;
    const int M = 1E5 + 100;
    struct P {
        P *ls, *rs;
        LL v;
        int d;
        void operator delete (void* ptr) {
            trash.push((P*)ptr);
        }
        void* operator new(size_t size) {
            if (trash.empty()) return pit++;
            void* ret = trash.front(); trash.pop(); return ret;
        }

        void prt() {
            if (this == null) return;
            cout << v << ' ';
            ls->prt(); rs->prt();
        }
    } pool[M], *pit = pool, *null = new P{0, 0, -1, -1};
    P* N(LL v) {
        return new P{null, null, v, 0};
    }
    P* merge(P* a, P* b) {
        if (a == null) return b;
        if (b == null) return a;
        if (a->v > b->v) swap(a, b);
        a->rs = merge(a->rs, b);
        if (a->ls->d < a->rs->d) swap(a->ls, a->rs);
        a->d = a->rs->d + 1;
        return a;
    }

    LL pop(P*& o) {
        LL ret = o->v;
        P* t = o;
        o = merge(o->ls, o->rs);
        delete t;
        return ret;
    }
}
```

可持久化

```
namespace LTree {
    extern struct P* null, *pit;
    queue<P*> trash;
    const int M = 1E6 + 100;
    struct P {
        P *ls, *rs;
        LL v;
        int d;
        void operator delete (void* ptr) {
            trash.push((P*)ptr);
        }
        void* operator new(size_t size) {
            if (trash.empty()) return pit++;
            void* ret = trash.front(); trash.pop(); return ret;
        }
    } pool[M], *pit = pool, *null = new P{0, 0, -1, -1};
    P* N(LL v, P* ls = null, P* rs = null) {
        if (ls->d < rs->d) swap(ls, rs);
        return new P{ls, rs, v, rs->d + 1};
    }
    P* merge(P* a, P* b) {
        if (a == null) return b;
        if (b == null) return a;
        if (a->v < b->v)
            return N(a->v, a->ls, merge(a->rs, b));
        else
            return N(b->v, b->ls, merge(b->rs, a));
    }

    LL pop(P*& o) {
        LL ret = o->v;
        o = merge(o->ls, o->rs);
        return ret;
    }
}
```

Treap

- 非旋 Treap
- v 小根堆
- 模板题 bzoj 3224
- lower 第一个大于等于的是第几个 (0-based)
- upper 第一个大于的是第几个 (0-based)
- split 左侧分割出 rk 个元素
- 树套树略

```
namespace treap {
    const int M = maxn * 17;
    extern struct P* const null;
    struct P {
        P *ls, *rs;
        int v, sz;
        unsigned rd;
        P(int v): ls(null), rs(null), v(v), sz(1), rd(rnd()) {}
        P(): sz(0) {}

        P* up() { sz = ls->sz + rs->sz + 1; return this; }
        int lower(int v) {
            if (this == null) return 0;
            return this->v >= v ? ls->lower(v) : rs->lower(v) + ls->sz + 1;
        }
        int upper(int v) {
            if (this == null) return 0;
            return this->v > v ? ls->upper(v) : rs->upper(v) + ls->sz + 1;
        }
    } *const null = new P, pool[M], *pit = pool;

    P* merge(P* l, P* r) {
        if (l == null) return r; if (r == null) return l;
        if (l->rd < r->rd) { l->rs = merge(l->rs, r); return l->up(); }
        else { r->ls = merge(l, r->ls); return r->up(); }
    }

    void split(P* o, int rk, P*& l, P*& r) {
        if (o == null) { l = r = null; return; }
        if (o->ls->sz >= rk) { split(o->ls, rk, l, o->ls); r = o->up(); }
        else { split(o->rs, rk - o->ls->sz - 1, o->rs, r); l = o->up(); }
    }
}
```

- 持久化 Treap

```
namespace treap {
    const int M = maxn * 17 * 12;
    extern struct P* const null, *pit;
    struct P {
        P *ls, *rs;
        int v, sz;
        LL sum;
        P(P* ls, P* rs, int v): ls(ls), rs(rs), v(v), sz(ls->sz + rs->sz + 1),
                                                    sum(ls->sum + rs->sum + v) {}
        P() {}

        void* operator new(size_t _) { return pit++; }
        template<typename T>
        int rk(int v, T&& cmp) {
            if (this == null) return 0;
            return cmp(this->v, v) ? ls->rk(v, cmp) : rs->rk(v, cmp) + ls->sz + 1;
        }
        int lower(int v) { return rk(v, greater_equal<int>()); }
        int upper(int v) { return rk(v, greater<int>()); }
    } pool[M], *pit = pool, *const null = new P;
    P* merge(P* l, P* r) {
        if (l == null) return r; if (r == null) return l;
        if (rnd() % (l->sz + r->sz) < l->sz) return new P{l->ls, merge(l->rs, r), l->v};
        else return new P{merge(l, r->ls), r->rs, r->v};
    }
    void split(P* o, int rk, P*& l, P*& r) {
        if (o == null) { l = r = null; return; }
        if (o->ls->sz >= rk) { split(o->ls, rk, l, r); r = new P{r, o->rs, o->v}; }
        else { split(o->rs, rk - o->ls->sz - 1, l, r); l = new P{o->ls, l, o->v}; }
    }
}
```

Treap-序列

- 区间 ADD，SUM

```
namespace treap {
    const int M = 8E5 + 100;
    extern struct P*const null;
    struct P {
        P *ls, *rs;
        int sz, val, add, sum;
        P(int v, P* ls = null, P* rs = null): ls(ls), rs(rs), sz(1), val(v), add(0), sum(v) {}
        P(): sz(0), val(0), add(0), sum(0) {}

        P* up() {
            assert(this != null);
            sz = ls->sz + rs->sz + 1;
            sum = ls->sum + rs->sum + val + add * sz;
            return this;
        }
        void upd(int v) {
            if (this == null) return;
            add += v;
            sum += sz * v;
        }
        P* down() {
            if (add) {
                ls->upd(add); rs->upd(add);
                val += add;
                add = 0;
            }
            return this;
        }

        P* select(int rk) {
            if (rk == ls->sz + 1) return this;
            return ls->sz >= rk ? ls->select(rk) : rs->select(rk - ls->sz - 1);
        }
    } pool[M], *pit = pool, *const null = new P, *rt = null;

    P* merge(P* a, P* b) {
        if (a == null) return b->up();
        if (b == null) return a->up();
        if (rand() % (a->sz + b->sz) < a->sz) {
            a->down()->rs = merge(a->rs, b);
            return a->up();
        } else {
            b->down()->ls = merge(a, b->ls);
            return b->up();
        }
    }

    void split(P* o, int rk, P*& l, P*& r) {
        if (o == null) { l = r = null; return; }
        o->down();
        if (o->ls->sz >= rk) {
            split(o->ls, rk, l, o->ls);
            r = o->up();
        } else {
            split(o->rs, rk - o->ls->sz - 1, o->rs, r);
            l = o->up();
        }
    }

    inline void insert(int k, int v) {
        P *l, *r;
        split(rt, k - 1, l, r);
        rt = merge(merge(l, new (pit++) P(v)), r);
    }

    inline void erase(int k) {
        P *l, *r, *_, *t;
        split(rt, k - 1, l, t);
        split(t, 1, _, r);
        rt = merge(l, r);
    }

    P* build(int l, int r, int* a) {
        if (l > r) return null;
        if (l == r) return new(pit++) P(a[l]);
        int m = (l + r) / 2;
        return (new(pit++) P(a[m], build(l, m - 1, a), build(m + 1, r, a)))->up();
    }
```

```
};
```

- 区间 REVERSE，ADD，MIN

```
namespace treap {
    extern struct P*const null;
    struct P {
        P *ls, *rs;
        int sz, v, add, m;
        bool flip;
        P(int v, P* ls = null, P* rs = null): ls(ls), rs(rs), sz(1), v(v), add(0), m(v), flip(0) {}
        P(): sz(0), v(INF), m(INF) {}

        void upd(int v) {
            if (this == null) return;
            add += v; m += v;
        }
        void rev() {
            if (this == null) return;
            swap(ls, rs);
            flip ^= 1;
        }
        P* up() {
            assert(this != null);
            sz = ls->sz + rs->sz + 1;
            m = min(min(ls->m, rs->m), v) + add;
            return this;
        }
        P* down() {
            if (add) {
                ls->upd(add); rs->upd(add);
                v += add;
                add = 0;
            }
            if (flip) {
                ls->rev(); rs->rev();
                flip = 0;
            }
            return this;
        }

        P* select(int k) {
            if (ls->sz + 1 == k) return this;
            if (ls->sz >= k) return ls->select(k);
            return rs->select(k - ls->sz - 1);
        }

    } pool[M], *const null = new P, *pit = pool, *rt = null;

    P* merge(P* a, P* b) {
        if (a == null) return b;
        if (b == null) return a;
        if (rnd() % (a->sz + b->sz) < a->sz) {
            a->down()->rs = merge(a->rs, b);
            return a->up();
        } else {
            b->down()->ls = merge(a, b->ls);
            return b->up();
        }
    }

    void split(P* o, int k, P*& l, P*& r) {
        if (o == null) { l = r = null; return; }
        o->down();
        if (o->ls->sz >= k) {
            split(o->ls, k, l, o->ls);
            r = o->up();
        } else {
            split(o->rs, k - o->ls->sz - 1, o->rs, r);
            l = o->up();
        }
    }

    P* build(int l, int r, int* v) {
        if (l > r) return null;
        int m = (l + r) >> 1;
        return (new (pit++) P(v[m], build(l, m - 1, v), build(m + 1, r, v)))->up();
    }

    void go(int x, int y, void f(P*&)) {
        P *l, *m, *r;
        split(rt, y, l, r);
        split(l, x - 1, l, m);
        f(m);
        rt = merge(merge(l, m), r);
    }
}
using namespace treap;
int a[maxn], n, x, y, Q, v, k, d;
char s[100];

int main() {
    cin >> n;
    FOR (i, 1, n + 1) scanf("%d", &a[i]);
    rt = build(1, n, a);
    cin >> Q;
    while (Q--) {
        scanf("%s", s);
        if (s[0] == 'A') {
            scanf("%d%d%d", &x, &y, &v);
            go(x, y, [](P*& o){ o->upd(v); });
        } else if (s[0] == 'R' && s[3] == 'E') {
            scanf("%d%d", &x, &y);
            go(x, y, [](P*& o){ o->rev(); });
        } else if (s[0] == 'R' && s[3] == 'O') {
            scanf("%d%d%d", &x, &y, &d);
            d %= y - x + 1;
            go(x, y, [](P*& o){
                P *l, *r;
                split(o, o->sz - d, l, r);
                o = merge(r, l);
            });
        } else if (s[0] == 'I') {
            scanf("%d%d", &k, &v);
            go(k + 1, k, [](P*& o){ o = new (pit++) P(v); });
        } else if (s[0] == 'D') {
            scanf("%d", &k);
            go(k, k, [](P*& o){ o = null; });
        } else if (s[0] == 'M') {
            scanf("%d%d", &x, &y);
            go(x, y, [](P*& o) {
                printf("%d\n", o->m);
            });
        }
    }
}
```

- 持久化

```
namespace treap {
    struct P;
    extern P*const null;
    P* N(P* ls, P* rs, LL v, bool fill);
    struct P {
        P *const ls, *const rs;
        const int sz, v;
        const LL sum;
        bool fill;
        int cnt;

        void split(int k, P*& l, P*& r) {
            if (this == null) { l = r = null; return; }
            if (ls->sz >= k) {
                ls->split(k, l, r);
                r = N(r, rs, v, fill);
            } else {
                rs->split(k - ls->sz - fill, l, r);
                l = N(ls, l, v, fill);
            }
        }


    } *const null = new P{0, 0, 0, 0, 0, 0, 1};

    P* N(P* ls, P* rs, LL v, bool fill) {
        ls->cnt++; rs->cnt++;
        return new P{ls, rs, ls->sz + rs->sz + fill, v, ls->sum + rs->sum + v, fill, 1};
    }

