

ICPC Template Library

nopostpone | SHU-Think Carefully

July 5, 2026

v1.3 

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	Formulas	8
2.2	Exgcd	9
2.3	Phi	9
2.4	Sieve	9
2.5	Pollard's Rho	10
2.6	Combination	10
2.7	Linear Basis	10
2.8	Gaussian Elimination	11
2.9	Polynomial	12
2.9.1	Lagrange Interpolation	12
2.9.2	Number Theory Transform	12
2.9.3	Fast Walsh Transform	14
2.9.4	Sum of Subset	15
3	Graph	15
3.1	Tree	15
3.1.1	Lowest Common Ancestor	15
3.1.2	Heavy-Light Decomposition	15
3.1.3	DSU on Tree	17
3.2	Connectivity	17
3.2.1	Strongly Connected Component	17
3.2.2	Block Cut Tree	18
3.3	Two Sat	18

4	String	19
4.1	String Hash	19
4.2	KMP	19
4.3	Z-function	19
4.4	AhoCorasick	20
4.5	Suffix Array	20
4.6	Suffix Automaton	21
4.7	Palindromic Tree	22
5	Geometry	23
5.1	Vector2D	23
5.2	ConvexHull	25
6	Util	26
6.1	Mod Integer	26
6.2	Fraction	27
7	Tables	28
7.1	Constant	28

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, weight;
40
41     DSU () {}
42     DSU(int n) {
43         init(n);
44     }
45     void init(int n) {
46         f.resize(n);
47         std::iota(f.begin(), f.end(), 0);

```

```

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

```

```

99 std::vector<std::pair<int *, int>> his;
100 int n;
101 std::vector<int> f, g, bip;
102 DSU(int n_) : n(n_), f(n, -1), g(n), bip(n, 1) {}
103 std::pair<int, int> find(int x) {
104     if (f[x] < 0) {
105         return {x, 0};
106     }
107     auto [u, v] = find(f[x]);
108     return {u, v ^ g[x]};
109 }
110 void set(int &a, int b) {
111     his.emplace_back(&a, a);
112     a = b;
113 }
114 void merge(int a, int b, int &ans) {
115     auto [u, xa] = find(a);
116     auto [v, xb] = find(b);
117     int w = xa ^ xb ^ 1;
118     if (u == v) {
119         if (bip[u] && w) {
120             set(bip[u], 0);
121             ans--;
122         }
123         return;
124     }
125     if (f[u] > f[v]) {
126         std::swap(u, v);
127     }
128     ans -= bip[u];
129     ans -= bip[v];
130     set(bip[u], bip[u] && bip[v]);
131     set(f[u], f[u] + f[v]);
132     set(f[v], u);
133     set(g[v], w);
134     ans += bip[u];
135 }
136 int cur() {
137     return his.size();
138 }
139 void rollback(int t) {
140     while (his.size() > t) {
141         auto [ptr, old] = his.back();
142         *ptr = old;
143         his.pop_back();
144     }
145 }
146 };

```

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_,
9       v_)); }
10  template <class T>
11  LazySegmentTree(const std::vector<T> &data) { init(data); }
12
13  template <class T>
14  void init(const std::vector<T> &data) {
15      n = data.size();
16      info.assign(4 << std::__lg(n), {});
17      tag.assign(4 << std::__lg(n), {});
18
19      auto build = [&](auto self, int p, int l, int r) → void {
20          if (r - l == 1) {
21              info[p] = data[l];
22              return;
23          }
24          int m = (l + r) / 2;
25          self(self, 2 * p, l, m);
26          self(self, 2 * p + 1, m, r);
27          pull(p);
28      };
29      build(build, 1, 0, n);
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,
70          y);
71  }
72  Info rangeQuery(int l, int r) {
73      return rangeQuery(1, 0, n, l, r);
74  }
75  void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
76      if (l ≥ y || r ≤ x) {
77          return;
78      }
79      if (l ≥ x && r ≤ y) {
80          apply(p, v);
81          return;
82      }
83      int m = (l + r) / 2;
84      push(p);
85      rangeApply(2 * p, l, m, x, y, v);
86      rangeApply(2 * p + 1, m, r, x, y, v);
87      pull(p);
88  }
89  void rangeApply(int l, int r, const Tag &v) {
90      return rangeApply(1, 0, n, l, r, v);
91  }
92  template <class F>
93  int findFirst(int p, int l, int r, int x, int y, const F &pred) {
94      if (l ≥ y || r ≤ x) {
95          return -1;
96      }
97      if (l ≥ x && r ≤ y && !pred(info[p])) {
98          return -1;
99      }
100     if (r - l == 1) {
101         return l;
102     }
103     push(p);
104     int m = (l + r) / 2;
105     int res = findFirst(2 * p, l, m, x, y, pred);
106     if (res == -1) {
107         res = findFirst(2 * p + 1, m, r, x, y, pred);

