

ICPC Template Library

postpone | Shu-i64

December 17, 2025

v1.0 •••••

ref: jiangly, skip2004, zeemanz, capps

Attention: 0-indexed, $[l, r)$.

Contents

1 Data Structure	2
1.1 Disjoint Set Union	2
1.2 Fenwick Tree	3
1.3 Lazy Segment Tree	3
1.4 Sparse Table	5
1.5 Mo	6
1.6 Cartesian Tree	7
1.7 Lichao Segment Tree	7
2 Math	8
2.1 Formula	8
2.2 Exgcd	8
2.3 Phi	9
2.4 Sieve	9
2.5 Pollard's Rho	9
2.6 Combination	10
2.7 Linear Basis	10
2.8 Gaussian Elimination	10
2.9 Polynomial	11
2.9.1 Lagrange Interpolation	11
2.9.2 Number Theory Transform	12
2.9.3 Fast Walsh Transform	17
2.9.4 Sum of Subset	17
3 Graph	18
3.1 Tree	18
3.1.1 Lowest Common Ancestor	18
3.1.2 Heavy-Light Decomposition	18
3.1.3 DSU on Tree	20
3.2 Connectivity	20
3.2.1 Strongly Connected Component	20
3.2.2 Block Cut Tree	21
3.3 Two Sat	21

4 String	22
4.1 String Hash	22
4.2 KMP	22
4.3 Z-function	22
4.4 AhoCorasick	22
4.5 Suffix Array	23
4.6 Suffix Automaton	24
4.7 Palindromic Tree	25
5 Geometry	25
6 Util	29
6.1 Mod Integer	29
6.2 Fraction	31
7 Tables	32
7.1 Constant	32

蒋凌宇

1 Data Structure

1.1 Disjoint Set Union

```

1 struct DSU {
2     std::vector<int> f, siz;
3     DSU() {}
4     DSU(int n) {
5         init(n);
6     }
7     void init(int n) {
8         f.resize(n);
9         std::iota(f.begin(), f.end(), 0);
10        siz.assign(n, 1);
11    }
12    int find(int x) {
13        while (x != f[x]) {
14            x = f[x] = f[f[x]];
15        }
16        return x;
17    }
18    bool same(int x, int y) {
19        return find(x) == find(y);
20    }
21    bool merge(int x, int y) {
22        x = find(x);
23        y = find(y);
24        if (x == y) {
25            return false;
26        }
27        siz[x] += siz[y];
28        f[y] = x;
29        return true;
30    }
31    int size(int x) {
32        return siz[find(x)];
33    }
34};
35
36 // DSU with Rollback
37 struct DSU {
38     std::vector<std::pair<int &, int>> his;
39     std::vector<int> f, siz;
40     DSU () {}
41     DSU(int n) {
42         init(n);
43     }
44     void init(int n) {
45         f.resize(n);
46         std::iota(f.begin(), f.end(), 0);
47         siz.assign(n, 1);

```

```

48     }
49     void set(int &a, int b) {
50         his.emplace_back(a, a);
51         a = b;
52     }
53     int find(int x) {
54         while (x != f[x]) {
55             x = f[x];
56         }
57         return x;
58     }
59     bool merge(int x, int y) {
60         x = find(x);
61         y = find(y);
62         if (x == y) {
63             return false;
64         }
65         if (siz[x] < siz[y]) {
66             std::swap(x, y);
67         }
68         set(siz[x], siz[x] + siz[y]);
69         set(f[y], x);
70         return true;
71     }
72     bool same(int x, int y) {
73         return find(x) == find(y);
74     }
75     int cur() {
76         return his.size();
77     }
78     void rollback(int t) {
79         while (his.size() > t) {
80             auto [x, y] = his.back();
81             x = y;
82             his.pop_back();
83         }
84     }
85 };
86
87 // Maintain whether each connected component is bipartite
88 struct DSU {
89     std::vector<std::pair<int &, int>> his;
90     int n;
91     std::vector<int> f, g, bip;
92     DSU(int n_) : n(n_), f(n, -1), g(n), bip(n, 1) {}
93     std::pair<int, int> find(int x) {
94         if (f[x] < 0) {
95             return {x, 0};
96         }
97         auto [u, v] = find(f[x]);
98         return {u, v ^ g[x]};

```

postpone

```

99 }
100 void set(int &a, int b) {
101     his.emplace_back(a, a);
102     a = b;
103 }
104 void merge(int a, int b, int &ans) {
105     auto [u, xa] = find(a);
106     auto [v, xb] = find(b);
107     int w = xa ^ xb ^ 1;
108     if (u == v) {
109         if (bip[u] && w) {
110             set(bip[u], 0);
111             ans--;
112         }
113         return;
114     }
115     if (f[u] > f[v]) {
116         std::swap(u, v);
117     }
118     ans -= bip[u];
119     ans -= bip[v];
120     set(bip[u], bip[u] && bip[v]);
121     set(f[u], f[u] + f[v]);
122     set(f[v], u);
123     set(g[v], w);
124     ans += bip[u];
125 }
126 int cur() {
127     return his.size();
128 }
129 void rollback(int t) {
130     while (his.size() > t) {
131         auto [x, y] = his.back();
132         x = y;
133         his.pop_back();
134     }
135 }
136 };

```

1.2 Fenwick Tree

```

1 template <typename T>
2 struct Fenwick {
3     int n;
4     std::vector<T> a;
5     Fenwick(int n_ = 0) {
6         init(n_);
7     }
8     void init(int n_) {
9         n = n_;

```

```

10         a.assign(n, T{});
11     }
12     void add(int x, const T &v) {
13         for (int i = x + 1; i <= n; i += i & -i) {
14             a[i - 1] = a[i - 1] + v;
15         }
16     }
17     T sum(int x) {
18         T ans{};
19         for (int i = x; i > 0; i -= i & -i) {
20             ans = ans + a[i - 1];
21         }
22         return ans;
23     }
24     T rangeSum(int l, int r) {
25         return sum(r) - sum(l);
26     }
27     int select(const T &k) {
28         int x = 0;
29         T cur{};
30         for (int i = 1 << std::lg(n); i; i /= 2) {
31             if (x + i <= n && cur + a[x + i - 1] <= k) {
32                 x += i;
33                 cur = cur + a[x - 1];
34             }
35         }
36         return x;
37     }
38 };

```

1.3 Lazy Segment Tree

```

1 template <class Info, class Tag>
2 struct LazySegmentTree {
3     int n;
4     std::vector<Info> info;
5     std::vector<Tag> tag;
6
7     LazySegmentTree() = delete;
8     LazySegmentTree(int n_, const Info &v_ = {}) { init(std::vector<Info>(n_, v_)); }
9     template <class T>
10    LazySegmentTree(const std::vector<T> &data) { init(data); }
11
12    template <class T>
13    void init(const std::vector<T> &data) {
14        n = data.size();
15        info.assign(4 << std::lg(n), {});
16        tag.assign(4 << std::lg(n), {});
17    }

```

```

18     auto build = [&](auto self, int p, int l, int r) → void {
19         if (r - l == 1) {
20             info[p] = data[l];
21             return;
22         }
23         int m = (l + r) / 2;
24         self(self, 2 * p, l, m);
25         self(self, 2 * p + 1, m, r);
26         pull(p);
27     };
28     build(build, 1, 0, n);
29 }
30
31 void pull(int p) {
32     info[p] = info[2 * p] + info[2 * p + 1];
33 }
34 void apply(int p, const Tag &v) {
35     info[p].apply(v);
36     tag[p].apply(v);
37 }
38 void push(int p) {
39     apply(2 * p, tag[p]);
40     apply(2 * p + 1, tag[p]);
41     tag[p] = {};
42 }
43 void modify(int p, int l, int r, int x, const Info &v) {
44     if (r - l == 1) {
45         info[p] = v;
46         return;
47     }
48     int m = (l + r) / 2;
49     push(p);
50     if (x < m) {
51         modify(2 * p, l, m, x, v);
52     } else {
53         modify(2 * p + 1, m, r, x, v);
54     }
55     pull(p);
56 }
57 void modify(int p, const Info &v) {
58     modify(1, 0, n, p, v);
59 }
60 Info rangeQuery(int p, int l, int r, int x, int y) {
61     if (l ≥ y || r ≤ x) {
62         return {};
63     }
64     if (l ≥ x && r ≤ y) {
65         return info[p];
66     }
67     int m = (l + r) / 2;
68     push(p);
69     return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
70 }
71 Info rangeQuery(int l, int r) {
72     return rangeQuery(1, 0, n, l, r);
73 }
74 void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
75     if (l ≥ y || r ≤ x) {
76         return;
77     }
78     if (l ≥ x && r ≤ y) {
79         apply(p, v);
80         return;
81     }
82     int m = (l + r) / 2;
83     push(p);
84     rangeApply(2 * p, l, m, x, y, v);
85     rangeApply(2 * p + 1, m, r, x, y, v);
86     pull(p);
87 }
88 void rangeApply(int l, int r, const Tag &v) {
89     return rangeApply(1, 0, n, l, r, v);
90 }
91 template <class F>
92 int findFirst(int p, int l, int r, int x, int y, const F &pred) {
93     if (l ≥ y || r ≤ x) {
94         return -1;
95     }
96     if (l ≥ x && r ≤ y && !pred(info[p])) {
97         return -1;
98     }
99     if (r - l == 1) {
100        return l;
101    }
102    push(p);
103    int m = (l + r) / 2;
104    int res = findFirst(2 * p, l, m, x, y, pred);
105    if (res == -1) {
106        res = findFirst(2 * p + 1, m, r, x, y, pred);
107    }
108    return res;
109 }
110 template <class F>
111 int findFirst(int l, int r, const F &pred) {
112     return findFirst(1, 0, n, l, r, pred);
113 }
114 template <class F>
115 int findLast(int p, int l, int r, int x, int y, const F &pred) {
116     if (l ≥ y || r ≤ x) {
117         return -1;
118     }

```

```

119     if (l >= x && r <= y && !pred(info[p])) {
120         return -1;
121     }
122     if (r - l == 1) {
123         return l;
124     }
125     push(p);
126     int m = (l + r) / 2;
127     int res = findLast(2 * p + 1, m, r, x, y, pred);
128     if (res == -1) {
129         res = findLast(2 * p, l, m, x, y, pred);
130     }
131     return res;
132 }
133 template <class F>
134 int findLast(int l, int r, const F &pred) {
135     return findLast(1, 0, n, l, r, pred);
136 }
137 };
138
139 struct Tag {
140     void apply(const Tag &t) {
141     }
142 };
143
144 struct Info {
145     void apply(const Tag &t) {
146     }
147 };
148
149 Info operator+(const Info &a, const Info &b) {
150 }
```

postpone

1.4 Sparse Table

```
1 // O(n log n) - O(1)
2 // template <class T, T e, T (*F)(T, T)>, if cpp < 20
3 template <class T, T e, auto F>
4 struct SparseTable {
5     int n;
6     std::vector<std::vector<T>> a;
7     SparseTable(const std::vector<T> &v = {}) {
8         init(v);
9     }
10    void init(const std::vector<T> &v) {
11        n = v.size();
12        if (n == 0) {
13            return;
14        }
15        const int m = std::lg(n);
```

```

67 }
68     for (int i = n - 2; i >= 0; i--) {
69         if (i % B != B - 1) {
70             suf[i] = std::min(suf[i], suf[i + 1], cmp);
71         }
72     }
73     for (int j = 0; j < lg; j++) {
74         for (int i = 0; i + (2 << j) <= M; i++) {
75             a[j + 1][i] = std::min(a[j][i], a[j][i + (1 << j)], cmp);
76         }
77     }
78     for (int i = 0; i < M; i++) {
79         const int l = i * B;
80         const int r = std::min(1U * n, l + B);
81         u64 s = 0;
82         for (int j = l; j < r; j++) {
83             while (s && cmp(v[j], v[std::lg(s) + l])) {
84                 s *= 1ULL << std::lg(s);
85             }
86             s |= 1ULL << (j - l);
87             stk[j] = s;
88         }
89     }
90 }
91 T operator()(int l, int r) {
92     if (l >= r) {
93         return e;
94     }
95     if (l / B != (r - 1) / B) {
96         T ans = std::min(suf[l], pre[r - 1], cmp);
97         l = l / B + 1;
98         r = r / B;
99         if (l < r) {
100             int k = std::lg(r - l);
101             ans = std::min({ans, a[k][l], a[k][r - (1 << k)]}, cmp);
102         }
103         return ans;
104     } else {
105         int x = B * (l / B);
106         return ini[__builtin_ctzll(stk[r - 1] >> (l - x)) + l];
107     }
108 }
109 };

