

# ICPC Template Library

postpone | Shu-i64

December 21, 2025

v1.0 •••••

ref: jiangly, skip2004, zeemanz, capps

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

## Contents

<b>1 Data Structure</b>	<b>2</b>
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
<b>2 Math</b>	<b>8</b>
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 . . . . .	14
2.9.4 Sum of Subset . . . . .	14
<b>3 Graph</b>	<b>14</b>
3.1 Tree . . . . .	14
3.1.1 Lowest Common Ancestor . . . . .	14
3.1.2 Heavy-Light Decomposition . . . . .	15
3.1.3 DSU on Tree . . . . .	16
3.2 Connectivity . . . . .	17
3.2.1 Strongly Connected Component . . . . .	17
3.2.2 Block Cut Tree . . . . .	17
3.3 Two Sat . . . . .	18

<b>4 String</b>	<b>18</b>
4.1 String Hash . . . . .	18
4.2 KMP . . . . .	19
4.3 Z-function . . . . .	19
4.4 AhoCorasick . . . . .	19
4.5 Suffix Array . . . . .	20
4.6 Suffix Automaton . . . . .	21
4.7 Palindromic Tree . . . . .	21
<b>5 Geometry</b>	<b>22</b>

<b>6 Util</b>	<b>26</b>
6.1 Mod Integer . . . . .	26
6.2 Fraction . . . . .	28

<b>7 Tables</b>	<b>29</b>
7.1 Constant . . . . .	29

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

```

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<T>>
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     }
69     T operator()(int l, int r) const {
70         if (l >= r) {
71             return e;
72         } else {
73             const int k = std::__lg(r - l);
74             return F(a[k][l], a[k][r - (1 << k)]);
75         }
76     }
77 };

```

## 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 constexpr 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 constexpr 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 Poly trunc(Poly a, int k) {
86     a.resize(k);
87     return a;
88 }
89
90 Poly operator+(const Poly &a, const Poly &b) {
91     Poly res(std::max(a.size(), b.size()));
92     for (int i = 0; i < a.size(); i++) {
93         res[i] = a[i];
```

## postpone

```

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() || 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 std::vector<int> &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 std::vector<int> &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 std::vector<int> &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(std::vector<int> 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(std::vector<int> 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)
```

## postpone

```

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 }

```

```

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        siz[u] = 1;
39        for (auto &&v : adj[u]) {
40            parent[v] = u;
41            dep[v] = dep[u] + 1;
42            dfs1(v);
43            siz[u] += siz[v];
44            if (siz[v] > siz[adj[u][0]]) {
45                std::swap(v, adj[u][0]);
46            }
47        }
48    }
49    void dfs2(int u) {
50        in[u] = cur++;
51        seq[in[u]] = u;
52        for (auto v : adj[u]) {
53            top[v] = v == adj[u][0] ? top[u] : v;
54        }
55    }

```

## 3.1.2 Heavy-Light Decomposition

```

1 struct HLD {
2     int n;
3     std::vector<int> siz, top, dep, parent, in, out, seq;

```

```

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        }

```

## postpone

```

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++;
30                }
31            } else {
32                low[x] = std::min(low[x], dfn[y]);
33            }
34        }
35    }
36    std::pair<int, std::vector<std::pair<int, int>>> work() {
37        for (int i = 0; i < n; i++) {
38            if (dfn[i] == -1) {
39                stk.clear();
40                dfs(i);
41            }
42        }
43        return {cnt, edges};
44    }
45};

```

```

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};

```

### 3.3 Two Sat

## 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 }
```

## postpone

```

11     return f;
12 }
```

## 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    }
```

## 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

## postpone

```

49     if (rk[i] == 0) {
50         j = 0;
51     } else {
52         for (j -= (j > 0); i + j < n && sa[rk[i] - 1] + j < n && s[i
53             + j] == s[sa[rk[i] - 1] + j]; j++)
54             ;
55         lc[rk[i] - 1] = j;
56     }
57 }
58 };

```

## 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     int next(int p, int x) {
52         return t[p].next[x];
53     }
54
55     int next(int p, char c, char offset = 'a') {
56         return next(p, c - 'a');
57     }
58
59     int link(int p) {
60         return t[p].link;
61     }
62
63     int len(int p) {
64         return t[p].len;
65     }
66
67     int size() {
68         return t.size();
69     }
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;

