



河南大學

Henan University

# Dissonant Interval

Contestant

曹家宁

Jianing Cao

罗施达

Shida Luo

孙小文

Xiaowen Sun

目录

1	数据结构	1
1.1	树状数组	1
1.1.1	树状数组	1
1.1.2	树状数组 2	1
1.2	线段树	2
1.2.1	线段树 simple	2
1.2.2	线段树	3
1.2.3	动态开点线段树	5
1.2.4	主席树	6
1.2.5	标记永久化主席树	8
1.2.6	线段树优化建图	10
1.3	ST 表	11
1.4	并查集	11
1.5	可撤销并查集	12
1.6	带权并查集	13
1.7	字典树	13
1.8	左偏树	14
1.9	智慧集	15
1.10	欧拉序	16
1.11	波纹疾走树	17
1.12	Splay	19
2	图论	21
2.1	图的连通性	21
2.1.1	Tarjan 割点	21
2.1.2	Tarjan 割边	21
2.1.3	Tarjan 强连通分量	22
2.1.4	Tarjan 点双连通分量	23
2.1.5	Tarjan 边双连通分量	23

2.2	重链剖分	24
2.3	长链剖分	25
2.4	树的重心	26
2.5	欧拉回路	27
2.6	流和匹配	28
2.6.1	EdmondsKarp	28
2.6.2	二分图判定	29
2.6.3	二分图最大匹配	30
3	字符串	30
3.1	KMP	30
3.2	EXKMP	31
3.3	字符串哈希	31
3.4	马拉车	32
4	数学	32
4.1	组合数学	32
4.1.1	组合数	32
4.1.2	卢卡斯定理	33
4.2	数论	33
4.2.1	线性筛	33
4.2.2	区间筛	34
4.2.3	MillerRabin	34
4.2.4	PollardRho	35
4.2.5	中国剩余定理	36
4.2.6	扩展中国剩余定理	36
4.3	线性代数	37
4.3.1	矩阵	37
4.3.2	线性基	38
4.4	多项式	39
4.4.1	FFT	39

4.4.2	NTT . . . . .	39
4.4.3	Poly . . . . .	40
<b>5</b>	<b>计算几何</b>	<b>42</b>
5.1	凸包 . . . . .	42
<b>6</b>	<b>杂项</b>	<b>46</b>
6.1	动态 bitset . . . . .	46
6.2	取整 . . . . .	48
6.3	康托展开 . . . . .	49
6.4	逆康托展开 . . . . .	49
6.5	日期公式 . . . . .	50
6.6	高精度 . . . . .	50
6.7	高维前缀和 . . . . .	52
6.8	命令行 . . . . .	53
<b>7</b>	<b>附录</b>	<b>53</b>
7.1	编译参数 . . . . .	53
7.2	OJ 测试 . . . . .	53
7.3	组合数学公式 . . . . .	53
7.4	随机素数 . . . . .	54
7.5	常数表 . . . . .	54

# 1 数据结构

## 1.1 树状数组

### 1.1.1 树状数组

```
1 #include<bits/stdc++.h>
2
3 //树状数组(Fenwick)
4 //https://www.luogu.com.cn/problem/P3374
5 template<typename T>
6 struct Fenwick {
7     Fenwick(int n) : v(n + 1) {};
8     void update(int x, T dx) {
9         for(int i = x; i < v.size(); i += (i & -i)) {
10             v[i] += dx;
11         }
12     }
13     T query(int x) {
14         T res{};
15         for(int i = x; i > 0; i -= (i & -i)) {
16             res += v[i];
17         }
18         return res;
19     }
20     T range(int l, int r) {
21         return query(r) - query(l - 1);
22     }
23     T kth(T k) {
24         int pos = 0;
25         int logn = std::bit_width(v.size() - 1);
26         for(int i = 1 << (logn - 1); i > 0; i >>= 1) {
27             if(pos + i < v.size() && v[pos + i] < k) {
28                 k -= v[pos + i];
29                 pos += i;
30             }
31         }
32         return pos + 1;
33     }
34     std::vector<T> v;
35 };
36
37 int main() {
38     std::ios::sync_with_stdio(false);
39     std::cin.tie(nullptr);
40     int n, m;
41     std::cin >> n >> m;
42     Fenwick<int> tr(n);
```

```
43     for(int i = 1; i <= n; ++i) {
44         int x;
45         std::cin >> x;
46         tr.update(i, x);
47     }
48     for(int i = 0; i < m; ++i) {
49         int o, x, y;
50         std::cin >> o >> x >> y;
51         if(o == 1) {
52             tr.update(x, y);
53         } else if (o == 2) {
54             std::cout << tr.range(x, y) << '\n';
55         }
56     }
57     return 0;
58 };
```

### 1.1.2 树状数组 2

```
1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 template<typename T>
5 struct Fenwick {
6     Fenwick(int n) : vec(n + 1), add(n + 1) {}
7     void rangeUpdate(int l, int r, T dx) {
8         update(l, dx);
9         update(r + 1, -dx);
10    }
11    T rangeQuery(int l, int r) {
12        return query(r) - query(l - 1);
13    }
14    void update(int pos, T dx) {
15        for(int i = pos; i < vec.size(); i += (i & -i)) {
16            vec[i] += dx;
17            add[i] += (pos - 1) * dx;
18        }
19    }
20    T query(int pos) {
21        T res{};
22        for(int i = pos; i >= 1; i -= (i & -i)) {
23            res += pos * vec[i] - add[i];
24        }
25        return res;
26    }
27    std::vector<T> vec, add;
28 };
29
```

```

30 //树状数组，区间修改，区间查询
31 //https://www.luogu.com.cn/problem/P3372
32 int main() {
33     std::ios::sync_with_stdio(false);
34     std::cin.tie(nullptr);
35     int n, m;
36     std::cin >> n >> m;
37     Fenwick<i64> tr(n);
38     for(int i = 1; i <= n; ++i) {
39         int x;
40         std::cin >> x;
41         tr.rangeUpdate(i, i, x);
42     }
43     for(int i = 1; i <= m; ++i) {
44         int opt;
45         std::cin >> opt;
46         if(opt == 1) {
47             int l, r, dx;
48             std::cin >> l >> r >> dx;
49             tr.rangeUpdate(l, r, dx);
50         } else if(opt == 2) {
51             int l, r;
52             std::cin >> l >> r;
53             std::cout << tr.rangeQuery(l, r) << '\n';
54         }
55     }
56     return 0;
57 }

```

## 1.2 线段树

### 1.2.1 线段树 simple

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 template<typename Info>
5 struct SegmentTree {
6     #define ls (id<<1)
7     #define rs (id<<1|1)
8     SegmentTree() = default;
9     SegmentTree(int n) : n(n), info(n << 2) {}
10    SegmentTree(const SegmentTree<Info> &o) : n(o.n), info(o.info) {}
11    SegmentTree(const std::vector<Info> &init) : SegmentTree((int)init.size()) {
12        auto build = [&](auto self, int id, int l, int r) ->void {
13            if(l == r) {
14                info[id] = init[l];
15                return;

```

```

16        }
17        int mid = (l + r) / 2;
18        self(self, ls, l, mid);
19        self(self, rs, mid + 1, r);
20        pushup(id);
21    };
22    build(build, 1, 0, n - 1);
23 }
24 void pushup(int id) {
25     info[id] = info[ls] + info[rs];
26 }
27 void update(int pos, const Info &val) {
28     update(1, 0, n - 1, pos, val);
29 }
30 Info query(int pos) {
31     return rangeQuery(pos, pos);
32 }
33 Info rangeQuery(int l, int r) {
34     return rangeQuery(1, 0, n - 1, l, r);
35 }
36 void update(int id, int l, int r, int pos, const Info &val) {
37     if(l == r) {
38         info[id] = val;
39         return;
40     }
41     int mid = (l + r) / 2;
42     if(pos <= mid) {
43         update(ls, l, mid, pos, val);
44     } else {
45         update(rs, mid + 1, r, pos, val);
46     }
47     pushup(id);
48 }
49 Info rangeQuery(int id, int l, int r, int x, int y) {
50     if(x <= l && r <= y) {
51         return info[id];
52     }
53     int mid = (l + r) / 2;
54     Info res;
55     if(x <= mid) {
56         res = res + rangeQuery(ls, l, mid, x, y);
57     }
58     if(y > mid) {
59         res = res + rangeQuery(rs, mid + 1, r, x, y);
60     }
61     return res;
62 }
63 #undef ls

```

```

64 #undef rs
65     int n;
66     std::vector<Info> info;
67 };
68
69 constexpr int INF = 2E9;
70
71 struct Info {
72     Info() = default;
73     Info(int x, int idx) {
74         lmn = rmx = x - idx;
75         lmx = rmn = x + idx;
76     }
77     int lmn = INF;
78     int rmn = -INF;
79     int lmx = INF;
80     int rmx = -INF;
81     int ans = 0;
82 };
83
84 Info operator+(const Info &x, const Info &y) {
85     Info res;
86     res.lmx = std::max(x.lmx, y.lmx);
87     res.rmx = std::max(x.rmx, y.rmx);
88     res.lmn = std::min(x.lmn, y.lmn);
89     res.rmn = std::min(x.rmn, y.rmn);
90     res.ans = std::max({x.ans, y.ans, x.lmx - y.rmn, y.rmx - x.lmn});
91     return res;
92 }
93
94 void solve() {
95     int n, q;
96     std::cin >> n >> q;
97     std::vector<Info> v(n + 1);
98     for(int i = 1; i <= n; ++i) {
99         int x;
100         std::cin >> x;
101         v[i] = Info(x, i);
102     }
103     SegmentTree<Info> tr(v);
104     std::cout << tr.rangeQuery(1, n).ans << '\n';
105     for(int i = 1; i <= q; ++i) {
106         int idx, x;
107         std::cin >> idx >> x;
108         tr.update(idx, Info(x, idx));
109         std::cout << tr.rangeQuery(1, n).ans << '\n';
110     }
111 }

```

```

112
113 int main() {
114     std::ios::sync_with_stdio(false);
115     std::cin.tie(nullptr);
116     int T = 1;
117     std::cin >> T;
118     while(T--) {
119         solve();
120     }
121     return 0;
122 }

```

### 1.2.2 线段树

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 //线段树，区间修改，区间查询
5 //https://www.luogu.com.cn/problem/P3372
6 template<typename Info, typename Tag>
7 struct SegmentTree {
8     #define ls (id<<1)
9     #define rs (id<<1|1)
10     SegmentTree() = default;
11     SegmentTree(int n) : n(n), info(n << 2), tag(n << 2), len(n << 2) {}
12     SegmentTree(const std::vector<Info> &init) : SegmentTree((int)init.size()) {
13         auto build = [&](auto self, int id, int l, int r) ->void {
14             len[id] = r - l + 1;
15             if(l == r) {
16                 info[id] = init[l];
17                 return;
18             }
19             int mid = (l + r) / 2;
20             self(self, ls, l, mid);
21             self(self, rs, mid + 1, r);
22             pushup(id);
23         };
24         build(build, 1, 0, n - 1);
25     }
26     void apply(int id, const Tag &dx) {
27         info[id].apply(dx, len[id]);
28         tag[id].apply(dx);
29     }
30     void pushup(int id) {
31         info[id] = info[ls] + info[rs];
32     }
33     void pushdown(int id) {
34         apply(ls, tag[id]);

```

```

35     apply(rs, tag[id]);
36     tag[id] = Tag();
37 }
38 void rangeUpdate(int l, int r, const Tag &dx) {
39     rangeUpdate(1, 0, n - 1, l, r, dx);
40 }
41 void update(int pos, const Tag &dx) {
42     rangeUpdate(pos, pos, dx);
43 }
44 Info rangeQuery(int l, int r) {
45     return rangeQuery(1, 0, n - 1, l, r);
46 }
47 Info query(int pos) {
48     return rangeQuery(pos, pos);
49 }
50 void rangeUpdate(int id, int l, int r, int x, int y, const Tag &dx) {
51     if(x <= l && r <= y) {
52         apply(id, dx);
53         return;
54     }
55     int mid = (l + r) / 2;
56     pushdown(id);
57     if(x <= mid) {
58         rangeUpdate(ls, l, mid, x, y, dx);
59     }
60     if(y > mid) {
61         rangeUpdate(rs, mid + 1, r, x, y, dx);
62     }
63     pushup(id);
64 }
65 Info rangeQuery(int id, int l, int r, int x, int y) {
66     if(x <= l && r <= y) {
67         return info[id];
68     }
69     int mid = (l + r) / 2;
70     pushdown(id);
71     Info res;
72     if(x <= mid) {
73         res = res + rangeQuery(ls, l, mid, x, y);
74     }
75     if(y > mid) {
76         res = res + rangeQuery(rs, mid + 1, r, x, y);
77     }
78     return res;
79 }
80 #undef ls
81 #undef rs
82 int n;

```

```

83     std::vector<Info> info;
84     std::vector<Tag> tag;
85     std::vector<int> len;
86 };
87
88 constexpr i64 INF = 4E18;
89 i64 P = 571373;
90
91 struct Tag {
92     i64 add = 0;
93     i64 mul = 1;
94     void apply(const Tag &dx) {
95         mul = (mul * dx.mul) % P;
96         add = (add * dx.mul + dx.add) % P;
97     }
98 };
99
100 struct Info {
101     i64 sum = 0;
102     void apply(const Tag &dx, const int &len) {
103         sum = (sum * dx.mul + dx.add * len) % P;
104     }
105 };
106
107 Info operator+(const Info &x, const Info &y) {
108     Info res;
109     res.sum = (x.sum + y.sum) % P;
110     return res;
111 }
112
113 int main() {
114     std::ios::sync_with_stdio(false);
115     std::cin.tie(nullptr);
116     int n, m;
117     std::cin >> n >> m >> P;
118     std::vector<Info> v(n + 1);
119     for(int i = 1; i <= n; ++i) {
120         i64 x;
121         std::cin >> x;
122         v[i] = {x % P};
123     }
124     SegmentTree<Info, Tag> tr(v);
125     while(m--) {
126         i64 opt, x, y;
127         std::cin >> opt >> x >> y;
128         if(opt == 1) {
129             i64 k;
130             std::cin >> k;

```

```

131         tr.rangeUpdate(x, y, Tag(0, k));
132     } else if(opt == 2) {
133         i64 k;
134         std::cin >> k;
135         tr.rangeUpdate(x, y, Tag(k, 1));
136     } else if(opt == 3) {
137         std::cout << tr.rangeQuery(x, y).sum << '\n';
138     }
139 }
140 return 0;
141 }

```

### 1.2.3 动态开点线段树

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3 constexpr int MAXN = 2E5;
4
5 template<typename Info, typename T = i64>
6 struct SegmentTree {
7     struct Node {
8         Node* l = nullptr;
9         Node* r = nullptr;
10        Info info;
11    };
12    SegmentTree() = default;
13    SegmentTree(T n) : L(0), R(n - 1) {}
14    SegmentTree(T L, T R) : L(L), R(R) {}
15    void pushup(Node* id) {
16        id->info = (id->l == nullptr ? Info() : id->l->info)
17            + (id->r == nullptr ? Info() : id->r->info);
18    }
19    void update(T pos, const Info &val) {
20        update(root, L, R, pos, val);
21    }
22    Info query(T pos) {
23        return rangeQuery(pos, pos);
24    }
25    Info rangeQuery(T l, T r) {
26        return rangeQuery(root, L, R, l, r);
27    }
28    void update(Node* &id, T l, T r, T pos, const Info &val) {
29        if(id == nullptr) id = new Node();
30        if(l == r) {
31            id->info = val;
32            return;
33        }
34        T mid = (l + r - 1) / 2;

```

```

35        if(pos <= mid) {
36            update(id->l, l, mid, pos, val);
37        } else {
38            update(id->r, mid + 1, r, pos, val);
39        }
40        pushup(id);
41    }
42    Info rangeQuery(Node* &id, T l, T r, T x, T y) {
43        if(y < l || x > r || id == nullptr) return Info();
44        if(x <= l && r <= y) {
45            return id->info;
46        }
47        T mid = (l + r - 1) / 2;
48        return rangeQuery(id->l, l, mid, x, y)
49            + rangeQuery(id->r, mid + 1, r, x, y);
50    }
51
52    void merge(SegmentTree<Info, T> seg) {
53        root = merge(root, seg.root, L, R);
54    }
55    Node* merge(Node* &xid, Node* &yid, T l, T r) {
56        if(xid == nullptr) return yid;
57        if(yid == nullptr) return xid;
58        if(l == r) {
59            xid->info = (xid->info ^ yid->info);
60            return xid;
61        }
62        T mid = (l + r - 1) / 2;
63        xid->l = merge(xid->l, yid->l, l, mid);
64        xid->r = merge(xid->r, yid->r, mid + 1, r);
65        pushup(xid);
66        return xid;
67    }
68
69    SegmentTree<Info, T> split(T L, T R) { //分裂出[L, R]的部分
70        SegmentTree<Info, T> seg = split(L - 1);
71        SegmentTree<Info, T> rem = seg.split(R);
72        merge(rem);
73        return seg;
74    }
75    SegmentTree<Info, T> split(T k) { //分裂出(k, ∞]的部分
76        SegmentTree<Info, T> seg(L, R);
77        seg.root = split(root, L, R, k);
78        return seg;
79    }
80    Node* split(Node* &id, T l, T r, T k) {
81        if(id == nullptr || l == r || k >= r) return nullptr;
82        Node* nid = new Node();

```



```

83     if(k < l) {
84         std::swap(nid, id);
85         return nid;
86     }
87     T mid = (l + r - 1) / 2;
88     if(k > mid) {
89         nid->r = split(id->r, mid + 1, r, k);
90     } else {
91         nid->l = split(id->l, l, mid, k);
92         std::swap(nid->r, id->r);
93     }
94     pushup(id);
95     pushup(nid);
96     return nid;
97 }
98 T queryk(T k) { //非通用函数
99     return queryk(root, L, R, k);
100 }
101 T queryk(Node *id, T l, T r, T k) {
102     if(id == nullptr) return -1;
103     if(id->info.sum < k) return -1;
104     if(l == r) return l;
105     int mid = (l + r - 1) / 2;
106     if(id->l != nullptr && id->l->info.sum >= k) {
107         return queryk(id->l, l, mid, k);
108     } else if(id->r != nullptr) {
109         return queryk(id->r, mid + 1, r, k - (id->l == nullptr ? 0 : id->l->info
110 .sum));
111     }
112     return -1;
113 }
114 T L, R;
115 Node* root = nullptr;
116 };
117 struct Info {
118     Info() = default;
119     Info(i64 _val) {
120         sum = _val;
121     }
122     i64 sum = 0;
123 };
124
125 Info operator+(const Info &x, const Info &y) {
126     Info res;
127     res.sum = x.sum + y.sum;
128     return res;
129 }

```

```

130
131 Info operator^(const Info &x, const Info &y) {
132     Info res;
133     res.sum = x.sum + y.sum;
134     return res;
135 }
136
137 int main() {
138     std::ios::sync_with_stdio(false);
139     std::cin.tie(nullptr);
140     int n, m, idx = 1;
141     std::cin >> n >> m;
142     std::vector<SegmentTree<Info>> segs(n + 1, SegmentTree<Info>(MAXN + 1));
143     for(int i = 1; i <= n; ++i) {
144         int x;
145         std::cin >> x;
146         segs[idx].update(i, x);
147     }
148     for(int i = 0; i < m; ++i) {
149         int opt;
150         std::cin >> opt;
151         if(opt == 0) {
152             int p, x, y;
153             std::cin >> p >> x >> y;
154             segs[++idx] = segs[p].split(x, y);
155         } else if(opt == 1) {
156             int p, t;
157             std::cin >> p >> t;
158             segs[p].merge(segs[t]);
159         } else if(opt == 2) {
160             int p, x, q;
161             std::cin >> p >> x >> q;
162             segs[p].update(q, segs[p].query(q).sum + x);
163         } else if(opt == 3) {
164             int p, x, y;
165             std::cin >> p >> x >> y;
166             std::cout << segs[p].rangeQuery(x, y).sum << '\n';
167         } else if(opt == 4) {
168             int p, k;
169             std::cin >> p >> k;
170             std::cout << segs[p].queryk(k) << '\n';
171         }
172     }
173     return 0;
174 }

```

## 1.2.4 主席树

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 template<typename Info, typename Tag>
5 struct PersistentTree {
6     struct Node {
7         int l = 0, r = 0;
8         Info info;
9         Tag tag;
10    };
11    #define ls(x) (node[x].l)
12    #define rs(x) (node[x].r)
13    PersistentTree(int n) : PersistentTree(std::vector<Info>(n + 1)) {}
14    PersistentTree(const std::vector<Info> &init) : n((int)init.size() - 1) {
15        node.reserve(n << 3);
16        auto build = [&](auto self, int l, int r) ->int {
17            node.push_back(Node());
18            int id = node.size() - 1;
19            if(l == r) {
20                node[id].info = init[l];
21            } else {
22                int mid = (l + r) / 2;
23                ls(id) = self(self, l, mid);
24                rs(id) = self(self, mid + 1, r);
25                node[id].info = node[ls(id)].info + node[rs(id)].info;
26            }
27            return id;
28        };
29        root.push_back(build(build, 1, n));
30    };
31    int update(int version, int pos, const Info &val) {
32        root.push_back(update(root[version], 1, n, pos, val));
33        return root.size() - 1;
34    }
35    int update(int version, int pos, const Tag &dx) {
36        root.push_back(update(root[version], 1, n, pos, dx));
37        return root.size() - 1;
38    }
39    Info query(int version, int pos) {
40        return rangeQuery(version, pos, pos);
41    }
42    Info rangeQuery(int version, int l, int r) {
43        return rangeQuery(root[version], 1, n, l, r);
44    }
45    int update(int lst, int l, int r, const int &pos, const Info &val) {
46        node.push_back(node[lst]);
47        int id = node.size() - 1;
48        if(l == r) {