```

```

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 }

```

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);
16        a.assign(m + 1, std::vector<T>(n));
17        a[0] = v;
18        for (int j = 0; j < m; j++) {
19            for (int i = 0; i + (2 << j) ≤ n; i++) {
20                a[j + 1][i] = F(a[j][i], a[j][i + (1 << j)]);
21            }
22        }
23    }
24    T operator()(int l, int r) const {
25        if (l ≥ r) {
26            return e;
27        } else {
28            const int k = std::__lg(r - l);
29            return F(a[k][l], a[k][r - (1 << k)]);
30        }
31    }
32 };
33
34 // O(n) - O(1)
35 // only for minmax
36 template<class T, T e, class Cmp = std::less<>>
37 struct SparseTable {
38     const Cmp cmp = Cmp();
39     static constexpr unsigned B = 64;
40     int n;
41     std::vector<std::vector<T>> a;
42     std::vector<T> pre, suf, ini;
43     std::vector<u64> stk;
44     SparseTable(const std::vector<T> &v = {}) {
45         init(v);
46     }
47    void init(const std::vector<T> &v) {
48        n = v.size();
49        if (n == 0) {
50            return;
51        }
52        pre = suf = ini = v;
53        stk.resize(n);
54        const int M = (n - 1) / B + 1;
55        const int lg = std::__lg(M);

```

```

56     a.assign(lg + 1, std::vector<T>(M));
57     for (int i = 0; i < M; i++) {
58         a[0][i] = v[i * B];
59         for (int j = 1; j < B && i * B + j < n; j++) {
60             a[0][i] = std::min(a[0][i], v[i * B + j], cmp);
61         }
62     }
63     for (int i = 1; i < n; i++) {
64         if (i % B) {
65             pre[i] = std::min(pre[i], pre[i - 1], cmp);
66         }
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::sort(ask.begin(), ask.end(),
48     [&](auto i, auto j) {
49     if (i[0] / B != j[0] / B)
50         return i[0] < j[0];
51     if (i[1] / B != j[1] / B)
52         return i[1] < j[1];
53     return (i[1] / B) & 1 ? i[2] < j[2] : i[2] > j[2];
54 });
55 auto add = [&](int c) {
56     // ...
57 };
58 auto del = [&](int c) {
59     // ...
60 };
61 auto modify = [&](int p, int l, int r) {
62     // if (l ≤ x and x < r) then del(ori) and add(cur)
63
64     // first modify : x → y
65     // second modify : y → x
66     // using swap
67 };
68 // [l, r)
69 int L = 0, R = 0, T = 0;
70 for (auto [l, r, t, i] : que) {
71     while (l < L) {
72         add(a[--L]);
73     }
74     while (R < r) {
75         add(a[R++]);
76     }
77     while (L < l) {
78         del(a[L++]);
79     }
80     while (R > r) {
81         del(a[--R]);
82     }
83     while (T < t) {
84         modify(T++, l, r);
85     }
86     while (T > t) {
87         modify(--T, l, r);
88     }
89     // ans[i] = ?
90 }
91 }

```

1.6 Cartesian Tree

```

1 // root = rangeMin
2 std::vector<int> lc(n, -1), rc(n, -1);
3 std::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 }
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) {

```

```

28         return;
29     }
30     if (p == nullptr) {
31         p = new Node();
32     }
33     int m = (l + r) / 2;
34     if (l >= x && r <= y) {
35         if (cmp(f(m), p->f(m))) {
36             std::swap(f, p->f);
37         }
38         if (r - l == 1) {
39             return;
40         }
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 Formulas