```

## postpone

```

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
60 template<class T>
61 struct Line {
62     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
68 template<class T>
69 T dot(const Point<T> &a, const Point<T> &b) {
70     return a.x * b.x + a.y * b.y;
71 }
72
73 template<class T>
74 T cross(const Point<T> &a, const Point<T> &b) {
75     return a.x * b.y - a.y * b.x;
76 }
77
78 template<class T>
79 T square(const Point<T> &p) {
80     return dot(p, p);
81 }
82
83 template<class T>
84 double length(const Point<T> &p) {
85     return std::sqrt(square(p));
86 }
87 template<class T>
88 double length(const Line<T> &l) {
89     return length(l.b - l.a);
90 }
91
92 template<class T>
93 Point<T> normalize(const Point<T> &p) {
94     return p / length(p);
95 }
96
97 template<class T>
98 bool parallel(const Line<T> &l1, const Line<T> &l2) {
99     return cross(l1.b - l1.a, l2.b - l2.a) == 0;
100 }
101
102 template<class T>
103 double distance(const Point<T> &a, const Point<T> &b) {
104     return length(a - b);
105 }
106
107 template<class T>
108 double distancePL(const Point<T> &p, const Line<T> &l) {
109     return std::abs(cross(l.a - l.b, l.a - p)) / length(l);
110 }
111
112 template<class T>
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
170     return t = 1;
171 }
172
173 // 0 : not intersect
174 // 1 : strictly intersect
175 // 2 : overlap
176 // 3 : intersect at endpoint
177 template<class T>
178 std::tuple<int, Point<T>, Point<T>> segmentIntersection(const Line<T> &l1,
179     const Line<T> &l2) {
180     if (std::max(l1.a.x, l1.b.x) < std::min(l2.a.x, l2.b.x)) {
181         return {0, Point<T>(), Point<T>()};
182     }
183     if (std::min(l1.a.x, l1.b.x) > std::max(l2.a.x, l2.b.x)) {
184         return {0, Point<T>(), Point<T>()};
185     }
186     if (std::max(l1.a.y, l1.b.y) < std::min(l2.a.y, l2.b.y)) {
187         return {0, Point<T>(), Point<T>()};
188     }
189     if (std::min(l1.a.y, l1.b.y) > std::max(l2.a.y, l2.b.y)) {
190         return {0, Point<T>(), Point<T>()};
191     }
192     if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
193         if (cross(l1.b - l1.a, l2.a - l1.a) ≠ 0) {
194             return {0, Point<T>(), Point<T>()};
195         } else {
196             auto maxx1 = std::max(l1.a.x, l1.b.x);
197             auto minx1 = std::min(l1.a.x, l1.b.x);
198             auto maxy1 = std::max(l1.a.y, l1.b.y);
199             auto miny1 = std::min(l1.a.y, l1.b.y);
200             auto maxx2 = std::max(l2.a.x, l2.b.x);
201             auto minx2 = std::min(l2.a.x, l2.b.x);
202             auto maxy2 = std::max(l2.a.y, l2.b.y);
203             auto miny2 = std::min(l2.a.y, l2.b.y);
204             Point<T> p1(std::max(minx1, minx2), std::max(miny1, miny2));
205             Point<T> p2(std::min(maxx1, maxx2), std::min(maxy1, maxy2));
206             if (!pointOnSegment(p1, l1)) {
207                 std::swap(p1.y, p2.y);
208             }
209             if (p1 == p2) {
210                 return {3, p1, p2};
211             } else {
212                 return {2, p1, p2};
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) {