```

```

49        node[id].info = val;
50    } else {
51        int mid = (l + r) / 2;
52        if(pos <= mid) {
53            ls(id) = update(ls(lst), l, mid, pos, val);
54        } else if(pos > mid) {
55            rs(id) = update(rs(lst), mid + 1, r, pos, val);
56        }
57        node[id].info = node[ls(id)].info + node[rs(id)].info;
58    }
59    return id;
60 }
61 int update(int lst, int l, int r, const int &pos, const Tag &dx) {
62     node.push_back(node[lst]);
63     int id = node.size() - 1;
64     if(l == r) {
65         node[id].info.apply(dx);
66     } else {
67         int mid = (l + r) / 2;
68         if(pos <= mid) {
69             ls(id) = update(ls(lst), l, mid, pos, dx);
70         } else if(pos > mid) {
71             rs(id) = update(rs(lst), mid + 1, r, pos, dx);
72         }
73         node[id].info = node[ls(id)].info + node[rs(id)].info;
74     }
75     return id;
76 }
77 Info rangeQuery(int id, int l, int r, const int &x, const int &y) {
78     if(x <= l && r <= y) {
79         return node[id].info;
80     }
81     int mid = (l + r) / 2;
82     Info res;
83     if(x <= mid) {
84         res = res + rangeQuery(ls(id), l, mid, x, y);
85     }
86     if(y > mid) {
87         res = res + rangeQuery(rs(id), mid + 1, r, x, y);
88     }
89     return res;
90 }
91 int kth(int versionl, int versionr, int k) {
92     return kth(root[versionl], root[versionr], 1, n, k);
93 }
94 int kth(int idx, int idy, int l, int r, int k) { //静态区间第k小, 不支持修改
95     if(l >= r) return l;
96     int mid = (l + r) / 2;

```

```

97     int dx = node[ls(idy)].info.sum - node[rs(idy)].info.sum;
98     if(dx >= k) {
99         return kth(ls(idy), ls(idy), l, mid, k);
100     } else {
101         return kth(rs(idy), rs(idy), mid + 1, r, k - dx);
102     }
103 }
104 #undef ls
105 #undef rs
106 const int n;
107 std::vector<Node> node;
108 std::vector<int> root;
109 };
110
111 struct Tag {
112     Tag(int dx = 0) : add(dx) {}
113     int add = 0;
114     void apply(const Tag &dx) {
115         add += dx.add;
116     }
117 };
118
119 struct Info {
120     int sum = 0;
121     void apply(const Tag &dx) {
122         sum += dx.add;
123     }
124 };
125
126 Info operator+(const Info &x, const Info &y) {
127     Info res;
128     res.sum = x.sum + y.sum;
129     return res;
130 }
131 //主席树(单点修改, 历史版本区间查询, 静态区间第k小)
132 //https://www.luogu.com.cn/problem/P3834
133 int main() {
134     std::ios::sync_with_stdio(false);
135     std::cin.tie(nullptr);
136     int n, q;
137     std::cin >> n >> q;
138     std::vector<int> v(n + 1), tmp(n + 1);
139     for(int i = 1; i <= n; ++i) {
140         std::cin >> v[i];
141         tmp[i] = v[i];
142     }
143     std::sort(tmp.begin() + 1, tmp.end());
144     tmp.erase(std::unique(tmp.begin() + 1, tmp.end()), tmp.end());

```

```

145     int m = tmp.size() - 1;
146     PersistentTree<Info, Tag> tr(std::vector<Info>(m + 1));
147     std::vector<int> version(n + 1);
148     version[0] = tr.root.size() - 1;
149     for(int i = 1; i <= n; ++i) {
150         int pos = std::lower_bound(tmp.begin() + 1, tmp.end(), v[i]) - tmp.begin();
151         version[i] = tr.update(version[i - 1], pos, Tag(1));
152     }
153     for(int i = 1; i <= q; ++i) {
154         int l, r, k;
155         std::cin >> l >> r >> k;
156         int pos = tr.kth(version[l - 1], version[r], k);
157         std::cout << tmp[pos] << '\n';
158     }
159     return 0;
160 }

```

### 1.2.5 标记永久化主席树

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 template<typename Info, typename Tag>
5 struct PersistentTree {
6     struct Node {
7         int l = 0, r = 0;
8         Info info;
9         Tag tag;
10    };
11    #define ls(x) (node[id].l)
12    #define rs(x) (node[id].r)
13    PersistentTree(int n) : n(n) {}
14    PersistentTree(const std::vector<Info> &init) : PersistentTree((int)init.size()
15        - 1) {
16        node.reserve(n << 3);
17        auto build = [&](auto self, int l, int r) ->int {
18            node.push_back(Node());
19            int id = node.size() - 1;
20            if(l == r) {
21                node[id].info = init[l];
22            } else {
23                int mid = (l + r) / 2;
24                ls(id) = self(self, l, mid);
25                rs(id) = self(self, mid + 1, r);
26                node[id].info = node[ls(id)].info + node[rs(id)].info;
27            }
28            return id;
29        };

```

```

29     root.push_back(build(build, 1, n));
30 };
31 int update(int version, int t, const Tag &dx) {
32     return rangeUpdate(version, t, t, dx);
33 }
34 Info query(int version, int t) {
35     return rangeQuery(version, t, t);
36 }
37 int rangeUpdate(int version, int l, int r, const Tag &dx) {
38     root.push_back(rangeUpdate(root[version], 1, n, l, r, dx));
39     return root.size() - 1;
40 }
41 Info rangeQuery(int version, int l, int r) {
42     return rangeQuery(root[version], 1, n, l, r);
43 }
44 int rangeUpdate(int lst, int l, int r, const int &x, const int &y, const Tag &dx
45 ) {
46     node.push_back(node[lst]);
47     int id = node.size() - 1;
48     node[id].info.apply(std::min(r, y) - std::max(l, x) + 1, dx);
49     if(x <= l && r <= y) {
50         node[id].tag.apply(dx);
51     } else {
52         int mid = (l + r) / 2;
53         if(x <= mid) {
54             ls(id) = rangeUpdate(ls(lst), l, mid, x, y, dx);
55         }
56         if(y > mid) {
57             rs(id) = rangeUpdate(rs(lst), mid + 1, r, x, y, dx);
58         }
59     }
60     return id;
61 }
62 Info rangeQuery(int id, int l, int r, const int &x, const int &y) {
63     if(x <= l && r <= y) {
64         return node[id].info;
65     }
66     int mid = (l + r) / 2;
67     Info res;
68     if(x <= mid) {
69         res = res + rangeQuery(ls(id), l, mid, x, y);
70     }
71     if(y > mid) {
72         res = res + rangeQuery(rs(id), mid + 1, r, x, y);
73     }
74     res.apply(std::min(r, y) - std::max(l, x) + 1, node[id].tag);
75     return res;
76 }

```

```

76 #undef ls
77 #undef rs
78     const int n;
79     std::vector<Node> node;
80     std::vector<int> root;
81 };
82
83 struct Tag {
84     Tag(int dx = 0) : add(dx) {}
85     int add = 0;
86     void apply(const Tag &dx) {
87         add += dx.add;
88     }
89 };
90
91 struct Info {
92     int sum = 0;
93     void apply(int len, const Tag &dx) {
94         sum += 1LL * len * dx.add;
95     }
96 };
97
98 Info operator+(const Info &x, const Info &y) {
99     Info res;
100     res.sum = x.sum + y.sum;
101     return res;
102 }
103
104 //可持久化线段树(区间修改, 区间历史查询)
105 //https://www.luogu.com.cn/problem/P3919
106 int main() {
107     std::ios::sync_with_stdio(false);
108     std::cin.tie(nullptr);
109     int n, q;
110     std::cin >> n >> q;
111     std::vector<Info> v(n + 1);
112     for(int i = 1; i <= n; ++i) {
113         std::cin >> v[i].sum;
114     }
115     PersistentTree<Info, Tag> tr(v);
116     std::vector<int> version(q + 1);
117     for(int i = 1; i <= q; ++i) {
118         int ver, opt, pos;
119         std::cin >> ver >> opt >> pos;
120         if(opt == 1) {
121             int x;
122             std::cin >> x;
123             int lst = tr.query(version[ver], pos).sum;

```

```

124         version[i] = tr.update(version[ver], pos, Tag(x - lst));
125     } else if(opt == 2) {
126         std::cout << tr.query(version[ver], pos).sum << '\n';
127         version[i] = version[ver];
128     }
129 }
130 return 0;
131 }

```

## 1.2.6 线段树优化建图

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 struct STOG {
5     #define ls (id<<1)
6     #define rs (id<<1|1)
7     STOG(int n) : n(n), in(n << 2), out(n << 2), v(n * 7) {
8         int tot = n;
9         auto build = [&](auto self, int id, int l, int r) ->void {
10             if(l == r) {
11                 in[id] = out[id] = l;
12                 return;
13             }
14             int mid = (l + r) / 2;
15             self(self, ls, l, mid);
16             self(self, rs, mid + 1, r);
17             in[id] = ++tot;
18             out[id] = ++tot;
19             update(in[id], in[ls], 0);
20             update(in[id], in[rs], 0);
21             update(out[ls], out[id], 0);
22             update(out[rs], out[id], 0);
23         };
24         build(build, 1, 1, n);
25     }
26     void update(int x, int y, int w) { //连一条从x 到 y的边, 边权为w
27         v[x].emplace_back(y, w);
28     }
29     //model == 0 时, 从pos 到 [x, y]连边, 边权为w
30     //model == 1 时, 从[x, y] 到 pos连边, 边权为w
31     void rangeUpdate(int pos, int x, int y, int w, int model) {
32         rangeUpdate(1, 1, n, pos, x, y, w, model);
33     }
34     void rangeUpdate(int id, int l, int r, int pos, int x, int y, int w, auto model)
35     {
36         if(x <= l && r <= y) {

```

```

37             update(pos, in[id], w);
38         } else {
39             update(out[id], pos, w);
40         }
41         return;
42     }
43     int mid = (l + r) / 2;
44     if(x <= mid) {
45         rangeUpdate(ls, l, mid, pos, x, y, w, model);
46     }
47     if(y > mid) {
48         rangeUpdate(rs, mid + 1, r, pos, x, y, w, model);
49     }
50 }
51 #undef ls
52 #undef rs
53 int n;
54 std::vector<int> in, out;
55 std::vector<std::vector<std::pair<int, int>>> v;
56 };
57
58 int main() {
59     std::ios::sync_with_stdio(false);
60     std::cin.tie(nullptr);
61     int n, q, s;
62     std::cin >> n >> q >> s;
63     STOG tr(n);
64     for(int i = 1; i <= q; ++i) {
65         int opt;
66         std::cin >> opt;
67         if(opt == 1) {
68             int pos, x, w;
69             std::cin >> pos >> x >> w;
70             tr.update(pos, x, w);
71         } else if(opt == 2) {
72             int pos, x, y, w;
73             std::cin >> pos >> x >> y >> w;
74             tr.rangeUpdate(pos, x, y, w, 0);
75         } else if(opt == 3) {
76             int pos, x, y, w;
77             std::cin >> pos >> x >> y >> w;
78             tr.rangeUpdate(pos, x, y, w, 1);
79         }
80     }
81     auto &graph = tr.v;
82     int m = tr.v.size() - 1;
83     std::vector<i64> dp(m + 1, LLONG_MAX);
84     std::priority_queue<std::pair<i64, int>, std::vector<std::pair<i64, int>>, std::

```

```

    greater<>> pq;
85 pq.emplace(0LL, s);
86 while(!pq.empty()) {
87     auto [w, id] = pq.top();
88     pq.pop();
89     if(w >= dp[id]) continue;
90     dp[id] = w;
91     for(const auto &[nxt, dx] : graph[id]) {
92         i64 ww = w + dx;
93         if(ww < dp[nxt]) {
94             pq.emplace(ww, nxt);
95         }
96     }
97 }
98 for(int i = 1; i <= n; ++i) {
99     std::cout << (dp[i] == LLONG_MAX ? -1 : dp[i]) << " \n"[i == n];
100 }
101 return 0;
102 }

```

### 1.3 ST 表

时间复杂度:

- 初始化:  $O(n \log(n))$
- 查询:  $O(1)$

空间复杂度:  $O(n \log(n))$

用途: RMQ 问题, 不支持修改

模板题: [Luogu P3865](https://www.luogu.com.cn/problem/P3865)

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 template <typename T, typename Func = std::function<T(const T&, const T&)>>
5 struct ST {
6     ST() = default;
7     ST(const std::vector<T> &v, Func func = [](const T& a, const T& b) {
8         return std::max(a, b);
9     }) : func(std::move(func)) {
10         int k = std::__lg(v.size());
11         st = std::vector<std::vector<T>>(k + 1, std::vector<T>(v.size()));
12         st[0] = v;
13         for(int i = 0; i < k; ++i) {
14             for(int j = 0; j + (1 << (i + 1)) - 1 < v.size(); ++j) {
15                 st[i + 1][j] = this->func(st[i][j], st[i][j + (1 << i)]);
16             }
17         }
18     }

```

```

19 T range(int l, int r) {
20     int t = std::__lg(r - l + 1);
21     return func(st[t][l], st[t][r + 1 - (1 << t)]);
22 }
23 std::vector<std::vector<T>> st;
24 Func func;
25 };
26
27 //ST表(sparseTable)
28 //https://www.luogu.com.cn/problem/P3865
29 int main() {
30     std::ios::sync_with_stdio(false);
31     std::cin.tie(nullptr);
32     int n, q;
33     std::cin >> n >> q;
34     std::vector<int> v(n + 1);
35     for(int i = 1; i <= n; ++i) {
36         std::cin >> v[i];
37     }
38     ST<int> st(v);
39     while(q--) {
40         int l, r;
41         std::cin >> l >> r;
42         std::cout << st.range(l, r) << '\n';
43     }
44     return 0;
45 }

```

### 1.4 并查集

```

1 #include <bits/stdc++.h>
2
3 //并查集(disjoint set union)
4 //https://www.luogu.com.cn/problem/P3367
5 struct DSU {
6     DSU() = default;
7     DSU(int n) : p(n), sz(n, 1) {
8         std::iota(p.begin(), p.end(), 0);
9     }
10     int find(int x) {
11         return p[x] == x ? x : p[x] = find(p[x]);
12     }
13     bool same(int x, int y) {
14         return find(x) == find(y);
15     }
16     int merge(int x, int y) {
17         if (same(x, y)) return 0;
18         x = find(x), y = find(y);

```

```

19     if (sz[x] < sz[y]) std::swap(x, y);
20     sz[x] += sz[y];
21     p[y] = x;
22     return x;
23 }
24 int& size(int x) {
25     return sz[find(x)];
26 }
27 std::vector<int> p, sz;
28 };
29
30 int main() {
31     std::ios::sync_with_stdio(false);
32     std::cin.tie(nullptr);
33     int n, m;
34     std::cin >> n >> m;
35     DSU dsu(n);
36     for(int i = 0; i < m; ++i) {
37         int z, x, y;
38         std::cin >> z >> x >> y;
39         if(z == 1) {
40             dsu.merge(x, y);
41         } else if(z == 2) {
42             std::cout << (dsu.same(x, y) ? 'Y' : 'N') << '\n';
43         }
44     }
45     return 0;
46 }

```

## 1.5 可撤销并查集

```

1 #include <bits/stdc++.h>
2
3 struct RDSU {
4     RDSU() = default;
5     RDSU(int n) : p(n), sz(n, 1) {
6         std::iota(p.begin(), p.end(), 0);
7     }
8     int find(int x) {
9         while(p[x] != x) x = p[x];
10        return x;
11    }
12    bool same(int x, int y) {
13        return find(x) == find(y);
14    }
15    int merge(int x, int y) {
16        if (same(x, y)) return 0;
17        x = find(x), y = find(y);

```

```

18        if (sz[x] < sz[y]) std::swap(x, y);
19        hsz.push({sz[x], sz[x]});
20        hfa.push({p[y], p[y]});
21        sz[x] += sz[y];
22        p[y] = x;
23        return x;
24    }
25    int& size(int x) {
26        return sz[find(x)];
27    }
28    size_t now() {
29        return hsz.size();
30    }
31    void version(int ver) {
32        rollback(now() - ver);
33    }
34    void rollback(int t = 1) {
35        for(int i = 1; i <= t && !hfa.empty(); ++i) {
36            hfa.top().first = hfa.top().second;
37            hsz.top().first = hsz.top().second;
38            hfa.pop(), hsz.pop();
39        }
40    }
41    std::vector<int> p, sz;
42    std::stack<std::pair<int&, int>> hsz, hfa;
43 };
44
45 //https://www.starrycoding.com/problem/9
46 int main() {
47     std::ios::sync_with_stdio(false);
48     std::cin.tie(nullptr);
49     int n, q;
50     std::cin >> n >> q;
51     RDSU rdsu(n);
52     for(int i = 1; i <= q; ++i) {
53         int opt;
54         std::cin >> opt;
55         if(opt == 1) {
56             int x, y;
57             std::cin >> x >> y;
58             rdsu.merge(x, y);
59         } else if(opt == 2) {
60             rdsu.rollback();
61         } else if(opt == 3) {
62             int x, y;
63             std::cin >> x >> y;
64             std::cout << (rdsu.same(x, y) ? "YES" : "NO") << '\n';
65         }

```

```

66 }
67 return 0;
68 }

```

## 1.6 带权并查集

模板题: [Luogu P1196](#)

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 struct DSUT {
5     DSUT() = default;
6     DSUT(int n) : fa(n), f(n), sz(n, 1) {
7         std::iota(fa.begin(), fa.end(), 0);
8     }
9     int find(int id) {
10         if(id == fa[id]) return id;
11         int root = find(fa[id]);
12         f[id] += f[fa[id]];
13         return fa[id] = root;
14     }
15     void merge(int x, int y) { //要依据题意修改
16         int fx = find(x), fy = find(y);
17         fa[fx] = fy;
18         f[fx] += sz[fy];
19         sz[fy] += sz[fx];
20     }
21     bool query(int x, int y) {
22         return find(x) == find(y);
23     }
24     void set(int pos, int val) {
25         f[pos] = val;
26     }
27     int get(int pos) {
28         return f[pos];
29     }
30     std::vector<int> fa, f, sz; //父节点, 到父节点的权值, 集合大小
31 };
32
33 int main() {
34     std::ios::sync_with_stdio(false);
35     std::cin.tie(nullptr);
36     int n;
37     std::cin >> n;
38     DSUT dsut(n);
39     for(int i = 1; i <= n; ++i) {
40         char opt;

```

```

41         int x, y;
42         std::cin >> opt >> x >> y;
43         if(opt == 'M') {
44             dsut.merge(x, y);
45         } else if(opt == 'C') {
46             if(!dsut.query(x, y)) {
47                 std::cout << -1 << '\n';
48             } else {
49                 std::cout << std::abs(dsut.get(x) - dsut.get(y)) - 1 << '\n';
50             }
51         }
52     }
53     return 0;
54 }

```

## 1.7 字典树

```

1 #include <bits/stdc++.h>
2
3 template<int N = 26>
4 struct Trie {
5     Trie() = default;
6     Trie(const std::string &rgx) : tree(1) {
7         int k = 0;
8         for(int i = 0, dx; i < rgx.size(); i += dx + 1) {
9             dx = 2 * (i + 2 < rgx.size() && rgx[i + 1] == '-');
10             for(int j = rgx[i]; j <= rgx[i + dx]; ++j) {
11                 dict[j] = k++;
12             }
13         }
14     }
15     void insert(const std::string &s) {
16         int cur = 0;
17         for(const auto &ch : s) {
18             int &nxt = tree[cur][dict[ch]];
19             if(nxt == 0) nxt = ++idx;
20             cur = nxt;
21             if(tree.size() <= cur) {
22                 tree.resize(cur + s.size());
23                 tot.resize(cur + s.size());
24             }
25             tot[cur]++;
26         }
27     }
28     int find(const std::string &s) {
29         int cur = 0;
30         for(const auto &ch : s) {
31             int &nxt = tree[cur][dict[ch]];