    P* merge(P* a, P* b) {
        if (a == null) return b;
        if (b == null) return a;
        if (rand() % (a->sz + b->sz) < a->sz)
            return N(a->ls, merge(a->rs, b), a->v, a->fill);
        else
            return N(merge(a, b->ls), b->rs, b->v, b->fill);
    }

    void go(P* o, int x, int y, P*& l, P*& m, P*& r) {
        o->split(y, l, r);
        l->split(x - 1, l, m);
    }
}
```

可回滚并查集

- 注意这个不是可持久化并查集
- 查找时不进行路径压缩
- 复杂度靠按秩合并解决

```
namespace uf {
    int fa[maxn], sz[maxn];
    int undo[maxn], top;
    void init() { memset(fa, -1, sizeof fa); memset(sz, 0, sizeof sz); top = 0; }
    int findset(int x) { while (fa[x] != -1) x = fa[x]; return x; }
    bool join(int x, int y) {
        x = findset(x); y = findset(y);
        if (x == y) return false;
        if (sz[x] > sz[y]) swap(x, y);
        undo[top++] = x;
        fa[x] = y;
        sz[y] += sz[x] + 1;
        return true;
    }
    inline int checkpoint() { return top; }
    void rewind(int t) {
        while (top > t) {
            int x = undo[--top];
            sz[fa[x]] -= sz[x] + 1;
            fa[x] = -1;
        }
    }
}
```

舞蹈链

- 注意 link 的 y 的范围是 [1, n]

- 注意在某些情况下替换掉 memset

- 精确覆盖

```
struct P {
    P *L, *R, *U, *D;
    int x, y;
};

const int INF = 1E9;

struct DLX {
#define TR(i, D, s) for (P* i = s->D; i != s; i = i->D)
    static const int M = 2E5;
    P pool[M], *h[M], *r[M], *pit;
    int sz[M];
    bool solved;
    stack<int> ans;
    void init(int n) {
        pit = pool;
        ++n;
        solved = false;
        while (!ans.empty()) ans.pop();
        memset(r, 0, sizeof r);
        memset(sz, 0, sizeof sz);
        FOR (i, 0, n)
            h[i] = new (pit++) P;
        FOR (i, 0, n) {
            h[i]->L = h[(i + n - 1) % n];
            h[i]->R = h[(i + 1) % n];
            h[i]->U = h[i]->D = h[i];
            h[i]->y = i;
        }
    }

    void link(int x, int y) {
        sz[y]++;
        auto p = new (pit++) P;
        p->x = x; p->y = y;
        p->U = h[y]->U; p->D = h[y];
        p->D->U = p->U->D = p;
        if (!r[x]) r[x] = p->L = p->R = p;
        else {
            p->L = r[x]; p->R = r[x]->R;
            p->L->R = p->R->L = p;
        }
    }

    void remove(P* p) {
        p->L->R = p->R; p->R->L = p->L;
        TR (i, D, p)
            TR (j, R, i) {
                j->D->U = j->U; j->U->D = j->D;
                sz[j->y]--;
            }
    }

    void recall(P* p) {
        p->L->R = p->R->L = p;
        TR (i, U, p)
            TR (j, L, i) {
                j->D->U = j->U->D = j;
                sz[j->y]++;
            }
    }

    bool dfs(int d) {
        if (solved) return true;
        if (h[0]->R == h[0]) return solved = true;
        int m = INF;
        P* c;
        TR (i, R, h[0])
            if (sz[i->y] < m) { m = sz[i->y]; c = i; }
        remove(c);
        TR (i, D, c) {
            ans.push(i->x);
            TR (j, R, i) remove(h[j->y]);
            if (dfs(d + 1)) return true;
            TR (j, L, i) recall(h[j->y]);
            ans.pop();
        }
        recall(c);
        return false;
    }
} dlx;
```

- 可重复覆盖

```
struct P {
    P *L, *R, *U, *D;
    int x, y;
};

const int INF = 1E9;

struct DLX {
#define TR(i, D, s) for (P* i = s->D; i != s; i = i->D)
    static const int M = 2E5;
    P pool[M], *h[M], *r[M], *pit;
    int sz[M], vis[M], ans, clk;
    void init(int n) {
        clk = 0;
        ans = INF;
        pit = pool;
        ++n;
        memset(r, 0, sizeof r);
        memset(sz, 0, sizeof sz);
        memset(vis, -1, sizeof vis);
        FOR (i, 0, n)
            h[i] = new (pit++) P;
        FOR (i, 0, n) {
            h[i]->L = h[(i + n - 1) % n];
            h[i]->R = h[(i + 1) % n];
            h[i]->U = h[i]->D = h[i];
            h[i]->y = i;
        }
    }

    void link(int x, int y) {
        sz[y]++;
        auto p = new (pit++) P;
        p->x = x; p->y = y;
        p->U = h[y]->U; p->D = h[y];
        p->D->U = p->U->D = p;
        if (!r[x]) r[x] = p->L = p->R = p;
        else {
            p->L = r[x]; p->R = r[x]->R;
            p->L->R = p->R->L = p;
        }
    }

    void remove(P* p) {
        TR (i, D, p) {
            i->L->R = i->R;
            i->R->L = i->L;
        }
    }

    void recall(P* p) {
        TR (i, U, p)
            i->L->R = i->R->L = i;
    }

    int eval() {
        ++clk;
        int ret = 0;
        TR (i, R, h[0])
```

```
            if (vis[i->y] != clk) {
                ++ret;
                vis[i->y] = clk;
                TR (j, D, i)
                    TR (k, R, j)
                        vis[k->y] = clk;
            }
        return ret;
    }

    void dfs(int d) {
        if (h[0]->R == h[0]) { ans = min(ans, d); return; }
        if (eval() + d >= ans) return;
        P* c;
        int m = INF;
        TR (i, R, h[0])
            if (sz[i->y] < m) { m = sz[i->y]; c = i; }
        TR (i, D, c) {
            remove(i);
            TR (j, R, i) remove(j);
            dfs(d + 1);
            TR (j, L, i) recall(j);
            recall(i);
        }
    }
} dlx;
```

CDQ 分治

```
const int maxn = 2E5 + 100;
struct P {
    int x, y;
    int* f;
    bool d1, d2;
} a[maxn], b[maxn], c[maxn];
int f[maxn];

void go2(int l, int r) {
    if (l + 1 == r) return;
    int m = (l + r) >> 1;
    go2(l, m); go2(m, r);
    FOR (i, l, m) b[i].d2 = 0;
    FOR (i, m, r) b[i].d2 = 1;
    merge(b + l, b + m, b + m, b + r, c + l, [](const P& a, const P& b)->bool {
            if (a.y != b.y) return a.y < b.y;
            return a.d2 > b.d2;
        });
    int mx = -1;
    FOR (i, l, r) {
        if (c[i].d1 && c[i].d2) *c[i].f = max(*c[i].f, mx + 1);
        if (!c[i].d1 && !c[i].d2) mx = max(mx, *c[i].f);
    }
    FOR (i, l, r) b[i] = c[i];
}

void go1(int l, int r) { // [l, r)
    if (l + 1 == r) return;
    int m = (l + r) >> 1;
    go1(l, m);
    FOR (i, l, m) a[i].d1 = 0;
    FOR (i, m, r) a[i].d1 = 1;
    copy(a + l, a + r, b + l);
    sort(b + l, b + r, [](const P& a, const P& b)->bool {
            if (a.x != b.x) return a.x < b.x;
            return a.d1 > b.d1;
        });
    go2(l, r);
    go1(m, r);
}
```

- k 维 LIS

```cpp
struct P {
    int v[K];
    LL f;
    bool d[K];
} o[N << 10];
P* a[K][N << 10];
int k;
void go(int now, int l, int r) {
    if (now == 0) {
        if (l + 1 == r) return;
        int m = (l + r) / 2;
        go(now, l, m);
        FOR (i, l, m) a[now][i]->d[now] = 0;
        FOR (i, m, r) a[now][i]->d[now] = 1;
        copy(a[now] + l, a[now] + r, a[now + 1] + l);
        sort(a[now + 1] + l, a[now + 1] + r, [now](const P* a, const P* b){
            if (a->v[now] != b->v[now]) return a->v[now] < b->v[now];
            return a->d[now] > b->d[now];
        });
        go(now + 1, l, r);
        go(now, m, r);
    } else {
        if (l + 1 == r) return;
        int m = (l + r) / 2;
        go(now, l, m); go(now, m, r);
        FOR (i, l, m) a[now][i]->d[now] = 0;
        FOR (i, m, r) a[now][i]->d[now] = 1;
        merge(a[now] + l, a[now] + m, a[now] + m, a[now] + r, a[now + 1] + l, [now](const P* a, const P* b){
            if (a->v[now] != b->v[now]) return a->v[now] < b->v[now];
            return a->d[now] > b->d[now];
        });
        copy(a[now + 1] + l, a[now + 1] + r, a[now] + l);
        if (now < k - 2) {
            go(now + 1, l, r);
        } else {
            LL sum = 0;
            dbg(l, r);
            FOR (i, l, r) {
                dbg(a[now][i]->v[0], a[now][i]->v[1], a[now][i]->f,
                                  a[now][i]->d[0], a[now][i]->d[1]);
                int cnt = 0;
                FOR (j, 0, now + 1) cnt += a[now][i]->d[j];
                dbg(cnt, now);
                if (cnt == 0) {
                    sum += a[now][i]->f;
                } else if (cnt == now + 1) {
                    a[now][i]->f = (a[now][i]->f + sum) % MOD;
                }
            }
        }
    }
}
```

哈希表

- 必须初始化
- 备选素数 1572869, 3145739, 6291469, 12582917, 25165843, 50331653

```cpp
const LL HASH_MOD=1572869;
LL key[HASH_MOD], val[HASH_MOD];
int head[HASH_MOD], next[HASH_MOD];
struct Hash {
    int sz;
    void init() {
        memset(head, -1, sizeof head);
        sz = 0;
    }
    LL insert(LL x, LL y) {
        int k = x % HASH_MOD;
        key[sz] = x;
        val[sz] = y;
        next[sz] = head[k];
        head[k] = sz++;
    }
    LL find(LL x) {
        int k = x % HASH_MOD;
        for (int i = head[k]; i != -1; i = next[i])
            if (key[i] == x)
                return val[i];
        return -1;
    }
};
```

笛卡尔树

```
void build(const vector<int>& a) {
    static P *stack[M], *x, *last;
    int p = 0;
    FOR (i, 0, a.size()) {
        x = new P(i + 1, a[i]);
        last = null;
        while (p && stack[p - 1]->v > x->v) {
            stack[p - 1]->maintain();
            last = stack[--p];
        }
        if (p) stack[p - 1]->rs = x;
        x->ls = last;
        stack[p++] = x;
    }
    while (p)
        stack[--p]->maintain();
    rt = stack[0];
}
```

```
void build() {
    static int s[N], last;
    int p = 0;
    FOR (x, 1, n + 1) {
        last = 0;
        while (p && val[s[p - 1]] > val[x]) last = s[--p];
        if (p) G[s[p - 1]][1] = x;
        if (last) G[x][0] = last;
        s[p++] = x;
    }
    rt = s[0];
}
```

Trie

- Trie 二进制版
- M 为二进制的位数
- 使用前必须初始化

```
struct Trie2 {
    int ch[N * M][2], sz;
    void init() {
        memset(ch, 0, sizeof ch);
        sz = 1;
    }
    void insert(LL x) {
        int u = 0;
        FORD (i, M, -1) {
            bool b = x & (1LL << i);
            if (!ch[u][b])
```

```
                ch[u][b] = sz++;
            u = ch[u][b];
        }
    }
} trie;
```

pb_ds

- 优先队列
- binary_heap_tag
- pairing_heap_tag 支持修改
- thin_heap_tag 如果修改只有 increase 则较快，不支持 join

