

# 目录

<b>1 STL</b>	<b>1</b>
1.1 unordered_map 对 pii 进行哈希	1
1.2 pb_ds 实现平衡树	2
1.3 rope	3
1.4 ST 表	4
1.5 笛卡尔树	5
1.5.1 建树	5
1.6 线段树	5
1.6.1 区间中所有元素都严格出现三次的区间个数 (CF1418G)	5
1.6.2 线段树分裂合并	8
1.6.3 单点修改 + 单点最大连通数 (HDU1540)	11
1.7 可持久化线段树 (主席树)	12
1.7.1 静态区间第 K 小	12
1.7.2 区间内不同数个数	13
1.7.3 静态区间第 K 小	14
1.7.4 区间 MEX	16
1.8 无旋 Treap (FHQ Treap)	17
1.8.1 区间翻转	17
1.8.2 可持久化 FHQ	20
1.9 树套树	23
1.9.1 带修主席树	23
1.9.2 区间修改区间查询第 K 大 ([ZJOI2013]K 大数查询)	25
1.10 KD 树	28
1.10.1 平面最近点对	28
1.10.2 K 远点对 ([CQOI2016])	30
1.10.3 高维空间上的操作	32
1.11 珂朵莉树/老司机树/ODT	35
1.11.1 set 实现珂朵莉树	35
<b>2 字符串</b>	<b>38</b>
2.1 字符串哈希	38
2.1.1 区间一维哈希	38
2.2 Next 函数	39
2.2.1 求 next 函数	39
2.2.2 求出每个循环节的数量和终点位置 (HDU1358)	39
2.2.3 求同时是前缀和后缀的串长 (POJ2752)	40
2.2.4 求字符串每个前缀和串匹配成功的次数和 (HDU3336)	40
2.2.5 求循环节数量 (POJ2406)	41
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)	42
2.2.7 求补上最少字母数量使得这是个循环串 (HDU3746)	42
2.2.8 习题整理	43
2.3 KMP	44

2.3.1	统计模式串出现次数, 出现位置, 前缀 <b>border</b> 长度	44
2.3.2	矩阵加速 <b>KMP</b> , 求长度为 <b>n</b> 的不包含长度为 <b>m</b> 的子串的串个数 ([HNOI2008]GT 考试)	44
2.4	<b>EXKMP</b>	45
2.4.1	求 <b>z</b> 函数和 <b>LCP</b>	45
2.4.2	循环位移有多少数比原数大小相等, 去重 (HDU4333)	46
2.5	<b>AC</b> 自动机	47
2.5.1	标准的 <b>AC</b> 自动机	47
2.6	字典树/Trie 树	49
2.7	后缀数组 <b>SA</b>	50
2.7.1	获取 <b>SA</b> 和 <b>rank</b> 数组	50
2.7.2	后缀数组 + <b>ST</b> 表求 <b>lcp</b>	52
2.8	后缀自动机 <b>SAM</b>	53
2.8.1	后缀自动机板子	53
2.8.2	每个子串在多少个主串中出现过 (SPOJ8093)	56
2.8.3	第 <b>k</b> 小字串	58
2.8.4	字典树建后缀自动机	62
2.8.5	暴力在线统计出现次数为 <b>k</b> 次的字符串个数 (HDU4641)	62
2.9	回文自动机 <b>PAM</b>	64
2.10	序列自动机 ([HEOI2015] 最短不公共子串)	66
2.11	最小表示法	68
2.12	Lyndon 分解	68
3	杂项	69
3.1	<b>LCA</b>	69
3.2	<b>CDQ</b>	70
3.2.1	三维偏序	70

# 1 STL

## 1.1 unordered\_map 对 pii 进行哈希

```
1 // call: unordered_map<pii, int, pair_hash> ma;
2 // 哈希方法1（较快）
3 template<typename T>
4 inline void hash_combine(std::size_t &seed, const T &val) {
5     seed ^= std::hash<T>()(val) + 0x9e3779b9 + (seed << 6) + (seed >> 2);
6 }
7
8 template<typename T>
9 inline void hash_val(std::size_t &seed, const T &val) {
10     hash_combine(seed, val);
11 }
12
13 template<typename T, typename... Types>
14 inline void hash_val(std::size_t &seed, const T &val, const Types &... args) {
15     hash_combine(seed, val);
16     hash_val(seed, args...);
17 }
18
19 template<typename... Types>
20 inline std::size_t hash_val(const Types &... args) {
21     std::size_t seed = 0;
22     hash_val(seed, args...);
23     return seed;
24 }
25
26 struct pair_hash {
27     template<class T1, class T2>
28     std::size_t operator()(const std::pair<T1, T2> &p) const {
29         return hash_val(p.first, p.second);
30     }
31 };
32
33 // 哈希方法2（部分冲不过）
34 struct pair_hash {
35     template<class T1, class T2>
36     std::size_t operator()(const std::pair<T1, T2> &p) const {
37         auto h1 = std::hash<T1>{}(p.first);
38         auto h2 = std::hash<T2>{}(p.second);
39         return h1 ^ h2;
40     }
41 }
```

41 |};

## 1.2 pb\_ds 实现平衡树

```
1 #define _EXT_CODECVT_SPECIALIZATIONS_H 1
2 #define _EXT_ENC_FILEBUF_H 1
3 #undef __MINGW32__
4
5 #include<bits/stdc++.h>
6 #include<bits/extc++.h>
7 using namespace std;
8 using namespace __gnu_pbds;
9 typedef long long ll;
10
11 class PBDS {
12 private:
13     typedef tree<ll, null_type, less<ll>, rb_tree_tag,
14         tree_order_statistics_node_update> Tree;
15     Tree T;
16 public:
17     void insert(ll x, int id) { // 插入x数, 需传入第几次操作进行简单处理
18         T.insert((x << 20) + id); // 简单的处理
19     }
20
21     void remove(ll x) { // 删除x数(若有多个相同的数, 因只删除一个)
22         T.erase(T.lower_bound(x << 20));
23     }
24
25     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)
26         return T.order_of_key(x << 20) + 1;
27     }
28
29     ll get_val(ll x) { // 查询排名为x的数
30         return *T.find_by_order(x - 1) >> 20;
31     }
32
33     ll get_pre(ll x) { // 求x的前驱(前驱定义为小于x, 且最大的数)
34         return *--T.lower_bound(x << 20) >> 20;
35     }
36
37     ll get_next(ll x) { // 求x的后继(后继定义为大于x, 且最小的数)
38         return *T.lower_bound((x + 1) << 20) >> 20;
39     }
40 }
```

```
39 } 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

```
1 #include <bits/stdc++.h>
2 #include <bits/extc++.h>
3
4 using namespace std;
5 using namespace __gnu_cxx;
6 rope<char> tree;
7 inline void read_str(char *s, int len) { // 读入长度为len的字符串
8     s[len] = '\0';
9     len--;
10    for (int i = 0; i <= len; i++) {
11        s[i] = '\0';
12        while (s[i] < 32 || 126 < s[i]) s[i] = getchar();
13    }
14 }
15
16 inline void read_int(int &x) {
17     x = 0;
18     char ch;
19     while (!isdigit(ch = getchar()));
20     x = ch - '0';
21     while (isdigit(ch = getchar())) x = x * 10 + ch - '0';
22 }
23 const int MAXN = 2e6 + 5;
24 char word[MAXN];
25
26 int main() {
27     int T;
28     int now = 0;
29     scanf("%d", &T);
30     while (T--) {
31         int opt = '1', x;
32         while (!isalpha(opt = getchar()));
33         while (isalpha(getchar()));
34         // 格式: Move k
35         // 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
36         if (opt == 'M') read_int(now);
37         // Insert n s
38         // 在光标处插入长度为n的字符串s, 光标位置不变n>=1
```

```

39     else if (opt == 'I') {
40         read_int(x);
41         read_str(word, x);
42         tree.insert(now, word);
43     }
44     // Delete n
45     // 删除光标后的n个字符，光标位置不变，n>=1
46     else if (opt == 'D') {
47         read_int(x);
48         tree.erase(now, x);
49     }
50     // Get n
51     // 输出光标后的n个字符，光标位置不变，n>=1
52     else if (opt == 'G') {
53         read_int(x);
54         x--;
55         for (int i = now; i <= now + x; i++) printf("%c", tree[i]);
56         printf("\n");
57     } else if (opt == 'P') now--; // 光标前移
58     else now++; // 光标后移
59 }
60 }

```

## 1.4 ST 表

```

1  const int MAXN = 5e4 + 5; // n
2  const int MAXL = 22;
3  int f[MAXN][MAXL], g[MAXN][MAXL], two[MAXN];
4  int h[MAXN];
5  void pre(int n) {
6      for (int i = 1; i <= n; i++) {
7          f[i][0] = h[i]; g[i][0] = h[i]; // 读入h
8      }
9      two[1] = 0; two[2] = 1;
10     for (int i = 3; i < MAXN; i++) {
11         two[i] = two[i / 2] + 1;
12     }
13     for (int j = 1; j <= MAXL; j++) {
14         for (int i = 1; i + (1 << j) - 1 <= n; i++)
15             f[i][j] = max(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
16     }
17     for (int j = 1; j <= MAXL; j++) {
18         for (int i = 1; i + (1 << j) - 1 <= n; i++)

```

```

19         g[i][j] = min(g[i][j - 1], g[i + (1 << (j - 1))][j - 1]);
20     }
21 }
22
23 int get_max(int x, int y) {
24     int s = two[y - x + 1];
25     return max(f[x][s], f[y - (1 << s) + 1][s]);
26 }
27
28 int get_min(int x, int y) {
29     int s = two[y - x + 1];
30     return min(g[x][s], g[y - (1 << s) + 1][s]);
31 }

```

## 1.5 笛卡尔树

### 1.5.1 建树

```

1 // p[i] 存放 [1, n] 数据
2 class cartesian {
3     int stk[MAXN], lson[MAXN], rson[MAXN];
4     void build() {
5         int tmp = 0, x;
6         for (int i = 1; i <= n; i++) {
7             x = tmp;
8             while (x && p[stk[x]] > p[i]) x--;
9             if (x) rson[stk[x]] = i;
10            if (x < tmp) lson[i] = stk[x + 1];
11            stk[++x] = i;
12            tmp = x;
13        }
14    }
15 } ;

```

## 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)

```

1 /*
2     input                output
3     9                    3
4     1 2 2 2 1 1 2 2 2
5     10                   0

```

```

6      1 2 3 4 1 2 3 1 2 3
7      12                      1
8      1 2 3 4 3 4 2 1 3 4 2 1
9  */
10 int arr[MAXN];
11 vector<int> vec[MAXN];
12 struct node {
13     int l, r, v;
14     node(int _l = 0, int _r = 0, int _v = 0) : l(_l), r(_r), v(_v) {}
15 };
16
17 class SEG { public:
18     struct node {
19         int l, r, mx, cnt;
20     } T[MAXN << 2];
21     int lazy[MAXN << 2];
22
23     void build(int rt, int l, int r) {
24         T[rt].l = l, T[rt].r = r, T[rt].mx = 0, T[rt].cnt = r - l;
25         if (l + 1 == r) {
26             T[rt << 1].l = T[rt << 1].r = T[rt << 1].mx = T[rt << 1].cnt = 0;
27             T[rt << 1 | 1].l = T[rt << 1 | 1].r = T[rt << 1 | 1].mx = T[rt <<
                1 | 1].cnt = 0;
28             return;
29         }
30         int mid = (l + r) >> 1;
31         build(rt << 1, l, mid);
32         build(rt << 1 | 1, mid, r);
33     }
34
35     inline void push_up(int rt) {
36         if (T[rt << 1].mx > T[rt << 1 | 1].mx) T[rt].mx = T[rt << 1].mx,
            T[rt].cnt = T[rt << 1].cnt;
37         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;
38         else T[rt].mx = T[rt << 1].mx, T[rt].cnt = T[rt << 1].cnt + T[rt << 1
            | 1].cnt;
39     }
40
41     inline void push_down(int rt) {
42         if (lazy[rt]) {
43             T[rt << 1].mx += lazy[rt], lazy[rt << 1] += lazy[rt];
44             T[rt << 1 | 1].mx += lazy[rt], lazy[rt << 1 | 1] += lazy[rt];
45             lazy[rt] = 0;

```



```

46     }
47 }
48
49 void update(int rt, int L, int R, int val) {
50     if (L <= T[rt].l && R >= T[rt].r) {
51         T[rt].mx += val, lazy[rt] += val;
52         return;
53     }
54     if (L >= T[rt].r || R <= T[rt].l) return ;
55     push_down(rt);
56     int mid = (T[rt].l + T[rt].r) >> 1;
57     if (L <= mid) update(rt << 1, L, R, val);
58     if (R >= mid) update(rt << 1 | 1, L, R, val);
59     push_up(rt);
60 }
61 } tree;
62
63 vector<node> event[MAXN];
64 void add(int l1, int l2, int r1, int r2) {
65     event[l1].push_back(node(r1, r2, 1));
66     event[l2].push_back(node(r1, r2, -1));
67 }
68
69 int main() {
70     int n;
71     scanf("%d", &n);
72     for (int i = 1; i <= n; i++) {
73         scanf("%d", &arr[i]);
74         vec[arr[i]].push_back(i);
75     }
76     for (int x = 1; x <= n; x++) {
77         int sz = SZ(vec[x]);
78         for (int i = 0; i <= sz; i++) {
79             add(i == 0 ? 1 : vec[x][i - 1] + 1, i == sz ? n + 2 : vec[x][i] +
1,
80             i == 0 ? 1 : vec[x][i - 1] + 1, i == sz ? n + 2 : vec[x][i] +
1);
81         }
82         for (int i = 0; i <= sz - 3; i++) {
83             add(i == 0 ? 1 : vec[x][i - 1] + 1, vec[x][i] + 1,
84             vec[x][i + 2] + 1, i + 3 == sz ? n + 2 : vec[x][i + 3] + 1);
85         }
86     }
87     tree.build(1, 1, n + 2);

```