```

1.5 Mo

```

1 {
2     std::vector<std::array<int, 3>> ask(q);
3     // ...
4     std::ranges::sort(ask, [&](auto i, auto j) {

```

```

5         if (i[0] / B != j[0] / B) {
6             return i[0] < j[0];
7         }
8         return (i[0] / B) % 2 ? i[1] > j[1] : i[1] < j[1];
9     });
10    // ...
11    // [l, r)
12    int L = 0, R = 0;
13    for (auto [l, r, i] : ask) {
14        while (L > l) {
15            add(--L);
16        }
17        while (R < r) {
18            add(R++);
19        }
20        while (L < l) {
21            del(L++);
22        }
23        while (R > r) {
24            del(--R);
25        }
26        // ans[i] = ?
27    }
28 }
29
30 // Mo With Modify
31 // B = n ^ (2 / 3)
32 {
33     // ...
34     std::vector<std::array<int, 4>> ask;
35     std::vector<std::pair<int, int>> mod;
36     for (int i = 0; i < m; i++) {
37         char o;
38         int x, y;
39         std::cin >> o >> x >> y;
40         x--;
41         if (modify) {
42             mod.emplace_back(x, y);
43         } else {
44             ask.push_back({x, y, (int)mod.size(), (int)ask.size()});
45         }
46     }
47     std::ranges::sort(ask, [&](auto i, auto j) {
48         if (i[0] / B != j[0] / B)
49             return i[0] < j[0];
50         if (i[1] / B != j[1] / B)
51             return i[1] < j[1];
52         return (i[1] / B) & 1 ? i[2] < j[2] : i[2] > j[2];
53     });
54     auto add = [&](int c) {
55         // ...

```

postpone

```

56 };
57 auto del = [&](int c) {
58     // ...
59 };
60 auto modify = [&](int p, int l, int r) {
61     // if (l <= x and x < r) then del(ori) and add(cur)
62
63     // first modify : x → y
64     // second modify : y → x
65     // using swap
66 };
67 // [l, r)
68 int L = 0, R = 0, T = 0;
69 for (auto [l, r, t, i] : que) {
70     while (l < L) {
71         add(a[--L]);
72     }
73     while (R < r) {
74         add(a[R++]);
75     }
76     while (L < l) {
77         del(a[L++]);
78     }
79     while (R > r) {
80         del(a[--R]);
81     }
82     while (T < t) {
83         modify(T++, l, r);
84     }
85     while (T > t) {
86         modify(--T, l, r);
87     }
88     // ans[i] = ?
89 }
90 }
```

1.6 Cartesian Tree

```

1 // root = rangeMin
2 vector<int> lc(n, -1), rc(n, -1);
3 vector<int> stk;
4 for (int i = 0; i < n; i++) {
5     while (not stk.empty() and p[i] < p[stk.back()]) {
6         int x = stk.back();
7         stk.pop_back();
8
9         rc[x] = lc[i];
10        lc[i] = x;
11    }
12    stk.push_back(i);
13 }
```

```

13 }
14 while (stk.size() > 1) {
15     int x = stk.back();
16     stk.pop_back();
17     rc[stk.back()] = x;
18 }
```

1.7 Lichao Segment Tree

```

1 template <class T, T inf, class C = std::less<>>
2 struct LiChaoSegmentTree {
3     static constexpr C cmp = {};
4     struct Line {
5         int i;
6         T k, b;
7         constexpr Line(int i = std::min(-1, 1, cmp), T k = 0, T b = std::max
8             (-inf, inf, cmp)) : i{i}, k{k}, b{b} {}
9         constexpr std::pair<T, int> operator()(int x) const {
10             return {k * x + b, i};
11         }
12     };
13     struct Node {
14         Node *l, *r;
15         Line f;
16         Node() : l{}, r{}, f{} {}
17     };
18     int n;
19     Node *t;
20     LiChaoSegmentTree(int n = 0) {
21         init(n);
22     }
23     void init(int n) {
24         this->n = n;
25         t = nullptr;
26     }
27     void insert(Node *&p, int l, int r, int x, int y, Line f) {
28         if (l >= y || r <= x) {
29             return;
30         }
31         if (p == nullptr) {
32             p = new Node();
33         }
34         int m = (l + r) / 2;
35         if (l >= x && r <= y) {
36             if (cmp(f(m), p->f(m))) {
37                 std::swap(f, p->f);
38             }
39             if (r - l == 1) {
40                 return;
41             }
42         }
43         if (x < m) {
44             insert(p->l, l, m, x, y, f);
45         }
46         if (y > m) {
47             insert(p->r, m, r, x, y, f);
48         }
49     }
50     void query(Node *p, int l, int r, int x, int y, Line &f) const {
51         if (l >= y || r <= x) {
52             return;
53         }
54         if (p == nullptr) {
55             return;
56         }
57         if (l >= x && r <= y) {
58             f = p->f;
59         }
60         if (x < m) {
61             query(p->l, l, m, x, y, f);
62         }
63         if (y > m) {
64             query(p->r, m, r, x, y, f);
65         }
66     }
67     void print(Node *p) const {
68         if (p == nullptr) {
69             return;
70         }
71         cout << p->f << endl;
72         print(p->l);
73         print(p->r);
74     }
75     void print() const {
76         print(t);
77     }
78     void printLine(Node *p) const {
79         if (p == nullptr) {
80             return;
81         }
82         cout << p->f << endl;
83         printLine(p->l);
84         printLine(p->r);
85     }
86     void printLine() const {
87         printLine(t);
88     }
89     void printNode(Node *p) const {
90         if (p == nullptr) {
91             return;
92         }
93         cout << p->f << endl;
94         printNode(p->l);
95         printNode(p->r);
96     }
97     void printNode() const {
98         printNode(t);
99     }
100 };
```

```

41     if (cmp(f(l), p->f(l))) {
42         insert(p->l, l, m, x, y, f);
43     } else {
44         insert(p->r, m, r, x, y, f);
45     }
46 } else {
47     insert(p->l, l, m, x, y, f);
48     insert(p->r, m, r, x, y, f);
49 }
50 }
51 void insert(int l, int r, Line f) {
52     insert(t, 0, n, l, r, f);
53 }
54 void insert(Line f) {
55     insert(t, 0, n, 0, n, f);
56 }
57 // {val, id}
58 std::pair<T, int> query(Node *p, int l, int r, int x) {
59     if (p == nullptr) {
60         return Line{}(x);
61     }
62     if (r - l == 1) {
63         return p->f(x);
64     }
65     int m = (l + r) / 2;
66     if (x < m) {
67         return std::min(p->f(x), query(p->l, l, m, x), cmp);
68     } else {
69         return std::min(p->f(x), query(p->r, m, r, x), cmp);
70     }
71 }
72 std::pair<T, int> query(int x) {
73     return query(t, 0, n, x);
74 }
75 };

```

2 Math

2.1 Formula

- Ex Euler's Theorem:

$$a^b \equiv \begin{cases} a^{b \bmod \varphi(m)}, & \gcd(a, b) = 1 \\ a^{(b \bmod \varphi(m)) + \varphi(m)}, & \gcd(a, b) \neq 1, b \geq \varphi(m) \end{cases} \pmod{m}$$

- Euclidean algorithm: $\gcd(a, b) = \gcd(b, a \bmod b)$
- Binomial Inversion

$$f(n) = |\bigcap_{1 \leq i \leq n} A_i|, g(n) = |\bigcap_{1 \leq i \leq n} A_i^C|$$

$$f(n) = \sum_{i=0}^n (-1)^i \binom{n}{i} g(i)$$

$$g(n) = \sum_{i=0}^n (-1)^i \binom{n}{i} f(i)$$

$$f(n) = \sum_{i=0}^n \binom{n}{i} h(i)$$

$$h(n) = \sum_{i=0}^n (-1)^{n-i} \binom{n}{i} f(i)$$

$f(n)$: select at least n . $\leftrightarrow g(n)$: select exactly n .

$$f(n) = \sum_{i=n}^m \binom{i}{n} g(i)$$

$$g(n) = \sum_{i=n}^m (-1)^{i-n} \binom{i}{n} f(i)$$

2.2 Exgcd

```

1 // ax + by = gcd(a, b)
2 // if a ≠ 0, b ≠ 0 then -b ≤ x ≤ b, -a ≤ y ≤ a
3 int exgcd(int a, int b, int &x, int &y) {
4     if (b == 0) {
5         x = 1;
6         y = 0;
7         return a;
8     }
9     int g = exgcd(b, a % b, y, x);
10    y -= a / b * x;
11    return g;
12 }

13
14 // {x, mod} → {_, inv(x)}
15 constexpr std::pair<i64, i64> invGcd(i64 a, i64 b) {
16     a %= b;
17     if (a < 0) {
18         a += b;
19     }
20     if (a == 0) {
21         return {b, 0};
22     }
23     i64 s = b, t = a;
24     i64 m0 = 0, m1 = 1;
25     while (t) {
26         i64 u = s / t;

```

postpone

```

27     s -= t * u;
28     m0 -= m1 * u;
29
30     std::swap(s, t);
31     std::swap(m0, m1);
32 }
33 if (m0 < 0) {
34     m0 += b / s;
35 }
36 return {s, m0};
37 }
```

2.3 Phi

```

1 int phi(int n) {
2     int res = n;
3     for (int i = 2; i * i <= n; i++) {
4         if (n % i == 0) {
5             while (n % i == 0) {
6                 n /= i;
7             }
8             res = res / i * (i - 1);
9         }
10    }
11    if (n > 1) {
12        res = res / n * (n - 1);
13    }
14    return res;
15 }
```

2.4 Sieve

```

1 vector<int> minp, primes;
2 vector<int> phi, mu;
3
4 void sieve(int n) {
5     minp.assign(n + 1, 0);
6     phi.assign(n + 1, 0);
7     mu.assign(n + 1, 0);
8     primes.clear();
9
10    phi[1] = 1;
11    mu[1] = 1;
12
13    for (int i = 2; i <= n; i++) {
14        if (minp[i] == 0) {
15            minp[i] = i;
16            primes.push_back(i);
17            phi[i] = i - 1;
18        }
19    }
20 }
```

```

18         mu[i] = -1;
19     }
20
21     for (auto p : primes) {
22         if (i * p > n) {
23             break;
24         }
25         minp[i * p] = p;
26         if (p == minp[i]) {
27             phi[i * p] = phi[i] * p;
28             mu[i * p] = 0;
29             break;
30         } else {
31             phi[i * p] = phi[i] * (p - 1);
32             mu[i * p] = -mu[i];
33         }
34     }
35 }
36 }
37
38 bool isprime(int n) {
39     return minp[n] == n;
40 }
```