```



```

32         if(nxt == 0) return 0;
33         cur = nxt;
34     }
35     return tot[cur];
36 }
37
38 int idx = 0;
39 std::vector<int> tot;
40 std::array<int, 128> dict{};
41 std::vector<std::array<int, N>> tree;
42 };
43
44 void solve() {
45     int n, q;
46     std::cin >> n >> q;
47     Trie<62> t("0-9A-Za-z");
48     for(int i = 1; i <= n; ++i) {
49         std::string s;
50         std::cin >> s;
51         t.insert(s);
52     }
53     for(int i = 1; i <= q; ++i) {
54         std::string s;
55         std::cin >> s;
56         std::cout << t.find(s) << '\n';
57     }
58 }
59
60 int main() {
61     std::ios::sync_with_stdio(false);
62     std::cin.tie(nullptr);
63     int T = 1;
64     std::cin >> T;
65     while(T--) {
66         solve();
67     }
68     return 0;
69 }

```

## 1.8 左偏树

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 template<typename T, typename _Compare = std::less<T>>
5 struct LeftistHeap {
6     struct Node {
7         Node(const T &_info) : info(_info){}

```

```

8         T info;
9         int dis = -1;
10        Node *ls = nullptr;
11        Node *rs = nullptr;
12    };
13    LeftistHeap() = default;
14    LeftistHeap(Comp _cmp) : cmp(std::move(_cmp)) {}
15    void merge(LeftistHeap &o) {
16        _size += o.size();
17        root = merge(root, o.root);
18    }
19    int dis(Node* &node) {
20        return node == nullptr ? -1 : node->dis;
21    }
22    Node* merge(Node* &x, Node* &y) {
23        if(x == nullptr) return y;
24        if(y == nullptr) return x;
25        if(cmp(x->info, y->info)) std::swap(x, y);
26        x->rs = merge(x->rs, y);
27        if(dis(x->ls) < dis(x->rs)) {
28            std::swap(x->ls, x->rs);
29        }
30        x->dis = dis(x->rs) + 1;
31        return x;
32    }
33    void push(const T &info) {
34        root = merge(root, new Node(info));
35        ++_size;
36    }
37    void pop() {
38        Node* temp = root;
39        root = merge(root->ls, root->rs);
40        delete temp;
41        --_size;
42    }
43    T top() {
44        assert(root != nullptr);
45        return root->info;
46    }
47    size_t size() {
48        return _size;
49    }
50    bool empty() {
51        return _size == 0;
52    }
53    _Compare cmp;
54    Node* root = nullptr;
55    size_t _size = 0;

```

```

56 };
57
58 int main() {
59     std::ios::sync_with_stdio(false);
60     std::cin.tie(nullptr);
61
62
63     return 0;
64 }

```

## 1.9 智慧集

时间复杂度:

- 插入:  $O(\log(n))$
- 删除:  $O(\log(n))$
- 第  $k$  小:  $O(1)$  前提: 每次操作  $k$  变化不大

空间复杂度:  $O(n)$

用途: 双指针中位数

模板题: 2023ICPC-Jinan-Regional K. Rainbow Subarray

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 template<typename T>
5 struct SmartSet {
6     std::multiset<T> small, large;
7     T smallSum, largeSum;
8     SmartSet(): small(), large(), smallSum(), largeSum() {}
9     void insert(T val) {
10         if(small.empty() || val > *small.rbegin()) {
11             large.insert(val);
12             largeSum += val;
13         } else {
14             small.insert(val);
15             smallSum += val;
16         }
17     }
18     void erase(T val) {
19         if(small.contains(val)) {
20             smallSum -= val;
21             small.extract(val);
22         } else if(large.contains(val)) {
23             largeSum -= val;
24             large.extract(val);
25         }
26     }
27     void balance(int k) {

```

```

28         k = std::max(0, std::min(k, size()));
29         while(small.size() > k) {
30             T val = *small.rbegin();
31             smallSum -= val;
32             largeSum += val;
33             large.insert(val);
34             small.extract(val);
35         }
36         while(small.size() < k) {
37             T val = *large.begin();
38             smallSum += val;
39             largeSum -= val;
40             small.insert(val);
41             large.extract(val);
42         }
43     }
44     int size() {
45         return small.size() + large.size();
46     }
47     int smallSize(int k) {
48         balance(k);
49         return small.size();
50     }
51     int largeSize(int k) {
52         balance(k);
53         return large.size();
54     }
55     T kth(int k) {
56         balance(k);
57         return *small.rbegin();
58     }
59     T getSmallSum(int k) {
60         balance(k);
61         return smallSum;
62     }
63     T getLargeSum(int k) {
64         balance(k);
65         return largeSum;
66     }
67 };
68
69 void solve() {
70     i64 n, k;
71     std::cin >> n >> k;
72     std::vector<i64> v(n + 1);
73     for(int i = 1; i <= n; ++i) {
74         std::cin >> v[i];
75         v[i] = v[i] - i + n;

```

```

76 }
77 SmartSet<i64> sst;
78 int ans = 1;
79 for(int i = 1, j = 1; j <= n; ++j) {
80     sst.insert(v[j]);
81     while(true) {
82         int len = (j - i + 1);
83         int mid = (len + 1) / 2;
84         i64 target = sst.kth(mid);
85         i64 res = 1LL * target * (sst.smallSize(mid) - sst.largeSize(mid)) + sst
.getLargeSum(mid) - sst.getSmallSum(mid);
86         if(res > k) {
87             sst.erase(v[i]);
88             ++i;
89         } else {
90             ans = std::max(ans, j - i + 1);
91             break;
92         }
93     }
94 }
95 }
96 std::cout << ans << '\n';
97 }
98
99
100 int main() {
101     std::ios::sync_with_stdio(false);
102     std::cin.tie(nullptr);
103     int T = 1;
104     std::cin >> T;
105     while(T--) {
106         solve();
107     }
108 }

```

## 1.10 欧拉序

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long i64;
4
5 struct SparseTable {
6     SparseTable() = default;
7     vector<int> v;
8     vector<vector<int>> st;
9     void init(vector<int> &v_) {
10         v = v_;
11         int k = __lg(v.size());

```

```

12     st = vector<vector<int>>(k + 1, vector<int>(v.size()));
13     iota(st[0].begin(), st[0].end(), 0);
14     for (int i = 0; i < k; ++i) {
15         for (int j = 0; j + (1 << (i + 1)) - 1 < v.size(); ++j) {
16             if (v[st[i][j]] < v[st[i][j + (1 << i)]]
17                 st[i + 1][j] = st[i][j];
18             else
19                 st[i + 1][j] = st[i][j + (1 << i)];
20         }
21     }
22 }
23 int query_id(int l, int r) {
24     int t = __lg(r - l + 1);
25     if (v[st[t][l]] < v[st[t][r + 1 - (1 << t)]]
26         return st[t][l];
27     else
28         return st[t][r + 1 - (1 << t)];
29 }
30 };
31
32 struct Euler_tours {
33     int n, cnt = 0;
34     vector<vector<int>> graph;
35     vector<int> et_dep, id, et;
36     SparseTable st; //节点个数, 图
37     Euler_tours(int n) : n(n), graph(n + 1), id(n + 1), et_dep(2 * n), et(2 * n) {}
38
39     void add_edg(int u, int v) {
40         graph[u].push_back(v);
41         graph[v].push_back(u);
42     }
43
44     void dfs(int u, int fa, int dep) {
45         et_dep[++cnt] = dep;
46         et[cnt] = u;
47         id[u] = cnt;
48         for (auto v : graph[u]) {
49             if (v != fa) {
50                 dfs(v, u, dep + 1);
51                 et_dep[++cnt] = dep;
52                 et[cnt] = u;
53             }
54         }
55         return;
56     }
57
58     void init(int root = 1) {
59         dfs(root, 0, 1);

```

```

60     st.init(et_dep);
61 }
62
63 int lca(int u, int v) {
64     int idu = id[u];
65     int idv = id[v];
66     if (idu > idv) {
67         swap(idu, idv);
68     }
69     int idlca = st.query_id(idu, idv);
70     return et[idlca];
71 }
72 };
73
74 void solve() {
75     int n, q, root;
76     cin >> n >> q >> root;
77     Euler_tours et(n);
78     for (i64 i = 1; i < n; i++) {
79         i64 u, v;
80         cin >> u >> v;
81         et.add_edg(u, v);
82     }
83     et.init(root);
84     while (q--) {
85         i64 u, v;
86         cin >> u >> v;
87         cout << et.lca(u, v) << "\n";
88     }
89     return;
90 }
91 int main() {
92     ios::sync_with_stdio(false);
93     cin.tie(0), cout.tie(0);
94     i64 T = 1;
95     // cin >> T;
96     while (T--) {
97         solve();
98     }
99     return 0;
100 }

```

### 1.11 波纹疾走树

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 struct BitRank {

```

```

5     // block 管理一行一行的bit
6     std::vector<unsigned long long> block;
7     std::vector<unsigned int> count;
8     BitRank() {}
9     // 位向量长度
10    void resize(const unsigned int num) {
11        block.resize(((num + 1) >> 6) + 1, 0);
12        count.resize(block.size(), 0);
13    }
14    // 设置i位bit
15    void set(const unsigned int i, const unsigned long long val) {
16        block[i >> 6] |= (val << (i & 63));
17    }
18    void build() {
19        for (unsigned int i = 1; i < block.size(); i++) {
20            count[i] = count[i - 1] + __builtin_popcountll(block[i - 1]);
21        }
22    }
23    // [0, i) 1的个数
24    unsigned int rank1(const unsigned int i) const {
25        return count[i >> 6] + __builtin_popcountll(block[i >> 6] & ((1ULL << (i & 63)) - 1ULL));
26    }
27    // [i, j) 1的个数
28    unsigned int rank1(const unsigned int i, const unsigned int j) const {
29        return rank1(j) - rank1(i);
30    }
31    // [0, i) 0的个数
32    unsigned int rank0(const unsigned int i) const {
33        return i - rank1(i);
34    }
35    // [i, j) 0的个数
36    unsigned int rank0(const unsigned int i, const unsigned int j) const {
37        return rank0(j) - rank0(i);
38    }
39 };
40
41 class WaveletMatrix {
42 private:
43     unsigned int height;
44     std::vector<BitRank> B;
45     std::vector<int> pos;
46 public:
47     WaveletMatrix() {}
48     WaveletMatrix(std::vector<int> vec) : WaveletMatrix(vec, *std::max_element(vec.
49         begin(), vec.end()) + 1) {}
50     // sigma: 字母表大小(字符串的话), 数字序列的话是数的种类

```

```

51 WaveletMatrix(std::vector<int> vec, const unsigned int sigma) {
52     height = (sigma == 1) ? 1 : (64 - __builtin_clzll(sigma - 1));
53     B.resize(height), pos.resize(height);
54     for (unsigned int i = 0; i < height; ++i) {
55         B[i].resize(vec.size());
56         for (unsigned int j = 0; j < vec.size(); ++j) {
57             B[i].set(j, get(vec[j], height - i - 1));
58         }
59         B[i].build();
60         auto it = stable_partition(vec.begin(), vec.end(), [&](int c) {
61             return !get(c, height - i - 1);
62         });
63         pos[i] = it - vec.begin();
64     }
65 }
66
67 int get(const int val, const int i) {
68     return (val >> i) & 1;
69 }
70
71 // [l, r] 中val出现的频率
72 int rank(const int l, const int r, const int val) {
73     return rank(r, val) - rank(l - 1, val);
74 }
75
76 // [0, i] 中val出现的频率
77 int rank(int i, int val) {
78     ++i;
79     int p = 0;
80     for (unsigned int j = 0; j < height; ++j) {
81         if (get(val, height - j - 1)) {
82             p = pos[j] + B[j].rank1(p);
83             i = pos[j] + B[j].rank1(i);
84         } else {
85             p = B[j].rank0(p);
86             i = B[j].rank0(i);
87         }
88     }
89     return i - p;
90 }
91
92 // [l, r] 中k小
93 int kth(int l, int r, int k) {
94     ++r;
95     int res = 0;
96     for (unsigned int i = 0; i < height; ++i) {
97         const int j = B[i].rank0(l, r);
98         if (j >= k) {

```

```

99             l = B[i].rank0(l);
100             r = B[i].rank0(r);
101         } else {
102             l = pos[i] + B[i].rank1(l);
103             r = pos[i] + B[i].rank1(r);
104             k -= j;
105             res |= (1 << (height - i - 1));
106         }
107     }
108     return res;
109 }
110
111 // [l,r] 在[a, b] 值域的数字个数
112 int rangeFreq(const int l, const int r, const int a, const int b) {
113     return rangeFreq(l, r + 1, a, b + 1, 0, 1 << height, 0);
114 }
115 int rangeFreq(const int i, const int j, const int a, const int b, const int l,
116 const int r, const int x) {
117     if (i == j || r <= a || b <= l) return 0;
118     const int mid = (l + r) >> 1;
119     if (a <= l && r <= b) {
120         return j - i;
121     } else {
122         const int left = rangeFreq(B[x].rank0(i), B[x].rank0(j), a, b, l, mid, x
123 + 1);
124         const int right = rangeFreq(pos[x] + B[x].rank1(i), pos[x] + B[x].rank1(
125 j), a, b, mid, r, x + 1);
126         return left + right;
127     }
128 }
129
130 // [l,r] 在[a,b] 值域内存在的最小值是什么, 不存在返回-1, 只支持非负整数
131 int rangeMin(int l, int r, int a, int b) {
132     return rangeMin(l, r + 1, a, b + 1, 0, 1 << height, 0, 0);
133 }
134 int rangeMin(const int i, const int j, const int a, const int b, const int l,
135 const int r, const int x, const int val) {
136     if (i == j || r <= a || b <= l) return -1;
137     if (r - l == 1) return val;
138     const int mid = (l + r) >> 1;
139     const int res = rangeMin(B[x].rank0(i), B[x].rank0(j), a, b, l, mid, x + 1,
140 val);
141     if (res < 0) {
142         return rangeMin(pos[x] + B[x].rank1(i), pos[x] + B[x].rank1(j), a, b,
143 mid, r, x + 1, val + (1 << (height - x - 1)));
144     } else {
145         return res;
146     }
147 }

```

```

141     }
142 };
143
144 //波纹疾走树(区间第k小, 区间val出现的频率, 区间在值域出现的次数和最小值)
145 //https://www.luogu.com.cn/problem/P3834
146 int main() {
147     std::ios::sync_with_stdio(false);
148     std::cin.tie(0);
149     int n, q;
150     std::cin >> n >> q;
151     std::vector<int> v(n + 1);
152     for(int i = 1; i <= n; ++i) {
153         std::cin >> v[i];
154     }
155     WaveletMatrix wlm(v);
156     for(int i = 1; i <= q; ++i) {
157         int l, r, k;
158         std::cin >> l >> r >> k;
159         std::cout << wlm.kth(l, r, k) << '\n';
160     }
161     return 0;
162 }

```

## 1.12 Splay

```

1 #include <bits/stdc++.h>
2
3 class SplayTree {
4 public:
5     SplayTree() {
6         tr.push_back(Node());
7         insert(INF);
8         insert(-INF);
9     }
10    void insert(int t) { //插入值为t的数
11        int id = root, fa = 0;
12        while(id && tr[id].val != t) {
13            fa = id;
14            id = tr[id].nxt[t > tr[id].val];
15        }
16        if(id) {
17            tr[id].cnt++;
18        } else {
19            id = ++size;
20            tr[fa].nxt[t > tr[fa].val] = id;
21            tr.push_back(Node(fa, t));
22        }
23        splay(id);

```

```

24    }
25    int get_pre(int t) { //查找t的前驱节点
26        find(t);
27        int id = root;
28        if(tr[id].val < t) return id;
29        id = tr[id].nxt[0];
30        while(tr[id].nxt[1]) {
31            id = tr[id].nxt[1];
32        }
33        splay(id);
34        return id;
35    }
36    int get_suc(int t) { //查找t的后继节点
37        find(t);
38        int id = root;
39        if(tr[id].val > t) return id;
40        id = tr[id].nxt[1];
41        while(tr[id].nxt[0]) {
42            id = tr[id].nxt[0];
43        }
44        splay(id);
45        return id;
46    }
47    void find(int t) { //查找值为t的节点, 并将该节点转到根
48        int id = root;
49        while(tr[id].nxt[t > tr[id].val] && t != tr[id].val) {
50            id = tr[id].nxt[t > tr[id].val];
51        }
52        splay(id);
53    }
54    void erase(int t) { //删除值为t的, 只删除1个
55        int pre = get_pre(t);
56        int suc = get_suc(t);
57        splay(pre);
58        splay(suc, pre);
59        int tid = tr[suc].nxt[0]; //目标节点
60        if(tr[tid].cnt > 1) {
61            tr[tid].cnt--;
62            splay(tid); //向上更新其他节点
63        } else {
64            tr[suc].nxt[0] = 0;
65            splay(suc); //向上更新其他节点
66        }
67    }
68    int get_root() {
69        return root;
70    }
71    int get_rank(int t) { //查一个数t的排名

```

```

72     insert(t);
73     int res = tr[tr[root].nxt[0]].size;
74     erase(t);
75     return res;
76 }
77 int get_kth(int t) { //查找第k个节点编号
78     t++; //有哨兵，所以++
79     int id = root;
80     while(true) {
81         pushdown(id); //向下传递懒标记
82         const auto &[x, y] = tr[id].nxt;
83         if(tr[x].size + tr[id].cnt < t) {
84             t -= tr[x].size + tr[id].cnt;
85             id = y;
86         } else {
87             if(tr[x].size >= t) {
88                 id = tr[id].nxt[0];
89             } else {
90                 return id;
91             }
92         }
93     }
94 }
95 int get_val(int t) { //查找排名为t的数的数值
96     int id = get_kth(t);
97     splay(id);
98     return tr[id].val;
99 }
100 void reverse(int l, int r) { //反转区间[l, r]
101     l = get_kth(l - 1), r = get_kth(r + 1);
102     splay(l, 0), splay(r, l);
103     tr[tr[r].nxt[0]].tag ^= 1;
104 }
105 void output(int id) { //中序遍历
106     pushdown(id);
107     const auto &[x, y] = tr[id].nxt;
108     if(x != 0) output(x);
109     if(std::abs(tr[id].val) != INF) {
110         std::cout << tr[id].val << ' ';
111     }
112     if(y) output(y);
113 }
114 int val(int id) {
115     return tr[id].val;
116 }
117 private:
118     class Node {
119     public:

```

```

120     Node() {
121         nxt = {0, 0};
122         lst = val = size = cnt = tag = 0;
123     }
124     Node(int _lst, int _val) : lst(_lst), val(_val) {
125         nxt = {0, 0};
126         tag = 0;
127         size = cnt = 1;
128     }
129     std::array<int, 2> nxt; //左右节点[0左, 1右]
130     int lst; //父亲
131     int val; //权值
132     int cnt; //权值数
133     int size; //子树大小
134     int tag; //懒标记[1翻, 0不翻]
135 };
136 void rotate(int id) {
137     int pid = tr[id].lst, gid = tr[pid].lst; //父节点, 爷节点
138     int k = (tr[pid].nxt[1] == id); //判断id是pid的左节点还是右节点
139     tr[pid].nxt[k] = tr[id].nxt[k ^ 1]; //将父节点的k号子节点设置为id的k^1号
    子节点
140     tr[tr[id].nxt[k ^ 1]].lst = pid; //id的k^1号子节点的父节点设为pid
141     tr[id].nxt[k ^ 1] = pid; //id的k^1号子节点设置为pid
142     tr[pid].lst = id; //pid的父节点设置为id
143     tr[id].lst = gid; //id的父节点设置为gid
144     tr[gid].nxt[tr[gid].nxt[1] == pid] = id; //gid的子节点设为id
145     pushup(pid); //更新pid
146     pushup(id); //更新id
147 }
148 void splay(int id, int t = 0) { //将id旋转到为t的子节点, 为0时id为根
149     while(tr[id].lst != t) {
150         int pid = tr[id].lst, gid = tr[pid].lst;
151         if(gid != t) { //非根做双旋
152             if((tr[pid].nxt[0] == id) == (tr[gid].nxt[0] == pid)) { //直线式转中
153                 rotate(pid);
154             } else { //折线式转中
155                 rotate(id);
156             }
157         }
158         rotate(id);
159     }
160     if(t == 0) root = id;
161 }
162 void pushup(int id) {
163     const auto &[x, y] = tr[id].nxt;
164     tr[id].size = tr[x].size + tr[y].size + tr[id].cnt;
165 }
166 void pushdown(int id) {

```

```

167         if(tr[id].tag) {
168             auto &x, y = tr[id].nxt;
169             std::swap(x, y);
170             tr[x].tag ^= 1;
171             tr[y].tag ^= 1;
172             tr[id].tag = 0;
173         }
174     }
175     std::vector<Node> tr;
176     int root = 0; //根节点编号
177     int size = 0; //节点个数
178     const int INF = INT_MAX;
179 };
180
181 int main() {
182     std::ios::sync_with_stdio(false);
183     std::cin.tie(nullptr);
184     int n, m;
185     std::cin >> n >> m;
186     SplayTree tr;
187     for(int i = 1; i <= n; ++i) {
188         tr.insert(i);
189     }
190     for(int i = 1; i <= m; ++i) {
191         int l, r;
192         std::cin >> l >> r;
193         tr.reverse(l, r);
194     }
195     tr.output(tr.get_root());
196     return 0;
197 }

