目录

1 STL

1.1 unordered_map 对 pii 进行哈希

```cpp
// call: unordered_map<pii, int, pair_hash> ma;
// 哈希方法1（较快）
template<typename T>
inline void hash_combine(std::size_t &seed, const T &val) {
    seed ^= std::hash<T>()(val) + 0x9e3779b9 + (seed << 6) + (seed >> 2);
}

template<typename T>
inline void hash_val(std::size_t &seed, const T &val) {
    hash_combine(seed, val);
}

template<typename T, typename... Types>
inline void hash_val(std::size_t &seed, const T &val, const Types &... args) {
    hash_combine(seed, val);
    hash_val(seed, args...);
}

template<typename... Types>
inline std::size_t hash_val(const Types &... args) {
    std::size_t seed = 0;
    hash_val(seed, args...);
    return seed;
}

struct pair_hash {
    template<class T1, class T2>
    std::size_t operator()(const std::pair<T1, T2> &p) const {
        return hash_val(p.first, p.second);
    }
};

// 哈希方法2（部分冲不过）
struct pair_hash {
    template<class T1, class T2>
    std::size_t operator()(const std::pair<T1, T2> &p) const {
        auto h1 = std::hash<T1>{}(p.first);
        auto h2 = std::hash<T2>{}(p.second);
        return h1 ^ h2;
    }
```

```
};
```

1.2 pb_ds 实现平衡树

```
#define _EXT_CODECVT_SPECIALIZATIONS_H 1
#define _EXT_ENC_FILEBUF_H 1
#undef __MINGW32__

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

class PBDS {
private:
    typedef tree<ll, null_type, less<ll>, rb_tree_tag,
       tree_order_statistics_node_update> Tree;
    Tree T;
public:
    void insert(ll x, int id) { // 插入x数，需传入第几次操作进行简单处理
        T.insert((x << 20) + id);    // 简单的处理
    }

    void remove(ll x) { // 删除x数(若有多个相同的数，因只删除一个)
        T.erase(T.lower_bound(x << 20));
    }

    ll get_rank(ll x) { // 查询x数的排名(排名定义为比当前数小的数的个数+1)
        return T.order_of_key(x << 20) + 1;
    }

    ll get_val(ll x) {  // 查询排名为x的数
        return *T.find_by_order(x - 1) >> 20;
    }

    ll get_pre(ll x) {  // 求x的前驱(前驱定义为小于x，且最大的数)
        return *--T.lower_bound(x << 20) >> 20;
    }

    ll get_next(ll x) { // 求x的后继(后继定义为大于x，且最小的数)
        return *T.lower_bound((x + 1) << 20) >> 20;
    }
```

```
} rbt;  // 注意不能叫tree，否则会冲突
```

1.3 rope

写一种数据结构，支持任意位置插入、删除和修改

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

using namespace std;
using namespace __gnu_cxx;
rope<char> tree;
inline void read_str(char *s, int len) {    // 读入长度为len的字符串
    s[len] = '\0';
    len--;
    for (int i = 0; i <= len; i++) {
        s[i] = '\0';
        while (s[i] < 32 || 126 < s[i]) s[i] = getchar();
    }
}

inline void read_int(int &x) {
    x = 0;
    char ch;
    while (!isdigit(ch = getchar()));
    x = ch - '0';
    while (isdigit(ch = getchar())) x = x * 10 + ch - '0';
}
const int MAXN = 2e6 + 5;
char word[MAXN];

int main() {
    int T;
    int now = 0;
    scanf("%d", &T);
    while (T--) {
        int opt = '1', x;
        while (!isalpha(opt = getchar()));
        while (isalpha(getchar()));
        // 格式:Move k
        // 将光标移动到第k个字符之后，如果k=0，将光标移到文本开头
        if (opt == 'M') read_int(now);
        // Insert n s
        // 在光标处插入长度为n的字符串s，光标位置不变n>=1
```

```
        else if (opt == 'I') {
            read_int(x);
            read_str(word, x);
            tree.insert(now, word);
        }
        // Delete n
        // 删除光标后的n个字符，光标位置不变，n>=1
        else if (opt == 'D') {
            read_int(x);
            tree.erase(now, x);
        }
        // Get n
        // 输出光标后的n个字符，光标位置不变，n>=1
        else if (opt == 'G') {
            read_int(x);
            x--;
            for (int i = now; i <= now + x; i++) printf("%c", tree[i]);
            printf("\n");
        } else if (opt == 'P') now--;   // 光标前移
        else now++; //光标后移
    }
}
```

1.4 ST 表

```
const int MAXN = 5e4 + 5;   // n
const int MAXL = 22;
int f[MAXN][MAXL], g[MAXN][MAXL], two[MAXN];
int h[MAXN];
void pre(int n) {
    for (int i = 1; i <= n; i++) {
        f[i][0] = h[i]; g[i][0] = h[i]; // 读入h
    }
    two[1] = 0; two[2] = 1;
    for (int i = 3; i < MAXN; i++) {
        two[i] = two[i / 2] + 1;
    }
    for (int j = 1; j <= MAXL; j++) {
        for (int i = 1; i + (1 << j) - 1 <= n; i++)
            f[i][j] = max(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
        }
    for (int j = 1; j <= MAXL; j++) {
        for (int i = 1; i + (1 << j) - 1 <= n; i++)
```

```
            g[i][j] = min(g[i][j - 1], g[i + (1 << (j - 1))][j - 1]);
    }
}

int get_max(int x, int y) {
    int s = two[y - x + 1];
    return max(f[x][s], f[y - (1 << s) + 1][s]);
}

int get_min(int x, int y) {
    int s = two[y - x + 1];
    return min(g[x][s], g[y - (1 << s) + 1][s]);
}
```

1.5 笛卡尔树

1.5.1 建树

```
// p[i]存放[1, n]数据
class cartesian {
    int stk[MAXN], lson[MAXN], rson[MAXN];
    void build() {
        int tmp = 0, x;
        for (int i = 1; i <= n; i++) {
            x = tmp;
            while (x && p[stk[x]] > p[i]) x--;
            if (x) rson[stk[x]] = i;
            if (x < tmp) lson[i] = stk[x + 1];
            stk[++x] = i;
            tmp = x;
        }
    }
} ;
```

1.6 线段树

1.6.1 区间中所有元素都严格出现三次的区间个数（**CF1418G**）

```
/*
    input               output
    9                   3
    1 2 2 2 1 1 2 2 2
    10                  0
```

```
    1 2 3 4 1 2 3 1 2 3
    12                  1
    1 2 3 4 3 4 2 1 3 4 2 1
*/
int arr[MAXN];
vector<int> vec[MAXN];
struct node {
    int l, r, v;
    node(int _l = 0, int _r = 0, int _v = 0) : l(_l), r(_r), v(_v) {}
};

class SEG { public:
    struct node {
        int l, r, mx, cnt;
    } T[MAXN << 2];
    int lazy[MAXN << 2];

    void build(int rt, int l, int r) {
        T[rt].l = l, T[rt].r = r, T[rt].mx = 0, T[rt].cnt = r - l;
        if (l + 1 == r) {
            T[rt << 1].l = T[rt << 1].r = T[rt << 1].mx = T[rt << 1].cnt = 0;
            T[rt << 1 | 1].l = T[rt << 1 | 1].r = T[rt << 1 | 1].mx = T[rt <<
               1 | 1].cnt = 0;
            return;
        }
        int mid = (l + r) >> 1;
        build(rt << 1, l, mid);
        build(rt << 1 | 1, mid, r);
    }

    inline void push_up(int rt) {
        if (T[rt << 1].mx > T[rt << 1 | 1].mx) T[rt].mx = T[rt << 1].mx,
           T[rt].cnt = T[rt << 1].cnt;
        else if (T[rt << 1].mx < T[rt << 1 | 1].mx) T[rt].mx = T[rt << 1 |
           1].mx, T[rt].cnt = T[rt << 1 | 1].cnt;
        else T[rt].mx = T[rt << 1].mx, T[rt].cnt = T[rt << 1].cnt + T[rt << 1
           | 1].cnt;
    }

    inline void push_down(int rt) {
        if (lazy[rt]) {
            T[rt << 1].mx += lazy[rt], lazy[rt << 1] += lazy[rt];
            T[rt << 1 | 1].mx += lazy[rt], lazy[rt << 1 | 1] += lazy[rt];
            lazy[rt] = 0;
```

```
        }
    }

    void update(int rt, int L, int R, int val) {
        if (L <= T[rt].l && R >= T[rt].r) {
            T[rt].mx += val, lazy[rt] += val;
            return;
        }
        if (L >= T[rt].r || R <= T[rt].l)  return ;
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (L <= mid) update(rt << 1, L, R, val);
        if (R >= mid) update(rt << 1 | 1, L, R, val);
        push_up(rt);
    }
} tree;

vector<node> event[MAXN];
void add(int l1, int l2, int r1, int r2) {
    event[l1].push_back(node(r1, r2, 1));
    event[l2].push_back(node(r1, r2, -1));
}

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &arr[i]);
        vec[arr[i]].push_back(i);
    }
    for (int x = 1; x <= n; x++) {
        int sz = SZ(vec[x]);
        for (int i = 0; i <= sz; i++) {
            add(i == 0 ? 1 : vec[x][i - 1] + 1, i == sz ? n + 2 : vec[x][i] +
                1,
                i == 0 ? 1 : vec[x][i - 1] + 1, i == sz ? n + 2 : vec[x][i] +
                    1);
        }
        for (int i = 0; i <= sz - 3; i++) {
            add(i == 0 ? 1 : vec[x][i - 1] + 1, vec[x][i] + 1,
                vec[x][i + 2] + 1, i + 3 == sz ? n + 2 : vec[x][i + 3] + 1);
        }
    }
    tree.build(1, 1, n + 2);
```

```
    ll res = 0;
    for (int l = 1; l <= n; l++) {  // enum left
        for (auto e: event[l]) {
            int el = e.l, er = e.r, ev = e.v;
            tree.update(1, el, er, ev);
        }
        tree.update(1, l, l+1, -1);
        if (tree.T[1].mx == n) {
            res += tree.T[1].cnt;
        }
    }
    printf("%lld\n", res);
}
```

1.6.2 线段树分裂合并

```
class SEG { public:
    int ch[MAXM][2];
    ll sum[MAXM];
#define lson ch[rt][0]
#define rson ch[rt][1]
    int pool[MAXM], pool_cnt, cnt;
    int New() { // recycle
        return pool_cnt ? pool[pool_cnt--] : ++cnt;
    }
    void Del(int rt) {
        pool[++pool_cnt] = rt;
        ch[rt][0] = ch[rt][1] = sum[rt] = 0;
    }
    inline void push_up(int rt) {
        sum[rt] = sum[lson] + sum[rson];
    }
    int change(int rt, int pos, int v, int be, int en) {
        if (!rt) rt = New();
        if (be == en) {
            sum[rt] += (ll) v;
            return rt;
        }
        int mid = (be + en) >> 1;
        if (pos <= mid) lson = change(lson, pos, v, be, mid);
        else
            rson = change(rson, pos, v, mid + 1, en);
        push_up(rt);
```

```
        return rt;
    }
    ll query_sum(int rt, int L, int R, int be, int en) {
        if (!rt) return 0;
        if (L <= be && R >= en) return sum[rt];
        int mid = (be + en) >> 1;
        ll ans = 0;
        if (L <= mid) ans += query_sum(lson, L, R, be, mid);
        if (R > mid) ans += query_sum(rson, L, R, mid + 1, en);
        return ans;
    }
    int Kth(int rt, int be, int en, int k) {
        if (be == en) return be;
        int mid = (be + en) >> 1;
        int ans = -1;
        if (sum[ch[rt][0]] >= k) ans = Kth(lson, be, mid, k);
        else ans = Kth(rson, mid + 1, en, k - sum[ch[rt][0]]);
        return ans;
    }
    int merge(int x, int y, int be, int en) {
        if (!x || !y) return x + y;
        if (be == en) {
            sum[x] += sum[y];
            return x;
        }
        int mid = (be + en) >> 1;
        ch[x][0] = merge(ch[x][0], ch[y][0], be, mid);
        ch[x][1] = merge(ch[x][1], ch[y][1], mid + 1, en);
        push_up(x);
        Del(y);
        return x;
    }
    void split(int &rt1, int &rt2, int L, int R, int be, int en) {
        if (en < L || R < be) return;
        if (!rt1) return;
        if (L <= be && en <= R) {
            rt2 = rt1, rt1 = 0;
            return;
        }
        if (!rt2) rt2 = New();
        int mid = (be + en) >> 1;
        split(ch[rt1][0], ch[rt2][0], L, R, be, mid);
        split(ch[rt1][1], ch[rt2][1], L, R, mid + 1, en);
        push_up(rt1), push_up(rt2);
```