- 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

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

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

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

or

$f(n)$: select **at most** $n \leftrightarrow g(n)$: select exactly n

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

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

$f(x)$: select **at least** $x \leftrightarrow g(x)$: select exactly x

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

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

- Vandermonde Convolution

$$\sum_{i=0}^k \binom{n}{i} \binom{m}{k-i} = \binom{n+m}{k}$$

$$\sum_{i=-r}^s \binom{n}{r+i} \binom{m}{s-i} = \binom{n+m}{r+s}$$

$$\sum_{i=1}^n \binom{n}{i} \binom{n}{i-1} = \binom{2n}{n-1}$$

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

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

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;
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            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     }
31     if (d != n) {
32         return d;
33     }
34     m++;
35 }
36 }
37
38 std::vector<i64> factorize(i64 n) {
39     std::vector<i64> p;
40     auto dfs = [&](auto &&self, i64 n) → void {
41         if (isPrime(n)) {
42             p.push_back(n);
43             return;
44         }
45         i64 d = factor(n);
46         self(self, d);
47         self(self, n / d);
48     };
49     dfs(dfs, n);
50     std::sort(p.begin(), p.end());
51     return p;
52 }

```

2.6 Combination

```

1 {
2     std::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);
4     for (int i = 0; i < int(lhs.size()); ++i)
5         for (int j = 0; j < int(rhs.size()); ++j)
6             res[i + j] = (res[i + j] + 1ll * lhs[i] * rhs[j]) % P;
7     return res;
8 }

```

```

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 }

```

```

46 std::vector<double> gauss(std::vector<std::vector<double>>> a, std::vector<
47 double> b) {
48     int n = a.size();
49     for (int i = 0; i < n; ++i) {
50         double x = a[i][i];
51         for (int j = i; j < n; ++j) a[i][j] /= x;
52         b[i] /= x;
53         for (int j = 0; j < n; ++j) {
54             if (i == j) continue;
55             x = a[j][i];
56             for (int k = i; k < n; ++k) a[j][k] -= a[i][k] * x;
57         }
58     }
59     return b;
60 }

```

```

56         b[j] -= b[i] * x;
57     }
58 }
59 return b;
60 }

```

```

41     res = ((n - 1 - i) % 2 ? -1 : 1) * fac[i] * fac[n - 1 - i];
42
43     ans += res;
44 }
45 }

```

2.9 Polynomial

2.9.1 Lagrange Interpolation

```

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

```

2.9.2 Number Theory Transform

```

1 constexpr int P = 998244353;
2
3 int norm(int x) {
4     if (x ≥ P) {
5         x -= P;
6     }
7     if (x < 0) {
8         x += P;
9     }
10    return x;
11 }
12 int power(int a, i64 b) {
13     int res = 1;
14     for (; b; b /= 2, a = 1ll * a * a % P) {
15         if (b % 2) {
16             res = 1ll * res * a % P;
17         }
18     }
19     return res;
20 }
21
22 std::vector<int> rev, roots{0, 1};
23
24 void dft(std::vector<int> &a) {
25     int n = a.size();
26     if (int(rev.size()) != n) {
27         int k = __builtin_ctz(n) - 1;
28         rev.resize(n);
29         for (int i = 0; i < n; i++) {
30             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
31         }
32     }
33     for (int i = 0; i < n; i++) {
34         if (rev[i] < i) {
35             std::swap(a[i], a[rev[i]]);
36         }
37     }
38     if (roots.size() < n) {
39         int k = __builtin_ctz(roots.size());
40         roots.resize(n);
41         while ((1 << k) < n) {
42             int e = power(31, 1 << (__builtin_ctz(P - 1) - k - 1));

```

```

43     for (int i = 1 << (k - 1); i < (1 << k); i++) {
44         roots[2 * i] = roots[i];
45         roots[2 * i + 1] = 1LL * roots[i] * e % P;
46     }
47     k++;
48 }
49 }
50
51 for (int k = 1; k < n; k *= 2) {
52     for (int i = 0; i < n; i += 2 * k) {
53         for (int j = 0; j < k; j++) {
54             int u = a[i + j];
55             int v = 1LL * a[i + j + k] * roots[k + j] % P;
56             a[i + j] = (u + v) % P;
57             a[i + j + k] = (u - v) % P;
58         }
59     }
60 }
61 }
62
63 void idft(std::vector<int> &a) {
64     int n = a.size();
65     std::reverse(a.begin() + 1, a.end());
66     dft(a);
67     int inv = (1 - P) / n;
68     for (int i = 0; i < n; i++) {
69         a[i] = 1ll * a[i] * inv % P;
70     }
71 }
72
73 using Poly = std::vector<int>;
74
75 Poly shift(Poly a, int k) {
76     if (k ≥ 0) {
77         a.insert(a.begin(), k, 0);
78         return a;
79     } else if (a.size() ≤ -k) {
80         return Poly();
81     } else {
82         return Poly(a.begin() - k, a.end());
83     }
84 }
85
86 Poly trunc(Poly a, int k) {
87     k = std::min(k, (int)a.size());
88     return Poly(a.begin(), a.begin() + k);
89 }
90
91 Poly operator+(const Poly &a, const Poly &b) {
92     Poly res(std::max(a.size(), b.size()));
93     for (int i = 0; i < a.size(); i++) {
94         res[i] = a[i];

```