2.5 Pollard's Rho

```

1 std::mt19937_64 rng(std::chrono::steady_clock::now().time_since_epoch().count());
2
3 i64 factor(i64 n) {
4     if (n % 2 == 0) {
5         return 2;
6     }
7     if (isPrime(n)) {
8         return n;
9     }
10    i64 m = 2;
11    while (true) {
12        i64 c = (rng() % (n - 1)) + 1;
13        auto f = [&](i64 x) { return (mul(x, x, n) + c) % n; };
14        i64 d = 1, x = m, y = m, p = 1, q = 0, v = 1;
15        while (d == 1) {
16            y = f(y);
17            q++;
18            v = mul(v, std::abs(x - y), n);
19            if (q % 127 == 0) {
20                d = std::gcd(v, n);
21                v = 1;
22            }
23            if (p == q) {
```

```

24         x = y;
25         p *= 2;
26         q = 0;
27         d = std::gcd(v, n);
28         v = 1;
29     }
30     if (d != n) {
31         return d;
32     }
33     m++;
34 }
35 }
36 }
37 std::vector<i64> factorize(i64 n) {
38     std::vector<i64> p;
39     auto dfs = [&](auto &&self, i64 n) → void {
40         if (isPrime(n)) {
41             p.push_back(n);
42             return;
43         }
44         i64 d = factor(n);
45         self(self, d);
46         self(self, n / d);
47     };
48     dfs(dfs, n);
49     std::sort(p.begin(), p.end());
50     return p;
51 }
52 }
```

2.6 Combination

```

1 {
2     vector<int> fac(n + 1), invfac(n + 1);
3     fac[0] = 1;
4     for (int i = 1; i ≤ n; i++) {
5         fac[i] = mul(fac[i - 1], i);
6     }
7     invfac[n] = power(fac[n], P - 2);
8     for (int i = n; i ≥ 1; i--) {
9         invfac[i - 1] = mul(invfac[i], i);
10    }
11    auto binom = [&](int n, int m) → int {
12        if (n < m or m < 0) {
13            return 0;
14        }
15        return i64(fac[n]) * invfac[m] % P * invfac[n - m] % P;
16    };
17 }
```

2.7 Linear Basis

```

1 // a : base
2 // k : dimension
3 {
4     auto insert = [&](int x) {
5         for (int d = k - 1; d ≥ 0 and x ≠ 0; d--) {
6             if (x >> d & 1) {
7                 if (a[d] == 0) {
8                     a[d] = x;
9                 }
10                x ≈ a[d];
11            }
12        }
13    };
14    // ...
15    int ans = 0;
16    for (int d = k - 1; d ≥ 0; d--) {
17        ans = max(ans, ans ^ a[d]);
18    }
19 }
20 // t : time stamp
21 {
22     auto insert = [&](int x, int i) {
23         for (int d = k - 1; d ≥ 0; d--) {
24             if (x >> d & 1) {
25                 if (i > t[d]) {
26                     swap(i, t[d]);
27                     swap(x, a[d]);
28                 }
29                 x ≈ a[d];
30             }
31         }
32    };
33    auto query = [&](int i) {
34        int res = 0;
35        for (int d = k - 1; d ≥ 0; d--) {
36            if (t[d] ≥ i) {
37                res = max(res, res ^ a[d]);
38            }
39        }
40        return res;
41    };
42 }
```

2.8 Gaussian Elimination

```

1 std::vector<int> operator*(const std::vector<int> &lhs, const std::vector<int>
2 > &rhs) {
3     std::vector<int> res(lhs.size() + rhs.size() - 1);
```

postpone

```

3   for (int i = 0; i < int(lhs.size()); ++i)
4     for (int j = 0; j < int(rhs.size()); ++j)
5       res[i + j] = (res[i + j] + 1ll * lhs[i] * rhs[j]) % P;
6   return res;
7 }
8 std::vector<int> operator%(const std::vector<int> &lhs, const std::vector<int>
9 > &rhs) {
10  auto res = lhs;
11  int m = rhs.size() - 1;
12  int inv = power(rhs.back(), P - 2);
13  for (int i = res.size() - 1; i ≥ m; --i) {
14    int x = 1ll * inv * res[i] % P;
15    for (int j = 0; j < m; ++j)
16      res[i - m + j] = (res[i - m + j] + 1ll * (P - x) * rhs[j]) % P;
17  }
18  if (int(res.size()) > m)
19    res.resize(m);
20  return res;
21 }
22 std::vector<int> gauss(std::vector<std::vector<int>> a, std::vector<int> b) {
23  int n = a.size();
24  for (int i = 0; i < n; ++i) {
25    int r = i;
26    while (a[r][i] == 0)
27      ++r;
28    std::swap(a[i], a[r]);
29    std::swap(b[i], b[r]);
30    int inv = power(a[i][i], P - 2);
31    for (int j = i; j < n; ++j)
32      a[i][j] = 1ll * a[i][j] * inv % P;
33    b[i] = 1ll * b[i] * inv % P;
34    for (int j = 0; j < n; ++j) {
35      if (i == j)
36        continue;
37      int x = a[j][i];
38      for (int k = i; k < n; ++k)
39        a[j][k] = (a[j][k] + 1ll * (P - x) * a[i][k]) % P;
40      b[j] = (b[j] + 1ll * (P - x) * b[i]) % P;
41    }
42  }
43  return b;
44 }
45 std::vector<double> gauss(std::vector<std::vector<double>> a, std::vector<
46 double> b) {
47  int n = a.size();
48  for (int i = 0; i < n; ++i) {
49    double x = a[i][i];
50    for (int j = i; j < n; ++j) a[i][j] /= x;
51    b[i] /= x;

```

```

52    for (int j = 0; j < n; ++j) {
53      if (i == j) continue;
54      x = a[j][i];
55      for (int k = i; k < n; ++k) a[j][k] -= a[i][k] * x;
56      b[j] -= b[i] * x;
57    }
58  }
59  return b;
60 }

```

2.9 Polynomial

2.9.1 Lagrange Interpolation

```

1 // n points → n - 1 polynomial      O(n ^ 2)
2 for (int i = 0; i < n; i++) {
3   Z num = y[i];
4   Z den = 1;
5   for (int j = 0; j < n; j++) {
6     if (i == j) {
7       continue;
8     }
9     num *= (k - x[j]); // f(k)
10    den *= (x[i] - x[j]);
11  }
12  ans += num / den;
13 }
14
15 // Continuous x      O(n)
16 vector<Z> fac(n + 1);
17 fac[0] = 1;
18 for (int i = 1; i ≤ n; i++) {
19   fac[i] = fac[i - 1] * i;
20 }
21
22 vector<Z> pre(n + 1);
23 pre[0] = 1;
24 for (int i = 0; i < n; i++) {
25   pre[i + 1] = pre[i] * (k - i);
26 }
27
28 vector<Z> suf(n + 1);
29 suf[n] = 1;
30 for (int i = n - 1; i ≥ 0; i--) {
31   suf[i] = suf[i + 1] * (k - i);
32 }
33
34 Z ans = 0;
35 for (int i = 0; i < n; i++) {
36   Z res = y[i];

```

```

37     res *= pre[i] * suf[i + 1];
38     res += ((n - 1 - i) % 2 ? -1 : 1) * fac[i] * fac[n - 1 - i];
39
40     ans += res;
41 }

```

2.9.2 Number Theory Transform

```

1 constexpr int P = 998244353;
2
3 int power(int a, int b) {
4     int res = 1;
5     for (; b; b /= 2, a = 1LL * a * a % P) {
6         if (b % 2) {
7             res = 1LL * res * a % P;
8         }
9     }
10    return res;
11 }
12
13 std::vector<int> rev, roots {0, 1};
14
15 void dft(std::vector<int> &a) {
16     int n = a.size();
17     if (int(rev.size()) != n) {
18         int k = __builtin_ctz(n) - 1;
19         rev.resize(n);
20         for (int i = 0; i < n; i++) {
21             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
22         }
23     }
24     for (int i = 0; i < n; i++) {
25         if (rev[i] < i) {
26             std::swap(a[i], a[rev[i]]);
27         }
28     }
29     if (roots.size() < n) {
30         int k = __builtin_ctz(roots.size());
31         roots.resize(n);
32         while ((1 << k) < n) {
33             int e = power(31, 1 << (__builtin_ctz(P - 1) - k - 1));
34             for (int i = 1 << (k - 1); i < (1 << k); i++) {
35                 roots[2 * i] = roots[i];
36                 roots[2 * i + 1] = 1LL * roots[i] * e % P;
37             }
38             k++;
39         }
40     }
41     for (int k = 1; k < n; k *= 2) {

```

```

43         for (int i = 0; i < n; i += 2 * k) {
44             for (int j = 0; j < k; j++) {
45                 int u = a[i + j];
46                 int v = 1LL * a[i + j + k] * roots[k + j] % P;
47                 a[i + j] = (u + v) % P;
48                 a[i + j + k] = (u - v) % P;
49             }
50         }
51     }
52 }
53
54 void idft(std::vector<int> &a) {
55     int n = a.size();
56     std::reverse(a.begin() + 1, a.end());
57     dft(a);
58     int inv = (1 - P) / n;
59     for (int i = 0; i < n; i++) {
60         a[i] = 1LL * a[i] * inv % P;
61     }
62 }
63
64 std::vector<int> mul(std::vector<int> a, std::vector<int> b) {
65     int n = 1, tot = a.size() + b.size() - 1;
66     while (n < tot) {
67         n *= 2;
68     }
69     a.resize(n);
70     b.resize(n);
71     dft(a);
72     dft(b);
73     for (int i = 0; i < n; i++) {
74         a[i] = 1LL * a[i] * b[i] % P;
75     }
76     idft(a);
77     a.resize(tot);
78     return a;
79 }

```

```

1 // with ModIntBase
2 std::vector<int> rev;
3 template <u32 P>
4 std::vector<ModInt<P>> roots{0, 1};
5
6 template <u32 P>
7 constexpr ModInt<P> findPrimitiveRoot() {
8     ModInt<P> i = 2;
9     int k = __builtin_ctz((int)P - 1);
10    while (true) {
11        if (power(i, (P - 1) / 2) != 1) {
12            break;
13        }

```

postpone

```

14     i += 1;
15 }
16 return power(i, (P - 1) >> k);
17 }

18 template <u32 P>
19 constexpr ModInt<P> primitiveRoot = findPrimitiveRoot<P>();
20
21 template <>
22 constexpr ModInt<998244353> primitiveRoot<998244353>{31};
23
24 template <u32 P>
25 constexpr void dft(std::vector<ModInt<P>> &a) {
26     int n = a.size();
27
28     if (int(rev.size()) != n) {
29         int k = __builtin_ctz(n) - 1;
30         rev.resize(n);
31         for (int i = 0; i < n; i++) {
32             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
33         }
34     }
35
36     for (int i = 0; i < n; i++) {
37         if (rev[i] < i) {
38             std::swap(a[i], a[rev[i]]);
39         }
40     }
41
42     if (roots<P>.size() < n) {
43         int k = __builtin_ctz(roots<P>.size());
44         roots<P>.resize(n);
45         while ((1 << k) < n) {
46             auto e = power(primitiveRoot<P>, 1 << (__builtin_ctz(P - 1) - k - 1));
47             for (int i = 1 << (k - 1); i < (1 << k); i++) {
48                 roots<P>[2 * i] = roots<P>[i];
49                 roots<P>[2 * i + 1] = roots<P>[i] * e;
50             }
51             k++;
52         }
53     }
54     for (int k = 1; k < n; k *= 2) {
55         for (int i = 0; i < n; i += 2 * k) {
56             for (int j = 0; j < k; j++) {
57                 ModInt<P> u = a[i + j];
58                 ModInt<P> v = a[i + j + k] * roots<P>[k + j];
59                 a[i + j] = u + v;
60                 a[i + j + k] = u - v;
61             }
62         }
63     }
}