```
    }
} tree;

int root[MAXN];
int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    int max_n_m = max(n, m);
    for (int i = 1; i <= n; i++) {
        int x;
        scanf("%d", &x);
        root[1] = tree.change(root[1], i, x, 1, max_n_m);
    }
    int id = 1;
    while (m--) {
        int opt;
        scanf("%d", &opt);
        if (opt == 0) { // 将可重集p中>=x且<=y的值放入一个新的可重集中
            int p, x, y;
            scanf("%d%d%d", &p, &x, &y);
            tree.split(root[p], root[++id], x, y, 1, max_n_m);
        } else if (opt == 1) {  // 将可重集t中的数放入可重集p，且清空可重集t
            int p, t;
            scanf("%d%d", &p, &t);  // 数据保证在此后的操作中不会出现可重集t
            root[p] = tree.merge(root[p], root[t], 1, max_n_m);
        } else if (opt == 2) {  // 在p这个可重集中加入x个数字q
            int p, x, q;
            scanf("%d%d%d", &p, &x, &q);
            root[p] = tree.change(root[p], q, x, 1, max_n_m);
        } else if (opt == 3) {  // 查询可重集p中大>=x且<=y的值的个数
            int p, x, y;
            scanf("%d%d%d", &p, &x, &y);
            printf("%lld\n", tree.query_sum(root[p], x, y, 1, max_n_m));
        } else {  // 查询在p这个可重集中第k小的数，不存在时输出-1
            int p, k;
            scanf("%d%d", &p, &k);
            if (tree.sum[root[p]] < k) {
                printf("-1\n");
            } else printf("%d\n", tree.Kth(root[p], 1, max_n_m, k));
        }
    }
}
```

1.6.3 单点修改 + 单点最大连通数（HDU1540）

```
class SEG { public:
    struct Node {
        int l, r;
        int pre, suf, len;
    } T[MAXN << 2];

    void Build(int rt, int l, int r) {
        T[rt].l = l, T[rt].r = r;
        T[rt].pre = T[rt].suf = T[rt].len = r - l + 1;
        if (l == r) return;
        int mid = (l + r) >> 1;
        Build(rt << 1, l, mid);
        Build(rt << 1 | 1, mid + 1, r);
    }

    inline void push_up(int rt) {
        T[rt].pre = T[rt << 1].pre;
        T[rt].suf = T[rt << 1 | 1].suf;
        T[rt].len = max(T[rt << 1].len, T[rt << 1 | 1].len);
        T[rt].len = max(T[rt].len, T[rt << 1].suf + T[rt << 1 | 1].pre);
        if (T[rt << 1].pre == T[rt << 1].r - T[rt << 1].l + 1)T[rt].pre +=
           T[rt << 1 | 1].pre;
        if (T[rt << 1 | 1].suf == T[rt << 1 | 1].r - T[rt << 1 | 1].l + 1)
            T[rt].suf += T[rt << 1].suf;
    }

    // call: tree.update(1, x, 0); 摧毁
    //       tree.update(1, x, 1); 修复

    void update(int rt, int pos, int val) {
        if (T[rt].l == T[rt].r) {
            T[rt].pre = T[rt].suf = T[rt].len = val;
            return;
        }
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (pos <= mid)update(rt << 1, pos, val);
        else update(rt << 1 | 1, pos, val);
        push_up(rt);
    }

    int query(int rt, int pos) {
```

```
        if (T[rt].l == T[rt].r || T[rt].len == 0 || T[rt].len == T[rt].r -
           T[rt].l + 1) {
            return T[rt].len;
        }
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (pos <= mid) {
            if (pos >= T[rt << 1].r - T[rt << 1].suf + 1)
                return query(rt << 1, pos) + query(rt << 1 | 1, mid + 1);
            else return query(rt << 1, pos);
        } else {
            if (pos <= T[rt << 1 | 1].l + T[rt << 1 | 1].pre - 1)
                return query(rt << 1 | 1, pos) + query(rt << 1, mid);
            else return query(rt << 1 | 1, pos);
        }
    }
} tree;
```

1.7 可持久化线段树（主席树）

1.7.1 静态区间第 K 小

```
/*
  input                           output
  5 5
  25957 6405 15770 26287 26465
  2 2 1                           6405
  3 4 1                           15770
  4 5 1                           26287
  1 2 2                           25957
  4 4 1                           26287
*/
class HJT { public:
    int ch[MAXN * 70][2], sum[MAXN * 70];
    int tot = 0;
    inline void push_up(int rt) {
        sum[rt] = sum[ch[rt][0]] + sum[ch[rt][1]];
    }
    int update(int rt, int pos, int val, int be, int en) {
        int nrt = ++tot;
        ch[nrt][0] = ch[nrt][1] = sum[nrt] = 0;
        if (be == en) {
            sum[nrt] = sum[rt] + val;
            return nrt;
        }
```

```
        int mid = (be + en) >> 1;
        if (pos <= mid) {
            ch[nrt][0] = update(ch[rt][0], pos, val, be, mid);
            ch[nrt][1] = ch[rt][1];
        } else {
            ch[nrt][0] = ch[rt][0];
            ch[nrt][1] = update(ch[rt][1], pos, val, mid + 1, en);
        }
        push_up(nrt);
        return nrt;
    }
    int query(int lrt, int rrt, int k, int be, int en) {
        if (be >= en) return be;
        int delta = sum[ch[rrt][0]] - sum[ch[lrt][0]];
        int mid = (be + en) >> 1;
        if (delta >= k) return query(ch[lrt][0], ch[rrt][0], k, be, mid);
        else return query(ch[lrt][1], ch[rrt][1], k - delta, mid + 1, en);
    }
} tree;

int ai[MAXN], root[MAXN];
int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &ai[i]);
        Discrete::insert(ai[i]);
    }
    Discrete::init();
    root[0] = 0;
    for (int i = 1; i <= n; i++) {
        root[i] = tree.update(root[i-1], Discrete::val2id(ai[i]), 1, 1,
           Discrete::blen);
    }
    while(m--) {
        int l, r, k;
        scanf("%d%d%d", &l, &r, &k);
        printf("%d\n", Discrete::id2val(tree.query(root[l-1], root[r], k, 1,
           Discrete::blen)));
    }
}
```

1.7.2 区间内不同数个数

```
class HJT { public:
    int query(int rt, int L, int R, int be, int en) {
        if (L <= be && en <= R) return sum[rt];
        int mid = (be + en) >> 1;
        int ans = 0;
        if (L <= mid) ans += query(ch[rt][0], L, R, be, mid);
        if (R > mid) ans += query(ch[rt][1], L, R, mid + 1, en);
        return ans;
    }
} tree;
int val[MAXN], root[MAXN];
unordered_map<int, int> pre;

int main() {
    int n; scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d", &val[i]);
    root[0] = 0;
    for (int i = 1; i <= n; i++) {
        if (pre[val[i]]) {
            int tmp = tree.change(root[i - 1], pre[val[i]], -1, 1, n);
            root[i] = tree.change(tmp, i, 1, 1, n);
        } else {
            root[i] = tree.change(root[i - 1], i, 1, 1, n);
        }
        pre[val[i]] = i;
    }
    int q; scanf("%d", &q);
    while (q--) {
        int l, r;
        scanf("%d%d", &l, &r);
        printf("%d\n", tree.query(root[r], l, r, 1, n));
    }
}
```

1.7.3 静态区间第 K 小

```
class HJT { public:
    int query(int lrt, int rrt, int fa, int faa, int k, int be, int en) {
        if (be >= en) return be;
        int mid = (be + en) >> 1;
        int delta = sum[ch[lrt][0]] + sum[ch[rrt][0]] - sum[ch[fa][0]] -
            sum[ch[faa][0]];
```

```
        if (delta >= k) return query(ch[lrt][0], ch[rrt][0], ch[fa][0],
            ch[faa][0], k, be, mid);
        else return query(ch[lrt][1], ch[rrt][1], ch[fa][1], ch[faa][1], k -
            delta, mid + 1, en);
    }
} tree;

struct Edge {
    int to, nex;
} e[MAXN << 1];
int head[MAXN], tol;
void addEdge(int u, int v) {
    e[tol].to = v, e[tol].nex = head[u], head[u] = tol, tol++;
}
int val[MAXN], root[MAXN], new_val[MAXN];
int dep[MAXN], fa[MAXN][32], lg[MAXN];
void init(int _n) {
    for (int i = 1; i <= _n; i++) lg[i] = lg[i - 1] + (1 << lg[i - 1] == i);
}
void dfs(int u, int f) {
    fa[u][0] = f;
    dep[u] = dep[f] + 1;
    root[u] = tree.update(root[f], new_val[u], 1, 1, Discrete::blen);
    for (int i = 1; i <= lg[dep[u]]; i++) fa[u][i] = fa[fa[u][i - 1]][i - 1];
    for (int i = head[u]; ~i; i = e[i].nex) {
        int v = e[i].to;
        if (v == f) continue;
        dfs(v, u);
    }
}
int LCA(int u, int v) {
    if (dep[u] < dep[v]) swap(u, v);
    while (dep[u] > dep[v]) u = fa[u][lg[dep[u] - dep[v]] - 1];
    if (u == v) return u;
    for (int k = lg[dep[u]] - 1; k >= 0; k--) {
        if (fa[u][k] != fa[v][k]) u = fa[u][k], v = fa[v][k];
    }
    return fa[u][0];
}
int main() {
    int n, m; scanf("%d%d", &n, &m);
    init(n);
    for (int i = 1; i <= n; i++) head[i] = -1;
    for (int i = 1; i <= n; i++) scanf("%d", &val[i]);
```

```
48
49      for (int i = 1; i <= n; i++) Discrete::insert(val[i]);
50      Discrete::init();
51      for (int i = 1; i <= n; i++) new_val[i] = Discrete::val2id(val[i]);
52      for (int i = 2; i <= n; i++) {
53          int u, v; scanf("%d%d", &u, &v);
54          addEdge(u, v), addEdge(v, u);
55      }
56      root[0] = 0;
57      dfs(1, 0);
58      while (m--) {
59          int u, v, k;
60          scanf("%d%d%d", &u, &v, &k);
61          int lca = LCA(u, v);
62          printf("%d\n",
63                 Discrete::id2val(tree.query(root[u], root[v], root[lca],
                      root[fa[lca][0]], k, 1, Discrete::blen)));
64      }
65  }
```

1.7.4 区间 MEX

```
1   int blen, bep;
2   class HJT {
3   public:
4       struct node {
5           int lson, rson, maxx;
6       } T[MAXN * 70];
7       int tol;
8   #define ls T[rt].lson
9   #define rs T[rt].rson
10      inline void push_up(int rt) {
11          T[rt].maxx = max(T[ls].maxx, T[rs].maxx);
12      }
13      int build(int l, int r) {
14          int nrt = ++tol;
15          int mid = (l + r) >> 1;
16          T[nrt].lson = T[nrt].rson = 0; T[nrt].maxx = bep;
17          if (l < r) {
18              T[nrt].lson = build(l, mid);
19              T[nrt].rson = build(mid + 1, r);
20              push_up(nrt);
21          }
```

```
        return nrt;
    }
    int query(int rt, int R, int be, int en) {
        if (be == en) return be;
        int ans = -1;
        int mid = (be + en) >> 1;
        if (T[ls].maxx > R) ans = query(T[rt].lson, R, be, mid);
        if (ans == -1 && T[rs].maxx > R) ans = query(T[rt].rson, R, mid + 1,
           en);
        return ans;
    }
} tree;

int arr[MAXN], root[MAXN];
int main() {
    int n, m; scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", &arr[i]);
    Discrete::push(0);
    for (int i = 1; i <= n; i++) Discrete::push(arr[i]),
       Discrete::push(arr[i] + 1);
    Discrete::init();
    bep = n + 1;
    root[n + 1] = tree.build(1, blen);
    for (int i = n; i >= 1; i--) {
        int x = Discrete::lb(arr[i]);
        root[i] = tree.update(root[i + 1], 1, blen, x, i);
    }
    while (m--) {
        int l, r;
        scanf("%d%d", &l, &r);
        int res = tree.query(root[l], r, 1, blen);
        if (res == -1) res = blen;
        printf("%d\n", Discrete::get_num(res));
    }
}
```

1.8 无旋 Treap（FHQ Treap）

1.8.1 区间翻转

```
class Treap { public:
    int ch[MAXN][2];
    int dat[MAXN], siz[MAXN], val[MAXN];
    bool fl[MAXN];
```

```
    int tot, root;

    void init() {
        tot = 0, root = 0;
    }
    Treap() { init(); }

    inline int Newnode(int v) {
        val[++tot] = v;
        dat[tot] = rand();
        siz[tot] = 1;
        return tot;
    }
    inline void push_up(int rt) {
        siz[rt] = siz[ch[rt][0]] + siz[ch[rt][1]] + 1;
    }

    int build(int l, int r) {
        if (l > r) return 0;
        int mid = (l + r) >> 1;
        int newnode = Newnode(mid);
        ch[newnode][0] = build(l, mid - 1);
        ch[newnode][1] = build(mid + 1, r);
        push_up(newnode);
        return newnode;
    }

    inline void push_down(int rt) {
        if (fl[rt]) {
            swap(ch[rt][0], ch[rt][1]);
            if (ch[rt][0]) fl[ch[rt][0]] ^= 1;
            if (ch[rt][1]) fl[ch[rt][1]] ^= 1;
            fl[rt] = 0;
        }
    }

    void split(int rt, int k, int &x, int &y) { // 按照编号进行分裂
        if (!rt) x = y = 0;
        else {
            push_down(rt);
            if (k <= siz[ch[rt][0]]) {
                y = rt;
                split(ch[rt][0], k, x, ch[rt][0]);
            } else {
```

