

Contents

1	Tools	2
1.1	vimrc	2
1.2	对拍	2
1.3	注意事项	2
2	Ds	2
2.1	链表	2
2.2	哈希表	2
2.3	并查集	2
2.4	树状数组	2
2.5	线段树	2
2.6	轻重链剖分	3
2.7	主席树	4
2.8	李超树	5
2.9	珂朵莉树	5
3	Dp	5
3.1	数位 DP	5
3.2	子集卷积/SOS DP	6
4	Math	6
4.1	线性求逆元以及组合数	6
4.2	快速幂	6
4.3	矩阵乘法	6
4.4	高斯消元	6
4.5	质数	6
4.6	约数	7
4.7	Exgcd	7
4.8	中国剩余定理	7
5	Graph	7
5.1	SPFA 以及负环	7
5.2	Dijkstra	8
5.3	Floyd 以及最小环	8
5.4	Kruskal	8
5.5	倍增 LCA	8
5.6	Tarjan LCA	8
5.7	拓扑排序	9
5.8	欧拉回路	9
5.9	强连通分量	9
5.10	边双连通分量	9
5.11	点双连通分量	10
5.12	虚树	10
6	String	10
6.1	Kmp	10
6.2	Trie	11
6.3	01Trie	11
6.4	Ac Automaton	12

1. Tools

1.1 vimrc

```

1 syntax on
2 color xterm16
3
4 set ts=4 sts=4 sw=4 ai
5 set wildmenu nocompatible nu rnu ruler mouse=a
6 set timeoutlen=666 ttimeoutlen=0 backup swapfile undofile
7 set ar acd backspace=indent,eol,start foldmethod=marker
8 set encoding=utf-8
9
10 nmap F :e ./<CR>
11 nmap <leader>n :tabnew<CR>
12 nmap <leader>tc :term<CR>
13
14 inoremap [ []<Esc>i
15 inoremap {<CR> {}<Esc>i<CR><Esc>O
16 inoremap ( )<Esc>i
17 inoremap ' ''<Esc>i
18 inoremap " ""<Esc>i

```

1.2 对拍

```

1 g++ gen.cpp -o gen -std=c++17 -Wall -O2
2 g++ ans.cpp -o ans -std=c++17 -Wall -O2
3 g++ usr.cpp -o ans -std=c++17 -Wall -O2
4 while true; do
5     ./gen > 1
6     ./ans < 1 > 2
7     ./usr < 1 > 3
8     diff 2 3
9     if [ $? -ne 0 ] ; then break; fi
10 done

```

1.3 注意事项

- Linux 下开大栈空间:如果 ulimit -a 是 unlimited, 那么写入 ulimit -s 65536; ulimit -m 1048576 即可。

2. Ds

2.1 链表

```

1 struct node {
2     int value;
3     node *next, *prev;
4 }, head, tail;
5 void init() {
6     head = new node(), tail = new node();
7     head -> next = tail, tail -> prev = head;
8 }
9 void insert(node *p, int val) {
10     node *q; q = new node();
11     q -> val = val, p -> next -> prev = q;
12     q -> next = p -> next, q -> prev = p, p -> next = q;
13 }
14 void remove(node *p) {
15     p -> prev -> next = p -> next,
16     | p -> next -> prev = p -> prev; delete p;
17 }

```

2.2 哈希表

```

1 const int si = 1e5 + 10;
2 int n, a[si], tot = 0;
3 int head[si], val[si], cnt[si], Next[si];
4
5 int H(int x) { return (x % p) + 1; }
6 bool insert(int x) {
7     bool exist = false;
8     int u = H(x);
9     for(int i = head[u]; ~i; i = Next[i]) {
10         if(val[i] == x) {
11             cnt[i]++; exist = true;
12             break;
13         }
14     }
15     if(exist) return true;
16     ++tot, Next[tot] = head[u], val[tot] = x, cnt[tot] = 1,
17     head[u] = tot;
18     return false;
19 }

```

```

19 int query(int x) {
20     int u = H(x);
21     for(int i = head[u]; ~i; i = Next[i])
22         if(val[i] == x) return cnt[i];
23     return 0;
24 }

```

2.3 并查集

```

1 int root(int x) {
2     if(pa[x] != x) pa[x] = root(pa[x]);
3     return pa[x];
4 }
5 void Merge(int x, int y) {
6     int rx = root(x), ry = root(y);
7     if(rx == ry) return;
8     if(siz[rx] < siz[ry])
9         pa[rx] = ry, siz[rx] += siz[ry];
10    else pa[ry] = rx, siz[ry] += siz[rx];
11 }
12 // remember to init!

```

2.4 树状数组

```

1 int lowbit(int x) { return x & -x; }
2 class Fenwick {
3     private:
4         int t[si], V;
5     public:
6         void init(int n) { V = n + 1; memset(t, 0, sizeof t);
7             }
8         void add(int x, int v) { while(x <= V) t[x] += v, x
9             += lowbit(x); }
10        int que(int x) { int ret = 0; while(x > 0) ret +=
11            t[x], x -= lowbit(x); return ret; }
12 } tr;

```

2.5 线段树

```

1 // {Lazytag} = {+}
2 class Segment_Tree {
3     private:
4         struct Node {
5             int l, r;
6             i64 dat, tag;
7         } t[si << 2];
8         inline void pushup(int p) {
9             t[p].dat = t[p << 1].dat + t[p << 1 | 1].dat;
10        }
11        inline void pushdown(int p) {
12            if(!t[p].tag) return;
13            t[p << 1].dat += 1ll * t[p].tag * (t[p << 1].r
14                - t[p << 1].l + 1);
15            t[p << 1 | 1].dat += 1ll * t[p].tag * (t[p << 1
16                | 1].r - t[p << 1 | 1].l + 1);
17            t[p << 1].tag += t[p].tag, t[p << 1 | 1].tag +=
18            t[p].tag, t[p].tag = 0;
19        }
20        public:
21            void build(int p, int l, int r) {
22                t[p].l = l, t[p].r = r, t[p].tag = 0;
23                if(l == r) {
24                    t[p].dat = a[l];
25                    return;
26                }
27                int mid = (l + r) >> 1;
28                build(p << 1, l, mid), build(p << 1 | 1, mid +
29                    1, r);
30                pushup(p); return;
31            }
32            void update(int p, int l, int r, int v) {
33                if(l <= t[p].l && t[p].r <= r) {
34                    t[p].dat += v * (t[p].r - t[p].l + 1);
35                    t[p].tag += v; return;
36                }
37                pushdown(p); // 没到可以直接返回的时候, 马上要递
38                归子树了, 也要 pushdown.
39                int mid = (t[p].l + t[p].r) >> 1;
40                if(l <= mid)
41                    update(p << 1, l, r, v);
42                if(r > mid)
43                    update(p << 1 | 1, l, r, v);
44                pushup(p); return;
45            }
46 }