```

88     ll res = 0;
89     for (int l = 1; l <= n; l++) { // enum left
90         for (auto e: event[l]) {
91             int el = e.l, er = e.r, ev = e.v;
92             tree.update(1, el, er, ev);
93         }
94         tree.update(1, l, l+1, -1);
95         if (tree.T[1].mx == n) {
96             res += tree.T[1].cnt;
97         }
98     }
99     printf("%lld\n", res);
100 }

```

### 1.6.2 线段树分裂合并

```

1  class SEG { public:
2      int ch[MAXM][2];
3      ll sum[MAXM];
4      #define lson ch[rt][0]
5      #define rson ch[rt][1]
6      int pool[MAXM], pool_cnt, cnt;
7      int New() { // recycle
8          return pool_cnt ? pool[pool_cnt-1] : ++cnt;
9      }
10     void Del(int rt) {
11         pool[++pool_cnt] = rt;
12         ch[rt][0] = ch[rt][1] = sum[rt] = 0;
13     }
14     inline void push_up(int rt) {
15         sum[rt] = sum[lson] + sum[rson];
16     }
17     int change(int rt, int pos, int v, int be, int en) {
18         if (!rt) rt = New();
19         if (be == en) {
20             sum[rt] += (ll) v;
21             return rt;
22         }
23         int mid = (be + en) >> 1;
24         if (pos <= mid) lson = change(lson, pos, v, be, mid);
25         else
26             rson = change(rson, pos, v, mid + 1, en);
27         push_up(rt);

```

```

28     return rt;
29 }
30 ll query_sum(int rt, int L, int R, int be, int en) {
31     if (!rt) return 0;
32     if (L <= be && R >= en) return sum[rt];
33     int mid = (be + en) >> 1;
34     ll ans = 0;
35     if (L <= mid) ans += query_sum(lson, L, R, be, mid);
36     if (R > mid) ans += query_sum(rson, L, R, mid + 1, en);
37     return ans;
38 }
39 int Kth(int rt, int be, int en, int k) {
40     if (be == en) return be;
41     int mid = (be + en) >> 1;
42     int ans = -1;
43     if (sum[ch[rt][0]] >= k) ans = Kth(lson, be, mid, k);
44     else ans = Kth(rson, mid + 1, en, k - sum[ch[rt][0]]);
45     return ans;
46 }
47 int merge(int x, int y, int be, int en) {
48     if (!x || !y) return x + y;
49     if (be == en) {
50         sum[x] += sum[y];
51         return x;
52     }
53     int mid = (be + en) >> 1;
54     ch[x][0] = merge(ch[x][0], ch[y][0], be, mid);
55     ch[x][1] = merge(ch[x][1], ch[y][1], mid + 1, en);
56     push_up(x);
57     Del(y);
58     return x;
59 }
60 void split(int &rt1, int &rt2, int L, int R, int be, int en) {
61     if (en < L || R < be) return;
62     if (!rt1) return;
63     if (L <= be && en <= R) {
64         rt2 = rt1, rt1 = 0;
65         return;
66     }
67     if (!rt2) rt2 = New();
68     int mid = (be + en) >> 1;
69     split(ch[rt1][0], ch[rt2][0], L, R, be, mid);
70     split(ch[rt1][1], ch[rt2][1], L, R, mid + 1, en);
71     push_up(rt1), push_up(rt2);

```

```

72     }
73 } tree;
74
75 int root[MAXN];
76 int main() {
77     int n, m;
78     scanf("%d%d", &n, &m);
79     int max_n_m = max(n, m);
80     for (int i = 1; i <= n; i++) {
81         int x;
82         scanf("%d", &x);
83         root[1] = tree.change(root[1], i, x, 1, max_n_m);
84     }
85     int id = 1;
86     while (m--) {
87         int opt;
88         scanf("%d", &opt);
89         if (opt == 0) { // 将可重集p中<=x且<=y的值放入一个新的可重集中
90             int p, x, y;
91             scanf("%d%d%d", &p, &x, &y);
92             tree.split(root[p], root[++id], x, y, 1, max_n_m);
93         } 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
94             int p, t;
95             scanf("%d%d", &p, &t); // 数据保证在此后的操作中不会出现可重集t
96             root[p] = tree.merge(root[p], root[t], 1, max_n_m);
97         } else if (opt == 2) { // 在p这个可重集中加入x个数字q
98             int p, x, q;
99             scanf("%d%d%d", &p, &x, &q);
100             root[p] = tree.change(root[p], q, x, 1, max_n_m);
101         } else if (opt == 3) { // 查询可重集p中大<=x且<=y的值的个数
102             int p, x, y;
103             scanf("%d%d%d", &p, &x, &y);
104             printf("%lld\n", tree.query_sum(root[p], x, y, 1, max_n_m));
105         } else { // 查询在p这个可重集中第k小的数，不存在时输出-1
106             int p, k;
107             scanf("%d%d", &p, &k);
108             if (tree.sum[root[p]] < k) {
109                 printf("-1\n");
110             } else printf("%d\n", tree.Kth(root[p], 1, max_n_m, k));
111         }
112     }
113 }

```

### 1.6.3 单点修改 + 单点最大连通数 (HDU1540)

```
1 class SEG { public:
2     struct Node {
3         int l, r;
4         int pre, suf, len;
5     } T[MAXN << 2];
6
7     void Build(int rt, int l, int r) {
8         T[rt].l = l, T[rt].r = r;
9         T[rt].pre = T[rt].suf = T[rt].len = r - l + 1;
10        if (l == r) return;
11        int mid = (l + r) >> 1;
12        Build(rt << 1, l, mid);
13        Build(rt << 1 | 1, mid + 1, r);
14    }
15
16    inline void push_up(int rt) {
17        T[rt].pre = T[rt << 1].pre;
18        T[rt].suf = T[rt << 1 | 1].suf;
19        T[rt].len = max(T[rt << 1].len, T[rt << 1 | 1].len);
20        T[rt].len = max(T[rt].len, T[rt << 1].suf + T[rt << 1 | 1].pre);
21        if (T[rt << 1].pre == T[rt << 1].r - T[rt << 1].l + 1) T[rt].pre +=
            T[rt << 1 | 1].pre;
22        if (T[rt << 1 | 1].suf == T[rt << 1 | 1].r - T[rt << 1 | 1].l + 1)
            T[rt].suf += T[rt << 1].suf;
23    }
24
25    // call: tree.update(1, x, 0); 摧毁
26    //       tree.update(1, x, 1); 修复
27
28
29    void update(int rt, int pos, int val) {
30        if (T[rt].l == T[rt].r) {
31            T[rt].pre = T[rt].suf = T[rt].len = val;
32            return;
33        }
34        int mid = (T[rt].l + T[rt].r) >> 1;
35        if (pos <= mid) update(rt << 1, pos, val);
36        else update(rt << 1 | 1, pos, val);
37        push_up(rt);
38    }
39
40    int query(int rt, int pos) {
```

```

41     if (T[rt].l == T[rt].r || T[rt].len == 0 || T[rt].len == T[rt].r -
42         T[rt].l + 1) {
43         return T[rt].len;
44     }
45     int mid = (T[rt].l + T[rt].r) >> 1;
46     if (pos <= mid) {
47         if (pos >= T[rt << 1].r - T[rt << 1].suf + 1)
48             return query(rt << 1, pos) + query(rt << 1 | 1, mid + 1);
49         else return query(rt << 1, pos);
50     } else {
51         if (pos <= T[rt << 1 | 1].l + T[rt << 1 | 1].pre - 1)
52             return query(rt << 1 | 1, pos) + query(rt << 1, mid);
53         else return query(rt << 1 | 1, pos);
54     }
55 } tree;

```

## 1.7 可持久化线段树（主席树）

### 1.7.1 静态区间第 K 小

```

1  /*
2      input                                output
3      5 5
4      25957 6405 15770 26287 26465
5      2 2 1                                6405
6      3 4 1                                15770
7      4 5 1                                26287
8      1 2 2                                25957
9      4 4 1                                26287
10 */
11 class HJT { public:
12     int ch[MAXN * 70][2], sum[MAXN * 70];
13     int tot = 0;
14     inline void push_up(int rt) {
15         sum[rt] = sum[ch[rt][0]] + sum[ch[rt][1]];
16     }
17     int update(int rt, int pos, int val, int be, int en) {
18         int nrt = ++tot;
19         ch[nrt][0] = ch[nrt][1] = sum[nrt] = 0;
20         if (be == en) {
21             sum[nrt] = sum[rt] + val;
22             return nrt;
23         }

```

```

24     int mid = (be + en) >> 1;
25     if (pos <= mid) {
26         ch[nrt][0] = update(ch[rt][0], pos, val, be, mid);
27         ch[nrt][1] = ch[rt][1];
28     } else {
29         ch[nrt][0] = ch[rt][0];
30         ch[nrt][1] = update(ch[rt][1], pos, val, mid + 1, en);
31     }
32     push_up(nrt);
33     return nrt;
34 }
35 int query(int lrt, int rrt, int k, int be, int en) {
36     if (be >= en) return be;
37     int delta = sum[ch[rrt][0]] - sum[ch[lrt][0]];
38     int mid = (be + en) >> 1;
39     if (delta >= k) return query(ch[lrt][0], ch[rrt][0], k, be, mid);
40     else return query(ch[lrt][1], ch[rrt][1], k - delta, mid + 1, en);
41 }
42 } tree;
43
44 int ai[MAXN], root[MAXN];
45 int main() {
46     int n, m;
47     scanf("%d%d", &n, &m);
48     for (int i = 1; i <= n; i++) {
49         scanf("%d", &ai[i]);
50         Discrete::insert(ai[i]);
51     }
52     Discrete::init();
53     root[0] = 0;
54     for (int i = 1; i <= n; i++) {
55         root[i] = tree.update(root[i-1], Discrete::val2id(ai[i]), 1, 1,
56             Discrete::blen);
57     }
58     while(m--) {
59         int l, r, k;
60         scanf("%d%d%d", &l, &r, &k);
61         printf("%d\n", Discrete::id2val(tree.query(root[l-1], root[r], k, 1,
62             Discrete::blen)));
63     }
64 }

```

### 1.7.2 区间内不同数个数

```

1 class HJT { public:
2     int query(int rt, int L, int R, int be, int en) {
3         if (L <= be && en <= R) return sum[rt];
4         int mid = (be + en) >> 1;
5         int ans = 0;
6         if (L <= mid) ans += query(ch[rt][0], L, R, be, mid);
7         if (R > mid) ans += query(ch[rt][1], L, R, mid + 1, en);
8         return ans;
9     }
10 } tree;
11 int val[MAXN], root[MAXN];
12 unordered_map<int, int> pre;
13
14 int main() {
15     int n; scanf("%d", &n);
16     for (int i = 1; i <= n; i++) scanf("%d", &val[i]);
17     root[0] = 0;
18     for (int i = 1; i <= n; i++) {
19         if (pre[val[i]]) {
20             int tmp = tree.change(root[i - 1], pre[val[i]], -1, 1, n);
21             root[i] = tree.change(tmp, i, 1, 1, n);
22         } else {
23             root[i] = tree.change(root[i - 1], i, 1, 1, n);
24         }
25         pre[val[i]] = i;
26     }
27     int q; scanf("%d", &q);
28     while (q--) {
29         int l, r;
30         scanf("%d%d", &l, &r);
31         printf("%d\n", tree.query(root[r], l, r, 1, n));
32     }
33 }

```

### 1.7.3 静态区间第 K 小