```
                x = rt;
                split(ch[rt][1], k - siz[ch[rt][0]] - 1, ch[rt][1], y);
            }
            push_up(rt);
        }
    }

    int merge(int x, int y) {
        if (!x || !y) return x + y;
        if (dat[x] < dat[y]) {
            push_down(x);
            ch[x][1] = merge(ch[x][1], y);
            push_up(x);
            return x;
        } else {
            push_down(y);
            ch[y][0] = merge(x, ch[y][0]);
            push_up(y);
            return y;
        }
    }

    void dfs(int rt) {
        push_down(rt);
        if (ch[rt][0]) dfs(ch[rt][0]);
        printf("%d ", val[rt]);
        if (ch[rt][1]) dfs(ch[rt][1]);
    }
} tree;

int main() {
    int n, q; scanf("%d%d", &n, &q);
    tree.build(1, n);
    while (q--) {
        int l, r; scanf("%d%d", &l, &r);
        int a, b, c;
        tree.split(tree.root, l - 1, a, b);
        tree.split(b, r - l + 1, b, c);
        tree.fl[b] ^= 1;
        tree.root = tree.merge(a, tree.merge(b, c));
    }
    tree.dfs(tree.root);
}
```

1.8.2 可持久化 FHQ

第一行包含一个正整数 n，表示操作的总数。
接下来 n 行，每行包含三个整数，第 i 行记为 v_i, opt_i, x_i。
v_i 表示基于的过去版本号，opt_i 表示操作的序号，x_i 表示参与操作的数值。

```
class FHQ { public:
    const int MAXM = 50 * MAXN; // 50倍注意!
    int ch[MAXM][2];
    int dat[MAXM], siz[MAXM], val[MAXM];
    int tot, root;

    void init() {   // 初始化
        tot = 0;
        root = 0;
        siz[0] = val[0] = 0;
    }

    inline void push_up(int rt) {   // 传递信息
        siz[rt] = siz[ch[rt][0]] + siz[ch[rt][1]] + 1;
    }

    inline int New(int v) { // 新建一个节点，value = v
        val[++tot] = v;
        dat[tot] = rand();
        siz[tot] = 1;
        ch[tot][0] = ch[tot][1] = 0;
        return tot;
    }

    inline int Copy(int rt) {   // 复制点的信息
        int newnode = ++tot;
        val[tot] = val[rt], dat[tot] = dat[rt], siz[tot] = siz[rt];
        ch[tot][0] = ch[rt][0], ch[tot][1] = ch[rt][1];
        return newnode;
    }

    inline int merge(int x, int y) {    // 合并
        if (!x || !y) return x + y;
        if (dat[x] < dat[y]) {
            int newnode = Copy(x);
            ch[newnode][1] = merge(ch[newnode][1], y);
            push_up(newnode);
            return newnode;
        } else {
```

```
            int newnode = Copy(y);
            ch[newnode][0] = merge(x, ch[newnode][0]);
            push_up(newnode);
            return newnode;
        }
    }

    inline void split(int rt, int v, int &x, int &y) { // 按照权值进行分裂
        if (!rt) x = y = 0;
        else {
            if (val[rt] <= v) {
                x = Copy(rt);
                split(ch[x][1], v, ch[x][1], y);
                push_up(x);
            } else {
                y = Copy(rt);
                split(ch[y][0], v, x, ch[y][0]);
                push_up(y);
            }
        }
    }

    void del(int &rt, int v) {  // 删除value为v的数
        int x = 0, y = 0, z = 0;
        split(rt, v, x, z);
        split(x, v - 1, x, y);
        y = merge(ch[y][0], ch[y][1]);
        rt = merge(x, merge(z, y));
    }

    void insert(int &rt, int v) {   // 插入value为v的数
        int x = 0, y = 0, z = 0;
        split(rt, v, x, y);
        z = New(v);
        rt = merge(x, merge(z, y));
    }

    int get_val(int rt, int k) {    // 得到第k大的数（从小到大）的value
        if (k == siz[ch[rt][0]] + 1) return val[rt];
        else if (k <= siz[ch[rt][0]]) return get_val(ch[rt][0], k);
        else return get_val(ch[rt][1], k - siz[ch[rt][0]] - 1);
    }

    int get_Kth(int &rt, int v) {   // 查询v的排名
```

```
        int x, y;
        split(rt, v - 1, x, y);
        int ans = siz[x] + 1;
        rt = merge(x, y);
        return ans;
    }

    int get_pre(int &rt, int v) {   // 求前驱，若不存在返回-2147483647
        int x, y, ans;
        split(rt, v - 1, x, y);
        if (!x) return -2147483647;
        ans = get_val(x, siz[x]);
        rt = merge(x, y);
        return ans;
    }

    int get_next(int &rt, int v) {  // 求后驱，若不存在返回2147483647
        int x, y, ans;
        split(rt, v, x, y);
        if (!y) return 2147483647;
        ans = get_val(y, 1);
        rt = merge(x, y);
        return ans;
    }
} tree;

int root[MAXN];

int main() {
    int q; scanf("%d", &q);
    tree.init();
    for (int i = 1; i <= q; i++) {
        int ver, opt, x;
        scanf("%d%d%d", &ver, &opt, &x);
        root[i] = root[ver];
        switch (opt) {
            // 插入x
            case 1: tree.insert(root[i], x); break;
            // 删除x（若有多个相同的数，应只删除一个，如果没有请忽略该操作）
            case 2: tree.del(root[i], x); break;
            // 查询x的排名（排名定义为比当前数小的数的个数+1）
            case 3: printf("%d\n", tree.get_Kth(root[i], x)); break;
            // 查询排名为x的数
            case 4: printf("%d\n", tree.get_val(root[i], x)); break;
```

```
            //
                求x的前驱（前驱定义为小于x，且最大的数，如不存在输出-2147483647）
            case 5: printf("%d\n", tree.get_pre(root[i], x)); break;
            //
                求x的后继（后继定义为大于x，且最小的数，如不存在输出2147483647）
            default: printf("%d\n", tree.get_next(root[i], x));
        }
    }

}
```

1.9 树套树

1.9.1 带修主席树

```
/*
  input       output
  5 3
  3 2 1 4 7
  Q 1 4 3     3
  C 2 6
  Q 2 5 3     6
*/
class BIT { public:
    // HJT begin
    int ch[MAXN * 400][2], sum[MAXN * 400], tot = 0;
    inline void push_up(int rt) {
        sum[rt] = sum[ch[rt][0]] + sum[ch[rt][1]];
    }
    int update(int rt, int pos, int val, int be, int en) {
        int nrt = ++tot;
        ch[nrt][0] = ch[nrt][1] = sum[nrt] = 0;
        if (be == en) {
            sum[nrt] = sum[rt] + val;
            return nrt;
        }
        int mid = (be + en) >> 1;
        if (pos <= mid) {
            ch[nrt][0] = update(ch[rt][0], pos, val, be, mid);
            ch[nrt][1] = ch[rt][1];
        } else {
            ch[nrt][0] = ch[rt][0];
            ch[nrt][1] = update(ch[rt][1], pos, val, mid + 1, en);
        }
```

```
        push_up(nrt);
        return nrt;
    }
    // HJT end
    int n, c_len, root[MAXN];
    void init(int _n, int _c_len) {
        c_len = _c_len, n = _n;
        for (int i = 1; i <= c_len; i++) root[i] = i;
        tot = c_len;
    }
    inline int lowbit(int x) { return x & (-x); }
    void insert(int pos, int pos_val, int val) {
        for (int i = pos; i <= n; i += lowbit(i)) root[i] = update(root[i],
            pos_val, val, 1, c_len);
    }
    int t1[MAXN], t2[MAXN], n1, n2;
    inline int Kth(int be, int en, int k) {
        if (be >= en) return be;
        int mid = (be + en) >> 1, delta = 0;

        for (int i = 1; i <= n1; i++) delta -= sum[ch[t1[i]][0]];
        for (int i = 1; i <= n2; i++) delta += sum[ch[t2[i]][0]];
        if (delta >= k) {
            for (int i = 1; i <= n1; i++) t1[i] = ch[t1[i]][0];
            for (int i = 1; i <= n2; i++) t2[i] = ch[t2[i]][0];
            return Kth(be, mid, k);
        } else {
            for (int i = 1; i <= n1; i++) t1[i] = ch[t1[i]][1];
            for (int i = 1; i <= n2; i++) t2[i] = ch[t2[i]][1];
            return Kth(mid + 1, en, k - delta);
        }
    }
    int query(int l, int r, int k) {
        n1 = n2 = 0;
        for (int i = l - 1; i >= 1; i -= lowbit(i)) t1[++n1] = root[i];
        for (int i = r; i >= 1; i -= lowbit(i)) t2[++n2] = root[i];
        return Kth(1, c_len, k);
    }
} tree;
int ai[MAXN];
int opt[MAXN], l[MAXN], r[MAXN], k[MAXN], x[MAXN], y[MAXN];
int main() {
    int n, m; cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> ai[i];
```

```
    for (int i = 1; i <= n; i++) Discrete::insert(ai[i]);
    for (int i = 1; i <= m; i++) {
        char op; cin >> op;
        if (op == 'Q') {
            opt[i] = 1; cin >> l[i] >> r[i] >> k[i];
        } else {
            opt[i] = 2; cin >> x[i] >> y[i];
            Discrete::insert(y[i]);
        }
    }
    Discrete::init();
    for (int i = 1; i <= n; i++) ai[i] = Discrete::val2id(ai[i]);
    for (int i = 1; i <= m; i++) {
        if (opt[i] == 2) y[i] = Discrete::val2id(y[i]);
    }
    tree.init(n, Discrete::blen);

    for (int i = 1; i <= n; i++) tree.insert(i, ai[i], 1);
    for (int i = 1; i <= m; i++) {
        if (opt[i] == 1) {  // 表示查询下标在区间[l,r]中的第k小的数
            cout << Discrete::id2val(tree.query(l[i], r[i], k[i])) << '\n';
        } else {  // 表示将a[x]改为y
            tree.insert(x[i], ai[x[i]], -1);
            ai[x[i]] = y[i];
            tree.insert(x[i], ai[x[i]], 1);
        }
    }
}
```

1.9.2 区间修改区间查询第 K 大（[ZJOI2013]K 大数查询）

```
/*
  树状数组套主席套线段树
  可重集的并是不去除重复元素的，如{1,1,4} & {5,1,4}={1,1,4,5,1,4}
  input     output
  2 5
  1 1 2 1
  1 1 2 2
  2 1 1 2   1
  2 1 1 1   2
  2 1 2 3   1
*/
class BIT { public:
```

```
    class SEG { public:
        int tot_rt, ch[MAXN * 400][2]; ll lazy[MAXN * 400], sum[MAXN * 400];
#define lson ch[rt][0]
#define rson ch[rt][1]
        inline void push_up(int rt) {
            sum[rt] = sum[lson] + sum[rson];
        }
        inline void push_down(int rt, int len_left, int len_right) {
            if (lazy[rt]) {
                if (!ch[rt][0]) ch[rt][0] = ++tot_rt;
                if (!ch[rt][1]) ch[rt][1] = ++tot_rt;
                sum[lson] += lazy[rt] * len_left, sum[rson] += lazy[rt] *
                   len_right;
                lazy[lson] += lazy[rt], lazy[rson] += lazy[rt];
                lazy[rt] = 0;
            }
        }
        int change(int rt, int L, int R, int val, int be, int en) {
            if (!rt) rt = ++tot_rt;
            if (L <= be && R >= en) {
                sum[rt] += (ll) (en - be + 1);
                lazy[rt] += (ll) val;
                return rt;
            }
            int mid = (be + en) >> 1;
            push_down(rt, mid - be + 1, en - (mid + 1) + 1);
            if (L <= mid) lson = change(lson, L, R, val, be, mid);
            if (R > mid) rson = change(rson, L, R, val, mid + 1, en);
            push_up(rt);
            return rt;
        }
        ll query(int rt, int L, int R, int be, int en) {
            if (!rt) return 0;
            if (L <= be && R >= en) return sum[rt];
            int mid = (be + en) >> 1;
            push_down(rt, mid - be + 1, en - (mid + 1) + 1);
            ll ans = 0;
            if (L <= mid) ans += query(lson, L, R, be, mid);
            if (R > mid) ans += query(rson, L, R, mid + 1, en);
            return ans;
        }
    } seg;

    int n, c_len, root[MAXN];
```

```
    void init(int _n, int _c_len) {
        c_len = _c_len, n = _n;
        for (int i = 1; i <= c_len; i++) root[i] = i;
        seg.tot_rt = _c_len;
    }
    inline int lowbit(int x) { return x & (-x); }
    inline int log(int x) { return 1ll << (int) (log2(x)); }
    void insert(int l, int r, int c) {
        for (int i = c_len - c + 1; i <= c_len; i += lowbit(i))
           seg.change(root[i], l, r, 1, 1, n);
    }
    int query(int l, int r, ll k) {
        int ans = 0;
        ll sum = 0;
        for (int i = log(c_len); i != 0; i >>= 1) {
            if (ans + i > c_len) continue;
            ll tmp = seg.query(root[ans + i], l, r, 1, n) + sum;
            if (tmp < k) ans += i, sum = tmp;
        }
        ans++;
        return c_len - ans + 1;
    }
} tree;

int opt[MAXN], l[MAXN], r[MAXN]; ll c[MAXN];
int main() {
    int n, m; scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; i++) {
        scanf("%d%d%d%lld", &opt[i], &l[i], &r[i], &c[i]);
        if (opt[i] == 1) Discrete::push(c[i]);
    }
    Discrete::init(); // 离散化
    int c_len = Discrete::blen;
    tree.init(n, c_len);  // n为[l,r]，c_len表示离散化后数字个数
    for (int i = 1; i <= m; i++) {
        if (opt[i] == 1) {  // 表示将c加入到编号在[l,r]内的集合中
            tree.insert(l[i], r[i], Discrete::val2id(c[i]));
        } else {  // 表示查询编号在[l,r]内的集合的并集中，第c大的数是多少
            printf("%lld\n", Discrete::id2val(tree.query(l[i], r[i], c[i])));
        }
    }
}
```

1.10 KD 树

1.10.1 平面最近点对

时间复杂度：单次查询最近点的时间复杂度 $O(n)$.