```

```

40     }
41     i64 query(int p, int l, int r) {
42         i64 res = 0;
43         if(l <= t[p].l && t[p].r <= r)
44             return t[p].dat;
45         pushdown(p); // 查询要查值, 需要子树信息, 必然要
46         ↪ pushdown.
47         int mid = (t[p].l + t[p].r) >> 1;
48         if(l <= mid)
49             res += query(p << 1, l, r);
50         if(r > mid)
51             res += query(p << 1 | 1, l, r);
52         return res;
53     }
54 };
55 // {Lazytag} = {+, *}
56 class Segment_Tree {
57     private :
58         struct Node {
59             int l, r;
60             i64 dat, add, mul;
61         }t[si << 2];
62         inline void pushup(int p) {
63             t[p].dat = (t[p << 1].dat + t[p << 1 | 1].dat)
64             ↪ % mod;
65         }
66         inline void pushdown(int p) {
67             if(!t[p].add && t[p].mul == 1) return;
68             t[p << 1].dat = (t[p << 1].dat * t[p].mul +
69             ↪ t[p].add * (t[p << 1].r - t[p << 1].l + 1)) % mod ;
70             t[p << 1 | 1].dat = (t[p << 1 | 1].dat *
71             ↪ t[p].mul + t[p].add * (t[p << 1 | 1].r - t[p << 1 |
72             ↪ 1].l + 1)) % mod;
73             t[p << 1].mul = (t[p << 1].mul * t[p].mul) %
74             ↪ mod;
75             t[p << 1 | 1].mul = (t[p << 1 | 1].mul *
76             ↪ t[p].mul) % mod;
77             t[p << 1].add = (t[p << 1].add * t[p].mul +
78             ↪ t[p].add) % mod;
79             t[p << 1 | 1].add = (t[p << 1 | 1].add *
80             ↪ t[p].mul + t[p].add) % mod;
81             t[p].add = 0, t[p].mul = 1;
82         }
83     public :
84         void build(int p, int l, int r) {
85             t[p].l = l, t[p].r = r, t[p].mul = 1ll,
86             ↪ t[p].add = 0ll;
87             if(l == r) {
88                 t[p].dat = a[l] % mod;
89                 return;
90             }
91             int mid = (l + r) >> 1;
92             build(p << 1, l, mid), build(p << 1 | 1, mid +
93             ↪ 1, r);
94             pushup(p); return;
95         }
96         void update_add(int p, int l, int r, i64 v) {
97             if(l <= t[p].l && t[p].r <= r) {
98                 t[p].add = (t[p].add + v) % mod;
99                 t[p].dat = (t[p].dat + v * (t[p].r - t[p].l
100             ↪ + 1)) % mod;
101                 return;
102             }
103             pushdown(p);
104             int mid = (t[p].l + t[p].r) >> 1;
105             if(l <= mid)
106                 update_add(p << 1, l, r, v);
107             if(r > mid)
108                 update_add(p << 1 | 1, l, r, v);
109             pushup(p); return;
110         }
111         void update_mul(int p, int l, int r, i64 v) {
112             if(l <= t[p].l && t[p].r <= r) {
113                 t[p].add = (t[p].add * v) % mod;
114                 t[p].mul = (t[p].mul * v) % mod;
115                 t[p].dat = (t[p].dat * v) % mod;
116                 return;
117             }
118             pushdown(p);
119             int mid = (t[p].l + t[p].r) >> 1;
120             if(l <= mid)
121                 update_mul(p << 1, l, r, v);
122             if(r > mid)
123                 update_mul(p << 1 | 1, l, r, v);
124             pushup(p); return;
125         }
126     };
127 
```

```

110     pushdown(p);
111     int mid = (t[p].l + t[p].r) >> 1;
112     if(l <= mid)
113         update_mul(p << 1, l, r, v);
114     if(r > mid)
115         update_mul(p << 1 | 1, l, r, v);
116     pushup(p); return;
117 }
118 i64 query(int p, int l, int r) {
119     i64 res = 0ll;
120     if(l <= t[p].l && t[p].r <= r)
121         return t[p].dat % mod;
122     pushdown(p);
123     int mid = (t[p].l + t[p].r) >> 1;
124     if(l <= mid)
125         res = (res + query(p << 1, l, r)) % mod;
126     if(r > mid)
127         res = (res + query(p << 1 | 1, l, r)) %
128         ↪ mod;
129     return res;
130 }
131 };
132 Segment_Tree tr;
133 // 不要到主函数里定义, 容易爆栈。
134 // Merge
135 int merge(int p, int q, int l, int r) {
136     if(!p) return q;
137     if(!q) return p;
138     if(l == r){
139         t[p].mx += t[q].mx;
140         return p;
141     }
142     int mid = (l + r) >> 1;
143     t[p].ls = merge(t[p].ls, t[q].ls, l, mid);
144     t[p].rs = merge(t[p].rs, t[q].rs, mid + 1, r);
145     pushup(p); return p;
146 }
147 // SweepLine
148 void change(int p, int l, int r, int v) {
149     int nl = t[p].l, nr = t[p].r;
150     if(l <= nl && nr <= r) {
151         t[p].cnt += v;
152         if(t[p].cnt == 0)
153             t[p].len = (nl == nr) ? 0 : t[p << 1].len + t[p
154             ↪ << 1 | 1].len;
155         // 虽然当前区间直接被覆盖的次数等于 0 了, 但还是要
156         ↪ 考虑下面的子树, 因为它们有可能没被修改完。
157         else t[p].len = raw[nr + 1] - raw[nl];
158         return;
159     }
160     int mid = (nl + nr) >> 1;
161     if(l <= mid) change(p << 1, l, r, v);
162     if(r > mid) change(p << 1 | 1, l, r, v);
163     if(t[p].cnt > 0) t[p].len = raw[nr + 1] - raw[nl];
164     else t[p].len = t[p << 1].len + t[p << 1 | 1].len;
165 }
166 
```

2.6 轻重链剖分

```

1 // 处理重儿子, 父亲, 深度, 子树大小
2 void dfs1(int u, int fa) {
3     int kot = 0;
4     hson[u] = -1, siz[u] = 1;
5     fat[u] = fa, dep[u] = dep[fa] + 1;
6     for(int i = head[u]; ~i; i = e[i].Next) {
7         int v = e[i].ver;
8         if(v == fa) continue;
9         dfs1(v, u), siz[u] += siz[v];
10        if(siz[v] > kot)
11            kot = siz[v], hson[u] = v;
12    }
13 }
14 // 处理 dfn, rnk, 并进行重链剖分。
15 void dfs2(int u, int tp) {
16     top[u] = tp, dfn[u] = ++tim, rnk[tim] = u;
17     if(hson[u] == -1) return;
18     dfs2(hson[u], tp);
19     // 先 dfs 重儿子, 保证重链上 dfn 连续, 维持重链的性质
20     for(int i = head[u]; ~i; i = e[i].Next) {

```

```

21     int v = e[i].ver;
22     if(v == fat[u] || v == hson[u]) continue;
23     dfs2(v, v);
24 }
25 }
26 void add_subtree(int u, int value) {
27     tr.update(1, dfn[u], dfn[u] + siz[u] - 1, value);
28     // 子树代表的区间的左右端点分别是 dfn[u], dfn[u] + siz[u]
    ↪ - 1;
29 }
30 int query_subtree(int u) {
31     return tr.query(1, dfn[u], dfn[u] + siz[u] - 1) % mod;
32 }
33 // 类似倍增 LCA 的跳重链过程
34 void add_path(int u, int v, int value) {
35     while(top[u] != top[v]) {
36         if(dep[top[u]] < dep[top[v]])
37             swap(u, v);
38         // 让链顶深度大的来跳
39
40         tr.update(1, dfn[top[u]], dfn[u], value);
41         // 把 u 到链顶的权值全部修改。
42         u = fat[top[u]];
43         // 跳到链顶的父亲节点。
44     }
45
46     if(dep[u] > dep[v]) swap(u, v);
47     tr.update(1, dfn[u], dfn[v], value);
48     // 一条重链上的 dfn 是连续的。
49 }
50 int query_path(int u, int v) {
51     int ret = 0;
52     while(top[u] != top[v]) {
53         if(dep[top[u]] < dep[top[v]])
54             swap(u, v);
55         ret = (ret + tr.query(1, dfn[top[u]], dfn[u])) %
    ↪ mod;
56         u = fat[top[u]];
57     }
58     if(dep[u] > dep[v]) swap(u, v);
59     ret = (ret + tr.query(1, dfn[u], dfn[v])) % mod;
60     return ret % mod;
61 }
62 int lca(int u, int v) {
63     while(top[u] != top[v]) {
64         if(dep[top[u]] < dep[top[v]])
65             swap(u, v);
66         u = fat[top[u]];
67     }
68     if(dep[u] > dep[v]) swap(u, v);
69     return u;
70 }

```

2.7 主席树

```

1 // 静态区间第 k 大
2 const int si = 1e5 + 10;
3 int n, m, len;
4 int a[si], id[si];
5 int tot = 0;
6 int ls[si << 5], rs[si << 5];
7 int root[si << 5], dat[si << 5];
8 int build(int l, int r) {
9     int p = ++tot;
10    if(l == r) return p;
11    int mid = (l + r) >> 1;
12    ls[p] = build(l, mid), rs[p] = build(mid + 1, r);
13    return p;
14 }
15 int insert(int last, int l, int r, int val) { // last 是上
    ↪ 一个版本的 [l, r] 节点。
16     int p = ++tot;
17     dat[p] = dat[last] + 1;
18     if(l == r) return p;
19     int mid = (l + r) >> 1;
20     if(val <= mid)
21         ls[p] = insert(ls[last], l, mid, val), rs[p] =
    ↪ rs[last];
22     else
23         rs[p] = insert(rs[last], mid + 1, r, val), ls[p] =
    ↪ ls[last];
24     return p;
25 }