```

94     }
95     for (int i = 0; i < b.size(); i++) {
96         res[i] = norm(res[i] + b[i]);
97     }
98     return res;
99 }
100 Poly operator-(const Poly &a, const Poly &b) {
101     Poly res(std::max(a.size(), b.size()));
102     for (int i = 0; i < a.size(); i++) {
103         res[i] = a[i];
104     }
105     for (int i = 0; i < b.size(); i++) {
106         res[i] = norm(res[i] - b[i]);
107     }
108     return res;
109 }
110 Poly operator-(const Poly &a) {
111     Poly res(a.size());
112     for (auto &x : res) {
113         x = -x;
114     }
115     return res;
116 }
117 Poly operator*(Poly a, Poly b) {
118     if (a.empty() or b.empty()) {
119         return Poly();
120     }
121     if (a.size() < b.size()) {
122         std::swap(a, b);
123     }
124     int n = 1;
125     int tot = a.size() + b.size() - 1;
126     while (n < tot) {
127         n *= 2;
128     }
129     if (b.size() < 128) {
130         Poly c(tot);
131         for (int i = 0; i < a.size(); i++) {
132             for (int j = 0; j < b.size(); j++) {
133                 c[i + j] = norm(c[i + j] + 1ll * a[i] * b[j] % P);
134             }
135         }
136         return c;
137     }
138
139     a.resize(n);
140     b.resize(n);
141     dft(a);
142     dft(b);
143     for (int i = 0; i < n; i++) {
144         a[i] = 1ll * a[i] * b[i] % P;

```

```

145     }
146     idft(a);
147     a.resize(tot);
148     return std::move(a);
149 }
150
151 Poly operator*(Poly a, int b) {
152     for (int i = 0; i < a.size(); i++) {
153         a[i] = 1ll * a[i] * b % P;
154     }
155     return a;
156 }
157 Poly operator/(Poly a, int b) {
158     const int invb = power(b, P - 2);
159     for (int i = 0; i < a.size(); i++) {
160         a[i] = 1ll * a[i] * invb % P;
161     }
162     return a;
163 }
164
165 Poly deriv(const Poly &a) {
166     if (a.empty()) {
167         return Poly();
168     }
169     Poly res(a.size() - 1);
170     for (int i = 0; i < res.size(); i++) {
171         res[i] = 1ll * (i + 1) * a[i + 1] % P;
172     }
173     return res;
174 }
175 Poly integr(const Poly &a) {
176     Poly res(a.size() + 1);
177     for (int i = 0; i < a.size(); ++i) {
178         res[i + 1] = 1ll * a[i] * power(i + 1, P - 2) % P;
179     }
180     return res;
181 }
182 Poly inv(const Poly &a, int m) {
183     Poly x {power(a[0], P - 2)};
184     for (int k = 1; k < m; ) {
185         k *= 2;
186         x = (x * (Poly {2} - trunc(a, k) * x));
187         x.resize(k);
188     }
189     x.resize(m);
190     return x;
191 }
192 Poly log(Poly a, int m) {
193     a = deriv(a) * inv(a, m);
194     a = integr(a);
195     a.resize(m);

```

```

196     return a;
197 }
198 Poly exp(Poly a, int m) {
199     Poly x {1};
200     for (int k = 1; k < m; ) {
201         k *= 2;
202         x = (x * (Poly {1} - log(x, k) + trunc(a, k)));
203         x.resize(k);
204     }
205     x.resize(m);
206     return x;
207 }

```

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     }
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 }

```

2.9.4 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 }

```

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) {
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 }