```

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

```



```

6         if (delta >= k) return query(ch[lrt][0], ch[rtr][0], ch[fa][0],
7             ch[faa][0], k, be, mid);
8         else return query(ch[lrt][1], ch[rtr][1], ch[fa][1], ch[faa][1], k -
9             delta, mid + 1, en);
10     }
11 } tree;
12
13 struct Edge {
14     int to, nex;
15 } e[MAXN << 1];
16 int head[MAXN], tol;
17 void addEdge(int u, int v) {
18     e[tol].to = v, e[tol].nex = head[u], head[u] = tol, tol++;
19 }
20 int val[MAXN], root[MAXN], new_val[MAXN];
21 int dep[MAXN], fa[MAXN][32], lg[MAXN];
22 void init(int _n) {
23     for (int i = 1; i <= _n; i++) lg[i] = lg[i - 1] + (1 << lg[i - 1] == i);
24 }
25 void dfs(int u, int f) {
26     fa[u][0] = f;
27     dep[u] = dep[f] + 1;
28     root[u] = tree.update(root[f], new_val[u], 1, 1, Discrete::blen);
29     for (int i = 1; i <= lg[dep[u]]; i++) fa[u][i] = fa[fa[u][i - 1]][i - 1];
30     for (int i = head[u]; ~i; i = e[i].nex) {
31         int v = e[i].to;
32         if (v == f) continue;
33         dfs(v, u);
34     }
35 }
36
37 int LCA(int u, int v) {
38     if (dep[u] < dep[v]) swap(u, v);
39     while (dep[u] > dep[v]) u = fa[u][lg[dep[u]] - dep[v]] - 1;
40     if (u == v) return u;
41     for (int k = lg[dep[u]] - 1; k >= 0; k--) {
42         if (fa[u][k] != fa[v][k]) u = fa[u][k], v = fa[v][k];
43     }
44     return fa[u][0];
45 }
46
47 int main() {
48     int n, m; scanf("%d%d", &n, &m);
49     init(n);
50     for (int i = 1; i <= n; i++) head[i] = -1;
51     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],
64                 root[fa[lca][0]], k, 1, Discrete::blen)));
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         }

```

```

22     return nrt;
23 }
24 int query(int rt, int R, int be, int en) {
25     if (be == en) return be;
26     int ans = -1;
27     int mid = (be + en) >> 1;
28     if (T[ls].maxx > R) ans = query(T[rt].lson, R, be, mid);
29     if (ans == -1 && T[rs].maxx > R) ans = query(T[rt].rson, R, mid + 1,
30         en);
31     return ans;
32 }
33 } tree;
34
35 int arr[MAXN], root[MAXN];
36 int main() {
37     int n, m; scanf("%d%d", &n, &m);
38     for (int i = 1; i <= n; i++) scanf("%d", &arr[i]);
39     Discrete::push(0);
40     for (int i = 1; i <= n; i++) Discrete::push(arr[i]),
41         Discrete::push(arr[i] + 1);
42     Discrete::init();
43     bep = n + 1;
44     root[n + 1] = tree.build(1, blen);
45     for (int i = n; i >= 1; i--) {
46         int x = Discrete::lb(arr[i]);
47         root[i] = tree.update(root[i + 1], 1, blen, x, i);
48     }
49     while (m--) {
50         int l, r;
51         scanf("%d%d", &l, &r);
52         int res = tree.query(root[l], r, 1, blen);
53         if (res == -1) res = blen;
54         printf("%d\n", Discrete::get_num(res));
55     }
56 }

```

## 1.8 无旋 Treap (FHQ Treap)

### 1.8.1 区间翻转

```

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

```

```

5  int tot, root;
6
7  void init() {
8      tot = 0, root = 0;
9  }
10 Treap() { init(); }
11
12 inline int Newnode(int v) {
13     val[++tot] = v;
14     dat[tot] = rand();
15     siz[tot] = 1;
16     return tot;
17 }
18 inline void push_up(int rt) {
19     siz[rt] = siz[ch[rt][0]] + siz[ch[rt][1]] + 1;
20 }
21
22 int build(int l, int r) {
23     if (l > r) return 0;
24     int mid = (l + r) >> 1;
25     int newnode = Newnode(mid);
26     ch[newnode][0] = build(l, mid - 1);
27     ch[newnode][1] = build(mid + 1, r);
28     push_up(newnode);
29     return newnode;
30 }
31
32 inline void push_down(int rt) {
33     if (fl[rt]) {
34         swap(ch[rt][0], ch[rt][1]);
35         if (ch[rt][0]) fl[ch[rt][0]] ^= 1;
36         if (ch[rt][1]) fl[ch[rt][1]] ^= 1;
37         fl[rt] = 0;
38     }
39 }
40
41 void split(int rt, int k, int &x, int &y) { // 按照编号进行分裂
42     if (!rt) x = y = 0;
43     else {
44         push_down(rt);
45         if (k <= siz[ch[rt][0]]) {
46             y = rt;
47             split(ch[rt][0], k, x, ch[rt][0]);
48         } else {

```

```

49         x = rt;
50         split(ch[rt][1], k - siz[ch[rt][0]] - 1, ch[rt][1], y);
51     }
52     push_up(rt);
53 }
54 }
55
56 int merge(int x, int y) {
57     if (!x || !y) return x + y;
58     if (dat[x] < dat[y]) {
59         push_down(x);
60         ch[x][1] = merge(ch[x][1], y);
61         push_up(x);
62         return x;
63     } else {
64         push_down(y);
65         ch[y][0] = merge(x, ch[y][0]);
66         push_up(y);
67         return y;
68     }
69 }
70
71 void dfs(int rt) {
72     push_down(rt);
73     if (ch[rt][0]) dfs(ch[rt][0]);
74     printf("%d ", val[rt]);
75     if (ch[rt][1]) dfs(ch[rt][1]);
76 }
77 } tree;
78
79 int main() {
80     int n, q; scanf("%d%d", &n, &q);
81     tree.build(1, n);
82     while (q--) {
83         int l, r; scanf("%d%d", &l, &r);
84         int a, b, c;
85         tree.split(tree.root, l - 1, a, b);
86         tree.split(b, r - l + 1, b, c);
87         tree.fl[b] ^= 1;
88         tree.root = tree.merge(a, tree.merge(b, c));
89     }
90     tree.dfs(tree.root);
91 }

```

## 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。

```
1 class FHQ { public:
2     const int MAXM = 50 * MAXN; // 50倍注意!
3     int ch[MAXM][2];
4     int dat[MAXM], siz[MAXM], val[MAXM];
5     int tot, root;
6
7     void init() { // 初始化
8         tot = 0;
9         root = 0;
10        siz[0] = val[0] = 0;
11    }
12
13    inline void push_up(int rt) { // 传递信息
14        siz[rt] = siz[ch[rt][0]] + siz[ch[rt][1]] + 1;
15    }
16
17    inline int New(int v) { // 新建一个节点, value = v
18        val[++tot] = v;
19        dat[tot] = rand();
20        siz[tot] = 1;
21        ch[tot][0] = ch[tot][1] = 0;
22        return tot;
23    }
24
25    inline int Copy(int rt) { // 复制点的信息
26        int newnode = ++tot;
27        val[tot] = val[rt], dat[tot] = dat[rt], siz[tot] = siz[rt];
28        ch[tot][0] = ch[rt][0], ch[tot][1] = ch[rt][1];
29        return newnode;
30    }
31
32    inline int merge(int x, int y) { // 合并
33        if (!x || !y) return x + y;
34        if (dat[x] < dat[y]) {
35            int newnode = Copy(x);
36            ch[newnode][1] = merge(ch[newnode][1], y);
37            push_up(newnode);
38            return newnode;
39        } else {
```

```

40     int newnode = Copy(y);
41     ch[newnode][0] = merge(x, ch[newnode][0]);
42     push_up(newnode);
43     return newnode;
44 }
45 }
46
47 inline void split(int rt, int v, int &x, int &y) { // 按照权值进行分裂
48     if (!rt) x = y = 0;
49     else {
50         if (val[rt] <= v) {
51             x = Copy(rt);
52             split(ch[x][1], v, ch[x][1], y);
53             push_up(x);
54         } else {
55             y = Copy(rt);
56             split(ch[y][0], v, x, ch[y][0]);
57             push_up(y);
58         }
59     }
60 }
61
62 void del(int &rt, int v) { // 删除value为v的数
63     int x = 0, y = 0, z = 0;
64     split(rt, v, x, z);
65     split(x, v - 1, x, y);
66     y = merge(ch[y][0], ch[y][1]);
67     rt = merge(x, merge(z, y));
68 }
69
70 void insert(int &rt, int v) { // 插入value为v的数
71     int x = 0, y = 0, z = 0;
72     split(rt, v, x, y);
73     z = New(v);
74     rt = merge(x, merge(z, y));
75 }
76
77 int get_val(int rt, int k) { // 得到第k大的数（从小到大）的value
78     if (k == siz[ch[rt][0]] + 1) return val[rt];
79     else if (k <= siz[ch[rt][0]]) return get_val(ch[rt][0], k);
80     else return get_val(ch[rt][1], k - siz[ch[rt][0]] - 1);
81 }
82
83 int get_Kth(int &rt, int v) { // 查询v的排名

```

```

84     int x, y;
85     split(rt, v - 1, x, y);
86     int ans = siz[x] + 1;
87     rt = merge(x, y);
88     return ans;
89 }
90
91 int get_pre(int &rt, int v) { // 求前驱, 若不存在返回-2147483647
92     int x, y, ans;
93     split(rt, v - 1, x, y);
94     if (!x) return -2147483647;
95     ans = get_val(x, siz[x]);
96     rt = merge(x, y);
97     return ans;
98 }
99
100 int get_next(int &rt, int v) { // 求后驱, 若不存在返回2147483647
101     int x, y, ans;
102     split(rt, v, x, y);
103     if (!y) return 2147483647;
104     ans = get_val(y, 1);
105     rt = merge(x, y);
106     return ans;
107 }
108 } tree;
109
110 int root[MAXN];
111
112 int main() {
113     int q; scanf("%d", &q);
114     tree.init();
115     for (int i = 1; i <= q; i++) {
116         int ver, opt, x;
117         scanf("%d%d%d", &ver, &opt, &x);
118         root[i] = root[ver];
119         switch (opt) {
120             // 插入x
121             case 1: tree.insert(root[i], x); break;
122             // 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)
123             case 2: tree.del(root[i], x); break;
124             // 查询x的排名 (排名定义为比当前数小的数的个数+1)
125             case 3: printf("%d\n", tree.get_Kth(root[i], x)); break;
126             // 查询排名为x的数
127             case 4: printf("%d\n", tree.get_val(root[i], x)); break;

```



```

128 //
129     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）
130 case 5: printf("%d\n", tree.get_pre(root[i], x)); break;
131 //
132     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）
133 default: printf("%d\n", tree.get_next(root[i], x));
134 }
135 }

```

## 1.9 树套树

### 1.9.1 带修主席树

```

1  /*
2   input      output
3   5 3
4   3 2 1 4 7
5   Q 1 4 3    3
6   C 2 6
7   Q 2 5 3    6
8  */
9  class BIT { public:
10     // HJT begin
11     int ch[MAXN * 400][2], sum[MAXN * 400], tot = 0;
12     inline void push_up(int rt) {
13         sum[rt] = sum[ch[rt][0]] + sum[ch[rt][1]];
14     }
15     int update(int rt, int pos, int val, int be, int en) {
16         int nrt = ++tot;
17         ch[nrt][0] = ch[nrt][1] = sum[nrt] = 0;
18         if (be == en) {
19             sum[nrt] = sum[rt] + val;
20             return nrt;
21         }
22         int mid = (be + en) >> 1;
23         if (pos <= mid) {
24             ch[nrt][0] = update(ch[rt][0], pos, val, be, mid);
25             ch[nrt][1] = ch[rt][1];
26         } else {
27             ch[nrt][0] = ch[rt][0];
28             ch[nrt][1] = update(ch[rt][1], pos, val, mid + 1, en);
29         }

```

```

30     push_up(nrt);
31     return nrt;
32 }
33 // HJT end
34 int n, c_len, root[MAXN];
35 void init(int _n, int _c_len) {
36     c_len = _c_len, n = _n;
37     for (int i = 1; i <= c_len; i++) root[i] = i;
38     tot = c_len;
39 }
40 inline int lowbit(int x) { return x & (-x); }
41 void insert(int pos, int pos_val, int val) {
42     for (int i = pos; i <= n; i += lowbit(i)) root[i] = update(root[i],
43         pos_val, val, 1, c_len);
44 }
45 int t1[MAXN], t2[MAXN], n1, n2;
46 inline int Kth(int be, int en, int k) {
47     if (be >= en) return be;
48     int mid = (be + en) >> 1, delta = 0;
49
50     for (int i = 1; i <= n1; i++) delta -= sum[ch[t1[i]]][0]];
51     for (int i = 1; i <= n2; i++) delta += sum[ch[t2[i]]][0]];
52     if (delta >= k) {
53         for (int i = 1; i <= n1; i++) t1[i] = ch[t1[i]][0]];
54         for (int i = 1; i <= n2; i++) t2[i] = ch[t2[i]][0]];
55         return Kth(be, mid, k);
56     } else {
57         for (int i = 1; i <= n1; i++) t1[i] = ch[t1[i]][1]];
58         for (int i = 1; i <= n2; i++) t2[i] = ch[t2[i]][1]];
59         return Kth(mid + 1, en, k - delta);
60     }
61 }
62 int query(int l, int r, int k) {
63     n1 = n2 = 0;
64     for (int i = l - 1; i >= 1; i -= lowbit(i)) t1[++n1] = root[i];
65     for (int i = r; i >= 1; i -= lowbit(i)) t2[++n2] = root[i];
66     return Kth(1, c_len, k);
67 }
68 } tree;
69 int ai[MAXN];
70 int opt[MAXN], l[MAXN], r[MAXN], k[MAXN], x[MAXN], y[MAXN];
71 int main() {
72     int n, m; cin >> n >> m;
73     for (int i = 1; i <= n; i++) cin >> ai[i];

```

