

Contents

一切的开始	3
宏定义	3
快速读	3
对拍	4
为什么 C++ 不自带这个?	5
数据结构	5
ST 表	5
线段树	5
均摊复杂度线段树	7
持久化线段树	8
K-D Tree	9
树状数组	11
主席树	13
左偏树	15
Treap	16
Treap-序列	18
可回滚并查集	21
舞蹈链	22
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	31
矩阵运算	31
筛	31
亚线性筛	32
min_25	32
杜教筛	33
素数测试	34
线性递推	34
扩展欧几里得	35
类欧几里得	36
逆元	36
组合数	36
第二类斯特灵数	37
simpson 自适应积分	38
快速乘	39
快速幂	39
高斯消元	40
质因数分解	41
原根	42
公式	42
中国剩余定理	42
伯努利数和等幂求和	43
单纯形	43
图论	44
LCA	44
最短路	44
网络流	45
树上路径交	48

树上点分治	48
树链剖分	51
二分图匹配	55
虚树	56
计算几何	57
圆的反演	57
二维	57
字符串	60
后缀自动机	60
回文自动机	64
哈希	64
后缀数组	66
KMP 自动机	69
Trie	69
杂项	70
STL	70
伪随机数	70
日期	70
子集枚举	71
权值最大上升子序列	71
数位 DP	72
土制 bitset	72
扩栈（本地使用）	73
心态崩了	73

一切的开始

宏定义

- 需要 C++11

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using LL = long long;
4 #define FOR(i, x, y) for (decay<decltype(y)>::type i = (x), _##i = (y); i < _##i; ++i)
5 #define FORD(i, x, y) for (decay<decltype(x)>::type i = (x), _##i = (y); i > _##i; --i)
6 #ifdef zero1
7 #define dbg(args...) do { cout << "\033[32;1m" << #args << " -> "; err(args); } while (0)
8 #else
9 #define dbg(...)
10 #endif
11 void err() { cout << "\033[39;0m" << endl; }
12 template<typename...> class T, typename t, typename... Args>
13 void err(T<t> a, Args... args) { for (auto x: a) cout << x << ' '; err(args...); }
14 template<typename T, typename... Args>
15 void err(T a, Args... args) { cout << a << ' '; err(args...); }
16 // -----
```

- 更多配色:

- 33 黄色
- 34 蓝色
- 31 橙色

- POJ/BZOJ version

```
1 #include <cstdio>
2 #include <iostream>
3 #include <algorithm>
4 #include <cmath>
5 #include <string>
6 #include <vector>
7 #include <set>
8 #include <queue>
9 #include <cstring>
10 #include <cassert>
11 using namespace std;
12 typedef long long LL;
13 #define FOR(i, x, y) for (LL i = (x), _##i = (y); i < _##i; ++i)
14 #define FORD(i, x, y) for (LL i = (x), _##i = (y); i > _##i; --i)
15 #ifdef zero1
16 #define dbg(args...) do { cout << "\033[32;1m" << #args << " -> "; err(args); } while (0)
17 #else
18 #define dbg(...)
19 #endif
20 void err() { cout << "\033[39;0m" << endl; }
21 template<typename T, typename... Args>
22 void err(T a, Args... args) {
23     cout << a << ' '; err(args...);
24 }
25 // -----
```

- HDU Assert Patch

```
1 #ifdef ONLINE_JUDGE
2 #define assert(condition) if (!(condition)) { int x = 1, y = 0; cout << x / y << endl; }
3 #endif
```

快速读

```
1 inline char next_char() {
2     static char buf[100000], *p1 = buf, *p2 = buf;
3     return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 100000, stdin), p1 == p2) ? EOF : *p1++;
4 }
5 inline bool maybe_digit(char c) {
```

```

6     return c >= '0' && c <= '9';
7 }
8 template <typename T>
9 void rn(T& _v) {
10     static char ch;
11     static bool negative = false;
12     _v = 0;
13     while (!maybe_digit(ch)) {
14         negative = ch == '-';
15         ch = next_char();
16     }
17     do _v = (_v << 1) + (_v << 3) + ch - '0';
18     while (maybe_digit(ch = next_char()));
19     if (negative) _v = -_v;
20 }
21
22 template <typename T>
23 void o(T p) {
24     static int stk[70], tp;
25     if (p == 0) {
26         putchar('0');
27         return;
28     }
29     if (p < 0) { p = -p; putchar('-'); }
30     while (p) stk[++tp] = p % 10, p /= 10;
31     while (tp) putchar(stk[tp--] + '0');
32 }

```

- 需要初始化
- 需要一次读入
- 不支持负数

```

1 const int MAXS = 100 * 1024 * 1024;
2 char buf[MAXS];
3 template<typename T>
4 inline bool read(T& x) {
5     static char* p = buf;
6     x = 0;
7     while (*p && !isdigit(*p)) ++p;
8     if (!*p) return false;
9     while (isdigit(*p)) x = x * 10 + *p++ - 48;
10    return true;
11 }
12
13 fread(buf, 1, MAXS, stdin);

```

对拍

```

1 #!/usr/bin/env bash
2 g++ -o r main.cpp -O2 -std=c++11
3 g++ -o std std.cpp -O2 -std=c++11
4 while true; do
5     python gen.py > in
6     ./std < in > stdout
7     ./r < in > out
8     if test $? -ne 0; then
9         exit 0
10    fi
11    if diff stdout out; then
12        printf "AC\n"
13    else
14        printf "GG\n"
15        exit 0
16    fi
17 done

```

为什么 C++ 不自带这个？

```
1 LL bin(LL x, LL n, LL MOD) {
2     LL ret = MOD != 1;
3     for (x %= MOD; n; n >>= 1, x = x * x % MOD)
4         if (n & 1) ret = ret * x % MOD;
5     return ret;
6 }
7 inline LL get_inv(LL x, LL p) { return bin(x, p - 2, p); }
```

数据结构

ST 表

● 二维

```
1 int f[maxn][maxn][10][10];
2 inline int highbit(int x) { return 31 - __builtin_clz(x); }
3 inline int calc(int x, int y, int xx, int yy, int p, int q) {
4     return max(
5         max(f[x][y][p][q], f[xx - (1 << p) + 1][yy - (1 << q) + 1][p][q]),
6         max(f[xx - (1 << p) + 1][y][p][q], f[x][yy - (1 << q) + 1][p][q])
7     );
8 }
9 void init() {
10     FOR (x, 0, highbit(n) + 1)
11     FOR (y, 0, highbit(m) + 1)
12     FOR (i, 0, n - (1 << x) + 1)
13     FOR (j, 0, m - (1 << y) + 1) {
14         if (!x && !y) { f[i][j][x][y] = a[i][j]; continue; }
15         f[i][j][x][y] = calc(
16             i, j,
17             i + (1 << x) - 1, j + (1 << y) - 1,
18             max(x - 1, 0), max(y - 1, 0)
19         );
20     }
21 }
22 inline int get_max(int x, int y, int xx, int yy) {
23     return calc(x, y, xx, yy, highbit(xx - x + 1), highbit(yy - y + 1));
24 }
```

● 一维

```
1 struct RMQ {
2     int f[22][M];
3     inline int highbit(int x) { return 31 - __builtin_clz(x); }
4     void init(int* v, int n) {
5         FOR (i, 0, n) f[0][i] = v[i];
6         FOR (x, 1, highbit(n) + 1)
7             FOR (i, 0, n - (1 << x) + 1)
8                 f[x][i] = min(f[x - 1][i], f[x - 1][i + (1 << (x - 1))]);
9     }
10    int get_min(int l, int r) {
11        assert(l <= r);
12        int t = highbit(r - l + 1);
13        return min(f[t][l], f[t][r - (1 << t) + 1]);
14    }
15 } rmq;
```

线段树

● 普适

```
1 namespace sg {
2     struct Q {
3         LL setv;
```

```

4         explicit Q(LL setv = -1): setv(setv) {}
5         void operator += (const Q& q) { if (q.setv != -1) setv = q.setv; }
6     };
7     struct P {
8         LL min;
9         explicit P(LL min = INF): min(min) {}
10        void up(Q& q) { if (q.setv != -1) min = q.setv; }
11    };
12    template<typename T>
13    P operator & (T&& a, T&& b) {
14        return P(min(a.min, b.min));
15    }
16    P p[maxn << 2];
17    Q q[maxn << 2];
18    #define lson o * 2, l, (l + r) / 2
19    #define rson o * 2 + 1, (l + r) / 2 + 1, r
20    void up(int o, int l, int r) {
21        if (l == r) p[o] = P();
22        else p[o] = p[o * 2] & p[o * 2 + 1];
23        p[o].up(q[o]);
24    }
25    void down(int o, int l, int r) {
26        q[o * 2] += q[o]; q[o * 2 + 1] += q[o];
27        q[o] = Q();
28        up(lson); up(rson);
29    }
30    template<typename T>
31    void build(T&& f, int o = 1, int l = 1, int r = n) {
32        if (l == r) q[o] = f(l);
33        else { build(f, lson); build(f, rson); q[o] = Q(); }
34        up(o, l, r);
35    }
36    P query(int ql, int qr, int o = 1, int l = 1, int r = n) {
37        if (ql > r || l > qr) return P();
38        if (ql <= l && r <= qr) return p[o];
39        down(o, l, r);
40        return query(ql, qr, lson) & query(ql, qr, rson);
41    }
42    void update(int ql, int qr, const Q& v, int o = 1, int l = 1, int r = n) {
43        if (ql > r || l > qr) return;
44        if (ql <= l && r <= qr) q[o] += v;
45        else {
46            down(o, l, r);
47            update(ql, qr, v, lson); update(ql, qr, v, rson);
48        }
49        up(o, l, r);
50    }
51 }

```

• SET+ADD

```

1     struct IntervalTree {
2         #define ls o * 2, l, m
3         #define rs o * 2 + 1, m + 1, r
4         static const LL M = maxn * 4, RS = 1E18 - 1;
5         LL addv[M], setv[M], minv[M], maxv[M], sumv[M];
6         void init() {
7             memset(addv, 0, sizeof addv);
8             fill(setv, setv + M, RS);
9             memset(minv, 0, sizeof minv);
10            memset(maxv, 0, sizeof maxv);
11            memset(sumv, 0, sizeof sumv);
12        }
13        void maintain(LL o, LL l, LL r) {
14            if (l < r) {
15                LL lc = o * 2, rc = o * 2 + 1;
16                sumv[o] = sumv[lc] + sumv[rc];
17                minv[o] = min(minv[lc], minv[rc]);
18                maxv[o] = max(maxv[lc], maxv[rc]);
19            } else sumv[o] = minv[o] = maxv[o] = 0;
20            if (setv[o] != RS) { minv[o] = maxv[o] = setv[o]; sumv[o] = setv[o] * (r - l + 1); }
21            if (addv[o]) { minv[o] += addv[o]; maxv[o] += addv[o]; sumv[o] += addv[o] * (r - l + 1); }

```

```

22     }
23     void build(LL o, LL l, LL r) {
24         if (l == r) addv[o] = a[l];
25         else {
26             LL m = (l + r) / 2;
27             build(ls); build(rs);
28         }
29         maintain(o, l, r);
30     }
31     void pushdown(LL o) {
32         LL lc = o * 2, rc = o * 2 + 1;
33         if (setv[o] != RS) {
34             setv[lc] = setv[rc] = setv[o];
35             addv[lc] = addv[rc] = 0;
36             setv[o] = RS;
37         }
38         if (addv[o]) {
39             addv[lc] += addv[o]; addv[rc] += addv[o];
40             addv[o] = 0;
41         }
42     }
43     void update(LL p, LL q, LL o, LL l, LL r, LL v, LL op) {
44         if (p <= r && l <= q)
45             if (p <= l && r <= q) {
46                 if (op == 2) { setv[o] = v; addv[o] = 0; }
47                 else addv[o] += v;
48             } else {
49                 pushdown(o);
50                 LL m = (l + r) / 2;
51                 update(p, q, ls, v, op); update(p, q, rs, v, op);
52             }
53         maintain(o, l, r);
54     }
55     void query(LL p, LL q, LL o, LL l, LL r, LL add, LL& ssum, LL& smin, LL& smax) {
56         if (p > r || l > q) return;
57         if (setv[o] != RS) {
58             LL v = setv[o] + add + addv[o];
59             ssum += v * (min(r, q) - max(l, p) + 1);
60             smin = min(smin, v);
61             smax = max(smax, v);
62         } else if (p <= l && r <= q) {
63             ssum += sumv[o] + add * (r - l + 1);
64             smin = min(smin, minv[o] + add);
65             smax = max(smax, maxv[o] + add);
66         } else {
67             LL m = (l + r) / 2;
68             query(p, q, ls, add + addv[o], ssum, smin, smax);
69             query(p, q, rs, add + addv[o], ssum, smin, smax);
70         }
71     }
72 } IT;

```

均摊复杂度线段树

- 区间取 max, 区间求和。

```

1 namespace R {
2     #define lson o * 2, l, (l + r) / 2
3     #define rson o * 2 + 1, (l + r) / 2 + 1, r
4     int m1[N], m2[N], cm1[N];
5     LL sum[N];
6     void up(int o) {
7         int lc = o * 2, rc = lc + 1;
8         m1[o] = max(m1[lc], m1[rc]);
9         sum[o] = sum[lc] + sum[rc];
10        if (m1[lc] == m1[rc]) {
11            cm1[o] = cm1[lc] + cm1[rc];
12            m2[o] = max(m2[lc], m2[rc]);
13        } else {

```

```

14         cm1[o] = m1[lc] > m1[rc] ? cm1[lc] : cm1[rc];
15         m2[o] = max(min(m1[lc], m1[rc]), max(m2[lc], m2[rc]));
16     }
17 }
18 void mod(int o, int x) {
19     if (x >= m1[o]) return;
20     assert(x > m2[o]);
21     sum[o] -= 1LL * (m1[o] - x) * cm1[o];
22     m1[o] = x;
23 }
24 void down(int o) {
25     int lc = o * 2, rc = lc + 1;
26     mod(lc, m1[o]); mod(rc, m1[o]);
27 }
28 void build(int o, int l, int r) {
29     if (l == r) { int t; read(t); sum[o] = m1[o] = t; m2[o] = -1; cm1[o] = 1; }
30     else { build(lson); build(rson); up(o); }
31 }
32 void update(int ql, int qr, int x, int o, int l, int r) {
33     if (r < ql || qr < l || m1[o] <= x) return;
34     if (ql <= l && r <= qr && m2[o] < x) { mod(o, x); return; }
35     down(o);
36     update(ql, qr, x, lson); update(ql, qr, x, rson);
37     up(o);
38 }
39 int qmax(int ql, int qr, int o, int l, int r) {
40     if (r < ql || qr < l) return -INF;
41     if (ql <= l && r <= qr) return m1[o];
42     down(o);
43     return max(qmax(ql, qr, lson), qmax(ql, qr, rson));
44 }
45 LL qsum(int ql, int qr, int o, int l, int r) {
46     if (r < ql || qr < l) return 0;
47     if (ql <= l && r <= qr) return sum[o];
48     down(o);
49     return qsum(ql, qr, lson) + qsum(ql, qr, rson);
50 }
51 }

```

持久化线段树

• ADD

```

1 namespace tree {
2     #define mid ((l + r) >> 1)
3     #define lson ql, qr, l, mid
4     #define rson ql, qr, mid + 1, r
5     struct P {
6         LL add, sum;
7         int ls, rs;
8     } tr[maxn * 45 * 2];
9     int sz = 1;
10    int N(LL add, int l, int r, int ls, int rs) {
11        tr[sz] = {add, tr[ls].sum + tr[rs].sum + add * (len[r] - len[l - 1]), ls, rs};
12        return sz++;
13    }
14    int update(int o, int ql, int qr, int l, int r, LL add) {
15        if (ql > r || l > qr) return o;
16        const P& t = tr[o];
17        if (ql <= l && r <= qr) return N(add + t.add, l, r, t.ls, t.rs);
18        return N(t.add, l, r, update(t.ls, lson, add), update(t.rs, rson, add));
19    }
20    LL query(int o, int ql, int qr, int l, int r, LL add = 0) {
21        if (ql > r || l > qr) return 0;
22        const P& t = tr[o];
23        if (ql <= l && r <= qr) return add * (len[r] - len[l - 1]) + t.sum;
24        return query(t.ls, lson, add + t.add) + query(t.rs, rson, add + t.add);
25    }
26 }

```


K-D Tree

最优化问题一定要用全局变量大力剪枝，而且左右儿子先递归潜力大的

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

```
1 namespace kd {
2     const int K = 2, inf = 1E9, M = N;
3     const double lim = 0.7;
4     struct P {
5         int d[K], l[K], r[K], sz, val;
6         LL sum;
7         P *ls, *rs;
8         P* up() {
9             sz = ls->sz + rs->sz + 1;
10            sum = ls->sum + rs->sum + val;
11            FOR (i, 0, K) {
12                l[i] = min(d[i], min(ls->l[i], rs->l[i]));
13                r[i] = max(d[i], max(ls->r[i], rs->r[i]));
14            }
15            return this;
16        }
17    } pool[M], *null = new P, *pit = pool;
18    static P *tmp[M], **pt;
19    void init() {
20        null->ls = null->rs = null;
21        FOR (i, 0, K) null->l[i] = inf, null->r[i] = -inf;
22        null->sum = null->val = 0;
23        null->sz = 0;
24    }
25
26    P* build(P** l, P** r, int d = 0) { // [l, r)
27        if (d == K) d = 0;
28        if (l >= r) return null;
29        P** m = l + (r - l) / 2; assert(l <= m && m < r);
30        nth_element(l, m, r, [&](const P* a, const P* b){
31            return a->d[d] < b->d[d];
32        });
33        P* o = *m;
34        o->ls = build(l, m, d + 1); o->rs = build(m + 1, r, d + 1);
35        return o->up();
36    }
37    P* Build() {
38        pt = tmp; FOR (it, pool, pit) *pt++ = it;
39        return build(tmp, pt);
40    }
41    inline bool inside(int p[], int q[], int l[], int r[]) {
42        FOR (i, 0, K) if (r[i] < q[i] || p[i] < l[i]) return false;
43        return true;
44    }
45    LL query(P* o, int l[], int r[]) {
46        if (o == null) return 0;
47        FOR (i, 0, K) if (o->r[i] < l[i] || r[i] < o->l[i]) return 0;
48        if (inside(o->l, o->r, l, r)) return o->sum;
49        return query(o->ls, l, r) + query(o->rs, l, r) +
50            (inside(o->d, o->d, l, r) ? o->val : 0);
51    }
52    void dfs(P* o) {
53        if (o == null) return;
54        *pt++ = o; dfs(o->ls); dfs(o->rs);
55    }
56    P* ins(P* o, P* x, int d = 0) {
57        if (d == K) d = 0;
58        if (o == null) return x->up();
59        P*& oo = x->d[d] <= o->d[d] ? o->ls : o->rs;
60        if (oo->sz > o->sz * lim) {
61            pt = tmp; dfs(o); *pt++ = x;
62            return build(tmp, pt, d);
63        }
64    }
```

```

64         oo = ins(oo, x, d + 1);
65         return o->up();
66     }
67 }

```

- 维护信息
- 带修改 (适合离线)

```

1  namespace kd {
2      const int K = 3, inf = 1E9, M = N << 3;
3      extern struct P* null;
4      struct P {
5          int d[K], l[K], r[K], val;
6          int Max;
7          P *ls, *rs, *fa;
8          P* up() {
9              Max = max(val, max(ls->Max, rs->Max));
10             FOR (i, 0, K) {
11                 l[i] = min(d[i], min(ls->l[i], rs->l[i]));
12                 r[i] = max(d[i], max(ls->r[i], rs->r[i]));
13             }
14             return ls->fa = rs->fa = this;
15         }
16     } pool[M], *null = new P, *pit = pool;
17     void upd(P* o, int val) {
18         o->val = val;
19         for (; o != null; o = o->fa)
20             o->Max = max(o->Max, val);
21     }
22     static P *tmp[M], **pt;
23     void init() {
24         null->ls = null->rs = null;
25         FOR (i, 0, K) null->l[i] = inf, null->r[i] = -inf;
26         null->Max = null->val = 0;
27     }
28     P* build(P** l, P** r, int d = 0) { // [l, r)
29         if (d == K) d = 0;
30         if (l >= r) return null;
31         P** m = l + (r - l) / 2; assert(l <= m && m < r);
32         nth_element(l, m, r, [&](const P* a, const P* b){
33             return a->d[d] < b->d[d];
34         });
35         P* o = *m;
36         o->ls = build(l, m, d + 1); o->rs = build(m + 1, r, d + 1);
37         return o->up();
38     }
39     P* Build() {
40         pt = tmp; FOR (it, pool, pit) *pt++ = it;
41         P* ret = build(tmp, pt); ret->fa = null;
42         return ret;
43     }
44     inline bool inside(int p[], int q[], int l[], int r[]) {
45         FOR (i, 0, K) if (r[i] < q[i] || p[i] < l[i]) return false;
46         return true;
47     }
48     int query(P* o, int l[], int r[]) {
49         if (o == null) return 0;
50         FOR (i, 0, K) if (o->r[i] < l[i] || r[i] < o->l[i]) return 0;
51         if (inside(o->l, o->r, l, r)) return o->Max;
52         int ret = 0;
53         if (o->val > ret && inside(o->d, o->d, l, r)) ret = max(ret, o->val);
54         if (o->ls->Max > ret) ret = max(ret, query(o->ls, l, r));
55         if (o->rs->Max > ret) ret = max(ret, query(o->rs, l, r));
56         return ret;
57     }
58 }

```

- 最近点对
- 要用全局变量大力剪枝