```

3.1.2 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  void init(int n) {
12      this->n = n;
13      siz.resize(n);
14      top.resize(n);
15      dep.resize(n);
16      parent.resize(n);
17      in.resize(n);
18      out.resize(n);
19      seq.resize(n);
20      cur = 0;
21      adj.assign(n, {});
22  }
23  void addEdge(int u, int v) {
24      adj[u].push_back(v);
25      adj[v].push_back(u);
26  }
27  void work(int root = 0) {
28      top[root] = root;
29      dep[root] = 0;
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      }
87      if (dep[u] > dep[v]) {
88          ret.push_back({in[v], in[u]});
89      } else {
90          ret.push_back({in[u], in[v]});
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) {
132     return lca(a, b) ^ lca(b, c) ^ lca(c, a);
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             stk.pop_back();
41         } while (y != x);
42         cnt++;
43     }
44 }
45 std::vector<int> work() {
46     for (int i = 0; i < n; i++) {
47         if (dfn[i] == -1) {
48             dfs(i);
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    }
46    std::pair<int, std::vector<std::pair<int, int>>> work() {
47        for (int i = 0; i < n; i++) {
48            if (dfn[i] == -1) {
49                stk.clear();
50                dfs(i);
51            }
52        }
53        return {cnt, edges};
54    }
55 };

```

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 using Hash = array<int, 2>;
2
3 constexpr int P = 998244353;
4 constexpr Hash B = {567, 1234}; // any
5 {
6     string s;
7     cin >> s;
8
9     const int n = s.size();
10
11    vector<Hash> h(n + 1), p(n + 1);
12    p[0] = {1, 1};
13    for (int j = 0; j < 2; j++) {
14        for (int i = 0; i < n; i++) {
15            h[i + 1][j] = (i64(B[j]) * h[i][j] + s[i]) % P;
16            p[i + 1][j] = (i64(B[j]) * p[i][j]) % P;
17        }
18    }
19
20    auto get = [&](int l, int r) {
21        Hash res{};
22        for (int i = 0; i < 2; i++) {
23            res[i] = (h[r][i] + 1ll * (P - h[l][i]) * p[r - l][i]) % P;
24        }
25        return res;
26    };
27 }

```

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 {
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                    q.push(t[x].next[i]);
54                }
55            }
56        }
57    }
58
59    int next(int p, int x) {
60        return t[p].next[x];
61    }
62
63    int link(int p) {
64        return t[p].link;
65    }
66
67    int len(int p) {
68        return t[p].len;
69    }
70
71    int size() {
72        return t.size();
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         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     }
59 }

```

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    }
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
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

5.1 Vector2D

```

1 // Point
2 struct Point {
3     double x;
4     double y;
5
6     Point operator-() const {
7         return Point(-x, -y);
8     }
9 };
10
11 // Basic Point Operations
12 Point operator+(Point a, Point b) {
13     return Point(a.x + b.x, a.y + b.y);
14 }
15 Point operator-(Point a, Point b) {
16     return Point(a.x + b.x, a.y + b.y);
17 }
18 Point operator*(double a, Point b) {
19     return Point(a * b.x, a * b.y);
20 }
21 Point operator*(Point a, double b) {
22     return b * a;
23 }
24 bool operator==(Point a, Point b) {
25     return a.x == b.x and a.y == b.y;
26 }
27
28 double dot(const Point &a, const Point &b) {
29     return a.x * b.x + a.y * b.y;
30 }
31 double cross(const Point &a, const Point &b) {
32     return a.x * b.y - a.y * b.x;
33 }
34 double square(const Point &p) {
35     return dot(p, p);
36 }
37 double length(const Point &p) {
38     return std::sqrt(square(p));
39 }
40
41 Point operator/(Point a, double b) {
42     return Point(a.x / b, a.y / b);
43 }
44 Point normalize(Point p) {
45     return p / length(p);
46 }
47

```

```

48 struct Line {
49     Point a;
50     Point b;
51 };
52
53 double length(const Line &l) {
54     return length(l.a - l.b);
55 }
56 bool parallel(const Line &l1, const Line &l2) {
57     return cross(l1.b - l1.a, l2.b - l2.a) == 0;
58 }
59
60 double distance(const Point &a, const Point &b) {
61     return length(a - b);
62 }
63 double distancePL(const Point &p, const Line &l) {
64     return std::abs(cross(l.a - l.b, l.a - p)) / length(l);
65 }
66 double distancePS(const Point &p, const Line &l) {
67     if (dot(p - l.a, l.b - l.a) < 0) {
68         return distance(p, l.a);
69     }
70     if (dot(p - l.b, l.a - l.b) < 0) {
71         return distance(p, l.b);
72     }
73     return distancePL(p, l);
74 }
75
76 Point rotate(const Point &a) {
77     return Point(-a.y, a.x);
78 }
79 int sgn(const Point &a) {
80     return a.y > 0 or (a.y == 0 and a.x > 0) ? 1 : -1;
81 }
82 bool pointOnLineLeft(const Point &p, const Line &l) {
83     return cross(l.b - l.a, p - l.a) > 0;
84 }
85
86 Point lineIntersection(const Line &l1, const Line &l2) {
87     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a, l1.a - l1.b));
88 }
89 // Includes endpoints
90 bool pointOnSegment(const Point &p, const Line &l) {
91     return cross(p - l.a, l.b - l.a) == 0 and std::min(l.a.x, l.b.x) ≤ p.x
92         and p.x ≤ std::max(l.a.x, l.b.x)
93         and std::min(l.a.y, l.b.y) ≤ p.y and p.y ≤ std::max(l.a.y, l.b.y);
94 }
95 // O(n)
96 // using O(log n) method if convex
97 bool pointInPolygon(const Point &a, const std::vector<Point> &p) {

```