```
#include<ext/pb_ds/priority_queue.hpp>
template<typename _Tv,
   typename Cmp_Fn = std::less<_Tv>,
   typename Tag = pairing_heap_tag,
   typename _Alloc = std::allocator<char> >
class priority_queue;
```

```
#include<ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;

typedef __gnu_pbds::priority_queue<LL, less<LL>, pairing_heap_tag> PQ;
__gnu_pbds::priority_queue<int, cmp, pairing_heap_tag>::point_iterator it;
PQ pq, pq2;

int main() {
    auto it = pq.push(2);
    pq.push(3);
    assert(pq.top() == 3);
    pq.modify(it, 4);
    assert(pq.top() == 4);
    pq2.push(5);
    pq.join(pq2);
    assert(pq.top() == 5);
}
```

- 树
- ov_tree_tag
- rb_tree_tag
- splay_tree_tag
- mapped: null_type 或 null_mapped_type（旧版本）为空
- Node_Update 为 tree_order_statistics_node_update 时才可以 find_by_order & order_of_key
- find_by_order 找 order + 1 小的元素（其实都是从 0 开始计数）
- order_of_key 有多少个比 r_key 小的元素
- join & split

```
template<typename Key, typename Mapped, typename Cmp_Fn = std::less<Key>,
   typename Tag = rb_tree_tag,
   template<typename Node_CItr, typename Node_Itr,
            typename Cmp_Fn_, typename _Alloc_>
   class Node_Update = null_node_update,
   typename _Alloc = std::allocator<char> >
class tree
```

Link-Cut Tree

- 图中相邻的结点在伸展树中不一定是父子关系
- 判断两个点是否连通要用 findroot_

```
struct P {
    P *fa, *ls, *rs;
    int v, maxv;
    bool rev;

    bool has_fa() { return fa->ls == this || fa->rs == this; }
    bool d() { return fa->ls == this; }
    P*& c(bool x) { return x ? ls : rs; }
    void do_rev() {
        if (this == null) return;
        rev ^= 1;
        swap(ls, rs);
    }
    P* up() {
        maxv = max(v, max(ls->maxv, rs->maxv));
        return this;
    }
    void down() {
        if (rev) {
            rev = 0;
            ls->do_rev(); rs->do_rev();
        }
    }
    void all_down() { if (has_fa()) fa->all_down(); down(); }
} *const null = new P{0, 0, 0, 0, 0, 0}, pool[M], *pit = pool;

void rot(P* o) {
    bool dd = o->d();
    P *f = o->fa, *t = o->c(!dd);
    if (f->has_fa()) f->fa->c(f->d()) = o; o->fa = f->fa;
    if (t != null) t->fa = f; f->c(dd) = t;
    o->c(!dd) = f->up(); f->fa = o;
}
void splay(P* o) {
    o->all_down();
    while (o->has_fa()) {
        if (o->fa->has_fa())
            rot(o->d() ^ o->fa->d() ? o : o->fa);
        rot(o);
    }
    o->up();
}
void access(P* u, P* v = null) {
    if (u == null) return;
    splay(u); u->rs = v;
    access(u->up()->fa, u);
}
void make_root(P* o) {
    access(o); splay(o); o->do_rev();
}
void split(P* o, P* u) {
    make_root(o); access(u); splay(u);
}
void link(P* u, P* v) {
    make_root(u); u->fa = v;
}
void cut(P* u, P* v) {
    split(u, v);
    u->fa = v->ls = null; v->up();
}
bool adj(P* u, P* v) {
    split(u, v);
    return v->ls == u && u->ls == null && u->rs == null;
}
bool linked(P* u, P* v) {
    split(u, v);
    return u == v || u->fa != null;
}
P* findrt(P* o) {
    access(o); splay(o);
    while (o->ls != null) o = o->ls;
    return o;
}
```

莫队

- [l, r)

```
while (l > q.l) mv(--l, 1);
while (r < q.r) mv(r++, 1);
while (l < q.l) mv(l++, -1);
while (r > q.r) mv(--r, -1);
```

- 树上莫队
- 注意初始状态 u = v = 1, flip(1)

```
struct Q {
    int u, v, idx;
    bool operator < (const Q& b) const {
        const Q& a = *this;
        return blk[a.u] < blk[b.u] || (blk[a.u] == blk[b.u] && in[a.v] < in[b.v]);
    }
};

void dfs(int u = 1, int d = 0) {
    static int S[maxn], sz = 0, blk_cnt = 0, clk = 0;
    in[u] = clk++;
    dep[u] = d;
    int btm = sz;
    for (int v: G[u]) {
        if (v == fa[u]) continue;
        fa[v] = u;
        dfs(v, d + 1);
        if (sz - btm >= B) {
            while (sz > btm) blk[S[--sz]] = blk_cnt;
            ++blk_cnt;
        }
    }
    S[sz++] = u;
    if (u == 1) while (sz) blk[S[--sz]] = blk_cnt - 1;
}

void flip(int k) {
    dbg(k);
    if (vis[k]) {
        // ...
    } else {
        // ...
    }
    vis[k] ^= 1;
}

void go(int& k) {
    if (bug == -1) {
        if (vis[k] && !vis[fa[k]]) bug = k;
        if (!vis[k] && vis[fa[k]]) bug = fa[k];
    }
    flip(k);
    k = fa[k];
}

void mv(int a, int b) {
    bug = -1;
    if (vis[b]) bug = b;
    if (dep[a] < dep[b]) swap(a, b);
    while (dep[a] > dep[b]) go(a);
    while (a != b) {
        go(a); go(b);
    }
    go(a); go(bug);
}

for (Q& q: query) {
    mv(u, q.u); u = q.u;
    mv(v, q.v); v = q.v;
    ans[q.idx] = Ans;
}
```

数学

矩阵运算

```
struct Mat {
    static const LL M = 2;
    LL v[M][M];
    Mat() { memset(v, 0, sizeof v); }
    void eye() { FOR (i, 0, M) v[i][i] = 1; }
    LL* operator [] (LL x) { return v[x]; }
    const LL* operator [] (LL x) const { return v[x]; }
    Mat operator * (const Mat& B) {
        const Mat& A = *this;
        Mat ret;
        FOR (i, 0, M)
            FOR (j, 0, M)
                 FOR (k, 0, M)
                     ret[i][j] = (ret[i][j] + A[i][k] * B[k][j]) % MOD;
        return ret;
    }
    Mat pow(LL n) const {
        Mat A = *this, ret; ret.eye();
        for (; n; n >>= 1, A = A * A)
            if (n & 1) ret = ret * A;
        return ret;
    }
    Mat operator + (const Mat& B) {
        const Mat& A = *this;
        Mat ret;
        FOR (i, 0, M)
            FOR (j, 0, M)
                 ret[i][j] = (A[i][j] + B[i][j]) % MOD;
        return ret;
    }
    void prt() const {
        FOR (i, 0, M)
            FOR (j, 0, M)
                 printf("%lld%c", (*this)[i][j], j == M - 1 ? '\n' : ' ');
    }
};
```

筛

- 线性筛

```
const LL p_max = 1E6 + 100;
LL prime[p_max], p_sz;
void get_prime() {
    static bool vis[p_max];
    FOR (i, 2, p_max) {
        if (!vis[i]) prime[p_sz++] = i;
        FOR (j, 0, p_sz) {
            if (prime[j] * i >= p_max) break;
```

```
                vis[prime[j] * i] = 1;
                if (i % prime[j] == 0) break;
            }
        }
}
```

- 线性筛 + 欧拉函数

```
const LL p_max = 1E5 + 100;
LL prime[p_max], p_sz, phi[p_max];
void get_phi() {
    static bool vis[p_max];
    FOR (i, 2, p_max) {
        if (!vis[i]) {
            prime[p_sz++] = i;
            phi[i] = i - 1;
        }
        static LL d;
        for (LL j = 0; j < p_sz && (d = i * prime[j]) < p_max; ++j) {
            vis[d] = 1;
            if (i % prime[j] == 0) {
                phi[d] = phi[i] * prime[j];
                break;
            }
            else phi[d] = phi[i] * (prime[j] - 1);
        }
    }
}
```

- 线性筛 + 莫比乌斯函数

```
const LL p_max = 1E5 + 100;
LL prime[p_max], p_sz, mu[p_max];
bool p_vis[p_max];
void get_mu() {
    mu[1] = 1;
    FOR (i, 2, p_max) {
        if (!p_vis[i]) {
            prime[p_sz++] = i;
            mu[i] = -1;
        }
        static LL d;
        for (LL j = 0; j < p_sz && (d = i * prime[j]) < p_max; ++j) {
            p_vis[d] = 1;
            if (i % prime[j] == 0) {
                mu[d] = 0;
                break;
            }
            else mu[d] = -mu[i];
        }
    }
}
```

素数测试

- 前置：快速乘、快速幂
- int 范围内只需检查 2, 7, 61
- long long 范围 2, 325, 9375, 28178, 450775, 9780504, 1795265022
- 3E15 内 2, 2570940, 880937, 610386380, 4130785767
- 4E13 内 2, 2570940, 211991001, 3749873356
- http://miller-rabin.appspot.com/

```
bool checkQ(LL a, LL n) {
    if (n == 2 || a >= n) return 1;
    if (n == 1 || !(n & 1)) return 0;
    LL d = n - 1;
    while (!(d & 1)) d >>= 1;
    LL t = pown(a, d, n);  // 不一定需要快速乘
    while (d != n - 1 && t != 1 && t != n - 1) {
        t = mul(t, t, n);
        d <<= 1;
    }
    return t == n - 1 || d & 1;
}

bool primeQ(LL n) {
    static vector<LL> t = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
    if (n <= 1) return false;
    for (LL k: t) if (!checkQ(k, n)) return false;
    return true;
}
```

线性递推

```
// k 为 m 最高次数 且  a[m] == 1
namespace BerlekampMassey {
    inline void up(LL& a, LL b) { (a += b) %= MOD; }

    V mul(const V& a, const V& b, const V& m, int k) {
        V r; r.resize(2 * k - 1);
        FOR (i, 0, k)
            FOR (j, 0, k)
                up(r[i + j], a[i] * b[j]);
        FORD (i, k - 2, - 1) {
            FOR (j, 0, k)
                up(r[i + j], r[i + k] * m[j]);
            r.pop_back();
        }
        return r;
    }

    V pow(LL n, const V& m) {
        int k = (int)m.size() - 1; assert(m[k] == -1 || m[k] == MOD - 1);
        V r(k), x(k); r[0] = x[1] = 1;
        for (; n; n >>= 1, x = mul(x, x, m, k))
            if (n & 1) r = mul(x, r, m, k);
        return r;
    }

    LL go(const V& a, const V& x, LL n) {
        // a: (-1, a1, a2, ..., ak).reverse
        // x: x1, x2, ..., xk
        // x[n] = sum[a[i]*x[n-i],{i,1,k}]
        int k = (int)a.size() - 1;
        if (n <= k) return x[n - 1];
        V r = pow(n - 1, a);
        LL ans = 0;
        FOR (i, 0, k)
            up(ans, r[i] * x[i]);
        return ans;
    }

    V BM(const V& x) {
        V a = {-1}, b = {233};
        FOR (i, 1, x.size()) {
            b.push_back(0);
            LL d = 0, la = a.size(), lb = b.size();
            FOR (j, 0, la) up(d, a[j] * x[i - la  + 1 + j]);
            if (d == 0) continue;
            V t; for (auto& v: b) t.push_back(d * v % MOD);
            FOR (j, 0, a.size()) up(t[lb - 1 - j], a[la - 1 - j]);
            if (lb > la) {
                b = a;
                LL inv = -get_inv(d, MOD);
                for (auto& v: b) v = v * inv % MOD;
            }
            a.swap(t);
        }
        for (auto& v: a) up(v, MOD);
        return a;
    }
}
```

扩展欧几里得

- 求 $ax + by = gcd(a, b)$ 的一组解
- 如果 a 和 b 互素，那么 x 是 a 在模 b 下的逆元
- 注意 x 和 y 可能是负数