```

## 2 图论

### 2.1 图的连通性

#### 2.1.1 Tarjan 割点

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 //tarjan求割点
5 //https://www.luogu.com.cn/problem/P3388
6 int main() {
7     std::ios::sync_with_stdio(false);
8     std::cin.tie(nullptr);
9     int n, m;

```

```

10     std::cin >> n >> m;
11     std::vector<std::vector<int>> v(n + 1);
12     for(int i = 1; i <= m; ++i) {
13         int x, y;
14         std::cin >> x >> y;
15         v[x].push_back(y);
16         v[y].push_back(x);
17     }
18     std::vector<int> dfn(n + 1), low(n + 1), cutPoint(n + 1);
19     int cnt = 0, root = 0;
20     auto dfs = [&](auto self, int id, int lst) ->void {
21         dfn[id] = low[id] = ++cnt;
22         int sz = 0; //儿子个数
23         for(auto nxt : v[id]) {
24             if(!dfn[nxt]) {
25                 sz++;
26                 self(self, nxt, id);
27                 low[id] = std::min(low[id], low[nxt]);
28                 if(low[nxt] >= dfn[id]) {
29                     cutPoint[id] = 1;
30                 }
31             } else if(nxt != lst) {
32                 low[id] = std::min(low[id], dfn[nxt]);
33             }
34         }
35         if(sz <= 1 && id == root) {
36             cutPoint[id] = 0;
37         }
38     };
39     for(int i = 1; i <= n; ++i) {
40         if(!dfn[i]) {
41             root = i;
42             dfs(dfs, i, 0);
43         }
44     }
45     std::cout << std::count(cutPoint.begin() + 1, cutPoint.end(), 1) << '\n';
46     for(int i = 1; i <= n; ++i) {
47         if(cutPoint[i] == 1) {
48             std::cout << i << ' ';
49         }
50     }
51     return 0;
52 }

```

#### 2.1.2 Tarjan 割边

```

1 #include <bits/stdc++.h>
2 using i64 = long long;

```



```

3
4 //tarjan求割边
5 //https://www.luogu.com.cn/problem/P1656
6 int main() {
7     std::ios::sync_with_stdio(false);
8     std::cin.tie(nullptr);
9     int n, m;
10    std::cin >> n >> m;
11    std::vector<std::vector<std::pair<int, int>>> v(n + 1);
12    for(int i = 1; i <= m; ++i) {
13        int x, y;
14        std::cin >> x >> y;
15        v[x].push_back({y, i}); //记录边id(从1开始), 防止重边
16        v[y].push_back({x, i});
17    }
18    std::vector<int> dfn(n + 1), low(n + 1);
19    std::vector<std::pair<int, int>> bridge;
20    int cnt = 0;
21    auto dfs = [&](auto self, int id, int lid) ->void {
22        dfn[id] = low[id] = ++cnt;
23        for(auto [nxt, eid] : v[id]) {
24            if(!dfn[nxt]) {
25                self(self, nxt, eid);
26                low[id] = std::min(low[id], low[nxt]);
27                if(low[nxt] == dfn[nxt]) { //是割边
28                    bridge.push_back({id, nxt});
29                }
30            } else if(eid != lid) {
31                low[id] = std::min(low[id], dfn[nxt]);
32            }
33        }
34    };
35    for(int i = 1; i <= n; ++i) {
36        if(!dfn[i]) {
37            dfs(dfs, i, 0);
38        }
39    }
40    std::sort(bridge.begin(), bridge.end());
41    for(auto [x, y] : bridge) {
42        std::cout << x << ' ' << y << '\n';
43    }
44    return 0;
45 }

```

### 2.1.3 Tarjan 强连通分量

```

1 #include <bits/stdc++.h>
2 using i64 = long long;

```

```

3
4 //tarjan求强连通分量(scc)
5 //https://www.luogu.com.cn/problem/B3609
6 int main() {
7     std::ios::sync_with_stdio(false);
8     std::cin.tie(nullptr);
9     int n, m;
10    std::cin >> n >> m;
11    std::vector<std::vector<int>> v(n + 1);
12    for(int i = 0; i < m; ++i) {
13        int x, y;
14        std::cin >> x >> y;
15        v[x].push_back(y);
16    }
17    std::vector<std::vector<int>> scc(n + 1);
18    std::vector<int> dfn(n + 1), low(n + 1), ins(n + 1), bel(n + 1);
19    std::stack<int> stk;
20    int cnt = 0, tot = 0;
21    auto dfs = [&](auto self, int id) ->void {
22        dfn[id] = low[id] = ++cnt;
23        stk.push(id);
24        ins[id] = 1;
25        for(auto nxt : v[id]) {
26            if(!dfn[nxt]) {
27                self(self, nxt);
28                low[id] = std::min(low[id], low[nxt]);
29            } else if(ins[nxt]) {
30                low[id] = std::min(low[id], low[nxt]);
31            }
32        }
33        if(dfn[id] == low[id]) {
34            ++tot;
35            while(true) {
36                int num = stk.top();
37                stk.pop();
38                ins[num] = 0;
39                bel[num] = tot;
40                scc[tot].push_back(num);
41                if(id == num) break;
42            }
43        }
44    };
45    for(int i = 1; i <= n; ++i) {
46        if(!dfn[i]) {
47            dfs(dfs, i);
48        }
49    }
50    for(int i = 1; i <= tot; ++i) {

```

```

51     std::sort(scc[i].begin(), scc[i].end());
52 }
53 std::sort(scc.begin() + 1, scc.begin() + tot + 1);
54 std::cout << tot << '\n';
55 for(int i = 1; i <= tot; ++i) {
56     for(int j = 0; j < scc[i].size(); ++j) {
57         std::cout << scc[i][j] << " \n"[j == scc[i].size() - 1];
58     }
59 }
60 return 0;
61 }

```

#### 2.1.4 Tarjan 点双连通分量

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 //tarjan求点双连通分量
5 //https://www.luogu.com.cn/problem/P8435
6 int main() {
7     std::ios::sync_with_stdio(false);
8     std::cin.tie(nullptr);
9     int n, m;
10    std::cin >> n >> m;
11    std::vector<std::vector<int>>> v(n + 1);
12    for(int i = 1; i <= m; ++i) {
13        int x, y;
14        std::cin >> x >> y;
15        v[x].push_back(y);
16        v[y].push_back(x);
17    }
18    std::vector<std::vector<int>>> vcc(n + 1);
19    std::vector<int> dfn(n + 1), low(n + 1);
20    std::stack<int> stk;
21    int cnt = 0, tot = 0;
22    auto dfs = [&](auto self, int id, int lst) ->void {
23        dfn[id] = low[id] = ++cnt;
24        stk.push(id);
25        int sz = 0;
26        for(auto nxt : v[id]) {
27            if(!dfn[nxt]) {
28                sz++;
29                self(self, nxt, id);
30                low[id] = std::min(low[id], low[nxt]);
31                if(low[nxt] >= dfn[id]) {
32                    ++tot;
33                    while(true) {
34                        int num = stk.top();

```

```

35                        stk.pop();
36                        vcc[tot].push_back(num);
37                        if(num == nxt) break;
38                    }
39                    vcc[tot].push_back(id);
40                }
41            } else if(nxt != lst) {
42                low[id] = std::min(low[id], dfn[nxt]);
43            }
44        }
45        if(lst == 0 && sz == 0) {
46            ++tot;
47            vcc[tot].push_back(id);
48        }
49    };
50    for(int i = 1; i <= n; ++i) {
51        if(!dfn[i]) {
52            dfs(dfs, i, 0);
53        }
54    }
55    std::cout << tot << '\n';
56    for(int i = 1; i <= tot; ++i) {
57        std::cout << vcc[i].size() << ' ';
58        for(int j = 0; j < vcc[i].size(); ++j) {
59            std::cout << vcc[i][j] << " \n"[j == vcc[i].size() - 1];
60        }
61    }
62    return 0;
63 }

```

#### 2.1.5 Tarjan 边双连通分量

用途：求边双连通分量

模板题：洛谷 P8436

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 int main() {
5     std::ios::sync_with_stdio(false);
6     std::cin.tie(nullptr);
7     int n, m;
8     std::cin >> n >> m;
9     std::vector<std::vector<std::pair<int, int>>>> v(n + 1);
10    for(int i = 1; i <= m; ++i) {
11        int x, y;
12        std::cin >> x >> y;
13        v[x].push_back({y, i});

```

```

14     v[y].push_back({x, i});
15 }
16 std::vector<std::vector<int>> ecc(n + 1);
17 std::vector<int> dfn(n + 1), low(n + 1);
18 std::stack<int> stk;
19 int cnt = 0, tot = 0;
20 auto dfs = [&](auto self, int id, int lid) ->void {
21     dfn[id] = low[id] = ++cnt;
22     stk.push(id);
23     for(auto [nxt, eid] : v[id]) {
24         if(!dfn[nxt]) {
25             self(self, nxt, eid);
26             low[id] = std::min(low[id], low[nxt]);
27         } else if(lid != eid) {
28             low[id] = std::min(low[id], dfn[nxt]);
29         }
30     }
31     if(dfn[id] == low[id]) {
32         ++tot;
33         while(true) {
34             int num = stk.top();
35             ecc[tot].push_back(num);
36             stk.pop();
37             if(id == num) break;
38         }
39     }
40 };
41 for(int i = 1; i <= n; ++i) {
42     if(!dfn[i]) {
43         dfs(dfs, i, 0);
44     }
45 }
46 std::cout << tot << '\n';
47 for(int i = 1; i <= tot; ++i) {
48     std::cout << ecc[i].size() << ' ';
49     for(int j = 0; j < ecc[i].size(); ++j) {
50         std::cout << ecc[i][j] << " \n"[j == ecc[i].size() - 1];
51     }
52 }
53 return 0;
54 }

```

## 2.2 重链剖分

```

1 #include <bits/stdc++.h>
2
3 struct HLD {
4     HLD(const int &n) : n(_n), v(_n + 1) {

```

```

5         fa = dep = son = sz = top = in = out = rin = std::vector<int>(_n + 1);
6     }
7
8     void addEdge(const int &x, const int &y) {
9         v[x].push_back(y);
10        v[y].push_back(x);
11    }
12
13    void dfs1(int id, int &t) {
14        sz[id] = 1;
15        in[id] = t;
16        rin[t] = id;
17        for(const auto &nxt : v[id]) {
18            if(nxt == fa[id]) continue;
19            fa[nxt] = id;
20            dep[nxt] = dep[id] + 1;
21            dfs1(nxt, ++t);
22            sz[id] += sz[nxt];
23            if(sz[son[id]] < sz[nxt]) {
24                son[id] = nxt;
25            }
26        }
27        out[id] = t;
28    }
29
30    void dfs2(int id, int t) {
31        top[id] = t;
32        if(son[id] == 0) return;
33        dfs2(son[id], t);
34        for(const auto &nxt : v[id]) {
35            if(nxt == fa[id] || nxt == son[id]) continue;
36            dfs2(nxt, nxt);
37        }
38    }
39
40    void work(int root = 1) {
41        int dfsn = 1;
42        dfs1(root, dfsn);
43        dfs2(root, root);
44    }
45
46    bool isAncestor(int x, int y) {
47        return in[x] <= in[y] && out[x] >= out[y];
48    }
49
50    int lca(int x, int y) {
51        while(top[x] != top[y]) {
52            if(dep[top[x]] < dep[top[y]]) {
53                std::swap(x, y);
54            }
55        }
56    }

```

```

53     x = fa[top[x]];
54 }
55 return (dep[x] < dep[y] ? x : y);
56 }
57
58 int dis(int x, int y) {
59     return dep[x] + dep[y] - 2 * dep[lca(x, y)];
60 }
61
62 int kth(int id, int k) {
63     if(k > dep[id]) return 0;
64     while(dep[id] - dep[top[id]] + 1 <= k) {
65         k -= (dep[id] - dep[top[id]] + 1);
66         id = fa[top[id]];
67     }
68     return rin[in[id] - k];
69 }
70
71 std::vector<std::vector<int>> v;
72 std::vector<int> fa, dep, son, sz, top, in, out, rin;
73 int n;
74 };
75
76 //树链剖分求LCA
77 //https://www.luogu.com.cn/problem/P3379
78 int main() {
79     std::ios::sync_with_stdio(0);
80     std::cin.tie(nullptr);
81     int n, m, s;
82     std::cin >> n >> m >> s;
83     HLD tree(n);
84     for(int i = 0; i < n - 1; ++i) {
85         int x, y;
86         std::cin >> x >> y;
87         tree.addEdge(x, y);
88     }
89     tree.work(s);
90     for(int i = 0; i < m; ++i) {
91         int x, y;
92         std::cin >> x >> y;
93         std::cout << tree.lca(x, y) << '\n';
94     }
95     return 0;
96 }

```

## 2.3 长链剖分

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

```

2 using i64 = long long;
3
4 struct LLD {
5     LLD(const int &n) : n(_n) {
6         son = len = dep = top = in = out = lg = std::vector<int>(n + 1);
7         v = up = down = std::vector<std::vector<int>>(n + 1);
8         m = std::bit_width(std::bit_ceil((unsigned)n));
9         fa = std::vector(m + 1, std::vector<int>(n + 1));
10        for(int i = 2; i <= n; ++i) {
11            lg[i] = lg[i >> 1] + 1;
12        }
13    }
14
15    void addEdge(const int &x, const int &y) {
16        v[x].push_back(y);
17        v[y].push_back(x);
18    }
19
20    void dfs1(int id, int &t) {
21        for(int i = 0; i < m; ++i) {
22            fa[i + 1][id] = fa[i][fa[i][id]];
23        }
24        in[id] = t;
25        len[id] = 1;
26        for(const auto &nxt : v[id]) {
27            if(nxt == fa[0][id]) continue;
28            fa[0][nxt] = id;
29            dep[nxt] = dep[id] + 1;
30            dfs1(nxt, ++t);
31            if(len[nxt] + 1 > len[id]) {
32                len[id] = len[nxt] + 1;
33                son[id] = nxt;
34            }
35        }
36        out[id] = t;
37    }
38
39    void dfs2(int id, int t) {
40        top[id] = t;
41        if(son[id] == 0) return;
42        dfs2(son[id], t);
43        for(const auto &nxt : v[id]) {
44            if(nxt == fa[0][id] || nxt == son[id]) continue;
45            dfs2(nxt, nxt);
46        }
47    }
48
49    void work(int root = 1) {

```

```

50     int dfsn = 1;
51     dfs1(root, dfsn);
52     dfs2(root, root);
53     for(int i = 1; i <= n; ++i) {
54         if(top[i] != i) continue;
55         for(int j = 1, now = i; j <= len[i] && now; ++j, now = fa[0][now]) {
56             up[i].push_back(now);
57         }
58         for(int j = 1, now = i; j <= len[i] && now; ++j, now = son[now]) {
59             down[i].push_back(now);
60         }
61     }
62 }
63
64 bool isAncestor(int x, int y) { //x是y的祖先
65     return in[x] <= in[y] && out[x] >= out[y];
66 }
67
68 int lca(int x, int y) {
69     while(top[x] != top[y]) {
70         if(dep[top[x]] < dep[top[y]]) {
71             std::swap(x, y);
72         }
73         x = fa[0][top[x]];
74     }
75     return (dep[x] < dep[y] ? x : y);
76 }
77
78 int dis(int x, int y) {
79     return dep[x] + dep[y] - 2 * dep[lca(x, y)];
80 }
81
82 int kth(int id, int k) {
83     if(k == 0) return id;
84     int t = lg[k];
85     k -= (1 << t);
86     id = fa[t][id];
87     int p = top[id];
88     if(dep[id] - dep[p] >= k) {
89         id = down[p][(dep[id] - dep[p]) - k];
90     } else {
91         id = up[p][k - (dep[id] - dep[p])];
92     }
93     return id;
94 }
95
96 std::vector<std::vector<int>> v, up, down, fa;
97 std::vector<int> son, len, dep, top, in, out, lg;

```

```

98     int n, m;
99 };
100
101 int main() {
102     std::ios::sync_with_stdio(false);
103     std::cin.tie(nullptr);
104     int n, m, s;
105     std::cin >> n >> m >> s;
106     LLD tree(n);
107     for(int i = 1; i <= n - 1; ++i) {
108         int x, y;
109         std::cin >> x >> y;
110         tree.addEdge(x, y);
111     }
112     tree.work(s);
113     for(int i = 1; i <= m; ++i) {
114         int x, y;
115         std::cin >> x >> y;
116         std::cout << tree.lca(x, y) << '\n';
117     }
118     return 0;
119 }

```

## 2.4 树的重心

**定义：**如果在树中选择某个节点并删除，这棵树将分为若干棵子树，统计子树节点数并记录最大值。取遍树上所有节点，使此最大值取到最小的节点被称为整个树的重心。

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 void solve() {
5     int n;
6     std::cin >> n;
7     std::vector<std::vector<int>> v(n + 1);
8     for(int i = 1; i <= n - 1; ++i) {
9         int x, y;
10        std::cin >> x >> y;
11        v[x].push_back(y);
12        v[y].push_back(x);
13    }
14    std::vector<int> sz(n + 1), weight(n + 1);
15    int ans = n;
16    auto dfs = [&](auto self, int id, int lst) ->void {
17        sz[id] = 1;
18        for(auto nxt : v[id]) {
19            if(nxt == lst) continue;
20            self(self, nxt, id);

```

```

21         weight[id] = std::max(weight[id], sz[nxt]);
22         sz[id] += sz[nxt];
23     }
24     weight[id] = std::max(weight[id], n - sz[id]);
25     ans = std::min(ans, weight[id]);
26 };
27 dfs(dfs, 1, 0);
28 for(int i = 1; i <= n; ++i) {
29     if(weight[i] == ans) {
30         std::cout << i << ' ';
31         break;
32     }
33 }
34 std::cout << ans << '\n';
35 }
36 //树的重心 (重心最多有两个)
37 //http://bailian.openjudge.cn/practice/1655/
38 int main() {
39     std::ios::sync_with_stdio(false);
40     std::cin.tie(nullptr);
41     int T = 1;
42     std::cin >> T;
43     while(T--) {
44         solve();
45     }
46     return 0;
47 }

```

## 2.5 欧拉回路

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 // 无向图欧拉回路or通路
4 struct Euler {
5     int id = 0;
6     vector<int> edg;           // 存储边
7     vector<vector<int>> graph; // 存储图
8     // 初始化
9     Euler(int n, int m) {
10         graph.resize(n + 1);
11         edg.resize(m + 1);
12     }
13     // 添加边
14     void add_edge(int u, int v) {
15         id++;
16         graph[u].push_back(id);
17         graph[v].push_back(id);
18         edg[id] = u ^ v;

```

```

19     }
20     // 判断是否存在欧拉通路and返回起点
21     int is_semiEuler() {
22         int n = graph.size() - 1;
23         int odd = 0, start = 0;
24         for (int i = 1; i <= n; i++) {
25             if (graph[i].size() & 1) {
26                 odd++;
27                 start = i;
28             }
29         }
30         if (odd == 0) return 1;
31         if (odd == 2) return start;
32         return 0;
33     }
34     // 判断是否存在欧拉回路
35     bool is_Euler() {
36         int n = graph.size() - 1;
37         for (int i = 1; i <= n; i++) {
38             if (graph[i].size() & 1) {
39                 return false;
40             }
41         }
42         return true;
43     }
44     // 求解欧拉回路or通路
45     vector<int> Euler_tour(int start) {
46         vector<int> tour;
47         function<void(int)> dfs = [&](int u) {
48             while (!graph[u].empty()) {
49                 int i = graph[u].back();
50                 graph[u].pop_back();
51                 if (edg[i] == -1) continue;
52                 int v = edg[i] ^ u;
53                 edg[i] = -1;
54                 dfs(v);
55             }
56             tour.push_back(u);
57         };
58         dfs(start);
59         return tour;
60     }
61 };
62 // 有向图欧拉回路or通路
63 struct Directed_Euler {
64     vector<int> inE, outE;       // 存储入,出度
65     vector<vector<int>> graph;   // 存储图
66     int n, m;

```

```

67 // 初始化
68 Directed_Euler(int n, int m) : n(n), m(m) {
69     graph.resize(n + 1);
70     inE.resize(n + 1);
71     outE.resize(n + 1);
72 }
73 // 添加边
74 void add_edge(int u, int v) {
75     graph[u].push_back(v);
76     outE[u]++;
77     inE[v]++;
78 }
79 // 判断是否存在欧拉通路and返回起点
80 int is_semiEuler() {
81     int odd = 0, neodd = 0, start = 0;
82     for (int i = 1; i <= n; i++) {
83         if (outE[i] - inE[i] == 1) {
84             odd++;
85             start = i;
86         } else if (inE[i] - outE[i] == 1) {
87             neodd++;
88         } else if (inE[i] != outE[i]) {
89             return 0;
90         }
91     }
92     if (odd == 1 && neodd == 1) {
93         return start;
94     }
95     if (odd == 0 && neodd == 0) {
96         return 1;
97     }
98     return 0;
99 }
100 // 判断是否存在欧拉回路
101 bool is_Euler() {
102     int n = graph.size() - 1;
103     for (int i = 1; i <= n; i++) {
104         if (inE[i] != outE[i])
105             return false;
106     }
107     return true;
108 }
109 // 求解欧拉回路or通路
110 vector<int> Euler_tour(int start) {
111     vector<int> tour;
112     function<void(int)> dfs = [&](int u) {
113         while (!graph[u].empty()) {
114             int v = graph[u].back();

```