```

97     int n = p.size();
98     for (int i = 0; i < n; i++) {
99         if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
100             return true;
101         }
102     }
103     int t = 0;
104     for (int i = 0; i < n; i++) {
105         auto u = p[i];
106         auto v = p[(i + 1) % n];
107         if (u.x < a.x and v.x ≥ a.x and pointOnLineLeft(a, Line(v, u))) {
108             t ^= 1;
109         }
110         if (u.x ≥ a.x and v.x < a.x and pointOnLineLeft(a, Line(u, v))) {
111             t ^= 1;
112         }
113     }
114     return t == 1;
115 }
116 // 0 : not intersect
117 // 1 : strictly intersect
118 // 2 : overlap
119 // 3 : intersect at endpoint
120 std::tuple<int, Point, Point> segmentIntersection(const Line &l1, const Line
&l2) {
121     if (std::max(l1.a.x, l1.b.x) < std::min(l2.a.x, l2.b.x)) {
122         return {0, Point(), Point()};
123     }
124     if (std::min(l1.a.x, l1.b.x) > std::max(l2.a.x, l2.b.x)) {
125         return {0, Point(), Point()};
126     }
127     if (std::max(l1.a.y, l1.b.y) < std::min(l2.a.y, l2.b.y)) {
128         return {0, Point(), Point()};
129     }
130     if (std::min(l1.a.y, l1.b.y) > std::max(l2.a.y, l2.b.y)) {
131         return {0, Point(), Point()};
132     }
133     if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
134         if (cross(l1.b - l1.a, l2.a - l1.a) ≠ 0) {
135             return {0, Point(), Point()};
136         } else {
137             auto maxx1 = std::max(l1.a.x, l1.b.x);
138             auto minx1 = std::min(l1.a.x, l1.b.x);
139             auto maxy1 = std::max(l1.a.y, l1.b.y);
140             auto miny1 = std::min(l1.a.y, l1.b.y);
141             auto maxx2 = std::max(l2.a.x, l2.b.x);
142             auto minx2 = std::min(l2.a.x, l2.b.x);
143             auto maxy2 = std::max(l2.a.y, l2.b.y);
144             auto miny2 = std::min(l2.a.y, l2.b.y);
145             Point p1(std::max(minx1, minx2), std::max(miny1, miny2));
146             Point p2(std::min(maxx1, maxx2), std::min(maxy1, maxy2));

```

```

147         if (!pointOnSegment(p1, l1)) {
148             std::swap(p1.y, p2.y);
149         }
150         if (p1 == p2) {
151             return {3, p1, p2};
152         } else {
153             return {2, p1, p2};
154         }
155     }
156 }
157 auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
158 auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
159 auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
160 auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);
161 if ((cp1 > 0 and cp2 > 0) or (cp1 < 0 and cp2 < 0) or (cp3 > 0 and cp4 >
0) or (cp3 < 0 and cp4 < 0)) {
162     return {0, Point(), Point()};
163 }
164 Point p = lineIntersection(l1, l2);
165 if (cp1 ≠ 0 and cp2 ≠ 0 and cp3 ≠ 0 and cp4 ≠ 0) {
166     return {1, p, p};
167 } else {
168     return {3, p, p};
169 }
170 }
171
172 double distanceSS(const Line &l1, const Line &l2) {
173     if (std::get<0>(segmentIntersection(l1, l2)) ≠ 0) {
174         return 0.0;
175     }
176     return std::min({distancePS(l1.a, l2), distancePS(l1.b, l2), distancePS(
l2.a, l1), distancePS(l2.b, l1)});
177 }
178
179 bool segmentInPolygon(const Line &l, const std::vector<Point> &p) {
180     int n = p.size();
181     if (not pointInPolygon(l.a, p)) {
182         return false;
183     }
184     if (not pointInPolygon(l.b, p)) {
185         return false;
186     }
187     for (int i = 0; i < n; i++) {
188         auto u = p[i];
189         auto v = p[(i + 1) % n];
190         auto w = p[(i + 2) % n];
191         auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
192         if (t == 1) {
193             return false;
194         }
195         if (t == 0) {

```