```
LL ex_gcd(LL a, LL b, LL &x, LL &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    LL ret = ex_gcd(b, a % b, y, x);
    y -= a / b * x;
    return ret;
}
```

逆元

- $ax \equiv 1 \pmod p$
- 如果 p 不是素数，使用拓展欧几里得
- 模数是素数，求一个数的逆元
- 前置模板：快速幂

```
inline LL get_inv(LL x, LL p) { return pown(x, p - 2, p); }
```

- 预处理

$$1 - n$$

的逆元

```
LL inv[N];
void inv_init(LL n, LL p) {
    inv[1] = 1;
    FOR (i, 2, n)
        inv[i] = (p - p / i) * inv[p % i] % p;
}
```

- 预处理阶乘及其逆元

```
LL invf[M], fac[M];
void fac_inv_init(LL n, LL p) {
    fac[0] = 1;
    FOR (i, 1, n)
        fac[i] = i * fac[i - 1] % p;
    invf[n - 1] = pown(fac[n - 1], p - 2, p);
    FORD (i, n - 2, -1)
        invf[i] = invf[i + 1] * (i + 1) % p;
}
```

组合数

- 如果数较小，模较大时使用逆元
- 前置模板：逆元-预处理阶乘及其逆元

```
inline LL C(LL n, LL m) { // m >= n >= 0
    return m < n || n < 0 ? 0 : fac[m] * invf[n] % MOD * invf[m - n] % MOD;
}
```

- 如果模数较小，数字较大，使用 Lucas 定理
- 前置模板可选 1：求组合数（如果使用阶乘逆元，需 `fac_inv_init(MOD, MOD);`）
- 前置模板可选 2：模数不固定下使用，无法单独使用。

```
LL C(LL n, LL m) { // m >= n >= 0
    if (m - n < n) n = m - n;
    if (n < 0) return 0;
    LL ret = 1;
    FOR (i, 1, n + 1)
        ret = ret * (m - n + i) % MOD * pown(i, MOD - 2, MOD) % MOD;
    return ret;
}
```

```
LL Lucas(LL n, LL m) { // m >= n >= 0
    return m ? C(n % MOD, m % MOD) * Lucas(n / MOD, m / MOD) % MOD : 1;
}
```

FFT & NTT

- NTT
- 前置：快速幂

```
const int MOD = 998244353;
const int G = 3;
typedef vector<LL> V;

void ntt(V& a, LL N, LL f) {
    LL i, j = 0, t, k;
    for (i = 1; i < N - 1; i++) {
        for (t = N; j ^= t >>= 1, ~j & t;);
        if (i < j) {
            swap(a[i], a[j]);
        }
    }
    for (i = 1; i < N; i <<= 1) {
        t = i << 1;
        LL wn = pown(G, (MOD - 1) / t);
        for (j = 0; j < N; j += t) {
            LL w = 1;
            for (k = 0; k < i; k++, w = w * wn % MOD) {
                LL x = a[j + k], y = w * a[j + k + i] % MOD;
                a[j + k] = (x + y) % MOD, a[j + k + i] = (x - y + MOD) % MOD;
            }
        }
    }
    if (f == -1) {
        reverse(a.begin() + 1, a.begin() + N);
        LL inv = pown(N, MOD - 2);
        for (i = 0; i < N; i++)
            a[i] = a[i] * inv % MOD;
    }
}

void solve(V& a, V& b) {
    LL N = 1, n = max(a.size(), b.size()) - 1;
    while (N <= 2 * n)
        N <<= 1;
    a.resize(N); b.resize(N);
    ntt(a, N, 1);
    ntt(b, N, 1);
    FOR (i, 0, N)
        a[i] = a[i] * b[i] % MOD;
    ntt(a, N, -1);
}
```

- FFT
- n 需补成 2 的幂（n 必须超过 a 和 b 的最高指数之和）

```
typedef double LD;
const LD PI = acos(-1);
struct C {
    LD r, i;
    C(LD r = 0, LD i = 0): r(r), i(i) {}
};
C operator + (const C& a, const C& b) {
    return C(a.r + b.r, a.i + b.i);
}
C operator - (const C& a, const C& b) {
    return C(a.r - b.r, a.i - b.i);
}
C operator * (const C& a, const C& b) {
    return C(a.r * b.r - a.i * b.i, a.r * b.i + a.i * b.r);
}

void FFT(C x[], int n, int p) {
    for (int i = 0, t = 0; i < n; ++i) {
        if (i > t) swap(x[i], x[t]);
        for (int j = n >> 1; (t ^= j) < j; j >>= 1);
    }
    for (int h = 2; h <= n; h <<= 1) {
        C wn(cos(p * 2 * PI / h), sin(p * 2 * PI / h));
        for (int i = 0; i < n; i += h) {
            C w(1, 0), u;
            for (int j = i, k = h >> 1; j < i + k; ++j) {
                u = x[j + k] * w;
                x[j + k] = x[j] - u;
                x[j] = x[j] + u;
                w = w * wn;
            }
        }
    }
    if (p == -1)
        FOR (i, 0, n)
            x[i].r /= n;
}

void conv(C a[], C b[], int n) {
    FFT(a, n, 1);
    FFT(b, n, 1);
    FOR (i, 0, n)
        a[i] = a[i] * b[i];
    FFT(a, n, -1);
}
```

simpson 自适应积分

```
LD simpson(LD l, LD r) {
    LD c = (l + r) / 2;
    return (f(l) + 4 * f(c) + f(r)) * (r - l) / 6;
}

LD asr(LD l, LD r, LD eps, LD S) {
    LD m = (l + r) / 2;
    LD L = simpson(l, m), R = simpson(m, r);
    if (fabs(L + R - S) < 15 * eps) return L + R + (L + R - S) / 15;
    return asr(l, m, eps / 2, L) + asr(m, r, eps / 2, R);
}

LD asr(LD l, LD r, LD eps) { return asr(l, r, eps, simpson(l, r)); }
```

快速乘

```
LL mul(LL a, LL b, LL m) {
    LL ret = 0;
    while (b) {
        if (b & 1) {
            ret += a;
            if (ret >= m) ret -= m;
        }
        a += a;
        if (a >= m) a -= m;
        b >>= 1;
    }
    return ret;
}
```

- O(1)

```
LL mul(LL u,LL v, LL p) {
    return (u * v - LL((long double) u * v / p) * p + p) % p;
}
```

快速幂

- 如果模数是素数，则可在函数体内加上 `n %= MOD - 1;`（费马小定理）。

```
LL pown(LL x, LL n, LL MOD) {
    LL ret = MOD != 1;
    for (x %= MOD; n; n >>= 1, x = x * x % MOD)
        if (n & 1) ret = ret * x % MOD;
    return ret;
}
```

- 防爆 LL
- 前置模板：快速乘

```
LL pown(LL x, LL n, LL MOD) {
    LL ret = MOD != 1;
    for (x %= MOD; n; n >>= 1, x = mul(x, x, MOD))
        if (n & 1) ret = mul(ret, x, MOD);
    return ret;
}
```

高斯消元

- n - 方程个数，m - 变量个数，a 是 n * (m + 1) 的增广矩阵，free 是否为自由变量

- 返回自由变量个数，-1 无解，-2 无整数解

- 浮点数版本

```
typedef double LD;
const LD eps = 1E-10;
const int maxn = 2000 + 10;

int n, m;
LD a[maxn][maxn], x[maxn];
bool free_x[maxn];

inline int sgn(LD x) { return (x > eps) - (x < -eps); }


int guass(LD a[maxn][maxn], int n, int m) {
    memset(free_x, 1, sizeof free_x); memset(x, 0, sizeof x);
    int r = 0, c = 0;
    while (r < n && c < m) {
        int m_r = r;
        FOR (i, r + 1, n)
            if (fabs(a[i][c]) > fabs(a[m_r][c])) m_r = i;
        if (m_r != r)
            FOR (j, c, m + 1)
                 swap(a[r][j], a[m_r][j]);
        if (!sgn(a[r][c])) {
            a[r][c] = 0;
            ++c;
            continue;
        }
        FOR (i, r + 1, n)
```

```
            if (a[i][c]) {
                LD t = a[i][c] / a[r][c];
                FOR (j, c, m + 1) a[i][j] -= a[r][j] * t;
            }
        ++r; ++c;
//        FOR (i, 0, n)
//            FOR (j, 0, m + 1)
//                printf("%.2f%c", a[i][j], j == _j - 1 ? '\n' : ' '); puts("");
    }
    FOR (i, r, n)
        if (sgn(a[i][m])) return -1;
    if (r < m) {
        FORD (i, r - 1, -1) {
            int f_cnt = 0, k = -1;
            FOR (j, 0, m)
                if (sgn(a[i][j]) && free_x[j]) {
                    ++f_cnt;
                    k = j;
                }
            if(f_cnt > 0) continue;
            LD s = a[i][m];
            FOR (j, 0, m)
                if (j != k) s -= a[i][j] * x[j];
            x[k] = s / a[i][k];
            free_x[k] = 0;
        }
        return m - r;
    }
    FORD (i, m - 1, -1) {
        LD s = a[i][m];
        FOR (j, i + 1, m)
            s -= a[i][j] * x[j];
        x[i] = s / a[i][i];
    }
    return 0;
}
```

- 数据

```
3 4
1 1 -2 2
2 -3 5 1
4 -1 1 5
5 0 -1 7
// many

3 4
1 1 -2 2
2 -3 5 1
4 -1 -1 5
5 0 -1 0 2
// no

3 4
1 1 -2 2
2 -3 5 1
4 -1 1 5
5 0 1 0 7
// one
```

质因数分解

- 前置模板：素数筛

- 带指数

```
LL factor[30], f_sz, factor_exp[30];
void get_factor(LL x) {
    f_sz = 0;
    LL t = sqrt(x + 0.5);
    for (LL i = 0; prime[i] <= t; ++i)
        if (x % prime[i] == 0) {
            factor_exp[f_sz] = 0;
            while (x % prime[i] == 0) {
                x /= prime[i];
                ++factor_exp[f_sz];
            }
            factor[f_sz++] = prime[i];
        }
    if (x > 1) {
        factor_exp[f_sz] = 1;
        factor[f_sz++] = x;
    }
}
```

- 不带指数

```
LL factor[30], f_sz;
void get_factor(LL x) {
    f_sz = 0;
    LL t = sqrt(x + 0.5);
    for (LL i = 0; prime[i] <= t; ++i)
        if (x % prime[i] == 0) {
            factor[f_sz++] = prime[i];
            while (x % prime[i] == 0) x /= prime[i];
        }
    if (x > 1) factor[f_sz++] = x;
}
```

原根

- 前置模板：素数筛，快速幂，分解质因数
- 要求 p 为质数

```
LL find_smallest_primitive_root(LL p) {
    get_factor(p - 1);
    FOR (i, 2, p) {
        bool flag = true;
        FOR (j, 0, f_sz)
            if (pown(i, (p - 1) / factor[j], p) == 1) {
                flag = false;
                break;
            }
        if (flag) return i;
    }
    assert(0); return -1;
}
```

公式

- 当 $x \geq \phi(p)$ 时有 $a^x \equiv a^{x \ mod \ \phi(p) + \phi(p)} \pmod p$
- $F_{a+b} = F_{a-1} \cdot F_b + F_a \cdot F_{b+1}$

中国剩余定理

- 无解返回 -1
- 前置模板：拓展欧几里得

```
LL CRT(LL *m, LL *r, LL n) {
    if (!n) return 0;
    LL M = m[0], R = r[0], x, y, d;
    FOR (i, 1, n) {
        d = ex_gcd(M, m[i], x, y);
        if ((r[i] - R) % d) return -1;
        x = (r[i] - R) / d * x % (m[i] / d);
        R += x * M;
        M = M / d * m[i];
        R %= M;
    }
    return R >= 0 ? R : R + M;
}
```

伯努利数和等幂求和