```

115         graph[u].pop_back();
116         dfs(v);
117     }
118     tour.push_back(u);
119 };
120 dfs(start);
121 return tour; // 返回的是逆序的欧拉回路or通路
122 }
123 };

```

## 2.6 流和匹配

### 2.6.1 EdmondsKarp

时间复杂度:  $O(|V||E|^2)$  实际情况一般远低于此复杂度

空间复杂度:  $O(|V| + |E|)$

用途: 求最大流

模板题: 洛谷 P3376

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 template<typename T>
5 struct MaxFlow {
6     struct Edge {
7         Edge() = default;
8         Edge(int _nxt, T _cap, int _enxt) : nxt(_nxt), cap(_cap), enxt(_enxt) {}
9         int nxt, enxt;
10        T cap;
11    };
12    MaxFlow(int n) : head(n, -1), pre(n), mf(n) {}
13    void addEdge(int x, int y, T cap) {
14        edge.push_back(Edge(y, cap, head[x]));
15        head[x] = edge.size() - 1;
16        edge.push_back(Edge(x, 0, head[y]));
17        head[y] = edge.size() - 1;
18    }
19    bool bfs(int s, int t) {
20        std::fill(mf.begin(), mf.end(), 0);
21        mf[s] = INF;
22        std::queue<int> q;
23        q.push(s);
24        while(!q.empty()) {
25            int id = q.front();
26            q.pop();
27            for(int eid = head[id]; ~eid; eid = edge[eid].enxt) {
28                auto &nxt, _, cap = edge[eid];
29                if(mf[nxt] == 0 && cap > 0) {

```

```

30         mf[nxt] = std::min(mf[id], cap);
31         pre[nxt] = eid;
32         if(nxt == t) return true;
33         q.push(nxt);
34     }
35 }
36 }
37 return false;
38 };
39 T flow(int s, int t) {
40     T flow = 0;
41     while(bfs(s, t)) { //找到增广路
42         for(int id = t; id != s; id = edge[pre[id] ^ 1].nxt) {
43             edge[pre[id]].cap -= mf[t];
44             edge[pre[id] ^ 1].cap += mf[t];
45         }
46         flow += mf[t];
47     }
48     return flow;
49 }
50 std::vector<Edge> edge;
51 std::vector<int> head, pre; // pre: id的前驱边
52 std::vector<T> mf; //每S~v的流量上限,
53 const T INF = INT_MAX;
54 };
55
56 int main() {
57     std::ios::sync_with_stdio(false);
58     std::cin.tie(nullptr);
59     int n, m, S, T;
60     std::cin >> n >> m >> S >> T;
61     MaxFolw<i64> mf(n + 1);
62     for(int i = 0; i < m; ++i) {
63         int x, y, cap;
64         std::cin >> x >> y >> cap;
65         mf.addEdge(x, y, cap);
66     }
67     std::cout << mf.flow(S, T) << '\n';
68     return 0;
69 }

```

## 2.6.2 二分图判定

时间复杂度:  $O(|V| + |E|)$

空间复杂度:  $O(|V|)$

模板题: [Luogu P1330](#)

```

2 using i64 = long long;
3
4 int main() {
5     std::ios::sync_with_stdio(false);
6     std::cin.tie(nullptr);
7     int n, m;
8     std::cin >> n >> m;
9     std::vector<std::vector<int>> v(n + 1);
10    for(int i = 1; i <= m; ++i) {
11        int x, y;
12        std::cin >> x >> y;
13        v[x].push_back(y);
14        v[y].push_back(x);
15    }
16    std::vector<int> col(n + 1), vis(n + 1); //染色值1/2, 是否标记
17    auto dfs = [&](auto self, int id, int val) ->bool { //判定是否是二分图
18        col[id] = val;
19        vis[id] = 1;
20        for(auto nxt : v[id]) {
21            if(!col[nxt]) {
22                if(!self(self, nxt, val ^ 3)) {
23                    return false;
24                }
25            } else if(col[nxt] == val) {
26                return false;
27            }
28        }
29        return true;
30    };
31    int ans = 0;
32    for(int i = 1; i <= n; ++i) {
33        if(!vis[i]) {
34            col = std::vector<int>(n + 1);
35            if(!dfs(dfs, i, 1)) {
36                std::cout << "Impossible\n";
37                exit(0);
38            }
39            int A = std::count(col.begin(), col.end(), 1);
40            int B = std::count(col.begin(), col.end(), 2);
41            ans += std::min(A, B);
42        }
43    }
44    std::cout << ans << '\n';
45    return 0;
46 }

```

1 #include <bits/stdc++.h>



### 2.6.3 二分图最大匹配

时间复杂度:  $O(|V_1||V_2|)$

空间复杂度:  $O(|E| + |V_1| + |V_2|)$

模板题: [Luogu P3386](#)

```
1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 struct BipartiteGraph {
5     BipartiteGraph(int n, int m)
6         : n(n), m(m), g(n + 1), vis(m + 1), mch(m + 1) {};
7     void add(int x, int y) {
8         g[x].push_back(y);
9     }
10    bool dfs(int id) {
11        for(auto nxt : g[id]) {
12            if(!vis[nxt]) {
13                vis[nxt] = 1;
14                if(!mch[nxt] || dfs(mch[nxt])) {
15                    mch[nxt] = id;
16                    return true;
17                }
18            }
19        }
20        return false;
21    }
22    int solve() { //求最大匹配
23        int res = 0;
24        for(int i = 1; i <= n; ++i) {
25            std::fill(vis.begin(), vis.end(), false);
26            res += dfs(i);
27        }
28        return res;
29    }
30    int n, m;
31    std::vector<std::vector<int>> g; //存图
32    std::vector<bool> vis; //标记是否搜索过
33    std::vector<int> mch; //mch[i]表示i号点匹配的编号
34 };
35
36 int main() {
37     std::ios::sync_with_stdio(false);
38     std::cin.tie(nullptr);
39     int n, m, k;
40     std::cin >> n >> m >> k;
41     BipartiteGraph bg(n + 1, m + 1);
42     for(int i = 1; i <= k; ++i) {
43         int x, y;
```

```
44         std::cin >> x >> y;
45         bg.add(x, y);
46     }
47     std::cout << bg.solve() << '\n';
48     return 0;
49 }
```

## 3 字符串

### 3.1 KMP

```
1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 int main() {
5     std::ios::sync_with_stdio(false);
6     std::cin.tie(nullptr);
7     std::string s, p;
8     std::cin >> s >> p;
9     int n = s.size(), m = p.size();
10    s = '#' + s, p = '#' + p;
11    std::vector<int> kmp(m + 1);
12    for(int i = 2, j = 0; i <= m; ++i) { //求kmp数组
13        while(j > 0 && p[i] != p[j + 1]) {
14            j = kmp[j];
15        }
16        if(p[j + 1] == p[i]) {
17            j++;
18        }
19        kmp[i] = j;
20    }
21    for(int i = 1, j = 0; i <= n; ++i) {
22        while(j > 0 && s[i] != p[j + 1]) {
23            j = kmp[j];
24        }
25        if(s[i] == p[j + 1]) {
26            j++;
27        }
28        if(j == m) {
29            std::cout << i - j + 1 << '\n';
30            j = kmp[j];
31        }
32    }
33    for(int i = 1; i <= m; ++i) {
34        std::cout << kmp[i] << " \n"[i == m];
35    }
36    return 0;
```

### 3.2 EXKMP

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 int main() {
5     std::ios::sync_with_stdio(false);
6     std::cin.tie(nullptr);
7     std::string a, b;
8     std::cin >> a >> b;
9     int n = a.size(), m = b.size();
10    a = '#' + a, b = '#' + b;
11    std::vector<int> z(m + 1), p(n + 1);
12    z[1] = m;
13    for(int i = 2, l = 0, r = 0; i <= m; ++i) {
14        if(i <= r) {
15            z[i] = std::min(z[i - l + 1], r - i + 1);
16        }
17        while(i + z[i] <= m && b[i + z[i]] == b[1 + z[i]]) {
18            z[i]++;
19        }
20        if(i + z[i] - 1 > r) {
21            l = i, r = i + z[i] - 1;
22        }
23    }
24    for(int i = 1, l = 0, r = 0; i <= n; ++i) {
25        if(i <= r) {
26            p[i] = std::min(z[i - l + 1], r - i + 1);
27        }
28        while(1 + p[i] <= m && i + p[i] <= n && b[1 + p[i]] == a[i + p[i]]) {
29            p[i]++;
30        }
31        if(i + p[i] - 1 > r) {
32            l = i, r = i + p[i] - 1;
33        }
34    }
35    i64 ans1 = 0, ans2 = 0;
36    for(int i = 1; i <= m; ++i) {
37        ans1 ^= 1LL * i * (z[i] + 1);
38    }
39    for(int i = 1; i <= n; ++i) {
40        ans2 ^= 1LL * i * (p[i] + 1);
41    }
42    std::cout << ans1 << '\n' << ans2 << '\n';
43    return 0;
44 }

```

### 3.3 字符串哈希

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 struct H64 {
5     H64() = default;
6     H64(i64 v) : v(v) {}
7     H64 operator+(const H64 &o) const {
8         return (v + o.v >= MOD ? v + o.v - MOD : v + o.v);
9     }
10    H64 operator-(const H64 &o) const {
11        return (o.v > v ? v + MOD - o.v : v - o.v);
12    }
13    H64 operator*(const H64 &o) const {
14        __int128 r = (__int128)v * o.v;
15        return H64((r >> 61) + (r & MOD));
16    }
17    auto operator<=>(const H64& o) const = default;
18    i64 v = 0;
19    constexpr static i64 MOD = (1LL << 61) - 1;
20 };
21
22 struct Hash {
23     Hash() = default;
24     Hash(const Hash &hs) : h(hs.h), n(1) {}
25     Hash(const int &val) : h(H64(val)), n(1) {}
26     Hash(const std::string &s) : n(s.size()) {
27         for(int i = 0; i < (int)s.size(); ++i) {
28             h = h + H64(s[i]) * fac(s.size() - i - 1);
29         }
30     }
31     Hash(const H64 &h, const int &n) : h(h), n(n) {}
32     auto operator<=>(const Hash& o) const = default;
33     Hash operator+(const Hash &o) {
34         return {h * fac(o.n) + o.h, n + o.n};
35     }
36     Hash operator-(const Hash &o) {
37         return {h - o.h * fac(n - o.n), n - o.n};
38     }
39     Hash operator+=(const Hash &o) {
40         return *this = (*this + o);
41     }
42     Hash operator-=(const Hash &o) {
43         return *this = (*this - o);
44     }

```

```

45 static H64 fac(int x) {
46     if(x >= f.size()) {
47         int l = f.size();
48         f.resize(x + 1);
49         for(int i = l; i < f.size(); ++i) {
50             f[i] = f[i - 1] * BASE;
51         }
52     }
53     return f[x];
54 }
55 int n{};
56 H64 h{};
57 constexpr static i64 BASE = 114514;
58 inline static std::vector<H64> f{H64(1)};
59 };
60
61 int main() {
62     std::ios::sync_with_stdio(false);
63     std::cin.tie(nullptr);
64     int n;
65     std::cin >> n;
66     std::set<Hash> st;
67     for(int i = 0; i < n; ++i) {
68         std::string s;
69         std::cin >> s;
70         st.insert(Hash(s));
71     }
72     std::cout << st.size() << '\n';
73     return 0;
74 }

```

```

16 for(int i = 0, mid = 0; i < n; ++i) { // mid为回文中心
17     if(i <= mid + v[mid]) {
18         v[i] = std::min(v[2 * mid - i], mid + v[mid] - i); // (t + i) / 2 = mid
19         <=> t = 2 * mid - i;
20     }
21     while(t[i - v[i] - 1] == t[i + v[i] + 1] && 0 <= i - v[i] - 1 && i + v[i] +
22         1 < n) {
23         ++v[i];
24     }
25     if(i + v[i] > mid + v[mid]) {
26         mid = i;
27     }
28     return v;
29 }
30 int main() {
31     std::ios::sync_with_stdio(false);
32     std::cin.tie(nullptr);
33     std::string s;
34     std::cin >> s;
35     std::vector<int> v = manacher(s);
36     int ans = 0;
37     for(int i = 0; i < v.size(); ++i) {
38         ans = std::max(ans, v[i]); // 求最长回文子串
39         std::cout << v[i] << " \n"[i == v.size() - 1];
40     }
41     std::cout << ans << '\n';
42     return 0;
43 }

```

### 3.4 马拉车

```

1 #include <bits/stdc++.h>
2
3 //马拉车(manacher)
4 //https://www.luogu.com.cn/problem/P3805
5
6 // 以第i个数为轴的最大回文 v[2 * i + 1]
7 // 以第i个数和i+1个数中间为轴的最大回文 v[2 * i + 2]
8 // 以[L, R] 区间中轴的最大回文为v[L + R + 1]
9 std::vector<int> manacher(const std::string& s) {
10     int n = 2 * s.length() + 1;
11     std::string t(n, '#'); // 处理字符串
12     for(int i = 0; i < s.length(); ++i) {
13         t[2 * i + 1] = s[i];
14     }
15     std::vector<int> v(n); // 记录回文半径 [l, r] <=> [mid - v[mid], mid + v[mid]]

```

## 4 数学

### 4.1 组合数学

#### 4.1.1 组合数

```

1 constexpr i64 P = 998244353;
2 constexpr i64 MAXN = 3000;
3
4 struct Comb {
5     Comb(int n) : _fac(n + 1, 1), _inv(n + 1, 1), _invfac(n + 1, 1) {
6         for(int i = 2; i <= n; ++i) {
7             _fac[i] = _fac[i - 1] * i % P;
8             _inv[i] = (P - P / i) * _inv[P % i] % P;
9             _invfac[i] = _inv[i] * _invfac[i - 1] % P;
10        }

```

```

11 }
12 i64 fac(int x) { return _fac[x]; }
13 i64 invfac(int x) { return _invfac[x]; }
14 i64 inv(int x) { return _inv[x]; }
15 i64 C(int n, int m) {
16     if(n < m || n < 0 || m < 0) return 0;
17     return _fac[n] * _invfac[m] % P * _invfac[n - m] % P;
18 }
19 i64 A(int n, int m) {
20     if(n < m || n < 0 || m < 0) return 0;
21     return _fac[n] * _invfac[n - m] % P;
22 }
23 std::vector<i64> _fac, _inv, _invfac; //阶乘, i的逆元, 阶乘的逆元
24 } comb(MAXN);

```

#### 4.1.2 卢卡斯定理

$$C_n^m \pmod p = C_{\lfloor \frac{n}{p} \rfloor}^{\lfloor \frac{m}{p} \rfloor} * C_{n \bmod p}^{m \bmod p}$$

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 i64 qpow(i64 a, i64 b, i64 p) {
5     i64 res = 1;
6     while(b) {
7         if(b & 1) {
8             res = res * a % p;
9         }
10        a = a * a % p;
11        b >>= 1;
12    }
13    return res;
14 }
15
16 void solve() {
17     int n, m, p;
18     std::cin >> n >> m >> p;
19     std::vector<i64> fac(p + 1, 1);
20     for(int i = 2; i <= p; ++i) {
21         fac[i] = fac[i - 1] * i % p;
22     }
23     auto comb = [&fac, &p](i64 n, i64 m) ->i64 {
24         return fac[n] * qpow(fac[m], p - 2, p) % p * qpow(fac[n - m], p - 2, p) % p;
25     };
26     auto lucas = [&fac, &p, &comb](auto self, i64 n, i64 m) ->i64 {
27         if(m == 0) return 1;
28         return self(self, n / p, m / p) * comb(n % p, m % p) % p;
29     };

```

```

30     std::cout << lucas(lucas, n + m, m) << '\n';
31 }
32
33 //lucas定理, 求大数组组合数
34 //https://www.luogu.com.cn/problem/P3807
35 int main() {
36     std::ios::sync_with_stdio(false);
37     std::cin.tie(nullptr);
38     int T = 1;
39     std::cin >> T;
40     while(T--) {
41         solve();
42     }
43     return 0;
44 }

```

## 4.2 数论

### 4.2.1 线性筛

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3 constexpr int MAXN = 1E8;
4
5 std::vector<int> minp, prime;
6 std::vector<int> phi, mu;
7
8 void sieve(int n = MAXN) {
9     minp = phi = mu = std::vector<int>(n + 1, 0);
10    phi[1] = mu[1] = 1;
11    for(int i = 2; i <= n; ++i) {
12        if(minp[i] == 0) {
13            minp[i] = i;
14            phi[i] = i - 1;
15            mu[i] = -1;
16            prime.push_back(i);
17        }
18        for(int j = 0; i * prime[j] <= n; ++j) {
19            minp[i * prime[j]] = prime[j];
20            if(minp[i] == prime[j]) {
21                phi[i * prime[j]] = phi[i] * prime[j];
22                break;
23            }
24            phi[i * prime[j]] = phi[i] * (prime[j] - 1);
25            mu[i * prime[j]] = -mu[i];
26        }
27    }
28 }

```

```

29
30 bool isPrime(int n) {
31     return minp[n] == n;
32 }
33
34 int main() {
35     std::ios::sync_with_stdio(false);
36     std::cin.tie(nullptr);
37     int n, q;
38     std::cin >> n >> q;
39     sieve(n);
40     while(q--) {
41         int idx;
42         std::cin >> idx;
43         std::cout << prime[idx - 1] << '\n';
44     }
45     return 0;
46 }

```

#### 4.2.2 区间筛

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 constexpr int MAXN = 2E5;
5 std::vector<int> prime;
6 std::vector<bool> nonPrime(MAXN + 1);
7 void findPrime(int n) {
8     nonPrime[0] = nonPrime[1] = 1;
9     for(int i = 2; i <= n; ++i) {
10         if(nonPrime[i] == false) {
11             prime.push_back(i);
12         }
13         for(int j = 0; i * prime[j] <= n; ++j) {
14             nonPrime[i * prime[j]] = true;
15             if(i % prime[j] == 0) break;
16         }
17     }
18 }
19
20 //区间筛，筛区间[L, R]的质数
21 //https://www.luogu.com.cn/problem/UVA10140
22 int main() {
23     i64 L, R;
24     findPrime(MAXN);
25     while(std::cin >> L >> R) {
26
27         std::vector<i64> res;

```

```

28         std::vector<bool> nonp(R - L + 1);
29         for(auto x : prime) {
30             if(x > R) break;
31             for(int j = std::max((L + x - 1) / x, 2LL); 1LL * j * x <= R; ++j) {
32                 nonp[j * x - L] = 1;
33             }
34         }
35         for(int i = 0; i <= R - L; ++i) {
36             if(nonp[i] == 0 && i + L >= 2) {
37                 res.push_back(i + L);
38             }
39         }
40
41         i64 mn = INT_MAX, mx = INT_MIN;
42         int mnidx = -1, mxidx = -1;
43         for(int i = 1; i < res.size(); ++i) {
44             if(res[i] - res[i - 1] < mn) {
45                 mn = res[i] - res[i - 1];
46                 mnidx = i;
47             }
48             if(res[i] - res[i - 1] > mx) {
49                 mx = res[i] - res[i - 1];
50                 mxidx = i;
51             }
52         }
53         if(res.size() <= 1) {
54             std::cout << "There are no adjacent primes.\n";
55         } else {
56             std::cout << res[mnidx - 1] << ',' << res[mnidx] << " are closest, "
57                 << res[mxidx - 1] << ',' << res[mxidx] << " are most distant.\n";
58         }
59     }
60     return 0;
61 }

```

#### 4.2.3 MillerRabin

模板题: [Luogu SP288](https://www.luogu.com.cn/problem/SP288)

时间复杂度:  $O(k \log^3(n)), k = 7$

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 i64 qpow(i64 a, i64 b, i64 p) {
5     i64 res = 1;
6     while(b) {
7         if(b & 1) {