```

1  namespace kd {
2      const int K = 3;

```

```

3  const int M = N;
4  const int inf = 1E9 + 100;
5  struct P {
6      int d[K];
7      int l[K], r[K];
8      P *ls, *rs;
9      P* up() {
10         FOR (i, 0, K) {
11             l[i] = min(d[i], min(ls->l[i], rs->l[i]));
12             r[i] = max(d[i], max(ls->r[i], rs->r[i]));
13         }
14         return this;
15     }
16 } pool[M], *null = new P, *pit = pool;
17 static P *tmp[M], **pt;
18 void init() {
19     null->ls = null->rs = null;
20     FOR (i, 0, K) null->l[i] = inf, null->r[i] = -inf;
21 }
22 P* build(P** l, P** r, int d = 0) { // [l, r]
23     if (d == K) d = 0;
24     if (l >= r) return null;
25     P** m = l + (r - l) / 2;
26     nth_element(l, m, r, [&](const P* a, const P* b){
27         return a->d[d] < b->d[d];
28     });
29     P* o = *m;
30     o->ls = build(l, m, d + 1); o->rs = build(m + 1, r, d + 1);
31     return o->up();
32 }
33 LL eval(P* o, int d[]) {
34     // ...
35 }
36 LL dist(int d1[], int d2[]) {
37     // ...
38 }
39 LL S;
40 LL query(P* o, int d[]) {
41     if (o == null) return 0;
42     S = max(S, dist(o->d, d));
43     LL mdl = eval(o->ls, d), mdr = eval(o->rs, d);
44     if (mdl < mdr) {
45         if (S > mdl) S = max(S, query(o->ls, d));
46         if (S > mdr) S = max(S, query(o->rs, d));
47     } else {
48         if (S > mdr) S = max(S, query(o->rs, d));
49         if (S > mdl) S = max(S, query(o->ls, d));
50     }
51     return S;
52 }
53 P* Build() {
54     pt = tmp; FOR (it, pool, pit) *pt++ = it;
55     return build(tmp, pt);
56 }
57 }

```

树状数组

- 注意: 0 是无效下标

```

1  namespace bit {
2      LL c[M];
3      inline int lowbit(int x) { return x & -x; }
4      void add(int x, LL v) {
5          for (; x < M; x += lowbit(x))
6              c[x] += v;
7      }
8      LL sum(int x) {
9          LL ret = 0;

```

```

10     for (; x > 0; x -= lowbit(x))
11         ret += c[x];
12     return ret;
13 }
14 int kth(LL k) {
15     int ret = 0;
16     LL cnt = 0;
17     FORD (i, 20, -1) {
18         ret += 1 << i;
19         if (ret >= M || cnt + c[ret] >= k)
20             ret -= 1 << i;
21         else cnt += c[ret];
22     }
23     return ret + 1;
24 }
25 }

```

- 区间修改 & 区间查询 (单点修改, 查询前缀和的前缀和)

```

1 namespace bit {
2     int c[maxn], cc[maxn];
3     inline int lowbit(int x) { return x & -x; }
4     void add(int x, int v) {
5         for (int i = x; i <= n; i += lowbit(i)) {
6             c[i] += v; cc[i] += x * v;
7         }
8     }
9     void add(int l, int r, int v) { add(l, v); add(r + 1, -v); }
10    int sum(int x) {
11        int ret = 0;
12        for (int i = x; i > 0; i -= lowbit(i))
13            ret += (x + 1) * c[i] - cc[i];
14        return ret;
15    }
16    int sum(int l, int r) { return sum(r) - sum(l - 1); }
17 }

```

- 单点修改, 查询前缀和的前缀和的前缀和 (有用才怪)

```

1 namespace bit {
2     LL c[N], cc[N], ccc[N];
3     inline LL lowbit(LL x) { return x & -x; }
4     void add(LL x, LL v) {
5         for (LL i = x; i < N; i += lowbit(i)) {
6             c[i] = (c[i] + v) % MOD;
7             cc[i] = (cc[i] + x * v) % MOD;
8             ccc[i] = (ccc[i] + x * x % MOD * v) % MOD;
9         }
10    }
11    void add(LL l, LL r, LL v) { add(l, v); add(r + 1, -v); }
12    LL sum(LL x) {
13        static LL INV2 = (MOD + 1) / 2;
14        LL ret = 0;
15        for (LL i = x; i > 0; i -= lowbit(i))
16            ret += (x + 1) * (x + 2) % MOD * c[i] % MOD
17                - (2 * x + 3) * cc[i] % MOD
18                + ccc[i];
19        return ret % MOD * INV2 % MOD;
20    }
21    LL sum(LL l, LL r) { return sum(r) - sum(l - 1); }
22 }

```

- 三维

```

1 inline int lowbit(int x) { return x & -x; }
2 void update(int x, int y, int z, int d) {
3     for (int i = x; i <= n; i += lowbit(i))
4         for (int j = y; j <= n; j += lowbit(j))
5             for (int k = z; k <= n; k += lowbit(k))
6                 c[i][j][k] += d;
7 }
8 LL query(int x, int y, int z) {
9     LL ret = 0;

```

```

10     for (int i = x; i > 0; i -= lowbit(i))
11         for (int j = y; j > 0; j -= lowbit(j))
12             for (int k = z; k > 0; k -= lowbit(k))
13                 ret += c[i][j][k];
14     return ret;
15 }
16 LL solve(int x, int y, int z, int xx, int yy, int zz) {
17     return
18         query(xx, yy, zz)
19         - query(xx, yy, z - 1)
20         - query(xx, y - 1, zz)
21         - query(x - 1, yy, zz)
22         + query(xx, y - 1, z - 1)
23         + query(x - 1, yy, z - 1)
24         + query(x - 1, y - 1, zz)
25         - query(x - 1, y - 1, z - 1);

```

主席树

● 正常主席树

```

1 namespace tree {
2     #define mid ((l + r) >> 1)
3     #define lson l, mid
4     #define rson mid + 1, r
5     const int MAGIC = M * 30;
6     struct P {
7         int sum, ls, rs;
8     } tr[MAGIC] = {{0, 0, 0}};
9     int sz = 1;
10    int N(int sum, int ls, int rs) {
11        if (sz == MAGIC) assert(0);
12        tr[sz] = {sum, ls, rs};
13        return sz++;
14    }
15    int ins(int o, int x, int v, int l = 1, int r = ls) {
16        if (x < l || x > r) return o;
17        const P& t = tr[o];
18        if (l == r) return N(t.sum + v, 0, 0);
19        return N(t.sum + v, ins(t.ls, x, v, lson), ins(t.rs, x, v, rson));
20    }
21    int query(int o, int ql, int qr, int l = 1, int r = ls) {
22        if (ql > r || l > qr) return 0;
23        const P& t = tr[o];
24        if (ql <= l && r <= qr) return t.sum;
25        return query(t.ls, ql, qr, lson) + query(t.rs, ql, qr, rson);
26    }
27 }

```

● 第k大

```

1 struct TREE {
2     #define mid ((l + r) >> 1)
3     #define lson l, mid
4     #define rson mid + 1, r
5     struct P {
6         int w, ls, rs;
7     } tr[maxn * 20];
8     int sz = 1;
9     TREE() { tr[0] = {0, 0, 0}; }
10    int N(int w, int ls, int rs) {
11        tr[sz] = {w, ls, rs};
12        return sz++;
13    }
14    int ins(int tt, int l, int r, int x) {
15        if (x < l || r < x) return tt;
16        const P& t = tr[tt];
17        if (l == r) return N(t.w + 1, 0, 0);
18        return N(t.w + 1, ins(t.ls, lson, x), ins(t.rs, rson, x));
19    }
20    int query(int pp, int qq, int l, int r, int k) { // (pp, qq]

```

```

21     if (l == r) return l;
22     const P &p = tr[pp], &q = tr[qq];
23     int w = tr[q.ls].w - tr[p.ls].w;
24     if (k <= w) return query(p.ls, q.ls, lson, k);
25     else return query(p.rs, q.rs, rson, k - w);
26 }
27 } tree;

```

● 树状数组套主席树

```

1  typedef vector<int> VI;
2  struct TREE {
3      #define mid ((l + r) >> 1)
4      #define lson l, mid
5      #define rson mid + 1, r
6      struct P {
7          int w, ls, rs;
8      } tr[maxn * 20 * 20];
9      int sz = 1;
10     TREE() { tr[0] = {0, 0, 0}; }
11     int N(int w, int ls, int rs) {
12         tr[sz] = {w, ls, rs};
13         return sz++;
14     }
15     int add(int tt, int l, int r, int x, int d) {
16         if (x < l || r < x) return tt;
17         const P& t = tr[tt];
18         if (l == r) return N(t.w + d, 0, 0);
19         return N(t.w + d, add(t.ls, lson, x, d), add(t.rs, rson, x, d));
20     }
21     int ls_sum(const VI& rt) {
22         int ret = 0;
23         FOR (i, 0, rt.size())
24             ret += tr[rt[i]].ls.w;
25         return ret;
26     }
27     inline void ls(VI& rt) { transform(rt.begin(), rt.end(), rt.begin(), [&](int x)->int{ return tr[x].ls; }); }
28     inline void rs(VI& rt) { transform(rt.begin(), rt.end(), rt.begin(), [&](int x)->int{ return tr[x].rs; }); }
29     int query(VI& p, VI& q, int l, int r, int k) {
30         if (l == r) return l;
31         int w = ls_sum(q) - ls_sum(p);
32         if (k <= w) {
33             ls(p); ls(q);
34             return query(p, q, lson, k);
35         }
36         else {
37             rs(p); rs(q);
38             return query(p, q, rson, k - w);
39         }
40     }
41 } tree;
42 struct BIT {
43     int root[maxn];
44     void init() { memset(root, 0, sizeof root); }
45     inline int lowbit(int x) { return x & -x; }
46     void update(int p, int x, int d) {
47         for (int i = p; i <= m; i += lowbit(i))
48             root[i] = tree.add(root[i], 1, m, x, d);
49     }
50     int query(int l, int r, int k) {
51         VI p, q;
52         for (int i = l - 1; i > 0; i -= lowbit(i)) p.push_back(root[i]);
53         for (int i = r; i > 0; i -= lowbit(i)) q.push_back(root[i]);
54         return tree.query(p, q, 1, m, k);
55     }
56 } bit;
57
58 void init() {
59     m = 10000;
60     tree.sz = 1;
61     bit.init();
62     FOR (i, 1, m + 1)

```

```

63         bit.update(i, a[i], 1);
64     }

```

左偏树

```

1  namespace LTree {
2      extern struct P* null, *pit;
3      queue<P*> trash;
4      const int M = 1E5 + 100;
5      struct P {
6          P *ls, *rs;
7          LL v;
8          int d;
9          void operator delete (void* ptr) {
10             trash.push((P*)ptr);
11         }
12         void* operator new(size_t size) {
13             if (trash.empty()) return pit++;
14             void* ret = trash.front(); trash.pop(); return ret;
15         }
16
17         void prt() {
18             if (this == null) return;
19             cout << v << ' ';
20             ls->prt(); rs->prt();
21         }
22     } pool[M], *pit = pool, *null = new P{0, 0, -1, -1};
23     P* N(LL v) {
24         return new P{null, null, v, 0};
25     }
26     P* merge(P* a, P* b) {
27         if (a == null) return b;
28         if (b == null) return a;
29         if (a->v > b->v) swap(a, b);
30         a->rs = merge(a->rs, b);
31         if (a->ls->d < a->rs->d) swap(a->ls, a->rs);
32         a->d = a->rs->d + 1;
33         return a;
34     }
35
36     LL pop(P*& o) {
37         LL ret = o->v;
38         P* t = o;
39         o = merge(o->ls, o->rs);
40         delete t;
41         return ret;
42     }
43 }

```

可持久化

```

1  namespace LTree {
2      extern struct P* null, *pit;
3      queue<P*> trash;
4      const int M = 1E6 + 100;
5      struct P {
6          P *ls, *rs;
7          LL v;
8          int d;
9          void operator delete (void* ptr) {
10             trash.push((P*)ptr);
11         }
12         void* operator new(size_t size) {
13             if (trash.empty()) return pit++;
14             void* ret = trash.front(); trash.pop(); return ret;
15         }
16     } pool[M], *pit = pool, *null = new P{0, 0, -1, -1};
17     P* N(LL v, P* ls = null, P* rs = null) {
18         if (ls->d < rs->d) swap(ls, rs);
19         return new P{ls, rs, v, rs->d + 1};

```

```

20     }
21     P* merge(P* a, P* b) {
22         if (a == null) return b;
23         if (b == null) return a;
24         if (a->v < b->v)
25             return N(a->v, a->ls, merge(a->rs, b));
26         else
27             return N(b->v, b->ls, merge(b->rs, a));
28     }
29
30     LL pop(P*& o) {
31         LL ret = o->v;
32         o = merge(o->ls, o->rs);
33         return ret;
34     }
35 }

```

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

```

1 namespace treap {
2     const int M = maxn * 17;
3     extern struct P* const null;
4     struct P {
5         P *ls, *rs;
6         int v, sz;
7         unsigned rd;
8         P(int v): ls(null), rs(null), v(v), sz(1), rd(rnd()) {}
9         P(): sz(0) {}
10
11         P* up() { sz = ls->sz + rs->sz + 1; return this; }
12         int lower(int v) {
13             if (this == null) return 0;
14             return this->v >= v ? ls->lower(v) : rs->lower(v) + ls->sz + 1;
15         }
16         int upper(int v) {
17             if (this == null) return 0;
18             return this->v > v ? ls->upper(v) : rs->upper(v) + ls->sz + 1;
19         }
20     } *const null = new P, pool[M], *pit = pool;
21
22     P* merge(P* l, P* r) {
23         if (l == null) return r; if (r == null) return l;
24         if (l->rd < r->rd) { l->rs = merge(l->rs, r); return l->up(); }
25         else { r->ls = merge(l, r->ls); return r->up(); }
26     }
27
28     void split(P* o, int rk, P*& l, P*& r) {
29         if (o == null) { l = r = null; return; }
30         if (o->ls->sz >= rk) { split(o->ls, rk, l, o->ls); r = o->up(); }
31         else { split(o->rs, rk - o->ls->sz - 1, o->rs, r); l = o->up(); }
32     }
33 }

```

- 持久化 Treap

```

1 namespace treap {
2     const int M = maxn * 17 * 12;
3     extern struct P* const null, *pit;
4     struct P {
5         P *ls, *rs;

```



```

6      int v, sz;
7      LL sum;
8      P(P* ls, P* rs, int v): ls(ls), rs(rs), v(v), sz(ls->sz + rs->sz + 1),
9                                     sum(ls->sum + rs->sum + v) {}
10
11      P() {}
12
13      void* operator new(size_t _) { return pit++; }
14      template<typename T>
15      int rk(int v, T&& cmp) {
16          if (this == null) return 0;
17          return cmp(this->v, v) ? ls->rk(v, cmp) : rs->rk(v, cmp) + ls->sz + 1;
18      }
19      int lower(int v) { return rk(v, greater_equal<int>()); }
20      int upper(int v) { return rk(v, greater<int>()); }
21 } pool[M], *pit = pool, *const null = new P;
22 P* merge(P* l, P* r) {
23     if (l == null) return r; if (r == null) return l;
24     if (rnd() % (l->sz + r->sz) < l->sz) return new P{l->ls, merge(l->rs, r), l->v};
25     else return new P{merge(l, r->ls), r->rs, r->v};
26 }
27 void split(P* o, int rk, P*& l, P*& r) {
28     if (o == null) { l = r = null; return; }
29     if (o->ls->sz >= rk) { split(o->ls, rk, l, r); r = new P{r, o->rs, o->v}; }
30     else { split(o->rs, rk - o->ls->sz - 1, l, r); l = new P{o->ls, l, o->v}; }
31 }

```

- 带 pushdown 的持久化 Treap
- 注意任何修改操作前一定要 FIX

```

1  int now;
2  namespace Treap {
3      const int M = 10000000;
4      extern struct P* const null, *pit;
5      struct P {
6          P *ls, *rs;
7          int sz, time;
8          LL cnt, sc, pos, add;
9          bool rev;
10
11          P* up() { sz = ls->sz + rs->sz + 1; sc = ls->sc + rs->sc + cnt; return this; } // MOD
12          P* check() {
13              if (time == now) return this;
14              P* t = new(pit++) P; *t = *this; t->time = now; return t;
15          };
16          P* _do_rev() { rev ^= 1; add *= -1; pos *= -1; swap(ls, rs); return this; } // MOD
17          P* _do_add(LL v) { add += v; pos += v; return this; } // MOD
18          P* do_rev() { if (this == null) return this; return check()->_do_rev(); } // FIX & MOD
19          P* do_add(LL v) { if (this == null) return this; return check()->_do_add(v); } // FIX & MOD
20          P* _down() { // MOD
21              if (rev) { ls = ls->do_rev(); rs = rs->do_rev(); rev = 0; }
22              if (add) { ls = ls->do_add(add); rs = rs->do_add(add); add = 0; }
23              return this;
24          }
25          P* down() { return check()->_down(); } // FIX & MOD
26          void _split(LL p, P*& l, P*& r) { // MOD
27              if (pos >= p) { ls->split(p, l, r); ls = r; r = up(); }
28              else { rs->split(p, l, r); rs = l; l = up(); }
29          }
30          void split(LL p, P*& l, P*& r) { // FIX & MOD
31              if (this == null) l = r = null;
32              else down()->_split(p, l, r);
33          }
34      } pool[M], *pit = pool, *const null = new P;
35      P* merge(P* a, P* b) {
36          if (a == null) return b; if (b == null) return a;
37          if (rand() % (a->sz + b->sz) < a->sz) { a = a->down(); a->rs = merge(a->rs, b); return a->up(); }
38          else { b = b->down(); b->ls = merge(a, b->ls); return b->up(); }
39      }
40  }

```

Treap-序列

- 区间 ADD, SUM

```
1 namespace treap {
2     const int M = 8E5 + 100;
3     extern struct P* const null;
4     struct P {
5         P *ls, *rs;
6         int sz, val, add, sum;
7         P(int v, P* ls = null, P* rs = null): ls(ls), rs(rs), sz(1), val(v), add(0), sum(v) {}
8         P(): sz(0), val(0), add(0), sum(0) {}
9
10        P* up() {
11            assert(this != null);
12            sz = ls->sz + rs->sz + 1;
13            sum = ls->sum + rs->sum + val + add * sz;
14            return this;
15        }
16        void upd(int v) {
17            if (this == null) return;
18            add += v;
19            sum += sz * v;
20        }
21        P* down() {
22            if (add) {
23                ls->upd(add); rs->upd(add);
24                val += add;
25                add = 0;
26            }
27            return this;
28        }
29
30        P* select(int rk) {
31            if (rk == ls->sz + 1) return this;
32            return ls->sz >= rk ? ls->select(rk) : rs->select(rk - ls->sz - 1);
33        }
34    } pool[M], *pit = pool, *const null = new P, *rt = null;
35
36    P* merge(P* a, P* b) {
37        if (a == null) return b->up();
38        if (b == null) return a->up();
39        if (rand() % (a->sz + b->sz) < a->sz) {
40            a->down()->rs = merge(a->rs, b);
41            return a->up();
42        } else {
43            b->down()->ls = merge(a, b->ls);
44            return b->up();
45        }
46    }
47
48    void split(P* o, int rk, P*& l, P*& r) {
49        if (o == null) { l = r = null; return; }
50        o->down();
51        if (o->ls->sz >= rk) {
52            split(o->ls, rk, l, o->ls);
53            r = o->up();
54        } else {
55            split(o->rs, rk - o->ls->sz - 1, o->rs, r);
56            l = o->up();
57        }
58    }
59
60    inline void insert(int k, int v) {
61        P *l, *r;
62        split(rt, k - 1, l, r);
63        rt = merge(merge(l, new (pit++) P(v)), r);
64    }
65
66    inline void erase(int k) {
67        P *l, *r, *_ , *t;
```