```

## postpone

```

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 template<class T>
241 bool segmentInPolygon(const Line<T> &l, const std::vector<Point<T>> &p) {
242     int n = p.size();
243     if (!pointInPolygon(l.a, p)) {
244         return false;
245     }
246     if (!pointInPolygon(l.b, p)) {
247         return false;
248     }
249     for (int i = 0; i < n; i++) {
250         auto u = p[i];
251         auto v = p[(i + 1) % n];
252         auto w = p[(i + 2) % n];
253         auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
254
255         if (t == 1) {
256             return false;
257         }
258         if (t == 0) {
259             continue;
260         }
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         }
299     }
300     if (pointOnLineLeft(u, l)) {
301         if (pointOnLineLeft(w, Line(l.b, l.a))
302             || pointOnLineLeft(w, Line(u, v))) {
303             return false;
304         }
305     } else {
306         if (pointOnLineLeft(w, l)
307             || pointOnLineLeft(w, Line(u, v))) {
308             return false;
309         }
310     }
311 }
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         if (cross(l.b - l.a, ls.back().b - ls.back().a) <= 0) {
339             ls.push_back(l);
340         } else {
341             ps.push_back(l.a);
342         }
343
344         if (cross(l.b - l.a, ls.back().b - ls.back().a) >= 0) {
345             ls.pop_back();
346         } else {
347             ps.push_back(l.b);
348         }
349
350         if (ls.empty() && !ps.empty()) {
351             ls.push_back(ps.back());
352             ps.pop_back();
353         }
354
355         if (ls.size() > 2) {
356             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
357                 ls.pop_back();
358             } else {
359                 ls.pop_back();
360                 ls.pop_back();
361             }
362         }
363
364         if (ls.size() > 2) {
365             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
366                 ls.pop_back();
367             } else {
368                 ls.pop_back();
369                 ls.pop_back();
370             }
371         }
372
373         if (ls.size() > 2) {
374             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
375                 ls.pop_back();
376             } else {
377                 ls.pop_back();
378                 ls.pop_back();
379             }
380         }
381
382         if (ls.size() > 2) {
383             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
384                 ls.pop_back();
385             } else {
386                 ls.pop_back();
387                 ls.pop_back();
388             }
389         }
390
391         if (ls.size() > 2) {
392             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
393                 ls.pop_back();
394             } else {
395                 ls.pop_back();
396                 ls.pop_back();
397             }
398         }
399
400         if (ls.size() > 2) {
401             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
402                 ls.pop_back();
403             } else {
404                 ls.pop_back();
405                 ls.pop_back();
406             }
407         }
408
409         if (ls.size() > 2) {
410             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
411                 ls.pop_back();
412             } else {
413                 ls.pop_back();
414                 ls.pop_back();
415             }
416         }
417
418         if (ls.size() > 2) {
419             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
420                 ls.pop_back();
421             } else {
422                 ls.pop_back();
423                 ls.pop_back();
424             }
425         }
426
427         if (ls.size() > 2) {
428             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
429                 ls.pop_back();
430             } else {
431                 ls.pop_back();
432                 ls.pop_back();
433             }
434
435             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
436                 ls.pop_back();
437             } else {
438                 ls.pop_back();
439                 ls.pop_back();
440             }
441
442             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
443                 ls.pop_back();
444             } else {
445                 ls.pop_back();
446                 ls.pop_back();
447             }
448
449             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
450                 ls.pop_back();
451             } else {
452                 ls.pop_back();
453                 ls.pop_back();
454             }
455
456             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
457                 ls.pop_back();
458             } else {
459                 ls.pop_back();
460                 ls.pop_back();
461             }
462
463             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
464                 ls.pop_back();
465             } else {
466                 ls.pop_back();
467                 ls.pop_back();
468             }
469
470             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
471                 ls.pop_back();
472             } else {
473                 ls.pop_back();
474                 ls.pop_back();
475             }
476
477             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
478                 ls.pop_back();
479             } else {
480                 ls.pop_back();
481                 ls.pop_back();
482             }
483
484             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
485                 ls.pop_back();
486             } else {
487                 ls.pop_back();
488                 ls.pop_back();
489             }
490
491             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
492                 ls.pop_back();
493             } else {
494                 ls.pop_back();
495                 ls.pop_back();
496             }
497
498             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
499                 ls.pop_back();
500             } else {
501                 ls.pop_back();
502                 ls.pop_back();
503             }
504
505             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
506                 ls.pop_back();
507             } else {
508                 ls.pop_back();
509                 ls.pop_back();
510             }
511
512             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
513                 ls.pop_back();
514             } else {
515                 ls.pop_back();
516                 ls.pop_back();
517             }
518
519             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
520                 ls.pop_back();
521             } else {
522                 ls.pop_back();
523                 ls.pop_back();
524             }
525
526             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
527                 ls.pop_back();
528             } else {
529                 ls.pop_back();
530                 ls.pop_back();
531             }
532
533             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
534                 ls.pop_back();
535             } else {
536                 ls.pop_back();
537                 ls.pop_back();
538             }