```

```

26 int ask(int p, int q, int l, int r, int kth) {
27     if(l == r) return l;
28     int mid = (l + r) >> 1;
29     int lcnt = dat[ls[q]] - dat[ls[p]];
30     if(kth <= lcnt)
31         return ask(ls[p], ls[q], l, mid, kth);
32     else
33         return ask(rs[p], rs[q], mid + 1, r, kth - lcnt);
34 }
35 int index(int val) {
36     return lower_bound(id + 1, id + 1 + len, val) - id;
37 }
38 int main() {
39     read(n), read(m);
40     for(int i = 1; i <= n; ++i)
41         read(a[i]), id[i] = a[i];
42     sort(id + 1, id + 1 + n);
43     len = unique(id + 1, id + 1 + n) - id - 1;
44     root[0] = build(1, len);
45     for(int i = 1; i <= n; ++i)
46         root[i] = insert(root[i - 1], 1, len, index(a[i]));
47     while(m--) {
48         int l, r, k; read(l), read(r), read(k);
49         write(id[ask(root[l - 1], root[r], 1, len, k)]);
50         write endl;
51     }
52
53     return 0;
54 }
55
56 // 单点修改
57 const int si = 2e5 + 10;
58
59 int n, m, len;
60 int a[si], id[si << 1];
61
62 int tot = 0;
63 int ls[si << 8], rs[si << 8];
64 int root[si << 8], dat[si << 8];
65
66 int cnt1, cnt2;
67 int tr1[si], tr2[si];
68
69 struct Query { char opt; int l, r, x; } q[si];
70 inline int lowbit(int x) { return x & -x; }
71 inline int getid(int val) { return lower_bound(id + 1, id +
    ↪ 1 + len, val) - id; }
72
73 int build(int l, int r) {
74     int p = ++tot;
75     if(l == r) return l;
76     int mid = (l + r) >> 1;
77     ls[p] = build(l, mid), rs[p] = build(mid + 1, r);
78     return p;
79 }
80 void insert(int &p, int last, int l, int r, int val, int
    ↪ delta) {
81     p = ++tot;
82     dat[p] = dat[last] + delta, ls[p] = ls[last], rs[p] =
    ↪ rs[last];
83     if(l == r) return;
84     int mid = (l + r) >> 1;
85     if(val <= mid) insert(ls[p], ls[last], l, mid, val,
    ↪ delta);
86     else insert(rs[p], rs[last], mid + 1, r, val, delta);
87 }
88 int ask(int l, int r, int kth) {
89     if(l == r) return l;
90     int mid = (l + r) >> 1;
91     int lcnt = 0;
92     for(int i = 1; i <= cnt2; ++i) lcnt += dat[ls[tr2[i]]];
93     for(int i = 1; i <= cnt1; ++i) lcnt -= dat[ls[tr1[i]]];
94     if(kth <= lcnt) {
95         for(int i = 1; i <= cnt1; ++i) tr1[i] = ls[tr1[i]];
96         for(int i = 1; i <= cnt2; ++i) tr2[i] = ls[tr2[i]];
97         return ask(l, mid, kth);
98     }
99     else {
100        for(int i = 1; i <= cnt1; ++i) tr1[i] = rs[tr1[i]];
101        for(int i = 1; i <= cnt2; ++i) tr2[i] = rs[tr2[i]];
102        return ask(mid + 1, r, kth - lcnt);
103    }

```

```

104 }
105 void change(int x, int v) {
106     int y = getid(a[x]);
107     while(x <= n) {
108         insert(root[x], root[x], 1, len, y, v);
109         x += lowbit(x);
110     }
111 }
112 int query(int l, int r, int kth) {
113     l --, cnt1 = cnt2 = 0;
114     while(l) tr1[++cnt1] = root[l], l -= lowbit(l);
115     while(r) tr2[++cnt2] = root[r], r -= lowbit(r);
116     return ask(1, len, kth);
117 }
118
119 int main() {
120     cin >> n >> m;
121     int cnt = 0;
122     for(int i = 1; i <= n; ++i)
123         cin >> a[i], id[++cnt] = a[i];
124     for(int i = 1; i <= m; ++i) {
125         Query &p = q[i];
126         cin >> p.opt;
127         if(p.opt == 'C')
128             cin >> p.l >> p.x, id[++cnt] = p.x;
129         if(p.opt == 'Q')
130             cin >> p.l >> p.r >> p.x;
131     }
132     sort(id + 1, id + 1 + cnt);
133     len = unique(id + 1, id + 1 + cnt) - id - 1;
134     for(int i = 1; i <= n; ++i) change(i, 1);
135     for(int i = 1; i <= m; ++i) {
136         Query &p = q[i];
137         if(p.opt == 'C') change(p.l, -1), a[p.l] = p.x,
138         ↪ change(p.l, 1);
139         if(p.opt == 'Q') cout << id[query(p.l, p.r, p.x)]
140         ↪ << endl;
141     }
142     return 0;
143 }

```

2.8 李超树

```

1 const ldb eps = 1e-9;
2 const int mod1 = 39989;
3 const int mod2 = 1e9;
4 const int si = 1e5 + 10;
5
6 int n, tot = 0;
7 struct Line { double k, b; } a[si];
8 ldb calc(int idx, int x) { return (a[idx].k * x +
9     ↪ a[idx].b); }
10 void add(int x, int y, int xx, int yy) {
11     ++tot;
12     if(x == xx) a[tot].k = 0, a[tot].b = max(y, yy);
13     else a[tot].k = (ldb)((1.0 * (yy - y)) / (1.0 * (xx -
14     ↪ x))), a[tot].b = y - a[tot].k * x;
15 }
16 int cmp(ldb x, ldb y) {
17     if((x - y) > eps) return 1; // Greater.
18     else if((y - x) > eps) return -1; // Less
19     return 0;
20 }
21 pdi Max(pdi x, pdi y) {
22     if(cmp(x.first, y.first) == 1) return x;
23     else if(cmp(y.first, x.first) == 1) return y;
24     return (x.second < y.second) ? x : y;
25 }
26
27 struct LichaoTree {
28     int id[si << 2];
29     void modify(int p, int l, int r, int u) {
30         int &v = id[p], mid = (l + r) >> 1;
31         if(cmp(calc(u, mid), calc(v, mid)) == 1)
32             swap(u, v);
33         Lich boundl = cmp(calc(u, l), calc(v, l));
34         int boundr = cmp(calc(u, r), calc(v, r));
35         if(boundl == 1 || (!boundl && u < v))
36             Lich modify(p << 1, l, mid, u);
37         if(boundr == 1 || (!boundr && u < v))
38             modify(p << 1 | 1, mid + 1, r, u);
39     } //

```

```

38 void update(int p, int nl, int nr, int l, int r, int u)
39     ↪ {
40     if(l <= nl && nr <= r)
41         return modify(p, nl, nr, u);
42     int mid = (nl + nr) >> 1;
43     if(l <= mid)
44         update(p << 1, nl, mid, l, r, u);
45     if(r > mid)
46         update(p << 1 | 1, mid + 1, nr, l, r, u);
47 }
48 pdi query(int p, int l, int r, int x) {
49     if(x < l || r < x)
50         return {0.0, 0};
51     ldb ret = calc(id[p], x), mid = (l + r) >> 1;
52     if(l == r)
53         return {ret, id[p]};
54     return Max({ret, id[p]}, Max(query(p << 1, l, mid,
55     ↪ x), query(p << 1 | 1, mid + 1, r, x)));
56 } tr;