```

68     split(rt, k - 1, l, t);
69     split(t, 1, _, r);
70     rt = merge(l, r);
71 }
72
73 P* build(int l, int r, int* a) {
74     if (l > r) return null;
75     if (l == r) return new(pit++) P(a[l]);
76     int m = (l + r) / 2;
77     return (new(pit++) P(a[m], build(l, m - 1, a), build(m + 1, r, a)))->up();
78 }
79 };

```

● 区间 REVERSE, ADD, MIN

```

1  namespace treap {
2      extern struct P*const null;
3      struct P {
4          P *ls, *rs;
5          int sz, v, add, m;
6          bool flip;
7          P(int v, P* ls = null, P* rs = null): ls(ls), rs(rs), sz(1), v(v), add(0), m(v), flip(0) {}
8          P(): sz(0), v(INF), m(INF) {}
9
10         void upd(int v) {
11             if (this == null) return;
12             add += v; m += v;
13         }
14         void rev() {
15             if (this == null) return;
16             swap(ls, rs);
17             flip ^= 1;
18         }
19         P* up() {
20             assert(this != null);
21             sz = ls->sz + rs->sz + 1;
22             m = min(min(ls->m, rs->m), v) + add;
23             return this;
24         }
25         P* down() {
26             if (add) {
27                 ls->upd(add); rs->upd(add);
28                 v += add;
29                 add = 0;
30             }
31             if (flip) {
32                 ls->rev(); rs->rev();
33                 flip = 0;
34             }
35             return this;
36         }
37
38         P* select(int k) {
39             if (ls->sz + 1 == k) return this;
40             if (ls->sz >= k) return ls->select(k);
41             return rs->select(k - ls->sz - 1);
42         }
43
44     } pool[M], *const null = new P, *pit = pool, *rt = null;
45
46     P* merge(P* a, P* b) {
47         if (a == null) return b;
48         if (b == null) return a;
49         if (rnd() % (a->sz + b->sz) < a->sz) {
50             a->down()->rs = merge(a->rs, b);
51             return a->up();
52         } else {
53             b->down()->ls = merge(a, b->ls);
54             return b->up();
55         }
56     }
57 }

```

```

58 void split(P* o, int k, P*& l, P*& r) {
59     if (o == null) { l = r = null; return; }
60     o->down();
61     if (o->ls->sz >= k) {
62         split(o->ls, k, l, o->ls);
63         r = o->up();
64     } else {
65         split(o->rs, k - o->ls->sz - 1, o->rs, r);
66         l = o->up();
67     }
68 }
69
70 P* build(int l, int r, int* v) {
71     if (l > r) return null;
72     int m = (l + r) >> 1;
73     return (new (pit++) P(v[m], build(l, m - 1, v), build(m + 1, r, v)))->up();
74 }
75
76 void go(int x, int y, void f(P*&)) {
77     P *l, *m, *r;
78     split(rt, y, l, r);
79     split(l, x - 1, l, m);
80     f(m);
81     rt = merge(merge(l, m), r);
82 }
83
84 using namespace treap;
85 int a[maxn], n, x, y, Q, v, k, d;
86 char s[100];
87
88 int main() {
89     cin >> n;
90     FOR (i, 1, n + 1) scanf("%d", &a[i]);
91     rt = build(1, n, a);
92     cin >> Q;
93     while (Q--) {
94         scanf("%s", s);
95         if (s[0] == 'A') {
96             scanf("%d%d%d", &x, &y, &v);
97             go(x, y, [](P*& o){ o->upd(v); });
98         } else if (s[0] == 'R' && s[3] == 'E') {
99             scanf("%d%d", &x, &y);
100            go(x, y, [](P*& o){ o->rev(); });
101        } else if (s[0] == 'R' && s[3] == 'O') {
102            scanf("%d%d%d", &x, &y, &d);
103            d %= y - x + 1;
104            go(x, y, [](P*& o){
105                P *l, *r;
106                split(o, o->sz - d, l, r);
107                o = merge(r, l);
108            });
109        } else if (s[0] == 'I') {
110            scanf("%d%d", &k, &v);
111            go(k + 1, k, [](P*& o){ o = new (pit++) P(v); });
112        } else if (s[0] == 'D') {
113            scanf("%d", &k);
114            go(k, k, [](P*& o){ o = null; });
115        } else if (s[0] == 'M') {
116            scanf("%d%d", &x, &y);
117            go(x, y, [](P*& o) {
118                printf("%d\n", o->m);
119            });
120        }
121    }
122 }

```

● 持久化

```

1 namespace treap {
2     struct P;
3     extern P*const null;
4     P* N(P* ls, P* rs, LL v, bool fill);

```

```

5 struct P {
6     P *const ls, *const rs;
7     const int sz, v;
8     const LL sum;
9     bool fill;
10    int cnt;
11
12    void split(int k, P*& l, P*& r) {
13        if (this == null) { l = r = null; return; }
14        if (ls->sz >= k) {
15            ls->split(k, l, r);
16            r = N(r, rs, v, fill);
17        } else {
18            rs->split(k - ls->sz - fill, l, r);
19            l = N(ls, l, v, fill);
20        }
21    }
22
23
24    } *const null = new P{0, 0, 0, 0, 0, 0, 1};
25
26    P* N(P* ls, P* rs, LL v, bool fill) {
27        ls->cnt++; rs->cnt++;
28        return new P{ls, rs, ls->sz + rs->sz + fill, v, ls->sum + rs->sum + v, fill, 1};
29    }
30
31    P* merge(P* a, P* b) {
32        if (a == null) return b;
33        if (b == null) return a;
34        if (rand() % (a->sz + b->sz) < a->sz)
35            return N(a->ls, merge(a->rs, b), a->v, a->fill);
36        else
37            return N(merge(a, b->ls), b->rs, b->v, b->fill);
38    }
39
40    void go(P* o, int x, int y, P*& l, P*& m, P*& r) {
41        o->split(y, l, r);
42        l->split(x - 1, l, m);
43    }
44 }

```

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

```

1 namespace uf {
2     int fa[maxn], sz[maxn];
3     int undo[maxn], top;
4     void init() { memset(fa, -1, sizeof fa); memset(sz, 0, sizeof sz); top = 0; }
5     int findset(int x) { while (fa[x] != -1) x = fa[x]; return x; }
6     bool join(int x, int y) {
7         x = findset(x); y = findset(y);
8         if (x == y) return false;
9         if (sz[x] > sz[y]) swap(x, y);
10        undo[top++] = x;
11        fa[x] = y;
12        sz[y] += sz[x] + 1;
13        return true;
14    }
15    inline int checkpoint() { return top; }
16    void rewind(int t) {
17        while (top > t) {
18            int x = undo[--top];
19            sz[fa[x]] -= sz[x] + 1;
20            fa[x] = -1;
21        }
22    }
23 }

```

```

22     }
23 }

```

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

```

1  struct P {
2      P *L, *R, *U, *D;
3      int x, y;
4  };
5
6  const int INF = 1E9;
7
8  struct DLX {
9      #define TR(i, D, s) for (P* i = s->D; i != s; i = i->D)
10     static const int M = 2E5;
11     P pool[M], *h[M], *r[M], *pit;
12     int sz[M];
13     bool solved;
14     stack<int> ans;
15     void init(int n) {
16         pit = pool;
17         ++n;
18         solved = false;
19         while (!ans.empty()) ans.pop();
20         memset(r, 0, sizeof r);
21         memset(sz, 0, sizeof sz);
22         FOR (i, 0, n)
23             h[i] = new (pit++) P;
24         FOR (i, 0, n) {
25             h[i]->L = h[(i + n - 1) % n];
26             h[i]->R = h[(i + 1) % n];
27             h[i]->U = h[i]->D = h[i];
28             h[i]->y = i;
29         }
30     }
31
32     void link(int x, int y) {
33         sz[y]++;
34         auto p = new (pit++) P;
35         p->x = x; p->y = y;
36         p->U = h[y]->U; p->D = h[y];
37         p->D->U = p->U->D = p;
38         if (!r[x]) r[x] = p->L = p->R = p;
39         else {
40             p->L = r[x]; p->R = r[x]->R;
41             p->L->R = p->R->L = p;
42         }
43     }
44
45     void remove(P* p) {
46         p->L->R = p->R; p->R->L = p->L;
47         TR (i, D, p)
48             TR (j, R, i) {
49                 j->D->U = j->U; j->U->D = j->D;
50                 sz[j->y]--;
51             }
52     }
53
54     void recall(P* p) {
55         p->L->R = p->R->L = p;
56         TR (i, U, p)
57             TR (j, L, i) {
58                 j->D->U = j->U->D = j;
59                 sz[j->y]++;

```

```

60         }
61     }
62
63     bool dfs(int d) {
64         if (solved) return true;
65         if (h[0]->R == h[0]) return solved = true;
66         int m = INF;
67         P* c;
68         TR (i, R, h[0])
69             if (sz[i->y] < m) { m = sz[i->y]; c = i; }
70         remove(c);
71         TR (i, D, c) {
72             ans.push(i->x);
73             TR (j, R, i) remove(h[j->y]);
74             if (dfs(d + 1)) return true;
75             TR (j, L, i) recall(h[j->y]);
76             ans.pop();
77         }
78         recall(c);
79         return false;
80     }
81 } dlx;

```

- 可重复覆盖

```

1  struct P {
2      P *L, *R, *U, *D;
3      int x, y;
4  };
5
6  const int INF = 1E9;
7
8  struct DLX {
9      #define TR(i, D, s) for (P* i = s->D; i != s; i = i->D)
10     static const int M = 2E5;
11     P pool[M], *h[M], *r[M], *pit;
12     int sz[M], vis[M], ans, clk;
13     void init(int n) {
14         clk = 0;
15         ans = INF;
16         pit = pool;
17         ++n;
18         memset(r, 0, sizeof r);
19         memset(sz, 0, sizeof sz);
20         memset(vis, -1, sizeof vis);
21         FOR (i, 0, n)
22             h[i] = new (pit++) P;
23         FOR (i, 0, n) {
24             h[i]->L = h[(i + n - 1) % n];
25             h[i]->R = h[(i + 1) % n];
26             h[i]->U = h[i]->D = h[i];
27             h[i]->y = i;
28         }
29     }
30
31     void link(int x, int y) {
32         sz[y]++;
33         auto p = new (pit++) P;
34         p->x = x; p->y = y;
35         p->U = h[y]->U; p->D = h[y];
36         p->D->U = p->U->D = p;
37         if (!r[x]) r[x] = p->L = p->R = p;
38         else {
39             p->L = r[x]; p->R = r[x]->R;
40             p->L->R = p->R->L = p;
41         }
42     }
43
44     void remove(P* p) {
45         TR (i, D, p) {
46             i->L->R = i->R;
47             i->R->L = i->L;

```

```

48     }
49 }
50
51 void recall(P* p) {
52     TR (i, U, p)
53     i->L->R = i->R->L = i;
54 }
55
56 int eval() {
57     ++clk;
58     int ret = 0;
59     TR (i, R, h[0])
60         if (vis[i->y] != clk) {
61             ++ret;
62             vis[i->y] = clk;
63             TR (j, D, i)
64                 TR (k, R, j)
65                     vis[k->y] = clk;
66         }
67     return ret;
68 }
69
70 void dfs(int d) {
71     if (h[0]->R == h[0]) { ans = min(ans, d); return; }
72     if (eval() + d >= ans) return;
73     P* c;
74     int m = INF;
75     TR (i, R, h[0])
76         if (sz[i->y] < m) { m = sz[i->y]; c = i; }
77     TR (i, D, c) {
78         remove(i);
79         TR (j, R, i) remove(j);
80         dfs(d + 1);
81         TR (j, L, i) recall(j);
82         recall(i);
83     }
84 }
85 } dlx;

```

CDQ 分治

```

1  const int maxn = 2E5 + 100;
2  struct P {
3      int x, y;
4      int* f;
5      bool d1, d2;
6  } a[maxn], b[maxn], c[maxn];
7  int f[maxn];
8
9  void go2(int l, int r) {
10     if (l + 1 == r) return;
11     int m = (l + r) >> 1;
12     go2(l, m); go2(m, r);
13     FOR (i, l, m) b[i].d2 = 0;
14     FOR (i, m, r) b[i].d2 = 1;
15     merge(b + l, b + m, b + m, b + r, c + l, [](const P& a, const P& b)->bool {
16         if (a.y != b.y) return a.y < b.y;
17         return a.d2 > b.d2;
18     });
19     int mx = -1;
20     FOR (i, l, r) {
21         if (c[i].d1 && c[i].d2) *c[i].f = max(*c[i].f, mx + 1);
22         if (!c[i].d1 && !c[i].d2) mx = max(mx, *c[i].f);
23     }
24     FOR (i, l, r) b[i] = c[i];
25 }
26
27 void go1(int l, int r) { // [l, r)
28     if (l + 1 == r) return;

```



```

29     int m = (l + r) >> 1;
30     go1(l, m);
31     FOR (i, l, m) a[i].d1 = 0;
32     FOR (i, m, r) a[i].d1 = 1;
33     copy(a + l, a + r, b + l);
34     sort(b + l, b + r, [] (const P& a, const P& b) -> bool {
35         if (a.x != b.x) return a.x < b.x;
36         return a.d1 > b.d1;
37     });
38     go2(l, r);
39     go1(m, r);
40 }

```

● k 维 LIS

```

1  struct P {
2      int v[K];
3      LL f;
4      bool d[K];
5  } o[N << 10];
6  P* a[K][N << 10];
7  int k;
8  void go(int now, int l, int r) {
9      if (now == 0) {
10         if (l + 1 == r) return;
11         int m = (l + r) / 2;
12         go(now, l, m);
13         FOR (i, l, m) a[now][i] -> d[now] = 0;
14         FOR (i, m, r) a[now][i] -> d[now] = 1;
15         copy(a[now] + l, a[now] + r, a[now + 1] + l);
16         sort(a[now + 1] + l, a[now + 1] + r, [now] (const P* a, const P* b) {
17             if (a -> v[now] != b -> v[now]) return a -> v[now] < b -> v[now];
18             return a -> d[now] > b -> d[now];
19         });
20         go(now + 1, l, r);
21         go(now, m, r);
22     } else {
23         if (l + 1 == r) return;
24         int m = (l + r) / 2;
25         go(now, l, m); go(now, m, r);
26         FOR (i, l, m) a[now][i] -> d[now] = 0;
27         FOR (i, m, r) a[now][i] -> d[now] = 1;
28         merge(a[now] + l, a[now] + m, a[now] + m, a[now] + r, a[now + 1] + l, [now] (const P* a, const P* b) {
29             if (a -> v[now] != b -> v[now]) return a -> v[now] < b -> v[now];
30             return a -> d[now] > b -> d[now];
31         });
32         copy(a[now + 1] + l, a[now + 1] + r, a[now] + l);
33         if (now < k - 2) {
34             go(now + 1, l, r);
35         } else {
36             LL sum = 0;
37             FOR (i, l, r) {
38                 dbg(a[now][i] -> v[0], a[now][i] -> v[1], a[now][i] -> f,
39                     a[now][i] -> d[0], a[now][i] -> d[1]);
40
41                 int cnt = 0;
42                 FOR (j, 0, now + 1) cnt += a[now][i] -> d[j];
43                 if (cnt == 0) {
44                     sum += a[now][i] -> f;
45                 } else if (cnt == now + 1) {
46                     a[now][i] -> f = (a[now][i] -> f + sum) % MOD;
47                 }
48             }
49         }
50     }
}

```

哈希表

● 必须初始化

- 备选素数 1572869, 3145739, 6291469, 12582917, 25165843, 50331653

```

1  const LL HASH_MOD=1572869;
2  LL key[HASH_MOD], val[HASH_MOD];
3  int head[HASH_MOD], next[HASH_MOD];
4  struct Hash {
5      int sz;
6      void init() {
7          memset(head, -1, sizeof head);
8          sz = 0;
9      }
10     LL insert(LL x, LL y) {
11         int k = x % HASH_MOD;
12         key[sz] = x;
13         val[sz] = y;
14         next[sz] = head[k];
15         head[k] = sz++;
16     }
17     LL find(LL x) {
18         int k = x % HASH_MOD;
19         for (int i = head[k]; i != -1; i = next[i])
20             if (key[i] == x)
21                 return val[i];
22         return -1;
23     }
24 };

```

笛卡尔树

```

1  void build(const vector<int>& a) {
2      static P *stack[M], *x, *last;
3      int p = 0;
4      FOR (i, 0, a.size()) {
5          x = new P(i + 1, a[i]);
6          last = null;
7          while (p && stack[p - 1]->v > x->v) {
8              stack[p - 1]->maintain();
9              last = stack[--p];
10         }
11         if (p) stack[p - 1]->rs = x;
12         x->ls = last;
13         stack[p++] = x;
14     }
15     while (p)
16         stack[--p]->maintain();
17     rt = stack[0];
18 }

1  void build() {
2      static int s[N], last;
3      int p = 0;
4      FOR (x, 1, n + 1) {
5          last = 0;
6          while (p && val[s[p - 1]] > val[x]) last = s[--p];
7          if (p) G[s[p - 1]][1] = x;
8          if (last) G[x][0] = last;
9          s[p++] = x;
10     }
11     rt = s[0];
12 }

```

Trie

- Trie 二进制版
- M 为二进制的位数
- 使用前必须初始化

```

1 struct Trie2 {
2     int ch[N * M][2], sz;
3     void init() {
4         memset(ch, 0, sizeof ch);
5         sz = 1;
6     }
7     void insert(LL x) {
8         int u = 0;
9         FOR (i, M, -1) {
10             bool b = x & (1LL << i);
11             if (!ch[u][b])
12                 ch[u][b] = sz++;
13             u = ch[u][b];
14         }
15     }
16 } 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

```

1 #include<ext/pb_ds/priority_queue.hpp>
2 template<typename _Tv,
3         typename Cmp_Fn = std::less<_Tv>,
4         typename Tag = pairing_heap_tag,
5         typename _Alloc = std::allocator<char> >
6 class priority_queue;

1 #include<ext/pb_ds/priority_queue.hpp>
2 using namespace __gnu_pbds;
3
4 typedef __gnu_pbds::priority_queue<LL, less<LL>, pairing_heap_tag> PQ;
5 __gnu_pbds::priority_queue<int, cmp, pairing_heap_tag>::point_iterator it;
6 PQ pq, pq2;
7
8 int main() {
9     auto it = pq.push(2);
10    pq.push(3);
11    assert(pq.top() == 3);
12    pq.modify(it, 4);
13    assert(pq.top() == 4);
14    pq2.push(5);
15    pq.join(pq2);
16    assert(pq.top() == 5);
17 }

```

- 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), 或者有 order 个元素比它小的 key
- order_of_key 有多少个比 r_key 小的元素
- join & split

```

1  template<typename Key, typename Mapped, typename Cmp_Fn = std::less<Key>,
2      typename Tag = rb_tree_tag,
3      template<typename Node_Citr, typename Node_Itr,
4          typename Cmp_Fn_, typename _Alloc_>
5          class Node_Update = null_node_update,
6      typename _Alloc = std::allocator<char> >
7  class tree
8
9  #include <ext/pb_ds/assoc_container.hpp>
10 using namespace __gnu_pbds;
11 using Tree = tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update>;
12 Tree t;

```

- hash table

```

1  #include<ext/pb_ds/assoc_container.hpp>
2  #include<ext/pb_ds/hash_policy.hpp>
3  using namespace __gnu_pbds;
4
5  gp_hash_table<int, int> mp;
6  cc_hash_table<int, int> mp;