```

73     for (int i = 1; i <= n; i++) Discrete::insert(ai[i]);
74     for (int i = 1; i <= m; i++) {
75         char op; cin >> op;
76         if (op == 'Q') {
77             opt[i] = 1; cin >> l[i] >> r[i] >> k[i];
78         } else {
79             opt[i] = 2; cin >> x[i] >> y[i];
80             Discrete::insert(y[i]);
81         }
82     }
83     Discrete::init();
84     for (int i = 1; i <= n; i++) ai[i] = Discrete::val2id(ai[i]);
85     for (int i = 1; i <= m; i++) {
86         if (opt[i] == 2) y[i] = Discrete::val2id(y[i]);
87     }
88     tree.init(n, Discrete::blen);
89
90     for (int i = 1; i <= n; i++) tree.insert(i, ai[i], 1);
91     for (int i = 1; i <= m; i++) {
92         if (opt[i] == 1) { // 表示查询下标在区间[l,r]中的第k小的数
93             cout << Discrete::id2val(tree.query(l[i], r[i], k[i])) << '\n';
94         } else { // 表示将a[x]改为y
95             tree.insert(x[i], ai[x[i]], -1);
96             ai[x[i]] = y[i];
97             tree.insert(x[i], ai[x[i]], 1);
98         }
99     }
100 }

```

## 1.9.2 区间修改区间查询第 K 大 ([ZJOI2013]K 大数查询)

```

1  /*
2   树状数组套主席套线段树
3   可重集的并不去除重复元素的，如{1,1,4} & {5,1,4}={1,1,4,5,1,4}
4   input      output
5   2 5
6   1 1 2 1
7   1 1 2 2
8   2 1 1 2    1
9   2 1 1 1    2
10  2 1 2 3    1
11  */
12 class BIT { public:

```

```

13     class SEG { public:
14         int tot_rt, ch[MAXN * 400][2]; ll lazy[MAXN * 400], sum[MAXN * 400];
15 #define lson ch[rt][0]
16 #define rson ch[rt][1]
17         inline void push_up(int rt) {
18             sum[rt] = sum[lson] + sum[rson];
19         }
20         inline void push_down(int rt, int len_left, int len_right) {
21             if (lazy[rt]) {
22                 if (!ch[rt][0]) ch[rt][0] = ++tot_rt;
23                 if (!ch[rt][1]) ch[rt][1] = ++tot_rt;
24                 sum[lson] += lazy[rt] * len_left, sum[rson] += lazy[rt] *
                    len_right;
25                 lazy[lson] += lazy[rt], lazy[rson] += lazy[rt];
26                 lazy[rt] = 0;
27             }
28         }
29         int change(int rt, int L, int R, int val, int be, int en) {
30             if (!rt) rt = ++tot_rt;
31             if (L <= be && R >= en) {
32                 sum[rt] += (ll) (en - be + 1);
33                 lazy[rt] += (ll) val;
34                 return rt;
35             }
36             int mid = (be + en) >> 1;
37             push_down(rt, mid - be + 1, en - (mid + 1) + 1);
38             if (L <= mid) lson = change(lson, L, R, val, be, mid);
39             if (R > mid) rson = change(rson, L, R, val, mid + 1, en);
40             push_up(rt);
41             return rt;
42         }
43         ll query(int rt, int L, int R, int be, int en) {
44             if (!rt) return 0;
45             if (L <= be && R >= en) return sum[rt];
46             int mid = (be + en) >> 1;
47             push_down(rt, mid - be + 1, en - (mid + 1) + 1);
48             ll ans = 0;
49             if (L <= mid) ans += query(lson, L, R, be, mid);
50             if (R > mid) ans += query(rson, L, R, mid + 1, en);
51             return ans;
52         }
53     } seg;
54
55     int n, c_len, root[MAXN];

```

```

56 void init(int _n, int _c_len) {
57     c_len = _c_len, n = _n;
58     for (int i = 1; i <= c_len; i++) root[i] = i;
59     seg.tot_rt = _c_len;
60 }
61 inline int lowbit(int x) { return x & (-x); }
62 inline int log(int x) { return 1ll << (int) (log2(x)); }
63 void insert(int l, int r, int c) {
64     for (int i = c_len - c + 1; i <= c_len; i += lowbit(i))
65         seg.change(root[i], l, r, 1, 1, n);
66 }
67 int query(int l, int r, ll k) {
68     int ans = 0;
69     ll sum = 0;
70     for (int i = log(c_len); i != 0; i >>= 1) {
71         if (ans + i > c_len) continue;
72         ll tmp = seg.query(root[ans + i], l, r, 1, n) + sum;
73         if (tmp < k) ans += i, sum = tmp;
74     }
75     ans++;
76     return c_len - ans + 1;
77 } tree;
78
79 int opt[MAXN], l[MAXN], r[MAXN]; ll c[MAXN];
80 int main() {
81     int n, m; scanf("%d%d", &n, &m);
82     for (int i = 1; i <= m; i++) {
83         scanf("%d%d%d%lld", &opt[i], &l[i], &r[i], &c[i]);
84         if (opt[i] == 1) Discrete::push(c[i]);
85     }
86     Discrete::init(); // 离散化
87     int c_len = Discrete::blen;
88     tree.init(n, c_len); // n为[l,r], c_len表示离散化后数字个数
89     for (int i = 1; i <= m; i++) {
90         if (opt[i] == 1) { // 表示将c加入到编号在[l,r]内的集合中
91             tree.insert(l[i], r[i], Discrete::val2id(c[i]));
92         } 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
93             printf("%lld\n", Discrete::id2val(tree.query(l[i], r[i], c[i])));
94         }
95     }
96 }

```

## 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  $O(n)$ .

```
1  const int MAXN = 2e5 + 5;    // 点的个数
2  class KD {
3  public:
4      struct node {
5          double x, y;
6      } T[MAXN];
7      int ch[MAXN][2];
8      double L[MAXN], R[MAXN], D[MAXN], U[MAXN];
9      inline void push_up(int rt) {
10         L[rt] = R[rt] = T[rt].x;
11         D[rt] = U[rt] = T[rt].y;
12         if (ch[rt][0]) {
13             L[rt] = min(L[rt], L[ch[rt][0]]), R[rt] = max(R[rt],
14                 R[ch[rt][0]]),
15             D[rt] = min(D[rt], D[ch[rt][0]]), U[rt] = max(U[rt],
16                 U[ch[rt][0]]);
17         }
18         if (ch[rt][1]) {
19             L[rt] = min(L[rt], L[ch[rt][1]]), R[rt] = max(R[rt],
20                 R[ch[rt][1]]),
21             D[rt] = min(D[rt], D[ch[rt][1]]), U[rt] = max(U[rt],
22                 U[ch[rt][1]]);
23         }
24     }
25
26     int d[MAXN];
27     int build(int l, int r) {
28         if (l > r) return 0;
29         int mid = (l + r) >> 1;
30         double av1 = 0, av2 = 0, va1 = 0, va2 = 0;
31         for (int i = l; i <= r; i++) av1 += T[i].x, av2 += T[i].y;
32         av1 /= (r - l + 1);
33         av2 /= (r - l + 1);
34         for (int i = l; i <= r; i++) va1 += (av1 - T[i].x) * (av1 - T[i].x),
35             va2 += (av2 - T[i].y) * (av2 - T[i].y);
36         if (va1 > va2)
37             d[mid] = 1, nth_element(T + l, T + mid, T + r + 1,
38                 [&](const node &ta, const node &tb) {
39                     return ta.x < tb.x; });
```

```

34     else
35         d[mid] = 2, nth_element(T + l, T + mid, T + r + 1,
36                                 [&](const node &ta, const node &tb) {
37                                     return ta.y < tb.y; });
38
39     ch[mid][0] = build(l, mid - 1);
40     ch[mid][1] = build(mid + 1, r);
41     push_up(mid);
42     return mid;
43 }
44
45 double f(int a, int b) {
46     double ans = 0;
47     if (L[b] > T[a].x) ans += (L[b] - T[a].x) * (L[b] - T[a].x);
48     if (R[b] < T[a].x) ans += (T[a].x - R[b]) * (T[a].x - R[b]);
49     if (D[b] > T[a].y) ans += (D[b] - T[a].y) * (D[b] - T[a].y);
50     if (U[b] < T[a].y) ans += (T[a].y - U[b]) * (T[a].y - U[b]);
51     return ans;
52 }
53
54 double ans;
55 double dist(int a, int b) {
56     return (T[a].x - T[b].x) * (T[a].x - T[b].x) + (T[a].y - T[b].y) *
57            (T[a].y - T[b].y);
58 }
59
60 void query(int l, int r, int rt) {
61     if (l > r) return;
62     int mid = (l + r) >> 1;
63     if (mid != rt) ans = min(ans, dist(rt, mid));
64     if (l == r) return;
65     double distl = f(rt, ch[mid][0]), distr = f(rt, ch[mid][1]);
66     if (distl < ans && distr < ans) {
67         if (distl < distr) {
68             query(l, mid - 1, rt);
69             if (distr < ans) query(mid + 1, r, rt);
70         } else {
71             query(mid + 1, r, rt);
72             if (distl < ans) query(l, mid - 1, rt);
73         }
74     } else {
75         if (distl < ans) query(l, mid - 1, rt);
76         if (distr < ans) query(mid + 1, r, rt);
77     }
78 }
79
80 void init() { ans = inf_ll; }
81 double get_res() { return ans; }

```

```

76 } tree;
77
78 int main() {
79     int n;
80     scanf("%d", &n);
81     for (int i = 1; i <= n; i++) scanf("%lf%lf", &tree.T[i].x, &tree.T[i].y);
82     tree.init();
83     tree.build(1, n);
84     for (int i = 1; i <= n; i++) {
85         tree.query(1, n, i);
86     }
87     printf("%.4lf\n", sqrt(tree.get_res()));
88 }

```

### 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：  $O(kn \log n)$ 。

```

1  /*
2      input
3      10 5
4      0 0, 0 1, 1 0, 1 1, 2 0, 2 1, 1 2, 0 2, 3 0, 3 1
5      ouput
6      9
7  */
8  const int MAXN = 1e5 + 5;    // 点的个数
9  class KD {
10 public:
11     priority_queue<ll, vector<ll>, greater<ll> > q;
12     struct node {
13         int x, y;
14     } T[MAXN];
15     int ch[MAXN][2], L[MAXN], R[MAXN], D[MAXN], U[MAXN];
16
17     inline void push_up(int rt) {
18         L[rt] = R[rt] = T[rt].x;
19         D[rt] = U[rt] = T[rt].y;
20         if (ch[rt][0]) {
21             L[rt] = min(L[rt], L[ch[rt][0]]), R[rt] = max(R[rt],
22                 R[ch[rt][0]]),

```



```

        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);

```

```

60         if (distr > q.top()) query(mid + 1, r, rt);
61     } else {
62         query(mid + 1, r, rt);
63         if (distl > q.top()) query(l, mid - 1, rt);
64     }
65 } else {
66     if (distl > q.top()) query(l, mid - 1, rt);
67     if (distr > q.top()) query(mid + 1, r, rt);
68 }
69 }
70 void init(int k) {
71     k *= 2;
72     for (int i = 1; i <= k; i++) q.push(0);
73 }
74 ll get_res() {
75     return q.top();
76 }
77 } tree;
78
79 int main() {
80     int n, k;
81     scanf("%d%d", &n, &k);
82     tree.init(k);
83     for (int i = 1; i <= n; i++) scanf("%d%d", &tree.T[i].x, &tree.T[i].y);
84     tree.build(1, n);
85     for (int i = 1; i <= n; i++) {
86         tree.query(1, n, i);
87     }
88     printf("%lld\n", tree.get_res());
89 }

```

### 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  $O(n^{1-\frac{1}{k}})$

```

1  /*
2     input      output
3     4
4     1 2 3 3
5     2 1 1 3 3   3

```

```

6      1 1 1 1
7      2 1 1 0 7   5
8      3
9  */
10 const int MAXN = 2e5 + 5;    // 操作次数
11 class KD {
12 public:
13     struct node {
14         int x, y, v;
15     } T[MAXN];
16     int ch[MAXN][2], L[MAXN], R[MAXN], D[MAXN], U[MAXN];
17     int siz[MAXN], sum[MAXN], g[MAXN], d[MAXN];
18
19     inline void push_up(int rt) {
20         siz[rt] = siz[ch[rt][0]] + siz[ch[rt][1]] + 1;
21         sum[rt] = sum[ch[rt][0]] + sum[ch[rt][1]] + T[rt].v;
22         L[rt] = R[rt] = T[rt].x;
23         U[rt] = D[rt] = T[rt].y;
24         if (ch[rt][0]) {
25             L[rt] = min(L[rt], L[ch[rt][0]]), R[rt] = max(R[rt],
26                 R[ch[rt][0]]),
27             D[rt] = min(D[rt], D[ch[rt][0]]), U[rt] = max(U[rt],
28                 U[ch[rt][0]]);
29         }
30         if (ch[rt][1]) {
31             L[rt] = min(L[rt], L[ch[rt][1]]), R[rt] = max(R[rt],
32                 R[ch[rt][1]]),
33             D[rt] = min(D[rt], D[ch[rt][1]]), U[rt] = max(U[rt],
34                 U[ch[rt][1]]);
35         }
36     }
37
38     int build(int l, int r) {
39         if (l > r) return 0;
40         int mid = (l + r) >> 1;
41         double av1 = 0, av2 = 0, va1 = 0, va2 = 0;
42         for (int i = l; i <= r; i++) av1 += T[g[i]].x, av2 += T[g[i]].y;
43         av1 /= (r - l + 1);
44         av2 /= (r - l + 1);
45         for (int i = l; i <= r; i++) {
46             va1 += (av1 - T[g[i]].x) * (av1 - T[g[i]].x),
47             va2 += (av2 - T[g[i]].y) * (av2 - T[g[i]].y);
48         }
49         if (va1 > va2)

```