```

2.9 珂朵莉树

```

1 struct node {
2     int l, r;
3     mutable int val;
4     node(const int &il, const int &ir, const int &iv) :
5     ↪ l(il), r(ir), val(iv) {}
6     bool operator < (const node &b) const { return l < b.l;
7     ↪ }
8 }; std::set<node> odt;
9
10 std::set<node>::iterator split(int pos) {
11     if(pos > n) return odt.end();
12     std::set<node>::iterator it =
13     | --odt.upper_bound((node){pos, 0, 0});
14     if(it -> l == pos) return it;
15     int l = it -> l, r = it -> r, v = it -> val;
16     odt.erase(it), odt.insert((node){l, pos - 1, v});
17     return odt.insert((node){pos, r, v}).first;
18 } // split the node [l,r] to two smaller node [l,pos),
19     ↪ [pos,r];
20 void assign(int l, int r, int v) {
21     std::set<node>::iterator itr = split(r + 1), itl =
22     ↪ split(l);
23     odt.erase(itl, itr), odt.insert((node){l, r, v});
24 } // change all element in the interval [l,r] to v;
25 void example(int l, int r, int v) {
26     std::set<node>::iterator itr = split(r + 1), itl =
27     ↪ split(l);
28     for(; itl != itr; ++itl) {
29         // blablabla...
30     }
31     return;
32 }

```

3. Dp

3.1 数位 DP

```

1 def dfs(当前位数 x, 当前状态 y, 前导零限制 st, 上界限制
2     ↪ limit):
3     if 到达边界 and 符合要求 then
4         返回边界的合法答案
5     if 到达边界 and 不符合要求 then
6         返回边界的不合法答案 # 这个一般不会有, 一般枚举填
7         ↪ 数的时候如果没有限制就会有 (只要会访问到边界不合法情况
8         ↪ 就要加上)。
9
10 if 当前的状态已经记忆化过 then
11     返回记录的答案
12
13 var result = 0 # 记录答案
14 var up = 9 # 当前位填数的上限
15 if 有上界限制 then
16     up = 当前位在 n 当中的数字 # n 是要求的 F(n) 的
17     ↪ 自变量
18
19 for 枚举当前位的填数值 from 0 to up :
20     if 当前位填的数不符合限制 then
21         continue

```

```

19     if 有前导零限制 and 当前填写的是 0 then
20         result += dfs(x - 1, 下一个状态, True, 是否触碰
           ↳ 上界限制)
21     else then
22         result += dfs(x - 1, 下一个状态, False, 是否触碰
           ↳ 上界限制)
23
24     记录当前状态的答案 f[x][y] = result
25     返回答案 result
26
27
28 def solve(要求的 F 的自变量 n):
29     存储每一位数字的 vector 清空
30     while n != 0 then:
31         vector <-- n % base          # base表示是哪一个进制
32         n /= base
33     清空状态数组
34     返回对应状态的答案 (调用 dfs)

```

3.2 子集卷积/SOS DP

```

1 for(int msk = 0; msk < (1 << n); ++msk)
2     dp[msk] = a[msk];
3 for(int i = 0; i < n; ++i) {
4     for(int msk = 0; msk < (1 << n); ++msk) {
5         if(msk & (1 << i))
6             dp[msk] += dp[msk ^ (1 << i)];
7     }
8 }

```

4. Math

4.1 线性求逆元以及组合数

```

1 int inv[si], fact[si], invf[si];
2 void init(int n) {
3     inv[1] = 1, fact[0] = invf[0] = 1;
4     for(int i = 2; i <= n; ++i)
5         inv[i] = 1ll * (mod - mod / i) * inv[mod % i] %
           ↳ mod;
6     for(int i = 1; i <= n; ++i)
7         fact[i] = 1ll * fact[i - 1] * i % mod,
8         invf[i] = 1ll * invf[i - 1] * inv[i] % mod;
9 }
10 int C(int n, int m) {
11     if(m < 0 || n < m) return 0;
12     return 1ll * fact[n] * invf[n - m] % mod * invf[m] %
       ↳ mod;
13 }
14 int Catalan(int n) {
15     return 1ll * C(n * 2, n) % mod * inv[n + 1] % mod;
16 }

```

4.2 快速幂

```

1 int Qpow(int a, int b) {
2     int ret = 1 % mod;
3     for(; b; b >>= 1) {
4         if(b & 1) ret = ret * a % mod;
5         a = a * a % mod;
6     }
7     return ret % mod;
8 }

```

4.3 矩阵乘法

```

1 struct Matrix {
2     int a[si][si];
3     Matrix() { memset(a, 0, sizeof a); }
4     Matrix operator * (const Matrix &B) const {
5         Matrix C, A = *this;
6         for(int i = 1; i <= cnt; ++i)
7             for(int j = 1; j <= cnt; ++j)
8                 for(int k = 1; k <= cnt; ++k)
9                     C.a[i][j] += A.a[i][k] * B.a[k][j];
10        return C;
11    }
12 };
13 // 循环的时候最好不要用 si。
14 // 用一个设定好的常数或者题目给的变量会比较。
15 // 但是如果乘法不止需要适用于一对 n,m,k, 那么就最好用 si - 1。
16 // 为啥不会有影响呢? 因为构造函数里把没有用到的设置成 0 了。

```

4.4 高斯消元

```

1 using i64 = long long;
2 using ldb = long double;
3
4 const int si = 50 + 10;
5 const ldb eps = 1e-5;
6
7 int n;
8 ldb c[si][si], d[si], x[si];
9
10 int Gauss() {
11     for(int i = 1; i <= n; ++i) {
12         int l = i;
13         for(int j = i + 1; j <= n; ++j)
14             if(fabs(c[j][i]) > fabs(c[l][i]))
15                 l = j; // 找到最大的
16         if(l != i) {
17             for(int j = 1; j <= n; ++j)
18                 swap(c[i][j], c[l][j]);
19             swap(d[i], d[l]);
20         } // 交换
21         if(fabs(c[i][i]) >= eps) {
22             for(int j = 1; j <= n; ++j) {
23                 if(j == i) continue;
24                 ldb rte = c[j][i] / c[i][i];
25                 for(int k = 1; k <= n; ++k)
26                     c[j][k] -= rte * c[i][k];
27                 d[j] -= rte * d[i];
28             }
29         } // 消元
30     }
31     bool nosol = false, infsol = false;
32     for(int i = 1; i <= n; ++i) {
33         int j = 1;
34         while(fabs(c[i][j]) < eps && j <= n)
35             j++;
36         j += (fabs(d[i]) < eps);
37         if(j > n + 1) infsol = true;
38         if(j == n + 1) nosol = true;
39     } // 检查自由元
40     if(nosol) return 0;
41     if(infsol) return 1;
42     for(int i = n; i >= 1; --i) {
43         for(int j = i + 1; j <= n; ++j)
44             d[i] -= x[j] * c[i][j];
45         x[i] = d[i] / c[i][i];
46     } // 回代
47     for(int i = 1; i <= n; ++i)
48         cout << "x" << i << "=" << fixed << setprecision(2)
49         ↳ << x[i] << endl;
50     return 2;
51 }

```

4.5 质数

```

1 bool is_prime(int n) {
2     if(n < 2) return false;
3     for(int i = 2; i * i <= n; ++i)
4         if(n % i == 0) return false;
5     return true;
6 }
7
8 int vis[si];
9 int m, prime[si];
10 // O(n log log n)
11 void get_primes(int n) {
12     m = 0;
13     memset(vis, 0, sizeof vis);
14     for(int i = 2; i <= n; ++i) {
15         if(!vis[i]) prime[++m] = i;
16         for(int j = i * i; j <= n; ++j)
17             vis[j] = 1;
18     }
19 }
20
21 // Euler O(n)
22 void get_primes(int n) {
23     m = 0;
24     memset(vis, 0, sizeof vis);
25     for(int i = 2; i <= n; ++i) {
26         if(vis[i] == 0) {

```



```

27     vis[i] = i;
28     prime[++m] = i;
29 }
30 for(int j = 1; j <= m; ++j) {
31     if(prime[j] > vis[i] || prime[j] * i > n)
32         break;
33     vis[prime[j] * i] = prime[j];
34 }
35 }
36 }
37
38 int c[si]; // exponential
39 int m = 0, p[si]; // prime factor
40 void divide(int n) {
41     m = 0;
42     for(int i = 2; i * i <= n; ++i) {
43         if(n % i == 0) {
44             p[++m] = i, c[m] = 0;
45             while(n % i == 0) n /= i, c[m]++;
46         }
47     }
48     if(n > 1) p[++m] = n, c[m] = 1;
49 }