```

```

64     }
65 }
66
67 template <u32 P>
68 constexpr void idft(std::vector<ModInt<P>> &a) {
69     int n = a.size();
70     std::reverse(a.begin() + 1, a.end());
71     dft(a);
72     ModInt<P> inv = (1 - (int)P) / n;
73     for (int i = 0; i < n; i++) {
74         a[i] *= inv;
75     }
76 }
77
78 template <int P = 998244353>
79 struct Poly : public std::vector<ModInt<P>> {
80     using V = ModInt<P>;
81
82     Poly() : std::vector<V>() {}
83     explicit constexpr Poly(int n) : std::vector<V>(n) {}
84
85     explicit constexpr Poly(const std::vector<V> &a) : std::vector<V>(a) {}
86     constexpr Poly(const std::initializer_list<V> &a) : std::vector<V>(a) {}
87
88     template <class InputIt, class = std::_RequireInputIter<InputIt>>
89     explicit constexpr Poly(InputIt first, InputIt last) : std::vector<V>(
90         first, last) {}
91
92     template <class F>
93     explicit constexpr Poly(int n, F f) : std::vector<V>(n) {
94         for (int i = 0; i < n; i++) {
95             (*this)[i] = f(i);
96         }
97     }
98
99     constexpr Poly shift(int k) const {
100         if (k >= 0) {
101             auto b = *this;
102             b.insert(b.begin(), k, 0);
103             return b;
104         } else if (this->size() <= -k) {
105             return Poly();
106         } else {
107             return Poly(this->begin() + (-k), this->end());
108         }
109     }
110
111     constexpr Poly trunc(int k) const {
112         Poly f = *this;
113         f.resize(k);
114         return f;
115     }

```

```

114  constexpr friend Poly operator+(const Poly &a, const Poly &b) {
115    Poly res(std::max(a.size(), b.size()));
116    for (int i = 0; i < a.size(); i++) {
117      res[i] += a[i];
118    }
119    for (int i = 0; i < b.size(); i++) {
120      res[i] += b[i];
121    }
122    return res;
123  }
124  constexpr friend Poly operator-(const Poly &a, const Poly &b) {
125    Poly res(std::max(a.size(), b.size()));
126    for (int i = 0; i < a.size(); i++) {
127      res[i] += a[i];
128    }
129    for (int i = 0; i < b.size(); i++) {
130      res[i] -= b[i];
131    }
132    return res;
133  }
134  constexpr friend Poly operator-(const Poly &a) {
135    std::vector<V> res(a.size());
136    for (int i = 0; i < int(res.size()); i++) {
137      res[i] = -a[i];
138    }
139    return Poly(res);
140  }
141  constexpr friend Poly operator*(Poly a, Poly b) {
142    if (a.size() == 0 || b.size() == 0) {
143      return Poly();
144    }
145    if (a.size() < b.size()) {
146      std::swap(a, b);
147    }
148    int n = 1, tot = a.size() + b.size() - 1;
149    while (n < tot) {
150      n *= 2;
151    }
152    if (((P - 1) & (n - 1)) != 0 || b.size() < 128) {
153      Poly c(a.size() + b.size() - 1);
154      for (int i = 0; i < a.size(); i++) {
155        for (int j = 0; j < b.size(); j++) {
156          c[i + j] += a[i] * b[j];
157        }
158      }
159      return c;
160    }
161    a.resize(n);
162    b.resize(n);
163    dft<P>(a);
164    dft<P>(b);
165    for (int i = 0; i < n; ++i) {
166      a[i] *= b[i];
167    }
168    idft<P>(a);
169    a.resize(tot);
170    return a;
171  }
172  constexpr friend Poly operator*(V a, Poly b) {
173    for (int i = 0; i < int(b.size()); i++) {
174      b[i] *= a;
175    }
176    return b;
177  }
178  constexpr friend Poly operator*(Poly a, V b) {
179    for (int i = 0; i < int(a.size()); i++) {
180      a[i] *= b;
181    }
182    return a;
183  }
184  constexpr friend Poly operator/(Poly a, V b) {
185    for (int i = 0; i < int(a.size()); i++) {
186      a[i] /= b;
187    }
188    return a;
189  }
190  constexpr Poly &operator+=(Poly b) {
191    return (*this) = (*this) + b;
192  }
193  constexpr Poly &operator-=(Poly b) {
194    return (*this) = (*this) - b;
195  }
196  constexpr Poly &operator*=(Poly b) {
197    return (*this) = (*this) * b;
198  }
199  constexpr Poly &operator*=(V b) {
200    return (*this) = (*this) * b;
201  }
202  constexpr Poly &operator/=(V b) {
203    return (*this) = (*this) / b;
204  }
205  constexpr Poly deriv() const {
206    if (this->empty()) {
207      return Poly();
208    }
209    Poly res((int)this->size() - 1);
210    for (int i = 0; i < this->size() - 1; ++i) {
211      res[i] = (i + 1) * (*this)[i + 1];
212    }
213    return res;
214  }
215  constexpr Poly integr() const {

```

postpone

```

216     Poly res(this->size() + 1);
217     for (int i = 0; i < this->size(); ++i) {
218         res[i + 1] = (*this)[i] / (i + 1);
219     }
220     return res;
221 }
222 constexpr Poly inv(int m) const {
223     Poly x{(*this)[0].inv()};
224     int k = 1;
225     while (k < m) {
226         k *= 2;
227         x = (x * (Poly{2} - trunc(k) * x)).trunc(k);
228     }
229     return x.trunc(m);
230 }
231 constexpr Poly log(int m) const {
232     return (deriv() * inv(m)).integr().trunc(m);
233 }
234 constexpr Poly exp(int m) const {
235     Poly x{1};
236     int k = 1;
237     while (k < m) {
238         k *= 2;
239         x = (x * (Poly{1} - x.log(k) + trunc(k))).trunc(k);
240     }
241     return x.trunc(m);
242 }
243 constexpr Poly pow(int k, int m) const {
244     int i = 0;
245     while (i < this->size() && (*this)[i] == 0) {
246         i++;
247     }
248     if (i == this->size() || 1LL * i * k >= m) {
249         return Poly(m);
250     }
251     V v = (*this)[i];
252     auto f = shift(-i) * v.inv();
253     return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * power(v,
254         k);
255 }
256 constexpr Poly sqrt(int m) const {
257     Poly x{1};
258     int k = 1;
259     while (k < m) {
260         k *= 2;
261         x = (x + (trunc(k) * x.inv(k)).trunc(k)) * CInv<2, P>;
262     }
263     return x.trunc(m);
264 }
265 constexpr Poly mult(Poly b) const {
266     if (b.size() == 0) {
267         return Poly();
268     }
269     int n = b.size();
270     std::reverse(b.begin(), b.end());
271     return ((*this) * b).shift(-(n - 1));
272 }
273 constexpr std::vector<V> eval(std::vector<V> x) const {
274     if (this->size() == 0) {
275         return std::vector<V>(x.size(), 0);
276     }
277     const int n = std::max(x.size(), this->size());
278     std::vector<Poly> q(4 * n);
279     std::vector<V> ans(x.size());
280     x.resize(n);
281     std::function<void(int, int, int)> build = [&](int p, int l, int r) {
282         if (r - l == 1) {
283             q[p] = Poly{1, -x[l]};
284         } else {
285             int m = (l + r) / 2;
286             build(2 * p, l, m);
287             build(2 * p + 1, m, r);
288             q[p] = q[2 * p] * q[2 * p + 1];
289         }
290     };
291     build(1, 0, n);
292     std::function<void(int, int, int, const Poly &)> work = [&](int p,
293         int l, int r, const Poly &num) {
294         if (r - l == 1) {
295             if (l < int(ans.size())) {
296                 ans[l] = num[0];
297             }
298         } else {
299             int m = (l + r) / 2;
300             work(2 * p, l, m, num.mult(q[2 * p + 1]).trunc(m - l));
301             work(2 * p + 1, m, r, num.mult(q[2 * p]).trunc(r - m));
302         }
303     };
304     work(1, 0, n, mult(q[1].inv(n)));
305     return ans;
306 };
307 template <int P = 998244353>
308 Poly<P> berlekampMassey(const Poly<P> &s) {
309     Poly<P> c;
310     Poly<P> oldC;
311     int f = -1;
312     for (int i = 0; i < s.size(); i++) {
313         auto delta = s[i];
314         for (int j = 1; j <= c.size(); j++) {
315             delta -= c[j - 1] * s[i - j];

```

```

316 }
317     if (delta == 0) {
318         continue;
319     }
320     if (f == -1) {
321         c.resize(i + 1);
322         f = i;
323     } else {
324         auto d = oldC;
325         d *= -1;
326         d.insert(d.begin(), 1);
327         ModInt<P> df1 = 0;
328         for (int j = 1; j <= d.size(); j++) {
329             df1 += d[j - 1] * s[f + 1 - j];
330         }
331         assert(df1 != 0);
332         auto coef = delta / df1;
333         d *= coef;
334         Poly<P> zeros(i - f - 1);
335         zeros.insert(zeros.end(), d.begin(), d.end());
336         d = zeros;
337         auto temp = c;
338         c += d;
339         if (i - temp.size() > f - oldC.size()) {
340             oldC = temp;
341             f = i;
342         }
343     }
344 }
345 c *= -1;
346 c.insert(c.begin(), 1);
347 return c;
348 }

350 template <int P = 998244353>
351 ModInt<P> linearRecurrence(Poly<P> p, Poly<P> q, i64 n) {
352     int m = q.size() - 1;
353     while (n > 0) {
354         auto newq = q;
355         for (int i = 1; i <= m; i += 2) {
356             newq[i] *= -1;
357         }
358         auto newp = p * newq;
359         newq = q * newq;
360         for (int i = 0; i < m; i++) {
361             p[i] = newp[i * 2 + n % 2];
362         }
363         for (int i = 0; i <= m; i++) {
364             q[i] = newq[i * 2];
365         }
366         n /= 2;
367     }
368     return p[0] / q[0];
369 }

370 struct Comb {
371     int n;
372     std::vector<Z> _fac;
373     std::vector<Z> _invfac;
374     std::vector<Z> _inv;
375
376     Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
377     Comb(int n) : Comb() {
378         init(n);
379     }
380
381     void init(int m) {
382         m = std::min(m, (int)Z::mod() - 1);
383         if (m <= n)
384             return;
385         _fac.resize(m + 1);
386         _invfac.resize(m + 1);
387         _inv.resize(m + 1);
388
389         for (int i = n + 1; i <= m; i++) {
390             _fac[i] = _fac[i - 1] * i;
391         }
392         _invfac[m] = _fac[m].inv();
393         for (int i = m; i > n; i--) {
394             _invfac[i - 1] = _invfac[i] * i;
395             _inv[i] = _invfac[i] * _fac[i - 1];
396         }
397         n = m;
398     }
399
400     Z fac(int m) {
401         if (m > n)
402             init(2 * m);
403         return _fac[m];
404     }
405
406     Z invfac(int m) {
407         if (m > n)
408             init(2 * m);
409         return _invfac[m];
410     }
411
412     Z inv(int m) {
413         if (m > n)
414             init(2 * m);
415         return _inv[m];
416     }
417
418     Z binom(int n, int m) {
419         if (n < m || m < 0)
420             return 0;
421         Z res = 1;
422         for (int i = 0; i < m; i++) {
423             res *= n - i;
424             res /= i + 1;
425         }
426         return res;
427     }
428 }
```