```
const int MAXN = 2e5 + 5;   // 点的个数
class KD {
public:
    struct node {
        double x, y;
    } T[MAXN];
    int ch[MAXN][2];
    double L[MAXN], R[MAXN], D[MAXN], U[MAXN];
    inline void push_up(int rt) {
        L[rt] = R[rt] = T[rt].x;
        D[rt] = U[rt] = T[rt].y;
        if (ch[rt][0]) {
            L[rt] = min(L[rt], L[ch[rt][0]]), R[rt] = max(R[rt],
               R[ch[rt][0]]),
            D[rt] = min(D[rt], D[ch[rt][0]]), U[rt] = max(U[rt],
               U[ch[rt][0]]);
        }
        if (ch[rt][1]) {
            L[rt] = min(L[rt], L[ch[rt][1]]), R[rt] = max(R[rt],
               R[ch[rt][1]]),
            D[rt] = min(D[rt], D[ch[rt][1]]), U[rt] = max(U[rt],
               U[ch[rt][1]]);
        }
    }

    int d[MAXN];
    int build(int l, int r) {
        if (l > r) return 0;
        int mid = (l + r) >> 1;
        double av1 = 0, av2 = 0, va1 = 0, va2 = 0;
        for (int i = l; i <= r; i++) av1 += T[i].x, av2 += T[i].y;
        av1 /= (r - l + 1);
        av2 /= (r - l + 1);
        for (int i = l; i <= r; i++) va1 += (av1 - T[i].x) * (av1 - T[i].x),
           va2 += (av2 - T[i].y) * (av2 - T[i].y);
        if (va1 > va2)
            d[mid] = 1, nth_element(T + l, T + mid, T + r + 1,
                                    [&](const node &ta, const node &tb) {
                                       return ta.x < tb.x; });
```

```
        else
            d[mid] = 2, nth_element(T + l, T + mid, T + r + 1,
                                    [&](const node &ta, const node &tb) {
                                        return ta.y < tb.y; });
        ch[mid][0] = build(l, mid - 1);
        ch[mid][1] = build(mid + 1, r);
        push_up(mid);
        return mid;
    }

    double f(int a, int b) {
        double ans = 0;
        if (L[b] > T[a].x) ans += (L[b] - T[a].x) * (L[b] - T[a].x);
        if (R[b] < T[a].x) ans += (T[a].x - R[b]) * (T[a].x - R[b]);
        if (D[b] > T[a].y) ans += (D[b] - T[a].y) * (D[b] - T[a].y);
        if (U[b] < T[a].y) ans += (T[a].y - U[b]) * (T[a].y - U[b]);
        return ans;
    }
    double ans;
    double dist(int a, int b) {
        return (T[a].x - T[b].x) * (T[a].x - T[b].x) + (T[a].y - T[b].y) *
               (T[a].y - T[b].y);
    }
    void query(int l, int r, int rt) {
        if (l > r) return;
        int mid = (l + r) >> 1;
        if (mid != rt) ans = min(ans, dist(rt, mid));
        if (l == r) return ;
        double distl = f(rt, ch[mid][0]), distr = f(rt, ch[mid][1]);
        if (distl < ans && distr < ans) {
            if (distl < distr) {
                query(l, mid - 1, rt);
                if (distr < ans) query(mid + 1, r, rt);
            } else {
                query(mid + 1, r, rt);
                if (distl < ans) query(l, mid - 1, rt);
            }
        } else {
            if (distl < ans) query(l, mid - 1, rt);
            if (distr < ans) query(mid + 1, r, rt);
        }
    }
    void init() { ans = inf_ll; }
    double get_res() { return ans; }
```

```
} tree;

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%lf%lf", &tree.T[i].x, &tree.T[i].y);
    tree.init();
    tree.build(1, n);
    for (int i = 1; i <= n; i++) {
        tree.query(1, n, i);
    }
    printf("%.4lf\n", sqrt(tree.get_res()));
}
```

1.10.2 K 远点对 ([CQOI2016])

已知平面内 N 个点的坐标，求欧氏距离下的第 K 远点对。
两个点的欧氏距离为 $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$
原题数据范围：$N \leq 1e5, 1 \leq K \leq 100$
时间复杂度：$O(kn \log n)$.

```
/*
    input
    10 5
    0 0, 0 1, 1 0, 1 1, 2 0, 2 1, 1 2, 0 2, 3 0, 3 1
    ouput
    9
*/
const int MAXN = 1e5 + 5;   // 点的个数
class KD {
public:
    priority_queue<ll, vector<ll>, greater<ll> > q;
    struct node {
        int x, y;
    } T[MAXN];
    int ch[MAXN][2], L[MAXN], R[MAXN], D[MAXN], U[MAXN];

    inline void push_up(int rt) {
        L[rt] = R[rt] = T[rt].x;
        D[rt] = U[rt] = T[rt].y;
        if (ch[rt][0]) {
            L[rt] = min(L[rt], L[ch[rt][0]]), R[rt] = max(R[rt],
               R[ch[rt][0]]),
            D[rt] = min(D[rt], D[ch[rt][0]]), U[rt] = max(U[rt],
```

```
            U[ch[rt][0]]);
        }
        if (ch[rt][1]) {
            L[rt] = min(L[rt], L[ch[rt][1]]), R[rt] = max(R[rt],
                R[ch[rt][1]]),
            D[rt] = min(D[rt], D[ch[rt][1]]), U[rt] = max(U[rt],
                U[ch[rt][1]]);
        }
    }

    int build(int l, int r) {
        if (l > r) return 0;
        int mid = (l + r) >> 1;
        double av1 = 0, av2 = 0, va1 = 0, va2 = 0;
        for (int i = l; i <= r; i++) av1 += T[i].x, av2 += T[i].y;
        av1 /= (r - l + 1);
        av2 /= (r - l + 1);
        for (int i = l; i <= r; i++) va1 += (av1 - T[i].x) * (av1 - T[i].x),
            va2 += (av2 - T[i].y) * (av2 - T[i].y);
        if (va1 > va2)
            nth_element(T + l, T + mid, T + r + 1, [&](const node &ta, const
                node &tb) { return ta.x < tb.x; });
        else nth_element(T + l, T + mid, T + r + 1, [&](const node &ta, const
            node &tb) { return ta.y < tb.y; });
        ch[mid][0] = build(l, mid - 1);
        ch[mid][1] = build(mid + 1, r);
        push_up(mid);
        return mid;
    }

    ll sq(int x) { return (ll) x * x; }
    ll dist(int a, int b) {
        return max(sq(T[a].x - L[b]), sq(T[a].x - R[b])) + max(sq(T[a].y -
            D[b]), sq(T[a].y - U[b]));
    }
    void query(int l, int r, int rt) {
        if (l > r) return;
        int mid = (l + r) >> 1;
        ll tmp = sq(T[mid].x - T[rt].x) + sq(T[mid].y - T[rt].y);
        if (tmp > q.top()) q.pop(), q.push((tmp));
        ll distl = dist(rt, ch[mid][0]), distr = dist(rt, ch[mid][1]);
        if (distl > q.top() && distr > q.top()) {
            if (distl > distr) {
                query(l, mid - 1, rt);
```

```
                if (distr > q.top()) query(mid + 1, r, rt);
            } else {
                query(mid + 1, r, rt);
                if (distl > q.top()) query(l, mid - 1, rt);
            }
        } else {
            if (distl > q.top()) query(l, mid - 1, rt);
            if (distr > q.top()) query(mid + 1, r, rt);
        }
    }
    void init(int k) {
        k *= 2;
        for (int i = 1; i <= k; i++) q.push(0);
    }
    ll get_res() {
        return q.top();
    }
} tree;

int main() {
    int n, k;
    scanf("%d%d", &n, &k);
    tree.init(k);
    for (int i = 1; i <= n; i++) scanf("%d%d", &tree.T[i].x, &tree.T[i].y);
    tree.build(1, n);
    for (int i = 1; i <= n; i++) {
        tree.query(1, n, i);
    }
    printf("%lld\n", tree.get_res());
}
```

1.10.3 高维空间上的操作

在一个初始值全为 0 的 $n \times n$ 的二维矩阵上，进行若干次操作，每次操作为以下两种之一：

1 x y A 将坐标 (x, y) 上的数加上 A。

2 x1 y1 x2 y2 输出以 $(x1, y1)$ 为左下角，$(x2, y2$ 为右上角的矩形内（包括矩形边界）的数字和。

原题数据范围：$1 \leq n \leq 5e5, 1 \leq q \leq 2e5$

时间复杂度：单次查询时间最优 $O(\log n)$，最坏 $O(\sqrt{n})$。将结论扩展至 k 维，最坏复杂度 $O(n^{1-\frac{1}{k}})$

```
/*
    intput      output
    4
    1 2 3 3
    2 1 1 3 3   3
```

```
    1 1 1 1
    2 1 1 0 7   5
    3
*/
const int MAXN = 2e5 + 5;   // 操作次数
class KD {
public:
    struct node {
        int x, y, v;
    } T[MAXN];
    int ch[MAXN][2], L[MAXN], R[MAXN], D[MAXN], U[MAXN];
    int siz[MAXN], sum[MAXN], g[MAXN], d[MAXN];

    inline void push_up(int rt) {
        siz[rt] = siz[ch[rt][0]] + siz[ch[rt][1]] + 1;
        sum[rt] = sum[ch[rt][0]] + sum[ch[rt][1]] + T[rt].v;
        L[rt] = R[rt] = T[rt].x;
        U[rt] = D[rt] = T[rt].y;
        if (ch[rt][0]) {
            L[rt] = min(L[rt], L[ch[rt][0]]), R[rt] = max(R[rt],
               R[ch[rt][0]]),
            D[rt] = min(D[rt], D[ch[rt][0]]), U[rt] = max(U[rt],
               U[ch[rt][0]]);
        }
        if (ch[rt][1]) {
            L[rt] = min(L[rt], L[ch[rt][1]]), R[rt] = max(R[rt],
               R[ch[rt][1]]),
            D[rt] = min(D[rt], D[ch[rt][1]]), U[rt] = max(U[rt],
               U[ch[rt][1]]);
        }
    }

    int build(int l, int r) {
        if (l > r) return 0;
        int mid = (l + r) >> 1;
        double av1 = 0, av2 = 0, va1 = 0, va2 = 0;
        for (int i = l; i <= r; i++) av1 += T[g[i]].x, av2 += T[g[i]].y;
        av1 /= (r - l + 1);
        av2 /= (r - l + 1);
        for (int i = l; i <= r; i++) {
            va1 += (av1 - T[g[i]].x) * (av1 - T[g[i]].x),
                    va2 += (av2 - T[g[i]].y) * (av2 - T[g[i]].y);
        }
        if (va1 > va2)
```

```
46            nth_element(g + l, g + mid, g + r + 1, [&](int ta, int tb) {
                  return T[ta].x < T[tb].x; }), d[g[mid]] = 1;
47        else nth_element(g + l, g + mid, g + r + 1, [&](int ta, int tb) {
                  return T[ta].y < T[tb].y; }), d[g[mid]] = 2;
48        ch[g[mid]][0] = build(l, mid - 1);
49        ch[g[mid]][1] = build(mid + 1, r);
50        push_up(g[mid]);
51        return g[mid];
52    }
53
54    const double bad_para = 0.725;
55    int tot;
56    bool bad(int rt) {
57        return bad_para * siz[rt] <= (double) max(siz[ch[rt][0]],
                  siz[ch[rt][1]]);
58    }
59    void dfs(int rt) {
60        if (!rt) return;
61        dfs(ch[rt][0]);
62        g[++tot] = rt;
63        dfs(ch[rt][1]);
64    }
65    void rebuild(int &rt) {
66        tot = 0;
67        dfs(rt);
68        rt = build(1, tot);
69    }
70    void insert(int &rt, int v) {
71        if (!rt) {
72            rt = v;
73            push_up(rt);
74            return;
75        }
76        if (d[rt] == 1) {
77            if (T[v].x <= T[rt].x) insert(ch[rt][0], v);
78            else insert(ch[rt][1], v);
79        } else {
80            if (T[v].y <= T[rt].y) insert(ch[rt][0], v);
81            else insert(ch[rt][1], v);
82        }
83        push_up(rt);
84        if (bad(rt)) rebuild(rt);
85    }
86    int query(int rt, int xl, int xr, int yl, int yr) {
```