```

4.6 约数

```

1 int m, div[si];
2 void get_factors(int n) {
3     m = 0;
4     for(int i = 1; i * i <= n; ++i)
5         if(n % i == 0) {
6             div[++m] = i;
7             if(i * i != n) div[++m] = n / i;
8         }
9 }
10
11 std::vector<int> fact[si];
12 void get_factors(int n) {
13     for(int i = 1; i <= n; ++i)
14         for(int j = 1; j <= n / i; ++j)
15             fact[i * j].emplace_back(i);
16 }
17
18 int gcd(int a, int b) {
19     return b ? gcd(b, a % b) : a;
20 }
21
22 int phi[si];
23 int m = 0, prime[si], vis[si];
24 void calc_euler_func(int n) {
25     m = 0, phi[1] = 1;
26     memset(vis, 0, sizeof vis);
27     for(int i = 2; i <= n; ++i) {
28         if(vis[i] == 0)
29             vis[i] = i, prime[++m] = i, phi[i] = i - 1;
30         for(int j = 1; j <= m; ++j) {
31             if(prime[j] > vis[i] || prime[j] * vis[i] > n)
32                 break;
33             vis[prime[j] * i] = prime[j];
34             if(i % prime[j] == 0)
35                 phi[prime[j] * i] = phi[i] * prime[j];
36             else
37                 phi[prime[j] * i] = phi[i] * (prime[j] - 1);
38         }
39     }

```

4.7 Exgcd

```

1 int exgcd(int a, int b, int &x, int &y) {
2     if(!b) { x = 1, y = 0; return a; }
3     int d = exgcd(b, a % b, x, y);
4     int z = x; x = y; y = z - y * (a / b);
5 }

```

4.8 中国剩余定理

```

1 #define int long long
2 int crt(std::vector<int> &r, std::vector<int> &m) {
3     int n = 1, ans = 0;
4     for(int i = 0; i < (int)m.size(); ++i)
5         n = n * m[i];
6     for(int i = 0; i < (int)m.size(); ++i) {

```

```

7         int mi = n / m[i], b, y;
8         exgcd(mi, m[i], b, y);
9         ans = (ans + r[i] * mi * b % n) % n;
10     }
11     return (ans % n + n) % n;
12 }

```

5. Graph

5.1 SPFA 以及负环

```

1 std::queue<int> q;
2 void spfa(int s) {
3     memset(dis, 0x3f, sizeof dis);
4     memset(vis, false, sizeof vis);
5     dis[s] = 0, q.push(s), vis[s] = true;
6     while(!q.empty()) {
7         int u = q.front();
8         q.pop(), vis[u] = false;
9         for(int i = head[u]; ~i; i = e[i].Next) {
10             int v = e[i].ver, w = e[i].w;
11             if(dis[v] > dis[u] + w) {
12                 dis[v] = dis[u] + w;
13                 if(!vis[v]) q.push(v), vis[v] = true;
14             }
15         }
16     }
17 }
18
19 // Minus Ring Check
20 bool vis[si];
21 std::queue<int> Q;
22 int dis[si], cnt[si];
23 bool spfa(int s) {
24     memset(dis, 0, sizeof dis);
25     memset(cnt, 0, sizeof cnt);
26     memset(vis, false, sizeof vis);
27     for(int i = 1; i <= n; ++i)
28         Q.push(i), vis[i] = true;
29     cnt[s] = 0; // 全部入队, 相当于建立一个超级源点。
30     while(!Q.empty()) {
31         int u = Q.front();
32         Q.pop(), vis[u] = false;
33         for(int i = head[u]; ~i; i = e[i].Next) {
34             int v = e[i].ver, w = e[i].w;
35             if(dis[v] > dis[u] + w) {
36                 dis[v] = dis[u] + w, cnt[v] = cnt[u] + 1;
37                 if(cnt[v] >= n) return true;
38                 if(!vis[v]) Q.push(v), vis[v] = true;
39             }
40         }
41     }
42     return false;
43 }
44
45 // SLF + Swap Optimize
46
47 struct Slfswap {
48     std::deque<int> dq;
49     Slfswap() { dq.clear(); }
50     void push(int x) {
51         if(!dq.empty()) {
52             if(dis[x] < dis[dq.front()])
53                 dq.push_front(x);
54             else dq.push_back(x);
55             if(dis[dq.front()] > dis[dq.back()])
56                 swap(dq.front(), dq.back());
57             // 这里的两重 if 可以保证只会在至少有两个元素的时候才交换。
58         } else dq.push_back(x);
59     }
60     void pop() {
61         dq.pop_front();
62         if(!dq.empty() && dis[dq.front()] > dis[dq.back()])
63             swap(dq.front(), dq.back());
64     }
65     int size() { return dq.size(); }
66     int front() { return dq.front(); }
67     bool empty() { return !dq.size(); }
68 } q;

```


5.2 Dijkstra

```

1 std::priority_queue<std::pair<int, int> > q;
2 void dijkstra(int s) {
3     memset(dis, 0x3f, sizeof dis);
4     memset(vis, false, sizeof vis);
5     dis[s] = 0, q.push({dis[s], s});
6     while(!q.empty()) {
7         int u = q.top().second;
8         | q.pop();
9         if(vis[u]) continue;
10        | vis[u] = true;
11        for(int i = head[u]; ~i; i = e[i].Next) {
12            int v = e[i].ver, w = e[i].w;
13            if(dis[v] > dis[u] + w)
14                | dis[v] = dis[u] + w, q.push({-dis[v], v}); //利
                ↳ 用相反数把大根堆-> 小根堆
                // 一定要先更新 dis[v] 再 q.push
15        }
16    }
17 }
18 }

```

5.3 Floyd 以及最小环

```

1 for(int k = 1; k <= n; ++k)
2     for(int i = 1; i <= n; ++i)
3         for(int j = 1; j <= n; ++j)
4             dis[i][j] = min(dis[i][j], dis[i][k] + dis[k]
5             ↳ [j]);
6 // 不要忘记初始化.
7 // 最小环:
8 std::vector<int> ans_path;
9 void gopath(int u, int v) {
10    if(pos[u][v] == 0)
11        return;
12    gopath(u, pos[u][v]), ans_path.push_back(pos[u][v]),
13    ↳ gopath(pos[u][v], v);
14 }
15 signed main() {
16    cin >> n >> m;
17    memset(a, 0x3f, sizeof a);
18    for(int i = 1; i <= n; ++i)
19        a[i][i] = 0;
20    for(int i = 1; i <= m; ++i) {
21        int u, v, w;
22        cin >> u >> v >> w;
23        a[u][v] = min(a[u][v], w), a[v][u] = a[u][v];
24    }
25    memcpy(dis, a, sizeof a);
26    int ans = 0x3f3f3f3f3f3f3f3f, tmp = ans;
27    for(int k = 1; k <= n; ++k)
28        for(int i = 1; i < k; ++i) // 注意是dp之前, 此时 dis
29        ↳ 还是 k-1 的时候的状态.
30            for(int j = i + 1; j < k; ++j)
31                if(a[j][k] < tmp / 2 && a[k][i] < tmp / 2
32                ↳ && ans > dis[i][j] + a[j][k] + a[k][i])
33                    ans = dis[i][j] + a[j][k] + a[k][i],
34                    ans_path.clear(),
35                    ↳ ans_path.push_back(i), gopath(i, j),
36                    ans_path.push_back(j),
37                    ↳ ans_path.push_back(k);
38                    // 不判的话 a[j][k]+a[k][i] 有可能爆, 导致答
39                    ↳ 案出错.
40                    // 更新最小环取min的过程
41                    for(int i = 1; i <= n; ++i)
42                        for(int j = 1; j <= n; ++j)
43                            if(dis[i][j] > dis[i][k] + dis[k][j])
44                                pos[i][j] = k, dis[i][j] = dis[i][k] +
45                                ↳ dis[k][j];
46                    // 正常的 Floyd
47    }
48    if(ans == 0x3f3f3f3f3f3f3f3f)
49        return puts("No solution."), 0;
50    for(auto x : ans_path)
51        cout << x << " ";
52    return puts(""), 0;
53 }