postpone

```

418     return 0;
419     return fac(n) * invfac(m) * invfac(n - m);
420 }
421 } comb;
422
423 template <int P = 998244353>
424 Poly<P> get(int n, int m) {
425     if (m == 0) {
426         return Poly(n + 1);
427     }
428     if (m % 2 == 1) {
429         auto f = get(n, m - 1);
430         Z p = 1;
431         for (int i = 0; i <= n; i++) {
432             f[n - i] += comb.binom(n, i) * p;
433             p *= m;
434         }
435         return f;
436     }
437     auto f = get(n, m / 2);
438     auto fm = f;
439     for (int i = 0; i <= n; i++) {
440         fm[i] *= comb.fac(i);
441     }
442     Poly pw(n + 1);
443     pw[0] = 1;
444     for (int i = 1; i <= n; i++) {
445         pw[i] = pw[i - 1] * (m / 2);
446     }
447     for (int i = 0; i <= n; i++) {
448         pw[i] *= comb.invfac(i);
449     }
450     fm = fm.mult(pw);
451     for (int i = 0; i <= n; i++) {
452         fm[i] *= comb.invfac(i);
453     }
454     return f + fm;
455 }
```

```

10    }
11    void orFwt(auto &a, int t) {
12        int n = a.size();
13        for (int i = 1; i < n; i *= 2) {
14            for (int s = 0; s < n; s++) {
15                if (~s & i) {
16                    a[s | i] += t * a[s];
17                }
18            }
19        }
20    }
21    void xorFwt(auto &a) {
22        int n = a.size();
23        for (int i = 1; i < n; i *= 2) {
24            for (int j = 0; j < n; j += 2 * i) {
25                for (int k = 0; k < i; k++) {
26                    auto u = a[j + k], v = a[i + j + k];
27                    a[j + k] = u + v;
28                    a[i + j + k] = u - v;
29                }
30            }
31        }
32    }
33    auto xorConv(auto a, auto b) {
34        int n = a.size();
35        xorFwt(a);
36        xorFwt(b);
37
38        for (int i = 0; i < n; i++) {
39            a[i] *= b[i];
40        }
41        xorFwt(a);
42        auto invn = Z(n).inv();
43        for (int i = 0; i < n; i++) {
44            a[i] *= invn;
45        }
46        // if not module:
47        // for (int i = 0; i < n; i++) {
48        //     a[i] >>= __lg(n);
49        // }
50        return move(a);
51    }
```

2.9.3 Fast Walsh Transform

```

1 void andFwt(auto &a, int t) {
2     int n = a.size();
3     for (int i = 1; i < n; i *= 2) {
4         for (int s = 0; s < n; s++) {
5             if (~s & i) {
6                 a[s] += t * a[s | i];
7             }
8         }
9     }
```

2.9.4 Sum of Subset

```

1 {
2     std::vector<int> f(1 << n);
3     // ...
4
5     // prefix
```

```

6   for (int i = 0; i < n; i++) {
7     for (int s = 0; s < 1 << n; s++) {
8       if (s >> i & 1) {
9         f[s] += f[s ^ (1 << i)];
10      }
11    }
12  }
13 // suffix
14 for (int i = 0; i < n; i++) {
15   for (int s = 0; s < 1 << n; s++) {
16     if (~s >> i & 1) {
17       f[s] += f[s ^ (1 << i)];
18     }
19   }
20 }
21 }
```

```

28   if (dep[x] < dep[y]) {
29     std::swap(x, y);
30   }
31   while (dep[x] > dep[y]) {
32     x = p[std::__lg(dep[x] - dep[y])][x];
33   }
34   if (x == y) {
35     return x;
36   }
37   for (int i = std::__lg(dep[x]); i ≥ 0; i--) {
38     if (p[i][x] ≠ p[i][y]) {
39       x = p[i][x];
40       y = p[i][y];
41     }
42   }
43   return p[0][x];
44 };
45 }
```

3 Graph

3.1 Tree

3.1.1 Lowest Common Ancestor

```

1 // Binary Lifting O(n log n) - O(log n)
2 {
3   const int logn = std::__lg(n);
4
5   std::vector<int> dep(n);
6   std::vector p(logn + 1, std::vector<int>(n));
7   auto dfs = [&](auto &self, int x) → void {
8     for (auto y : adj[x]) {
9       if (y == p[0][x]) {
10         continue;
11       }
12       p[0][y] = x;
13       dep[y] = dep[x] + 1;
14       self(self, y);
15     }
16   };
17
18   p[0][s] = s;
19   dfs(dfs, s);
20
21   for (int j = 0; j < logn; j++) {
22     for (int i = 0; i < n; i++) {
23       p[j + 1][i] = p[j][p[j][i]];
24     }
25   }
26
27   auto lca = [&](int x, int y) {
```

3.1.2 Heavy-Light Decomposition

```

1 struct HLD {
2   int n;
3   std::vector<int> siz, top, dep, parent, in, out, seq;
4   std::vector<std::vector<int>> adj;
5   int cur;
6
7   HLD() {}
8   HLD(int n) {
9     init(n);
10 }
11
12 void init(int n) {
13   this→n = n;
14   siz.resize(n);
15   top.resize(n);
16   dep.resize(n);
17   parent.resize(n);
18   in.resize(n);
19   out.resize(n);
20   seq.resize(n);
21   cur = 0;
22   adj.assign(n, {});
23 }
24
25 void addEdge(int u, int v) {
26   adj[u].push_back(v);
27   adj[v].push_back(u);
28 }
29
30 void work(int root = 0) {
31   top[root] = root;
32   dep[root] = 0;
```

postpone

```

30     parent[root] = -1;
31     dfs1(root);
32     dfs2(root);
33 }
34 void dfs1(int u) {
35     if (parent[u] != -1) {
36         adj[u].erase(find(adj[u].begin(), adj[u].end(), parent[u]));
37     }
38
39     siz[u] = 1;
40     for (auto &v : adj[u]) {
41         parent[v] = u;
42         dep[v] = dep[u] + 1;
43         dfs1(v);
44         siz[u] += siz[v];
45         if (siz[v] > siz[adj[u][0]]) {
46             std::swap(v, adj[u][0]);
47         }
48     }
49 }
50 void dfs2(int u) {
51     in[u] = cur++;
52     seq[in[u]] = u;
53     for (auto v : adj[u]) {
54         top[v] = v == adj[u][0] ? top[u] : v;
55         dfs2(v);
56     }
57     out[u] = cur;
58 }
59 int lca(int u, int v) {
60     while (top[u] != top[v]) {
61         if (dep[top[u]] > dep[top[v]]) {
62             u = parent[top[u]];
63         } else {
64             v = parent[top[v]];
65         }
66     }
67     return dep[u] < dep[v] ? u : v;
68 }
69 int dist(int u, int v) {
70     return dep[u] + dep[v] - 2 * dep[lca(u, v)];
71 }
72 bool isAncestor(int fa, int son) {
73     return in[fa] <= in[son] && in[son] < out[fa];
74 }
75 // [u, v]
76 auto getPath(int u, int v) {
77     std::vector<std::pair<int, int>> ret;
78     while (top[u] != top[v]) {
79         if (dep[top[u]] > dep[top[v]]) {
80             ret.push_back({in[top[u]], in[u]});
81             u = parent[top[u]];
82         } else {
83             ret.push_back({in[top[v]], in[v]});
84             v = parent[top[v]];
85         }
86         if (dep[u] > dep[v]) {
87             ret.push_back({in[v], in[u]});
88         } else {
89             ret.push_back({in[u], in[v]});
90         }
91     }
92     return std::move(ret);
93 }
94 // [u, v]
95 std::pair<int, int> getTree(int u) {
96     return pair(in[u], out[u]);
97 }
98 int jump(int u, int k) {
99     if (dep[u] < k) {
100         return -1;
101     }
102     int d = dep[u] - k;
103     while (dep[top[u]] > d) {
104         u = parent[top[u]];
105     }
106     return seq[in[u] - dep[u] + d];
107 }
108 int rootedParent(int u, int v) {
109     std::swap(u, v);
110     if (u == v) {
111         return u;
112     }
113     if (!isAncestor(u, v)) {
114         return parent[u];
115     }
116     auto it = std::upper_bound(adj[u].begin(), adj[u].end(), v, [&](int x
117 , int y) {
118         return in[x] < in[y];
119     }) - 1;
120     return *it;
121 }
122 int rootedSize(int u, int v) {
123     if (u == v) {
124         return n;
125     }
126     if (!isAncestor(v, u)) {
127         return siz[v];
128     }
129     return n - siz[rootedParent(u, v)];
130 }
131 int rootedLca(int a, int b, int c) {

```

```

131     return lca(a, b) ^ lca(b, c) ^ lca(c, a);
132 }
133 };

```

3.1.3 DSU on Tree

```

1 {
2     int n;
3     std::vector<std::vector<int>> adj(n);
4     // ...
5     std::vector<int> siz(n);
6     [&](this auto &&self, int x, int p) → void {
7         if (p ≠ -1) {
8             adj[x].erase(find(adj[x].begin(), adj[x].end(), p));
9         }
10        siz[x] = 1;
11        // &y
12        for (auto &y : adj[x]) {
13            self(y, x);
14            siz[x] += siz[y];
15            if (siz[y] > siz[adj[x][0]]) {
16                swap(adj[x][0], y);
17            }
18        }
19    } (0, -1);
20
21    auto addv = [&](int color, int t) {
22        // freq[color] += t
23    };
24    auto add = [&](auto &&self, int x, int t) → void {
25        // addv(color[x], t);
26        for (auto y : adj[x]) {
27            self(self, y, t);
28        }
29    };
30    auto dfs = [&](auto &&self, int x) → void {
31        for (auto y : adj[x]) {
32            if (y ≠ adj[x][0]) {
33                self(self, y);
34                add(add, y, -1);
35            }
36        }
37        if (not adj[x].empty()) {
38            self(self, adj[x][0]);
39            for (auto y : adj[x]) {
40                if (y ≠ adj[x][0]) {
41                    add(add, y, 1);
42                }
43            }
44        }

```

```

45         addv(color[x], 1);
46         // ans[x] = ?
47     };
48     // dfs(0);
49 }

```

3.2 Connectivity

3.2.1 Strongly Connected Component

```

1 struct SCC {
2     int n;
3     std::vector<std::vector<int>> adj;
4     std::vector<int> stk;
5     std::vector<int> dfn, low, bel;
6     int cur, cnt;
7
8     SCC() {}
9     SCC(int n) {
10         init(n);
11     }
12     void init(int n) {
13         this→n = n;
14         adj.assign(n, {});
15         dfn.assign(n, -1);
16         low.resize(n);
17         bel.assign(n, -1);
18         stk.clear();
19         cur = cnt = 0;
20     }
21     void addEdge(int u, int v) {
22         adj[u].push_back(v);
23     }
24     void dfs(int x) {
25         dfn[x] = low[x] = cur++;
26         stk.push_back(x);
27         for (auto y : adj[x]) {
28             if (dfn[y] == -1) {
29                 dfs(y);
30                 low[x] = std::min(low[x], low[y]);
31             } else if (bel[y] == -1) {
32                 low[x] = std::min(low[x], dfn[y]);
33             }
34         }
35         if (dfn[x] == low[x]) {
36             int y;
37             do {
38                 y = stk.back();
39                 bel[y] = cnt;
40             }
41             while (y != x);
42         }
43     }
44 }