```

196     continue;
197 }
198 if (t == 2) {
199     if (pointOnSegment(v, l) and v != l.a and v != l.b) {
200         if (cross(v - u, w - v) > 0) {
201             return false;
202         }
203     }
204 } else {
205     if (p1 != u and p1 != v) {
206         if (pointOnLineLeft(l.a, Line(v, u)) or
207             pointOnLineLeft(l.b, Line(v, u))) {
208             return false;
209         }
210     } else if (p1 == v) {
211         if (l.a == v) {
212             if (pointOnLineLeft(u, l)) {
213                 if (pointOnLineLeft(w, l) and
214                     pointOnLineLeft(w, Line(u, v))) {
215                     return false;
216                 }
217             } else {
218                 if (pointOnLineLeft(w, l) or
219                     pointOnLineLeft(w, Line(u, v))) {
220                     return false;
221                 }
222             }
223         } else if (l.b == v) {
224             if (pointOnLineLeft(u, Line(l.b, l.a))) {
225                 if (pointOnLineLeft(w, Line(l.b, l.a)) and
226                     pointOnLineLeft(w, Line(u, v))) {
227                     return false;
228                 }
229             } else {
230                 if (pointOnLineLeft(w, Line(l.b, l.a)) or
231                     pointOnLineLeft(w, Line(u, v))) {
232                     return false;
233                 }
234             }
235         } else {
236             if (pointOnLineLeft(u, l)) {
237                 if (pointOnLineLeft(w, Line(l.b, l.a)) or
238                     pointOnLineLeft(w, Line(u, v))) {
239                     return false;
240                 }
241             } else {
242                 if (pointOnLineLeft(w, l) or
243                     pointOnLineLeft(w, Line(u, v))) {
244                     return false;
245                 }
246             }
247         }
248     }
249 }

```

```

247     }
248 }
249 }
250 }
251 return true;
252 }

```

5.2 ConvexHull

```

1 struct Point {
2     i64 x, y;
3 };
4 struct Line {
5     Point a {}, b {};
6 };
7
8 bool operator==(const Point &a, const Point &b) {
9     return a.x == b.x && a.y == b.y;
10 }
11 Point operator+(const Point &a, const Point &b) {
12     return Point(a.x + b.x, a.y + b.y);
13 }
14 Point operator-(const Point &a, const Point &b) {
15     return Point(a.x - b.x, a.y - b.y);
16 }
17 i64 dot(const Point &a, const Point &b) {
18     return a.x * b.x + a.y * b.y;
19 }
20 i64 cross(const Point &a, const Point &b) {
21     return a.x * b.y - a.y * b.x;
22 }
23 i64 square(const Point &a) {
24     return a.x * a.x + a.y * a.y;
25 }
26 double length(const Point &a) {
27     return std::sqrt(square(a));
28 }
29
30 i64 get(const Point &p, const Line &l) {
31     return cross(l.b - l.a, p - l.a);
32 }
33 bool pointInConvex(const Point &a, const std::vector<Point> &p) {
34     int n = p.size();
35     if (get(a, Line(p[0], p[1])) < 0) {
36         return false;
37     }
38     if (get(a, Line(p[0], p[n - 1])) > 0) {
39         return false;
40     }
41 }

```

```

42 int lo = 1, hi = n;
43 while (lo < hi) {
44     int x = (lo + hi) / 2;
45     if (get(a, Line(p[0], p[x])) ≤ 0) {
46         hi = x;
47     } else {
48         lo = x + 1;
49     }
50 }
51 if (lo == 1) {
52     return square(a - p[0]) ≤ square(p[1] - p[0]);
53 }
54 return get(a, Line(p[lo - 1], p[lo])) ≥ 0;
55 }
56
57 std::vector<Point> convexHull(std::vector<Point> a) {
58     int n = a.size();
59     if (n ≤ 1) {
60         return a;
61     }
62
63     sort(a.begin(), a.end(),
64         [&](auto a, auto b) {
65             return a.x < b.x or (a.x == b.x and a.y < b.y);
66         });
67
68     std::vector<Point> h;
69
70     for (int i = 0; i < n; i++) {
71         while (h.size() ≥ 2 and cross(h[h.size() - 2] - a[i], h.back() - a[i]) ≤ 0) {
72             h.pop_back();
73         }
74         h.push_back(a[i]);
75     }
76
77     int k = h.size();
78     for (int i = n - 2; i ≥ 0; i--) {
79         while (h.size() ≥ k + 1 and cross(h[h.size() - 2] - a[i], h.back() - a[i]) ≤ 0) {
80             h.pop_back();
81         }
82         h.push_back(a[i]);
83     }
84
85     h.pop_back();
86     return h;
87 }
88
89 std::vector<Point> minkowski(const std::vector<Point> &a, const std::vector<
    Point> &b) {

```