```

5.4 Kruskal

```

1 struct Edge {
2     int x, y, z;

```

```

3     bool operator < (const Edge &b) const {
4         return z < b.z;
5     }
6 } a[si_m];
7
8 for(int i = 1; i <= m; ++i)
9     | cin >> a[i].x >> a[i].y >> a[i].z;
10 sort(a + 1, a + 1 + m);
11 int ans = 0;
12 for(int i = 1; i <= m; ++i) {
13     | if(dsu.same(a[i].x, a[i].y))
14     | | continue;
15     | dsu.Union(a[i].x, a[i].y), ans += a[i].z;
16 }

```

Prim

```

1 void Prim() {
2     | memset(dis, 0x3f, sizeof dis);
3     memset(vis, false, sizeof vis), dis[1] = 0;
4     for(int i = 1; i < n; ++i) {
5         int x = 0;
6         for(int j = 1; j <= n; ++j)
7             if(!vis[j] && (x == 0 || dis[j] < dis[x]))
8                 x = j;
9         vis[x] = true;
10        for(int y = 1; y <= n; ++y)
11            if(!vis[y]) dis[y] = min(dis[y], a[x][y]);
12    }
13 }
14
15 memset(a, 0x3f, sizeof a);
16 for(int i = 1; i < n; ++i) {
17     | a[i][i] = 0;
18     | for(int j = 1; j <= n; ++j) {
19         | | int value;
20         | | cin >> value;
21         | | a[i][j] = a[j][i] = min(a[i][j], value);
22     }
23 }
24 Prim();
25 int ans = 0;
26 for(int i = 2; i <= n; ++i)
27     | ans += dis[i];

```

5.5 倍增 LCA

```

1 int dep[si_n], f[si_n][20];
2 void dfs(int u, int fa) {
3     dep[u] = dep[fa] + 1, f[u][0] = fa;
4     for(int i = 1; i <= 19; ++i)
5         f[u][i] = f[f[u][i-1]][i-1];
6     for(int i = head[u]; ~i; i = e[i].Next) {
7         int v = e[i].ver;
8         if(v == fa) continue;
9         dfs(v, u);
10    }
11 }
12 int lca(int x, int y) {
13     if(dep[x] < dep[y]) swap(x, y);
14     for(int i = 19; i >= 0; --i)
15         if(dep[f[x][i]] >= dep[y]) x = f[x][i];
16     if(x == y) return x;
17     for(int i = 19; i >= 0; --i)
18         if(f[x][i] != f[y][i]) x = f[x][i], y = f[y][i];
19     return f[x][0];
20 }

```

5.6 Tarjan LCA

```

1 int pa[si];
2 int root(int x) {
3     if(pa[x] != x)
4         return pa[x] = root(pa[x]);
5     return pa[x];
6 }
7
8 int n, q, s;
9 int lca[si];
10 bool vis[si];
11 std::vector<int> que[si], pos[si];

```

```

12 void tarjan(int u) {
13     vis[u] = true;
14     for(int i = head[u]; ~i; i = e[i].Next) {
15         int v = e[i].ver;
16         if(vis[v] == true) continue;
17         tarjan(v), pa[v] = root(u);
18     }
19     for(int i = 0; i < (int)que[u].size(); ++i) {
20         int v = que[u][i], po = pos[u][i];
21         if(vis[v] == true) lca[po] = root(v);
22     }
23 }
24
25 int main(){
26     cin >> n >> q >> s;
27     for(int i = 1; i <= n; ++i)
28         pa[i] = i, vis[i] = false,
29         que[i].clear(), pos[i].clear();
30     for(int i = 1; i < n; ++i) {
31         int u, v;
32         cin >> u >> v;
33         add(u, v), add(v, u);
34     }
35     for(int i = 1; i <= q; ++i) {
36         int u, v;
37         cin >> u >> v;
38         if(u == v) lca[i] = u;
39         else {
40             que[u].pb(v), que[v].pb(u);
41             pos[u].pb(i), pos[v].pb(i);
42         }
43     }
44     tarjan(s);
45     for(int i = 1; i <= q; ++i)
46         cout << lca[i] << endl;
47     return 0;
48 }

```

5.7 拓扑排序

```

1 int cnt = 0;
2 std::queue<int> q;
3 for(int i = 1; i <= n; ++i)
4     if(!ind[i]) q.push(i);
5 while(!q.empty()) {
6     int u = q.front(); q.pop();
7     ord[u] = ++cnt; // topo 序
8     for(auto v : G[u]) if(!(--ind[v])) q.push(v);
9     // 删掉边, 顺便判一下要不要入队。
10 }

```

5.8 欧拉回路

```

1 std::stack<int> s;
2 void dfs(int u) {
3     for(int i = head[u]; ~i; i = e[i].Next) {
4         int v = e[i].ver;
5         if(!vis[i]) { // 当前边没有访问过
6             vis[i] = true; // 注意一定要访问到就直接标记, 不
7             // 然复杂度会假。
8             dfs(v), s.push(v);
9         }
10    }
11 }
12 dfs(1); // 因为有欧拉回路, 所以其实从哪个点开始都一样。
13 std::vector<int> ans;
14 while(!s.empty())
15     ans.push_back(s.top()), s.pop();
16 reverse(ans.begin(), ans.end());
17 for(auto x : ans) cout << x << " "; // 倒序输出。

```

5.9 强连通分量

```

1 bool ins[si];
2 std::stack<int> s;
3 std::vector<int> scc[si];
4 int n, m, cnt_t = 0, tot = 0;
5 int dfn[si], low[si], c[si];
6
7 void tarjan(int u) {
8     dfn[u] = low[u] = ++cnt_t;

```

```

10     s.push(u), ins[u] = true;
11     for(int i = head[u]; ~i; i = e[i].Next) {
12         int v = e[i].ver;
13         if(!dfn[v])
14             tarjan(v), low[u] = min(low[u], low[v]);
15         // 没有访问过, 递归搜索然后更新 low。
16         else if(ins[v]) low[u] = min(low[u], dfn[v]);
17         // 已经在栈中了, 用 dfn[v] 来更新 low[u]。
18     }
19     if(dfn[u] == low[u]) {
20         ++tot; int x;
21         do {
22             x = s.top(), s.pop(), ins[x] = false;
23             c[x] = tot, scc[tot].pb(x);
24         } while(u != x);
25     } // 出现了一个 SCC。
26 }
27
28 Edge edag[si << 1];
29 void contract() {
30     for(int u = 1; u <= n; ++u) {
31         for(int i = head[u]; ~i; i = e[i].Next) {
32             int v = e[i].ver;
33             if(c[u] == c[v]) continue;
34             add_n(c[u], c[v]);
35         }
36     } // 缩点。
37 }
38
39 for(int i = 1; i <= n; ++i)
40     if(!dfn[i]) tarjan(i);

```

5.10 边双连通分量

```

1 int n, m, q;
2 // 原图
3 int head[si], tot1 = 0;
4 struct Edge { int ver, Next; } e[si << 2];
5 inline void add1(int u, int v) { e[tot1] = (Edge){v,
6     head[u]}, head[u] = tot1++; }
7
8 // 缩完点之后的图
9 // 如果原来的图是连通图的话
10 // 可以证明缩完点之后必然是一棵树。
11 int Head[si], tot2 = 0;
12 struct Tree { int ver, Next; } t[si << 2];
13 inline void add2(int u, int v) { t[tot2] = (Tree){v,
14     Head[u]}, Head[u] = tot2++; }
15
16 // E-dcc 的个数。
17 int cnt = 0;
18 int dfn[si], low[si], tim = 0, c[si];
19 bool bridge[si << 2]; // 是否是桥
20
21 // in_edge 是用来消除重边的影响的。
22 // 表示当前状态是从哪一条边过来的。
23 void tarjan(int u, int in_edge) {
24     dfn[u] = low[u] = ++tim;
25     for(int i = head[u]; ~i; i = e[i].Next) {
26         int v = e[i].ver;
27         if(!dfn[v]) {
28             tarjan(v, i);
29             low[u] = min(low[u], low[v]);
30             if(dfn[u] < low[v]) bridge[i] = bridge[i ^ 1] =
31                 true;
32         }
33         else if((i ^ 1) != in_edge) low[u] = min(low[u],
34             dfn[v]);
35     }
36 }
37
38 // 去掉桥边的连通块染色
39 void dfs(int u, int col) {
40     c[u] = col;
41     for(int i = head[u]; ~i; i = e[i].Next) {
42         int v = e[i].ver;
43         if(c[v] || bridge[i]) continue;
44         dfs(v, col);
45     }
46 }
47 void Construct() {
48     for(int i = 1; i <= n; ++i){