```

postpone

```

41     } while (y != x);
42     cnt++;
43   }
44   std::vector<int> work() {
45     for (int i = 0; i < n; i++) {
46       if (dfn[i] == -1) {
47         dfs(i);
48       }
49     }
50   }
51   return bel;
52 }
53 };

```

3.2.2 Block Cut Tree

```

1 struct BlockCutTree {
2   int n;
3   std::vector<std::vector<int>> adj;
4   std::vector<int> dfn, low, stk;
5   int cnt, cur;
6   std::vector<std::pair<int, int>> edges;
7   BlockCutTree() {}
8   BlockCutTree(int n) {
9     init(n);
10  }
11  void init(int n) {
12    this->n = n;
13    adj.assign(n, {});
14    dfn.assign(n, -1);
15    low.resize(n);
16    stk.clear();
17    cnt = cur = 0;
18    edges.clear();
19  }
20  void addEdge(int u, int v) {
21    adj[u].push_back(v);
22    adj[v].push_back(u);
23  }
24  void dfs(int x) {
25    stk.push_back(x);
26    dfn[x] = low[x] = cur++;
27    for (auto y : adj[x]) {
28      if (dfn[y] == -1) {
29        dfs(y);
30        low[x] = std::min(low[x], low[y]);
31        if (low[y] == dfn[x]) {
32          int v;
33          do {
34            v = stk.back();

```

```

35          stk.pop_back();
36          edges.emplace_back(n + cnt, v);
37        } while (v != y);
38        edges.emplace_back(x, n + cnt);
39        cnt++;
40      }
41    } else {
42      low[x] = std::min(low[x], dfn[y]);
43    }
44  }
45  std::pair<int, std::vector<std::pair<int, int>>> work() {
46    for (int i = 0; i < n; i++) {
47      if (dfn[i] == -1) {
48        stk.clear();
49        dfs(i);
50      }
51    }
52    return {cnt, edges};
53  }
54 };

```

3.3 Two Sat

```

1 struct TwoSat {
2   int n;
3   std::vector<std::vector<int>> e;
4   std::vector<bool> ans;
5   TwoSat(int n) : n(n), e(2 * n), ans(n) {}
6   void addClause(int u, bool f, int v, bool g) {
7     e[2 * u + !f].push_back(2 * v + g);
8     e[2 * v + !g].push_back(2 * u + f);
9   }
10  bool satisfiable() {
11    std::vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 * n, -1);
12    std::vector<int> stk;
13    int now = 0, cnt = 0;
14    std::function<void(int)> tarjan = [&](int u) {
15      stk.push_back(u);
16      dfn[u] = low[u] = now++;
17      for (auto v : e[u]) {
18        if (dfn[v] == -1) {
19          tarjan(v);
20          low[u] = std::min(low[u], low[v]);
21        } else if (id[v] == -1) {
22          low[u] = std::min(low[u], dfn[v]);
23        }
24      }
25      if (dfn[u] == low[u]) {
26        int v;

```

```

27     do {
28         v = stk.back();
29         stk.pop_back();
30         id[v] = cnt;
31     } while (v != u);
32     ++cnt;
33 }
34 ;
35 for (int i = 0; i < 2 * n; ++i) if (dfn[i] == -1) tarjan(i);
36 for (int i = 0; i < n; ++i) {
37     if (id[2 * i] == id[2 * i + 1]) return false;
38     ans[i] = id[2 * i] > id[2 * i + 1];
39 }
40     return true;
41 }
42 std::vector<bool> answer() { return ans; }
43 };

```

4 String

4.1 String Hash

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 using i64 = long long;
5 using Hash = array<int, 2>;
6
7 constexpr int P = 998244353;
8 constexpr Hash base = {567, 1234}; // any
9
10 int main() {
11     ios::sync_with_stdio(false);
12     cin.tie(nullptr);
13
14     string s;
15     cin >> s;
16
17     const int n = s.size();
18
19     vector<Hash> h(n + 1), p(n + 1);
20     p[0] = {1, 1};
21     for (int j = 0; j < 2; j++) {
22         for (int i = 0; i < n; i++) {
23             h[i + 1][j] = (i64(base[j]) * h[i][j] + s[i]) % P;
24             p[i + 1][j] = (i64(base[j]) * p[i][j]) % P;
25         }
26     }
27 }

```

```

28     auto get = [&](int l, int r) {
29         Hash res{};
30         for (int i = 0; i < 2; i++) {
31             res[i] = (h[r][i] + 1ll * (P - h[l][i]) * p[r - l][i]) % P;
32         }
33         return res;
34     };
35
36     return 0;
37 }

```

4.2 KMP

```

1 std::vector<int> kmp(std::string s) {
2     int n = s.size();
3     std::vector<int> f(n + 1);
4     for (int i = 1, j = 0; i < n; i++) {
5         while (j && s[i] != s[j]) {
6             j = f[j];
7         }
8         j += (s[i] == s[j]);
9         f[i + 1] = j;
10    }
11    return f;
12 }

```

4.3 Z-function

```

1 std::vector<int> Z(std::string s) {
2     int n = s.size();
3     std::vector<int> z(n + 1);
4     z[0] = n;
5     for (int i = 1, j = 1; i < n; i++) {
6         z[i] = std::max(0, std::min(j + z[j] - i, z[i - j]));
7         while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
8             z[i]++;
9         }
10        if (i + z[i] > j + z[j]) {
11            j = i;
12        }
13    }
14    return z;
15 }

```

4.4 AhoCorasick

```

1 struct AhoCorasick {

```

postpone

```

2 static constexpr int ALPHABET = 26;
3 struct Node {
4     int len;
5     int link;
6     std::array<int, ALPHABET> next;
7     Node() : len{0}, link{0}, next{} {}
8 };
9
10 std::vector<Node> t;
11
12 AhoCorasick() {
13     init();
14 }
15
16 void init() {
17     t.assign(2, Node());
18     t[0].next.fill(1);
19     t[0].len = -1;
20 }
21
22 int newNode() {
23     t.emplace_back();
24     return t.size() - 1;
25 }
26
27 int add(const std::string &a) {
28     int p = 1;
29     for (auto c : a) {
30         int x = c - 'a';
31         if (t[p].next[x] == 0) {
32             t[p].next[x] = newNode();
33             t[t[p].next[x]].len = t[p].len + 1;
34         }
35         p = t[p].next[x];
36     }
37     return p;
38 }
39
40 void work() {
41     std::queue<int> q;
42     q.push(1);
43
44     while (!q.empty()) {
45         int x = q.front();
46         q.pop();
47
48         for (int i = 0; i < ALPHABET; i++) {
49             if (t[x].next[i] == 0) {
50                 t[x].next[i] = t[t[x].link].next[i];
51             } else {
52                 t[t[x].next[i]].link = t[t[x].link].next[i];
53             }
54         }
55     }
56 }
57
58 int next(int p, int x) {
59     return t[p].next[x];
60 }
61
62 int link(int p) {
63     return t[p].link;
64 }
65
66 int len(int p) {
67     return t[p].len;
68 }
69
70 int size() {
71     return t.size();
72 }
73 }
74 
```

4.5 Suffix Array

```

1 struct SuffixArray {
2     int n;
3     std::vector<int> sa, rk, lc;
4     SuffixArray(const std::string &s) {
5         n = s.length();
6         sa.resize(n);
7         lc.resize(n - 1);
8         rk.resize(n);
9         std::iota(sa.begin(), sa.end(), 0);
10        std::sort(sa.begin(), sa.end(),
11                  [&](int a, int b) {
12                      return s[a] < s[b];
13                  });
14        rk[sa[0]] = 0;
15        for (int i = 1; i < n; i++) {
16            rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
17        }
18        int k = 1;
19        std::vector<int> tmp, cnt(n);
20        tmp.reserve(n);
21        while (rk[sa[n - 1]] < n - 1) {
22            tmp.clear();
23            for (int i = 0; i < k; i++) {
24                tmp.push_back(n - k + i);
25            }
26        }
27    }
28 }
```

```

26     for (auto i : sa) {
27         if (i >= k) {
28             tmp.push_back(i - k);
29         }
30     }
31     std::fill(cnt.begin(), cnt.end(), 0);
32     for (int i = 0; i < n; i++) {
33         cnt[rk[i]]++;
34     }
35     for (int i = 1; i < n; i++) {
36         cnt[i] += cnt[i - 1];
37     }
38     for (int i = n - 1; i >= 0; i--) {
39         sa[~cnt[rk[tmp[i]]]] = tmp[i];
40     }
41     std::swap(rk, tmp);
42     rk[sa[0]] = 0;
43     for (int i = 1; i < n; i++) {
44         rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i - 1]] < tmp[sa[i]] || 
45             sa[i - 1] + k == n || tmp[sa[i - 1] + k] < tmp[sa[i] + k]);
46     }
47     k *= 2;
48 }
49 for (int i = 0, j = 0; i < n; i++) {
50     if (rk[i] == 0) {
51         j = 0;
52     } else {
53         for (j -= (j > 0); i + j < n && sa[rk[i] - 1] + j < n && s[i
54             + j] == s[sa[rk[i] - 1] + j]; j++)
55         ;
56         lc[rk[i] - 1] = j;
57     }
58 }

```

```

13 void init() {
14     t.assign(2, Node());
15     t[0].next.fill(1);
16     t[0].len = -1;
17 }
18 int newNode() {
19     t.emplace_back();
20     return t.size() - 1;
21 }
22 int extend(int p, int c) {
23     if (t[p].next[c]) {
24         int q = t[p].next[c];
25         if (t[q].len == t[p].len + 1) {
26             return q;
27         }
28         int r = newNode();
29         t[r].len = t[p].len + 1;
30         t[r].link = t[q].link;
31         t[r].next = t[q].next;
32         t[q].link = r;
33         while (t[p].next[c] == q) {
34             t[p].next[c] = r;
35             p = t[p].link;
36         }
37         return r;
38     }
39     int cur = newNode();
40     t[cur].len = t[p].len + 1;
41     while (!t[p].next[c]) {
42         t[p].next[c] = cur;
43         p = t[p].link;
44     }
45     t[cur].link = extend(p, c);
46     return cur;
47 }
48 int extend(int p, char c, char offset = 'a') {
49     return extend(p, c - offset);
50 }
51
52 int next(int p, int x) {
53     return t[p].next[x];
54 }
55
56 int next(int p, char c, char offset = 'a') {
57     return next(p, c - 'a');
58 }
59
60 int link(int p) {
61     return t[p].link;
62 }
63

```

4.6 Suffix Automaton

```

1 struct SAM {
2     static constexpr int ALPHABET_SIZE = 26;
3     struct Node {
4         int len;
5         int link;
6         std::array<int, ALPHABET_SIZE> next;
7         Node() : len{}, link{}, next{} {}
8     };
9     std::vector<Node> t;
10    SAM() {
11        init();
12    }

```

postpone

```

64     int len(int p) {
65         return t[p].len;
66     }
67
68     int size() {
69         return t.size();
70     }
71 };

```

4.7 Palindromic Tree

```

1 struct PAM {
2     static constexpr int ALPHABET_SIZE = 26;
3     struct Node {
4         int len;
5         int link;
6         int cnt;
7         std::array<int, ALPHABET_SIZE> next;
8         Node() : len{}, link{}, cnt{}, next{} {}
9     };
10    std::vector<Node> t;
11    int suff;
12    std::string s;
13    PAM() {
14        init();
15    }
16    void init() {
17        t.assign(2, Node());
18        t[0].len = -1;
19        suff = 1;
20        s.clear();
21    }
22    int newNode() {
23        t.emplace_back();
24        return t.size() - 1;
25    }
26    bool add(char c) {
27        int pos = s.size();
28        s += c;
29        int let = c - 'a';
30        int cur = suff, curlen = 0;
31        while (true) {
32            curlen = t[cur].len;
33            if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
34                break;
35            }
36            cur = t[cur].link;
37        }
38        if (t[cur].next[let]) {
39            suff = t[cur].next[let];

```