```
87          if (!rt ||  xr < L[rt] || xl > R[rt] || yr < D[rt] || yl > U[rt])
               return 0;
88          if (xl <= L[rt] && R[rt] <= xr &&  yl <= D[rt] && U[rt] <= yr) return
               sum[rt];
89          int ans = 0;
90          if (xl <= T[rt].x && T[rt].x <= xr && yl <= T[rt].y && T[rt].y <= yr)
               ans += T[rt].v;
91          return query(ch[rt][0], xl, xr, yl, yr) + query(ch[rt][1], xl, xr,
               yl, yr) + ans;
92      }
93
94  } tree;
95
96  int main() {
97      int N, opt;
98      scanf("%d", &N);
99      int tot = 0, lastans = 0, root = 0;
100     while (~scanf("%d", &opt)) {
101         if (opt == 1) {
102             tot++;
103             scanf("%d%d%d", &tree.T[tot].x, &tree.T[tot].y, &tree.T[tot].v);
104             tree.T[tot].x ^= lastans, tree.T[tot].y ^= lastans, tree.T[tot].v
                   ^= lastans;
105             tree.insert(root, tot);
106         } else if (opt == 2) {
107             int xl, yl, xr, yr;
108             scanf("%d%d%d%d", &xl, &yl, &xr, &yr);
109             xl ^= lastans, yl ^= lastans, xr ^= lastans, yr ^= lastans;
110             lastans = tree.query(root, xl, xr, yl, yr);
111             printf("%d\n", lastans);
112         } else break;
113     }
114 }
```

1.11 珂朵莉树/老司机树/ODT

1.11.1 set 实现珂朵莉树

1 l r x 将 $[l,r]$ 区间所有数加上 x

2 l r x 将 $[l,r]$ 区间所有数改成 x

3 l r x 输出将 $[l,r]$ 区间从小到大排序后的第 x 个数是的多少 (即区间第 x 小，数字大小相同算多次，保证 $1 \le x \le r-l+1$)

4 l r x y 输出 $[l,r]$ 区间每个数字的 x 次方的和模 y 的值 (即 $\sum_{i=l}^{r} a_i^x \% y$)

时间复杂度：用 set 实现 $O(n \log\log n)$

如果要保证复杂度正确，必须保证数据随机。

```
#define IT set<node>::iterator
const int MAXN = 1e5 + 5;
const int MOD7 = 1e9 + 7;

ll powmod(ll base, ll times, ll mod) {
    ll p = 1;
    ll ans = base % mod;
    while (times) {
        if (times & 1) p = p * ans % mod;
        ans = ans * ans % mod;
        times >>= 1;
    }
    return p;
}
class ODT {
public:
    struct node {
        int l, r;
        mutable ll val; // 玄学mutable注意！
        node() {}
        node(int _l, int _r = -1, ll _val = 0) { l = _l, r = _r, val = _val; }
        bool operator<(const node &tb) const { return l < tb.l; }
    };
    set<node> st;
    IT split(int pos) {
        IT it = st.lower_bound(node(pos));
        if (it != st.end() && it->l == pos) return it;
        --it;
        int ul = it->l, ur = it->r; ll uv = it->val;
        st.erase(it);
        st.insert(node(ul, pos - 1, uv));
        return st.insert(node(pos, ur, uv)).first;
    }
    void add(int l, int r, ll v = 1) {  // 对一段区间加上一个数
        IT itl = split(l), itr = split(r + 1);
        for (; itl != itr; itl++) itl->val += v;
    }
    void assign_val(int l, int r, ll v = 0) {   // 对一段区间进行赋值
        IT itl = split(l), itr = split(r + 1);
        st.erase(itl, itr);
        st.insert(node(l, r, v));
    }
```

```
    ll rank(int l, int r, int k) {
        vector<pair<ll, int>> vp;
        IT itl = split(l), itr = split(r + 1);
        vp.clear();
        for (; itl != itr; itl++)
            vp.push_back(pair<ll, int>(itl->val, (itl->r) - (itl->l) + 1));
        sort(vp.begin(), vp.end());
        for (vector<pair<ll, int>>::iterator it = vp.begin(); it != vp.end();
           it++) {
            k -= it->second;
            if (k <= 0) return it->first;
        }
        return -1ll;
    }
    ll sum(int l, int r, int times, int mod) {
        IT itl = split(l), itr = split(r + 1);
        ll ans = 0;
        for (; itl != itr; itl++)
            ans = (ans + (ll) (itl->r - itl->l + 1) * powmod(itl->val, (ll)
               times, (ll) mod) % mod) % mod;
        return ans;
    }
    void insert(int l, int r, ll v) {
        st.insert(node(l, r, v));
    }
} tree;

ll seed, vmax;
ll rnd() {
    ll ret = seed;
    seed = (seed * 7 + 13) % MOD7;
    return ret;
}

ll a[MAXN];
int main() {
    int n, m;
    scanf("%d%d%lld%lld", &n, &m, &seed, &vmax);
    for (int i = 1; i <= n; i++) {
        a[i] = (rnd() % vmax) + 1;
        tree.insert(i, i, a[i]);
    }
    tree.insert(n + 1, n + 1, 0);
    int lines = 0;
```

```
    for (int i = 1; i <= m; i++) {
        int opt = int(rnd() % 4) + 1, l = int(rnd() % n) + 1, r = int(rnd() %
            n) + 1;
        if (l > r) swap(l, r);
        int x, y;
        // 原题神奇的操作，可以不用理会
        if (opt == 3) x = int(rnd() % (r - l + 1)) + 1;
        else x = int(rnd() % vmax) + 1;
        if (opt == 4) y = int(rnd() % vmax) + 1;

        if (opt == 1) tree.add(l, r, x);
        else if (opt == 2) tree.assign_val(l, r, x);
        else if (opt == 3) printf("%lld\n", tree.rank(l, r, x));
        else printf("%lld\n", tree.sum(l, r, x, y));
    }
}
```

2 字符串

2.1 字符串哈希

2.1.1 区间一维哈希

```
typedef unsigned long long ull;
namespace hash {
    const ull seed = 19260817;
    ull base[SIZE], hash[SIZE];

    void init() {
        base[0] = 1;
        for (int i = 1; i < SIZE; i++)base[i] = base[i - 1] * seed;
    }

    ull _hash(int l, int r) {
        return hash[r] - hash[l - 1] * base[r - l + 1];
    }

    void getHash(char str[], int len) {
        for (int i = 1; i <= len; i++) hash[i] = hash[i - 1] * seed + str[i]
            - 'a' + 3;
    }
```

2.2 Next 函数

2.2.1 求 next 函数

```
/*
    input: ABCABDABCD
    index   0   1   2   3   4   5   6   7   8   9   10
    x[]     A   B   C   A   B   D   A   B   C   D   \0
    nxt[]   -1  -1  -1  0   1   -1  0   1   2   -1  0
*/
// call: scanf("%s", str+1); get_next(str+1, strlen(str+1), nex+1);
void get_next(char x[], int x_len, int nxt[]) {
    int i, j;
    for (nxt[0] = j = -1, i = 1; i < x_len; nxt[i++] = j) {
        while (~j && x[j + 1] != x[i])j = nxt[j];
        if (x[j + 1] == x[i])j++;
    }
}

/*
    input: ABCABDABCD
    index   1   2   3   4   5   6   7   8   9   10  11
    x[]     A   B   C   A   B   D   A   B   C   D   \0
    nxt[]   0   0   0   1   2   0   1   2   3   0   0
*/
// call: scanf("%s", str+1); get_next(str, strlen(str+1), nex);
void get_next(char x[], int x_len, int nxt[]) {
    nxt[1] = 0;
    for (int i = 2, j = 0; i <= x_len; i++) {
        while (j && x[j + 1] != x[i]) j = nxt[j];
        if (x[j + 1] == x[i]) ++j;
        nxt[i] = j;
    }
}
```

2.2.2 求出每个循环节的数量和终点位置（HDU1358）

```
/*
    input   output
    3       2 2
    aaa     3 3

    12              2 2
    aabaabaabaab    6 2
```

```
                    9 3
                    12 4
*/
int main() {
    int n;
    int cas = 0;
    while (~scanf("%d", &n) && n) {
        scanf("%s", str+1);
        get_next(str+1, n, nex+1);  // 方法1
        for (int i = 2; i <= n; i++) {
            if (nex[i] != -1 && (i % (i - nex[i] - 1) == 0))
                printf("%d %d\n", i, i / (i - nex[i] - 1));
        }
        puts("");
    }
}
```

2.2.3 求同时是前缀和后缀的串长（**POJ2752**）

```
/*
    input               output
    ababcababababcabab  2 4 9 18
    aaaaa               1 2 3 4 5
*/
int main() {
    while (~scanf("%s", s + 1)) {
        int cnt = 0;
        int len;
        get_next(s + 1, len = strlen(s + 1), nex + 1);  // 方法1
        for (int t = nex[len]; ~t; t = nex[t+1]) {
            if (s[t+1] == s[len]) ans[cnt++] = t + 1;
        }
        for (int i = cnt - 1; i >= 0; i--)printf("%d ", ans[i]);
        printf("%d\n", len);
    }
}
```

2.2.4 求字符串每个前缀和串匹配成功的次数和（**HDU3336**）

```
/*
    input   output
    4       6
    abab
```

```
    6       12
    ababab
*/
int get_next(char x[], int x_len, int nxt[], int i, int j) {
    while (~j && x[j + 1] != x[i])j = nxt[j];
    if (x[j + 1] == x[i])j++;
    nxt[i++] = j;
    return j;
}

char str[MAXN];
int nex[MAXN], val[MAXN];

int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        int n;
        scanf("%d", &n);
        scanf("%s", str + 1);
        int len = strlen(str + 1);
        int last = -1;
        nex[1] = -1;
        int res = 0;
        for (int i = 1; i <= len; i++) {
            last = get_next(str + 1, len, nex + 1, i, last);
            if (nex[i] < 0) val[i] = 1;
            else val[i] = (val[nex[i] + 1] + 1) % mod;
            res = (res + val[i]) % mod;
        }
        printf("%d\n", res);
    }
}
```

2.2.5 求循环节数量（POJ2406）

```
/*
    input   output
    abcd    1
    aaaa    4
    ababab  3
*/
int main() {
```

```
    while (~scanf("%s", s + 1)) {
        if (s[1] == '.')break;
        int len;
        get_next(s + 1, len = strlen(s + 1), nex + 1);  // 方法1
        printf("%d\n", len % (len - nex[len] - 1) ? 1 : len / (len - nex[len]
            - 1));
    }
}
```

2.2.6 求第一个串的前缀和第二个串的后缀的最大匹配（HDU2594）

```
/*
    input       output
    clinton     0
    homer
    riemann     rie 3
    marjorie
*/
int main() {
    while (~scanf("%s%s", a + 1, b + 1)) {
        int la = strlen(a + 1), lb = strlen(b + 1);
        strcat(a + 1, b + 1);
        int len = la + lb;
        get_next(a + 1, len, nex + 1);  // 方法1
        int k;
        for (k = nex[len]; k >= la || k >= lb; k = nex[k+1]);
        if (k == -1) puts("0");
        else {
            for (int i = 0; i <= k; i++)printf("%c", a[i+1]);
            printf(" %d\n", k + 1);
        }
    }
}
```

2.2.7 求补上最少字母数量使得这是个循环串（HDU3746）

```
/*
    input:  output
    3
    aaa     0
    abca    2
    abcde   5
*/
```

```cpp
int main() {
    int T, len;
    scanf("%d", &T);
    while (T--) {
        scanf("%s", s + 1);
        get_next(s + 1, len = strlen(s + 1), nex + 1);  // 方法1
        int L = len - (nex[len] + 1);
        if (L < len && len % L == 0) puts("0");
        else printf("%d\n", L - len % L);
    }
}
```

2.2.8 习题整理

[NOI2014] 动物园

对于字符串 S 的前 i 个字符构成的子串，既是它的后缀同时又是它的前缀，并且该后缀与该前缀不重叠，将这种字符串的数量记作 $num[i]$.

res 为 $(num[i]+1)$ 的乘积.

时间复杂度：$O(n)$.