```

46     nth_element(g + l, g + mid, g + r + 1, [&](int ta, int tb) {
47         return T[ta].x < T[tb].x; }), d[g[mid]] = 1;
48     else nth_element(g + l, g + mid, g + r + 1, [&](int ta, int tb) {
49         return T[ta].y < T[tb].y; }), d[g[mid]] = 2;
50     ch[g[mid]][0] = build(l, mid - 1);
51     ch[g[mid]][1] = build(mid + 1, r);
52     push_up(g[mid]);
53     return g[mid];
54 }
55
56 const double bad_para = 0.725;
57 int tot;
58 bool bad(int rt) {
59     return bad_para * siz[rt] <= (double) max(siz[ch[rt][0]],
60         siz[ch[rt][1]]);
61 }
62 void dfs(int rt) {
63     if (!rt) return;
64     dfs(ch[rt][0]);
65     g[++tot] = rt;
66     dfs(ch[rt][1]);
67 }
68 void rebuild(int &rt) {
69     tot = 0;
70     dfs(rt);
71     rt = build(1, tot);
72 }
73 void insert(int &rt, int v) {
74     if (!rt) {
75         rt = v;
76         push_up(rt);
77         return;
78     }
79     if (d[rt] == 1) {
80         if (T[v].x <= T[rt].x) insert(ch[rt][0], v);
81         else insert(ch[rt][1], v);
82     } else {
83         if (T[v].y <= T[rt].y) insert(ch[rt][0], v);
84         else insert(ch[rt][1], v);
85     }
86     push_up(rt);
87     if (bad(rt)) rebuild(rt);
88 }
89 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  $1 \leq x \leq r - l + 1$ )

**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。

```
1 #define IT set<node>::iterator
2 const int MAXN = 1e5 + 5;
3 const int MOD7 = 1e9 + 7;
4
5 ll powmod(ll base, ll times, ll mod) {
6     ll p = 1;
7     ll ans = base % mod;
8     while (times) {
9         if (times & 1) p = p * ans % mod;
10        ans = ans * ans % mod;
11        times >>= 1;
12    }
13    return p;
14 }
15 class ODT {
16 public:
17     struct node {
18         int l, r;
19         mutable ll val; // 玄学mutable注意!
20         node() {}
21         node(int _l, int _r = -1, ll _val = 0) { l = _l, r = _r, val = _val; }
22         bool operator<(const node &tb) const { return l < tb.l; }
23     };
24     set<node> st;
25     IT split(int pos) {
26         IT it = st.lower_bound(node(pos));
27         if (it != st.end() && it->l == pos) return it;
28         --it;
29         int ul = it->l, ur = it->r; ll uv = it->val;
30         st.erase(it);
31         st.insert(node(ul, pos - 1, uv));
32         return st.insert(node(pos, ur, uv)).first;
33     }
34     void add(int l, int r, ll v = 1) { // 对一段区间加上一个数
35         IT itl = split(l), itr = split(r + 1);
36         for (; itl != itr; itl++) itl->val += v;
37     }
38     void assign_val(int l, int r, ll v = 0) { // 对一段区间进行赋值
39         IT itl = split(l), itr = split(r + 1);
40         st.erase(itl, itr);
41         st.insert(node(l, r, v));
42     }
```

```

43 ll rank(int l, int r, int k) {
44     vector<pair<ll, int>> vp;
45     IT itl = split(l), itr = split(r + 1);
46     vp.clear();
47     for (; itl != itr; itl++)
48         vp.push_back(pair<ll, int>(itl->val, (itl->r) - (itl->l) + 1));
49     sort(vp.begin(), vp.end());
50     for (vector<pair<ll, int>>::iterator it = vp.begin(); it != vp.end();
51         it++) {
52         k -= it->second;
53         if (k <= 0) return it->first;
54     }
55     return -1ll;
56 }
57 ll sum(int l, int r, int times, int mod) {
58     IT itl = split(l), itr = split(r + 1);
59     ll ans = 0;
60     for (; itl != itr; itl++)
61         ans = (ans + (ll) (itl->r - itl->l + 1) * powmod(itl->val, (ll)
62             times, (ll) mod) % mod) % mod;
63     return ans;
64 }
65 void insert(int l, int r, ll v) {
66     st.insert(node(l, r, v));
67 }
68 } tree;
69
70 ll seed, vmax;
71 ll rnd() {
72     ll ret = seed;
73     seed = (seed * 7 + 13) % MOD7;
74     return ret;
75 }
76
77 ll a[MAXN];
78 int main() {
79     int n, m;
80     scanf("%d%d%lld%lld", &n, &m, &seed, &vmax);
81     for (int i = 1; i <= n; i++) {
82         a[i] = (rnd() % vmax) + 1;
83         tree.insert(i, i, a[i]);
84     }
85     tree.insert(n + 1, n + 1, 0);
86     int lines = 0;

```

```

85 for (int i = 1; i <= m; i++) {
86     int opt = int(rnd() % 4) + 1, l = int(rnd() % n) + 1, r = int(rnd() %
      n) + 1;
87     if (l > r) swap(l, r);
88     int x, y;
89     // 原题神奇的操作，可以不用理会
90     if (opt == 3) x = int(rnd() % (r - l + 1)) + 1;
91     else x = int(rnd() % vmax) + 1;
92     if (opt == 4) y = int(rnd() % vmax) + 1;
93
94     if (opt == 1) tree.add(l, r, x);
95     else if (opt == 2) tree.assign_val(l, r, x);
96     else if (opt == 3) printf("%lld\n", tree.rank(l, r, x));
97     else printf("%lld\n", tree.sum(l, r, x, y));
98 }
99 }

```

## 2 字符串

### 2.1 字符串哈希

#### 2.1.1 区间一维哈希

```

1 typedef unsigned long long ull;
2 namespace hash {
3     const ull seed = 19260817;
4     ull base[SIZE], hash[SIZE];
5
6     void init() {
7         base[0] = 1;
8         for (int i = 1; i < SIZE; i++) base[i] = base[i - 1] * seed;
9     }
10
11     ull _hash(int l, int r) {
12         return hash[r] - hash[l - 1] * base[r - l + 1];
13     }
14
15     void getHash(char str[], int len) {
16         for (int i = 1; i <= len; i++) hash[i] = hash[i - 1] * seed + str[i]
          - 'a' + 3;
17     }

```



## 2.2 Next 函数

### 2.2.1 求 next 函数

```
1  /*
2      input: ABCABDABCD
3      index  0  1  2  3  4  5  6  7  8  9  10
4      x[]    A  B  C  A  B  D  A  B  C  D  \0
5      nxt[]  -1 -1 -1 0  1 -1 0  1  2 -1 0
6  */
7  // call: scanf("%s", str+1); get_next(str+1, strlen(str+1), nex+1);
8  void get_next(char x[], int x_len, int nxt[]) {
9      int i, j;
10     for (nxt[0] = j = -1, i = 1; i < x_len; nxt[i++] = j) {
11         while (~j && x[j + 1] != x[i]) j = nxt[j];
12         if (x[j + 1] == x[i]) j++;
13     }
14 }
15
16 /*
17     input: ABCABDABCD
18     index  1  2  3  4  5  6  7  8  9  10  11
19     x[]    A  B  C  A  B  D  A  B  C  D  \0
20     nxt[]  0  0  0  1  2  0  1  2  3  0  0
21 */
22 // call: scanf("%s", str+1); get_next(str, strlen(str+1), nex);
23 void get_next(char x[], int x_len, int nxt[]) {
24     nxt[1] = 0;
25     for (int i = 2, j = 0; i <= x_len; i++) {
26         while (j && x[j + 1] != x[i]) j = nxt[j];
27         if (x[j + 1] == x[i]) ++j;
28         nxt[i] = j;
29     }
30 }
```

### 2.2.2 求出每个循环节的数量和终点位置 (HDU1358)

```
1  /*
2      input    output
3      3        2 2
4      aaa      3 3
5
6      12       2 2
7      aabaabaab 6 2
```

```

8           9 3
9          12 4
10  */
11 int main() {
12     int n;
13     int cas = 0;
14     while (~scanf("%d", &n) && n) {
15         scanf("%s", str+1);
16         get_next(str+1, n, nex+1); // 方法1
17         for (int i = 2; i <= n; i++) {
18             if (nex[i] != -1 && (i % (i - nex[i] - 1) == 0))
19                 printf("%d %d\n", i, i / (i - nex[i] - 1));
20         }
21         puts("");
22     }
23 }

```

### 2.2.3 求同时是前缀和后缀的串长 (POJ2752)

```

1  /*
2      input          output
3      ababcabababababababab  2 4 9 18
4      aaaaaa          1 2 3 4 5
5  */
6  int main() {
7      while (~scanf("%s", s + 1)) {
8          int cnt = 0;
9          int len;
10         get_next(s + 1, len = strlen(s + 1), nex + 1); // 方法1
11         for (int t = nex[len]; ~t; t = nex[t+1]) {
12             if (s[t+1] == s[len]) ans[cnt++] = t + 1;
13         }
14         for (int i = cnt - 1; i >= 0; i--) printf("%d ", ans[i]);
15         printf("%d\n", len);
16     }
17 }

```

### 2.2.4 求字符串每个前缀和串匹配成功的次数和 (HDU3336)

```

1  /*
2      input  output
3      4      6
4      abab

```

```

5      6      12
6      ababab
7  */
8  int get_next(char x[], int x_len, int nxt[], int i, int j) {
9      while (~j && x[j + 1] != x[i]) j = nxt[j];
10     if (x[j + 1] == x[i]) j++;
11     nxt[i++] = j;
12     return j;
13 }
14
15 char str[MAXN];
16 int nex[MAXN], val[MAXN];
17
18 int main() {
19     int T;
20     scanf("%d", &T);
21     while (T--) {
22         int n;
23         scanf("%d", &n);
24         scanf("%s", str + 1);
25         int len = strlen(str + 1);
26         int last = -1;
27         nex[1] = -1;
28         int res = 0;
29         for (int i = 1; i <= len; i++) {
30             last = get_next(str + 1, len, nex + 1, i, last);
31             if (nex[i] < 0) val[i] = 1;
32             else val[i] = (val[nex[i] + 1] + 1) % mod;
33             res = (res + val[i]) % mod;
34         }
35         printf("%d\n", res);
36     }
37 }

```

### 2.2.5 求循环节数量 (POJ2406)

```

1  /*
2      input    output
3      abcd     1
4      aaaa     4
5      ababab   3
6  */
7  int main() {

```

```

8   while (~scanf("%s", s + 1)) {
9       if (s[1] == '.')break;
10      int len;
11      get_next(s + 1, len = strlen(s + 1), nex + 1); // 方法1
12      printf("%d\n", len % (len - nex[len] - 1) ? 1 : len / (len - nex[len]
13          - 1));
14  }

```

## 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)

```

1  /*
2      input      output
3      clinton    0
4      homer
5      riemann    rie 3
6      marjorie
7  */
8  int main() {
9      while (~scanf("%s%s", a + 1, b + 1)) {
10         int la = strlen(a + 1), lb = strlen(b + 1);
11         strcat(a + 1, b + 1);
12         int len = la + lb;
13         get_next(a + 1, len, nex + 1); // 方法1
14         int k;
15         for (k = nex[len]; k >= la || k >= lb; k = nex[k+1]);
16         if (k == -1) puts("0");
17         else {
18             for (int i = 0; i <= k; i++)printf("%c", a[i+1]);
19             printf(" %d\n", k + 1);
20         }
21     }
22 }

```

## 2.2.7 求补上最少字母数量使得这是个循环串 (HDU3746)

```

1  /*
2      input:  output
3      3
4      aaa     0
5      abca    2
6      abcde   5
7  */

```

```

8 int main() {
9     int T, len;
10    scanf("%d", &T);
11    while (T--) {
12        scanf("%s", s + 1);
13        get_next(s + 1, len = strlen(s + 1), nex + 1); // 方法1
14        int L = len - (nex[len] + 1);
15        if (L < len && len % L == 0) puts("0");
16        else printf("%d\n", L - len % L);
17    }
18 }

```

### 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： $O(n)$ 。

```

1 int main() {
2     int T;
3     scanf("%d", &T);
4     while (T--) {
5         scanf("%s", str + 1);
6         int len;
7         get_next(str, len = strlen(str + 1), nex); // 方法2
8         for (int i = 1; i <= len; i++) {
9             val[i] = (val[nex[i]] + 1) % mod;
10        }
11        ll res = 1ll;
12        for (int i = 2, j = 0; i <= len; i++) {
13            while (j && str[j + 1] != str[i]) j = nex[j];
14            if (str[j + 1] == str[i]) j++;
15            while (j * 2 > i) j = nex[j]; // 去除重叠的
16            num[i] = val[j];
17            // res = (res * ((1ll) val[j] + 1ll)) % mod;
18        }
19        printf("%lld\n", res);
20    }
21 }