```

Link-Cut Tree

- 图中相邻的结点在伸展树中不一定是父子关系
- 遇事不决 make_root
- 跑左右儿子的时候不要忘记 down

```

1  namespace lct {
2      extern struct P *const null;
3      const int M = N;
4      struct P {
5          P *fa, *ls, *rs;
6          int v, maxv;
7          bool rev;
8
9          bool has_fa() { return fa->ls == this || fa->rs == this; }
10         bool d() { return fa->ls == this; }
11         P*& c(bool x) { return x ? ls : rs; }
12         void do_rev() {
13             if (this == null) return;
14             rev ^= 1;
15             swap(ls, rs);
16         }
17         P* up() {
18             maxv = max(v, max(ls->maxv, rs->maxv));
19             return this;
20         }
21         void down() {
22             if (rev) {
23                 rev = 0;
24                 ls->do_rev(); rs->do_rev();
25             }
26         }
27         void all_down() { if (has_fa()) fa->all_down(); down(); }
28     } *const null = new P{0, 0, 0, 0, 0, 0}, pool[M], *pit = pool;
29
30     void rot(P* o) {
31         bool dd = o->d();
32         P *f = o->fa, *t = o->c(!dd);
33         if (f->has_fa()) f->fa->c(f->d()) = o; o->fa = f->fa;
34         if (t != null) t->fa = f; f->c(dd) = t;
35         o->c(!dd) = f->up(); f->fa = o;
36     }
37     void splay(P* o) {
38         o->all_down();
39         while (o->has_fa()) {
40             if (o->fa->has_fa())
41                 rot(o->d() ^ o->fa->d() ? o : o->fa);

```

```

42         rot(o);
43     }
44     o->up();
45 }
46 void access(P* u, P* v = null) {
47     if (u == null) return;
48     splay(u); u->rs = v;
49     access(u->up()->fa, u);
50 }
51 void make_root(P* o) {
52     access(o); splay(o); o->do_rev();
53 }
54 void split(P* o, P* u) {
55     make_root(o); access(u); splay(u);
56 }
57 void link(P* u, P* v) {
58     make_root(u); u->fa = v;
59 }
60 void cut(P* u, P* v) {
61     split(u, v);
62     u->fa = v->ls = null; v->up();
63 }
64 bool adj(P* u, P* v) {
65     split(u, v);
66     return v->ls == u && u->ls == null && u->rs == null;
67 }
68 bool linked(P* u, P* v) {
69     split(u, v);
70     return u == v || u->fa != null;
71 }
72 P* findrt(P* o) {
73     access(o); splay(o);
74     while (o->ls != null) o = o->ls;
75     return o;
76 }
77 P* findfa(P* rt, P* u) {
78     split(rt, u);
79     u = u->ls;
80     while (u->rs != null) {
81         u = u->rs;
82         u->down();
83     }
84     return u;
85 }
86 }

```

● 维护子树大小

```

1 P* up() {
2     sz = ls->sz + rs->sz + _sz + 1;
3     return this;
4 }
5 void access(P* u, P* v = null) {
6     if (u == null) return;
7     splay(u);
8     u->_sz += u->rs->sz - v->sz;
9     u->rs = v;
10    access(u->up()->fa, u);
11 }
12 void link(P* u, P* v) {
13     split(u, v);
14     u->fa = v; v->_sz += u->sz;
15     v->up();
16 }

```

莫队

● $[l, r)$

```

1 while (l > q.l) mv(--l, 1);

```

```

2 while (r < q.r) mv(r++, 1);
3 while (l < q.l) mv(l++, -1);
4 while (r > q.r) mv(--r, -1);

    • 树上莫队
    • 注意初始状态 u = v = 1, flip(1)

1 struct Q {
2     int u, v, idx;
3     bool operator < (const Q& b) const {
4         const Q& a = *this;
5         return blk[a.u] < blk[b.u] || (blk[a.u] == blk[b.u] && in[a.v] < in[b.v]);
6     }
7 };
8
9 void dfs(int u = 1, int d = 0) {
10     static int S[maxn], sz = 0, blk_cnt = 0, clk = 0;
11     in[u] = clk++;
12     dep[u] = d;
13     int btm = sz;
14     for (int v: G[u]) {
15         if (v == fa[u]) continue;
16         fa[v] = u;
17         dfs(v, d + 1);
18         if (sz - btm >= B) {
19             while (sz > btm) blk[S[--sz]] = blk_cnt;
20             ++blk_cnt;
21         }
22     }
23     S[sz++] = u;
24     if (u == 1) while (sz) blk[S[--sz]] = blk_cnt - 1;
25 }
26
27 void flip(int k) {
28     dbg(k);
29     if (vis[k]) {
30         // ...
31     } else {
32         // ...
33     }
34     vis[k] ^= 1;
35 }
36
37 void go(int& k) {
38     if (bug == -1) {
39         if (vis[k] && !vis[fa[k]]) bug = k;
40         if (!vis[k] && vis[fa[k]]) bug = fa[k];
41     }
42     flip(k);
43     k = fa[k];
44 }
45
46 void mv(int a, int b) {
47     bug = -1;
48     if (vis[b]) bug = b;
49     if (dep[a] < dep[b]) swap(a, b);
50     while (dep[a] > dep[b]) go(a);
51     while (a != b) {
52         go(a); go(b);
53     }
54     go(a); go(bug);
55 }
56
57 for (Q& q: query) {
58     mv(u, q.u); u = q.u;
59     mv(v, q.v); v = q.v;
60     ans[q.idx] = Ans;
61 }

```

数学

矩阵运算

```
1 struct Mat {
2     static const LL M = 2;
3     LL v[M][M];
4     Mat() { memset(v, 0, sizeof v); }
5     void eye() { FOR (i, 0, M) v[i][i] = 1; }
6     LL* operator [] (LL x) { return v[x]; }
7     const LL* operator [] (LL x) const { return v[x]; }
8     Mat operator * (const Mat& B) {
9         const Mat& A = *this;
10        Mat ret;
11        FOR (i, 0, M)
12            FOR (j, 0, M)
13                FOR (k, 0, M)
14                    ret[i][j] = (ret[i][j] + A[i][k] * B[k][j]) % MOD;
15        return ret;
16    }
17    Mat pow(LL n) const {
18        Mat A = *this, ret; ret.eye();
19        for (; n; n >>= 1, A = A * A)
20            if (n & 1) ret = ret * A;
21        return ret;
22    }
23    Mat operator + (const Mat& B) {
24        const Mat& A = *this;
25        Mat ret;
26        FOR (i, 0, M)
27            FOR (j, 0, M)
28                ret[i][j] = (A[i][j] + B[i][j]) % MOD;
29        return ret;
30    }
31    void prt() const {
32        FOR (i, 0, M)
33            FOR (j, 0, M)
34                printf("%lld%c", (*this)[i][j], j == M - 1 ? '\n' : ' ');
35    }
36 };
```

筛

● 线性筛

```
1 const LL p_max = 1E6 + 100;
2 LL pr[p_max], p_sz;
3 void get_prime() {
4     static bool vis[p_max];
5     FOR (i, 2, p_max) {
6         if (!vis[i]) pr[p_sz++] = i;
7         FOR (j, 0, p_sz) {
8             if (pr[j] * i >= p_max) break;
9             vis[pr[j] * i] = 1;
10            if (i % pr[j] == 0) break;
11        }
12    }
13 }
```

● 线性筛 + 欧拉函数

```
1 const LL p_max = 1E5 + 100;
2 LL phi[p_max] = {-1, 1};
3 void get_phi() {
4     static bool vis[p_max];
5     static LL prime[p_max], p_sz, d;
6     FOR (i, 2, p_max) {
7         if (!vis[i]) {
```

```

8         prime[p_sz++] = i;
9         phi[i] = i - 1;
10    }
11    for (LL j = 0; j < p_sz && (d = i * prime[j]) < p_max; ++j) {
12        vis[d] = 1;
13        if (i % prime[j] == 0) {
14            phi[d] = phi[i] * prime[j];
15            break;
16        }
17        else phi[d] = phi[i] * (prime[j] - 1);
18    }
19 }
20 }

```

● 线性筛 + 莫比乌斯函数

```

1  const LL p_max = 1E5 + 100;
2  LL mu[p_max] = {-1, 1};
3  void get_mu() {
4      static bool vis[p_max];
5      static LL prime[p_max], p_sz, d;
6      mu[1] = 1;
7      FOR (i, 2, p_max) {
8          if (!vis[i]) {
9              prime[p_sz++] = i;
10             mu[i] = -1;
11         }
12         for (LL j = 0; j < p_sz && (d = i * prime[j]) < p_max; ++j) {
13             vis[d] = 1;
14             if (i % prime[j] == 0) {
15                 mu[d] = 0;
16                 break;
17             }
18             else mu[d] = -mu[i];
19         }
20     }
21 }

```

亚线性筛

min_25

```

1  namespace min25 {
2      const int M = 1E6 + 100;
3      LL B, N;
4
5      // g(x)
6      inline LL pg(LL x) { return 1; }
7      inline LL ph(LL x) { return x % MOD; }
8      // Sum[g(i), {x, 2, x}]
9      inline LL psg(LL x) { return x % MOD - 1; }
10     inline LL psh(LL x) {
11         static LL inv2 = (MOD + 1) / 2;
12         x = x % MOD;
13         return x * (x + 1) % MOD * inv2 % MOD - 1;
14     }
15     // f(pp=p^k)
16     inline LL fpk(LL p, LL e, LL pp) { return (pp - pp / p) % MOD; }
17     // f(p) = fgh(g(p), h(p))
18     inline LL fgh(LL g, LL h) { return h - g; }
19
20     LL pr[M], pc, sg[M], sh[M];
21     void get_prime(LL n) {
22         static bool vis[M]; pc = 0;
23         FOR (i, 2, n + 1) {
24             if (!vis[i]) {
25                 pr[pc++] = i;
26                 sg[pc] = (sg[pc - 1] + pg(i)) % MOD;
27                 sh[pc] = (sh[pc - 1] + ph(i)) % MOD;

```



```

28     }
29     FOR (j, 0, pc) {
30         if (pr[j] * i > n) break;
31         vis[pr[j] * i] = 1;
32         if (i % pr[j] == 0) break;
33     }
34 }
35 }
36
37 LL w[M];
38 LL id1[M], id2[M], h[M], g[M];
39 inline LL id(LL x) { return x <= B ? id1[x] : id2[N / x]; }
40
41 LL go(LL x, LL k) {
42     if (x <= 1 || (k >= 0 && pr[k] > x)) return 0;
43     LL t = id(x);
44     LL ans = fgh((g[t] - sg[k + 1]), (h[t] - sh[k + 1]));
45     FOR (i, k + 1, pc) {
46         LL p = pr[i];
47         if (p * p > x) break;
48         ans -= fgh(pg(p), ph(p));
49         for (LL pp = p, e = 1; pp <= x; ++e, pp = pp * p)
50             ans += fpk(p, e, pp) * (1 + go(x / pp, i)) % MOD;
51     }
52     return ans % MOD;
53 }
54
55 LL solve(LL _N) {
56     N = _N;
57     B = sqrt(N + 0.5);
58     get_prime(B);
59     int sz = 0;
60     for (LL l = 1, v, r; l <= N; l = r + 1) {
61         v = N / l; r = N / v;
62         w[sz] = v; g[sz] = psg(v); h[sz] = psh(v);
63         if (v <= B) id1[v] = sz; else id2[r] = sz;
64         sz++;
65     }
66     FOR (k, 0, pc) {
67         LL p = pr[k];
68         FOR (i, 0, sz) {
69             LL v = w[i]; if (p * p > v) break;
70             LL t = id(v / p);
71             g[i] = (g[i] - (g[t] - sg[k]) * pg(p)) % MOD;
72             h[i] = (h[i] - (h[t] - sh[k]) * ph(p)) % MOD;
73         }
74     }
75     return (go(N, -1) % MOD + MOD + 1) % MOD;
76 }
77 }

```

杜教筛

```

1 namespace dujiao {
2     const int M = 5E6;
3     LL f[M] = {0, 1};
4     void init() {
5         static bool vis[M];
6         static LL pr[M], p_sz, d;
7         FOR (i, 2, M) {
8             if (!vis[i]) { pr[p_sz++] = i; f[i] = -1; }
9             FOR (j, 0, p_sz) {
10                 if ((d = pr[j] * i) >= M) break;
11                 vis[d] = 1;
12                 if (i % pr[j] == 0) {
13                     f[d] = 0;
14                     break;
15                 } else f[d] = -f[i];
16             }
17         }
18     }
19 }

```

```

18     FOR (i, 2, M) f[i] += f[i - 1];
19 }
20 inline LL s_fg(LL n) { return 1; }
21 inline LL s_g(LL n) { return n; }
22
23 LL N, rd[M];
24 bool vis[M];
25 LL go(LL n) {
26     if (n < M) return f[n];
27     LL id = N / n;
28     if (vis[id]) return rd[id];
29     vis[id] = true;
30     LL& ret = rd[id] = s_fg(n);
31     for (LL l = 2, v, r; l <= n; l = r + 1) {
32         v = n / l; r = n / v;
33         ret -= (s_g(r) - s_g(l - 1)) * go(v);
34     }
35     return ret;
36 }
37 LL solve(LL n) {
38     N = n;
39     memset(vis, 0, sizeof vis);
40     return go(n);
41 }
42 }

```

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/>

```

1 bool checkQ(LL a, LL n) {
2     if (n == 2 || a >= n) return 1;
3     if (n == 1 || !(n & 1)) return 0;
4     LL d = n - 1;
5     while (!(d & 1)) d >>= 1;
6     LL t = bin(a, d, n); // 不一定需要快速乘
7     while (d != n - 1 && t != 1 && t != n - 1) {
8         t = mul(t, t, n);
9         d <<= 1;
10    }
11    return t == n - 1 || d & 1;
12 }
13
14 bool primeQ(LL n) {
15     static vector<LL> t = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
16     if (n <= 1) return false;
17     for (LL k: t) if (!checkQ(k, n)) return false;
18     return true;
19 }

```

线性递推

```

1 // k 为 m 最高次数 且 a[m] == 1
2 namespace BerlekampMassey {
3     inline void up(LL& a, LL b) { (a += b) %= MOD; }
4
5     V mul(const V& a, const V& b, const V& m, int k) {
6         V r; r.resize(2 * k - 1);
7         FOR (i, 0, k)
8             FOR (j, 0, k)
9                 up(r[i + j], a[i] * b[j]);

```

```

10     FORD (i, k - 2, - 1) {
11         FOR (j, 0, k)
12             up(r[i + j], r[i + k] * m[j]);
13         r.pop_back();
14     }
15     return r;
16 }
17
18 V pow(LL n, const V& m) {
19     int k = (int)m.size() - 1; assert(m[k] == -1 || m[k] == MOD - 1);
20     V r(k), x(k); r[0] = x[1] = 1;
21     for (; n; n >>= 1, x = mul(x, x, m, k))
22         if (n & 1) r = mul(x, r, m, k);
23     return r;
24 }
25
26 LL go(const V& a, const V& x, LL n) {
27     // a: (-1, a1, a2, ..., ak).reverse
28     // x: x1, x2, ..., xk
29     // x[n] = sum[a[i]*x[n-i], {i, 1, k}]
30     int k = (int)a.size() - 1;
31     if (n <= k) return x[n - 1];
32     V r = pow(n - 1, a);
33     LL ans = 0;
34     FOR (i, 0, k)
35         up(ans, r[i] * x[i]);
36     return ans;
37 }
38
39 V BM(const V& x) {
40     V a = {-1}, b = {233};
41     FOR (i, 1, x.size()) {
42         b.push_back(0);
43         LL d = 0, la = a.size(), lb = b.size();
44         FOR (j, 0, la) up(d, a[j] * x[i - la + 1 + j]);
45         if (d == 0) continue;
46         V t; for (auto& v: b) t.push_back(d * v % MOD);
47         FOR (j, 0, a.size()) up(t[lb - 1 - j], a[la - 1 - j]);
48         if (lb > la) {
49             b = a;
50             LL inv = -get_inv(d, MOD);
51             for (auto& v: b) v = v * inv % MOD;
52         }
53         a.swap(t);
54     }
55     for (auto& v: a) up(v, MOD);
56     return a;
57 }
58 }

```

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

```

1 LL ex_gcd(LL a, LL b, LL &x, LL &y) {
2     if (b == 0) {
3         x = 1;
4         y = 0;
5         return a;
6     }
7     LL ret = ex_gcd(b, a % b, y, x);
8     y -= a / b * x;
9     return ret;
10 }

```

类欧几里得

- $m = \lfloor \frac{an+b}{c} \rfloor$.
- $f(a, b, c, n) = \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor$: 当 $a \geq c$ or $b \geq c$ 时, $f(a, b, c, n) = (\frac{a}{c})n(n+1)/2 + (\frac{b}{c})(n+1) + f(a \bmod c, b \bmod c, c, n)$; 否则 $f(a, b, c, n) = nm - f(c, c-b-1, a, m-1)$ 。
- $g(a, b, c, n) = \sum_{i=0}^n i \lfloor \frac{ai+b}{c} \rfloor$: 当 $a \geq c$ or $b \geq c$ 时, $g(a, b, c, n) = (\frac{a}{c})n(n+1)(2n+1)/6 + (\frac{b}{c})n(n+1)/2 + g(a \bmod c, b \bmod c, c, n)$; 否则 $g(a, b, c, n) = \frac{1}{2}(n(n+1)m - f(c, c-b-1, a, m-1) - h(c, c-b-1, a, m-1))$ 。
- $h(a, b, c, n) = \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor^2$: 当 $a \geq c$ or $b \geq c$ 时, $h(a, b, c, n) = (\frac{a}{c})^2 n(n+1)(2n+1)/6 + (\frac{b}{c})^2 (n+1) + (\frac{a}{c})(\frac{b}{c})n(n+1) + h(a \bmod c, b \bmod c, c, n) + 2(\frac{a}{c})g(a \bmod c, b \bmod c, c, n) + 2(\frac{b}{c})f(a \bmod c, b \bmod c, c, n)$; 否则 $h(a, b, c, n) = nm(m+1) - 2g(c, c-b-1, a, m-1) - 2f(c, c-b-1, a, m-1) - f(a, b, c, n)$ 。

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

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

- 预处理

$$1 - n$$

的逆元

```
1 LL inv[N] = {-1, 1};
2 void inv_init(LL n, LL p) {
3     inv[1] = 1;
4     FOR (i, 2, n)
5         inv[i] = (p - p / i) * inv[p % i] % p;
6 }
```

- 预处理阶乘及其逆元

```
1 LL invf[M], fac[M] = {1};
2 void fac_inv_init(LL n, LL p) {
3     FOR (i, 1, n)
4         fac[i] = i * fac[i - 1] % p;
5     invf[n - 1] = bin(fac[n - 1], p - 2, p);
6     FORD (i, n - 2, -1)
7         invf[i] = invf[i + 1] * (i + 1) % p;
8 }
```

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

```
1 inline LL C(LL n, LL m) { // n >= m >= 0
2     return n < m || m < 0 ? 0 : fac[n] * invf[m] % MOD * invf[n - m] % MOD;
3 }
```

- 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。

```
1 LL C(LL n, LL m) { // m >= n >= 0
2     if (m - n < n) n = m - n;
3     if (n < 0) return 0;
4     LL ret = 1;
5     FOR (i, 1, n + 1)
6         ret = ret * (m - n + i) % MOD * bin(i, MOD - 2, MOD) % MOD;
```

```

7     return ret;
8 }

1 LL Lucas(LL n, LL m) { // m >= n >= 0
2     return m ? C(n % MOD, m % MOD) * Lucas(n / MOD, m / MOD) % MOD : 1;
3 }

```

- 组合数预处理

```

1 LL C[M][M];
2 void init_C(int n) {
3     FOR (i, 0, n) {
4         C[i][0] = C[i][i] = 1;
5         FOR (j, 1, i)
6             C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % MOD;
7     }
8 }

```

第二类斯特灵数

```

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

```

FFT & NTT & FWT

- NTT
- 前置：快速幂

```

1 LL wn[N << 2], rev[N << 2];
2 int NTT_init(int n_) {
3     int step = 0; int n = 1;
4     for (; n < n_; n <= 1) ++step;
5     FOR (i, 1, n)
6         rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (step - 1));
7     int g = bin(G, (MOD - 1) / n, MOD);
8     wn[0] = 1;
9     for (int i = 1; i <= n; ++i)
10         wn[i] = wn[i - 1] * g % MOD;
11     return n;
12 }
13
14 void NTT(LL a[], int n, int f) {
15     FOR (i, 0, n) if (i < rev[i])
16         std::swap(a[i], a[rev[i]]);
17     for (int k = 1; k < n; k <= 1) {
18         for (int i = 0; i < n; i += (k << 1)) {
19             int t = n / (k << 1);
20             FOR (j, 0, k) {
21                 LL w = f == 1 ? wn[t * j] : wn[n - t * j];
22                 LL x = a[i + j];
23                 LL y = a[i + j + k] * w % MOD;
24                 a[i + j] = (x + y) % MOD;
25                 a[i + j + k] = (x - y + MOD) % MOD;
26             }
27         }
28     }
29     if (f == -1) {
30         LL ninv = get_inv(n, MOD);
31         FOR (i, 0, n)
32             a[i] = a[i] * ninv % MOD;
33     }
34 }