```

90 int n = a.size(), m = b.size();
91 if (n == 0 or m == 0) {
92     return {};
93 }
94
95 std::vector<Point> c{a[0] + b[0]};
96 int i = 0, j = 0;
97 while (i < n and j < m) {
98     auto va = a[(i + 1) % n] - a[i];
99     auto vb = b[(j + 1) % m] - b[j];
100
101     auto v = cross(va, vb);
102     if (v > 0) {
103         c.push_back(c.back() + va);
104         i++;
105     } else if (v < 0) {
106         c.push_back(c.back() + vb);
107         j++;
108     } else {
109         c.push_back(c.back() + va + vb);
110         i++;
111         j++;
112     }
113 }
114 while (i < n) {
115     c.push_back(c.back() + a[(i + 1) % n] - a[i]);
116     i++;
117 }
118 while (j < m) {
119     c.push_back(c.back() + b[(j + 1) % m] - b[j]);
120     j++;
121 }
122 c.pop_back();
123
124 return c;
125 }

```

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 };

```

6.2 Fraction

```

1 using i128 = __int128;
2
3 struct Frac {
4     i128 num;
5     i128 den;
6     Frac(i128 num_, i128 den_) : num(num_), den(den_) {
7         if (den < 0) {
8             den = -den;
9             num = -num;
10        }
11    }
12    Frac() : Frac(0, 1) {}
13    Frac(i128 num_) : Frac(num_, 1) {}
14    explicit operator double() const {
15        return 1. * num / den;
16    }
17    Frac &operator+=(const Frac &rhs) {
18        num = num * rhs.den + rhs.num * den;
19        den *= rhs.den;
20        return *this;
21    }
22    Frac &operator-=(const Frac &rhs) {
23        num = num * rhs.den - rhs.num * den;
24        den *= rhs.den;
25        return *this;

```

```

26 }
27 Frac &operator*=(const Frac &rhs) {
28     num *= rhs.num;
29     den *= rhs.den;
30     return *this;
31 }
32 Frac &operator/=(const Frac &rhs) {
33     num *= rhs.den;
34     den *= rhs.num;
35     if (den < 0) {
36         num = -num;
37         den = -den;
38     }
39     return *this;
40 }
41 friend Frac operator+(Frac lhs, const Frac &rhs) {
42     return lhs += rhs;
43 }
44 friend Frac operator-(Frac lhs, const Frac &rhs) {
45     return lhs -= rhs;
46 }
47 friend Frac operator*(Frac lhs, const Frac &rhs) {
48     return lhs *= rhs;
49 }
50 friend Frac operator/(Frac lhs, const Frac &rhs) {
51     return lhs /= rhs;
52 }
53 friend Frac operator-(const Frac &a) {
54     return Frac(-a.num, a.den);
55 }
56 friend bool operator==(const Frac &lhs, const Frac &rhs) {
57     return lhs.num * rhs.den == rhs.num * lhs.den;
58 }
59 friend bool operator!=(const Frac &lhs, const Frac &rhs) {
60     return lhs.num * rhs.den != rhs.num * lhs.den;
61 }
62 friend bool operator<(const Frac &lhs, const Frac &rhs) {
63     return lhs.num * rhs.den < rhs.num * lhs.den;
64 }
65 friend bool operator>(const Frac &lhs, const Frac &rhs) {
66     return lhs.num * rhs.den > rhs.num * lhs.den;
67 }
68 friend bool operator<=(const Frac &lhs, const Frac &rhs) {
69     return lhs.num * rhs.den <= rhs.num * lhs.den;
70 }
71 friend bool operator>=(const Frac &lhs, const Frac &rhs) {
72     return lhs.num * rhs.den >= rhs.num * lhs.den;
73 }
74 friend std::ostream &operator<<(std::ostream &os, Frac x) {
75     i128 g = std::gcd(x.num, x.den);
76     if (x.den == g) {

```

```

77         return os << x.num / g;
78     } else {
79         return os << x.num / g << "/" << x.den / g;
80     }
81 }
82 };

```

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: $\pi(x) \sim x / \log(x)$