539
540             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
541                 ls.pop_back();
542             } else {
543                 ls.pop_back();
544                 ls.pop_back();
545             }
546
547             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
548                 ls.pop_back();
549             } else {
550                 ls.pop_back();
551                 ls.pop_back();
552             }
553
554             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
555                 ls.pop_back();
556             } else {
557                 ls.pop_back();
558                 ls.pop_back();
559             }
560
561             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
562                 ls.pop_back();
563             } else {
564                 ls.pop_back();
565                 ls.pop_back();
566             }
567
568             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
569                 ls.pop_back();
570             } else {
571                 ls.pop_back();
572                 ls.pop_back();
573             }
574
575             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
576                 ls.pop_back();
577             } else {
578                 ls.pop_back();
579                 ls.pop_back();
580             }
581
582             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
583                 ls.pop_back();
584             } else {
585                 ls.pop_back();
586                 ls.pop_back();
587             }
588
589             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
590                 ls.pop_back();
591             } else {
592                 ls.pop_back();
593                 ls.pop_back();
594             }
595
596             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
597                 ls.pop_back();
598             } else {
599                 ls.pop_back();
600                 ls.pop_back();
601             }
602
603             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
604                 ls.pop_back();
605             } else {
606                 ls.pop_back();
607                 ls.pop_back();
608             }
609
610             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
611                 ls.pop_back();
612             } else {
613                 ls.pop_back();
614                 ls.pop_back();
615             }
616
617             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
618                 ls.pop_back();
619             } else {
620                 ls.pop_back();
621                 ls.pop_back();
622             }
623
624             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
625                 ls.pop_back();
626             } else {
627                 ls.pop_back();
628                 ls.pop_back();
629             }
630
631             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
632                 ls.pop_back();
633             } else {
634                 ls.pop_back();
635                 ls.pop_back();
636             }
637
638             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
639                 ls.pop_back();
640             } else {
641                 ls.pop_back();
642                 ls.pop_back();
643             }
644
645             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
646                 ls.pop_back();
647             } else {
648                 ls.pop_back();
649                 ls.pop_back();
650             }
651
652             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
653                 ls.pop_back();
654             } else {
655                 ls.pop_back();
656                 ls.pop_back();
657             }
658
659             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
660                 ls.pop_back();
661             } else {
662                 ls.pop_back();
663                 ls.pop_back();
664             }
665
666             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
667                 ls.pop_back();
668             } else {
669                 ls.pop_back();
670                 ls.pop_back();
671             }
672
673             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
674                 ls.pop_back();
675             } else {
676                 ls.pop_back();
677                 ls.pop_back();
678             }
679
680             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
681                 ls.pop_back();
682             } else {
683                 ls.pop_back();
684                 ls.pop_back();
685             }
686
687             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
688                 ls.pop_back();
689             } else {
690                 ls.pop_back();
691                 ls.pop_back();
692             }
693
694             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
695                 ls.pop_back();
696             } else {
697                 ls.pop_back();
698                 ls.pop_back();
699             }
700
701             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
702                 ls.pop_back();
703             } else {
704                 ls.pop_back();
705                 ls.pop_back();
706             }
707
708             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
709                 ls.pop_back();
710             } else {
711                 ls.pop_back();
712                 ls.pop_back();
713             }
714
715             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
716                 ls.pop_back();
717             } else {
718                 ls.pop_back();
719                 ls.pop_back();
720             }
721
722             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
723                 ls.pop_back();
724             } else {
725                 ls.pop_back();
726                 ls.pop_back();
727             }
728
729             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
730                 ls.pop_back();
731             } else {
732                 ls.pop_back();
733                 ls.pop_back();
734             }
735
736             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
737                 ls.pop_back();
738             } else {
739                 ls.pop_back();
740                 ls.pop_back();
741             }
742
743             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
744                 ls.pop_back();
745             } else {
746                 ls.pop_back();
747                 ls.pop_back();
748             }
749
750             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
751                 ls.pop_back();
752             } else {
753                 ls.pop_back();
754                 ls.pop_back();
755             }
756
757             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
758                 ls.pop_back();
759             } else {
760                 ls.pop_back();
761                 ls.pop_back();
762             }
763
764             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
765                 ls.pop_back();
766             } else {
767                 ls.pop_back();
768                 ls.pop_back();
769             }
770
771             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