```

## 2.3 KMP

### 2.3.1 统计模式串出现次数，出现位置，前缀 **border** 长度

```
1  /*
2      时间复杂度O(x_len + y_len)，空间复杂度O(x_len)
3  */
4  int nex[MAXN];
5  // x为模式串， y为文本串
6  // call: scanf("%s %s", a+1, b+1); kmp(b+1, strlen(b+1), a+1, strlen(a+1));
7  int kmp(char x[], int x_len, char y[], int y_len) {
8      int i, j;
9      int ans = 0;
10     get_next(x, x_len, nex);    // 方法1
11     for (j = -1, i = 0; i < y_len; i++) {
12         while (~j && x[j + 1] != y[i]) j = nex[j];
13         if (x[j + 1] == y[i]) j++;
14         if (j == x_len - 1) {
15             printf("%d\n", i - x_len + 2); // 出现的位置，可选，从1开始计数
16             ans++, j = nex[j];
17         }
18     }
19     for (i = 0; i < x_len; i++) printf("%d ", nex[i] + 1); //
20     // 每个前缀的最长border的长度
21     return ans;
22 }
```

### 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

```
1  /*
2      kmp+矩阵加速
3      求长度为n的不包含长度为m的子串的串个数
4      input   4 3 100
5              111
6      output  81
7  */
8  int nex[MAXN];
9  mat get_g(char x[], int x_len) {
```

```

10     get_next(x, x_len, nex);
11     mat g = mat(x_len, x_len);
12     for (int i = 0; i < x_len; i++) {
13         for (char ch = '0'; ch <= '9'; ch++) {
14             int j = i;
15             while (j && x[j+1] != ch) j = nex[j];
16             if (x[j+1] == ch) j++;
17             g.v[i][j] = (ll)(g.v[i][j] + 1ll) % mod;
18         }
19     }
20     return g;
21 }
22 int n, m;
23 char str[MAXN];
24 int main() {
25     scanf("%d%d%lld", &n, &m, &mod);
26     scanf("%s", str + 1);
27     mat g = get_g(str, strlen(str+1));
28     g = g^n;
29     mat f(m, 1);
30     f.v[0][0] = 1;
31     f = f*g;
32     ll res = 0;
33     for (int i = 0; i < m; i++) {
34         res = (ll)(res + f.v[0][i]) % mod;
35     }
36     printf("%lld\n", res);
37 }

```

## 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:  $O(|a| + |b|)$ 。

```

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

```

```

9 void getLCP(char T[], int T_len, int z[]) {
10     int i, len = T_len;
11     z[0] = len;
12     for (i = 0; i < len - 1 && T[i] == T[i + 1]; i++);
13     z[1] = i;
14     int a = 1;
15     for (int k = 2; k < len; k++) {
16         int p = a + z[a] - 1, L = z[k - a];
17         if ((k - 1) + L >= p) {
18             int j = max((p - k + 1), 0);
19             while (k + j < len && T[k + j] == T[j])j++;
20             z[k] = j, a = k;
21         } else z[k] = L;
22     }
23 }
24
25 // call: scanf("%s%s", a+1, b+1); exkmp(a+1, strlen(a+1), b+1, strlen(b+1),
26 //      ex+1, z+1);
27 void exkmp(char S[], int S_len, char T[], int T_len, int extend[], int z[]) {
28     getLCP(T, T_len, z);
29     int a = 0;
30     int MinLen = min(S_len, T_len);
31     while (a < MinLen && S[a] == T[a])a++;
32     extend[0] = a, a = 0;
33     for (int k = 1; k < S_len; k++) {
34         int p = a + extend[a] - 1, L = z[k - a];
35         if ((k - 1) + L >= p) {
36             int j = max((p - k + 1), 0);
37             while (k + j < S_len && j < T_len && S[k + j] == T[j])j++;
38             extend[k] = j;
39             a = k;
40         } else extend[k] = L;
41     }
42 }

```

## 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：  $O(n)$

```

1  /*
2      input    output
3      1        Case 1: 1 1 1
4      341

```



```

5  */
6  char str[MAXN];
7  int z[MAXN];
8  int main() {
9      int T; scanf("%d", &T);
10     int kase = 1;
11     while (T--) {
12         scanf("%s", str + 1);
13         int len = strlen(str + 1);
14         for (int i = 1; i <= len; i++) str[len + i] = str[i];
15         getLCP(str + 1, len * 2, z + 1);
16         int L = 0, E = 0, G = 0;
17         for (int i = 1; i <= len; i++) {
18             if (z[i] >= len) E++;
19             else if (str[z[i]+1] > str[z[i]+i]) L++;
20             else G++;
21         }
22         printf("Case %d: ", kase++);
23         printf("%d %d %d\n", L/E, E/E, G/E);    // 去重相关
24     }
25 }

```

## 2.5 AC 自动机

### 2.5.1 标准的 AC 自动机

```

1  const int MAXN = 1e5 + 5;
2
3  class AC_Automaton {
4  public:
5      int T[MAXN][26], val[MAXN], top;    //Trie相关
6      int fail[MAXN];
7      queue<int> q;
8      // int pid[SIZE];    //对应字符串编号
9
10     void init() {
11         top = 1;
12         memset(T[0], 0, sizeof(T[0]));
13         memset(val, 0, sizeof(val));
14         // memset(pid, 0, sizeof(pid));
15     }
16
17     AC_Automaton() {
18         init();
19     }
20 }

```

```

19 }
20
21 void insert(char str[], int lenstr, int _pid) {
22     int u = 0;
23     for (int i = 1; i <= lenstr; i++) {
24         int ch = str[i] - 'a';
25         if (!T[u][ch]) {
26             memset(T[top], 0, sizeof(T[top]));
27             T[u][ch] = top++;
28         }
29         u = T[u][ch];
30     }
31     val[u]++;
32     // pid[u] = _pid;
33 }
34
35 void build() {
36     for (int i = 0; i < 26; i++)
37         if (T[0][i]) {
38             fail[T[0][i]] = 0;
39             q.push(T[0][i]);
40         }
41     while (!q.empty()) {
42         int u = q.front();
43         q.pop();
44         for (int i = 0; i < 26; i++)
45             if (T[u][i]) {
46                 fail[T[u][i]] = T[fail[u]][i];
47                 q.push(T[u][i]);
48             } else T[u][i] = T[fail[u]][i];
49     }
50 }
51
52 int query(char str[], int lenstr) {
53     int u = 0, ans = 0;
54     for (int i = 1; i <= lenstr; i++) {
55         int id = str[i] - 'a';
56         u = T[u][id];
57         for (int j = u; j && (~val[j]); j = fail[j]) {
58             ans += val[j]; //val[j]=-1;
59             if (pid[j]) {
60                 qs[pid[j]].cnt++;
61             }
62         }

```

```

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 =
12         // strlen(str+1), tree.insert(str, len);
13         int u = 0;
14         for (int i = 1; i <= lenstr; i++) {
15             int ch = str[i] - 'a';
16             if (!T[u][ch]) {
17                 memset(T[top], 0, sizeof(T[top]));
18                 T[u][ch] = top++;
19             }
20             u = T[u][ch];
21         }
22     }
23
24     int search(char str[], int lenstr) {
25         int u = 0;
26         for (int i = 1; i <= lenstr; i++) {
27             int ch = str[i] - 'a';
28             if (!T[u][ch]) return -1; //找不到
29             u = T[u][ch];
30         }
31         if (!val[u]) {
32             val[u] = 1;
33             return 0;
34         }
35         return val[u];

```

```

35     }
36 } tree;

```

## 2.7 后缀数组 SA

### 2.7.1 获取 SA 和 rank 数组

```

1  /*
2      [input] aaabaabaaaab
3      [output]
4      rank    Suffix          pos(sa)    index
5      1       aaaab          8           1
6      2       aaab           9           2
7      3       aaabaabaaaab   1           3
8      4       aab            10          4
9      5       aabaaaab       5           5
10     6       aabaabaaaab    2           6
11     7       ab             11          7
12     8       abaaaab        6           8
13     9       abaabaaaab     3           9
14    10       b              12          10
15    11       baaaab         7           11
16    12       baabaaaab      4           12
17 */
18 namespace SA { // private ver.
19     int len;
20     int sa[MAXN], rk[MAXN << 1], oldrk[MAXN << 1], id[MAXN], cnt[MAXN];
21
22     void run(char s[], int _len) { // call: run(str, strlen(str)+1);
23         len = _len;
24         int m = max(len, 300);
25         // memset(cnt, 0, sizeof(cnt));
26         for (int i = 0; i <= m; i++) cnt[i] = 0;
27         for (int i = 1; i <= len; i++) ++cnt[rk[i] = s[i]];
28         for (int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
29         for (int i = len; i >= 1; i--) sa[cnt[rk[i]]--] = i;
30
31         for (int w = 1; w <= len; w <= 1) {
32             // memset(cnt, 0, sizeof(cnt));
33             for (int i = 0; i <= m; i++) cnt[i] = 0;
34             for (int i = 1; i <= len; i++) id[i] = sa[i];
35             for (int i = 1; i <= len; i++) ++cnt[rk[id[i] + w]];
36             for (int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
37             for (int i = len; i >= 1; i--) sa[cnt[rk[id[i] + w]]--] = id[i];

```

```

38     for (int i = 0; i <= m; i++) cnt[i] = 0;
39     // memset(cnt, 0, sizeof(cnt));
40     for (int i = 1; i <= len; i++) id[i] = sa[i];
41     for (int i = 1; i <= len; i++) ++cnt[rk[id[i]]];
42     for (int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
43     for (int i = len; i >= 1; i--) sa[cnt[rk[id[i]]]--] = id[i];
44     // memcpy(olldrk, rk, sizeof(rk));
45     for (int i = 0; i <= len; i++) oldrk[i] = rk[i];
46     for (int p = 0, i = 1; i <= len; i++) {
47         if (olldrk[sa[i]] == oldrk[sa[i - 1]] && oldrk[sa[i] + w] ==
48             oldrk[sa[i - 1] + w]) rk[sa[i]] = p;
49         else rk[sa[i]] = ++p;
50     }
51 }
52 }
53
54 namespace SA { // 77 ver.
55     int len;
56     int sa[MAXN], rk[MAXN], oldrk[MAXN << 1], id[MAXN], cnt[MAXN], px[MAXN];
57
58     bool cmp(int x, int y, int w) {
59         return oldrk[x] == oldrk[y] && oldrk[x + w] == oldrk[y + w];
60     }
61
62     void run(char s[], int _len) { // call: run(str, strlen(str)+1);
63         int i, m = 300, p, w;
64         len = _len;
65         // memset(cnt, 0, sizeof(cnt));
66         for (i = 1; i <= m; i++) cnt[i] = 0;
67         for (i = 1; i <= len; i++) ++cnt[rk[i] = s[i]];
68         for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
69         for (i = len; i >= 1; i--) sa[cnt[rk[i]]--] = i;
70
71         for (w = 1; w <= len; w <= 1, m = p) {
72             for (p = 0, i = len; i > len - w; i--) id[++p] = i;
73             for (i = 1; i <= len; i++)
74                 if (sa[i] > w) id[++p] = sa[i] - w;
75
76             // memset(cnt, 0, sizeof(cnt));
77             for (i = 0; i <= m; i++) cnt[i] = 0;
78             for (i = 1; i <= len; i++) ++cnt[px[i] = rk[id[i]]];
79             for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
80             for (i = len; i >= 1; i--) sa[cnt[px[i]]--] = id[i];

```

```

81
82     // memcpy(olldr, rk, sizeof(rk));
83     for (i = 0; i <= len; i++) oldrk[i] = rk[i];
84     for (p = 0, i = 1; i <= len; i++) {
85         rk[sa[i]] = cmp(sa[i], sa[i - 1], w) ? p : ++p;
86     }
87 }
88 }
89 }

```

## 2.7.2 后缀数组 +ST 表求 lcp

```

1  int height[MAXN];
2  namespace SA { // private ver.
3      int len;
4      int sa[MAXN], rk[MAXN << 1], oldrk[MAXN << 1], id[MAXN], cnt[MAXN];
5
6      void run(char s[], int _len) {} // step1 call: run(str, strlen(str)+1));
7
8      void get_height(char s[]) { // step2 call: get_height(str)
9          int k = 0;
10         for (int i = 1; i <= len; i++) rk[sa[i]] = i;
11         for (int i = 1; i <= len; i++) {
12             if (rk[i] == 1) continue;
13             if (k) --k;
14             int j = sa[rk[i] - 1];
15             while (j + k <= len && i + k <= len && s[i + k] == s[j + k]) k++;
16             height[rk[i]] = k;
17         }
18     }
19 }
20 const int MAXL = 22;
21 namespace RMQ { // ST, O(1) get LCP
22     int mm[MAXN], best[MAXL][MAXN];
23
24     void init(int n) { // step3 call: init(strlen(str)+1)
25         mm[0] = -1;
26         for (int i = 1; i <= n; i++)
27             mm[i] = ((i & (i - 1)) == 0) ? mm[i - 1] + 1 : mm[i - 1];
28         for (int i = 1; i <= n; i++) best[0][i] = i;
29         for (int i = 1; i <= mm[n]; i++)
30             for (int j = 1; j + (1 << i) - 1 <= n; j++) {
31                 int a = best[i - 1][j];