```
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
#define FOR(i, x, y) for (decay<decltype(y)>::type i = (x), _##i = (y); i < _##i; ++i)
#define FORD(i, x, y) for (decay<decltype(x)>::type i = (x), _##i = (y); i > _##i; --i)
#ifdef zerol
#define dbg(args...) do { cout << "\033[32;1m" << #args<< " -> "; err(args); } while (0)
#else
#define dbg(...)
#endif
void err() { cout << "\033[39;0m" << endl; }
template<typename T, typename... Args>
void err(T a, Args... args) { cout << a << ' '; err(args...); }
// -----------------------------------------------------------------------------
const LL MOD = 1E9 + 7; // TODO
const int M = 2000 + 50;

LL inv[M];
void inv_init(LL n, LL p) {
    inv[1] = 1;
    FOR (i, 2, n)
        inv[i] = (p - p / i) * inv[p % i] % p;
}

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;
    }
}

LL B[M];
void init() {
    inv_init(M, MOD);
    init_C(M);
    B[0] = 1;
    FOR (i, 1, M - 1) {
        LL& s = B[i] = 0;
        FOR (j, 0, i)
            s += C[i + 1][j] * B[j] % MOD;
        s = (s % MOD * -inv[i + 1] % MOD + MOD) % MOD;
    }
}

LL T, n, k;
LL p[M] = {1};

LL go() {
    LL ret = 0;
    FOR (i, 1, k + 2)
        ret += C[k + 1][i] * B[k + 1 - i] % MOD * p[i] % MOD;
    ret = ret % MOD * inv[k + 1] % MOD;
    return ret;
}

int main() {
    init();
    cin >> T;
    while (T--) {
        scanf("%lld%lld", &n, &k);
        n %= MOD;
        FOR (i, 1, k + 2)
            p[i] = p[i - 1] * (n + 1) % MOD;
        LL ans = go();
        printf("%lld\n", ans);
    }
}
```

图论

LCA

- 倍增

```
void dfs(int u, int fa) {
    pa[u][0] = fa; dep[u] = dep[fa] + 1;
    FOR (i, 1, SP) pa[u][i] = pa[pa[u][i - 1]][i - 1];
    for (int& v: G[u]) {
        if (v == fa) continue;
        dfs(v, u);
    }
}

int lca(int u, int v) {
    if (dep[u] < dep[v]) swap(u, v);
    int t = dep[u] - dep[v];
    FOR (i, 0, SP) if (t & (1 << i)) u = pa[u][i];
    FORD (i, SP - 1, -1) {
        int uu = pa[u][i], vv = pa[v][i];
        if (uu != vv) { u = uu; v = vv; }
    }
    return u == v ? u : pa[u][0];
}
```

最短路

```
bool BF() {
    queue<int> q;
    FOR (i, 1, n) d[i] = INF;
    d[0] = 0; inq[0] = true; q.push(0);
    while (!q.empty()) {
        int u = q.front(); q.pop();
        inq[u] = false;
        for (E& e: G[u]) {
            int v = e.to;
            if (d[u] < INF && d[v] > d[u] + e.d) {
                d[v] = d[u] + e.d;
                if (!inq[v]) {
                    q.push(v); inq[v] = true;
                    if (++cnt[v] > n) return false;
                }
            }
        }
    }
    return true;
}
```

网络流

- 最大流

```
struct E {
    int to, cp;
    E(int to, int cp): to(to), cp(cp) {}
};

struct Dinic {
    static const int M = 1E5 * 5;
    int m, s, t;
    vector<E> edges;
    vector<int> G[M];
    int d[M];
    int cur[M];

    void init(int n, int s, int t) {
        this->s = s; this->t = t;
        for (int i = 0; i <= n; i++) G[i].clear();
        edges.clear(); m = 0;
    }

    void addedge(int u, int v, int cap) {
        edges.emplace_back(v, cap);
        edges.emplace_back(u, 0);
        G[u].push_back(m++);
        G[v].push_back(m++);
    }

    bool BFS() {
        memset(d, 0, sizeof d);
        queue<int> Q;
        Q.push(s); d[s] = 1;
        while (!Q.empty()) {
            int x = Q.front(); Q.pop();
            for (int& i: G[x]) {
                E &e = edges[i];
                if (!d[e.to] && e.cp > 0) {
                    d[e.to] = d[x] + 1;
                    Q.push(e.to);
                }
            }
        }
        return d[t];
    }

    int DFS(int u, int cp) {
        if (u == t || !cp) return cp;
        int tmp = cp, f;
        for (int& i = cur[u]; i < G[u].size(); i++) {
            E& e = edges[G[u][i]];
            if (d[u] + 1 == d[e.to]) {
                f = DFS(e.to, min(cp, e.cp));
                e.cp -= f;
                edges[G[u][i] ^ 1].cp += f;
                cp -= f;
                if (!cp) break;
            }
        }
        return tmp - cp;
    }

    int go() {
        int flow = 0;
        while (BFS()) {
            memset(cur, 0, sizeof cur);
            flow += DFS(s, INF);
        }
        return flow;
    }
} DC;
```

- 费用流

```
struct E {
    int from, to, cp, v;
    E() {}
    E(int f, int t, int cp, int v) : from(f), to(t), cp(cp), v(v) {}
};

struct MCMF {
    int n, m, s, t;
    vector<E> edges;
    vector<int> G[maxn];
    bool inq[maxn];     //是否在队列
    int d[maxn];        //Bellman_ford 单源最短路径
    int p[maxn];        //p[i] 表从 s 到 i 的最小费用路径上的最后一条弧编号
    int a[maxn];        //a[i] 表示从 s 到 i 的最小残量

    void init(int _n, int _s, int _t) {
        n = _n; s = _s; t = _t;
        FOR (i, 0, n + 1) G[i].clear();
        edges.clear(); m = 0;
    }

    void addedge(int from, int to, int cap, int cost) {
        edges.emplace_back(from, to, cap, cost);
        edges.emplace_back(to, from, 0, -cost);
        G[from].push_back(m++);
        G[to].push_back(m++);
    }

    bool BellmanFord(int &flow, int &cost) {
        FOR (i, 0, n + 1) d[i] = INF;
        memset(inq, 0, sizeof inq);
        d[s] = 0, a[s] = INF, inq[s] = true;
        queue<int> Q; Q.push(s);
        while (!Q.empty()) {
            int u = Q.front(); Q.pop();
            inq[u] = false;
            for (int& idx: G[u]) {
                E &e = edges[idx];
                if (e.cp && d[e.to] > d[u] + e.v) {
                    d[e.to] = d[u] + e.v;
                    p[e.to] = idx;
                    a[e.to] = min(a[u], e.cp);
                    if (!inq[e.to]) {
                        Q.push(e.to);
                        inq[e.to] = true;
                    }
                }
            }
        }
        if (d[t] == INF) return false;
        flow += a[t];
        cost += a[t] * d[t];
        int u = t;
        while (u != s) {
            edges[p[u]].cp -= a[t];
            edges[p[u] ^ 1].cp += a[t];
            u = edges[p[u]].from;
        }
        return true;
    }

    int go() {
        int flow = 0, cost = 0;
        while (BellmanFord(flow, cost));
        return cost;
    }
} MM;
```

- zkw 费用流（代码长度没有优势）
- 不允许有负权边

```
struct E {
    int to, cp, v;
    E() {}
    E(int to, int cp, int v): to(to), cp(cp), v(v) {}
};

struct MCMF {
    int n, m, s, t, cost, D;
    vector<E> edges;
    vector<int> G[maxn];
    bool vis[maxn];

    void init(int _n, int _s, int _t) {
        n = _n; s = _s; t = _t;
        FOR (i, 0, n + 1) G[i].clear();
        edges.clear(); m = 0;
    }

    void addedge(int from, int to, int cap, int cost) {
        edges.emplace_back(to, cap, cost);
        edges.emplace_back(from, 0, -cost);
        G[from].push_back(m++);
        G[to].push_back(m++);
    }

    int aug(int u, int cp) {
        if (u == t) {
            cost += D * cp;
            return cp;
        }
        vis[u] = true;
        int tmp = cp;
        for (int idx: G[u]) {
            E& e = edges[idx];
            if (e.cp && !e.v && !vis[e.to]) {
                int f = aug(e.to, min(cp, e.cp));
                e.cp -= f;
                edges[idx ^ 1].cp += f;
                cp -= f;
                if (!cp) break;
            }
        }
        return tmp - cp;
    }

    bool modlabel() {
        int d = INF;
        FOR (u, 0, n + 1)
            if (vis[u])
                for (int& idx: G[u]) {
                    E& e = edges[idx];
                    if (e.cp && !vis[e.to]) d = min(d, e.v);
                }
        if (d == INF) return false;
        FOR (u, 0, n + 1)
            if (vis[u])
                for (int& idx: G[u]) {
                    edges[idx].v -= d;
                    edges[idx ^ 1].v += d;
                }
        D += d;
        return true;
    }

    int go(int k) {
        cost = D = 0;
        int flow = 0;
        while (true) {
            memset(vis, 0, sizeof vis);
            int t = aug(s, INF);
            if (!t && !modlabel()) break;
            flow += t;
        }
        return cost;
    }
} MM;
```

树上路径交

```
int intersection(int x, int y, int xx, int yy) {
    int t[4] = {lca(x, xx), lca(x, yy), lca(y, xx), lca(y, yy)};
    sort(t, t + 4);
    int r = lca(x, y), rr = lca(xx, yy);
    if (dep[t[0]] < min(dep[r], dep[rr]) || dep[t[2]] < max(dep[r], dep[rr]))
        return 0;
    int tt = lca(t[2], t[3]);
    int ret = 1 + dep[t[2]] + dep[t[3]] - dep[tt] * 2;
    return ret;
}
```

树上点分治

```
int get_sz(int u, int fa) {
    int& s = sz[u] = 1;
    for (E& e: G[u]) {
        int v = e.to;
        if (vis[v] || v == fa) continue;
        s += get_sz(v, u);
    }
    return s;
}

void get_rt(int u, int fa, int s, int& m, int& rt) {
    int t = s - sz[u];
    for (E& e: G[u]) {
        int v = e.to;
        if (vis[v] || v == fa) continue;
        get_rt(v, u, s, m, rt);
        t = max(t, sz[v]);
    }
    if (t < m) { m = t; rt = u; }
}

void dfs(int u) {
    int tmp = INF; get_rt(u, -1, get_sz(u, -1), tmp, u);
    vis[u] = true;
    get_dep(u, -1, 0);
    // ...
    for (E& e: G[u]) {
        int v = e.to;
        if (vis[v]) continue;
        // ...
        dfs(v);
    }
}
```

- 动态点分治

```
const int maxn = 15E4 + 100, INF = 1E9;
struct E {
    int to, d;
};
vector<E> G[maxn];
int n, Q, w[maxn];
LL A, ans;

```