772                 ls.pop_back();
773             } else {
774                 ls.pop_back();
775                 ls.pop_back();
776             }
777
778             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
779                 ls.pop_back();
780             } else {
781                 ls.pop_back();
782                 ls.pop_back();
783             }
784
785             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
786                 ls.pop_back();
787             } else {
788                 ls.pop_back();
789                 ls.pop_back();
790             }
791
792             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
793                 ls.pop_back();
794             } else {
795                 ls.pop_back();
796                 ls.pop_back();
797             }
798
799             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
800                 ls.pop_back();
801             } else {
802                 ls.pop_back();
803                 ls.pop_back();
804             }
805
806             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
807                 ls.pop_back();
808             } else {
809                 ls.pop_back();
810                 ls.pop_back();
811             }
812
813             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
814                 ls.pop_back();
815             } else {
816                 ls.pop_back();
817                 ls.pop_back();
818             }
819
820             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
821                 ls.pop_back();
822             } else {
823                 ls.pop_back();
824                 ls.pop_back();
825             }
826
827             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
828                 ls.pop_back();
829             } else {
830                 ls.pop_back();
831                 ls.pop_back();
832             }
833
834             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
835                 ls.pop_back();
836             } else {
837                 ls.pop_back();
838                 ls.pop_back();
839             }
840
841             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
842                 ls.pop_back();
843             } else {
844                 ls.pop_back();
845                 ls.pop_back();
846             }
847
848             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
849                 ls.pop_back();
850             } else {
851                 ls.pop_back();
852                 ls.pop_back();
853             }
854
855             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
856                 ls.pop_back();
857             } else {
858                 ls.pop_back();
859                 ls.pop_back();
860             }
861
862             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
863                 ls.pop_back();
864             } else {
865                 ls.pop_back();
866                 ls.pop_back();
867             }
868
869             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
870                 ls.pop_back();
871             } else {
872                 ls.pop_back();
873                 ls.pop_back();
874             }
875
876             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
877                 ls.pop_back();
878             } else {
879                 ls.pop_back();
880                 ls.pop_back();
881             }
882
883             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
884                 ls.pop_back();
885             } else {
886                 ls.pop_back();
887                 ls.pop_back();
888             }
889
890             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
891                 ls.pop_back();
892             } else {
893                 ls.pop_back();
894                 ls.pop_back();
895             }
896
897             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
898                 ls.pop_back();
899             } else {
900                 ls.pop_back();
901                 ls.pop_back();
902             }
903
904             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
905                 ls.pop_back();
906             } else {
907                 ls.pop_back();
908                 ls.pop_back();
909             }
910
911             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
912                 ls.pop_back();
913             } else {
914                 ls.pop_back();
915                 ls.pop_back();
916             }
917
918             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
919                 ls.pop_back();
920             } else {
921                 ls.pop_back();
922                 ls.pop_back();
923             }
924
925             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
926                 ls.pop_back();
927             } else {
928                 ls.pop_back();
929                 ls.pop_back();
930             }
931
932             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
933                 ls.pop_back();
934             } else {
935                 ls.pop_back();
936                 ls.pop_back();
937             }
938
939             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
940                 ls.pop_back();
941             } else {
942                 ls.pop_back();
943                 ls.pop_back();
944             }
945
946             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
947                 ls.pop_back();
948             } else {
949                 ls.pop_back();
950                 ls.pop_back();
951             }
952
953             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
954                 ls.pop_back();
955             } else {
956                 ls.pop_back();
957                 ls.pop_back();
958             }
959
960             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
961                 ls.pop_back();
962             } else {
963                 ls.pop_back();
964                 ls.pop_back();
965             }
966
967             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
968                 ls.pop_back();
969             } else {
970                 ls.pop_back();
971                 ls.pop_back();
972             }
973
974             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
975                 ls.pop_back();
976             } else {
977                 ls.pop_back();
978                 ls.pop_back();
979             }
980
981             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
982                 ls.pop_back();
983             } else {
984                 ls.pop_back();
985                 ls.pop_back();
986             }
987
988             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) <= 0) {
989                 ls.pop_back();
990             } else {
991                 ls.pop_back();
992                 ls.pop_back();
993             }
994
995             if (cross(ls[1].b - ls[1].a, ls[2].b - ls[2].a) >= 0) {
996                 ls.pop_back();
997             } else {
998                 ls.pop_back();
999                 ls.pop_back();
1000            }
1001        }
1002    }
1003 }
```

```

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>;