```

```

45     for(int j = head[i]; ~j; j = e[j].Next) {
46         int v = e[j].ver;
47         if(c[i] == c[v]) continue;
48         // 只需要加一次, 遍历到反向边的时候会自动补全成无
    ↪ 向边
49         add2(c[i], c[v]);
50     }
51 }
52 }
53
54 int main() {
55     memset(head, -1, sizeof head);
56     memset(Head, -1, sizeof Head);
57     memset(bridge, false, sizeof bridge);
58     cin >> n >> m;
59     for(int i = 1; i <= m; ++i) {
60         int u, v;
61         cin >> u >> v;
62         add1(u, v), add1(v, u);
63     }
64     for(int i = 1; i <= n; ++i) if(!dfn[i]) tarjan(i, -1);
65     for(int i = 1; i <= n; ++i) if(!c[i]) ++cnt, dfs(i,
    ↪ cnt);
66     Construct();
67 }

```

5.11 点双连通分量

```

1  int n, m, root;
2
3  int head[si], tot1 = 0;
4  int Head[si], tot2 = 0;
5  struct Edge { int ver, Next; } e[si << 2], g[si << 2];
6  void add1(int u, int v) { e[tot1] = (Edge){v, head[u]},
    ↪ head[u] = tot1++; }
7  void add2(int u, int v) { g[tot2] = (Edge){v, Head[u]},
    ↪ Head[u] = tot2++; }
8
9  // Vdcc 的个数
10 int cnt = 0;
11 int dfn[si], low[si], c[si], tim;
12 int new_id[si]; // 割点的新编号
13 bool cut[si]; // 是否是割点
14 std::stack<int> s;
15 std::vector<int> vdcc[si];
16
17 void tarjan(int u) {
18     dfn[u] = low[u] = ++tim;
19     s.push(u);
20     // 孤立点
21     if(u == root && head[u] == -1) {
22         vdcc[++cnt].emplace_back(u);
23         return;
24     }
25     int flag = 0;
26     for(int i = head[u]; ~i; i = e[i].Next) {
27         int v = e[i].ver;
28         if(!dfn[v]) {
29             tarjan(v), low[u] = min(low[u], low[v]);
30             if(dfn[u] <= low[v]) {
31                 ++flag;
32                 // 根节点特判
33                 if(u != root || flag > 1) { // 注意这里是短
    ↪ 路运算符, 不要打反了。
34                     cut[u] = true;
35                 }
36                 int x; ++cnt;
37                 do {
38                     x = s.top(), s.pop();
39                     vdcc[cnt].emplace_back(x);
40                 } while(v != x);
41                 // 注意这里要是 v 不是 u
42                 // 如果 u 被弹出了, 之后的连通块就会少 u。
43                 vdcc[cnt].emplace_back(u);
44             }
45         }
46         else low[u] = min(low[u], dfn[v]);
47     }
48 }
49
50 int num;
51 void Construct() {
52     num = cnt;

```

```

53     for(int u = 1; u <= n; ++u)
54         if(cut[u]) new_id[u] = ++num;
55     for(int i = 1; i <= cnt; ++i) {
56         for(int j : vdcc[i]) {
57             if(cut[j]) add2(i, new_id[j]), add2(new_id[j],
    ↪ i);
58             else c[j] = i;
59         }
60         // 如果是割点, 就和这个割点所在的 v-Dcc 连边
61         // 反之染色。
62     }
63     // 编号 1~cnt 的是 v-Dcc, 编号 > cnt 的是原图割点
64 }
65
66 int main() {
67     memset(head, -1, sizeof head);
68     memset(Head, -1, sizeof Head);
69
70     cin >> n >> m;
71     for(int i = 1; i <= m; ++i) {
72         int u, v;
73         cin >> u >> v;
74         // 判重边
75         if(u == v) continue;
76         add1(u, v), add1(v, u);
77     }
78     for(int i = 1; i <= n; ++i)
79         if(!dfn[i]) root = i, tarjan(i);
80     Construct();
81     return 0;
82 }

```

5.12 虚树

```

1  int k, a[si];
2  int stk[si], top = 0;
3  bool cmp(int x, int y) { return dfn[x] < dfn[y]; }
4  inline void ADD(int u, int v, int w) { E[Tot] = (Edge){v,
    ↪ Head[u], w}, Head[u] = Tot++; }
5  inline void Add(int u, int v) { int w = dist(u, v); ADD(u,
    ↪ v, w), ADD(v, u, w); }
6  void build() {
7     sort(a + 1, a + 1 + k, cmp);
8     stk[top = 1] = 1, Tot = 0, Head[1] = -1; // 这样清空复杂
    ↪ 度才是对的。
9     for(int i = 1, Lca; i <= k; ++i) {
10         if(a[i] == 1) continue;
11         Lca = lca(a[i], stk[top]);
12         if(Lca != stk[top]) {
13             while(dfn[Lca] < dfn[stk[top - 1]])
14                 Add(stk[top - 1], stk[top]), --top;
15             if(dfn[Lca] > dfn[stk[top - 1]])
16                 Head[Lca] = -1, Add(Lca, stk[top]),
    ↪ stk[top] = Lca;
17             else Add(Lca, stk[top--]); // Lca = stk[top -
    ↪ 1].
18         }
19         Head[a[i]] = -1, stk[++top] = a[i];
20     }
21     for(int i = 1; i < top; ++i)
22         Add(stk[i], stk[i + 1]);
23     return;
24 }

```

6. String

6.1 Kmp

```

1  Next[1] = 0;
2  for(int i = 2, j = 0; i <= n; ++i) {
3      while(j > 0 && s[i] != s[j + 1]) j = Next[j];
4      if(s[i] == s[j + 1]) j++;
5      Next[i] = j;
6  }
7  for(int i = 1, j = 0; i <= m; ++i) {
8      while(j > 0 && (j == n || s[i] != s[j + 1])) j =
    ↪ Next[j];
9      if(t[i] == s[j + 1]) ++j;
10     f[i] = j;
11     if(f[i] == n) orc[++cnt] = i - n + 1;
12 }

```

6.2 Trie

```

1 // 定义 NULL 为 0, 字符集为 a~z.
2 int tr[si][27];
3 bool exist[si];
4 int tot, root;
5
6 void init() {
7     memset(tr, 0, sizeof tr);
8     memset(exist, false, sizeof exist);
9     tot = 0, root = ++tot;
10 }
11 void insert(string s) {
12     int p = root;
13     for(int i = 0; i < (int)s.size(); ++i) {
14         int ch = (int)(s[i] - 'a') + 1;
15         if(!tr[p][ch])
16             tr[p][ch] = ++tot;
17         p = tr[p][ch];
18     }
19     exist[p] = true;
20 }
21 bool query(string s) {
22     int p = root;
23     for(int i = 0; i < (int)s.size(); ++i) {
24         int ch = (int)(s[i] - 'a') + 1;
25         if(!tr[p][ch])
26             return false;
27         p = tr[p][ch];
28     }
29     return exist[p];
30 }