```

```

8         res = (__int128)res * a % p;
9     }
10    a = (__int128)a * a % p;
11    b >>= 1;
12 }
13 return res;
14 }
15
16 bool Minller(i64 n) {
17     if(n <= 1 || n % 2 == 0) return (n == 2);
18     i64 u = n - 1, k = 0;
19     while(u % 2 == 0) u /= 2, ++k;
20     static std::vector<i64> base = {2, 325, 9375, 28178, 450775, 9780504,
1795265022};
21     for(auto x : base) {
22         i64 res = qpow(x, u, n);
23         if(res == 0 || res == 1 || res == n - 1) continue;
24         for(int i = 1; i <= k; ++i) {
25             res = (__int128)res * res % n;
26             if(res == n - 1) break;
27             if(i == k) return false;
28         }
29     }
30     return true;
31 }
32
33 void solve() {
34     i64 x;
35     std::cin >> x;
36     std::cout << (Minller(x) ? "YES" : "NO") << '\n';
37 }
38
39 int main() {
40     std::ios::sync_with_stdio(false);
41     std::cin.tie(nullptr);
42     int T = 1;
43     std::cin >> T;
44     while(T--) {
45         solve();
46     }
47     return 0;
48 }

```

#### 4.2.4 PollardRho

模板题: Factorize

时间复杂度:  $O(n^{\frac{1}{4}})$

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 i64 qpow(i64 a, i64 b, i64 p) {
5     i64 res = 1;
6     while(b) {
7         if(b & 1) {
8             res = (__int128)res * a % p;
9         }
10        a = (__int128)a * a % p;
11        b >>= 1;
12    }
13    return res;
14 }
15
16 //Miller_rabin判断质数
17 bool Miller(i64 n) {
18     if(n <= 1 || n % 2 == 0) return (n == 2);
19     i64 u = n - 1, k = 0;
20     while(u % 2 == 0) u /= 2, ++k;
21     static std::vector<i64> base = {2, 325, 9375, 28178, 450775, 9780504,
1795265022};
22     for(auto x : base) {
23         i64 res = qpow(x, u, n);
24         if(res == 0 || res == 1 || res == n - 1) continue;
25         for(int i = 1; i <= k; ++i) {
26             res = (__int128)res * res % n;
27             if(res == n - 1) break;
28             if(i == k) return false;
29         }
30     }
31     return true;
32 }
33
34 //Pollard_rho找因子
35 i64 Pollard_rho(i64 n) {
36     assert(n >= 2);
37     if (!(n & 1)) return 2;
38     auto f = [&](i64 x) {
39         return (__int128)x * x % n + 5;
40     };
41     i64 x = 0, y = 0, prod = 1;
42     for (int i = 30, z = 0; i % 64 || std::gcd(prod, n) == 1; ++i) {
43         if (x == y) {
44             x = ++z;
45             y = f(x);
46         }
47         i64 q = (__int128)prod * (x + n - y) % n;

```

```

48     if (q) prod = q;
49     x = f(x), y = f(f(y));
50 }
51 return std::gcd(prod, n);
52 }
53
54 std::vector<i64> factorize(i64 x) {
55     std::vector<i64> res;
56     auto f = [&](auto f, i64 x) ->void {
57         if (x == 1) return;
58         if (Miller(x)) {
59             res.push_back(x);
60             return;
61         }
62         i64 y = Pollard_rho(x);
63         f(f, y), f(f, x / y);
64     };
65     f(f, x);
66     std::ranges::sort(res);
67     return res;
68 }
69
70 void solve() {
71     i64 x;
72     std::cin >> x;
73     auto res = factorize(x);
74     std::cout << res.size() << " \n"[res.empty()];
75     for(int i = 0; i < res.size(); ++i) {
76         std::cout << res[i] << " \n"[i + 1 == res.size()];
77     }
78 }
79
80 int main() {
81     std::ios::sync_with_stdio(false);
82     std::cin.tie(nullptr);
83     int T = 1;
84     std::cin >> T;
85     while(T--) {
86         solve();
87     }
88     return 0;
89 }

```

#### 4.2.5 中国剩余定理

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3

```

```

4 std::array<i64, 3> exgcd(i64 a, i64 b) {
5     if(b == 0) {
6         return {a, 1, 0};
7     }
8     auto [gd, x, y] = exgcd(b, a % b);
9     return {gd, y, x - a / b * y};
10 }
11
12 i64 CRT(std::vector<std::pair<i64, i64>> &v) {
13     i64 M = 1;
14     for(const auto &[m, r] : v) {
15         M *= m;
16     }
17     i64 ans = 0;
18     for(const auto &[m, r] : v) {
19         i64 Mi = M / m;
20         auto [gd, x, y] = exgcd(Mi, m);
21         ans += (__int128)r * Mi % M * x % M;
22     }
23     return (ans % M + M) % M;
24 }
25
26 int main() {
27     std::ios::sync_with_stdio(false);
28     std::cin.tie(nullptr);
29     int n;
30     std::cin >> n;
31     std::vector<std::pair<i64, i64>> v(n);
32     for(auto &[m, r] : v) {
33         std::cin >> m >> r;
34     }
35     std::cout << CRT(v) << '\n';
36     return 0;
37 }

```

#### 4.2.6 扩展中国剩余定理

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 std::array<i64, 3> exgcd(i64 a, i64 b) {
5     if(b == 0) {
6         return {a, 1, 0};
7     }
8     auto [gd, x, y] = exgcd(b, a % b);
9     return {gd, y, x - a / b * y};
10 }
11

```

```

12 i64 EXCRT(std::vector<std::pair<i64, i64>> &v) {
13     i64 M = v[0].first, ans = v[0].second;
14     for(int i = 1; i < v.size(); ++i) {
15         auto [m, r] = v[i];
16         auto [gd, x, y] = exgcd(M, m);
17         i64 dx = ((ans - r) % m + m) % m;
18         if(dx % gd != 0) return -1;
19         i64 k = m / gd;
20         x = __int128(dx / gd) % k * x % k;
21         ans -= x * M;
22         M *= k;
23         ans = (ans % M + M) % M;
24     }
25     return ans;
26 }
27
28 int main() {
29     std::ios::sync_with_stdio(false);
30     std::cin.tie(nullptr);
31     int n;
32     std::cin >> n;
33     std::vector<std::pair<i64, i64>> v(n);
34     for(auto &[m, r] : v) {
35         std::cin >> m >> r;
36     }
37     std::cout << EXCRT(v) << '\n';
38     return 0;
39 }

```

## 4.3 线性代数

### 4.3.1 矩阵

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 template<typename T>
5 struct Matrix {
6     Matrix() : n(0), m(0) {};
7     Matrix(int _n, int _m) : n(_n), m(_m), mt(n, std::vector<T>(m)){}
8     Matrix(const std::vector<std::vector<T>> &v) : Matrix(v.size(), v[0].size()) {
9         for(int i = 0; i < n; ++i) {
10             assert(v[i].size() == m);
11             for(int j = 0; j < m; ++j) {
12                 mt[i][j] = v[i][j];
13             }
14         }
15     }

```

```

16     Matrix<T> operator*(const Matrix<T> &o) {
17         assert(m == o.n);
18         Matrix<T> res(n, o.m);
19         for(int i = 0; i < n; ++i) {
20             for(int j = 0; j < o.m; ++j) {
21                 for(int k = 0; k < m; ++k) {
22                     res.mt[i][j] = res.mt[i][k] * o.mt[k][j];
23                 }
24             }
25         }
26         return res;
27     }
28     Matrix<T> operator*=(const Matrix<T> &o) {
29         return *this = *this * o;
30     }
31     Matrix<T> operator+(const Matrix<T> &o) {
32         assert(n == o.n && m == o.m);
33         Matrix<T> res(n, m);
34         for(int i = 0; i < n; ++i) {
35             for(int j = 0; j < m; ++j) {
36                 res.mt[i][j] = mt[i][j] + o.mt[i][j];
37             }
38         }
39         return res;
40     }
41     Matrix<T> operator-(const Matrix<T> &o) {
42         assert(n == o.n && m == o.m);
43         Matrix<T> res(n, m);
44         for(int i = 0; i < n; ++i) {
45             for(int j = 0; j < m; ++j) {
46                 res.mt[i][j] = mt[i][j] - o.mt[i][j];
47             }
48         }
49         return res;
50     }
51     static Matrix<T> eye(int n) {
52         Matrix<T> res(n, n);
53         for(int i = 0; i < n; ++i) {
54             res.mt[i][i] = 1;
55         }
56         return res;
57     }
58     static Matrix<T> qpow(Matrix<T> a, i64 b) {
59         Matrix<T> res(Matrix::eye(a.n));
60         while(b != 0) {
61             if(b & 1) {
62                 res = res * a;
63             }

```



```

64     a = a * a;
65     b >>= 1;
66 }
67 return res;
68 }
69 friend std::ostream& operator<<(std::ostream& os, const Matrix<T>& o) {
70     for(int i = 0; i < o.n; ++i) {
71         for(int j = 0; j < o.m; ++j) {
72             os << o.mt[i][j] << " \n"[j + 1 == o.m];
73         }
74     }
75     return os;
76 }
77 int n, m;
78 std::vector<std::vector<T>> mt;
79 };
80
81 int main() {
82     std::ios::sync_with_stdio(false);
83     std::cin.tie(nullptr);
84     Matrix<int> res({{1, 2}, {2, 3}});
85     Matrix<int> b(res);
86     std::cout << Matrix<int>::qpow(res, 3);
87     return 0;
88 }

```

#### 4.3.2 线性基

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 template<typename T, int N = std::numeric_limits<T>::digits>
5 struct Basis {
6     bool insert(T x) {
7         for(int i = N - 1; i >= 0; --i) {
8             if(x >> i & 1) {
9                 if(!base[i]) {
10                     base[i] = x;
11                     _count++;
12                     return true;
13                 }
14                 x ^= base[i];
15             }
16         }
17         flag = true;
18         return false;
19     }
20     T kth(T k) {

```

```

21         k -= flag;
22         T res{};
23         int len = _count - 1;
24         for (int i = N - 1; i >= 0; i--) {
25             if(!base[i]) continue;
26             if ((k >> len & 1) ^ (res >> i & 1)) {
27                 res ^= base[i];
28             }
29             len--;
30         }
31         return res;
32     }
33     T querymax() {
34         T res{};
35         for(int i = N - 1; i >= 0; --i) {
36             res = std::max(res ^ base[i], res);
37         }
38         return res;
39     }
40     int count() {
41         return _count;
42     }
43     T size() {
44         return ((T)1 << count()) - (T)!flag;
45     }
46     bool flag = false;
47     std::array<T, N> base{};
48     int _count{};
49 };
50
51 void solve() {
52     int n;
53     std::cin >> n;
54     Basis<i64> basis;
55     for(int i = 0; i < n; ++i) {
56         i64 x;
57         std::cin >> x;
58         basis.insert(x);
59     }
60     std::cout << basis.querymax() << '\n';
61 }
62
63 int main() {
64     std::ios::sync_with_stdio(false);
65     std::cin.tie(nullptr);
66     int T = 1;
67     // std::cin >> T;
68     while(T--) {

```

```

69     solve();
70 }
71     return 0;
72 }

```

## 4.4 多项式

### 4.4.1 FFT

模板题: [Luogu P3803](#) 模板题: [ABC 392G](#)

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 constexpr double PI = std::numbers::pi_v<double>;
5
6 void FFT(std::vector<std::complex<double>> &A, int opt = 1) {
7     int n = A.size();
8     std::vector<int> p(n);
9     for(int i = 0; i < n; ++i) {
10         p[i] = p[i / 2] / 2 + (n / 2) * (i & 1);
11     }
12     for(int i = 0; i < n; ++i) {
13         if(i < p[i]) {
14             std::swap(A[i], A[p[i]]);
15         }
16     }
17     for(int len = 2; len <= n; len <= 1) {
18         std::complex<double> w1 = {cos(2 * PI / len), sin(2 * PI / len) * opt};
19         for(int i = 0; i < n; i += len) {
20             std::complex<double> wk = {1, 0};
21             for(int j = 0; j < len / 2; ++j) {
22                 std::complex<double> x = A[i + j];
23                 std::complex<double> y = A[i + j + len / 2] * wk;
24                 A[i + j] = x + y;
25                 A[i + j + len / 2] = x - y;
26                 wk *= w1;
27             }
28         }
29     }
30 }
31
32 template<typename T>
33 std::vector<T> multiply(const std::vector<T> &A, const std::vector<T> &B) {
34     int n = std::bit_ceil(A.size() + B.size() - 1);
35     std::vector<std::complex<double>> v(n);
36     for(int i = 0; i < A.size(); ++i) {
37         v[i].real(A[i]);

```

```

38     }
39     for(int i = 0; i < B.size(); ++i) {
40         v[i].imag(B[i]);
41     }
42     v.resize(n);
43     FFT(v);
44     for(int i = 0; i < n; ++i) {
45         v[i] *= v[i];
46     }
47     FFT(v, -1);
48     std::vector<T> res(A.size() + B.size() - 1);
49     for(int i = 0; i < res.size(); ++i) {
50         res[i] = (T)round(v[i].imag() / 2 / n);
51     }
52     return res;
53 }
54
55 template<typename T>
56 std::vector<T> convolution(const std::vector<T> &A, std::vector<T> kernel) {
57     std::reverse(kernel.begin(), kernel.end());
58     auto res = multiply(A, kernel);
59     return std::vector(res.begin() + kernel.size() - 1, res.begin() + A.size());
60 }
61
62 int main() {
63     std::ios::sync_with_stdio(false);
64     std::cin.tie(nullptr);
65     int n, m;
66     std::cin >> n >> m;
67     std::vector<int> a(n + 1), b(m + 1);
68     for(int i = 0; i <= n; ++i) {
69         std::cin >> a[i];
70     }
71     for(int i = 0; i <= m; ++i) {
72         std::cin >> b[i];
73     }
74     auto c = multiply(a, b);
75     for(int i = 0; i < c.size(); ++i) {
76         std::cout << c[i] << " \n"[i + 1 == c.size()];
77     }
78     return 0;
79 }

```

### 4.4.2 NTT

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3

```

```

4 constexpr i64 P = 998244353;
5 constexpr i64 G = 3;
6 constexpr i64 INVG = 332748118;
7
8 i64 qpow(i64 a, i64 b) {
9     i64 res = 1;
10    while(b) {
11        if(b & 1) {
12            res = res * a % P;
13        }
14        b >>= 1;
15        a = a * a % P;
16    }
17    return res;
18 }
19
20 void NTT(std::vector<i64> &A, int opt = 1) {
21     int n = A.size();
22     std::vector<int> p(n);
23     for(int i = 0; i < n; ++i) {
24         p[i] = p[i / 2] / 2 + (n / 2) * (i & 1);
25     }
26     for(int i = 0; i < n; ++i) {
27         if(i < p[i]) {
28             std::swap(A[i], A[p[i]]);
29         }
30     }
31     for(int len = 2; len <= n; len <= 1) {
32         i64 g1 = qpow(opt == 1 ? G : INVG, (P - 1) / len);
33         for(int i = 0; i < n; i += len) {
34             i64 gk = 1;
35             for(int j = 0; j < len / 2; ++j) {
36                 i64 x = A[i + j];
37                 i64 y = A[i + j + len / 2] * gk % P;
38                 A[i + j] = (x + y) % P;
39                 A[i + j + len / 2] = (x - y + P) % P;
40                 gk = (gk * g1) % P;
41             }
42         }
43     }
44 }
45
46 std::vector<i64> multiply(std::vector<i64> A, std::vector<i64> B) {
47     auto k = A.size() + B.size() - 1;
48     int n = std::bit_ceil(k);
49     A.resize(n), B.resize(n);
50     NTT(A), NTT(B);
51     for (int i = 0; i < n; ++i) {

```

```

52         A[i] = A[i] * B[i] % P;
53     }
54     NTT(A, -1);
55     i64 invn = qpow(n, P - 2);
56     A.resize(k);
57     for (auto &x : A) {
58         x = x * invn % P;
59     }
60     return A;
61 }
62
63 int main() {
64     std::ios::sync_with_stdio(false);
65     std::cin.tie(nullptr);
66     int n, m;
67     std::cin >> n >> m;
68     std::vector<i64> a(n + 1), b(m + 1);
69     for(int i = 0; i <= n; ++i) {
70         std::cin >> a[i];
71     }
72     for(int i = 0; i <= m; ++i) {
73         std::cin >> b[i];
74     }
75     auto c = multiply(a, b);
76     for(int i = 0; i < c.size(); ++i) {
77         std::cout << c[i] << " \n"[i + 1 == c.size()];
78     }
79     return 0;
80 }

```

#### 4.4.3 Poly

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 constexpr i64 P = 998244353;
5 constexpr i64 G = 3;
6 constexpr i64 INVG = 332748118;
7
8 i64 qpow(i64 a, i64 b) {
9     i64 res = 1;
10    while(b) {
11        if(b & 1) {
12            res = res * a % P;
13        }
14        b >>= 1;
15        a = a * a % P;
16    }

```

```

17     return res;
18 }
19
20 struct Poly : std::vector<i64> {
21     using std::vector<i64>::vector;
22     friend Poly& operator+=(Poly &A, const Poly &B) {
23         A.resize(std::max(A.size(), B.size()));
24         for(int i = 0; i < B.size(); ++i) {
25             A[i] = (A[i] + B[i]) % P;
26         }
27         return A;
28     }
29     friend Poly& operator-=(Poly &A, const Poly &B) {
30         A.resize(std::max(A.size(), B.size()));
31         for(int i = 0; i < B.size(); ++i) {
32             A[i] = (A[i] - B[i] + P) % P;
33         }
34         return A;
35     }
36     friend Poly& operator*=(Poly &A, Poly B) {
37         auto k = A.size() + B.size() - 1;
38         auto n = std::bit_ceil(k);
39         if(1LL * A.size() * B.size() <= 3 * n * std::bit_width(n)) {
40             Poly res(k);
41             for(int i = 0; i < A.size(); ++i) {
42                 for (int j = 0; j < B.size(); ++j) {
43                     res[i + j] = (res[i + j] + A[i] * B[j]) % P;
44                 }
45             }
46             A = std::move(res);
47         } else {
48             A.resize(n), B.resize(n);
49             NTT(A), NTT(B);
50             for (int i = 0; i < n; ++i) {
51                 A[i] = A[i] * B[i] % P;
52             }
53             NTT(A, -1);
54             i64 invn = qpow(n, P - 2);
55             A.resize(k);
56             for (auto &x : A) {
57                 x = x * invn % P;
58             }
59         }
60         return A;
61     }
62     friend Poly& operator/=(Poly &A, Poly B) {
63         if(A.size() < B.size()) return A = Poly{0};
64         int n = A.size() - B.size() + 1;

```

```

65         std::reverse(A.begin(), A.end());
66         std::reverse(B.begin(), B.end());
67         A = A % n * inv(B, n) % n;
68         std::reverse(A.begin(), A.end());
69         return A;
70     }
71     friend Poly& operator%=(Poly &A, const Poly &B) {
72         A = (A - A / B * B) % (B.size() - 1);
73         return A;
74     }
75     friend Poly& operator%=(Poly &A, size_t k) {
76         A.resize(std::min(A.size(), k));
77         return A;
78     }
79     friend Poly operator+(Poly A, const Poly &B) { return A += B; }
80     friend Poly operator-(Poly A, const Poly &B) { return A -= B; }
81     friend Poly operator*(Poly A, const Poly &B) { return A *= B; }
82     friend Poly operator/(Poly A, const Poly &B) { return A /= B; }
83     friend Poly operator%(Poly A, const Poly &B) { return A %= B; }
84     friend Poly operator%(Poly A, size_t k) { return A %= k; }
85
86     static Poly derive(const Poly &A) {
87         if(A.size() <= 1) return Poly{0};
88         Poly res(A.size() - 1);
89         for(int i = 1; i < A.size(); ++i) {
90             res[i - 1] = A[i] * i % P;
91         }
92         return res;
93     }
94     static Poly integral(const Poly &A) {
95         Poly res(A.size() + 1);
96         res[1] = A[0];
97         std::vector<i64> _inv(A.size() + 1, 1);
98         for(int i = 2; i <= A.size(); ++i) {
99             _inv[i] = (P - P / i) * _inv[P % i] % P;
100             res[i] = A[i - 1] * _inv[i] % P;
101         }
102         return res;
103     }
104     static Poly ln(const Poly &A, int n) {
105         return integral(derive(A) * inv(A, n) % n) % n;
106     }
107     static Poly pow(const Poly &A, i64 k, int n) {
108         return exp(Poly{k} * ln(A, n) % n) % n;
109     }
110     static Poly inv(const Poly &A, int n) {
111         Poly res = {qpow(A[0], P - 2)};
112         for(int k = 2; k < 2 * n; k <= 1) {