```

```

32         int b = best[i - 1][j + (1 << (i - 1))];
33         if (height[a] < height[b])best[i][j] = a;
34         else best[i][j] = b;
35     }
36 }
37
38 int askRMQ(int a, int b) {
39     int t = mm[b - a + 1];
40     b -= (1 << t) - 1;
41     a = best[t][a];
42     b = best[t][b];
43     return height[a] < height[b] ? a : b;
44 }
45
46 /*
47     get_SA.cpp example's index
48     get_LCP(2, 4) => 2
49 */
50 int get_LCP(int a, int b) {
51     if (a == b)return INF;
52     if (a > b)swap(a, b);
53     return height[askRMQ(a + 1, b)];
54 }
55
56 }

```

## 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  $link$  节点的对应值  $\frac{maxlen[link[i]]*(maxlen[link[i]]+1)}{2}$  就是该节点的净贡献

```

1 class Suffix_Automaton {
2 public:
3     int rt, link[MAXN], maxlen[MAXN], trans[MAXN][MAXC];
4     int val[MAXN]; // 用于统计某一串出现的次数
5

```

```

6 void init() {
7     rt = 1;
8     link[1] = maxlen[1] = 0;
9     memset(trans[1], 0, sizeof(trans[1]));
10 }
11
12 Suffix_Automaton() { init(); }
13
14 inline int insert(int ch, int last) { // main: last = 1
15     if (trans[last][ch]) {
16         int p = last, x = trans[p][ch];
17         // 注意：这里返回的这个节点保存了多个模式串的状态，
18         // 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，
19         // 如果要记录endpos大小的话，
20         // 则需要给每个模式串都单独维护一个siz数组依次更新，
21         // 而不能全部揉成一坨
22         if (maxlen[p] + 1 == maxlen[x]) { //
23             特判1：这个节点已经存在于SAM中
24             val[x]++; // 统计在整颗字典树上出现次数
25             return x;
26         }
27         else {
28             int y = ++rt;
29             maxlen[y] = maxlen[p] + 1;
30             for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
31             while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
32             link[y] = link[x], link[x] = y;
33             val[y]++; // 统计在整颗字典树上出现次数
34             return y;
35         }
36     }
37     int z = ++rt, p = last;
38     val[z] = 1; // 统计在整颗字典树上出现次数
39     memset(trans[z], 0, sizeof(trans[z]));
40     maxlen[z] = maxlen[last] + 1;
41     while (p && !trans[p][ch]) trans[p][ch] = z, p = link[p];
42     if (!p) link[z] = 1;
43     else {
44         int x = trans[p][ch];
45         if (maxlen[p] + 1 == maxlen[x]) link[z] = x;
46         else {
47             int y = ++rt;
48             maxlen[y] = maxlen[p] + 1;
49             for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];

```



```

49         while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
50         link[y] = link[x], link[z] = link[x] = y;
51     }
52 }
53 ans_1 += maxlen[z] - maxlen[link[z]]; // 【应用1】统计不同字符串个数
54 return z;
55 }
56
57 struct Edge {
58     int to, nex;
59 } e[MAXN << 1];
60 int head[MAXN], tol;
61
62 void addEdge(int u, int v) {
63     e[tol].to = v; e[tol].nex = head[u]; head[u] = tol; tol++;
64 }
65
66
67 /* 统计出现次数为k的字符串个数
68 input          output
69 2              (输入组数)
70 2              6
71 abcababc
72 3              9
73 abcabcabcabc
74 */
75 ll ans = 0;
76 void dfs(int u, int k) {
77     for (int i = head[u]; ~i; i = e[i].nex) {
78         int v = e[i].to;
79         dfs(v, k);
80         val[u] += val[v];
81     }
82     if (val[u] == k) { // val为出现次数
83         ans += 1ll * (maxlen[u] - maxlen[link[u]]); //
            以当前状态st为结尾的长度
84     }
85 }
86
87 int build(int k) {
88     tol = 0;
89     for (int i = 0; i <= rt; i++) head[i] = -1;
90     for (int i = 2; i <= rt; i++) addEdge(link[i], i); // 建fail树
91     dfs(1, k);

```

```

92     }
93 } 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  $O(\sum |S| \sqrt{\sum |S|})$

```

1  /*
2      input      output
3      3 3
4      abcabcabc
5      aaa
6      aafe
7      abc      1
8      a      3
9      ca      1
10 */
11 class Suffix_Automaton {
12 public:
13     int rt, link[MAXN], maxlen[MAXN], trans[MAXN][MAXC];
14     int val[MAXN];
15
16     void init() {
17         rt = 1;
18         link[1] = maxlen[1] = 0;
19         memset(trans[1], 0, sizeof(trans[1]));
20     }
21
22     Suffix_Automaton() { init(); }
23
24     inline int insert(int ch, int last) { // main: last = 1
25         if (trans[last][ch]) {
26             int p = last, x = trans[p][ch];
27             if (maxlen[p] + 1 == maxlen[x]) { //
28                 特判1: 这个节点已经存在于SAM中
29                 return x;
30             } else {
31                 int y = ++rt;
32                 maxlen[y] = maxlen[p] + 1;
33                 for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
34                 while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
35                 link[y] = link[x], link[x] = y;
36                 return y;
37             }
38         }
39     }
40 }

```

```

36         }
37     }
38     int z = ++rt, p = last;
39     memset(trans[z], 0, sizeof(trans[z]));
40     maxlen[z] = maxlen[last] + 1;
41     while (p && !trans[p][ch]) trans[p][ch] = z, p = link[p];
42     if (!p) link[z] = 1;
43     else {
44         int x = trans[p][ch];
45         if (maxlen[p] + 1 == maxlen[x]) link[z] = x;
46         else {
47             int y = ++rt;
48             maxlen[y] = maxlen[p] + 1;
49             for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
50             while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
51             link[y] = link[x], link[z] = link[x] = y;
52         }
53     }
54     return z;
55 }
56 } sa;
57
58 char str_sum[MAXN];
59 char str[MAXN];
60 int len[MAXN];
61
62 int las[MAXN]; // 记得清空las数组
63
64 inline void update(int x, int id) { // 暴力跳Link链
65     for (; x && las[x] != id; x = sa.link[x]) {
66         sa.val[x]++;
67         las[x] = id;
68     }
69 }
70
71 int main() {
72     int n, q;
73     scanf("%d%d", &n, &q);
74     int tot = 0;
75     for (int i = 1; i <= n; i++) {
76         scanf("%s", str + 1);
77         int last = 1;
78         len[i] = strlen(str + 1);
79         for (int j = 1; j <= len[i]; j++) {

```

```

80         str_sum[++tot] = str[j];
81         last = sa.insert(str[j] - 'a', last);
82     }
83 }
84 sa.debug();
85 tot = 0;
86 for (int i = 1; i <= n; i++) {
87     for (int j = 1, x = 1; j <= len[i]; j++) {
88         update(x = sa.trans[x][str_sum[++tot] - 'a'], i);
89     }
90 }
91 while (q--) {
92     scanf("%s", str + 1);
93     len[0] = strlen(str + 1);
94     int flag = 1, u = 1;
95     for (int i = 1; i <= len[0]; i++) {
96         int ch = str[i] - 'a';
97         if (sa.trans[u][ch]) {
98             u = sa.trans[u][ch];
99         } else {
100             flag = 0; break;
101         }
102     }
103     if (flag) printf("%d\n", sa.val[u]);
104     else printf("0\n");
105 }
106 }

```

### 2.8.3 第 k 小字符串

```

1  /*
2      input    output
3      aabc     aab
4      0 3
5      aabc     aa
6      1 3
7      aabc     -1
8      1 11
9  */
10 const int MAXN = 1e6 + 5;
11 const int MAXC = 26;
12
13 class Suffix_Automaton { public:

```

```

14 int rt, link[MAXN], maxlen[MAXN], trans[MAXN][MAXC];
15 int val[MAXN];
16
17 void init() {
18     rt = 1;
19     link[1] = maxlen[1] = 0;
20     memset(trans[0], 0, sizeof(trans[0]));
21 }
22 Suffix_Automaton() { init(); }
23
24 inline int insert(int ch, int last) { // main: last = 1
25     if (trans[last][ch]) {
26         int p = last, x = trans[p][ch];
27         if (maxlen[p] + 1 == maxlen[x]) return x;
28         else {
29             int y = ++rt;
30             maxlen[y] = maxlen[p] + 1;
31             for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
32             while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
33             link[y] = link[x], link[x] = y;
34             return y;
35         }
36     }
37     int z = ++rt, p = last;
38     val[z] = 1; // dfs树统计出现次数
39     maxlen[z] = maxlen[last] + 1;
40     while (p && !trans[p][ch]) trans[p][ch] = z, p = link[p];
41     if (!p) link[z] = 1;
42     else {
43         int x = trans[p][ch];
44         if (maxlen[p] + 1 == maxlen[x]) link[z] = x;
45         else {
46             int y = ++rt;
47             maxlen[y] = maxlen[p] + 1;
48             for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
49             while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
50             link[y] = link[x], link[z] = link[x] = y;
51         }
52     }
53     return z;
54 }
55
56 int sz1[MAXN], sz2[MAXN];
57 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[maxn[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[maxn[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

```

```

102     for (int i = rt; i >= 1; i--) {
103         sz2[topo_id[i]] = val[topo_id[i]];
104         if (topo_id[i] == 1) sz2[topo_id[i]] = 0;
105         for (int j = 0; j < MAXC; j++) {
106             int v = trans[topo_id[i]][j];
107             if (!v) continue;
108             sz2[topo_id[i]] += sz2[v];
109         }
110     }
111 }
112
113 void query0(int k) {    // 不同位置的相同子串算作一个
114     if (sz1[1] < k) {
115         printf("-1\n"); return ;
116     }
117     int u = 1;
118     while (k) {
119         for (int i = 0; i < MAXC; i++) {
120             if (trans[u][i]) {
121                 if (sz1[trans[u][i]] >= k) {
122                     printf("%c", 'a' + i);
123                     u = trans[u][i];
124                     k--;
125                     break;
126                 } else k -= sz1[trans[u][i]];
127             }
128         }
129     }
130     printf("\n");
131 }
132
133 void query1(int k) {    // 不同位置的相同子串算作多个
134     if (sz2[1] < k) {
135         printf("-1\n"); return ;
136     }
137     int u = 1;
138     while(k > 0) {
139         for (int i = 0; i < MAXC; i++) {
140             if (trans[u][i]) {
141                 int v = trans[u][i];
142                 if (sz2[v] >= k) {
143                     printf("%c", 'a'+i);
144                     u = v;
145                     k -= val[v];

```

```

146         break;
147     } else k -= sz2[v];
148 }
149 }
150 }
151 printf("\n");
152 }
153 } sa;

```

## 2.8.4 字典树建后缀自动机

```

1 void bfs() { // 传说常数小
2     queue<node> q;
3     q.push(node(1, 1));
4     int last = 1;
5     while (!q.empty()) {
6         node u = q.front(); q.pop();
7         int nls = sa.insert(str[u.v] - 'A', u.last);
8         pos[u.v] = nls;
9         for (int i = head[u.v]; ~i; i = e[i].nex) {
10             int to = e[i].to;
11             q.push(node(to, nls));
12         }
13     }
14 }
15
16 void dfs(int u, int last = 1) { // 传说常数大
17     pos[u] = last = sa.insert(str[u] - 'A', last);
18     for (int i = head[u]; ~i; i = e[i].nex) {
19         int v = e[i].to;
20         dfs(v, last);
21     }
22 }

```

## 2.8.5 暴力在线统计出现次数为 $k$ 次的字符串个数 (HDU4641)

```

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

```



```

8  int ans = 0;
9  void init() {
10     rt = 1;
11     link[1] = maxlen[1] = 0;
12     memset(trans[1], 0, sizeof(trans[1]));
13     ans = 0;
14 }
15
16 Suffix_Automaton() { init(); }
17
18 inline int insert(int ch, int last) { // main: last = 1
19     if (trans[last][ch]) {
20         int p = last, x = trans[p][ch];
21         if (maxlen[p] + 1 == maxlen[x]) return x;
22         else {
23             int y = ++rt;
24             maxlen[y] = maxlen[p] + 1;
25             val[y] = val[x];
26             for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
27             while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
28             link[y] = link[x], link[x] = y;
29             return y;
30         }
31     }
32     int z = ++rt, p = last;
33     val[z]=0;
34     memset(trans[z], 0, sizeof(trans[z]));
35     maxlen[z] = maxlen[last] + 1;
36     while (p && !trans[p][ch]) trans[p][ch] = z, p = link[p];
37     if (!p) link[z] = 1;
38     else {
39         int x = trans[p][ch];
40         if (maxlen[p] + 1 == maxlen[x]) link[z] = x;
41         else {
42             int y = ++rt;
43             maxlen[y] = maxlen[p] + 1;
44             val[y] = val[x];
45             for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
46             while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
47             link[y] = link[x], link[z] = link[x] = y;
48         }
49     }
50     int t = z;
51     while (t && val[t] < K) {

```