```

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

```

1 typedef double LD;
2 const LD PI = acos(-1);
3 struct C {
4     LD r, i;

```

```

5     C(LD r = 0, LD i = 0): r(r), i(i) {}
6 };
7 C operator + (const C& a, const C& b) {
8     return C(a.r + b.r, a.i + b.i);
9 }
10 C operator - (const C& a, const C& b) {
11     return C(a.r - b.r, a.i - b.i);
12 }
13 C operator * (const C& a, const C& b) {
14     return C(a.r * b.r - a.i * b.i, a.r * b.i + a.i * b.r);
15 }
16
17 void FFT(C x[], int n, int p) {
18     for (int i = 0, t = 0; i < n; ++i) {
19         if (i > t) swap(x[i], x[t]);
20         for (int j = n >> 1; (t ^= j) < j; j >>= 1);
21     }
22     for (int h = 2; h <= n; h <= 1) {
23         C wn(cos(p * 2 * PI / h), sin(p * 2 * PI / h));
24         for (int i = 0; i < n; i += h) {
25             C w(1, 0), u;
26             for (int j = i, k = h >> 1; j < i + k; ++j) {
27                 u = x[j + k] * w;
28                 x[j + k] = x[j] - u;
29                 x[j] = x[j] + u;
30                 w = w * wn;
31             }
32         }
33     }
34     if (p == -1)
35         FOR (i, 0, n)
36             x[i].r /= n;
37 }
38
39 void conv(C a[], C b[], int n) {
40     FFT(a, n, 1);
41     FFT(b, n, 1);
42     FOR (i, 0, n)
43         a[i] = a[i] * b[i];
44     FFT(a, n, -1);
45 }

```

● FWT

```

1 template<typename T>
2 void fwt(LL a[], int n, T f) {
3     for (int d = 1; d < n; d *= 2)
4         for (int i = 0, t = d * 2; i < n; i += t)
5             FOR (j, 0, d)
6                 f(a[i + j], a[i + j + d]);
7 }
8
9 void AND(LL& a, LL& b) { a += b; }
10 void OR(LL& a, LL& b) { b += a; }
11 void XOR (LL& a, LL& b) {
12     LL x = a, y = b;
13     a = (x + y) % MOD;
14     b = (x - y + MOD) % MOD;
15 }

```

simpson 自适应积分

```

1 LD simpson(LD l, LD r) {
2     LD c = (l + r) / 2;
3     return (f(l) + 4 * f(c) + f(r)) * (r - l) / 6;
4 }
5
6 LD asr(LD l, LD r, LD eps, LD S) {
7     LD m = (l + r) / 2;
8     LD L = simpson(l, m), R = simpson(m, r);

```

```

9     if (fabs(L + R - S) < 15 * eps) return L + R + (L + R - S) / 15;
10    return asr(l, m, eps / 2, L) + asr(m, r, eps / 2, R);
11 }
12
13 LD asr(LD l, LD r, LD eps) { return asr(l, r, eps, simpson(l, r)); }

```

● FWT

```

1  template<typename T>
2  void fwt(LL a[], int n, T f) {
3      for (int d = 1; d < n; d *= 2)
4          for (int i = 0, t = d * 2; i < n; i += t)
5              FOR (j, 0, d)
6                  f(a[i + j], a[i + j + d]);
7  }
8
9  auto f = [] (LL& a, LL& b) { // xor
10      LL x = a, y = b;
11      a = (x + y) % MOD;
12      b = (x - y + MOD) % MOD;
13  };

```

快速乘

```

1  LL mul(LL a, LL b, LL m) {
2      LL ret = 0;
3      while (b) {
4          if (b & 1) {
5              ret += a;
6              if (ret >= m) ret -= m;
7          }
8          a += a;
9          if (a >= m) a -= m;
10         b >>= 1;
11     }
12     return ret;
13 }

```

● O(1)

```

1  LL mul(LL u, LL v, LL p) {
2      return (u * v - LL((long double) u * v / p) * p + p) % p;
3  }

```

快速幂

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

```

1  LL bin(LL x, LL n, LL MOD) {
2      LL ret = MOD != 1;
3      for (x %= MOD; n; n >>= 1, x = x * x % MOD)
4          if (n & 1) ret = ret * x % MOD;
5      return ret;
6  }

```

● 防爆 LL

● 前置模板：快速乘

```

1  LL bin(LL x, LL n, LL MOD) {
2      LL ret = MOD != 1;
3      for (x %= MOD; n; n >>= 1, x = mul(x, x, MOD))
4          if (n & 1) ret = mul(ret, x, MOD);
5      return ret;
6  }

```

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

```
1  typedef double LD;
2  const LD eps = 1E-10;
3  const int maxn = 2000 + 10;
4
5  int n, m;
6  LD a[maxn][maxn], x[maxn];
7  bool free_x[maxn];
8
9  inline int sgn(LD x) { return (x > eps) - (x < -eps); }
10
11
12 int guass(LD a[maxn][maxn], int n, int m) {
13     memset(free_x, 1, sizeof free_x); memset(x, 0, sizeof x);
14     int r = 0, c = 0;
15     while (r < n && c < m) {
16         int m_r = r;
17         FOR (i, r + 1, n)
18             if (fabs(a[i][c]) > fabs(a[m_r][c])) m_r = i;
19         if (m_r != r)
20             FOR (j, c, m + 1)
21                 swap(a[r][j], a[m_r][j]);
22         if (!sgn(a[r][c])) {
23             a[r][c] = 0;
24             ++c;
25             continue;
26         }
27         FOR (i, r + 1, n)
28             if (a[i][c]) {
29                 LD t = a[i][c] / a[r][c];
30                 FOR (j, c, m + 1) a[i][j] -= a[r][j] * t;
31             }
32         ++r; ++c;
33         // FOR (i, 0, n)
34         //     FOR (j, 0, m + 1)
35         //         printf("%.2f%c", a[i][j], j == _j - 1 ? '\n' : ' '); puts("");
36     }
37     FOR (i, r, n)
38         if (sgn(a[i][m])) return -1;
39     if (r < m) {
40         FORD (i, r - 1, -1) {
41             int f_cnt = 0, k = -1;
42             FOR (j, 0, m)
43                 if (sgn(a[i][j]) && free_x[j]) {
44                     ++f_cnt;
45                     k = j;
46                 }
47             if (f_cnt > 0) continue;
48             LD s = a[i][m];
49             FOR (j, 0, m)
50                 if (j != k) s -= a[i][j] * x[j];
51             x[k] = s / a[i][k];
52             free_x[k] = 0;
53         }
54         return m - r;
55     }
56     FORD (i, m - 1, -1) {
57         LD s = a[i][m];
58         FOR (j, i + 1, m)
59             s -= a[i][j] * x[j];
60         x[i] = s / a[i][i];
61     }
62     return 0;
63 }
```


- 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

```

1 LL factor[30], f_sz, factor_exp[30];
2 void get_factor(LL x) {
3     f_sz = 0;
4     LL t = sqrt(x + 0.5);
5     for (LL i = 0; pr[i] <= t; ++i)
6         if (x % pr[i] == 0) {
7             factor_exp[f_sz] = 0;
8             while (x % pr[i] == 0) {
9                 x /= pr[i];
10                ++factor_exp[f_sz];
11            }
12            factor[f_sz++] = pr[i];
13        }
14    if (x > 1) {
15        factor_exp[f_sz] = 1;
16        factor[f_sz++] = x;
17    }
18 }

```

- 不带指数

```

1 LL factor[30], f_sz;
2 void get_factor(LL x) {
3     f_sz = 0;
4     LL t = sqrt(x + 0.5);
5     for (LL i = 0; pr[i] <= t; ++i)
6         if (x % pr[i] == 0) {
7             factor[f_sz++] = pr[i];
8             while (x % pr[i] == 0) x /= pr[i];
9         }
10    if (x > 1) factor[f_sz++] = x;
11 }

```

原根

- 前置模板：素数筛，快速幂，分解质因数
- 要求 p 为质数

```
1 LL find_smallest_primitive_root(LL p) {
2     get_factor(p - 1);
3     FOR (i, 2, p) {
4         bool flag = true;
5         FOR (j, 0, f_sz)
6             if (bin(i, (p - 1) / factor[j], p) == 1) {
7                 flag = false;
8                 break;
9             }
10        if (flag) return i;
11    }
12    assert(0); return -1;
13 }
```

公式

- 当 $x \geq \phi(p)$ 时有 $a^x \equiv a^{x \bmod \phi(p) + \phi(p)} \pmod{p}$

斐波那契数列性质

- $F_{a+b} = F_{a-1} \cdot F_b + F_a \cdot F_{b+1}$

常见生成函数

- $(1 + ax)^n = \sum_{k=0}^n \binom{n}{k} a^k x^k$
- $\frac{1 - x^{r+1}}{1 - x} = \sum_{k=0}^n x^k$
- $\frac{1}{1 - ax} = \sum_{k=0}^{\infty} a^k x^k$
- $\frac{1}{(1 - x)^2} = \sum_{k=0}^{\infty} (k + 1) x^k$
- $\frac{1}{(1 - x)^n} = \sum_{k=0}^{\infty} \binom{n+k-1}{k} x^k$
- $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$
- $\ln(1 + x) = \sum_{k=0}^{\infty} \frac{(-1)^{k+1}}{k} x^k$

中国剩余定理

- 无解返回 -1
- 前置模板：拓展欧几里得

```
1 LL CRT(LL *m, LL *r, LL n) {
2     if (!n) return 0;
3     LL M = m[0], R = r[0], x, y, d;
4     FOR (i, 1, n) {
5         d = ex_gcd(M, m[i], x, y);
6         if ((r[i] - R) % d) return -1;
7         x = (r[i] - R) / d * x % (m[i] / d);
8         R += x * M;
9         M = M / d * m[i];
10        R %= M;
11    }
12    return R >= 0 ? R : R + M;
13 }
```

伯努利数和等幂求和

- 预处理逆元
- 预处理组合数
- $\sum_{i=0}^n i^k = \frac{1}{k+1} \sum_{i=0}^k \binom{k+1}{i} B_{k+1-i} (n+1)^i$.
- 也可以 $\sum_{i=0}^n i^k = \frac{1}{k+1} \sum_{i=0}^k \binom{k+1}{i} B_{k+1-i}^+ n^i$ 。区别在于 $B_1^+ = 1/2$ 。(心态崩了)

```
1 namespace Bernoulli {
2     const int M = 100;
3     LL inv[M] = {-1, 1};
4     void inv_init(LL n, LL p) {
5         FOR (i, 2, n)
6             inv[i] = (p - p / i) * inv[p % i] % p;
7     }
8
9     LL C[M][M];
10    void init_C(int n) {
11        FOR (i, 0, n) {
12            C[i][0] = C[i][i] = 1;
13            FOR (j, 1, i)
14                C[i][j] = (C[i-1][j] + C[i-1][j-1]) % MOD;
15        }
16    }
17
18    LL B[M] = {1};
19    void init() {
20        inv_init(M, MOD);
21        init_C(M);
22        FOR (i, 1, M-1) {
23            LL& s = B[i] = 0;
24            FOR (j, 0, i)
25                s += C[i+1][j] * B[j] % MOD;
26            s = (s % MOD * -inv[i+1] % MOD + MOD) % MOD;
27        }
28    }
29
30    LL p[M] = {1};
31    LL go(LL n, LL k) {
32        n %= MOD;
33        if (k == 0) return n;
34        FOR (i, 1, k+2)
35            p[i] = p[i-1] * (n+1) % MOD;
36        LL ret = 0;
37        FOR (i, 1, k+2)
38            ret += C[k+1][i] * B[k+1-i] % MOD * p[i] % MOD;
39        ret = ret % MOD * inv[k+1] % MOD;
40        return ret;
41    }
42 }
```

单纯形

- 要求有基本解，也就是 x 为零向量可行
- v 要初始化为 0, n 表示向量长度, m 表示约束个数

```
1 // min{ b x } / max { c x }
2 // A x >= c / A x <= b
3 // x >= 0
4 namespace lp {
5     int n, m;
6     double a[M][N], b[M], c[N], v;
7
8     void pivot(int l, int e) {
9         b[l] /= a[l][e];
10        FOR (j, 0, n) if (j != e) a[l][j] /= a[l][e];
11        a[l][e] = 1 / a[l][e];
12
13        FOR (i, 0, m)
```

```

14         if (i != l && fabs(a[i][e]) > 0) {
15             b[i] -= a[i][e] * b[l];
16             FOR (j, 0, n)
17                 if (j != e) a[i][j] -= a[i][e] * a[l][j];
18             a[i][e] = -a[i][e] * a[l][e];
19         }
20     v += c[e] * b[l];
21     FOR (j, 0, n) if (j != e) c[j] -= c[e] * a[l][j];
22     c[e] = -c[e] * a[l][e];
23 }
24 double simplex() {
25     while (1) {
26         int e = -1, l = -1;
27         FOR (i, 0, n) if (c[i] > eps) { e = i; break; }
28         if (e == -1) return v;
29         double t = INF;
30         FOR (i, 0, m)
31             if (a[i][e] > eps && t > b[i] / a[i][e]) {
32                 t = b[i] / a[i][e];
33                 l = i;
34             }
35         if (l == -1) return INF;
36         pivot(l, e);
37     }
38 }
39 }

```

图论

LCA

● 倍增

```

1 void dfs(int u, int fa) {
2     pa[u][0] = fa; dep[u] = dep[fa] + 1;
3     FOR (i, 1, SP) pa[u][i] = pa[pa[u][i-1]][i-1];
4     for (int& v: G[u]) {
5         if (v == fa) continue;
6         dfs(v, u);
7     }
8 }
9
10 int lca(int u, int v) {
11     if (dep[u] < dep[v]) swap(u, v);
12     int t = dep[u] - dep[v];
13     FOR (i, 0, SP) if (t & (1 << i)) u = pa[u][i];
14     FORD (i, SP-1, -1) {
15         int uu = pa[u][i], vv = pa[v][i];
16         if (uu != vv) { u = uu; v = vv; }
17     }
18     return u == v ? u : pa[u][0];
19 }

```

最短路

```

1 bool BF() {
2     queue<int> q;
3     FOR (i, 1, n) d[i] = INF;
4     d[0] = 0; inq[0] = true; q.push(0);
5     while (!q.empty()) {
6         int u = q.front(); q.pop();
7         inq[u] = false;
8         for (E& e: G[u]) {
9             int v = e.to;
10            if (d[u] < INF && d[v] > d[u] + e.d) {
11                d[v] = d[u] + e.d;

```

```

12         if (!inq[v]) {
13             q.push(v); inq[v] = true;
14             if (++cnt[v] > n) return false;
15         }
16     }
17 }
18 }
19 return true;
20 }

```

网络流

• 最大流

```

1  struct E {
2      int to, cp;
3      E(int to, int cp): to(to), cp(cp) {}
4  };
5
6  struct Dinic {
7      static const int M = 1E5 * 5;
8      int m, s, t;
9      vector<E> edges;
10     vector<int> G[M];
11     int d[M];
12     int cur[M];
13
14     void init(int n, int s, int t) {
15         this->s = s; this->t = t;
16         for (int i = 0; i <= n; i++) G[i].clear();
17         edges.clear(); m = 0;
18     }
19
20     void addedge(int u, int v, int cap) {
21         edges.emplace_back(v, cap);
22         edges.emplace_back(u, 0);
23         G[u].push_back(m++);
24         G[v].push_back(m++);
25     }
26
27     bool BFS() {
28         memset(d, 0, sizeof d);
29         queue<int> Q;
30         Q.push(s); d[s] = 1;
31         while (!Q.empty()) {
32             int x = Q.front(); Q.pop();
33             for (int& i: G[x]) {
34                 E &e = edges[i];
35                 if (!d[e.to] && e.cp > 0) {
36                     d[e.to] = d[x] + 1;
37                     Q.push(e.to);
38                 }
39             }
40         }
41         return d[t];
42     }
43
44     int DFS(int u, int cp) {
45         if (u == t || !cp) return cp;
46         int tmp = cp, f;
47         for (int& i = cur[u]; i < G[u].size(); i++) {
48             E& e = edges[G[u][i]];
49             if (d[u] + 1 == d[e.to]) {
50                 f = DFS(e.to, min(cp, e.cp));
51                 e.cp -= f;
52                 edges[G[u][i] ^ 1].cp += f;
53                 cp -= f;
54                 if (!cp) break;
55             }

```

```

56     }
57     return tmp - cp;
58 }
59
60 int go() {
61     int flow = 0;
62     while (BFS()) {
63         memset(cur, 0, sizeof cur);
64         flow += DFS(s, INF);
65     }
66     return flow;
67 }
68 } DC;

```

● 费用流

```

1  struct E {
2      int from, to, cp, v;
3      E() {}
4      E(int f, int t, int cp, int v) : from(f), to(t), cp(cp), v(v) {}
5  };
6
7  struct MCMF {
8      int n, m, s, t;
9      vector<E> edges;
10     vector<int> G[maxn];
11     bool inq[maxn]; //是否在队列
12     int d[maxn]; //Bellman_ford 单源最短路径
13     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
14     int a[maxn]; //a[i] 表示从 s 到 i 的最小残量
15
16     void init(int _n, int _s, int _t) {
17         n = _n; s = _s; t = _t;
18         FOR (i, 0, n + 1) G[i].clear();
19         edges.clear(); m = 0;
20     }
21
22     void addedge(int from, int to, int cap, int cost) {
23         edges.emplace_back(from, to, cap, cost);
24         edges.emplace_back(to, from, 0, -cost);
25         G[from].push_back(m++);
26         G[to].push_back(m++);
27     }
28
29     bool BellmanFord(int &flow, int &cost) {
30         FOR (i, 0, n + 1) d[i] = INF;
31         memset(inq, 0, sizeof inq);
32         d[s] = 0, a[s] = INF, inq[s] = true;
33         queue<int> Q; Q.push(s);
34         while (!Q.empty()) {
35             int u = Q.front(); Q.pop();
36             inq[u] = false;
37             for (int& idx: G[u]) {
38                 E &e = edges[idx];
39                 if (e.cp && d[e.to] > d[u] + e.v) {
40                     d[e.to] = d[u] + e.v;
41                     p[e.to] = idx;
42                     a[e.to] = min(a[u], e.cp);
43                     if (!inq[e.to]) {
44                         Q.push(e.to);
45                         inq[e.to] = true;
46                     }
47                 }
48             }
49         }
50         if (d[t] == INF) return false;
51         flow += a[t];
52         cost += a[t] * d[t];
53         int u = t;
54         while (u != s) {
55             edges[p[u]].cp -= a[t];
56             edges[p[u] ^ 1].cp += a[t];

```

```

57         u = edges[p[u]].from;
58     }
59     return true;
60 }
61
62 int go() {
63     int flow = 0, cost = 0;
64     while (BellmanFord(flow, cost));
65     return cost;
66 }
67 } MM;

```

- zkw 费用流（代码长度没有优势）
- 不允许有负权边

```

1  struct E {
2      int to, cp, v;
3      E() {}
4      E(int to, int cp, int v): to(to), cp(cp), v(v) {}
5  };
6
7  struct MCMF {
8      int n, m, s, t, cost, D;
9      vector<E> edges;
10     vector<int> G[maxn];
11     bool vis[maxn];
12
13     void init(int _n, int _s, int _t) {
14         n = _n; s = _s; t = _t;
15         FOR (i, 0, n + 1) G[i].clear();
16         edges.clear(); m = 0;
17     }
18
19     void addedge(int from, int to, int cap, int cost) {
20         edges.emplace_back(to, cap, cost);
21         edges.emplace_back(from, 0, -cost);
22         G[from].push_back(m++);
23         G[to].push_back(m++);
24     }
25
26     int aug(int u, int cp) {
27         if (u == t) {
28             cost += D * cp;
29             return cp;
30         }
31         vis[u] = true;
32         int tmp = cp;
33         for (int idx: G[u]) {
34             E& e = edges[idx];
35             if (e.cp && !e.v && !vis[e.to]) {
36                 int f = aug(e.to, min(cp, e.cp));
37                 e.cp -= f;
38                 edges[idx ^ 1].cp += f;
39                 cp -= f;
40                 if (!cp) break;
41             }
42         }
43         return tmp - cp;
44     }
45
46     bool modlabel() {
47         int d = INF;
48         FOR (u, 0, n + 1)
49             if (vis[u])
50                 for (int& idx: G[u]) {
51                     E& e = edges[idx];
52                     if (e.cp && !vis[e.to]) d = min(d, e.v);
53                 }
54         if (d == INF) return false;
55         FOR (u, 0, n + 1)
56             if (vis[u])