```

```

113     res = res * (Poly{2} - A % k * res) % k;
114 }
115 return res % n;
116 }
117 static Poly exp(const Poly &A, int n) {
118     Poly res = {1};
119     for(int k = 2; k < 2 * A.size(); k <= 1) {
120         res = res * (Poly{1} - ln(res, k) + A % k) % k;
121     }
122     return res % n;
123 }
124 static Poly sqrt(const Poly &A, int n) {
125     Poly res = {1};
126     for(int k = 2; k < 2 * A.size(); k <= 1) {
127         res = (res + A % k * inv(res, k) % k) * Poly{(P + 1) / 2} % k;
128     }
129     return res % n;
130 }
131 static Poly ln(const Poly &A) { return ln(A, A.size()); }
132 static Poly inv(const Poly &A) { return inv(A, A.size()); }
133 static Poly exp(const Poly &A) { return exp(A, A.size()); }
134 static Poly sqrt(const Poly &A) { return sqrt(A, A.size()); }
135 static Poly pow(const Poly &A, i64 k) { return pow(A, k, A.size()); }
136
137 Poly shift(int k) {
138     auto A = *this;
139     if (k >= 0) {
140         A.insert(A.begin(), k, 0);
141         return A;
142     } else if (A.size() <= -k) {
143         return Poly();
144     } else {
145         return Poly(A.begin() + (-k), A.end());
146     }
147 }
148 static void NTT(std::vector<i64> &A, int opt = 1) {
149     int n = A.size();
150     for(int i = 0, j = 0; i < n; ++i) {
151         if(i > j) {
152             std::swap(A[i], A[j]);
153         }
154         for(int k = n >> 1; (j ^= k) < k; k >>= 1);
155     }
156     init(n);
157     for(int i = 1; i < n; i <= 1) {
158         for(int j = 0; j < n; j += (i << 1)) {
159             for(int k = j; k < i + j; ++k) {
160                 int t = w[k + i - j] * A[i + k] % P;

```

```

161         A[k + i] = (A[k] - t < 0 ? A[k] - t + P : A[k] - t);
162         A[k] = (A[k] + t >= P ? A[k] + t - P : A[k] + t);
163     }
164 }
165 }
166 if(opt == -1) {
167     std::reverse(A.begin() + 1, A.end());
168 }
169 }
170 static void init(int n) {
171     if (n <= w.size()) return;
172     int l = w.size();
173     w.resize(n);
174     i64 wn = qpow(G, (P - 1) / n);
175     w[n >> 1] = 1;
176     for (int i = (n >> 1) + 1; i < n; ++i) {
177         w[i] = w[i - 1] * wn % P;
178     }
179     for (int i = n - 1; i >= l; --i) {
180         w[i >> 1] = w[i];
181     }
182 }
183 inline static std::vector<i64> w{1};
184 };
185
186 int main() {
187     std::ios::sync_with_stdio(false);
188     std::cin.tie(nullptr);
189     Poly A{1, 2, 3};
190
191     return 0;
192 }

```

## 5 计算几何

### 5.1 凸包

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 constexpr long double EPS = 1E-10;
5
6 template<typename T>
7 int sgn(T a){
8     return (a > EPS) - (a < -EPS);
9 }
10

```

```

11 template<typename T>
12 struct Point;
13
14 template<typename T>
15 struct Vector { //向量
16     T x = 0, y = 0;
17     Vector(const Point<T> &p) : x(p.x), y(p.y) {}
18     Vector(T x_ = 0, T y_ = 0) : x(x_), y(y_) {}
19     template<typename U> operator Vector<U>() {return Vector<U>(U(x), U(y));}
20     Vector operator+(const Vector<T> &o) const {return {x + o.x, y + o.y};}
21     Vector operator-(const Vector<T> &o) const {return {x - o.x, y - o.y};}
22     Vector operator*(T f) const {return {x * f, y * f};}
23     Vector operator/(T f) const {return {x / f, y / f};}
24     friend std::istream &operator>>(std::istream &is, Vector &p) {
25         return is >> p.x >> p.y;
26     }
27     friend std::ostream &operator<<(std::ostream &os, Vector p) {
28         return os << "(" << p.x << ", " << p.y << ")";
29     }
30 };
31
32 template<typename T>
33 struct Point { //点
34     T x = 0, y = 0;
35     Point(T x_ = 0, T y_ = 0) : x(x_), y(y_) {}
36     template<typename U> operator Point<U>() {return Point<U>(U(x), U(y));}
37     Point operator+(const Vector<T> &o) const {return {x + o.x, y + o.y};}
38     Vector<T> operator-(const Point<T> &o) const {return {x - o.x, y - o.y};}
39     Point operator-() const {return {-x, -y};}
40     Point operator*(T f) const {return {x * f, y * f};}
41     Point operator/(T f) const {return {x / f, y / f};}
42     friend Point operator*(T f, Point p) {return {p * f};}
43     bool operator==(const Point &o) const {
44         return sign(x - o.x) == 0 && sign(y - o.y) == 0;
45     }
46     constexpr std::strong_ordering operator<=>(const Point &o) const {
47         if(sign(x - o.x) == 0) {
48             return sign(y - o.y) <=> 0;
49         } else {
50             return sign(x - o.x) <=> 0;
51         }
52     }
53     friend std::istream &operator>>(std::istream &is, Point &p) {
54         return is >> p.x >> p.y;
55     }
56     friend std::ostream &operator<<(std::ostream &os, Point p) {
57         return os << "(" << p.x << ", " << p.y << ")";
58     }
59 }

```

```

59 };
60
61 template<typename T>
62 struct Line { //直线
63     Point<T> s, t;
64     Line() = default;
65     Line(Point<T> _s, Point<T> _t) : s(_s), t(_t) {}
66 };
67
68 template<typename T>
69 struct Seg { //线段
70     Point<T> s, t;
71     Seg() = default;
72     Seg(Point<T> _s, Point<T> _t) : s(_s), t(_t) {}
73 };
74
75 template<typename T, typename U>
76 struct Circle { //圆
77     Point<T> o;
78     U r{};
79 };
80
81 template<typename T>
82 T dot(const Vector<T> &a, const Vector<T> &b) { //向量a和向量b的点积
83     return a.x * b.x + a.y * b.y;
84 }
85
86 template<typename T>
87 T dot(const Point<T> &a, const Point<T> &b, const Point<T> &c) { //向量a->b和向量a->
88     //c的点积
89     return dot(b - a, c - a);
90 }
91
92 template<typename T>
93 T cross(const Vector<T> &a, const Vector<T> &b) { //向量a和向量b的叉积
94     return a.x * b.y - a.y * b.x;
95 }
96
97 template<typename T>
98 T cross(const Point<T> &a, const Point<T> &b, const Point<T> &c) { //向量a->b和向量a
99     //->c的叉积
100     return cross(b - a, c - a);
101 }
102
103 template<typename T>
104 T len2(const Vector<T> &a) { //向量a的模长的平方
105     return a.x * a.x + a.y * a.y;
106 }

```

```

105 template<typename T>
106 long double len(const Vector<T> &a) { //向量a的模长
107     return sqrtl(len2(a));
108 }
109
110 template<typename T>
111 Vector<T> standardize(const Vector<T> &a) { //向量a的单位向量
112     return a / len(a);
113 }
114
115 template<typename T>
116 long double angle(Vector<T> a, Vector<T> b) { //求两向量夹角[0, pi]
117     return fabs(atan2l(cross(a, b), dot(a, b)));
118 }
119
120 template<typename T>
121 T dis2(const Point<T> &a, const Point<T> &b) { //点a和点b距离的平方（防精度损失）
122     return (b.x - a.x) * (b.x - a.x) + (b.y - a.y) * (b.y - a.y);
123 }
124
125 template<typename T>
126 long double dis(const Point<T> &a, const Point<T> &b) { //点a到点b距离
127     return sqrtl(dis2(a, b));
128 }
129
130 template<typename T>
131 long double angle(const Vector<T> &a, const Vector<T> &b) { //向量a和向量b的夹角弧度
132     return acosl(dot(a, b) / len(a) / len(b));
133 }
134
135 template<typename T>
136 Vector<T> rotate(const Vector<T> &a) { //向量a逆时针旋转pi/2
137     return {-a.y, a.x};
138 }
139
140 template<typename T>
141 Vector<T> rotate(const Vector<T> &a, long double rad) { //向量a逆时针旋转rad弧度
142     return {a.x * cosl(rad) - a.y * sinl(rad), a.x * sinl(rad) + a.y * cosl(rad)};
143 }
144
145 template<typename T>
146 Point<T> rotate(const Point<T> &a, const Point<T> &b, long double rad) { //点b绕点a
    //逆时针旋转rad弧度
147     return {
148         (b.x - a.x) * cosl(rad) - (b.y - a.y) * sinl(rad) + a.x,
149         (b.x - a.x) * sinl(rad) + (b.y - a.y) * cosl(rad) + a.y
150     };
151 }

```

```

152 }
153
154 template<typename T>
155 bool intersect(const Seg<T> &a, const Seg<T> &b) { //线段a和线段b不严格相交，可以包
    //含端点相交
156     return sgn(cross(a.s, a.t, b.s) * cross(a.s, a.t, b.t)) <= 0
157         && sgn(cross(b.s, b.t, a.s) * cross(b.s, b.t, a.t)) <= 0;
158 }
159
160 template<typename T>
161 bool intersectStrictly(const Seg<T> &a, const Seg<T> &b) { //线段a和线段b严格相交，
    //不包含端点相交
162     return sgn(cross(a.s, a.t, b.s) * cross(a.s, a.t, b.t)) < 0
163         && sgn(cross(b.s, b.t, a.s) * cross(b.s, b.t, a.t)) < 0;
164 }
165
166 template<typename T>
167 Point<T> getNode(const Line<T> &a, const Line<T> &b) { //求直线a与直线b的交点（需要
    //先判断是否相交）
168     T dx = cross(a.s, b.s, b.t) / cross(a.t - a.s, b.t - b.s);
169     return a.s + (a.t - a.s) * dx;
170 }
171
172 template<typename T>
173 Point<T> getNode(const Seg<T> &a, const Seg<T> &b) { //求线段a与线段b的交点（需要先
    //判断是否相交）
174     T dx = cross(a.s, b.s, b.t) / cross(a.t - a.s, b.t - b.s);
175     return a.s + (a.t - a.s) * dx;
176 }
177
178 template<typename T>
179 Line<T> midseg(const Seg<T> &seg) { //求线段的垂直平分线
180     Point mid = (seg.s + seg.t) / 2;
181     return {mid, mid + rotate(seg.t - seg.s)};
182 }
183
184 template<typename T>
185 Circle<T, long double> getCircle(const Point<T> &p1, const Point<T> &p2, const Point
    <T> &p3) { //求三点所在圆
186     Line<T> l1 = midseg(Seg<T>{p1, p2});
187     Line<T> l2 = midseg(Seg<T>{p1, p3});
188     Point<T> o = getNode(l1, l2);
189     return {o, dis(o, p1)};
190 }
191
192 template<typename T>
193 Circle<T, long double> getCircle(const Point<T> &p1, const Point<T> &p2) { //求以两
    //点为直径所在圆

```

```

194     Point<T> O = (p1 + p2) / 2;
195     return {0, dis(O, p1)};
196 }
197
198 template<typename T>
199 bool onSeg(const Point<T> &p, const Seg<T> &s) { //判断点是否在线段上
200     if(sgn(cross(s.s, s.t, p)) != 0) return false;
201     return std::min(s.s, s.t) <= p && p <= std::max(s.s, s.t);
202 }
203
204 template<typename T, typename U>
205 int pointOnCircle(const Point<T> &p, const Circle<T, U> &c) { //点和圆的位置关系, -1
    点在圆外, 0点在圆上, 1点在圆外,
206     return sgn(c.r * c.r - dis2(p, c.o));
207 }
208
209 template<typename T>
210 std::vector<Point<T>> andrew(std::vector<Point<T>> v) { //Andrew求凸包
211     std::sort(v.begin(), v.end());
212     std::vector<Point<T>> stk;
213     for(int i = 0; i < v.size(); ++i) {
214         while(stk.size() > 1 && sgn(cross(stk[stk.size() - 2], stk.back(), v[i])) <=
0) {
215             stk.pop_back();
216         }
217         stk.push_back(v[i]);
218     }
219     int t = stk.size();
220     for(int i = (int)v.size() - 2; i >= 0; --i) {
221         while(stk.size() > t && sgn(cross(stk[stk.size() - 2], stk.back(), v[i])) <=
0) {
222             stk.pop_back();
223         }
224         stk.push_back(v[i]);
225     }
226     stk.pop_back();
227     return stk;
228 };
229
230 template<typename T>
231 std::vector<Point<T>> graham(std::vector<Point<T>> v) { //Graham求凸包
232     Point<T> base = *min_element(v.begin(), v.end());
233     std::ranges::sort(v, [&](auto p1, auto p2) ->bool {
234         if(sgn(cross(base, p1, p2)) == 0) return p1 < p2;
235         return sgn(cross(base, p1, p2)) == -1;
236     });
237     std::vector<Point<T>> stk;
238     for(int i = 0; i < v.size(); ++i) {

```

```

239         while(stk.size() > 1 && sgn(cross(stk[stk.size() - 2], stk.back(), v[i])) >=
0) {
240             stk.pop_back();
241         }
242         stk.push_back(v[i]);
243     }
244     return stk;
245 }
246
247 template<typename T>
248 std::vector<Line<T>> halfcut(std::vector<Line<T>> lines) { //半平面交, 默认左侧
249     std::sort(lines.begin(), lines.end(), [&](const auto &l1, const auto &l2) ->bool
{
250         auto v1 = l1.t - l1.s, v2 = l2.t - l2.s;
251         auto a1 = atan2(v1.y, v1.x), a2 = atan2(v2.y, v2.x);
252         if(sgn(a1 - a2) != 0) return sgn(a1 - a2) == -1;
253         return sgn(cross(l1.s, l1.t, l2.t)) < 0;
254     });
255     int l = 0, r = -1;
256     std::vector<Line<T>> ls(lines.size());
257     for(const auto &line : lines) {
258         if(r >= l && sgn(cross(line.t - line.s, ls[r].t - ls[r].s)) == 0) continue;
259         while(r > l && sgn(cross(line.s, line.t, intersection(ls[r], ls[r - 1]))) ==
-1) r--;
260         while(r > l && sgn(cross(line.s, line.t, intersection(ls[l], ls[l + 1]))) ==
-1) l++;
261         ls[++r] = line;
262     }
263     while(r > l + 1 && sgn(cross(ls[l].s, ls[l].t, intersection(ls[r], ls[r - 1])))
== -1) r--;
264     return std::vector<Line<T>>(ls.begin() + l, ls.begin() + r + 1);
265 }
266
267 template<typename T>
268 std::vector<Point<T>> linesToPoints(const std::vector<Line<T>> &lines) { //直线式凸
    包转为点凸包
269     std::vector<Point<T>> v;
270     for(int i = 0; i < lines.size(); ++i) {
271         v.push_back(intersection(lines[i], lines[(i + 1) % lines.size()]));
272     }
273     return v;
274 };
275
276 template<typename T>
277 long double area(const std::vector<Point<T>> &v) { //求凸包面积
278     long double ans = 0;
279     for(int i = 1; i + 1 < v.size(); ++i) {
280         ans += cross(v[0], v[i], v[i + 1]);

```



```

281     }
282     ans /= 2;
283     return ans;
284 }
285
286 template<typename T>
287 T diameter2(const std::vector<Point<T>> &v) { //旋转卡壳求凸包直径的平方
288     int n = v.size();
289     T res = 0;
290     for(int i = 0, j = 1; i < n; ++i) {
291         while(sgn(cross(v[i], v[(i + 1) % n], v[j]) - cross(v[i], v[(i + 1) % n], v
292             [(j + 1) % n])) <= 0) {
293             j = (j + 1) % n;
294         }
295         res = std::max({res, dis2(v[i], v[j]), dis2(v[(i + 1) % n], v[j])});
296     }
297     return res;
298 }
299
300 template<typename T>
301 long double diameter(const std::vector<Point<T>> &v) { //旋转卡壳求凸包直径
302     return sqrtl(diameter2(v));
303 }
304
305 template<typename T>
306 long double mindistance(std::vector<Point<T>> v) { //平面最近点对O(nlog(n))
307     std::sort(v.begin(), v.end());
308     long double d = 1E18;
309     auto cmp = [](const Point<T>& a, const Point<T>& b) {
310         return a.y < b.y;
311     };
312     std::multiset<Point<T>, decltype(cmp)> st;
313     for(int i = 0, j = 0; i < v.size(); ++i) {
314         while(j < i && sgn(v[j].x - v[i].x + d) <= 0) {
315             st.erase(v[j]);
316             ++j;
317         }
318         for(auto it = st.lower_bound({v[i].x, v[i].y - T(d)}); it != st.end() && sgn
319             (it->y - v[i].y - T(d + 1)) <= 0; ++it) {
320             d = std::min(d, dis(v[i], *it));
321         }
322         st.insert(v[i]);
323     }
324     return d;
325 }
326
327 template<typename T>
328 long double grith(const std::vector<Point<T>> &v) { //求凸包周长

```

```

327     long double ans = 0;
328     for(int i = 0; i < v.size(); ++i) {
329         ans += dis(v[i], v[(i + 1) % v.size()]);
330     }
331     return ans;
332 }
333
334 void solve() {
335 }
336
337 int main() {
338     std::ios::sync_with_stdio(false);
339     std::cin.tie(nullptr);
340     int T = 1;
341     // std::cin >> T;
342     while(T--) {
343         solve();
344     }
345     return 0;
346 }
347

```

## 6 杂项

### 6.1 动态 bitset

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 struct dynamic_bitset {
5     dynamic_bitset() = default;
6     dynamic_bitset(const uint64_t &n) : v(1, n), _count(std::popcount(n)), _size(64)
7     {}
8     dynamic_bitset(const int sz, bool bit) {
9         resize(sz);
10        if(bit) {
11            set();
12        }
13    }
14    dynamic_bitset(const std::string &s) {
15        resize(s.size());
16        for(int i = 0; i < s.size(); ++i) {
17            if(s[i] == '0') continue;
18            set(i);
19        }
20    }

```

```

21 #define blk(x) (x >> 6)
22 #define idx(x) (x & 63)
23 bool operator==(const dynamic_bitset &o) const {
24     return _size == o._size && v == o.v;
25 }
26 bool operator!=(const dynamic_bitset &o) const {
27     return !(*this == o);
28 }
29 dynamic_bitset operator&=(const dynamic_bitset &o) {
30     combine(o, [](auto &x, auto y) { x &= y; });
31     return *this;
32 }
33 dynamic_bitset operator|=(const dynamic_bitset &o) {
34     combine(o, [](auto &x, auto y) { x |= y; });
35     return *this;
36 }
37 dynamic_bitset operator^=(const dynamic_bitset &o) {
38     combine(o, [](auto &x, auto y) { x ^= y; });
39     return *this;
40 }
41 dynamic_bitset operator&(const dynamic_bitset &o) {
42     dynamic_bitset res = *this;
43     res &= o;
44     return res;
45 }
46 dynamic_bitset operator|(const dynamic_bitset &o) {
47     dynamic_bitset res = *this;
48     res |= o;
49     return res;
50 }
51 dynamic_bitset operator^(const dynamic_bitset &o) {
52     dynamic_bitset res = *this;
53     res ^= o;
54     return res;
55 }
56
57 dynamic_bitset operator~() const {
58     dynamic_bitset res;
59     res.resize(_size);
60     for (size_t i = 0; i < v.size(); ++i) {
61         res.v[i] = ~v[i];
62     }
63     if (!v.empty()) {
64         res.v.back() &= get(0, idx(_size - 1));
65     }
66     res._count = _size - _count;
67     return res;
68 }

```

```

69 dynamic_bitset operator>>(int dx) const {
70     dynamic_bitset res;
71     res.resize(std::max(0, (int)_size - dx));
72     for(int i = 0; i < res.v.size(); ++i) {
73         int l = (i << 6) + dx;
74         int r = std::min((int)_size - 1, l + 63);
75         assert(blk(l) < v.size());
76         res.v[i] = (v[blk(l)] >> idx(l)) | (v[blk(r)] << (63 - idx(r)));
77         res._count += std::popcount(res.v[i]);
78     }
79     res.reset(res._size, (res.v.size() << 6) - 1);
80     return res;
81 }
82 dynamic_bitset operator<<(int dx) const {
83     dynamic_bitset res;
84     res.resize(_size + dx);
85     for(int i = 0; i < v.size(); ++i) {
86         int l = (i << 6) + dx;
87         int r = std::min((int)res._size - 1, l + 63);
88         res.v[blk(l)] |= v[i] << idx(dx);
89         res.v[blk(r)] |= v[i] >> (63 - idx(dx));
90     }
91     res._count = _count;
92     return res;
93 }
94
95 dynamic_bitset sub(int l, int r) const {
96     dynamic_bitset res = *this >> l;
97     res.resize(r - l + 1);
98     return res;
99 }
100 bool operator[](const int &pos) const {
101     return v[blk(pos)] >> idx(pos) & 1ULL;
102 }
103
104 unsigned long to_ulong() const { return v.empty() ? 0 : (unsigned long)v[0]; }
105 unsigned long to_ullong() const { return v.empty() ? 0 : (unsigned long long)v[0]; }
106 std::string to_string() const {
107     std::string res(_size, '0');
108     for(int i = 0; i < _size; ++i) {
109         res[i] = '0' + (*this)[i];
110     }
111     return res;
112 }
113 void set(int l, int r) {
114     update(l, r, [](auto &x, auto mask) { x |= mask; });
115 }