```
bool vis[maxn];
int sz[maxn];

int get_rt(int u) {
//    dbg(u);
    static int q[N], fa[N], sz[N], mx[N];
    int p = 0, cur = -1;
    q[p++] = u; fa[u] = -1;
    while (++cur < p) {
        u = q[cur]; mx[u] = 0; sz[u] = 1;
        for (int& v: G[u])
            if (!vis[v] && v != fa[u]) fa[q[p++] = v] = u;
    }
    FORD (i, p - 1, -1) {
        u = q[i];
        mx[u] = max(mx[u], p - sz[u]);
        if (mx[u] * 2 <= p) return u;
        sz[fa[u]] += sz[u];
        mx[fa[u]] = max(mx[fa[u]], sz[u]);
    }
    assert(0);
}

int get_sz(int u, int fa) {
    int& s = sz[u] = 1;
    for (E& e: G[u]) {
        int v = e.to;
        if (vis[v] || v == fa) continue;
        s += get_sz(v, u);
    }
    return s;
}

void get_rt(int u, int fa, int s, int& m, int& rt) {
    int t = s - sz[u];
    for (E& e: G[u]) {
        int v = e.to;
        if (vis[v] || v == fa) continue;
        get_rt(v, u, s, m, rt);
        t = max(t, sz[v]);
    }
    if (t < m) { m = t; rt = u; }
}

int dep[maxn], md[maxn];
void get_dep(int u, int fa, int d) {
    dep[u] = d; md[u] = 0;
    for (E& e: G[u]) {
        int v = e.to;
        if (vis[v] || v == fa) continue;
        get_dep(v, u, d + e.d);
        md[u] = max(md[u], md[v] + 1);
    }
}

struct P {
    int w;
    LL s;
};
using VP = vector<P>;
struct R {
    VP *rt, *rt2;
    int dep;
};
VP pool[maxn << 1], *pit = pool;
vector<R> tr[maxn];

void go(int u, int fa, VP* rt, VP* rt2) {
    tr[u].push_back({rt, rt2, dep[u]});
    for (E& e: G[u]) {
        int v = e.to;
        if (v == fa || vis[v]) continue;
        go(v, u, rt, rt2);
    }
}

void dfs(int u) {
    int tmp = INF; get_rt(u, -1, get_sz(u, -1), tmp, u);
    vis[u] = true;
    get_dep(u, -1, 0);
    VP* rt = pit++; tr[u].push_back({rt, nullptr, 0});
    for (E& e: G[u]) {
        int v = e.to;
        if (vis[v]) continue;
        go(v, u, rt, pit++);
        dfs(v);
    }
}

bool cmp(const P& a, const P& b) { return a.w < b.w; }

LL query(VP& p, int d, int l, int r) {
    l = lower_bound(p.begin(), p.end(), P{l, -1}, cmp) - p.begin();
    r = upper_bound(p.begin(), p.end(), P{r, -1}, cmp) - p.begin() - 1;
    return p[r].s - p[l - 1].s + 1LL * (r - l + 1) * d;
}

int main() {
    cin >> n >> Q >> A;
    FOR (i, 1, n + 1) scanf("%d", &w[i]);
    FOR (_, 1, n) {
        int u, v, d; scanf("%d%d%d", &u, &v, &d);
        G[u].push_back({v, d}); G[v].push_back({u, d});
    }
    dfs(1);
    FOR (i, 1, n + 1)
        for (R& x: tr[i]) {
            x.rt->push_back({w[i], x.dep});
            if (x.rt2) x.rt2->push_back({w[i], x.dep});
        }
    FOR (it, pool, pit) {
        it->push_back({-INF, 0});
        sort(it->begin(), it->end(), cmp);
        FOR (i, 1, it->size())
            (*it)[i].s += (*it)[i - 1].s;
    }
    while (Q--) {
        int u; LL a, b; scanf("%d%lld%lld", &u, &a, &b);
        a = (a + ans) % A; b = (b + ans) % A;
        int l = min(a, b), r = max(a, b);
        ans = 0;
        for (R& x: tr[u]) {
            ans += query(*(x.rt), x.dep, l, r);
            if (x.rt2) ans -= query(*(x.rt2), x.dep, l, r);
        }
        printf("%lld\n", ans);
    }
}
```

树链剖分

```
int fa[maxn], dep[maxn], idx[maxn], out[maxn], ridx[maxn];
namespace hld {
    int sz[maxn], son[maxn], top[maxn], clk;
    void predfs(int u, int d) {
        dep[u] = d; sz[u] = 1;
        int& maxs = son[u] = -1;
        for (int& v: G[u]) {
            if (v == fa[u]) continue;
            fa[v] = u;
            predfs(v, d + 1);
            sz[u] += sz[v];
            if (maxs == -1 || sz[v] > sz[maxs]) maxs = v;
        }
    }
    void dfs(int u, int tp) {
        top[u] = tp; idx[u] = ++clk; ridx[clk] = u;
        if (son[u] != -1) dfs(son[u], tp);
        for (int& v: G[u])
            if (v != fa[u] && v != son[u]) dfs(v, v);
        out[u] = clk;
    }
    template<typename T>
    int go(int u, int v, T&& f = [](int, int) {}) {
        int uu = top[u], vv = top[v];
        while (uu != vv) {
            if (dep[uu] < dep[vv]) { swap(uu, vv); swap(u, v); }
            f(idx[uu], idx[u]);
            u = fa[uu]; uu = top[u];
        }
        if (dep[u] < dep[v]) swap(u, v);
        // f(idx[v], idx[u]);
        // if (u != v) f(idx[v] + 1, idx[u]);
        return v;
    }
    int up(int u, int d) {
        while (d) {
            if (dep[u] - dep[top[u]] < d) {
                d -= dep[u] - dep[top[u]];
                u = top[u];
            } else return ridx[idx[u] - d];
            u = fa[u]; --d;
        }
        return u;
    }
}
```

- HDU 3966

```
# include <bits/stdc++.h>
using namespace std;
typedef long long LL;
#define FOR(i, x, y) for (decay<decltype(y)>::type i = (x), _##i = (y); i < _##i; ++i)
#define FORD(i, x, y) for (decay<decltype(x)>::type i = (x), _##i = (y); i > _##i; --i)
#ifdef zerol
#define dbg(args...) do { cout << "\033[32;1m" << #args<< " -> "; err(args); } while (0)
#else
#define dbg(...)
#endif
void err() { cout << "\033[39;0m" << endl; }
template<typename T, typename... Args>
void err(T a, Args... args) { cout << a << ' '; err(args...); }
// -----------------------------------------------------------------------------
const int maxn = 5E4 + 100;
vector<int> G[maxn];
int dep[maxn], sz[maxn], son[maxn], fa[maxn], idx[maxn], top[maxn];
int clk, n, Q;

struct IntervalTree {
#define ls o * 2, l, (l + r) >> 1
#define rs o * 2 + 1, ((l + r) >> 1) + 1, r
    static const int M = maxn << 2;
    int addv[M];
    void init() { memset(addv, 0, sizeof addv); }
    int query(int k, int o, int l, int r, int add = 0) {
        if (k < l || r < k) return 0;
        if (l == r) return add + addv[o];
        return query(k, ls, add + addv[o]) + query(k, rs, add + addv[o]);
    }
    void update(int p, int q, int o, int l, int r, int add) {
        assert(l <= r && r <= n);
        if (q < l || r < p) return;
        if (p <= l && r <= q) addv[o] += add;
        else { update(p, q, ls, add); update(p, q, rs, add); }
    }
} IT;

void predfs(int u, int d) {
    dep[u] = d;
    sz[u] = 1;
    int& maxs = son[u] = -1;
    for (int v: G[u])
        if (v != fa[u]) {
            fa[v] = u;
            predfs(v, d + 1);
            sz[u] += sz[v];
            if (maxs == -1 || sz[v] > sz[maxs])
                maxs = v;
        }
}

void dfs(int u, int tp) {
    top[u] = tp;
    idx[u] = ++clk;
    if (son[u] != -1) dfs(son[u], tp);
    for (int v: G[u])
        if (v != son[u] && v != fa[u])
            dfs(v, v);
}

void update(int u, int v, int add) {
    int uu = top[u], vv = top[v];
    while (uu != vv) {
        if (dep[uu] < dep[vv]) { swap(uu, vv); swap(u, v); }
        IT.update(idx[uu], idx[u], 1, 1, n, add);
        u = fa[uu];
        uu = top[u];
    }
    if (dep[u] < dep[v]) swap(u, v);
    dbg(u, v, idx[u], idx[v]);
    IT.update(idx[v], idx[u], 1, 1, n, add);
}

int a[maxn];
void init();
int main() {
    int u, v, l, r, k, d;
    char s[100];
    while (cin >> n >> Q >> Q) {
        init();
        FOR (i, 1, n + 1) scanf("%d", &a[i]);
        FOR (i, 1, n) {
            scanf("%d%d", &u, &v);
            G[u].push_back(v);
            G[v].push_back(u);
        }
        predfs(1, 1);
        dfs(1, 1);
        while (Q--) {
            scanf("%s", s);
            if (s[0] == 'I') {
                scanf("%d%d%d", &l, &r, &d);
                update(l, r, d);
            } else if (s[0] == 'D') {
                scanf("%d%d%d", &l, &r, &d);
                update(l, r, -d);
            } else {
                scanf("%d", &k);
                printf("%d\n", a[k] + IT.query(idx[k], 1, 1, n));
            }
        }
    }
}
```

```
void init() {
    clk = 0;
    fa[1] = 0;
    IT.init();
    FOR (i, 0, n + 1) G[i].clear();
}
```

- SPOJ QTREE

```
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
#define FOR(i, x, y) for (decay<decltype(y)>::type i = (x), _##i = (y); i < _##i; ++i)
#define FORD(i, x, y) for (decay<decltype(x)>::type i = (x), _##i = (y); i > _##i; --i)
#ifdef zerol
#define dbg(args...) do { cout << "\033[32;1m" << #args<< " -> "; err(args); } while (0)
#else
#define dbg(...)
#endif
void err() { cout << "\033[39m" << endl; }
template<typename T, typename... Args>
void err(T a, Args... args) {
    cout << a << ' ';
    err(args...);
}
// ------------------------------------------------------------------------------
const int maxn = 10000 * 2 * 4 + 100;
struct Edge {
    int from, to, c;
    Edge(int u, int v, int c): from(u), to(v), c(c) {}
};
vector<Edge> edge;
vector<int> G[maxn];
int fa[maxn], dep[maxn], sz[maxn], son[maxn], top[maxn], idx[maxn], w[maxn], val[maxn];
LL sum[maxn];
int n, clk, len;

struct IntervalTree {
#define lson p, q, o * 2, l, m
#define rson p, q, o * 2 + 1, m + 1, r
    int maxv[maxn];
    void init() { memset(maxv, 0, sizeof maxv); }
    int query(int p, int q, int o, int l, int r) {
//        dbg(p, q);
        assert(p <= q);
        if (p > r || q < l) return 0;
        if (p <= l && r <= q) return maxv[o];
        int m = (l + r) / 2;
        return max(query(lson), query(rson));
    }
    void maintain(int o, int l, int r) {
        if (l < r)
            maxv[o] = max(maxv[o * 2], maxv[o * 2 + 1]);
    }
    void update(int p, int q, int o, int l, int r, int v) {
//        dbg(p, q, o, l, r, v);
        assert(p <= q);
        if (p > r || q < l) return;
        if (p <= l && r <= q) maxv[o] = v;
        else {
            int m = (l + r) / 2;
            update(lson, v); update(rson, v);
            maintain(o, l, r);
        }
    }
} IT;

void dfs1(int u, int d) {
    dep[u] = d;
    sz[u] = 1;
    FOR (i, 0, G[u].size()) {
        Edge& e = edge[G[u][i]];
        int v = e.to;
        if (v == fa[u]) continue;
        val[v] = e.c;
//        dbg(v, e.from, e.to, e.c);
        fa[v] = u;
        dfs1(v, d + 1);
        sz[u] += sz[v];
        if (son[u] == -1 || sz[v] > sz[son[u]])
            son[u] = v;
    }
}

void dfs2(int u, int tp) {
    top[u] = tp;
    idx[u] = ++clk;
    w[idx[u]] = tp;
    if (son[u] == -1) return;
    dfs2(son[u], tp);
    FOR (i, 0, G[u].size()) {
        int v = edge[G[u][i]].to;
        if (v != son[u] && v != fa[u])
            dfs2(v, v);
    }
}

int query(int u, int v) {
    dbg(u, v);
    int uu = top[u], vv = top[v], ret = 0;
    while (uu != vv) {
        if (dep[uu] < dep[vv]) { swap(u, v); swap(uu, vv); }
//        dbg(u, v, uu, vv, dep[uu], dep[vv], idx[uu], idx[u]);
        ret = max(ret, IT.query(idx[uu], idx[u], 1, 1, len));
        u = fa[uu];
        uu = top[u];
    }
    if (dep[u] < dep[v]) swap(u, v);
//    dbg(idx[v], idx[u]);
    if (u != v) ret = max(ret, IT.query(idx[v] + 1, idx[u], 1, 1, len));
    return ret;
}

void init();
void add_edge(int u, int v, int c);