```

40             return false;
41         }
42         int num = newNode();
43         suff = num;
44         t[num].len = t[cur].len + 2;
45         t[cur].next[let] = num;
46         if (t[num].len == 1) {
47             t[num].link = 1;
48             t[num].cnt = 1;
49             return true;
50         }
51         while (true) {
52             cur = t[cur].link;
53             curlen = t[cur].len;
54             if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
55                 t[num].link = t[cur].next[let];
56                 break;
57             }
58         }
59         t[num].cnt = 1 + t[t[num].link].cnt;
60         return true;
61     }
62     int next(int p, int x) {
63         return t[p].next[x];
64     }
65     int link(int p) {
66         return t[p].link;
67     }
68     int len(int p) {
69         return t[p].len;
70     }
71     int size() {
72         return t.size();
73     }
74 };

```

5 Geometry

```

1 template<class T>
2 struct Point {
3     T x;
4     T y;
5     Point(const T &x_ = 0, const T &y_ = 0) : x(x_), y(y_) {}
6
7     template<class U>
8     operator Point<U>() {
9         return Point<U>(U(x), U(y));
10    }
11    Point &operator+=(const Point &p) & {

```

```

12     x += p.x;
13     y += p.y;
14     return *this;
15 }
16 Point &operator=(const Point &p) & {
17     x -= p.x;
18     y -= p.y;
19     return *this;
20 }
21 Point &operator*=(const T &v) & {
22     x *= v;
23     y *= v;
24     return *this;
25 }
26 Point &operator/=(const T &v) & {
27     x /= v;
28     y /= v;
29     return *this;
30 }
31 Point operator-() const {
32     return Point(-x, -y);
33 }
34 friend Point operator+(Point a, const Point &b) {
35     return a += b;
36 }
37 friend Point operator-(Point a, const Point &b) {
38     return a -= b;
39 }
40 friend Point operator*(Point a, const T &b) {
41     return a *= b;
42 }
43 friend Point operator/(Point a, const T &b) {
44     return a /= b;
45 }
46 friend Point operator*(const T &a, Point b) {
47     return b *= a;
48 }
49 friend bool operator==(const Point &a, const Point &b) {
50     return a.x == b.x && a.y == b.y;
51 }
52 friend std::istream &operator>>(std::istream &is, Point &p) {
53     return is >> p.x >> p.y;
54 }
55 friend std::ostream &operator<<(std::ostream &os, const Point &p) {
56     return os << "(" << p.x << ", " << p.y << ")";
57 }
58 };
59 template<class T>
60 struct Line {
61     Point<T> a;

```

```

63     Point<T> b;
64     Line(const Point<T> &a_ = Point<T>(), const Point<T> &b_ = Point<T>()) :
65         a(a_), b(b_) {}
66 }
67 template<class T>
68 T dot(const Point<T> &a, const Point<T> &b) {
69     return a.x * b.x + a.y * b.y;
70 }
71 template<class T>
72 T cross(const Point<T> &a, const Point<T> &b) {
73     return a.x * b.y - a.y * b.x;
74 }
75 }
76 template<class T>
77 T square(const Point<T> &p) {
78     return dot(p, p);
79 }
80 }
81 template<class T>
82 double length(const Point<T> &p) {
83     return std::sqrt(square(p));
84 }
85 }
86 template<class T>
87 double length(const Line<T> &l) {
88     return length(l.a - l.b);
89 }
90 }
91 template<class T>
92 Point<T> normalize(const Point<T> &p) {
93     return p / length(p);
94 }
95 }
96 template<class T>
97 bool parallel(const Line<T> &l1, const Line<T> &l2) {
98     return cross(l1.b - l1.a, l2.b - l2.a) == 0;
99 }
100 }
101 template<class T>
102 double distance(const Point<T> &a, const Point<T> &b) {
103     return length(a - b);
104 }
105 }
106 template<class T>
107 double distancePL(const Point<T> &p, const Line<T> &l) {
108     return std::abs(cross(l.a - l.b, l.a - p)) / length(l);
109 }
110 }
111 template<class T>

```

postpone

```

113 double distancePS(const Point<T> &p, const Line<T> &l) {
114     if (dot(p - l.a, l.b - l.a) < 0) {
115         return distance(p, l.a);
116     }
117     if (dot(p - l.b, l.a - l.b) < 0) {
118         return distance(p, l.b);
119     }
120     return distancePL(p, l);
121 }
122
123 template<class T>
124 Point<T> rotate(const Point<T> &a) {
125     return Point(-a.y, a.x);
126 }
127
128 template<class T>
129 int sgn(const Point<T> &a) {
130     return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
131 }
132
133 template<class T>
134 bool pointOnLineLeft(const Point<T> &p, const Line<T> &l) {
135     return cross(l.b - l.a, p - l.a) > 0;
136 }
137
138 template<class T>
139 Point<T> lineIntersection(const Line<T> &l1, const Line<T> &l2) {
140     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2
141         .b - l2.a, l1.a - l1.b));
142
143 template<class T>
144 bool pointOnSegment(const Point<T> &p, const Line<T> &l) {
145     return cross(p - l.a, l.b - l.a) == 0 && std::min(l.a.x, l.b.x) <= p.x &&
146         p.x <= std::max(l.a.x, l.b.x)
147         && std::min(l.a.y, l.b.y) <= p.y && p.y <= std::max(l.a.y, l.b.y);
148
149 template<class T>
150 bool pointInPolygon(const Point<T> &a, const std::vector<Point<T>> &p) {
151     int n = p.size();
152     for (int i = 0; i < n; i++) {
153         if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
154             return true;
155         }
156     }
157
158     int t = 0;
159     for (int i = 0; i < n; i++) {
160         auto u = p[i];
161         auto v = p[(i + 1) % n];
162         if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
163             t += 1;
164         }
165         if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
166             t -= 1;
167         }
168     }
169     return t == 1;
170 }
171
172 // 0 : not intersect
173 // 1 : strictly intersect
174 // 2 : overlap
175 // 3 : intersect at endpoint
176 template<class T>
177 std::tuple<int, Point<T>, Point<T>> segmentIntersection(const Line<T> &l1,
178     const Line<T> &l2) {
179     if (std::max(l1.a.x, l1.b.x) < std::min(l2.a.x, l2.b.x)) {
180         return {0, Point<T>(), Point<T>()};
181     }
182     if (std::min(l1.a.x, l1.b.x) > std::max(l2.a.x, l2.b.x)) {
183         return {0, Point<T>(), Point<T>()};
184     }
185     if (std::max(l1.a.y, l1.b.y) < std::min(l2.a.y, l2.b.y)) {
186         return {0, Point<T>(), Point<T>()};
187     }
188     if (std::min(l1.a.y, l1.b.y) > std::max(l2.a.y, l2.b.y)) {
189         return {0, Point<T>(), Point<T>()};
190     }
191     if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
192         if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
193             return {0, Point<T>(), Point<T>()};
194         } else {
195             auto maxx1 = std::max(l1.a.x, l1.b.x);
196             auto minx1 = std::min(l1.a.x, l1.b.x);
197             auto maxy1 = std::max(l1.a.y, l1.b.y);
198             auto miny1 = std::min(l1.a.y, l1.b.y);
199             auto maxx2 = std::max(l2.a.x, l2.b.x);
200             auto minx2 = std::min(l2.a.x, l2.b.x);
201             auto maxy2 = std::max(l2.a.y, l2.b.y);
202             auto miny2 = std::min(l2.a.y, l2.b.y);
203             Point<T> p1(std::max(minx1, minx2), std::max(miny1, miny2));
204             Point<T> p2(std::min(maxx1, maxx2), std::min(maxy1, maxy2));
205             if (!pointOnSegment(p1, l1)) {
206                 std::swap(p1.y, p2.y);
207             }
208             if (p1 == p2) {
209                 return {3, p1, p2};
210             } else {
211                 return {2, p1, p2};
212             }
213         }
214     }
215 }

```

```

212     }
213 }
214
215 auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
216 auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
217 auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
218 auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);
219
220 if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0)
221   || (cp3 < 0 && cp4 < 0)) {
222   return {0, Point<T>(), Point<T>()};
223 }
224
225 Point p = lineIntersection(l1, l2);
226 if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
227   return {1, p, p};
228 } else {
229   return {3, p, p};
230 }
231
232 template<class T>
233 double distanceSS(const Line<T> &l1, const Line<T> &l2) {
234   if (std::get<0>(segmentIntersection(l1, l2)) != 0) {
235     return 0.0;
236   }
237   return std::min({distancePS(l1.a, l2), distancePS(l1.b, l2), distancePS(
238     l2.a, l1), distancePS(l2.b, l1)});
239 }
240
241 template<class T>
242 bool segmentInPolygon(const Line<T> &l, const std::vector<Point<T>> &p) {
243   int n = p.size();
244   if (!pointInPolygon(l.a, p)) {
245     return false;
246   }
247   if (!pointInPolygon(l.b, p)) {
248     return false;
249   }
250   for (int i = 0; i < n; i++) {
251     auto u = p[i];
252     auto v = p[(i + 1) % n];
253     auto w = p[(i + 2) % n];
254     auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
255
256     if (t == 1) {
257       return false;
258     }
259     if (t == 0) {
260       continue;
261     }
262   }
263 }
```

```

261   if (t == 2) {
262     if (pointOnSegment(v, l) && v != l.a && v != l.b) {
263       if (cross(v - u, w - v) > 0) {
264         return false;
265       }
266     }
267   } else {
268     if (p1 != u && p1 != v) {
269       if (pointOnLineLeft(l.a, Line(v, u))
270         || pointOnLineLeft(l.b, Line(v, u))) {
271         return false;
272       }
273     } else if (p1 == v) {
274       if (l.a == v) {
275         if (pointOnLineLeft(u, l)) {
276           if (pointOnLineLeft(w, l)
277             && pointOnLineLeft(w, Line(u, v))) {
278             return false;
279           }
280         } else {
281           if (pointOnLineLeft(w, l)
282             || pointOnLineLeft(w, Line(u, v))) {
283               return false;
284             }
285         }
286       } else if (l.b == v) {
287         if (pointOnLineLeft(u, Line(l.b, l.a))) {
288           if (pointOnLineLeft(w, Line(l.b, l.a))
289             && pointOnLineLeft(w, Line(u, v))) {
290               return false;
291             }
292         } else {
293           if (pointOnLineLeft(w, Line(l.b, l.a))
294             || pointOnLineLeft(w, Line(u, v))) {
295               return false;
296             }
297         }
298       } else {
299         if (pointOnLineLeft(u, l)) {
300           if (pointOnLineLeft(w, Line(l.b, l.a))
301             || pointOnLineLeft(w, Line(u, v))) {
302               return false;
303             }
304         } else {
305           if (pointOnLineLeft(w, l)
306             || pointOnLineLeft(w, Line(u, v))) {
307               return false;
308             }
309         }
310       }
311     }
312   }
313 }
```

postpone