```

```

116 void reset(int l, int r) {
117     update(l, r, [](auto &x, auto mask) { x &= ~mask; });
118 }
119 void flip(int l, int r) {
120     update(l, r, [](auto &x, auto mask) { x ^= mask; });
121 }
122 void set() { set(0, _size - 1); }
123 void reset() { reset(0, _size - 1); }
124 void flip() { flip(0, _size - 1); }
125 void set(int pos) { set(pos, pos); }
126 void reset(int pos) { reset(pos, pos); }
127 void flip(int pos) { flip(pos, pos); }
128
129 void push_back(bool bit) {
130     resize(_size + 1);
131     if(!bit) return;
132     set(_size - 1);
133 }
134 void pop_back() {
135     resize(_size - 1);
136 }
137
138 size_t size() const { return _size; }
139 size_t count() const { return _count; }
140 size_t num_blocks() const { return v.size(); }
141 bool any() const { return _count != 0; }
142 bool all() const { return _count == _size; }
143 bool none() const { return _count == 0; }
144
145 void resize(size_t sz) {
146     if(sz < _size) {
147         reset(sz, _size - 1);
148     }
149     v.resize((sz + 63) / 64);
150     _size = sz;
151 }
152
153 uint64_t get(int l, int r) const {
154     return (~0ULL << l) & (~0ULL >> (63 - r));
155 }
156 void update(int l, int r, auto func) {
157     for(int i = blk(l); i <= blk(r); ++i) {
158         _count -= std::popcount(v[i]);
159         int bl = i == blk(l) ? idx(l) : 0;
160         int br = i == blk(r) ? idx(r) : 63;
161         func(v[i], get(bl, br));
162         _count += std::popcount(v[i]);
163     }

```

```

164 }
165 void combine(const dynamic_bitset &o, auto func) {
166     resize(std::max(_size, o._size));
167     _count = 0;
168     for(int i = 0; i < o.v.size(); ++i) {
169         func(v[i], i < o.v.size() ? o.v[i] : 0ULL);
170         _count += std::popcount(v[i]);
171     }
172 }
173 friend std::ostream &operator<<(std::ostream &os, const dynamic_bitset &db){
174     for(int i = 0; i < db.v.size(); ++i) {
175         if(i + 1 == db.v.size()) {
176             for(int j = (int)db._size - 1; j >= i * 64; --j) {
177                 os << db[j];
178             }
179         } else {
180             os << std::bitset<64>(db.v[i]);
181         }
182     }
183     return os;
184 }
185 std::vector<uint64_t> v;
186 size_t _count = 0;
187 size_t _size = 0;
188
189 #undef blk
190 #undef idx
191 };
192
193 int main() {
194     dynamic_bitset db;
195     return 0;
196 }

```

## 6.2 取整

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 template<typename T>
5 T floorDiv(T a, T b) {
6     return a / b - (((a < 0) ^ (b < 0)) && a % b != 0);
7 }
8
9 template<typename T>
10 T ceilDiv(T a, T b) {
11     return a / b + (!((a < 0) ^ (b < 0)) && a % b != 0);
12 }

```

```

13
14 template<typename T>
15 T roundDiv(T a, T b) {
16     return a / b + (abs(a % b) * 2 >= abs(b) ? ((a < 0) ^ (b < 0) ? -1 : 1) : 0);
17 }
18
19 int main() {
20     std::ios::sync_with_stdio(false);
21     std::cin.tie(nullptr);
22     return 0;
23 }

```

### 6.3 康托展开

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3 constexpr i64 P = 998244353;
4
5 template<typename T>
6 class Fenwick {
7 public:
8     Fenwick(int n) : v(std::vector<T>(n + 1)) {};
9     void update(int x, T dx) {
10         for(int i = x; i < v.size(); i += (i & -i)) {
11             v[i] += dx;
12         }
13     }
14     T query(int x) {
15         T res{};
16         for(int i = x; i > 0; i -= (i & -i)) {
17             res += v[i];
18         }
19         return res;
20     }
21     T range(int l, int r) {
22         return query(r) - query(l - 1);
23     }
24 private:
25     std::vector<T> v;
26 };
27
28 //康托展开(求排列的排名)
29 //https://www.luogu.com.cn/problem/P5367
30 int main() {
31     std::ios::sync_with_stdio(false);
32     std::cin.tie(nullptr);
33     int n;
34     std::cin >> n;

```

```

35     Fenwick<int> tr(n);
36     std::vector<int> p(n + 1);
37     std::vector<i64> fac(n + 1, 1);
38     for(int i = 1; i <= n; ++i) {
39         std::cin >> p[i];
40         tr.update(p[i], 1);
41         fac[i] = fac[i - 1] * i % P;
42     }
43     i64 ans = 1;
44     for(int i = 1; i <= n; ++i) {
45         ans = (ans + fac[n - i] * tr.query(p[i] - 1)) % P;
46         tr.update(p[i], -1);
47     }
48     std::cout << ans << '\n';
49     return 0;
50 }

```

### 6.4 逆康托展开

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 template<typename T>
5 class Fenwick {
6 public:
7     Fenwick(int n) : v(std::vector<T>(n + 1)) {};
8     void update(int x, T dx) {
9         for(int i = x; i < v.size(); i += (i & -i)) {
10             v[i] += dx;
11         }
12     }
13     T query(int x) {
14         T res{};
15         for(int i = x; i > 0; i -= (i & -i)) {
16             res += v[i];
17         }
18         return res;
19     }
20     T range(int l, int r) {
21         return query(r) - query(l - 1);
22     }
23 private:
24     std::vector<T> v;
25 };
26
27 //逆康托展开
28 //https://acm.hdu.edu.cn/showproblem.php?pid=1027
29 int main() {

```

```

30 std::ios::sync_with_stdio(false);
31 std::cin.tie(nullptr);
32 int n, m;
33 while(std::cin >> n >> m) {
34     Fenwick<int> tr(n);
35     std::vector<i64> fac(n + 1, 1);
36     for(int i = 1; i <= n; ++i) {
37         if(fac[i - 1] > m) {
38             fac[i] = fac[i - 1];
39         } else {
40             fac[i] = fac[i - 1] * i;
41         }
42         tr.update(i, 1);
43     }
44     m--;
45     for(int i = 1; i <= n; ++i) {
46         int k = m / fac[n - i];
47         int l = k + 1, r = n, res = 1;
48         while(l <= r) {
49             int mid = (l + r) / 2;
50             if(tr.query(mid - 1) <= k) {
51                 res = mid;
52                 l = mid + 1;
53             } else {
54                 r = mid - 1;
55             }
56         }
57         tr.update(res, -1);
58         m = m % fac[n - i];
59         std::cout << res << " \n"[i == n];
60     }
61 }
62 return 0;
63 }

```

## 6.5 日期公式

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 constexpr int BASE = 1900;
5 constexpr int DAY[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
6 constexpr int PRE[] = {0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365};
7
8 bool isLeapYear(int y) {
9     return y % 400 == 0 || (y % 4 == 0 && y % 100 != 0);
10 }
11

```

```

12 int dayId(int y, int m, int d) { // (y, m, d) 离 (BASE, 1, 1) 的天数, 从0开始
13     int res = (y - BASE) * 365 + PRE[m - 1] + d - 1;
14     int y1 = BASE - 1, y2 = y - 1;
15     res += (y2 / 4 - y2 / 100 + y2 / 400);
16     res -= (y1 / 4 - y1 / 100 + y1 / 400);
17     return res + (m > 2 && isLeapYear(y));
18 }
19
20 int wday(int y, int m, int d) { //求周几
21     return (d + 2 * m + 3 * (m + 1) / 5 + y + y / 4 - y / 100 + y / 400) % 7 + 1;
22 }
23
24 std::array<int, 3> idToDate(i64 id) { // 通过 dayId 反向获取日期
25     int y = BASE + (id * 10000) / 3652425;
26     while (dayId(y, 1, 1) > id) y--;
27     while (dayId(y + 1, 1, 1) <= id) y++;
28     int m = (id - dayId(y, 1, 1)) * 1000 / 30419 + 1;
29     while (dayId(y, m, 1) > id) m--;
30     while (dayId(y, m + 1, 1) <= id) m++;
31     int d = id - dayId(y, m, 1) + 1;
32     return {y, m, d};
33 }
34
35 int main() {
36     std::ios::sync_with_stdio(false);
37     std::cin.tie(nullptr);
38
39     return 0;
40 }

```

## 6.6 高精度

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 // using i128= __int128;
5 // std::istream&operator>>(std::istream &is,i128 &n){
6 //     std::string s;is>>s;
7 //     n=0;
8 //     for(char i:s) n=n*10+i-'0';
9 //     return is;
10 // }
11 // std::ostream &operator<<(std::ostream &os,i128 n){
12 //     std::string s;
13 //     while(n){
14 //         s+='0'+n%10;
15 //         n/=10;
16 //     }

```

```

17 // std::reverse(s.begin(),s.end());
18 // return os<<s;
19 // }
20
21 struct Bigint {
22     std::string a;
23     int sign;
24     Bigint() {}
25     Bigint(std::string b) {
26         (*this) = b;
27     }
28     int size() {
29         return a.size();
30     }
31     Bigint normalize(int newSign) { //removes leading 0, fixes sign (base)
32         for(int i = a.size() - 1; i > 0 && a[i] == '0'; --i) {
33             a.erase(a.begin() + i);
34         }
35         sign = (a.size() == 1 && a[0] == '0') ? 1 : newSign;
36         return (*this);
37     }
38     void operator=(std::string b) {
39         a = b[0] == '-' ? b.substr(1) : b;
40         reverse(a.begin(), a.end());
41         this->normalize(b[0] == '-' ? -1 : 1);
42     }
43     bool operator<(const Bigint &b) const {
44         if(sign != b.sign) {
45             return sign < b.sign;
46         }
47         if(a.size() != b.a.size()) {
48             return sign == 1 ? a.size() < b.a.size() : a.size() > b.a.size();
49         }
50         for(int i = a.size() - 1; i >= 0; --i) {
51             if(a[i] != b.a[i]) {
52                 return sign == 1 ? a[i] < b.a[i] : a[i] > b.a[i];
53             }
54         }
55         return false;
56     }
57     bool operator==(const Bigint &b) const {
58         return (a == b.a && sign == b.sign);
59     }
60     bool operator!=(const Bigint &b) const {
61         return !operator==(b);
62     }
63     Bigint operator+(Bigint b) {
64         if(sign != b.sign) {

```

```

65         return (*this) - (-b); //don't modify here
66     }
67     Bigint c;
68     for(int i = 0, carry = 0; i < a.size() || i < b.size() || carry; ++i) {
69         carry += (i < a.size() ? a[i] - 48 : 0) + (i < b.a.size() ? b.a[i] - 48
70 : 0);
71         c.a += (carry % 10 + 48);
72         carry /= 10;
73     }
74     return c.normalize(sign);
75 }
76 Bigint operator-(Bigint b) {
77     sign *= -1;
78     return (*this);
79 }
80 Bigint operator-(Bigint b) {
81     if(sign != b.sign) {
82         return (*this) + (-b);
83     }
84     int s = sign; sign = b.sign = 1;
85     if((*this) < b) {
86         return (b - (-(*this))).normalize(-s);
87     }
88     Bigint c;
89     for(int i = 0, borrow = 0; i < a.size(); ++i) {
90         borrow = (a[i] - borrow - (i < b.size() ? b.a[i] : 48));
91         c.a += (borrow >= 0 ? borrow + 48 : borrow + 58);
92         borrow = (borrow >= 0 ? 0 : 1);
93     }
94     return c.normalize(s);
95 }
96 Bigint operator*(Bigint b) {
97     Bigint c("0");
98     for(int i = 0, k = a[i] - 48; i < a.size(); ++i, k = a[i] - 48) {
99         while(k-- > 0) c = c + b;
100         b.a.insert(b.a.begin(), '0');
101     }
102     return c.normalize(sign * b.sign);
103 }
104 Bigint operator/(Bigint b) {
105     assert(b != Bigint("0"));
106     if(b.size() == 1 && b.a[0] == '0') {
107         b.a[0] /= (b.a[0] - 48);
108     }
109     Bigint c("0"), d;
110     for(int j = 0; j < a.size(); ++j) {
111         d.a += "0";

```

```

112     int dSign = sign * b.sign; b.sign = 1;
113     for(int i = a.size() - 1; i >= 0; --i) {
114         c.a.insert( c.a.begin(), '0');
115         c = c + a.substr( i, 1 );
116         while(!(c < b)) {
117             c = c - b, d.a[i]++;
118         }
119     }
120     return d.normalize(dSign);
121 }
122 BigInt operator%(BigInt b) {
123     assert(b != BigInt("0"));
124     if(b.size() == 1 && b.a[0] == '0') {
125         b.a[0] /= (b.a[0] - 48);
126     }
127     BigInt c("0");
128     b.sign = 1;
129     for(int i = a.size() - 1; i >= 0; --i) {
130         c.a.insert(c.a.begin(), '0');
131         c = c + a.substr(i, 1);
132         while(!( c < b )) c = c - b;
133     }
134     return c.normalize(sign);
135 }
136 friend std::istream& operator>>(std::istream& is, BigInt& integer) {
137     std::string input;
138     is >> input;
139     integer = input;
140     return is;
141 }
142 friend std::ostream& operator<<(std::ostream& os, const BigInt& integer) {
143     if (integer.sign == -1) {
144         os << "-";
145     }
146     for (int i = integer.a.size() - 1; i >= 0; --i) {
147         os << integer.a[i];
148     }
149     return os;
150 }
151 };
152
153 int main() {
154     BigInt a, b;
155     std::cin >> a >> b;
156     std::cout << a + b << '\n';
157     std::cout << a - b << '\n';
158     std::cout << a * b << '\n';
159     std::cout << a / b << '\n';

```

```

160     std::cout << a % b << '\n';
161     std::cout << (a == b ? "" : "not ") << "equal\n";
162     std::cout << "a is " << (a < b ? "" : "not") << "smaller than b\n";
163     std::cout << "the max number is:" << std::max(a, b) << '\n';
164     std::cout << "the min number is:" << std::min(a, b) << '\n';
165     return 0;
166 }

```

## 6.7 高维前缀和

时间复杂度:  $O(n2^n)$

空间复杂度:  $O(n2^n)$

用途: 位集合中, 求出某个集合的所有子集值之和以及其他可加性操作

模板题: [AtCoder ARC100 C](#)

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 int main() {
5     std::ios::sync_with_stdio(false);
6     std::cin.tie(nullptr);
7     int n;
8     std::cin >> n;
9     std::vector<std::pair<int, int>> v(1 << n);
10    for(int i = 0; i < (1 << n); ++i) {
11        std::cin >> v[i].first;
12    }
13    for(int i = 0; i < n; ++i) {
14        for(int j = 0; j < (1 << n); ++j) {
15            if(j >> i & 1) { //条件取反 !(j >> i & 1) 即为高维后缀和
16                //f[j] = f[j] + f[j ^ (1 << i)]; 一般情况: 求真子集和
17                if(v[j ^ (1 << i)].first > v[j].first) {
18                    v[j].second = v[j].first;
19                    v[j].first = v[j ^ (1 << i)].first;
20                } else if(v[j ^ (1 << i)].first > v[j].second) {
21                    v[j].second = v[j ^ (1 << i)].first;
22                }
23            }
24        }
25    }
26    int ans = 0;
27    for(int i = 1; i < (1 << n); ++i) {
28        ans = std::max(ans, v[i].first + v[i].second);
29        std::cout << ans << '\n';
30    }
31    return 0;
32 }

```

6.8 命令行

```
1 ***** bat *****
2 @echo off
3 g++ %1.cpp -std=c++20 -O2 -Wall -o %1 -D_GLIBCXX_DEBUG
4 .\%1 < in.txt > out.txt
5 @REM type out.txt
6 *****
7
8 ***** sh ***** chmod +x
9 #!/bin/bash
10 g++ -std=c++20 -O2 -Wall "$1.cpp" -o "$1" -D_GLIBCXX_DEBUG
11 ./"$1" < in.txt > out.txt
12 cat out.txt
13 *****
14
15 ***** sh ***** chmod +x
16 while true; do
17     ./gen > 1.in
18     ./std < 1.in > std.out
19     ./my < 1.in > my.out
20     if diff my.out std.out; then
21         echo ac
22     else
23         echo wa
24         break
25     fi
26 done
27 *****
28
29 ***** bat *****
30 @echo off
31 :loop
32     gen.exe > 1.in
33     std.exe < 1.in > std.out
34     my.exe < 1.in > my.out
35     fc my.out std.out > nul
36     if %errorlevel%==0 (
37         echo ac
38         goto loop
39     ) else (
40         echo wa
41         goto :eof
42     )
43 *****
```

7 附录

7.1 编译参数

-D\_GLIBCXX\_DEBUG : STL debugmode  
-fsanitize=address : 内存错误检查  
-fsanitize=undefined :UB 检查

7.2 OJ 测试

```
1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 int main() {
5     std::ios::sync_with_stdio(false);
6     std::cin.tie(nullptr);
7     volatile unsigned j = 1;
8     for(unsigned i = 1; i <= (unsigned)65E7; i++) {
9         j = 0;
10        j += i;
11        j *= i;
12        j /= i;
13    }
14    return 0;
15 }
```

Local	QOJ	AtCoder	LuoGu	Codeforces	Nowcoder
1000ms	1090ms	1120ms	1572ms	1718ms	18070ms

7.3 组合数学公式

性质 1:

$$C_n^m = C_n^{n-m}$$

性质 2:

$$C_{n+m+1}^m = \sum_{i=0}^m C_{n+i}^i$$

性质 3:

$$C_n^m \cdot C_m^r = C_n^r \cdot C_{n-r}^{m-r}$$

性质 4 (二项式定理):



$\sum_{i=0}^n (C_n^i \cdot x^i) = (1 + x)^n$

$\sum_{i=0}^n C_n^i = 2^n$

性质 5:

$\sum_{i=0}^n ((-1)^i \cdot C_n^i) = 0$

性质 6:

$C_n^0 + C_n^2 + \cdots = C_n^1 + C_n^3 + \cdots = 2^{n-1}$

性质 7:

$C_{n+m}^r = \sum_{i=0}^{\min(n,m,r)} (C_n^i \cdot C_m^{r-i})$

$C_{n+m}^n = C_{n+m}^m = \sum_{i=0}^{\min(n,m)} (C_n^i \cdot C_m^i), \quad (r = n \mid r = m)$

性质 8:

$m \cdot C_n^m = n \cdot C_{n-1}^{m-1}$

性质 9:

$\sum_{i=0}^n (C_n^i \cdot i^2) = n(n + 1) \cdot 2^{n-2}$

性质 10:

$\sum_{i=0}^n (C_n^i)^2 = C_{2n}^n$

7.4 随机素数

979345007 986854057502126921  
935359631 949054338673679153  
931936021 989518940305146613  
984974633 972090414870546877  
984858209 956380060632801307

7.5 常数表

<i>n</i>	log <sub>10</sub> <i>n</i>	<i>n</i> !	<i>C</i> ( <i>n</i> , <i>n</i> /2)	LCM(1.. <i>n</i> )	<i>P<sub>n</sub></i>
2	0.30102999	2	2	2	2
3	0.47712125	6	3	6	3
4	0.60205999	24	6	12	5
5	0.69897000	120	10	60	7
6	0.77815125	720	20	60	11
7	0.84509804	5040	35	420	15
8	0.90308998	40320	70	840	22
9	0.95424251	362880	126	2520	30
10	1.00000000	3628800	252	2520	42
11	1.04139269	39916800	462	27720	56
12	1.07918125	479001600	924	27720	77
15	1.17609126	1.31e12	6435	360360	176
20	1.30103000	2.43e18	184756	232792560	627
25	1.39794001	1.55e25	5200300	26771144400	1958
30	1.47712125	2.65e32	155117520	1.444e14	5604
<i>P<sub>n</sub></i>	37338 <sub>40</sub>	204226 <sub>50</sub>	966467 <sub>60</sub>	190569292 <sub>100</sub>	1e9 <sub>114</sub>

max ω(*n*): 小于等于 *n* 中的数最大质因数个数

max *d*(*n*): 小于等于 *n* 中的数最大因数个数

π(*n*): 小于等于 *n* 中的数最大互质数个数

<i>n</i> ≤	10	100	1e3	1e4	1e5	1e6
max ω( <i>n</i> )	2	3	4	5	6	7
max <i>d</i> ( <i>n</i> )	4	12	32	64	128	240
π( <i>n</i> )	4	25	168	1229	9592	78498
<i>n</i> ≤	1e7	1e8	1e9	1e10	1e11	1e12
max ω( <i>n</i> )	8	8	9	10	10	11
max <i>d</i> ( <i>n</i> )	448	768	1344	2304	4032	6720
π( <i>n</i> )	664579	5761455	5.08e7	4.55e8	4.12e9	3.7e10
<i>n</i> ≤	1e13	1e14	1e15	1e16	1e17	1e18
max ω( <i>n</i> )	12	12	13	13	14	15
max <i>d</i> ( <i>n</i> )	10752	17280	26880	41472	64512	103680
π( <i>n</i> )	Prime number theorem: π( <i>x</i> ) ~ $\frac{x}{\log(x)}$					