```

```

57         for (int& idx: G[u]) {
58             edges[idx].v -= d;
59             edges[idx ^ 1].v += d;
60         }
61         D += d;
62         return true;
63     }
64
65     int go(int k) {
66         cost = D = 0;
67         int flow = 0;
68         while (true) {
69             memset(vis, 0, sizeof vis);
70             int t = aug(s, INF);
71             if (!t && !modlabel()) break;
72             flow += t;
73         }
74         return cost;
75     }
76 } MM;

```

树上路径交

```

1  int intersection(int x, int y, int xx, int yy) {
2      int t[4] = {lca(x, xx), lca(x, yy), lca(y, xx), lca(y, yy)};
3      sort(t, t + 4);
4      int r = lca(x, y), rr = lca(xx, yy);
5      if (dep[t[0]] < min(dep[r], dep[rr]) || dep[t[2]] < max(dep[r], dep[rr]))
6          return 0;
7      int tt = lca(t[2], t[3]);
8      int ret = 1 + dep[t[2]] + dep[t[3]] - dep[tt] * 2;
9      return ret;
10 }

```

树上点分治

```

1  int get_sz(int u, int fa) {
2      int& s = sz[u] = 1;
3      for (E& e: G[u]) {
4          int v = e.to;
5          if (vis[v] || v == fa) continue;
6          s += get_sz(v, u);
7      }
8      return s;
9  }
10
11 void get_rt(int u, int fa, int s, int& m, int& rt) {
12     int t = s - sz[u];
13     for (E& e: G[u]) {
14         int v = e.to;
15         if (vis[v] || v == fa) continue;
16         get_rt(v, u, s, m, rt);
17         t = max(t, sz[v]);
18     }
19     if (t < m) { m = t; rt = u; }
20 }
21
22 void dfs(int u) {
23     int tmp = INF; get_rt(u, -1, get_sz(u, -1), tmp, u);
24     vis[u] = true;
25     get_dep(u, -1, 0);
26     // ...
27     for (E& e: G[u]) {
28         int v = e.to;
29         if (vis[v]) continue;
30         // ...
31         dfs(v);

```



```

32     }
33 }

```

● 动态点分治

```

1  const int maxn = 15E4 + 100, INF = 1E9;
2  struct E {
3      int to, d;
4  };
5  vector<E> G[maxn];
6  int n, Q, w[maxn];
7  LL A, ans;
8
9  bool vis[maxn];
10 int sz[maxn];
11
12 int get_rt(int u) {
13     // dbg(u);
14     static int q[N], fa[N], sz[N], mx[N];
15     int p = 0, cur = -1;
16     q[p++] = u; fa[u] = -1;
17     while (++cur < p) {
18         u = q[cur]; mx[u] = 0; sz[u] = 1;
19         for (int& v: G[u])
20             if (!vis[v] && v != fa[u]) fa[q[p++]] = v; u;
21     }
22     FORD (i, p - 1, -1) {
23         u = q[i];
24         mx[u] = max(mx[u], p - sz[u]);
25         if (mx[u] * 2 <= p) return u;
26         sz[fa[u]] += sz[u];
27         mx[fa[u]] = max(mx[fa[u]], sz[u]);
28     }
29     assert(0);
30 }
31
32 int get_sz(int u, int fa) {
33     int& s = sz[u] = 1;
34     for (E& e: G[u]) {
35         int v = e.to;
36         if (vis[v] || v == fa) continue;
37         s += get_sz(v, u);
38     }
39     return s;
40 }
41
42 void get_rt(int u, int fa, int s, int& m, int& rt) {
43     int t = s - sz[u];
44     for (E& e: G[u]) {
45         int v = e.to;
46         if (vis[v] || v == fa) continue;
47         get_rt(v, u, s, m, rt);
48         t = max(t, sz[v]);
49     }
50     if (t < m) { m = t; rt = u; }
51 }
52
53 int dep[maxn], md[maxn];
54 void get_dep(int u, int fa, int d) {
55     dep[u] = d; md[u] = 0;
56     for (E& e: G[u]) {
57         int v = e.to;
58         if (vis[v] || v == fa) continue;
59         get_dep(v, u, d + e.d);
60         md[u] = max(md[u], md[v] + 1);
61     }
62 }
63
64 struct P {
65     int w;
66     LL s;
67 };

```

```

68 using VP = vector<P>;
69 struct R {
70     VP *rt, *rt2;
71     int dep;
72 };
73 VP pool[maxn << 1], *pit = pool;
74 vector<R> tr[maxn];
75
76 void go(int u, int fa, VP* rt, VP* rt2) {
77     tr[u].push_back({rt, rt2, dep[u]});
78     for (E& e: G[u]) {
79         int v = e.to;
80         if (v == fa || vis[v]) continue;
81         go(v, u, rt, rt2);
82     }
83 }
84
85 void dfs(int u) {
86     int tmp = INF; get_rt(u, -1, get_sz(u, -1), tmp, u);
87     vis[u] = true;
88     get_dep(u, -1, 0);
89     VP* rt = pit++; tr[u].push_back({rt, nullptr, 0});
90     for (E& e: G[u]) {
91         int v = e.to;
92         if (vis[v]) continue;
93         go(v, u, rt, pit++);
94         dfs(v);
95     }
96 }
97
98 bool cmp(const P& a, const P& b) { return a.w < b.w; }
99
100 LL query(VP& p, int d, int l, int r) {
101     l = lower_bound(p.begin(), p.end(), P{l, -1}, cmp) - p.begin();
102     r = upper_bound(p.begin(), p.end(), P{r, -1}, cmp) - p.begin() - 1;
103     return p[r].s - p[l - 1].s + 1LL * (r - l + 1) * d;
104 }
105
106 int main() {
107     cin >> n >> Q >> A;
108     FOR (i, 1, n + 1) scanf("%d", &w[i]);
109     FOR (_, 1, n) {
110         int u, v, d; scanf("%d%d%d", &u, &v, &d);
111         G[u].push_back({v, d}); G[v].push_back({u, d});
112     }
113     dfs(1);
114     FOR (i, 1, n + 1)
115         for (R& x: tr[i]) {
116             x.rt->push_back({w[i], x.dep});
117             if (x.rt2) x.rt2->push_back({w[i], x.dep});
118         }
119     FOR (it, pool, pit) {
120         it->push_back({-INF, 0});
121         sort(it->begin(), it->end(), cmp);
122         FOR (i, 1, it->size())
123             (*it)[i].s += (*it)[i - 1].s;
124     }
125     while (Q--) {
126         int u; LL a, b; scanf("%d%lld%lld", &u, &a, &b);
127         a = (a + ans) % A; b = (b + ans) % A;
128         int l = min(a, b), r = max(a, b);
129         ans = 0;
130         for (R& x: tr[u]) {
131             ans += query(*(x.rt), x.dep, l, r);
132             if (x.rt2) ans -= query(*(x.rt2), x.dep, l, r);
133         }
134         printf("%lld\n", ans);
135     }
136 }

```

树链剖分

```

1  int fa[N], dep[N], idx[N], out[N], ridx[N];
2  namespace hld {
3      int sz[N], son[N], top[N], clk;
4      void predfs(int u, int d) {
5          dep[u] = d; sz[u] = 1;
6          int& maxs = son[u] = -1;
7          for (int& v: G[u]) {
8              if (v == fa[u]) continue;
9              fa[v] = u;
10             predfs(v, d + 1);
11             sz[u] += sz[v];
12             if (maxs == -1 || sz[v] > sz[maxs]) maxs = v;
13         }
14     }
15     void dfs(int u, int tp) {
16         top[u] = tp; idx[u] = ++clk; ridx[clk] = u;
17         if (son[u] != -1) dfs(son[u], tp);
18         for (int& v: G[u])
19             if (v != fa[u] && v != son[u]) dfs(v, v);
20         out[u] = clk;
21     }
22     template<typename T>
23     int go(int u, int v, T&& f = [] (int, int) {}) {
24         int uu = top[u], vv = top[v];
25         while (uu != vv) {
26             if (dep[uu] < dep[vv]) { swap(uu, vv); swap(u, v); }
27             f(idx[uu], idx[u]);
28             u = fa[uu]; uu = top[u];
29         }
30         if (dep[u] < dep[v]) swap(u, v);
31         // f(idx[v], idx[u]);
32         // if (u != v) f(idx[v] + 1, idx[u]);
33         return v;
34     }
35     int up(int u, int d) {
36         while (d) {
37             if (dep[u] - dep[top[u]] < d) {
38                 d -= dep[u] - dep[top[u]];
39                 u = top[u];
40             } else return ridx[idx[u] - d];
41             u = fa[u]; --d;
42         }
43         return u;
44     }
45 }

```

● HDU 3966

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long LL;
4  #define FOR(i, x, y) for (decay<decltype(y)>::type i = (x), _##i = (y); i < _##i; ++i)
5  #define FORD(i, x, y) for (decay<decltype(x)>::type i = (x), _##i = (y); i > _##i; --i)
6  #ifdef zero1
7  #define dbg(args...) do { cout << "\033[32;1m" << #args << " -> "; err(args); } while (0)
8  #else
9  #define dbg(...)
10 #endif
11 void err() { cout << "\033[39;0m" << endl; }
12 template<typename T, typename... Args>
13 void err(T a, Args... args) { cout << a << ' '; err(args...); }
14 // -----
15 const int maxn = 5E4 + 100;
16 vector<int> G[maxn];
17 int dep[maxn], sz[maxn], son[maxn], fa[maxn], idx[maxn], top[maxn];
18 int clk, n, Q;
19
20 struct IntervalTree {
21     #define ls o * 2, l, (l + r) >> 1

```

```

22 #define rs o * 2 + 1, ((l + r) >> 1) + 1, r
23 static const int M = maxn << 2;
24 int addv[M];
25 void init() { memset(addv, 0, sizeof addv); }
26 int query(int k, int o, int l, int r, int add = 0) {
27     if (k < l || r < k) return 0;
28     if (l == r) return add + addv[o];
29     return query(k, ls, add + addv[o]) + query(k, rs, add + addv[o]);
30 }
31 void update(int p, int q, int o, int l, int r, int add) {
32     assert(l <= r && r <= n);
33     if (q < l || r < p) return;
34     if (p <= l && r <= q) addv[o] += add;
35     else { update(p, q, ls, add); update(p, q, rs, add); }
36 }
37 } IT;
38
39 void predfs(int u, int d) {
40     dep[u] = d;
41     sz[u] = 1;
42     int& maxs = son[u] = -1;
43     for (int v: G[u])
44         if (v != fa[u]) {
45             fa[v] = u;
46             predfs(v, d + 1);
47             sz[u] += sz[v];
48             if (maxs == -1 || sz[v] > sz[maxs])
49                 maxs = v;
50         }
51 }
52
53 void dfs(int u, int tp) {
54     top[u] = tp;
55     idx[u] = ++clk;
56     if (son[u] != -1) dfs(son[u], tp);
57     for (int v: G[u])
58         if (v != son[u] && v != fa[u])
59             dfs(v, v);
60 }
61
62 void update(int u, int v, int add) {
63     int uu = top[u], vv = top[v];
64     while (uu != vv) {
65         if (dep[uu] < dep[vv]) { swap(uu, vv); swap(u, v); }
66         IT.update(idx[uu], idx[u], 1, 1, n, add);
67         u = fa[uu];
68         uu = top[u];
69     }
70     if (dep[u] < dep[v]) swap(u, v);
71     dbg(u, v, idx[u], idx[v]);
72     IT.update(idx[v], idx[u], 1, 1, n, add);
73 }
74
75 int a[maxn];
76 void init();
77 int main() {
78     int u, v, l, r, k, d;
79     char s[100];
80     while (cin >> n >> Q >> Q) {
81         init();
82         FOR (i, 1, n + 1) scanf("%d", &a[i]);
83         FOR (i, 1, n) {
84             scanf("%d%d", &u, &v);
85             G[u].push_back(v);
86             G[v].push_back(u);
87         }
88         predfs(1, 1);
89         dfs(1, 1);
90         while (Q--) {
91             scanf("%s", s);
92             if (s[0] == 'I') {

```

```

93         scanf("%d%d%d", &l, &r, &d);
94         update(l, r, d);
95     } else if (s[0] == 'D') {
96         scanf("%d%d%d", &l, &r, &d);
97         update(l, r, -d);
98     } else {
99         scanf("%d", &k);
100         printf("%d\n", a[k] + IT.query(idx[k], 1, 1, n));
101     }
102 }
103 }
104 }
105
106 void init() {
107     clk = 0;
108     fa[1] = 0;
109     IT.init();
110     FOR (i, 0, n + 1) G[i].clear();
111 }

```

• SPOJ QTREE

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long LL;
4  #define FOR(i, x, y) for (decay<decltype(y)>::type i = (x), _##i = (y); i < _##i; ++i)
5  #define FORD(i, x, y) for (decay<decltype(x)>::type i = (x), _##i = (y); i > _##i; --i)
6  #ifdef zero1
7  #define dbg(args...) do { cout << "\033[32;1m" << #args << " -> "; err(args); } while (0)
8  #else
9  #define dbg(...)
10 #endif
11 void err() { cout << "\033[39m" << endl; }
12 template<typename T, typename... Args>
13 void err(T a, Args... args) {
14     cout << a << ' ';
15     err(args...);
16 }
17 // -----
18 const int maxn = 10000 * 2 * 4 + 100;
19 struct Edge {
20     int from, to, c;
21     Edge(int u, int v, int c): from(u), to(v), c(c) {}
22 };
23 vector<Edge> edge;
24 vector<int> G[maxn];
25 int fa[maxn], dep[maxn], sz[maxn], son[maxn], top[maxn], idx[maxn], w[maxn], val[maxn];
26 LL sum[maxn];
27 int n, clk, len;
28
29 struct IntervalTree {
30     #define lson p, q, o * 2, l, m
31     #define rson p, q, o * 2 + 1, m + 1, r
32     int maxv[maxn];
33     void init() { memset(maxv, 0, sizeof maxv); }
34     int query(int p, int q, int o, int l, int r) {
35         // dbg(p, q);
36         assert(p <= q);
37         if (p > r || q < l) return 0;
38         if (p <= l && r <= q) return maxv[o];
39         int m = (l + r) / 2;
40         return max(query(lson), query(rson));
41     }
42     void maintain(int o, int l, int r) {
43         if (l < r)
44             maxv[o] = max(maxv[o * 2], maxv[o * 2 + 1]);
45     }
46     void update(int p, int q, int o, int l, int r, int v) {
47         // dbg(p, q, o, l, r, v);
48         assert(p <= q);
49         if (p > r || q < l) return;
50         if (p <= l && r <= q) maxv[o] = v;

```

```

51     else {
52         int m = (l + r) / 2;
53         update(lson, v); update(rson, v);
54         maintain(o, l, r);
55     }
56 }
57 } IT;
58
59 void dfs1(int u, int d) {
60     dep[u] = d;
61     sz[u] = 1;
62     FOR (i, 0, G[u].size()) {
63         Edge& e = edge[G[u][i]];
64         int v = e.to;
65         if (v == fa[u]) continue;
66         val[v] = e.c;
67         // dbg(v, e.from, e.to, e.c);
68         fa[v] = u;
69         dfs1(v, d + 1);
70         sz[u] += sz[v];
71         if (son[u] == -1 || sz[v] > sz[son[u]])
72             son[u] = v;
73     }
74 }
75
76 void dfs2(int u, int tp) {
77     top[u] = tp;
78     idx[u] = ++clk;
79     w[idx[u]] = tp;
80     if (son[u] == -1) return;
81     dfs2(son[u], tp);
82     FOR (i, 0, G[u].size()) {
83         int v = edge[G[u][i]].to;
84         if (v != son[u] && v != fa[u])
85             dfs2(v, v);
86     }
87 }
88
89 int query(int u, int v) {
90     dbg(u, v);
91     int uu = top[u], vv = top[v], ret = 0;
92     while (uu != vv) {
93         if (dep[uu] < dep[vv]) { swap(u, v); swap(uu, vv); }
94         // dbg(u, v, uu, vv, dep[uu], dep[vv], idx[uu], idx[u]);
95         ret = max(ret, IT.query(idx[uu], idx[u], 1, 1, len));
96         u = fa[uu];
97         uu = top[u];
98     }
99     if (dep[u] < dep[v]) swap(u, v);
100    // dbg(idx[v], idx[u]);
101    if (u != v) ret = max(ret, IT.query(idx[v] + 1, idx[u], 1, 1, len));
102    return ret;
103 }
104
105 void init();
106 void add_edge(int u, int v, int c);
107
108 int main() {
109     #ifdef zerol
110         freopen("in", "r", stdin);
111     #endif
112     int T, u, v, c;
113     char s[100];
114     cin >> T;
115     while (T--) {
116         cin >> n;
117         for (len = 1; len < n; len *= 2);
118         init();
119         FOR (i, 1, n) {
120             scanf("%d%d%d", &u, &v, &c);
121             add_edge(u, v, c);

```

```

122         add_edge(v, u, c);
123     }
124     dfs1(1, 0);
125     dfs2(1, 1);
126     // FOR (i, 1, n + 1) dbg(idx[i], w[i]);
127     FOR (i, 2, n + 1)
128         IT.update(idx[i], idx[i], 1, 1, len, val[i]);
129     while (scanf("%s", s) && s[0] != 'D') {
130         scanf("%d%d", &u, &v);
131         if (s[0] == 'C') {
132             Edge& e = edge[u * 2 - 1];
133             dbg(u, e.from, e.to);
134             int t = max(idx[e.from], idx[e.to]);
135             IT.update(t, t, 1, 1, len, v);
136             dbg("upd", t, v);
137         }
138         if (s[0] == 'Q') printf("%d\n", query(u, v));
139     }
140     FOR (i, 1, n + 1) if (idx[i] == 2) dbg(i, idx[i]);
141     dbg(IT.query(idx[2], idx[2], 1, 1, len));
142     dbg(IT.query(idx[6], idx[6], 1, 1, len));
143 }
144 }
145
146 void init() {
147     edge.clear();
148     memset(son, -1, sizeof son);
149     memset(sum, 0, sizeof sum);
150     IT.init();
151     FOR (i, 0, n + 1) G[i].clear();
152     clk = 0;
153     fa[1] = 0;
154     sum[0] = sum[1] = 0;
155 }
156
157 void add_edge(int u, int v, int c) {
158     edge.emplace_back(u, v, c);
159     G[u].push_back(edge.size() - 1);
160 }

```

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

```

1  struct MaxMatch {
2      int n;
3      vector<int> G[maxn];
4      int vis[maxn], left[maxn], clk;
5
6      void init(int n) {
7          this->n = n;
8          FOR (i, 0, n + 1) G[i].clear();
9          memset(left, -1, sizeof left);
10         memset(vis, -1, sizeof vis);
11     }
12
13     bool dfs(int u) {
14         for (int v: G[u])
15             if (vis[v] != clk) {
16                 vis[v] = clk;
17                 if (left[v] == -1 || dfs(left[v])) {
18                     left[v] = u;
19                     return true;
20                 }
21             }
22         return false;
23     }

```

```

24
25     int match() {
26         int ret = 0;
27         for (clk = 0; clk <= n; ++clk)
28             if (dfs(clk)) ++ret;
29         return ret;
30     }
31 } MM;

```

● 二分图最大权完美匹配 KM

```

1  namespace R {
2      const int maxn = 300 + 10;
3      int n, m;
4      int left[maxn], L[maxn], R[maxn];
5      int w[maxn][maxn], slack[maxn];
6      bool visL[maxn], visR[maxn];
7
8      bool dfs(int u) {
9          visL[u] = true;
10         FOR (v, 0, m) {
11             if (visR[v]) continue;
12             int t = L[u] + R[v] - w[u][v];
13             if (t == 0) {
14                 visR[v] = true;
15                 if (left[v] == -1 || dfs(left[v])) {
16                     left[v] = u;
17                     return true;
18                 }
19             } else slack[v] = min(slack[v], t);
20         }
21         return false;
22     }
23
24     int go() {
25         memset(left, -1, sizeof left);
26         memset(R, 0, sizeof R);
27         memset(L, 0, sizeof L);
28         FOR (i, 0, n)
29             FOR (j, 0, m)
30                 L[i] = max(L[i], w[i][j]);
31
32         FOR (i, 0, n) {
33             memset(slack, 0x3f, sizeof slack);
34             while (1) {
35                 memset(visL, 0, sizeof visL); memset(visR, 0, sizeof visR);
36                 if (dfs(i)) break;
37                 int d = 0x3f3f3f3f;
38                 FOR (j, 0, m) if (!visR[j]) d = min(d, slack[j]);
39                 FOR (j, 0, n) if (visL[j]) L[j] -= d;
40                 FOR (j, 0, m) if (visR[j]) R[j] += d; else slack[j] -= d;
41             }
42         }
43         int ret = 0;
44         FOR (i, 0, m) if (left[i] != -1) ret += w[left[i]][i];
45         return ret;
46     }
47 }