```

```

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     }

```

## 6 Util

### 6.1 Mod Integer

```

1 constexpr int P = 998244353;
2 using i64 = long long;
3 // assume -P <= x < 2P

```

## postpone

```

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     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

```

```

75     rhs) {
76         lhs *= rhs;
77         return lhs;
78     }
79     friend constexpr ModIntBase operator+(ModIntBase lhs, const ModIntBase &
80         rhs) {
81         lhs += rhs;
82         return lhs;
83     }
84     friend constexpr ModIntBase operator-(ModIntBase lhs, const ModIntBase &
85         rhs) {
86         lhs -= rhs;
87         return lhs;
88     }
89     friend constexpr ModIntBase operator/(ModIntBase lhs, const ModIntBase &
90         rhs) {
91         lhs /= rhs;
92         return lhs;
93     }
94
95     friend constexpr std::istream &operator>>(std::istream &is, ModIntBase &a
96     ) {
97         i64 i;
98         is >> i;
99         a = i;
100        return is;
101    }
102    friend constexpr std::ostream &operator<<(std::ostream &os, const
103        ModIntBase &a) {
104        return os << a.val();
105    }
106
107    friend constexpr bool operator==(const ModIntBase &lhs, const ModIntBase
108        &rhs) {
109        return lhs.val() == rhs.val();
110    }
111    friend constexpr std::strong_ordering operator==(const ModIntBase &lhs,
112        const ModIntBase &rhs) {
113        return lhs.val() == rhs.val();
114    }
115
116    private:
117        U x;
118    };
119
120    template <u32 P>
121    using ModInt = ModIntBase<u32, P>;
122
123    template <u64 P>
124    using ModInt64 = ModIntBase<u64, P>;
125
126    template <int V, u32 P>
127
```

```

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    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    }

```

## postpone

```

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     return os << rhs.num << "/" << rhs.den;
65 }
66 };

```

$n \leq$	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	$5.08 \times 10^7$	$4.55 \times 10^8$	$4.12 \times 10^9$	$3.7 \times 10^{10}$
$n \leq$	$10^{13}$	$10^{14}$	$10^{15}$	$10^{16}$	$10^{17}$	$10^{18}$
$\max \omega(n)$	12	12	13	13	14	15
$\max d(n)$	10752	17280	26880	41472	64512	103680

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

## 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	$1.31 \times 10^{12}$	6435	360360	176
20	1.30103000	$2.43 \times 10^{18}$	184756	232792560	627
25	1.39794001	$1.55 \times 10^{25}$	5200300	26771144400	1958
30	1.47712125	$2.65 \times 10^{32}$	155117520	$1.444 \times 10^{14}$	5604