```

312     }
313 }
314     return true;
315 }
316
317 template<class T>
318 std::vector<Point<T>> hp(std::vector<Line<T>> lines) {
319     std::sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
320         auto d1 = l1.b - l1.a;
321         auto d2 = l2.b - l2.a;
322
323         if (sgn(d1) ≠ sgn(d2)) {
324             return sgn(d1) = 1;
325         }
326
327         return cross(d1, d2) > 0;
328     });
329
330     std::deque<Line<T>> ls;
331     std::deque<Point<T>> ps;
332     for (auto l : lines) {
333         if (ls.empty()) {
334             ls.push_back(l);
335             continue;
336         }
337
338         while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
339             ps.pop_back();
340             ls.pop_back();
341         }
342
343         while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
344             ps.pop_front();
345             ls.pop_front();
346         }
347
348         if (cross(l.b - l.a, ls.back().b - ls.back().a) = 0) {
349             if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
350
351                 if (!pointOnLineLeft(ls.back().a, l)) {
352                     assert(ls.size() = 1);
353                     ls[0] = l;
354                 }
355                 continue;
356             }
357             return {};
358         }
359
360         ps.push_back(lineIntersection(ls.back(), l));
361         ls.push_back(l);
362     }

```

```

363
364     while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
365         ps.pop_back();
366         ls.pop_back();
367     }
368     if (ls.size() ≤ 2) {
369         return {};
370     }
371     ps.push_back(lineIntersection(ls[0], ls.back()));
372
373     return std::vector(ps.begin(), ps.end());
374 }
375
376 using P = Point<double>;

```

6 Util

6.1 Mod Integer

```

1 constexpr int P = 998244353;
2 using i64 = long long;
3 // assume -P ≤ x < 2P
4 int norm(int x) {
5     if (x < 0) {
6         x += P;
7     }
8     if (x ≥ P) {
9         x -= P;
10    }
11    return x;
12 }
13 template<class T>
14 T power(T a, i64 b) {
15     T res = 1;
16     for (; b; b ≈ 2, a *= a) {
17         if (b % 2) {
18             res *= a;
19         }
20     }
21     return res;
22 }
23 struct Z {
24     int x;
25     Z(int x = 0) : x(norm(x)) {}
26     Z(i64 x) : x(norm(x % P)) {}
27     int val() const {
28         return x;
29     }
30     Z operator-() const {

```

```

31     return Z(norm(P - x));
32 }
33 Z inv() const {
34     assert(x != 0);
35     return power(*this, P - 2);
36 }
37 Z &operator*=(const Z &rhs) {
38     x = i64(x) * rhs.x % P;
39     return *this;
40 }
41 Z &operator+=(const Z &rhs) {
42     x = norm(x + rhs.x);
43     return *this;
44 }
45 Z &operator-=(const Z &rhs) {
46     x = norm(x - rhs.x);
47     return *this;
48 }
49 Z &operator/=(const Z &rhs) {
50     return *this *= rhs.inv();
51 }
52 friend Z operator*(const Z &lhs, const Z &rhs) {
53     Z res = lhs;
54     res *= rhs;
55     return res;
56 }
57 friend Z operator+(const Z &lhs, const Z &rhs) {
58     Z res = lhs;
59     res += rhs;
60     return res;
61 }
62 friend Z operator-(const Z &lhs, const Z &rhs) {
63     Z res = lhs;
64     res -= rhs;
65     return res;
66 }
67 friend Z operator/(const Z &lhs, const Z &rhs) {
68     Z res = lhs;
69     res /= rhs;
70     return res;
71 }
72 friend std::istream &operator>>(std::istream &is, Z &a) {
73     i64 v;
74     is >> v;
75     a = Z(v);
76     return is;
77 }
78 friend std::ostream &operator<<(std::ostream &os, const Z &a) {
79     return os << a.val();
80 }
81 };

```

```

1 template <class T>
2 constexpr T power(T a, u64 b, T res = 1) {
3     for (; b != 0; b /= 2, a *= a) {
4         if (b & 1) {
5             res *= a;
6         }
7     }
8     return res;
9 }
10 template <u32 P>
11 constexpr u32 mulMod(u32 a, u32 b) {
12     return u64(a) * b % P;
13 }
14 template <u64 P>
15 constexpr u64 mulMod(u64 a, u64 b) {
16     u64 res = a * b - u64(1.L * a * b / P - 0.5L) * P;
17     res %= P;
18     return res;
19 }
20 template <std::unsigned_integral U, U P>
21 struct ModIntBase {
22 public:
23     constexpr ModIntBase() : x(0) {}
24     template <std::unsigned_integral T>
25     constexpr ModIntBase(T x_) : x(x_ % mod()) {}
26     template <std::signed_integral T>
27     constexpr ModIntBase(T x_) {
28         using S = std::make_signed_t<U>;
29         S v = x_ % S(mod());
30         if (v < 0) {
31             v += mod();
32         }
33         x = v;
34     }
35     constexpr static U mod() {
36         return P;
37     }
38     constexpr U val() const {
39         return x;
40     }
41     constexpr ModIntBase operator-() const {
42         ModIntBase res;
43         res.x = (x == 0 ? 0 : mod() - x);
44         return res;
45     }
46     constexpr ModIntBase inv() const {
47         return power(*this, mod() - 2);
48     }
49     constexpr ModIntBase pow(u64 b) const {
50         return power(*this, b);
51     }
52     friend std::istream &operator>>(std::istream &is, ModIntBase &a) {
53         i64 v;
54         is >> v;
55         a = ModIntBase(v);
56         return is;
57     }
58     friend std::ostream &operator<<(std::ostream &os, const ModIntBase &a) {
59         return os << a.val();
60     }
61 };

```

```

51 }
52 constexpr ModIntBase &operator*=(const ModIntBase &rhs) & {
53     x = mulMod<mod()>(x, rhs.val());
54     return *this;
55 }
56 constexpr ModIntBase &operator+=(const ModIntBase &rhs) & {
57     x += rhs.val();
58     if (x ≥ mod()) {
59         x -= mod();
60     }
61     return *this;
62 }
63 constexpr ModIntBase &operator-=(const ModIntBase &rhs) & {
64     x -= rhs.val();
65     if (x ≥ mod()) {
66         x += mod();
67     }
68     return *this;
69 }
70 constexpr ModIntBase &operator/=(const ModIntBase &rhs) & {
71     return *this *= rhs.inv();
72 }

73 friend constexpr ModIntBase operator*(ModIntBase lhs, const ModIntBase &
74     rhs) {
75     lhs *= rhs;
76     return lhs;
77 }
78 friend constexpr ModIntBase operator+(ModIntBase lhs, const ModIntBase &
79     rhs) {
80     lhs += rhs;
81     return lhs;
82 }
83 friend constexpr ModIntBase operator-(ModIntBase lhs, const ModIntBase &
84     rhs) {
85     lhs -= rhs;
86     return lhs;
87 }
88 friend constexpr ModIntBase operator/(ModIntBase lhs, const ModIntBase &
89     rhs) {
90     lhs /= rhs;
91     return lhs;
92 }
93 friend constexpr std::istream &operator>>(std::istream &is, ModIntBase &a
94 ) {
95     i64 i;
96     is >> i;
97     a = i;
98     return is;
99 }
```

```

97     friend constexpr std::ostream &operator<<(std::ostream &os, const
98         ModIntBase &a) {
99         return os << a.val();
100    }
101    friend constexpr bool operator==(const ModIntBase &lhs, const ModIntBase
102        &rhs) {
103        return lhs.val() == rhs.val();
104    }
105    friend constexpr std::strong_ordering operator<=(const ModIntBase &lhs,
106        const ModIntBase &rhs) {
107        return lhs.val() ≤ rhs.val();
108    }
109 private:
110     U x;
111 };
112 template <u32 P>
113 using ModInt = ModIntBase<u32, P>;
114 template <u64 P>
115 using ModInt64 = ModIntBase<u64, P>;
116
117 template <int V, u32 P>
118 constexpr ModInt<P> CInv = ModInt<P>(V).inv();
119
120 using Z = ModInt<998244353>;
```

6.2 Fraction

```

1 struct Frac {
2     i64 num, den;
3     constexpr Frac(i64 x = 0) : Frac(x, 1) {}
4     constexpr Frac(i64 num, i64 den) : num{num}, den{den} {
5         norm();
6     }
7     constexpr void norm() {
8         if (den < 0) {
9             num = -num;
10            den = -den;
11        }
12    }
13    constexpr Frac operator-() const {
14        return Frac(-num, den);
15    }
16    constexpr Frac &operator+=(const Frac &rhs) {
17        num = num * rhs.den + rhs.num * den;
18        den *= rhs.den;
19        norm();
20        return *this;
21    }
22}
```

```

21 }
22 constexpr Frac &operator-=(const Frac &rhs) {
23     return *this += -rhs;
24 }
25 constexpr Frac &operator*=(const Frac &rhs) {
26     num *= rhs.num;
27     den *= rhs.den;
28     norm();
29     return *this;
30 }
31 constexpr Frac &operator/=(const Frac &rhs) {
32     num *= rhs.den;
33     den *= rhs.num;
34     norm();
35     return *this;
36 }
37 friend constexpr Frac operator+(const Frac &lhs, const Frac &rhs) {
38     Frac res = lhs;
39     res += rhs;
40     return res;
41 }
42 friend constexpr Frac operator-(const Frac &lhs, const Frac &rhs) {
43     Frac res = lhs;
44     res -= rhs;
45     return res;
46 }
47 friend constexpr Frac operator*(const Frac &lhs, const Frac &rhs) {
48     Frac res = lhs;
49     res *= rhs;
50     return res;
51 }
52 friend constexpr Frac operator/(const Frac &lhs, const Frac &rhs) {
53     Frac res = lhs;
54     res /= rhs;
55     return res;
56 }
57 friend constexpr bool operator<(const Frac &lhs, const Frac &rhs) {
58     return static_cast<__int128>(lhs.num) * rhs.den < static_cast<__int128>(rhs.num) * lhs.den;
59 }
60 friend constexpr bool operator>(const Frac &lhs, const Frac &rhs) {
61     return static_cast<__int128>(lhs.num) * rhs.den > static_cast<__int128>(rhs.num) * lhs.den;
62 }
63 friend std::ostream &operator<<(std::ostream &os, const Frac &rhs) {
64     os << rhs.num << "/" << rhs.den;
65 }
66 };

```

7 Tables

7.1 Constant

n	$\log_{10} n$	$n!$	$C(n, n/2)$	$\text{LCM}(1 \dots n)$	P_n
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	3628800	252	2520	42
11	1.04139269	39916800	462	27720	56
12	1.07918125	479001600	924	27720	77
15	1.17609126	1.31×10^{12}	6435	360360	176
20	1.30103000	2.43×10^{18}	184756	232792560	627
25	1.39794001	1.55×10^{25}	5200300	26771144400	1958
30	1.47712125	2.65×10^{32}	155117520	1.444×10^{14}	5604

$n \leq$	10	100	10^3	10^4	10^5	10^6
$\max \omega(n)$	2	3	4	5	6	7
$\max d(n)$	4	12	32	64	128	240
$\pi(n)$	4	25	168	1229	9592	78498
$n \leq$	10^7	10^8	10^9	10^{10}	10^{11}	10^{12}
$\max \omega(n)$	8	8	9	10	10	11
$\max d(n)$	448	768	1344	2304	4032	6720
$\pi(n)$	664579	5761455	5.08×10^7	4.55×10^8	4.12×10^9	3.7×10^{10}
$n \leq$	10^{13}	10^{14}	10^{15}	10^{16}	10^{17}	10^{18}
$\max \omega(n)$	12	12	13	13	14	15
$\max d(n)$	10752	17280	26880	41472	64512	103680

Prime number theorem: $\pi(x) \sim x / \log(x)$