```cpp
int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        scanf("%s", str + 1);
        int len;
        get_next(str, len = strlen(str + 1), nex);  // 方法2
        for (int i = 1; i <= len; i++) {
            val[i] = (val[nex[i]] + 1) % mod;
        }
        ll res = 1ll;
        for (int i = 2, j = 0; i <= len; i++) {
            while (j && str[j + 1] != str[i]) j = nex[j];
            if (str[j + 1] == str[i]) j++;
            while (j * 2 > i) j = nex[j];   // 去除重叠的
            num[i] = val[j];
            // res = (res * ((ll) val[j] + 1ll)) % mod;
        }
        printf("%lld\n", res);
    }
}
```

2.3 KMP

2.3.1 统计模式串出现次数，出现位置，前缀 border 长度

```
/*
    时间复杂度O(x_len + y_len)，空间复杂度O(x_len)
*/
int nex[MAXN];
// x为模式串， y为文本串
// call: scanf("%s %s", a+1, b+1); kmp(b+1, strlen(b+1), a+1, strlen(a+1));
int kmp(char x[], int x_len, char y[], int y_len) {
    int i, j;
    int ans = 0;
    get_next(x, x_len, nex);    // 方法1
    for (j = -1, i = 0; i < y_len; i++) {
        while (~j && x[j + 1] != y[i])j = nex[j];
        if (x[j + 1] == y[i])j++;
        if (j == x_len - 1) {
            printf("%d\n", i - x_len + 2);  // 出现的位置，可选，从1开始计数
            ans++, j = nex[j];
        }
    }
    for (i = 0; i < x_len; i++) printf("%d ", nex[i] + 1);  //
       每个前缀的最长border的长度
    return ans;
}
```

2.3.2 矩阵加速 KMP, 求长度为 n 的不包含长度为 m 的子串的串个数（[HNOI2008]GT 考试）

$$\sum_{k=0}^{m-1} f[i-1][k] * g[k][j]$$

$f[i][j]$ 为长串匹配到第 i 位，短串最多可以匹配到第 j 位的方案数

$g[j][k]$ 为了计算长度为 j 的已经匹配好了的串可以用多少种数字变为 k，枚举一个数字，看它在短串中最长可以匹配到最多多长的前缀

```
/*
    kmp+矩阵加速
    求长度为n的不包含长度为m的子串的串个数
    input   4 3 100
            111
    output  81
*/
int nex[MAXN];
mat get_g(char x[], int x_len) {
```

```
    get_next(x, x_len, nex);
    mat g = mat(x_len, x_len);
    for (int i = 0; i < x_len; i++) {
        for (char ch = '0'; ch <= '9'; ch++) {
            int j = i;
            while (j && x[j+1] != ch) j = nex[j];
            if (x[j+1] == ch) j++;
            g.v[i][j] = (ll)(g.v[i][j] + 1ll) % mod;
        }
    }
    return g;
}
int n, m;
char str[MAXN];
int main() {
    scanf("%d%d%lld", &n, &m, &mod);
    scanf("%s", str + 1);
    mat g = get_g(str, strlen(str+1));
    g = g^n;
    mat f(m, 1);
    f.v[0][0] = 1;
    f =f*g;
    ll res = 0;
    for (int i = 0; i < m; i++) {
        res = (ll)(res + f.v[0][i]) % mod;
    }
    printf("%lld\n", res);
}
```

2.4 EXKMP

2.4.1 求 z 函数和 LCP

LCP：最长公共前缀

z 函数数组 z：串 b 与 b 的每一个后缀的 LCP 长度。

$extend$ 数组：串 b 与串 a 的每一个后缀的 LCP 长度。总时间复杂度：$O(|a|+|b|)$.

```
/*
    index       1   2   3   4   5   6   7   8
    char a[]    a   a   a   a   b   a   a   '\0'
    extend[]    4   3   2   1   0   2   1
    char b[]    a   a   a   a   a   '\0'
    z[]         5   4   3   2   1
*/
// call: scanf("%s", a+1); getLCP(a+1, strlen(a+1), z+1);
```

```
void getLCP(char T[], int T_len, int z[]) {
    int i, len = T_len;
    z[0] = len;
    for (i = 0; i < len - 1 && T[i] == T[i + 1]; i++);
    z[1] = i;
    int a = 1;
    for (int k = 2; k < len; k++) {
        int p = a + z[a] - 1, L = z[k - a];
        if ((k - 1) + L >= p) {
            int j = max((p - k + 1), 0);
            while (k + j < len && T[k + j] == T[j])j++;
            z[k] = j, a = k;
        } else z[k] = L;
    }
}

// call: scanf("%s%s", a+1, b+1); exkmp(a+1, strlen(a+1), b+1, strlen(b+1),
   ex+1, z+1);
void exkmp(char S[], int S_len, char T[], int T_len, int extend[], int z[]) {
    getLCP(T, T_len, z);
    int a = 0;
    int MinLen = min(S_len, T_len);
    while (a < MinLen && S[a] == T[a])a++;
    extend[0] = a, a = 0;
    for (int k = 1; k < S_len; k++) {
        int p = a + extend[a] - 1, L = z[k - a];
        if ((k - 1) + L >= p) {
            int j = max((p - k + 1), 0);
            while (k + j < S_len && j < T_len && S[k + j] == T[j])j++;
            extend[k] = j;
            a = k;
        } else extend[k] = L;
    }
}
```

2.4.2 循环位移有多少数比原数大小相等，去重（HDU4333）

包含对获得的串进行去重。

总时间复杂度：$O(n)$

```
/*
    input   output
    1       Case 1: 1 1 1
    341
```

```
*/
char str[MAXN];
int z[MAXN];
int main() {
    int T; scanf("%d", &T);
    int kase = 1;
    while (T--) {
        scanf("%s", str + 1);
        int len = strlen(str + 1);
        for (int i = 1; i <= len; i++) str[len + i] = str[i];
        getLCP(str + 1, len * 2, z + 1);
        int L = 0, E = 0, G = 0;
        for (int i = 1; i <= len; i++) {
            if (z[i] >= len) E++;
            else if (str[z[i]+1] > str[z[i]+i]) L++;
            else G++;
        }
        printf("Case %d: ", kase++);
        printf("%d %d %d\n", L/E, E/E, G/E);    // 去重相关
    }
}
```

2.5 AC 自动机

2.5.1 标准的 AC 自动机

```
const int MAXN = 1e5 + 5;

class AC_Automaton {
public:
    int T[MAXN][26], val[MAXN], top;  //Trie相关
    int fail[MAXN];
    queue<int> q;
    // int pid[SIZE];  //对应字符串编号

    void init() {
        top = 1;
        memset(T[0], 0, sizeof(T[0]));
        memset(val, 0, sizeof(val));
//        memset(pid, 0, sizeof(pid));
    }

    AC_Automaton() {
        init();
```

```
    }

    void insert(char str[], int lenstr, int _pid) {
        int u = 0;
        for (int i = 1; i <= lenstr; i++) {
            int ch = str[i] - 'a';
            if (!T[u][ch]) {
                memset(T[top], 0, sizeof(T[top]));
                T[u][ch] = top++;
            }
            u = T[u][ch];
        }
        val[u]++;
        // pid[u] = _pid;
    }

    void build() {
        for (int i = 0; i < 26; i++)
            if (T[0][i]) {
                fail[T[0][i]] = 0;
                q.push(T[0][i]);
            }
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int i = 0; i < 26; i++)
                if (T[u][i]) {
                    fail[T[u][i]] = T[fail[u]][i];
                    q.push(T[u][i]);
                } else T[u][i] = T[fail[u]][i];
        }
    }

    int query(char str[], int lenstr) {
        int u = 0, ans = 0;
        for (int i = 1; i <= lenstr; i++) {
            int id = str[i] - 'a';
            u = T[u][id];
            for (int j = u; j && (~val[j]); j = fail[j]) {
                ans += val[j]; //val[j]=-1;
                if (pid[j]) {
                    qs[pid[j]].cnt++;
                }
            }
```

```
63
64          }
65          return ans;
66      }
67  } tree;
```

2.6 字典树/Trie 树

```
1   class Trie {
2   public:
3       int T[MAXN][26], val[MAXN], top;
4
5       Trie() {
6           top = 1;
7           memset(T[0], 0, sizeof(T[0]));
8           memset(val, 0, sizeof(val));
9       }
10
11      void insert(char str[], int lenstr) { // cal: scanf("%s", str+1), len =
            strlen(str+1), tree.insert(str, len);
12          int u = 0;
13          for (int i = 1; i <= lenstr; i++) {
14              int ch = str[i] - 'a';
15              if (!T[u][ch]) {
16                  memset(T[top], 0, sizeof(T[top]));
17                  T[u][ch] = top++;
18              }
19              u = T[u][ch];
20          }
21      }
22
23      int search(char str[], int lenstr) {
24          int u = 0;
25          for (int i = 1; i <= lenstr; i++) {
26              int ch = str[i] - 'a';
27              if (!T[u][ch])return -1; //找不到
28              u = T[u][ch];
29          }
30          if (!val[u]) {
31              val[u] = 1;
32              return 0;
33          }
34          return val[u];
```

```
    }
} tree;
```

2.7 后缀数组 SA

2.7.1 获取 SA 和 rank 数组

```
/*
    [input] aaabaabaaaab
    [output]
    rank    Suffix          pos(sa)     index
    1       aaaab           8           1
    2       aaab            9           2
    3       aaabaabaaab     1           3
    4       aab             10          4
    5       aabaaaab        5           5
    6       aabaabaaab      2           6
    7       ab              11          7
    8       abaaaab         6           8
    9       abaabaaaab      3           9
    10      b               12          10
    11      baaaab          7           11
    12      baabaaaab       4           12
*/
namespace SA {  // private ver.
    int len;
    int sa[MAXN], rk[MAXN << 1], oldrk[MAXN << 1], id[MAXN], cnt[MAXN];

    void run(char s[], int _len) {   // call: run(str, strlen(str+1));
        len = _len;
        int m = max(len, 300);
        // memset(cnt, 0, sizeof(cnt));
        for (int i = 0; i <= m; i++) cnt[i] = 0;
        for (int i = 1; i <= len; i++) ++cnt[rk[i] = s[i]];
        for (int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
        for (int i = len; i >= 1; i--) sa[cnt[rk[i]]--] = i;

        for (int w = 1; w <= len; w <<= 1) {
            // memset(cnt, 0, sizeof(cnt));
            for (int i = 0; i <= m; i++) cnt[i] = 0;
            for (int i = 1; i <= len; i++) id[i] = sa[i];
            for (int i = 1; i <= len; i++) ++cnt[rk[id[i] + w]];
            for (int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
            for (int i = len; i >= 1; i--) sa[cnt[rk[id[i] + w]]--] = id[i];
```

```
            for (int i = 0; i <= m; i++) cnt[i] = 0;
            // memset(cnt, 0, sizeof(cnt));
            for (int i = 1; i <= len; i++) id[i] = sa[i];
            for (int i = 1; i <= len; i++) ++cnt[rk[id[i]]];
            for (int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
            for (int i = len; i >= 1; i--) sa[cnt[rk[id[i]]]--] = id[i];
            // memcpy(oldrk, rk, sizeof(rk));
            for (int i = 0; i <= len; i++) oldrk[i] = rk[i];
            for (int p = 0, i = 1; i <= len; i++) {
                if (oldrk[sa[i]] == oldrk[sa[i - 1]] && oldrk[sa[i] + w] ==
                    oldrk[sa[i - 1] + w]) rk[sa[i]] = p;
                else rk[sa[i]] = ++p;
            }
        }
    }
}

namespace SA {  // 77 ver.
    int len;
    int sa[MAXN], rk[MAXN], oldrk[MAXN << 1], id[MAXN], cnt[MAXN], px[MAXN];

    bool cmp(int x, int y, int w) {
        return oldrk[x] == oldrk[y] && oldrk[x + w] == oldrk[y + w];
    }

    void run(char s[], int _len) {   // call: run(str, strlen(str+1));
        int i, m = 300, p, w;
        len = _len;
        // memset(cnt, 0, sizeof(cnt));
        for (i = 1; i <= m; i++) cnt[i] = 0;
        for (i = 1; i <= len; i++) ++cnt[rk[i] = s[i]];
        for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
        for (i = len; i >= 1; i--) sa[cnt[rk[i]]--] = i;

        for (w = 1; w <= len; w <<= 1, m = p) {
            for (p = 0, i = len; i > len - w; i--) id[++p] = i;
            for (i = 1; i <= len; i++)
                if (sa[i] > w) id[++p] = sa[i] - w;

            // memset(cnt, 0, sizeof(cnt));
            for (i = 0; i <= m; i++) cnt[i] = 0;
            for (i = 1; i <= len; i++) ++cnt[px[i] = rk[id[i]]];
            for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
            for (i = len; i >= 1; i--) sa[cnt[px[i]]--] = id[i];
```

```
            // memcpy(oldrk, rk, sizeof(rk));
            for (i = 0; i <= len; i++) oldrk[i] = rk[i];
            for (p = 0, i = 1; i <= len; i++) {
                rk[sa[i]] = cmp(sa[i], sa[i - 1], w) ? p : ++p;
            }
        }
    }
}
```

2.7.2 后缀数组 +ST 表求 lcp

```
int height[MAXN];
namespace SA {  // private ver.
    int len;
    int sa[MAXN], rk[MAXN << 1], oldrk[MAXN << 1], id[MAXN], cnt[MAXN];

    void run(char s[], int _len) {}   // step1 call: run(str, strlen(str+1));

    void get_height(char s[]) { // step2 call: get_height(str)
        int k = 0;
        for (int i = 1; i <= len; i++) rk[sa[i]] = i;
        for (int i = 1; i <= len; i++) {
            if (rk[i] == 1) continue;
            if (k) --k;
            int j = sa[rk[i] - 1];
            while (j + k <= len && i + k <= len && s[i + k] == s[j + k]) k++;
            height[rk[i]] = k;
        }
    }
}
const int MAXL = 22;
namespace RMQ { // ST, O(1) get LCP
    int mm[MAXN], best[MAXL][MAXN];

    void init(int n) {  // step3 call: init(strlen(str+1))
        mm[0] = -1;
        for (int i = 1; i <= n; i++)
            mm[i] = ((i & (i - 1)) == 0) ? mm[i - 1] + 1 : mm[i - 1];
        for (int i = 1; i <= n; i++)best[0][i] = i;
        for (int i = 1; i <= mm[n]; i++)
            for (int j = 1; j + (1 << i) - 1 <= n; j++) {
                int a = best[i - 1][j];
```

```
                int b = best[i - 1][j + (1 << (i - 1))];
                if (height[a] < height[b])best[i][j] = a;
                else best[i][j] = b;
            }
    }

    int askRMQ(int a, int b) {
        int t = mm[b - a + 1];
        b -= (1 << t) - 1;
        a = best[t][a];
        b = best[t][b];
        return height[a] < height[b] ? a : b;
    }