```

52         val[t]++;
53         if (val[t] >= K) ans+=maxlen[t] - maxlen[link[t]];
54         t = link[t];
55     }
56     return z;
57 }
58
59 } sa;
60 char str[MAXN];
61 int main() {
62     int n, m;
63     while(~scanf("%d%d%d", &n, &m, &K)) {
64         sa.init();
65         scanf("%s", str + 1);
66         int last = 1;
67         for (int i = 1; i <= n; i++) last = sa.insert(str[i] - 'a', last);
68         while (m--) {
69             int op;
70             scanf("%d", &op);
71             if (op == 1) {
72                 getchar();
73                 char ch;
74                 scanf("%c", &ch);
75                 last = sa.insert(ch - 'a', last);
76             } else {
77                 printf("%d\n", sa.ans);
78             }
79         }
80     }
81 }

```

## 2.9 回文自动机 PAM

```

1  int pos[MAXN]; // +1后对应为所建的fail树上的点
2  class Palindrome_Tree { public:
3      struct node {
4          int ch[MAXC], fail, len, num; // num: 以该位置结尾的回文子串个数
5      } T[MAXN];
6      int las, tot, c[MAXN];
7      inline int get_fail(int x, int pos) {
8          while (c[pos - T[x].len - 1] != c[pos]) {
9              x = T[x].fail;
10         }

```

```

11     return x;
12 }
13 void init() { // 传入字符串长度
14     memset(T[0].ch, 0, sizeof(T[0].ch));
15     memset(T[1].ch, 0, sizeof(T[1].ch));
16     T[0].len = 0, T[1].len = -1;
17     T[0].fail = 1, T[1].fail = 0;
18     las = 0, tot = 1;
19 }
20 void insert(char s[], int len) { // call: insert(str, strlen(str+1));
21     c[0] = -1;
22     for (int i = 1; i <= len; i++) {
23         c[i] = s[i] - 'a';
24         int p = get_fail(las, i);
25         if (!T[p].ch[c[i]]) {
26             T[++tot].len = T[p].len + 2;
27             memset(T[tot].ch, 0, sizeof(T[tot].ch));
28             int u = get_fail(T[p].fail, i);
29             T[tot].fail = T[u].ch[c[i]];
30             T[tot].num = T[T[tot].fail].num + 1;
31             T[p].ch[c[i]] = tot;
32         }
33         las = T[p].ch[c[i]];
34         pos[i] = las;
35     }
36 }
37 struct Edge {
38     int to, nex;
39 } e[MAXN << 1];
40 int head[MAXN], tol;
41 int len[MAXN];
42 void addEdge(int u, int v) {
43     e[tol].to = v, e[tol].nex = head[u], head[u] = tol, tol++;
44 }
45 void build() { // build tree
46     tol = 0;
47     for (int i = 0; i <= tot; i++) head[i+1] = -1;
48     for (int i = 1; i <= tot; i++) addEdge(T[i].fail + 1, i + 1);
49     for (int i = 0; i <= tot; i++) len[i + 1] = T[i].len;
50 }
51 } tree;
52
53 char str[MAXN];
54 int main() {

```

```

55     scanf("%s", str + 1);
56     tree.init();
57     tree.insert(str, strlen(str + 1));
58     tree.build();
59     for (int i = 1; i <= len; i++) printf("%d ", pos[i]+1); //
        fail 树上对应位置
60 }

```

## 2.10 序列自动机 ([HEOI2015] 最短不公共子串)

时间复杂度:  $O(n|\Sigma|)$ , 其中  $|\Sigma|$  为字符集大小

```

1  /*
2      input      output
3      aabbcc      2 4 2 4
4      abcabc
5      aabbcc      -1 -1 2 -1
6      aabbcc
7  */
8  char str1[MAXN], str2[MAXN];
9  int nex[MAXC], na[MAXN][MAXC], nb[MAXN][MAXC];
10 int dp[MAXN][MAXN]; // 记得开大两倍
11 int main() {
12     scanf("%s%s", str1 + 1, str2 + 1);
13     int len1 = strlen(str1 + 1), len2 = strlen(str2 + 1);
14     int last = 1;
15     for (int i = 1; i <= len2; i++) last = sa.insert(str2[i] - 'a', last);
        // 调用后缀自动机
16     for (int i = 0; i < 26; i++) nex[i] = len1 + 1;
17     for (int i = len1; i >= 0; i--) {
18         memcpy(na[i], nex, sizeof(nex));
19         nex[str1[i] - 'a'] = i;
20     }
21     for (int i = 0; i < 26; i++) nex[i] = len2 + 1;
22     for (int i = len2; i >= 0; i--) {
23         memcpy(nb[i], nex, sizeof(nex));
24         nex[str2[i] - 'a'] = i;
25     }
26     int res = MAXN;
27     for (int l = 1; l <= len1; l++) {
28         for (int r = l, u = 1; r <= len1; r++) {
29             u = sa.trans[u][str1[r] - 'a'];
30             if (!u) {
31                 res = min(res, r - l + 1);

```

```

32         break;
33     }
34 }
35 }
36 printf("%d\n", res == MAXN ? -1 : res); //
    str1的一个最短的子串，它不是str2的子串。
37 res = MAXN;
38 for (int l = 1; l <= len1; l++) {
39     for (int r = l, u = 0; r <= len1; r++) {
40         u = nb[u][str1[r] - 'a'];
41         if (u == len2 + 1) {
42             res = min(res, r - l + 1);
43             break;
44         }
45     }
46 }
47 printf("%d\n", res == MAXN ? -1 : res); //
    str1的一个最短的子串，它不是str2的子序列。
48 for (int i = len1; i >= 0; i--) {
49     for (int j = 1; j <= sa.rt; j++) {
50         dp[i][j] = MAXN;
51         for (int ch = 0; ch < MAXC; ch++) {
52             int u = na[i][ch], v = sa.trans[j][ch];
53             if (u <= len1) dp[i][j] = min(dp[i][j], dp[u][v] + 1);
54         }
55     }
56 }
57 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。
58 memset(dp, 0, sizeof(dp));
59 for (int i = len1; i >= 0; i--) {
60     for (int j = 0; j <= len2; j++) {
61         dp[i][j] = MAXN;
62         for (int ch = 0; ch < MAXC; ch++) {
63             int u = na[i][ch], v = nb[j][ch];
64             if (u <= len1) dp[i][j] = min(dp[i][j], dp[u][v] + 1);
65         }
66     }
67 }
68 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。
69 }

```

## 2.11 最小表示法

时间复杂度:  $O(n)$

```
1  /*
2      input
3      10
4      10 9 8 7 6 5 4 3 2 1
5      output
6      1 10 9 8 7 6 5 4 3 2
7  */
8  int Min_show(int arr[], int n) {
9      int i = 0, j = 1, k = 0;
10     while (i < n && j < n && k < n) {
11         if (arr[(i + k) % n] == arr[(j + k) % n]) k++;
12         else {
13             if (arr[(i + k) % n] > arr[(j + k) % n]) i += k + 1;
14             else j += k + 1;
15             if (i == j) i++;
16             k = 0;
17         }
18     }
19     return min(i, j);
20 }
21 int main() {
22     int n;
23     scanf("%d", &n);
24     for (int i = 0; i < n; i++) scanf("%d", &a[i]);
25     int ans = Min_show(a,n);
26     for (int i = 0; i < n; i++) printf("%d ", a[(i + ans) % n]);
27 }
```

## 2.12 Lyndon 分解

将字符串分成若干部分  $s = s_1 s_2 s_3 \dots s_m$ , 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。

```
1  /*
2      input                output
3      ababa                2 4 5
4      ababa = ab + ab + a
5      bbababaabaaabaaab  1 2 4 6 9 13 18
6      bbababaabaaabaaab
7      = b + b + ab + ab + aab + aaaab + aaaab
8  */
```

```

9 vector<int> Lyndon_Arr; // 分解后的串的右端点
10 // call: Lyndon_Word(str, strlen(str+1));
11 void Lyndon_Word(char s[], int s_len) {
12     int i = 1;
13     while (i <= s_len) {
14         int j = i, k = j + 1;
15         while (k <= s_len && s[j] <= s[k]) {
16             if (s[j] < s[k]) j = i;
17             else j++;
18             k++;
19         }
20         while (i <= j) {
21             Lyndon_Arr.push_back(i-j+k-1);
22             i += k - j;
23         }
24     }
25 }

```

## 3 杂项

### 3.1 LCA

```

1 const int MAXLOG = 22;
2 struct Edge {
3     int to, nex;
4 } e[MAXM << 1];
5 int head[MAXN], tol;
6 void addEdge(int u, int v) {
7     e[tol].to = v, e[tol].nex = head[u], head[u] = tol, tol++;
8 }
9 namespace LCA {
10     int dep[MAXN], fa[MAXN][MAXLOG], lg[MAXN];
11     void init(int _n) { // n为点的个数, 最坏情况为一条链
12         for (int i = 1; i <= _n; i++) {
13             lg[i] = lg[i-1] + (1 << lg[i-1] == i);
14         }
15     }
16     void dfs(int u, int f) {
17         fa[u][0] = f; dep[u] = dep[f] + 1;
18         for (int i = 1; i <= lg[dep[u]]; i++) fa[u][i] = fa[fa[u][i-1]][i-1];
19         for (int i = head[u]; ~i; i = e[i].nex) {
20             int v = e[i].to;
21             if (v == f) continue;

```

```

22         dfs(v, u);
23     }
24 }
25 int LCA(int u, int v) {
26     if (dep[u] < dep[v]) swap(u, v);
27     while (dep[u] > dep[v]) u = fa[u][lg[dep[u]] - dep[v] - 1];
28     if (u == v) return u;
29     for (int k = lg[dep[u]] - 1; k >= 0; k--) {
30         if (fa[u][k] != fa[v][k]) u = fa[u][k], v = fa[v][k];
31     }
32     return fa[u][0];
33 }
34 }

```

## 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  $a_j \leq a_i$  且  $b_j \leq b_i$  且  $c_j \leq c_i$  且  $j \neq i$  的  $j$  的数量。

对于  $d \in [0, n)$ ，求  $f(i) = d$  的数量。

```

1  class TREE { public:
2      int T[MAXN], n;
3      inline int lowbit(int x) { return x & (-x); }
4      void add(int pos, int val) {
5          while (pos <= n) {
6              T[pos] += val; pos += lowbit(pos);
7          }
8      }
9      int query(int pos) {
10         int ans = 0;
11         while (pos) {
12             ans += T[pos]; pos -= lowbit(pos);
13         }
14         return ans;
15     }
16 } tree;
17
18 struct node {
19     int a, b, c;
20     int cnt, ans;
21     bool operator==(const node &tb) const {
22         if (a == tb.a && b == tb.b && c == tb.c) return 1;
23         else return 0;

```



```

24     }
25 } p1[MAXN], p2[MAXN];
26
27 void cdq(int l, int r) {
28     if (l == r) return;
29     int mid = (l + r) >> 1;
30     cdq(l, mid), cdq(mid + 1, r);
31     sort(p2 + l, p2 + mid + 1, [&](const node &x, const node &y) {
32         if (x.b != y.b) return x.b < y.b;
33         else return x.c < y.c;
34     });
35     sort(p2 + mid + 1, p2 + r + 1, [&](const node &x, const node &y) {
36         if (x.b != y.b) return x.b < y.b;
37         else return x.c < y.c;
38     });
39     int j = l;
40     for (int i = mid + 1; i <= r; i++) {
41         while (p2[i].b >= p2[j].b && j <= mid) {
42             tree.add(p2[j].c, p2[j].cnt);
43             j++;
44         }
45         p2[i].ans += tree.query(p2[i].c);
46     }
47     for (int i = l; i < j; i++) tree.add(p2[i].c, -p2[i].cnt);
48 }
49
50 int res[MAXN];
51 int main() {
52     int n, k; scanf("%d%d", &n, &k);
53     for (int i = 1; i <= n; i++) scanf("%d%d%d", &p1[i].a, &p1[i].b,
54         &p1[i].c);
55     p1[n+1].a = p1[n+1].b = p1[n+1].c = 0;
56     sort(p1 + 1, p1 + 1 + n, [&](const node &x, const node &y) {
57         if (x.a != y.a) return x.a < y.a;
58         if (x.b != y.b) return x.b < y.b;
59         return x.c < y.c;
60     });
61     int tot = 0, m = 0;
62     for (int i = 1; i <= n; i++) {
63         tot++;
64         if (p1[i].a != p1[i + 1].a || p1[i].b != p1[i + 1].b || p1[i].c !=
65             p1[i + 1].c) {
66             m++;
67             p2[m].a = p1[i].a, p2[m].b = p1[i].b, p2[m].c = p1[i].c,

```

```

        p2[m].cnt = tot;
66         tot = 0;
67     }
68 }
69 tree.n = k;
70 cdq(1, m);
71 for (int i = 1; i <= m; i++) res[p2[i].ans + p2[i].cnt - 1] += p2[i].cnt;
72 for (int d = 0; d < n; d++) printf("%d\n", res[d]); // 输出每个f(i) =
        d的数量
73 }
```