```

虚树

```

1  void go(vector<int>& V, int& k) {
2      int u = V[k]; f[u] = 0;
3      dbg(u, k);
4      for (auto& e: G[u]) {
5          int v = e.to;
6          if (v == pa[u][0]) continue;
7          while (k + 1 < V.size()) {
8              int to = V[k + 1];
9              if (in[to] <= out[v]) {

```



```

10         go(V, ++k);
11         if (key[to]) f[u] += w[to];
12         else f[u] += min(f[to], (LL)w[to]);
13     } else break;
14 }
15 }
16 dbg(u, f[u]);
17 }
18 inline bool cmp(int a, int b) { return in[a] < in[b]; }
19 LL solve(vector<int>& V) {
20     static vector<int> a; a.clear();
21     for (int& x: V) a.push_back(x);
22     sort(a.begin(), a.end(), cmp);
23     FOR (i, 1, a.size())
24         a.push_back(lca(a[i], a[i - 1]));
25     a.push_back(1);
26     sort(a.begin(), a.end(), cmp);
27     a.erase(unique(a.begin(), a.end()), a.end());
28     dbg(a);
29     int tmp; go(a, tmp = 0);
30     return f[1];
31 }

```

计算几何

圆的反演

```

1  typedef double LD;
2  const LD PI = 3.14159265358979323846;
3  const LD eps = 1E-10;
4  const LD R2 = 1.0;
5  int sgn(LD x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); }
6  struct P {
7      LD x, y;
8      P(LD x = 0, LD y = 0): x(x), y(y) {}
9      P operator * (LD k) { return P(x * k, y * k); }
10     P operator / (LD k) { return P(x / k, y / k); }
11     string prt() const {
12         char s[100];
13         sprintf(s, "(%.2f, %.2f)", x, y);
14         return string(s);
15     }
16 };
17 typedef P V;
18 P operator - (const P& a, const P& b) { return P(a.x - b.x, a.y - b.y); }
19 P operator + (const P& a, const P& b) { return P(a.x + b.x, a.y + b.y); }
20 struct C {
21     P p;
22     LD r;
23     C(LD x = 0, LD y = 0, LD r = 0): p(x, y), r(r) {}
24 };
25 LD dist(V v) { return sqrt(v.x * v.x + v.y * v.y); }
26
27 C inv(C c, const P& o) {
28     LD d = dist(c.p - o);
29     assert(sgn(d) != 0);
30     LD a = 1 / (d - c.r);
31     LD b = 1 / (d + c.r);
32     c.p = (a - b) / 2 * R2;
33     c.p = o + (c.p - o) * ((a + b) * R2 / 2 / d);
34     return c;
35 }

```

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

```

1  #define y1 yy1
2  #define nxt(i) ((i + 1) % s.size())
3  typedef double LD;
4  const LD PI = 3.14159265358979323846;
5  const LD eps = 1E-10;
6  int sgn(LD x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); }
7  struct L;
8  struct P;
9  //struct PP;
10 typedef P V;
11 struct P {
12     LD x, y;
13     explicit P(LD x = 0, LD y = 0): x(x), y(y) {}
14     P(const L& l);
15     // P(const PP& pp);
16     string prt() const {
17         char s[100];
18         sprintf(s, "(%.2f, %.2f)", x, y);
19         return string(s);
20     }
21 };
22 struct L {
23     P s, t;
24     L() {}
25     L(P s, P t): s(s), t(t) {}
26 };
27
28 P operator + (const P& a, const P& b) { return P(a.x + b.x, a.y + b.y); }
29 P operator - (const P& a, const P& b) { return P(a.x - b.x, a.y - b.y); }
30 P operator * (const P& a, LD k) { return P(a.x * k, a.y * k); }
31 P operator / (const P& a, LD k) { return P(a.x / k, a.y / k); }
32 bool operator == (const P& a, const P& b) { return !sgn(a.x - b.x) && !sgn(a.y - b.y); }
33 P::P(const L& l) { *this = l.t - l.s; }
34
35 // -----
36
37 //struct PP {
38 //    P p;
39 //    LD v, l;
40 //};
41 //P::P(const PP& pp) { *this = pp.p; }
42 typedef P PP;
43
44 typedef vector<PP> S;
45
46 // -----
47 LD dist(const P& p) { return sqrt(p.x * p.x + p.y * p.y); }
48 LD dot(const V& a, const V& b) { return a.x * b.x + a.y * b.y; }
49 LD det(const V& a, const V& b) { return a.x * b.y - a.y * b.x; }
50 LD cross(const P& s, const P& t, const P& o) { return det(s - o, t - o); }
51
52 // 如需支持 unique, 需要加 eps
53 bool cmp_xy(const P& a, const P& b) { return a.x < b.x || a.x == b.x && a.y < b.y; }
54
55 // 象限
56 int quad(P p) {
57     int x = sgn(p.x), y = sgn(p.y);
58     if (x > 0 && y >= 0) return 1;
59     if (x <= 0 && y > 0) return 2;
60     if (x < 0 && y <= 0) return 3;
61     if (x >= 0 && y < 0) return 4;
62     assert(0);
63 }
64
65 // 仅适用于参照点在所有点一侧的情况
66 struct cmp_angle {
67     P p;
68     bool operator () (const P& a, const P& b) {

```

```

69 //      int qa = quad(a), qb = quad(b);
70 //      if (qa != qb) return qa < qb;
71      int d = sgn(cross(a, b, p));
72      if (d) return d > 0;
73      return dist(a - p) < dist(b - p);
74  }
75 };
76
77 // -----线-----
78
79 // 是否平行
80 bool parallel(const L& a, const L& b) {
81     return !sgn(det(a, b));
82 }
83 // 直线是否相等
84 bool l_eq(const L& a, const L& b) {
85     return parallel(a, b) && parallel(L(a.s, b.t), L(b.s, a.t));
86 }
87 // 逆时针旋转 r 弧度
88 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)); }
89 // 单位法向量
90 V normal(const V& v) { return V(-v.y, v.x) / dist(v); }
91
92 // -----点和线-----
93
94 // 点在线段上  <= 0 包含端点 < 0 则不包含
95 bool p_on_seg(const P& p, const L& seg) {
96     P a = seg.s, b = seg.t;
97     return !sgn(det(p - a, b - a)) && sgn(dot(p - a, p - b)) <= 0;
98 }
99 // 点到直线距离
100 LD dist_to_line(const P& p, const L& l) {
101     return fabs(cross(l.s, l.t, p)) / dist(l);
102 }
103 // 点到线段距离
104 LD dist_to_seg(const P& p, const L& l) {
105     if (l.s == l.t) return dist(p - l);
106     V vs = p - l.s, vt = p - l.t;
107     if (sgn(dot(l, vs)) < 0) return dist(vs);
108     else if (sgn(dot(l, vt)) > 0) return dist(vt);
109     else return dist_to_line(p, l);
110 }
111
112 // -----线和线-----
113
114 // 求直线交 需要事先保证有界
115 P l_intersection(const L& a, const L& b) {
116     LD s1 = det(a, b.s - a.s), s2 = det(a, b.t - a.s);
117     return (b.s * s2 - b.t * s1) / (s2 - s1);
118 }
119 // 向量夹角的弧度
120 LD angle(const V& a, const V& b) {
121     LD r = asin(fabs(det(a, b)) / dist(a) / dist(b));
122     if (sgn(dot(a, b)) < 0) r = PI - r;
123     return r;
124 }
125 // 线段和直线是否有交 1 = 规范, 2 = 不规范
126 int s_l_cross(const L& seg, const L& line) {
127     int d1 = sgn(cross(line.s, line.t, seg.s));
128     int d2 = sgn(cross(line.s, line.t, seg.t));
129     if ((d1 ^ d2) == -2) return 1; // proper
130     if (d1 == 0 || d2 == 0) return 2;
131     return 0;
132 }
133 // 线段的交 1 = 规范, 2 = 不规范
134 int s_cross(const L& a, const L& b, P& p) {
135     int d1 = sgn(cross(a.t, b.s, a.s)), d2 = sgn(cross(a.t, b.t, a.s));
136     int d3 = sgn(cross(b.t, a.s, b.s)), d4 = sgn(cross(b.t, a.t, b.s));

```

```

140     if ((d1 ^ d2) == -2 && (d3 ^ d4) == -2) { p = l_intersection(a, b); return 1; }
141     if (!d1 && p_on_seg(b.s, a)) { p = b.s; return 2; }
142     if (!d2 && p_on_seg(b.t, a)) { p = b.t; return 2; }
143     if (!d3 && p_on_seg(a.s, b)) { p = a.s; return 2; }
144     if (!d4 && p_on_seg(a.t, b)) { p = a.t; return 2; }
145     return 0;
146 }
147
148
149 // -----多边形-----
150
151 // 点是否在多边形中 0 = 在外部 1 = 在内部 -1 = 在边界上
152 int inside(const S& s, const P& p) {
153     int cnt = 0;
154     FOR (i, 0, s.size()) {
155         P a = s[i], b = s[nxt(i)];
156         if (p_on_seg(p, L(a, b))) return -1;
157         if (sgn(a.y - b.y) <= 0) swap(a, b);
158         if (sgn(p.y - a.y) > 0) continue;
159         if (sgn(p.y - b.y) <= 0) continue;
160         cnt += sgn(cross(b, a, p)) > 0;
161     }
162     return bool(cnt & 1);
163 }
164 // 多边形面积
165 LD polygon_area(const S& s) {
166     LD ret = 0;
167     FOR (i, 1, (LL)s.size() - 1)
168         ret += cross(s[i], s[i + 1], s[0]);
169     return ret / 2;
170 }
171 // 构建凸包 点不可以重复 < 0 边上可以有点, <= 0 则不能
172 // 会改变输入点的顺序
173 const int MAX_N = 1000;
174 S convex_hull(S& s) {
175     // assert(s.size() >= 3);
176     sort(s.begin(), s.end(), cmp_xy);
177     S ret(MAX_N * 2);
178     int sz = 0;
179     FOR (i, 0, s.size()) {
180         while (sz > 1 && sgn(cross(ret[sz - 1], s[i], ret[sz - 2])) < 0) --sz;
181         ret[sz++] = s[i];
182     }
183     int k = sz;
184     FORD (i, (LL)s.size() - 2, -1) {
185         while (sz > k && sgn(cross(ret[sz - 1], s[i], ret[sz - 2])) < 0) --sz;
186         ret[sz++] = s[i];
187     }
188     ret.resize(sz - (s.size() > 1));
189     return ret;
190 }
191
192 // -----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）。
- rsort 需要初始化

```

1 namespace sam {
2     const int M = N << 1;
3     int t[M][26], len[M] = {-1}, fa[M], sz = 2, last = 1;
4     void ins(int ch) {

```

```

5     int p = last, np = last = sz++;
6     len[np] = len[p] + 1;
7     for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
8     if (!p) { fa[np] = 1; return; }
9     int q = t[p][ch];
10    if (len[p] + 1 == len[q]) fa[np] = q;
11    else {
12        int nq = sz++; len[nq] = len[p] + 1;
13        memcpy(t[nq], t[q], sizeof t[0]);
14        fa[nq] = fa[q];
15        fa[np] = fa[q] = nq;
16        for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
17    }
18 }
19
20 int c[M] = {1}, a[M];
21 void rsort() {
22     FOR (i, 1, sz) c[i] = 0;
23     FOR (i, 1, sz) c[len[i]]++;
24     FOR (i, 1, sz) c[i] += c[i - 1];
25     FOR (i, 1, sz) a[--c[len[i]]] = i;
26 }
27 }

```

● 真·广义后缀自动机

```

1  int t[M][26], len[M] = {-1}, fa[M], sz = 2, last = 1;
2  LL cnt[M][2];
3  void ins(int ch, int id) {
4      int p = last, np = 0, nq = 0, q = -1;
5      if (!t[p][ch]) {
6          np = sz++;
7          len[np] = len[p] + 1;
8          for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
9      }
10     if (!p) fa[np] = 1;
11     else {
12         q = t[p][ch];
13         if (len[p] + 1 == len[q]) fa[np] = q;
14         else {
15             nq = sz++; len[nq] = len[p] + 1;
16             memcpy(t[nq], t[q], sizeof t[0]);
17             fa[nq] = fa[q];
18             fa[np] = fa[q] = nq;
19             for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
20         }
21     }
22     last = np ? np : nq ? nq : q;
23     cnt[last][id] = 1;
24 }

```

● 按字典序建立后缀树注意逆序插入

```

1  void ins(int ch, int pp) {
2      int p = last, np = last = sz++;
3      len[np] = len[p] + 1; one[np] = pos[np] = pp;
4      for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
5      if (!p) { fa[np] = 1; return; }
6      int q = t[p][ch];
7      if (len[q] == len[p] + 1) fa[np] = q;
8      else {
9          int nq = sz++; len[nq] = len[p] + 1; one[nq] = one[q];
10         t[nq] = t[q];
11         fa[nq] = fa[q];
12         fa[q] = fa[np] = nq;
13         for (; p && t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
14     }
15 }
16
17 int up[M], c[256] = {2}, a[M];
18 void rsort2() {
19     FOR (i, 1, 256) c[i] = 0;

```

```

20     FOR (i, 2, sz) up[i] = s[one[i] + len[fa[i]]];
21     FOR (i, 2, sz) c[up[i]]++;
22     FOR (i, 1, 256) c[i] += c[i - 1];
23     FOR (i, 2, sz) a[--c[up[i]]] = i;
24     FOR (i, 2, sz) G[fa[a[i]]].push_back(a[i]);
25 }

```

- 广义后缀自动机建后缀树，必须反向插入

```

1  int t[M][26], len[M] = {0}, fa[M], sz = 2, last = 1;
2  char* one[M];
3  void ins(int ch, char* pp) {
4      int p = last, np = 0, nq = 0, q = -1;
5      if (!t[p][ch]) {
6          np = sz++; one[np] = pp;
7          len[np] = len[p] + 1;
8          for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
9      }
10     if (!p) fa[np] = 1;
11     else {
12         q = t[p][ch];
13         if (len[p] + 1 == len[q]) fa[np] = q;
14         else {
15             nq = sz++; len[nq] = len[p] + 1; one[nq] = one[q];
16             memcpy(t[nq], t[q], sizeof t[0]);
17             fa[nq] = fa[q];
18             fa[np] = fa[q] = nq;
19             for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
20         }
21     }
22     last = np ? np : nq ? nq : q;
23 }
24 int up[M], c[256] = {2}, aa[M];
25 vector<int> G[M];
26 void rsort() {
27     FOR (i, 1, 256) c[i] = 0;
28     FOR (i, 2, sz) up[i] = *(one[i] + len[fa[i]]);
29     FOR (i, 2, sz) c[up[i]]++;
30     FOR (i, 1, 256) c[i] += c[i - 1];
31     FOR (i, 2, sz) aa[--c[up[i]]] = i;
32     FOR (i, 2, sz) G[fa[aa[i]]].push_back(aa[i]);
33 }

```

- 匹配

```

1  int u = 1, l = 0;
2  FOR (i, 0, strlen(s)) {
3      int ch = s[i] - 'a';
4      while (u && !t[u][ch]) { u = fa[u]; l = len[u]; }
5      ++l; u = t[u][ch];
6      if (!u) u = 1;
7      // do something...
8  }

```

- 获取子串状态

```

1  int get_state(int l, int r) {
2      int u = rpos[r], s = r - l + 1;
3      FOR (i, SP - 1, -1) if (len[pa[u][i]] >= s) u = pa[u][i];
4      return u;
5  }

```

- 配合 LCT

```

1  namespace lct_sam {
2      extern struct P *const null;
3      const int M = N;
4      struct P {
5          P *fa, *ls, *rs;
6          int last;
7
8          bool has_fa() { return fa->ls == this || fa->rs == this; }
9          bool d() { return fa->ls == this; }

```

```

10     P*& c(bool x) { return x ? ls : rs; }
11     P* up() { return this; }
12     void down() {
13         if (ls != null) ls->last = last;
14         if (rs != null) rs->last = last;
15     }
16     void all_down() { if (has_fa()) fa->all_down(); down(); }
17 } *const null = new P{0, 0, 0, 0}, pool[M], *pit = pool;
18 P* G[N];
19 int t[M][26], len[M] = {-1}, fa[M], sz = 2, last = 1;
20
21 void rot(P* o) {
22     bool dd = o->d();
23     P *f = o->fa, *t = o->c(!dd);
24     if (f->has_fa()) f->fa->c(f->d()) = o; o->fa = f->fa;
25     if (t != null) t->fa = f; f->c(dd) = t;
26     o->c(!dd) = f->up(); f->fa = o;
27 }
28 void splay(P* o) {
29     o->all_down();
30     while (o->has_fa()) {
31         if (o->fa->has_fa())
32             rot(o->d() ^ o->fa->d() ? o : o->fa);
33         rot(o);
34     }
35     o->up();
36 }
37 void access(int last, P* u, P* v = null) {
38     if (u == null) { v->last = last; return; }
39     splay(u);
40     P *t = u;
41     while (t->ls != null) t = t->ls;
42     int L = len[fa[t - pool]] + 1, R = len[u - pool];
43
44     if (u->last) bit::add(u->last - R + 2, u->last - L + 2, 1);
45     else bit::add(1, 1, R - L + 1);
46     bit::add(last - R + 2, last - L + 2, -1);
47
48     u->rs = v;
49     access(last, u->up()->fa, u);
50 }
51 void insert(P* u, P* v, P* t) {
52     if (v != null) { splay(v); v->rs = null; }
53     splay(u);
54     u->fa = t; t->fa = v;
55 }
56
57 void ins(int ch, int pp) {
58     int p = last, np = last = sz++;
59     len[np] = len[p] + 1;
60     for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
61     if (!p) fa[np] = 1;
62     else {
63         int q = t[p][ch];
64         if (len[p] + 1 == len[q]) { fa[np] = q; G[np]->fa = G[q]; }
65         else {
66             int nq = sz++; len[nq] = len[p] + 1;
67             memcpy(t[nq], t[q], sizeof t[0]);
68             insert(G[q], G[fa[q]], G[nq]);
69             G[nq]->last = G[q]->last;
70             fa[nq] = fa[q];
71             fa[np] = fa[q] = nq;
72             G[np]->fa = G[nq];
73             for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
74         }
75     }
76     access(pp + 1, G[np]);
77 }
78
79 void init() {
80     ++pit;

```

```

81     FOR (i, 1, N) {
82         G[i] = pit++;
83         G[i]->ls = G[i]->rs = G[i]->fa = null;
84     }
85     G[1] = null;
86 }
87 }

```

回文自动机

```

1  namespace pam {
2      int t[N][26], fa[N], len[N], rs[N], cnt[N];
3      int sz, n, last;
4      int _new(int l) {
5          memset(t[sz], 0, sizeof t[0]);
6          len[sz] = l; cnt[sz] = 0;
7          return sz++;
8      }
9      void init() {
10         rs[n = sz = 0] = -1;
11         last = _new(0);
12         fa[last] = _new(-1);
13     }
14     int get_fa(int x) {
15         while (rs[n - 1 - len[x]] != rs[n]) x = fa[x];
16         return x;
17     }
18     void ins(int ch) {
19         rs[++n] = ch;
20         int p = get_fa(last);
21         if (!t[p][ch]) {
22             int np = _new(len[p] + 2);
23             fa[np] = t[get_fa(fa[p])][ch];
24             t[p][ch] = np;
25         }
26         ++cnt[last = t[p][ch]];
27     }
28 }

```

哈希

内置了自动双哈希开关（小心 TLE）。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  #define ENABLE_DOUBLE_HASH
5
6  typedef long long LL;
7  typedef unsigned long long ULL;
8
9  const int x = 135;
10 const int N = 4e5 + 10;
11 const int p1 = 1e9 + 7, p2 = 1e9 + 9;
12 ULL xp1[N], xp2[N], xp[N];
13
14 void init_xp() {
15     xp1[0] = xp2[0] = xp[0] = 1;
16     for (int i = 1; i < N; ++i) {
17         xp1[i] = xp1[i - 1] * x % p1;
18         xp2[i] = xp2[i - 1] * x % p2;
19         xp[i] = xp[i - 1] * x;
20     }
21 }
22
23 struct String {
24     char s[N];
25     int length, subsize;