    /*
        get_SA.cpp example's index
        get_LCP(2, 4) => 2
    */
    int get_LCP(int a, int b) {
        if (a == b)return INF;
        if (a > b)swap(a, b);
        return height[askRMQ(a + 1, b)];
    }

}
```

2.8 后缀自动机 SAM

2.8.1 后缀自动机板子

应用 1：不同子串个数

给一个字符串 S，计算不同子串的个数。

解法：利用后缀自动机的树形结构。每个节点对应的不同子串数量 (不同位置算作同一个) 是 $maxlen[i] - maxlen[link[i]]$。

总时间复杂度：$O(|S|)$.

应用 2：所有不同子串的总长度

给定一个字符串 S，计算所有不同子串的总长度。解法：利用上述后缀自动机的树形结构。每个节点对应的所有后缀长度是 $\frac{maxlen[i]*(maxlen[i]+1)}{2}$，减去其 $linke$ 节点的对应值 $\frac{maxlen[link[i]]*(maxlen[link[i]]+1)}{2}$ 就是该节点的净贡献

```
class Suffix_Automaton {
public:
    int rt, link[MAXN], maxlen[MAXN], trans[MAXN][MAXC];
    int val[MAXN];  // 用于统计某一串出现的次数

```

```
    void init() {
        rt = 1;
        link[1] = maxlen[1] = 0;
        memset(trans[1], 0, sizeof(trans[1]));
    }

    Suffix_Automaton() { init(); }

    inline int insert(int ch, int last) {   // main: last = 1
        if (trans[last][ch]) {
            int p = last, x = trans[p][ch];
            // 注意：这里返回的这个节点保存了多个模式串的状态，
            // 即将多个不同模式串的相同子串信息压缩在了这一个节点内，
            // 如果要记录endpos大小的话，
            // 则需要给每个模式串都单独维护一个siz数组依次更新，
            // 而不能全部揉成一坨
            if (maxlen[p] + 1 == maxlen[x]) {   //
                特判1：这个节点已经存在于SAM中
                val[x]++;   // 统计在整颗字典树上出现次数
                return x;
            }
            else {
                int y = ++rt;
                maxlen[y] = maxlen[p] + 1;
                for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
                while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
                link[y] = link[x], link[x] = y;
                val[y]++;   // 统计在整颗字典树上出现次数
                return y;
            }
        }
        int z = ++rt, p = last;
        val[z] = 1; // 统计在整颗字典树上出现次数
        memset(trans[z], 0, sizeof(trans[z]));
        maxlen[z] = maxlen[last] + 1;
        while (p && !trans[p][ch]) trans[p][ch] = z, p = link[p];
        if (!p) link[z] = 1;
        else {
            int x = trans[p][ch];
            if (maxlen[p] + 1 == maxlen[x]) link[z] = x;
            else {
                int y = ++rt;
                maxlen[y] = maxlen[p] + 1;
                for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
```

```
                while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
                link[y] = link[x], link[z] = link[x] = y;
            }
        }
        ans_1 += maxlen[z] - maxlen[link[z]]; // 【应用1】统计不同字串个数
        return z;
    }

    struct Edge {
        int to, nex;
    } e[MAXN << 1];
    int head[MAXN], tol;

    void addEdge(int u, int v) {
        e[tol].to = v; e[tol].nex = head[u]; head[u] = tol; tol++;
    }


/*  统计出现次数为k的字串个数
    intput          output
    2               (输入组数)
    2               6
    abcabc
    3               9
    abcabcabcabc
*/
    ll ans = 0;
    void dfs(int u, int k) {
        for (int i = head[u]; ~i; i = e[i].nex) {
            int v = e[i].to;
            dfs(v, k);
            val[u] += val[v];
        }
        if (val[u] == k) {  // val为出现次数
            ans += 1ll * (maxlen[u] - maxlen[link[u]]); //
                以当前状态st为结尾的长度
        }
    }

    int build(int k) {
        tol = 0;
        for (int i = 0; i <= rt; i++) head[i] = -1;
        for (int i = 2; i <= rt; i++) addEdge(link[i], i);  // 建fail树
        dfs(1, k);
```

```
    }
} sa;
```

2.8.2 每个子串在多少个主串中出现过（SPOJ8093）

暴力跳 Link 链.

时间复杂度：均摊$O(\sum|S|\sqrt{\sum|S|})$

```
/*
    input       output
    3 3
    abcabcabc
    aaa
    aafe
    abc         1
    a           3
    ca          1
*/
class Suffix_Automaton {
public:
    int rt, link[MAXN], maxlen[MAXN], trans[MAXN][MAXC];
    int val[MAXN];

    void init() {
        rt = 1;
        link[1] = maxlen[1] = 0;
        memset(trans[1], 0, sizeof(trans[1]));
    }

    Suffix_Automaton() { init(); }

    inline int insert(int ch, int last) {   // main: last = 1
        if (trans[last][ch]) {
            int p = last, x = trans[p][ch];
            if (maxlen[p] + 1 == maxlen[x]) {   //
                特判1：这个节点已经存在于SAM中
                return x;
            } else {
                int y = ++rt;
                maxlen[y] = maxlen[p] + 1;
                for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
                while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
                link[y] = link[x], link[x] = y;
                return y;
```

```
            }
        }
        int z = ++rt, p = last;
        memset(trans[z], 0, sizeof(trans[z]));
        maxlen[z] = maxlen[last] + 1;
        while (p && !trans[p][ch]) trans[p][ch] = z, p = link[p];
        if (!p) link[z] = 1;
        else {
            int x = trans[p][ch];
            if (maxlen[p] + 1 == maxlen[x]) link[z] = x;
            else {
                int y = ++rt;
                maxlen[y] = maxlen[p] + 1;
                for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
                while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
                link[y] = link[x], link[z] = link[x] = y;
            }
        }
        return z;
    }
} sa;

char str_sum[MAXN];
char str[MAXN];
int len[MAXN];

int las[MAXN];  // 记得清空las数组

inline void update(int x, int id) { // 暴力跳Link链
    for (; x && las[x] != id; x = sa.link[x]) {
        sa.val[x]++;
        las[x] = id;
    }
}

int main() {
   int n, q;
   scanf("%d%d", &n, &q);
   int tot = 0;
   for (int i = 1; i <= n; i++) {
       scanf("%s", str + 1);
       int last = 1;
       len[i] = strlen(str + 1);
       for (int j = 1; j <= len[i]; j++) {
```

```
            str_sum[++tot] = str[j];
            last = sa.insert(str[j] - 'a', last);
        }
    }
    sa.debug();
    tot = 0;
    for (int i = 1; i <= n; i++) {
        for (int j = 1, x = 1; j <= len[i]; j++) {
            update(x = sa.trans[x][str_sum[++tot] - 'a'], i);
        }
    }
    while (q--) {
        scanf("%s", str + 1);
        len[0] = strlen(str + 1);
        int flag = 1, u = 1;
        for (int i = 1; i <= len[0]; i++) {
            int ch = str[i]- 'a';
            if (sa.trans[u][ch]) {
                u = sa.trans[u][ch];
            } else {
                flag = 0; break;
            }
        }
        if (flag) printf("%d\n", sa.val[u]);
        else printf("0\n");
    }
}
```

2.8.3 第 k 小字串

```
/*
    input   output
    aabc    aab
    0 3
    aabc    aa
    1 3
    aabc    -1
    1 11
*/
const int MAXN = 1e6 + 5;
const int MAXC = 26;

class Suffix_Automaton { public:
```

```
    int rt, link[MAXN], maxlen[MAXN], trans[MAXN][MAXC];
    int val[MAXN];

    void init() {
        rt = 1;
        link[1] = maxlen[1] = 0;
        memset(trans[0], 0, sizeof(trans[0]));
    }
    Suffix_Automaton() { init(); }

    inline int insert(int ch, int last) {   // main: last = 1
        if (trans[last][ch]) {
            int p = last, x = trans[p][ch];
            if (maxlen[p] + 1 == maxlen[x]) return x;
            else {
                int y = ++rt;
                maxlen[y] = maxlen[p] + 1;
                for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
                while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
                link[y] = link[x], link[x] = y;
                return y;
            }
        }
        int z = ++rt, p = last;
        val[z] = 1; // dfs树统计出现次数
        maxlen[z] = maxlen[last] + 1;
        while (p && !trans[p][ch]) trans[p][ch] = z, p = link[p];
        if (!p) link[z] = 1;
        else {
            int x = trans[p][ch];
            if (maxlen[p] + 1 == maxlen[x]) link[z] = x;
            else {
                int y = ++rt;
                maxlen[y] = maxlen[p] + 1;
                for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
                while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
                link[y] = link[x], link[z] = link[x] = y;
            }
        }
        return z;
    }

    int sz1[MAXN], sz2[MAXN];
    int topo[MAXN], topo_id[MAXN];
```

```
58
59      // Graph
60      struct Edge {
61          int to, nex;
62      } e[MAXN];
63      int head[MAXN], tol;
64
65      void addEdge(int u, int v) {
66          e[tol].to = v;
67          e[tol].nex = head[u];
68          head[u] = tol;
69          tol++;
70      }
71
72      void dfs(int u) {
73          for (int i = head[u]; ~i; i = e[i].nex) {
74              int v = e[i].to;
75              dfs(v);
76              val[u] += val[v];
77          }
78      }
79
80      void build() {
81          // get topo index
82          for (int i = 1; i <= rt; i++) topo[maxlen[i]]++;
83          for (int i = 1; i <= rt; i++) topo[i] += topo[i - 1];
84          for (int i = 1; i <= rt; i++) topo_id[topo[maxlen[i]]--] = i;
85          // when t = 0
86          for (int i = rt; i >= 1; i--) {
87              sz1[topo_id[i]] = 1;
88              if (topo_id[i] == 1) sz1[topo_id[i]] = 0;
89              for (int j = 0; j < MAXC; j++) {
90                  int v = trans[topo_id[i]][j];
91                  if (!v) continue;
92                  sz1[topo_id[i]] += sz1[v];
93              }
94          }
95          // fail tree build begin
96          for (int i = 0; i <= rt; i++) head[i] = -1;
97          tol = 0;
98          for (int i = 2; i <= rt; i++) addEdge(link[i], i);
99          // fail tree build end
100         dfs(1); // dfs val
101         // when t = 1
```

```
        for (int i = rt; i >= 1; i--) {
            sz2[topo_id[i]] = val[topo_id[i]];
            if (topo_id[i] == 1) sz2[topo_id[i]] = 0;
            for (int j = 0; j < MAXC; j++) {
                int v = trans[topo_id[i]][j];
                if (!v) continue;
                sz2[topo_id[i]] += sz2[v];
            }
        }
    }

    void query0(int k) {    // 不同位置的相同子串算作一个
        if (sz1[1] < k) {
            printf("-1\n"); return ;
        }
        int u = 1;
        while (k) {
            for (int i = 0; i < MAXC; i++) {
                if (trans[u][i]) {
                    if (sz1[trans[u][i]] >= k) {
                        printf("%c", 'a' + i);
                        u = trans[u][i];
                        k--;
                        break;
                    } else k -= sz1[trans[u][i]];
                }
            }
        }
        printf("\n");
    }

    void query1(int k) {    // 不同位置的相同子串算作多个
        if (sz2[1] < k) {
            printf("-1\n"); return ;
        }
        int u = 1;
        while(k > 0) {
            for (int i = 0; i < MAXC; i++) {
                if (trans[u][i]) {
                    int v = trans[u][i];
                    if (sz2[v] >= k) {
                        printf("%c", 'a'+i);
                        u = v;
                        k -= val[v];
```

```
                        break;
                    } else k -= sz2[v];
                }
            }
        }
        printf("\n");
    }
} sa;
```

2.8.4 字典树建后缀自动机

```
void bfs() {    // 传说常数小
    queue<node> q;
    q.push(node(1, 1));
    int last = 1;
    while (!q.empty()) {
        node u = q.front(); q.pop();
        int nls = sa.insert(str[u.v]-'A', u.last);
        pos[u.v] = nls;
        for (int i = head[u.v]; ~i; i = e[i].nex) {
            int to = e[i].to;
            q.push(node(to, nls));
        }
    }
}

void dfs(int u, int last = 1) { // 传说常数大
    pos[u] = last = sa.insert(str[u] - 'A', last);
    for (int i = head[u]; ~i; i = e[i].nex) {
        int v = e[i].to;
        dfs(v, last);
    }
}
```

2.8.5 暴力在线统计出现次数为 k 次的字符串个数（HDU4641）

```
const int MAXN = 6e5 + 5;
const int MAXC = 26;
int K;
class Suffix_Automaton {
public:
    int rt, link[MAXN], maxlen[MAXN], trans[MAXN][MAXC];
    int val[MAXN];
```

```
    int ans = 0;
    void init() {
        rt = 1;
        link[1] = maxlen[1] = 0;
        memset(trans[1], 0, sizeof(trans[1]));
        ans = 0;
    }