```

6.3 01Trie

```

1 using i64 = long long;
2
3 const int si = 1e5 + 10;
4 const int k = 32;
5 int tr[k * si][2];
6 i64 value[k * si];
7 int tot = 0, root = ++tot;
8
9 int newnode() {
10     tr[++tot][0] = tr[tot][1] = value[tot] = 0;
11     return tot;
12 }
13 int cacid(int num, int pos) {
14     return (num >> pos) & 1;
15 }
16 void insert(int num) {
17     int p = root;
18     for(int i = 32; i >= 0; --i) {
19         int ch = cacid(num, i);
20         if(!tr[p][ch])
21             tr[p][ch] = newnode();
22         p = tr[p][ch];
23     }
24     value[p] = num;
25 }
26 // 查询异或 x 最大的一个。
27 i64 query(i64 num) {
28     int p = root;
29     for(int i = 32; i >= 0; --i) {
30         int ch = cacid(num, i);
31         if(tr[p][ch ^ 1])
32             p = tr[p][ch ^ 1];
33         else
34             p = tr[p][ch];
35     }
36     return value[p];
37 }
38
39 // 维护异或和, 全局加一。
40 const int si = 1e4 + 10;
41 const int MaxDepth = 21;
42
43 int tr[si * (MaxDepth + 1)][2];
44 int wei[si * (MaxDepth + 1)], xorv[si * (MaxDepth + 1)];
45 int tot = 0, root = ++tot;
46 // 其实这里 root 可以不用赋值, 递归开点的时候会自动给编号的。
47
48 int newnode() {
49     tr[++tot][0] = tr[tot][1] = wei[tot] = xorv[tot] = 0;

```

```

50     return tot;
51 }
52 void maintain(int p) {
53     wei[p] = xorv[p] = 0;
54     // 为了应对不断的删除和插入, 每次维护 p 的时候都令 wei,
55     // 也就是每次都**重新收集一次信息**, 而不是从原来的基础上
56     // 修改。
57     if(tr[p][0]) {
58         wei[p] += wei[tr[p][0]];
59         xorv[p] ^= (xorv[tr[p][0]] << 1);
60         // 因为儿子所维护的异或和实际上比 p 少一位,
61         // 如果要按位异或就要让儿子的异或和左移一位, 和 p 对齐。
62     }
63     if(tr[p][1]) {
64         wei[p] += wei[tr[p][1]];
65         xorv[p] ^= (xorv[tr[p][1]] << 1) | (wei[tr[p][1]] &
66         // 利用奇偶性计算。
67     }
68     wei[p] = wei[p] & 1;
69     // 每插入一次或者删除一次, 奇偶性都会变化。
70 }
71 // 类似线段树的 pushup, 从底向上收集信息。
72 // 换种说法, 是更新节点 p 的信息。
73 void insert(int &p, int x, int depth) {
74     if(!p)
75         p = newnode();
76     if(depth > MaxDepth) {
77         wei[p] += 1;
78         return;
79     }
80     insert(tr[p][x & 1], x >> 1, depth + 1);
81     // 从低到高位插入, 所以是 x >> 1。
82     maintain(p);
83 }
84 // 插入元素 x。
85 void remove(int p, int x, int depth) {
86     // 不知道是不是应该写 > MaxDepth - 1 还是 > MaxDepth ?
87     if(depth == MaxDepth) {
88         wei[p] -= 1;
89         return;
90     }
91     remove(tr[p][x & 1], x >> 1, depth + 1);
92     maintain(p);
93 }
94 // 删除元素 x, 但是 x 不能是不存在的元素。
95 // 否则会访问空节点 0 然后继续往下, 会出错。
96 void addall(int p) {
97     swap(tr[p][0], tr[p][1]);
98     if(tr[p][0])
99         addall(tr[p][0]);
100     maintain(p);
101     // 交换后下面都被更改了, 需要再次 maintain。
102 }
103 // 全部加一
104
105 int main() {
106     int n;
107     cin >> n;
108     std::vector<int> v(n + 1);
109     for(int i = 1; i <= n; ++i) {
110         cin >> v[i],
111         insert(root, v[i], 0);
112     }
113     cout << xorv[root] << endl;
114     // 查询总异或和
115     int m;
116     cin >> m;
117     for(int i = 1; i <= m; ++i) {
118         int x, y;
119         cin >> y >> x;
120         if(y == 0)
121             remove(root, x, 0);
122         // remove 和 addall 混用时小心 remove 掉不存在的
123         // 元素!
124         else
125             addall(root);
126         cout << xorv[root] << endl;
127     }

```

```

126 }
127
128 // merge
129 int merge(int p, int q) {
130     if(!p)
131         return q;
132     if(!q)
133         return p;
134     wei[p] += wei[q], xorv[p] ^= xorv[q];
135     tr[p][0] = merge(tr[p][0], tr[q][0]);
136     tr[p][1] = merge(tr[p][1], tr[q][1]);
137     return p;
138 }

```

6.4 Ac Automaton

```

1 // 求有多少个 s 在 t 中出现过。
2 namespace Ac_Automaton{
3     const int si = 1e6 + 10;
4     int root = 0, tot = 0;
5     int tr[si][27], End[si], fail[si];
6     int cal(char ch) { return (int)(ch - 'a') + 1; }
7     void init() {
8         tot = 0;
9         memset(tr, 0, sizeof tr);
10        memset(End, 0, sizeof End);
11        memset(fail, 0, sizeof fail);
12    }
13    void insert(char *s) {
14        int u = 0;
15        for(int i = 1; s[i]; ++i) {
16            if(!tr[u][cal(s[i])])
17                tr[u][cal(s[i])] = ++tot;
18            u = tr[u][cal(s[i])];
19        }
20        ++End[u];
21    }
22    std::queue<int>q;
23    void build() {
24        for(int i = 1; i <= 26; ++i)
25            if(tr[root][i]) q.push(tr[root][i]);
26        while(!q.empty()) {
27            int u = q.front();
28            q.pop();
29            for(int i = 1; i <= 26; ++i) {
30                if(tr[u][i])
31                    fail[tr[u][i]] = tr[fail[u]][i],
32                    q.push(tr[u][i]);
33                else tr[u][i] = tr[fail[u]][i];
34            }
35        }
36        int query(char *t) {
37            int u = 0, res = 0;
38            for(int i = 1; t[i]; ++i) {
39                u = tr[u][cal(t[i])];
40                for(int j = u; j && End[j] != -1; j = fail[j])
41                    res += End[j], End[j] = -1;
42            }
43            return res;
44        }
45    }

```

```

46 using namespace Ac_Automaton;
47
48 // 求次数。
49 namespace Ac_Automaton {
50     const int si = 2e6 + 10;
51     int root = 0, tot = 0, cnt_f = 0;
52     int tr[si][27], End[si], fail[si], cnt[si];
53     int cal(char ch) { return (int)(ch - 'a') + 1; }
54     void init() {
55         tot = 0;
56         memset(tr, 0, sizeof tr);
57         memset(cnt, 0, sizeof cnt);
58         memset(End, 0, sizeof End);
59         memset(fail, 0, sizeof fail);
60     }
61     void insert(char *s, int nu) {
62         int u = 0;
63         for(int i = 1; s[i]; ++i) {
64             if(!tr[u][cal(s[i])])
65                 tr[u][cal(s[i])] = ++tot;
66             u = tr[u][cal(s[i])];
67         }
68         End[nu] = u; // 这里改为记录第 nu 个模式串的结尾的位置。
69     }
70     int head[si];
71     struct Fail_Tree{ int ver, Next; }ft[si << 1];
72     void add(int u, int v) { ft[cnt_f] = (Fail_Tree){v,
73         head[u]}, head[u] = cnt_f++; }
74     std::queue<int>q;
75     void build() {
76         for(int i = 1; i <= 26; ++i)
77             if(tr[root][i]) q.push(tr[root][i]);
78         while(!q.empty()) {
79             int u = q.front();
80             add(fail[u], u), q.pop(); // 构建 Fail 树
81             for(int i = 1; i <= 26; ++i) {
82                 if(tr[u][i])
83                     fail[tr[u][i]] = tr[fail[u]][i],
84                     q.push(tr[u][i]);
85                 else tr[u][i] = tr[fail[u]][i];
86             }
87         }
88     }
89     void dfs(int u, int fa) {
90         for(int i = head[u]; ~i; i = ft[i].Next) {
91             int v = ft[i].ver;
92             if(v == fa) continue;
93             dfs(v, u), cnt[u] += cnt[v];
94         } // 统计
95     }
96     void query(char *t) {
97         int u = 0;
98         for(int i = 1; t[i]; ++i)
99             u = tr[u][cal(t[i])], ++cnt[u];
100         // 记录每个状态被匹配多少次
101         dfs(root, -1);
102         for(int i = 1; i <= n; ++i)
103             printf("%d\n", cnt[End[i]]);
104     }
105 }
106 using namespace Ac_Automaton;

```

Good Luck && Have Fun!