```



```

26     bool sorted;
27     ULL h[N], hl[N];
28
29     ULL hash() {
30         length = strlen(s);
31         ULL res1 = 0, res2 = 0;
32         h[length] = 0; // ATTENTION!
33         for (int j = length - 1; j >= 0; --j) {
34             #ifdef ENABLE_DOUBLE_HASH
35                 res1 = (res1 * x + s[j]) % p1;
36                 res2 = (res2 * x + s[j]) % p2;
37                 h[j] = (res1 << 32) | res2;
38             #else
39                 res1 = res1 * x + s[j];
40                 h[j] = res1;
41             #endif
42             // printf("%llu\n", h[j]);
43         }
44         return h[0];
45     }
46
47     // 获取子串哈希, 左闭右开区间
48     ULL get_substring_hash(int left, int right) const {
49         int len = right - left;
50         #ifdef ENABLE_DOUBLE_HASH
51             // get hash of s[left...right-1]
52             unsigned int mask32 = ~(0u);
53             ULL left1 = h[left] >> 32, right1 = h[right] >> 32;
54             ULL left2 = h[left] & mask32, right2 = h[right] & mask32;
55             return (((left1 - right1 * xp1[len] % p1 + p1) % p1) << 32) |
56                 (((left2 - right2 * xp2[len] % p2 + p2) % p2));
57         #else
58             return h[left] - h[right] * xp[len];
59         #endif
60     }
61
62     void get_all_subs_hash(int sublen) {
63         subsize = length - sublen + 1;
64         for (int i = 0; i < subsize; ++i)
65             hl[i] = get_substring_hash(i, i + sublen);
66         sorted = 0;
67     }
68
69     void sort_substring_hash() {
70         sort(hl, hl + subsize);
71         sorted = 1;
72     }
73
74     bool match(ULL key) const {
75         if (!sorted) assert (0);
76         if (!subsize) return false;
77         return binary_search(hl, hl + subsize, key);
78     }
79
80     void init(const char *t) {
81         length = strlen(t);
82         strcpy(s, t);
83     }
84 };
85
86 int LCP(const String &a, const String &b, int ai, int bi) {
87     // Find LCP of a[ai...] and b[bi...]
88     int l = 0, r = min(a.length - ai, b.length - bi);
89     while (l < r) {
90         int mid = (l + r + 1) / 2;
91         if (a.get_substring_hash(ai, ai + mid) == b.get_substring_hash(bi, bi + mid))
92             l = mid;
93         else r = mid - 1;
94     }
95     return l;
96 }

```

```

97
98 int check(int ans) {
99     if (T.length < ans) return 1;
100     T.get_all_subs_hash(ans); T.sort_substring_hash();
101     for (int i = 0; i < S.length - ans + 1; ++i)
102         if (!T.match(S.get_substring_hash(i, i + ans)))
103             return 1;
104     return 0;
105 }
106
107 int main() {
108     init_xp(); // DON'T FORGET TO DO THIS!
109
110     for (int tt = 1; tt <= kases; ++tt) {
111         scanf("%d", &n); scanf("%s", str);
112         S.init(str);
113         S.hash(); T.hash();
114     }
115 }

```

后缀数组

构造时间: $O(L \log L)$; 查询时间 $O(\log L)$ 。suffix 数组是排好序的后缀下标, suffix 的反数组是后缀数组。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int N = 2e5 + 10;
5  const int Nlog = 18;
6
7  struct SuffixArray {
8      const int L;
9      vector<vector<int>> > P;
10     vector<pair<pair<int, int>, int>> > M;
11     int s[N], sa[N], rank[N], height[N];
12     // s: raw string
13     // sa[i]=k: s[k...L-1] ranks i (0 based)
14     // rank[i]=k: the rank of s[i...L-1] is k (0 based)
15     // height[i] = lcp(sa[i-1], sa[i])
16
17     SuffixArray(const string &raw_s) : L(raw_s.length()), P(1, vector<int>(L, 0)), M(L) {
18         for (int i = 0; i < L; i++)
19             P[0][i] = this->s[i] = int(raw_s[i]);
20         for (int skip = 1, level = 1; skip < L; skip *= 2, level++) {
21             P.push_back(vector<int>(L, 0));
22             for (int i = 0; i < L; i++)
23                 M[i] = make_pair(make_pair(P[level - 1][i], i + skip < L ? P[level - 1][i + skip] : -1000), i);
24             sort(M.begin(), M.end());
25             for (int i = 0; i < L; i++)
26                 P[level][M[i].second] = (i > 0 && M[i].first == M[i - 1].first) ? P[level][M[i - 1].second] : i;
27         }
28         for (unsigned i = 0; i < P.back().size(); ++i) {
29             rank[i] = P.back()[i];
30             sa[rank[i]] = i;
31         }
32     }
33
34     // This is a traditional way to calculate LCP
35     void getHeight() {
36         memset(height, 0, sizeof height);
37         int k = 0;
38         for (int i = 0; i < L; ++i) {
39             if (rank[i] == 0) continue;
40             if (k) k--;
41             int j = sa[rank[i] - 1];
42             while (i + k < L && j + k < L && s[i + k] == s[j + k]) ++k;
43             height[rank[i]] = k;
44         }
45         rmq_init(height, L);

```

```

46     }
47
48     int f[N][Nlog];
49     inline int highbit(int x) {
50         return 31 - __builtin_clz(x);
51     }
52
53     int rmq_query(int x, int y) {
54         int p = highbit(y - x + 1);
55         return min(f[x][p], f[y - (1 << p) + 1][p]);
56     }
57
58     // arr has to be 0 based
59     void rmq_init(int *arr, int length) {
60         for (int x = 0; x <= highbit(length); ++x)
61             for (int i = 0; i <= length - (1 << x); ++i) {
62                 if (!x) f[i][x] = arr[i];
63                 else f[i][x] = min(f[i][x - 1], f[i + (1 << (x - 1))][x - 1]);
64             }
65     }
66
67     #ifdef NEW
68     // returns the length of the longest common prefix of s[i...L-1] and s[j...L-1]
69     int LongestCommonPrefix(int i, int j) {
70         int len = 0;
71         if (i == j) return L - i;
72         for (int k = (int) P.size() - 1; k >= 0 && i < L && j < L; k--) {
73             if (P[k][i] == P[k][j]) {
74                 i += 1 << k;
75                 j += 1 << k;
76                 len += 1 << k;
77             }
78         }
79         return len;
80     }
81     #else
82     int LongestCommonPrefix(int i, int j) {
83         // getHeight() must be called first
84         if (i == j) return L - i;
85         if (i > j) swap(i, j);
86         return rmq_query(i + 1, j);
87     }
88     #endif
89
90     int checkNonOverlappingSubstring(int K) {
91         // check if there is two non-overlapping identical substring of length K
92         int minsa = 0, maxsa = 0;
93         for (int i = 0; i < L; ++i) {
94             if (height[i] < K) {
95                 minsa = sa[i]; maxsa = sa[i];
96             } else {
97                 minsa = min(minsa, sa[i]);
98                 maxsa = max(maxsa, sa[i]);
99                 if (maxsa - minsa >= K) return 1;
100             }
101         }
102         return 0;
103     }
104
105     int checkBelongToDifferentSubstring(int K, int split) {
106         int minsa = 0, maxsa = 0;
107         for (int i = 0; i < L; ++i) {
108             if (height[i] < K) {
109                 minsa = sa[i]; maxsa = sa[i];
110             } else {
111                 minsa = min(minsa, sa[i]);
112                 maxsa = max(maxsa, sa[i]);
113                 if (maxsa > split && minsa < split) return 1;
114             }
115         }
116         return 0;

```

```

117     }
118
119 } *S;
120
121 int main() {
122     string s, t;
123     cin >> s >> t;
124     int sp = s.length();
125     s += "*" + t;
126     S = new SuffixArray(s);
127     S->getHeight();
128     int left = 0, right = sp;
129     while (left < right) {
130         int mid = (left + right + 1) / 2;
131         if (S->checkBelongToDifferentSubstring(mid, sp))
132             left = mid;
133         else right = mid - 1;
134     }
135     printf("%d\n", left);
136 }

```

- SA-IS
- 仅在后缀自动机被卡内存或者卡常且需要 $O(1)$ LCA 的情况下使用（比赛中敲这个我觉得不行）

```

1 // rk [0..len-1] -> [1..len], sa/ht [1..len]
2 // s[i] > 0 && s[len] = 0
3 template<size_t size>
4 struct SuffixArray {
5     bool type[size << 1];
6     int bucket[size], bucket1[size];
7     int sa[size], rk[size], ht[size];
8     inline bool isLMS(const int i, const bool *type) { return i > 0 && type[i] && !type[i - 1]; }
9     template<class T>
10     inline void inducedSort(T s, int *sa, const int len, const int sigma, const int bucketSize, bool *type, int
11     ↪ *bucket, int *cntbuf, int *p) {
12         memset(bucket, 0, sizeof(int) * sigma);
13         memset(sa, -1, sizeof(int) * len);
14         for (int i = 0; i < len; i++) bucket[s[i]]++;
15         cntbuf[0] = bucket[0];
16         for (int i = 1; i < sigma; i++) cntbuf[i] = cntbuf[i - 1] + bucket[i];
17         for (int i = bucketSize - 1; i >= 0; i--) sa[--cntbuf[s[p[i]]]] = p[i];
18         for (int i = 1; i < sigma; i++) cntbuf[i] = cntbuf[i - 1] + bucket[i - 1];
19         for (int i = 0; i < len; i++) if (sa[i] > 0 && !type[sa[i] - 1]) sa[cntbuf[s[sa[i] - 1]]++] = sa[i] - 1;
20         cntbuf[0] = bucket[0];
21         for (int i = 1; i < sigma; i++) cntbuf[i] = cntbuf[i - 1] + bucket[i];
22         for (int i = len - 1; i >= 0; i--) if (sa[i] > 0 && type[sa[i] - 1]) sa[--cntbuf[s[sa[i] - 1]]] = sa[i] - 1;
23     }
24     template<typename T>
25     inline void sais(T s, int *sa, int len, bool *type, int *bucket, int *bucket1, int sigma) {
26         int i, j, bucketSize = 0, cnt = 0, p = -1, x, *cntbuf = bucket + sigma;
27         type[len - 1] = 1;
28         for (i = len - 2; i >= 0; i--) type[i] = s[i] < s[i + 1] || (s[i] == s[i + 1] && type[i + 1]);
29         for (i = 1; i < len; i++) if (type[i] && !type[i - 1]) bucket1[bucketSize++] = i;
30         inducedSort(s, sa, len, sigma, bucketSize, type, bucket, cntbuf, bucket1);
31         for (i = bucketSize = 0; i < len; i++) if (isLMS(sa[i], type)) sa[bucketSize++] = sa[i];
32         for (i = bucketSize; i < len; i++) sa[i] = -1;
33         for (i = 0; i < bucketSize; i++) {
34             x = sa[i];
35             for (j = 0; j < len; j++) {
36                 if (p == -1 || s[x + j] != s[p + j] || type[x + j] != type[p + j]) { cnt++, p = x; break; }
37                 else if (j > 0 && (isLMS(x + j, type) || isLMS(p + j, type))) break;
38             }
39             x = (~x & 1 ? x >> 1 : x - 1 >> 1), sa[bucketSize + x] = cnt - 1;
40         }
41         for (i = j = len - 1; i >= bucketSize; i--) if (sa[i] >= 0) sa[j--] = sa[i];
42         int *s1 = sa + len - bucketSize, *bucket2 = bucket1 + bucketSize;
43         if (cnt < bucketSize) sais(s1, sa, bucketSize, type + len, bucket, bucket1 + bucketSize, cnt);
44         else for (i = 0; i < bucketSize; i++) sa[s1[i]] = i;
45         for (i = 0; i < bucketSize; i++) bucket2[i] = bucket1[sa[i]];
46         inducedSort(s, sa, len, sigma, bucketSize, type, bucket, cntbuf, bucket2);
47     }
48 }

```

```

47     template<typename T>
48     inline void getHeight(T s, const int len, const int *sa) {
49         for (int i = 0, k = 0; i < len; i++) {
50             if (rk[i] == 0) k = 0;
51             else {
52                 if (k > 0) k--;
53                 int j = sa[rk[i] - 1];
54                 while (i + k < len && j + k < len && s[i + k] == s[j + k]) k++;
55             }
56             ht[rk[i]] = k;
57         }
58     }
59     template<class T>
60     inline void init(T s, int len, int sigma) {
61         sais(s, sa, ++len, type, bucket, bucket1, sigma);
62         for (int i = 1; i < len; i++) rk[sa[i]] = i;
63         getHeight(s, len, sa);
64     }
65 };

```

KMP 自动机

```

1  int m; int pat[N];
2  namespace kmp {
3      int f[N]; // f[i] 表示已匹配成功 i 个, 失配要去哪里
4
5      template<typename T>
6      int go(int stat, T c, bool& acc) {
7          // stat 是当前态 (表示已经匹配了 stat 个字符), c 是要走的边
8          while (stat && c != pat[stat]) stat = f[stat];
9          if (c == pat[stat]) stat++;
10         if (stat == m) acc = true;
11         return stat;
12     }
13
14     void getFail() {
15         static int f2[N];
16         f[0] = f[1] = 0;
17         f2[0] = f2[1] = 0;
18         FOR (i, 1, m) {
19             int j = f2[i];
20             while (j && pat[i] != pat[j]) j = f2[j];
21             f2[i+1] = f[i+1] = (pat[i] == pat[j]) ? j+1 : 0;
22             if (f[i+1] == j+1 && pat[i+1] == pat[j+1]) f[i+1] = f[j+1];
23         }
24         FOR (i, 0, m) dbg(i, f[i]);
25     }
26 }

```

Trie

```

1  namespace trie {
2      int t[N][26], sz, ed[N];
3      void init() { sz = 2; memset(ed, 0, sizeof ed); }
4      int _new() { memset(t[sz], 0, sizeof t[sz]); return sz++; }
5      void ins(char* s, int p) {
6          int u = 1;
7          FOR (i, 0, strlen(s)) {
8              int c = s[i] - 'a';
9              if (!t[u][c]) t[u][c] = _new();
10             u = t[u][c];
11         }
12         ed[u] = p;
13     }
14 }

```

杂项

STL

- copy

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

- merge (如果相等, 第一个优先)

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

- for_each

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

- transform

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

- numeric_limits

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

- iota

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

伪随机数

```
1 unsigned rnd() {
2     static unsigned A = 1 << 16 | 3, B = 33333331, C = 2341;
3     return C = A * C + B;
4 }
```

日期

```
1 // Routines for performing computations on dates. In these routines,
2 // months are expressed as integers from 1 to 12, days are expressed
3 // as integers from 1 to 31, and years are expressed as 4-digit
4 // integers.
5
6 string dayOfWeek[] = {"Mo", "Tu", "We", "Th", "Fr", "Sa", "Su"};
7
8 // converts Gregorian date to integer (Julian day number)
9
10 int DateToInt (int m, int d, int y){
11     return
12         1461 * (y + 4800 + (m - 14) / 12) / 4 +
13         367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
14         3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
15         d - 32075;
16 }
17
18 // converts integer (Julian day number) to Gregorian date: month/day/year
19
20 void IntToDate (int jd, int &m, int &d, int &y){
21     int x, n, i, j;
22 }
```

```

23     x = jd + 68569;
24     n = 4 * x / 146097;
25     x -= (146097 * n + 3) / 4;
26     i = (4000 * (x + 1)) / 1461001;
27     x -= 1461 * i / 4 - 31;
28     j = 80 * x / 2447;
29     d = x - 2447 * j / 80;
30     x = j / 11;
31     m = j + 2 - 12 * x;
32     y = 100 * (n - 49) + i + x;
33 }
34
35 // converts integer (Julian day number) to day of week
36
37 string IntToDay (int jd){
38     return dayOfWeek[jd % 7];
39 }

```

子集枚举

- 枚举真子集

```

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

```

- 枚举大小为 k 的子集

```

1 template<typename T>
2 void subset(int k, int n, T&& f) {
3     int t = (1 << k) - 1;
4     while (t < 1 << n) {
5         f(t);
6         int x = t & -t, y = t + x;
7         t = ((t & ~y) / x >> 1) | y;
8     }
9 }

```

权值最大上升子序列

```

1 const LL maxn = 1E5 + 10;
2 const LL INF = 1E10;
3 struct P {
4     LL k, v;
5     bool operator < (const P& rhs) const {
6         return k < rhs.k || (k == rhs.k && v < rhs.v);
7     }
8 };
9 LL k[maxn], v[maxn], n, T;
10 set<P> s;
11
12 int main() {
13     cin >> T;
14     while (T--) {
15         s.clear();
16         s.insert({-INF, 0});
17         cin >> n;
18         FOR (i, 0, n) scanf("%lld", &k[i]);
19         FOR (i, 0, n) scanf("%lld", &v[i]);
20         FOR (i, 0, n) {
21             auto it = s.lower_bound({k[i], INF});
22             LL vv = (--it)->v + v[i];
23             ++it;
24             while (it != s.end() && it->v <= vv)
25                 it = s.erase(it);
26             if (it == s.end() || it->k != k[i]) s.insert({k[i], vv});
27         }
28         cout << s.rbegin()->v << endl;
29     }
30 }

```

数位 DP

```
1 LL dfs(LL base, LL pos, LL len, LL s, bool limit) {
2     if (pos == -1) return s ? base : 1;
3     if (!limit && dp[base][pos][len][s] != -1) return dp[base][pos][len][s];
4     LL ret = 0;
5     LL ed = limit ? a[pos] : base - 1;
6     FOR (i, 0, ed + 1) {
7         tmp[pos] = i;
8         if (len == pos)
9             ret += dfs(base, pos - 1, len - (i == 0), s, limit && i == a[pos]);
10        else if (s && pos < (len + 1) / 2)
11            ret += dfs(base, pos - 1, len, tmp[len - pos] == i, limit && i == a[pos]);
12        else
13            ret += dfs(base, pos - 1, len, s, limit && i == a[pos]);
14    }
15    if (!limit) dp[base][pos][len][s] = ret;
16    return ret;
17 }
18
19 LL solve(LL x, LL base) {
20     LL sz = 0;
21     while (x) {
22         a[sz++] = x % base;
23         x /= base;
24     }
25     return dfs(base, sz - 1, sz - 1, 1, true);
26 }
```

土制 bitset

```
1 // M 要开大至少 1 个 64
2 const int M = (1E4 + 200) / 64;
3 typedef unsigned long long ULL;
4 const ULL ONE = 1;
5
6 struct Bitset {
7     ULL a[M];
8     void go(int x) {
9         int offset = x / 64; x %= 64;
10        for (int i = offset, j = 0; i + 1 < M; ++i, ++j) {
11            a[j] |= a[i] >> x;
12            if (x) a[j] |= a[i + 1] << (64 - x); // 不能左移 64 位
13        }
14    }
15    void init() { memset(a, 0, sizeof a); }
16    void set(int x) {
17        int offset = x / 64; x %= 64;
18        a[offset] |= (ONE << x);
19    }
20    void prt() {
21        FOR (i, 0, M) FOR (j, 0, 64) putchar((a[i] & (ONE << j)) ? '1' : '0');
22        puts("");
23    }
24    int lowbit() {
25        FOR (i, 0, M) if (a[i]) return i * 64 + __builtin_ctzll(a[i]);
26        assert(0);
27    }
28    int highbit(int x) {
29        // [0, x) 的最高位
30        int offset = x / 64; x %= 64;
31        FOR (i, offset, -1) {
32            if (!a[i]) continue;
33            if (i == offset) {
34                FOR (j, x - 1, -1) if ((ONE << j) & a[i]) { return i * 64 + j; }
35            } else return i * 64 + 63 - __builtin_clzll(a[i]);
36        }
37        assert(0);
38    }
39 }
```



```

38     }
39 };

```

扩栈（本地使用）

```

1  #include <sys/resource.h>
2  void init_stack(){
3      const rlim_t kStackSize = 512 * 1024 * 1024;
4      struct rlimit rl;
5      int result;
6      result = getrlimit(RLIMIT_STACK, &rl);
7      if (result == 0) {
8          if (rl.rlim_cur < kStackSize) {
9              rl.rlim_cur = kStackSize;
10             result = setrlimit(RLIMIT_STACK, &rl);
11             if (result != 0) {
12                 fprintf(stderr, "setrlimit returned result = %d\n", result);
13             }
14         }
15     }
16 }

```

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/dfs 序，初始化或者询问不要忘记 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
- 不要把 int 开成 char
- 模意义下不要用除法
- 哈希不要自然溢出
- 最短路不要 SPFA，乖乖写 Dijkstra
- 上取整以及 GCD 小心负数
- mid 用 $l + (r - l) / 2$ 可以避免溢出和负数的问题