    Suffix_Automaton() { init(); }

    inline int insert(int ch, int last) {   // main: last = 1
        if (trans[last][ch]) {
            int p = last, x = trans[p][ch];
            if (maxlen[p] + 1 == maxlen[x]) return x;
            else {
                int y = ++rt;
                maxlen[y] = maxlen[p] + 1;
                val[y] = val[x];
                for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
                while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
                link[y] = link[x], link[x] = y;
                return y;
            }
        }
        int z = ++rt, p = last;
        val[z]=0;
        memset(trans[z], 0, sizeof(trans[z]));
        maxlen[z] = maxlen[last] + 1;
        while (p && !trans[p][ch]) trans[p][ch] = z, p = link[p];
        if (!p) link[z] = 1;
        else {
            int x = trans[p][ch];
            if (maxlen[p] + 1 == maxlen[x]) link[z] = x;
            else {
                int y = ++rt;
                maxlen[y] = maxlen[p] + 1;
                val[y] = val[x];
                for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
                while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
                link[y] = link[x], link[z] = link[x] = y;
            }
        }
        int t = z;
        while (t && val[t] < K) {
```

```
            val[t]++;
            if (val[t] >= K) ans+=maxlen[t] - maxlen[link[t]];
            t = link[t];
        }
        return z;
    }

} sa;
char str[MAXN];
int main() {
    int n, m;
    while(~scanf("%d%d%d", &n, &m, &K)) {
        sa.init();
        scanf("%s", str + 1);
        int last = 1;
        for (int i = 1; i <= n; i++) last = sa.insert(str[i] - 'a', last);
        while (m--) {
            int op;
            scanf("%d", &op);
            if (op == 1) {
                getchar();
                char ch;
                scanf("%c", &ch);
                last = sa.insert(ch - 'a', last);
            } else {
                printf("%d\n", sa.ans);
            }
        }
    }
}
```

2.9 回文自动机 PAM

```
int pos[MAXN];  // +1后对应为所建的fail树上的点
class Palindrome_Tree { public:
    struct node {
        int ch[MAXC], fail, len, num;   // num：以该位置结尾的回文子串个数
    } T[MAXN];
    int las, tot, c[MAXN];
    inline int get_fail(int x, int pos) {
        while (c[pos - T[x].len - 1] != c[pos]) {
            x = T[x].fail;
        }
```

```
        return x;
    }
    void init() { // 传入字符串长度
        memset(T[0].ch, 0, sizeof(T[0].ch));
        memset(T[1].ch, 0, sizeof(T[1].ch));
        T[0].len = 0, T[1].len = -1;
        T[0].fail = 1, T[1].fail = 0;
        las = 0, tot = 1;
    }
    void insert(char s[], int len) {  // call: insert(str, strlen(str+1));
        c[0] = -1;
        for (int i = 1; i <= len; i++) {
            c[i] = s[i] - 'a';
            int p = get_fail(las, i);
            if (!T[p].ch[c[i]]) {
                T[++tot].len = T[p].len + 2;
                memset(T[tot].ch, 0, sizeof(T[tot].ch));
                int u = get_fail(T[p].fail, i);
                T[tot].fail = T[u].ch[c[i]];
                T[tol].num = T[T[tol].fail].num + 1;
                T[p].ch[c[i]] = tot;
            }
            las = T[p].ch[c[i]];
            pos[i] = las;
        }
    }
    struct Edge {
        int to, nex;
    } e[MAXN << 1];
    int head[MAXN], tol;
    int len[MAXN];
    void addEdge(int u, int v) {
        e[tol].to = v, e[tol].nex = head[u], head[u] = tol, tol++;
    }
    void build() { // build tree
        tol = 0;
        for (int i = 0; i <= tot; i++) head[i+1] = -1;
        for (int i = 1; i <= tot; i++) addEdge(T[i].fail + 1, i + 1);
        for (int i = 0; i <= tot; i++) len[i + 1] = T[i].len;
    }
} tree;

char str[MAXN];
int main() {
```

```
    scanf("%s", str + 1);
    tree.init();
    tree.insert(str, strlen(str + 1));
    tree.build();
    for (int i = 1; i <= len; i++) printf("%d ", pos[i]+1); // fail树上对应位置
}
```

2.10 序列自动机（[HEOI2015] 最短不公共子串）

时间复杂度：$O(n|\sum|)$，其中 $|\sum|$ 为字符集大小

```
/*
    input       output
    aabbcc      2 4 2 4
    abcabc
    aabbcc      -1 -1 2 -1
    aabbcc
*/
char str1[MAXN], str2[MAXN];
int nex[MAXC], na[MAXN][MAXC], nb[MAXN][MAXC];
int dp[MAXN][MAXN]; // 记得开大两倍
int main() {
    scanf("%s%s", str1 + 1, str2 + 1);
    int len1 = strlen(str1 + 1), len2 = strlen(str2 + 1);
    int last = 1;
    for (int i = 1; i <= len2; i++) last = sa.insert(str2[i] - 'a', last); // 调用后缀自动机
    for (int i = 0; i < 26; i++) nex[i] = len1 + 1;
    for (int i = len1; i >= 0; i--) {
        memcpy(na[i], nex, sizeof(nex));
        nex[str1[i] - 'a'] = i;
    }
    for (int i = 0; i < 26; i++) nex[i] = len2 + 1;
    for (int i = len2; i >= 0; i--) {
        memcpy(nb[i], nex, sizeof(nex));
        nex[str2[i] - 'a'] = i;
    }
    int res = MAXN;
    for (int l = 1; l <= len1; l++) {
        for (int r = l, u = 1; r <= len1; r++) {
            u = sa.trans[u][str1[r] - 'a'];
            if (!u) {
                res = min(res, r - l + 1);
```

```
                break;
            }
        }
    }
    printf("%d\n", res == MAXN ? -1 : res); // str1的一个最短的子串，它不是str2的子串。
    res = MAXN;
    for (int l = 1; l <= len1; l++) {
        for (int r = l, u = 0; r <= len1; r++) {
            u = nb[u][str1[r] - 'a'];
            if (u == len2 + 1) {
                res = min(res, r - l + 1);
                break;
            }
        }
    }
    printf("%d\n", res == MAXN ? -1 : res); // str1的一个最短的子串，它不是str2的子序列。
    for (int i = len1; i >= 0; i--) {
        for (int j = 1; j <= sa.rt; j++) {
            dp[i][j] = MAXN;
            for (int ch = 0; ch < MAXC; ch++) {
                int u = na[i][ch], v = sa.trans[j][ch];
                if (u <= len1) dp[i][j] = min(dp[i][j], dp[u][v] + 1);
            }
        }
    }
    printf("%d\n", dp[0][1] == MAXN ? -1 : dp[0][1]);   // str1的一个最短的子序列，它不是str2的子串。
    memset(dp, 0, sizeof(dp));
    for (int i = len1; i >= 0; i--) {
        for (int j = 0; j <= len2; j++) {
            dp[i][j] = MAXN;
            for (int ch = 0; ch < MAXC; ch++) {
                int u = na[i][ch], v = nb[j][ch];
                if (u <= len1) dp[i][j] = min(dp[i][j], dp[u][v] + 1);
            }
        }
    }
    printf("%d\n", dp[0][0] == MAXN ? -1 : dp[0][0]);   // str1的一个最短的子序列，它不是str2的子序列。
}
```

2.11 最小表示法

时间复杂度：$O(n)$

```
/*
    input
    10
    10 9 8 7 6 5 4 3 2 1
    output
    1 10 9 8 7 6 5 4 3 2
*/
int Min_show(int arr[], int n) {
    int i = 0, j = 1, k = 0;
    while (i < n && j < n && k < n) {
        if (arr[(i + k) % n] == arr[(j + k) % n]) k++;
        else {
            if (arr[(i + k) % n] > arr[(j + k) % n]) i += k + 1;
            else j += k + 1;
            if (i == j)i++;
            k = 0;
        }
    }
    return min(i, j);
}
int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%d", &a[i]);
    int ans = Min_show(a,n);
    for (int i = 0; i < n; i++) printf("%d ", a[(i + ans) % n]);
}
```

2.12 Lyndon 分解

将字符串分成若干部分 $s = s_1 s_2 s_3 ... s_m$，使得每个 s_i 都是 $LyndonWord$。

$LyndonWord$：当且仅当 s 是其所有后缀中最小字符串。

```
/*
    input                   output
    ababa                   2 4 5
    ababa = ab + ab + a
    bbababaabaaabaaaab  1 2 4 6 9 13 18
    bbababaabaaabaaaab
    = b + b + ab + ab + aab + aaaab + aaaab
*/
```

```cpp
vector<int> Lyndon_Arr; // 分解后的串的右端点
// call: Lyndon_Word(str, strlen(str+1));
void Lyndon_Word(char s[], int s_len) {
    int i = 1;
    while (i <= s_len) {
        int j = i, k = j + 1;
        while (k <= s_len && s[j] <= s[k]) {
            if (s[j] < s[k]) j = i;
            else j++;
            k++;
        }
        while (i <= j) {
            Lyndon_Arr.push_back(i-j+k-1);
            i += k - j;
        }
    }
}
```

3 杂项

3.1 LCA

```cpp
const int MAXLOG = 22;
struct Edge {
    int to, nex;
} e[MAXM << 1];
int head[MAXN], tol;
void addEdge(int u, int v) {
    e[tol].to = v, e[tol].nex = head[u], head[u] = tol, tol++;
}
namespace LCA {
    int dep[MAXN], fa[MAXN][MAXLOG], lg[MAXN];
    void init(int _n) { // n为点的个数，最坏情况为一条链
        for (int i = 1; i <= _n; i++) {
            lg[i] = lg[i-1] + (1 << lg[i-1] == i);
        }
    }
    void dfs(int u, int f) {
        fa[u][0] = f; dep[u] = dep[f] + 1;
        for (int i = 1; i <= lg[dep[u]]; i++) fa[u][i] = fa[fa[u][i-1]][i-1];
        for (int i = head[u]; ~i; i = e[i].nex) {
            int v = e[i].to;
            if (v == f) continue;
```

```
            dfs(v, u);
        }
    }
    int LCA(int u, int v) {
        if (dep[u] < dep[v]) swap(u, v);
        while (dep[u] > dep[v]) u = fa[u][lg[dep[u] - dep[v]] - 1];
        if (u == v) return u;
        for (int k = lg[dep[u]] - 1; k >= 0; k--) {
            if (fa[u][k] != fa[v][k]) u = fa[u][k], v = fa[v][k];
        }
        return fa[u][0];
    }
}
```

3.2 CDQ

3.2.1 三维偏序

有 n 个元素，第 i 个元素有 a_i,b_i,c_i 三个属性，设 $f(i)$ 表示满足 $a_j \le a_i$ 且 $b_j \le b_i$ 且 $c_j \le c_i$ 且 $j \ne i$ 的 j 的数量。

对于 $d \in [0,n)$，求 $f(i) = d$ 的数量。

```
class TREE { public:
    int T[MAXN], n;
    inline int lowbit(int x) { return x & (-x); }
    void add(int pos, int val) {
        while (pos <= n) {
            T[pos] += val; pos += lowbit(pos);
        }
    }
    int query(int pos) {
        int ans = 0;
        while (pos) {
            ans += T[pos]; pos -= lowbit(pos);
        }
        return ans;
    }
} tree;

struct node {
    int a, b, c;
    int cnt, ans;
    bool operator==(const node &tb) const {
        if (a == tb.a && b == tb.b && c == tb.c) return 1;
        else return 0;
```

```
    }
} p1[MAXN], p2[MAXN];

void cdq(int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    cdq(l, mid), cdq(mid + 1, r);
    sort(p2 + l, p2 + mid + 1, [&](const node &x, const node &y) {
        if (x.b != y.b) return x.b < y.b;
        else return x.c < y.c;
    });
    sort(p2 + mid + 1, p2 + r + 1, [&](const node &x, const node &y) {
        if (x.b != y.b) return x.b < y.b;
        else return x.c < y.c;
    });
    int j = l;
    for (int i = mid + 1; i <= r; i++) {
        while (p2[i].b >= p2[j].b && j <= mid) {
            tree.add(p2[j].c, p2[j].cnt);
            j++;
        }
        p2[i].ans += tree.query(p2[i].c);
    }
    for (int i = l; i < j; i++) tree.add(p2[i].c, -p2[i].cnt);
}

int res[MAXN];
int main() {
    int n, k; scanf("%d%d", &n, &k);
    for (int i = 1; i <= n; i++) scanf("%d%d%d", &p1[i].a, &p1[i].b,
       &p1[i].c);
    p1[n+1].a = p1[n+1].b = p1[n+1].c = 0;
    sort(p1 + 1, p1 + 1 + n, [&](const node &x, const node &y) {
        if (x.a != y.a) return x.a < y.a;
        if (x.b != y.b) return x.b < y.b;
        return x.c < y.c;
    });
    int tot = 0, m = 0;
    for (int i = 1; i <= n; i++) {
        tot++;
        if (p1[i].a != p1[i + 1].a || p1[i].b != p1[i + 1].b || p1[i].c !=
           p1[i + 1].c) {
            m++;
            p2[m].a = p1[i].a, p2[m].b = p1[i].b, p2[m].c = p1[i].c,
```

```
                p2[m].cnt = tot;
            tot = 0;
        }
    }
    tree.n = k;
    cdq(1, m);
    for (int i = 1; i <= m; i++) res[p2[i].ans + p2[i].cnt -1] += p2[i].cnt;
    for (int d = 0; d < n; d++) printf("%d\n", res[d]); // 输出每个f(i) =
        d的数量
}
```