int main() {
#ifdef zerol
    freopen("in", "r", stdin);
#endif
    int T, u, v, c;
    char s[100];
    cin >> T;
    while (T--) {
        cin >> n;
        for (len = 1; len < n; len *= 2);
        init();
        FOR (i, 1, n) {
            scanf("%d%d%d", &u, &v, &c);
            add_edge(u, v, c);
            add_edge(v, u, c);
        }
        dfs1(1, 0);
        dfs2(1, 1);
//        FOR (i, 1, n + 1) dbg(idx[i], w[i]);
        FOR (i, 2, n + 1)
            IT.update(idx[i], idx[i], 1, 1, len, val[i]);
        while (scanf("%s", s) && s[0] != 'D') {
            scanf("%d%d", &u, &v);
            if (s[0] == 'C') {
                Edge& e = edge[u * 2 - 1];
                dbg(u, e.from, e.to);
                int t = max(idx[e.from], idx[e.to]);
                IT.update(t, t, 1, 1, len, v);
                dbg("upd", t, v);
            }
            if (s[0] == 'Q') printf("%d\n", query(u, v));
        }
        FOR (i, 1, n + 1) if (idx[i] == 2) dbg(i, idx[i]);
        dbg(IT.query(idx[2], idx[2], 1, 1, len));
        dbg(IT.query(idx[6], idx[6], 1, 1, len));
    }
}

void init() {
    edge.clear();
    memset(son, -1, sizeof son);
    memset(sum, 0, sizeof sum);
    IT.init();
    FOR (i, 0, n + 1) G[i].clear();
    clk = 0;
    fa[1] = 0;
    sum[0] = sum[1] = 0;
}

void add_edge(int u, int v, int c) {
    edge.emplace_back(u, v, c);
    G[u].push_back(edge.size() - 1);
}
```

二分图匹配

- 最小覆盖数 = 最大匹配数
- 最大独立集 = 顶点数 - 二分图匹配数
- DAG 最小路径覆盖数 = 结点数 - 拆点后二分图最大匹配数

```
struct MaxMatch {
    int n;
    vector<int> G[maxn];
    int vis[maxn], left[maxn], clk;

    void init(int n) {
        this->n = n;
        FOR (i, 0, n + 1) G[i].clear();
        memset(left, -1, sizeof left);
        memset(vis, -1, sizeof vis);
    }

    bool dfs(int u) {
        for (int v: G[u])
            if (vis[v] != clk) {
                vis[v] = clk;
                if (left[v] == -1 || dfs(left[v])) {
                    left[v] = u;
                    return true;
                }
            }
        return false;
    }

    int match() {
        int ret = 0;
        for (clk = 0; clk <= n; ++clk)
            if (dfs(clk)) ++ret;
        return ret;
    }
} MM;
```

- 二分图最大权完美匹配 KM

```
namespace R {
    const int maxn = 300 + 10;
    int n, m;
    int left[maxn], L[maxn], R[maxn];
    int w[maxn][maxn], slack[maxn];
    bool visL[maxn], visR[maxn];

    bool dfs(int u) {
        visL[u] = true;
        FOR (v, 0, m) {
            if (visR[v]) continue;
            int t = L[u] + R[v] - w[u][v];
            if (t == 0) {
                visR[v] = true;
                if (left[v] == -1 || dfs(left[v])) {
                    left[v] = u;
                    return true;
                }
            } else slack[v] = min(slack[v], t);
        }
        return false;
    }

    int go() {
        memset(left, -1, sizeof left);
        memset(R, 0, sizeof R);
        memset(L, 0, sizeof L);
        FOR (i, 0, n)
            FOR (j, 0, m)
                L[i] = max(L[i], w[i][j]);

        FOR (i, 0, n) {
            memset(slack, 0x3f, sizeof slack);
            while (1) {
                memset(visL, 0, sizeof visL); memset(visR, 0, sizeof visR);
                if (dfs(i)) break;
                int d = 0x3f3f3f3f;
                FOR (j, 0, m) if (!visR[j]) d = min(d, slack[j]);
                FOR (j, 0, n) if (visL[j]) L[j] -= d;
                FOR (j, 0, m) if (visR[j]) R[j] += d; else slack[j] -= d;
            }
        }
        int ret = 0;
        FOR (i, 0, m) if (left[i] != -1) ret += w[left[i]][i];
        return ret;
    }
}
```

虚树

```
void go(vector<int>& V, int& k) {
    int u = V[k]; f[u] = 0;
    dbg(u, k);
    for (auto& e: G[u]) {
        int v = e.to;
        if (v == pa[u][0]) continue;
        while (k + 1 < V.size()) {
            int to = V[k + 1];
            if (in[to] <= out[v]) {
                go(V, ++k);
                if (key[to]) f[u] += w[to];
                else f[u] += min(f[to], (LL)w[to]);
            } else break;
        }
    }
    dbg(u, f[u]);
}
inline bool cmp(int a, int b) { return in[a] < in[b]; }
LL solve(vector<int>& V) {
    static vector<int> a; a.clear();
    for (int& x: V) a.push_back(x);
    sort(a.begin(), a.end(), cmp);
    FOR (i, 1, a.size())
        a.push_back(lca(a[i], a[i - 1]));
    a.push_back(1);
    sort(a.begin(), a.end(), cmp);
    a.erase(unique(a.begin(), a.end()), a.end());
    dbg(a);
    int tmp; go(a, tmp = 0);
    return f[1];
}
```

计算几何

圆的反演

```
typedef double LD;
const LD PI = 3.14159265358979323846;
const LD eps = 1E-10;
const LD R2 = 1.0;
int sgn(LD x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); }
struct P {
    LD x, y;
    P(LD x = 0, LD y = 0): x(x), y(y) {}
    P operator * (LD k) { return P(x * k, y * k); }
    P operator / (LD k) { return P(x / k, y / k); }
    string prt() const {
        char s[100];
        sprintf(s, "(%.2f, %.2f)", x, y);
        return string(s);
    }
};
typedef P V;
P operator - (const P& a, const P& b) { return P(a.x - b.x, a.y - b.y); }
P operator + (const P& a, const P& b) { return P(a.x + b.x, a.y + b.y); }
struct C {
    P p;
    LD r;
    C(LD x = 0, LD y = 0, LD r = 0): p(x, y), r(r) {}
};
LD dist(V v) { return sqrt(v.x * v.x + v.y * v.y); }

C inv(C c, const P& o) {
    LD d = dist(c.p - o);
    assert(sgn(d) != 0);
    LD a = 1 / (d - c.r);
    LD b = 1 / (d + c.r);
    c.r = (a - b) / 2 * R2;
    c.p = o + (c.p - o) * ((a + b) * R2 / 2 / d);
    return c;
}
```

二维

- nxt宏要求多边形变量名为s
- L可隐式转换为V(P)
- 可以自定义结构体PP，可隐式转换为P

```
#define y1 yy1
#define nxt(i) ((i + 1) % s.size())
typedef double LD;
const LD PI = 3.14159265358979323846;
const LD eps = 1E-10;
int sgn(LD x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); }
struct L;
struct P;
//struct PP;
typedef P V;
struct P {
    LD x, y;
    explicit P(LD x = 0, LD y = 0): x(x), y(y) {}
    P(const L& l);
//    P(const PP& pp);
    string prt() const {
        char s[100];
        sprintf(s, "(%.2f, %.2f)", x, y);
        return string(s);
    }
};
struct L {
    P s, t;
    L() {}
    L(P s, P t): s(s), t(t) {}
};

P operator + (const P& a, const P& b) { return P(a.x + b.x, a.y + b.y); }
P operator - (const P& a, const P& b) { return P(a.x - b.x, a.y - b.y); }
P operator * (const P& a, LD k) { return P(a.x * k, a.y * k); }
P operator / (const P& a, LD k) { return P(a.x / k, a.y / k); }
bool operator == (const P& a, const P& b) { return !sgn(a.x - b.x) && !sgn(a.y - b.y); }
P::P(const L& l) { *this = l.t - l.s; }

// --------------------------------------------

//struct PP {
//    P p;
//    LD v, l;
//};
//P::P(const PP& pp) { *this = pp.p; }
typedef P PP;

typedef vector<PP> S;

// --------------------------------------------
LD dist(const P& p) { return sqrt(p.x * p.x + p.y * p.y); }
LD dot(const V& a, const V& b) { return a.x * b.x + a.y * b.y; }
LD det(const V& a, const V& b) { return a.x * b.y - a.y * b.x; }
LD cross(const P& s, const P& t, const P& o) { return det(s - o, t - o); }

// 如需支持 unique，需要加 eps
bool cmp_xy(const P& a, const P& b) { return a.x < b.x || a.x == b.x && a.y < b.y; }

// 象限
int quad(P p) {
    int x = sgn(p.x), y = sgn(p.y);
    if (x > 0 && y >= 0) return 1;
    if (x <= 0 && y > 0) return 2;
    if (x < 0 && y <= 0) return 3;
    if (x >= 0 && y < 0) return 4;
    assert(0);
}

// 仅适用于参照点在所有点一侧的情况
struct cmp_angle {
    P p;
    bool operator () (const P& a, const P& b) {
//        int qa = quad(a), qb = quad(b);
//        if (qa != qb) return qa < qb;
        int d = sgn(cross(a, b, p));
        if (d) return d > 0;
        return dist(a - p) < dist(b - p);
    }
};


// ----------------线----------------

// 是否平行
bool parallel(const L& a, const L& b) {
    return !sgn(det(a, b));
}
// 直线是否相等
bool l_eq(const L& a, const L& b) {
    return parallel(a, b) && parallel(L(a.s, b.t), L(b.s, a.t));
}
// 逆时针旋转 r 弧度
P rotation(const P& p, const LD& r) { return P(p.x * cos(r) - p.y * sin(r), p.x * sin(r) + p.y * cos(r)); }
// 单位法向量
V normal(const V& v) { return V(-v.y, v.x) / dist(v); }


// ----------------点和线----------------

// 点在线段上  <= 0 包含端点 < 0 则不包含
bool p_on_seg(const P& p, const L& seg) {
    P a = seg.s, b = seg.t;
    return !sgn(det(p - a, b - a)) && sgn(dot(p - a, p - b)) <= 0;
}
// 点到直线距离
LD dist_to_line(const P& p, const L& l) {
    return fabs(cross(l.s, l.t, p)) / dist(l);
}
// 点到线段距离
LD dist_to_seg(const P& p, const L& l) {
    if (l.s == l.t) return dist(p - l);
    V vs = p - l.s, vt = p - l.t;
    if (sgn(dot(l, vs)) < 0) return dist(vs);
    else if (sgn(dot(l, vt)) > 0) return dist(vt);
    else return dist_to_line(p, l);
}


// ----------------线和线----------------

// 求直线交 需要事先保证有界
P l_intersection(const L& a, const L& b) {
    LD s1 = det(a, b.s - a.s), s2 = det(a, b.t - a.s);
    return (b.s * s2 - b.t * s1) / (s2 - s1);
}
// 向量夹角的弧度
LD angle(const V& a, const V& b) {
    LD r = asin(fabs(det(a, b)) / dist(a) / dist(b));
    if (sgn(dot(a, b)) < 0) r = PI - r;
    return r;
}
// 线段和直线是否有交   1 = 规范, 2 = 不规范
int s_l_cross(const L& seg, const L& line) {
    int d1 = sgn(cross(line.s, line.t, seg.s));
    int d2 = sgn(cross(line.s, line.t, seg.t));
    if ((d1 ^ d2) == -2) return 1; // proper
    if (d1 == 0 || d2 == 0) return 2;
    return 0;
}
// 线段的交   1 = 规范, 2 = 不规范
int s_cross(const L& a, const L& b, P& p) {
    int d1 = sgn(cross(a.t, b.s, a.s)), d2 = sgn(cross(a.t, b.t, a.s));
    int d3 = sgn(cross(b.t, a.s, b.s)), d4 = sgn(cross(b.t, a.t, b.s));
    if ((d1 ^ d2) == -2 && (d3 ^ d4) == -2) { p = l_intersection(a, b); return 1; }
    if (!d1 && p_on_seg(b.s, a)) { p = b.s; return 2; }
    if (!d2 && p_on_seg(b.t, a)) { p = b.t; return 2; }
    if (!d3 && p_on_seg(a.s, b)) { p = a.s; return 2; }
    if (!d4 && p_on_seg(a.t, b)) { p = a.t; return 2; }
    return 0;
}


// ----------------多边形----------------
// 点是否在多边形中 0 = 在外部 1 = 在内部 -1 = 在边界上
int inside(const S& s, const P& p) {
    int cnt = 0;
    FOR (i, 0, s.size()) {
        P a = s[i], b = s[nxt(i)];
        if (p_on_seg(p, L(a, b))) return -1;
        if (sgn(a.y - b.y) <= 0) swap(a, b);
        if (sgn(p.y - a.y) > 0) continue;
        if (sgn(p.y - b.y) <= 0) continue;
        cnt += sgn(cross(b, a, p)) > 0;
    }
    return bool(cnt & 1);
}
// 多边形面积
LD polygon_area(const S& s) {
    LD ret = 0;
    FOR (i, 1, (LL)s.size() - 1)
        ret += cross(s[i], s[i + 1], s[0]);
    return ret / 2;
}
// 构建凸包 点不可以重复 < 0 边上可以有点， <= 0 则不能
// 会改变输入点的顺序
const int MAX_N = 1000;
S convex_hull(S& s) {
//    assert(s.size() >= 3);
    sort(s.begin(), s.end(), cmp_xy);
    S ret(MAX_N * 2);
    int sz = 0;
    FOR (i, 0, s.size()) {
        while (sz > 1 && sgn(cross(ret[sz - 1], s[i], ret[sz - 2])) < 0) --sz;
        ret[sz++] = s[i];
    }
    int k = sz;
    FORD (i, (LL)s.size() - 2, -1) {
        while (sz > k && sgn(cross(ret[sz - 1], s[i], ret[sz - 2])) < 0) --sz;
        ret[sz++] = s[i];
    }
    ret.resize(sz - (s.size() > 1));
    return ret;
}

// ----------------模板结束----------------
```

字符串

后缀自动机

- 广义后缀自动机如果直接使用以下代码的话会产生一些冗余状态（置last为1），所以要用拓扑排序。用len基数排序不能。
- 字符集大的话要使用 `map`。
- 树上dp时注意边界（root和null）。

```
namespace SAM {
    const int M = maxn << 1;
    int t[M][26], len[M] = {-1}, fa[M], sz = 2, last = 1;
    void ins(int ch) {
        int p = last, np = last = sz++;
        len[np] = len[p] + 1;
        for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
        if (!p) { fa[np] = 1; return; }
        int q = t[p][ch];
        if (len[p] + 1 == len[q]) fa[np] = q;
        else {
            int nq = sz++; len[nq] = len[p] + 1;
            memcpy(t[nq], t[q], sizeof t[0]);
            fa[nq] = fa[q];
            fa[np] = fa[q] = nq;
            for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
        }
    }

    int c[maxn] = {1}, a[M];
    void rsort() {
        FOR (i, 1, sz) c[i] = 0;
        FOR (i, 1, sz) c[len[i]]++;
        FOR (i, 1, maxn) c[i] += c[i - 1];
        FOR (i, 1, sz) a[--c[len[i]]] = i;
    }
}
```

- 真·广义后缀自动机

```
int t[M][26], len[M] = {-1}, fa[M], sz = 2, last = 1;
LL cnt[M][2];
void ins(int ch, int id) {
    int p = last, np = 0, nq = 0, q = -1;
    if (!t[p][ch]) {
        np = sz++;
        len[np] = len[p] + 1;
        for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
    }
    if (!p) fa[np] = 1;
    else {
        q = t[p][ch];
        if (len[p] + 1 == len[q]) fa[np] = q;
        else {
            nq = sz++; len[nq] = len[p] + 1;
            memcpy(t[nq], t[q], sizeof t[0]);
            fa[nq] = fa[q];
            fa[np] = fa[q] = nq;
            for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
        }
    }
    last = np ? np : nq ? nq : q;
    cnt[last][id] = 1;
}
```

- 按字典序建立后缀树注意逆序插入

```
void ins(int ch, int pp) {
    int p = last, np = last = sz++;
    len[np] = len[p] + 1; one[np] = pos[np] = pp;
    for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
    if (!p) { fa[np] = 1; return; }
    int q = t[p][ch];
    if (len[q] == len[p] + 1) fa[np] = q;
    else {
        int nq = sz++; len[nq] = len[p] + 1; one[nq] = one[q];
        t[nq] = t[q];
        fa[nq] = fa[q];
        fa[q] = fa[np] = nq;
        for (; p && t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
    }
}

int up[M], c[256] = {2}, a[M];
void rsort2() {
    FOR (i, 1, sz) c[i] = 0;
    FOR (i, 2, sz) up[i] = s[one[i] + len[fa[i]]];
    FOR (i, 2, sz) c[up[i]]++;
    FOR (i, 1, 256) c[i] += c[i - 1];
    FOR (i, 2, sz) a[--c[up[i]]] = i;
    FOR (i, 2, sz) G[fa[a[i]]].push_back(a[i]);
}
```

- 匹配

```
int u = 1, l = 0;
FOR (i, 0, strlen(s)) {
    int ch = s[i] - 'a';
    while (u && !t[u][ch]) { u = fa[u]; l = len[u]; }
    ++l; u = t[u][ch];
    if (!u) u = 1;
    // do something...
}
```

- 获取子串状态

```
int get_state(int l, int r) {
    int u = rpos[r], s = r - l + 1;
    FORD (i, SP - 1, -1) if (len[pa[u][i]] >= s) u = pa[u][i];
    return u;
}
```

回文自动机

```
namespace pam {
    int t[maxn][26], fa[maxn], len[maxn], rs[maxn], num[maxn], cnt[maxn];
    int sz, n, last;
    int N(int l) {
        memset(t[sz], 0, sizeof t[0]);
        len[sz] = l;
        return sz++;
    }
    void init() {
        rs[n = sz = 0] = -1;
        last = N(0);
        fa[last] = N(-1);
    }
    int get_fa(int x) {
        while (rs[n - 1 - len[x]] != rs[n]) x = fa[x];
        return x;
    }
    void ins(int ch) {
        rs[++n] = ch;
        int p = get_fa(last);
        if (!t[p][ch]) {
            int np = N(len[p] + 2);
            fa[np] = t[get_fa(fa[p])][ch];
            num[np] = num[fa[np]] + 1;
            t[p][ch] = np;
            cnt[np] = 1;
        }
        last = t[p][ch];
    }
    void get_cnt() { FOR (i, 2, sz) cnt[fa[i]] += cnt[i]; }
}
```

杂项

STL

- copy

```
template <class InputIterator, class OutputIterator>
  OutputIterator copy (InputIterator first, InputIterator last, OutputIterator result);
```

- merge (如果相等，第一个优先)

```
template <class InputIterator1, class InputIterator2,
          class OutputIterator, class Compare>
  OutputIterator merge (InputIterator1 first1, InputIterator1 last1,
                        InputIterator2 first2, InputIterator2 last2,
                        OutputIterator result, Compare comp);
```

- for_each

```
template <class InputIterator, class Function>
   Function for_each (InputIterator first, InputIterator last, Function fn);
```

- transform

```
template <class InputIterator, class OutputIterator, class UnaryOperation>
  OutputIterator transform (InputIterator first1, InputIterator last1,
                            OutputIterator result, UnaryOperation op);
```

- numeric_limits

```
template <class T> numeric_limits;
```

- iota

```
template< class ForwardIterator, class T >
void iota( ForwardIterator first, ForwardIterator last, T value );
```

伪随机数

```
unsigned rnd() {
    static unsigned A = 1 << 16 | 3, B = 33333331, C = 2341;
    return C = A * C + B;
}
```

日期

```
// Routines for performing computations on dates.  In these routines,
// months are exprsesed as integers from 1 to 12, days are expressed
// as integers from 1 to 31, and years are expressed as 4-digit
// integers.

string dayOfWeek[] = {"Mo", "Tu", "We", "Th", "Fr", "Sa", "Su"};

// converts Gregorian date to integer (Julian day number)

int DateToInt (int m, int d, int y){
  return
    1461 * (y + 4800 + (m - 14) / 12) / 4 +
    367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
    3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
    d - 32075;
}

// converts integer (Julian day number) to Gregorian date: month/day/year

void IntToDate (int jd, int &m, int &d, int &y){
  int x, n, i, j;

  x = jd + 68569;
  n = 4 * x / 146097;
  x -= (146097 * n + 3) / 4;
  i = (4000 * (x + 1)) / 1461001;
  x -= 1461 * i / 4 - 31;
  j = 80 * x / 2447;
  d = x - 2447 * j / 80;
  x = j / 11;
  m = j + 2 - 12 * x;
  y = 100 * (n - 49) + i + x;
}

// converts integer (Julian day number) to day of week

string IntToDay (int jd){
  return dayOfWeek[jd % 7];
}
```

子集枚举

- 枚举真子集

```
for (int s = (S - 1) & S; s; s = (s - 1) & S)
```

- 枚举大小为 k 的集合

```
template<typename T>
void subset(int k, int n, T&& f) {
    int t = (1 << k) - 1;
    while (t < 1 << n) {
        f(t);
        int x = t & -t, y = t + x;
        t = ((t & ~y) / x >> 1) | y;
    }
}
```

权值最大上升子序列

```
const LL maxn = 1E5 + 10;
const LL INF = 1E10;
struct P {
    LL k, v;
    bool operator < (const P& rhs) const {
        return k < rhs.k || (k == rhs.k && v < rhs.v);
    }
};
LL k[maxn], v[maxn], n, T;
set<P> s;

int main() {
    cin >> T;
    while (T--) {
        s.clear();
        s.insert({-INF, 0});
        cin >> n;
        FOR (i, 0, n) scanf("%lld", &k[i]);
        FOR (i, 0, n) scanf("%lld", &v[i]);
        FOR (i, 0, n) {
            auto it = s.lower_bound({k[i], INF});
            LL vv = (--it)->v + v[i];
            ++it;
            while (it != s.end() && it->v <= vv)
                it = s.erase(it);
            if (it == s.end() || it->k != k[i]) s.insert({k[i], vv});
        }
        cout << s.rbegin()->v << endl;
    }
}
```

数位 DP

```
LL dfs(LL base, LL pos, LL len, LL s, bool limit) {
    if (pos == -1) return s ? base : 1;
    if (!limit && dp[base][pos][len][s] != -1) return dp[base][pos][len][s];
    LL ret = 0;
    LL ed = limit ? a[pos] : base - 1;
    FOR (i, 0, ed + 1) {
        tmp[pos] = i;
        if (len == pos)
            ret += dfs(base, pos - 1, len - (i == 0), s, limit && i == a[pos]);
        else if (s &&pos < (len + 1) / 2)
            ret += dfs(base, pos - 1, len, tmp[len - pos] == i, limit && i == a[pos]);
        else
            ret += dfs(base, pos - 1, len, s, limit && i == a[pos]);
    }
    if (!limit) dp[base][pos][len][s] = ret;
    return ret;
}

LL solve(LL x, LL base) {
    LL sz = 0;
    while (x) {
        a[sz++] = x % base;
        x /= base;
    }
    return dfs(base, sz - 1, sz - 1, 1, true);
}
```

心态崩了

- `(int)v.size()`
- `1LL << k`
- 递归函数用全局或者 static 变量要小心
- 预处理组合数注意上限
- 想清楚到底是要 `multiset` 还是 `set`
- 提交之前看一下数据范围，测一下边界
- 数据结构注意数组大小（2 倍，4 倍）
- 字符串注意数据集
- 如果函数中使用了默认参数的话，注意调用时的参数个数。
- 注意要读完
- 构造参数无法使用自己
- 树链剖分询问不要忘记 idx, ridx
- 排序时注意结构体的所有属性是不是考虑了
- 不要把 while 写成 if