



XCPC Algorithm Template (II)

Jianly's Online version

v8.5 2024.06.01
WIDA

目录

杂类	1
int128 输出流自定义	1
常用库函数重载	1
图与网络	2
强连通分量缩点 (SCC)	2
割边与割边缩点 (EBCC)	3
二分图最大权匹配 (MaxAssignment 基于 KM)【久远】	4
一般图最大匹配 (Graph 带花树算法)【久远】	6
TwoSat (2-Sat)	8
最大流	9
最大流 (Flow 旧版其一, 整数应用)	9
最大流 (Flow 旧版其二, 浮点数应用)	10
最大流 (MaxFlow 新版)	11
费用流	13
费用流 (MCFGGraph 最小费用可行流)	13
费用流 (MCFGGraph 最小费用最大流)	14
树链剖分 (HLD)	15
数论、几何、多项式	17
快速幂	17
欧拉筛	17
莫比乌斯函数筛 (莫比乌斯函数/反演)	17
求解单个数的欧拉函数	19
扩展欧几里得 (exGCD)	19
组合数 (Comb, with. MInt & MLong)	19
二项式 (Binomial 任意模数计算)	20
素数测试与因式分解 (Miller-Rabin & Pollard-Rho)	22
平面几何	23
静态凸包	29
静态凸包 (with. Point)	29
静态凸包 (with. std::complex)	30
多项式相关	31
多项式相关 (Poly, with. Z)	31
多项式相关 (Poly, with. MInt & MLong)	35
数据结构	43
树状数组 (Fenwick 新版)	43
并查集 (DSU)	43
线段树	44
线段树 (SegmentTree 基础区间加乘)	44
线段树 (SegmentTree+Info 查找前驱后继)	45
线段树 (SegmentTree+Info+Merge 区间合并)	47
懒标记线段树	49
懒标记线段树 (LazySegmentTree 基础区间修改)	49
懒标记线段树 (LazySegmentTree 查找前驱后继)	51
懒标记线段树 (LazySegmentTree 二分修改)	54
取模类	57
取模类 (MLong & MInt)	57

取模类 (MLong & MInt 新版)	58
状压 RMQ (RMQ)	62
Splay.....	63
其他平衡树.....	69
分数四则运算 (Frac)	74
线性基 (Basis)	76
字符串	77
马拉车 (Manacher 新版)	77
Z 函数	77
后缀数组 (SA)	77
后缀自动机.....	78
后缀自动机 (SuffixAutomaton 旧版)	78
后缀自动机 (SAM 新版)	79
回文自动机 (PAM)	80
AC 自动机	81
AC 自动机 (AC 旧版)	81
AC 自动机 (AhoCorasick 新新版)	83
随机生成模底 字符串哈希 (例题)	84

1 杂类

1.1 int128 输出流自定义

```
1 using i128 = __int128;
2
3 std::ostream &operator<<(std::ostream &os, i128 n) {
4     std::string s;
5     while (n) {
6         s += '0' + n % 10;
7         n /= 10;
8     }
9     std::reverse(s.begin(), s.end());
10    return os << s;
11 }
```

1.2 常用库函数重载

```
1 using i64 = long long;
2 using i128 = __int128;
3
4 i64 ceilDiv(i64 n, i64 m) {
5     if (n >= 0) {
6         return (n + m - 1) / m;
7     } else {
8         return n / m;
9     }
10 }
11
12 i64 floorDiv(i64 n, i64 m) {
13     if (n >= 0) {
14         return n / m;
15     } else {
16         return (n - m + 1) / m;
17     }
18 }
19
20 template<class T>
21 void chmax(T &a, T b) {
22     if (a < b) {
23         a = b;
24     }
25 }
26
27 i128 gcd(i128 a, i128 b) {
28     return b ? gcd(b, a % b) : a;
29 }
```

/END/

2 图与网络

2.1 强连通分量缩点 (SCC)

```
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
13     void init(int n) {
14         this->n = n;
15         adj.assign(n, {});
16         dfn.assign(n, -1);
17         low.resize(n);
18         bel.assign(n, -1);
19         stk.clear();
20         cur = cnt = 0;
21     }
22
23     void addEdge(int u, int v) {
24         adj[u].push_back(v);
25     }
26
27     void dfs(int x) {
28         dfn[x] = low[x] = cur++;
29         stk.push_back(x);
30
31         for (auto y : adj[x]) {
32             if (dfn[y] == -1) {
33                 dfs(y);
34                 low[x] = std::min(low[x], low[y]);
35             } else if (bel[y] == -1) {
36                 low[x] = std::min(low[x], dfn[y]);
37             }
38         }
39
40         if (dfn[x] == low[x]) {
41             int y;
42             do {
43                 y = stk.back();
44                 bel[y] = cnt;
45                 stk.pop_back();
46             } while (y != x);
47             cnt++;
48         }
49     }
50
51     std::vector<int> work() {
52         for (int i = 0; i < n; i++) {
53             if (dfn[i] == -1) {
54                 dfs(i);
55             }
56         }
57         return bel;
58     }
```

2.2 割边与割边缩点 (EBCC)

```

1  std::set<std::pair<int, int>> E;
2
3  struct EBCC {
4      int n;
5      std::vector<std::vector<int>> adj;
6      std::vector<int> stk;
7      std::vector<int> dfn, low, bel;
8      int cur, cnt;
9
10     EBCC() {}
11     EBCC(int n) {
12         init(n);
13     }
14
15     void init(int n) {
16         this->n = n;
17         adj.assign(n, {});
18         dfn.assign(n, -1);
19         low.resize(n);
20         bel.assign(n, -1);
21         stk.clear();
22         cur = cnt = 0;
23     }
24
25     void addEdge(int u, int v) {
26         adj[u].push_back(v);
27         adj[v].push_back(u);
28     }
29
30     void dfs(int x, int p) {
31         dfn[x] = low[x] = cur++;
32         stk.push_back(x);
33
34         for (auto y : adj[x]) {
35             if (y == p) {
36                 continue;
37             }
38             if (dfn[y] == -1) {
39                 E.emplace(x, y);
40                 dfs(y, x);
41                 low[x] = std::min(low[x], low[y]);
42             } else if (bel[y] == -1 && dfn[y] < dfn[x]) {
43                 E.emplace(x, y);
44                 low[x] = std::min(low[x], dfn[y]);
45             }
46         }
47
48         if (dfn[x] == low[x]) {
49             int y;
50             do {
51                 y = stk.back();
52                 bel[y] = cnt;
53                 stk.pop_back();
54             } while (y != x);
55             cnt++;
56         }
57     }
58

```

```

59     std::vector<int> work() {
60         dfs(0, -1);
61         return bel;
62     }
63
64     struct Graph {
65         int n;
66         std::vector<std::pair<int, int>> edges;
67         std::vector<int> siz;
68         std::vector<int> cnte;
69     };
70     Graph compress() {
71         Graph g;
72         g.n = cnt;
73         g.siz.resize(cnt);
74         g.cnte.resize(cnt);
75         for (int i = 0; i < n; i++) {
76             g.siz[bel[i]]++;
77             for (auto j : adj[i]) {
78                 if (bel[i] < bel[j]) {
79                     g.edges.emplace_back(bel[i], bel[j]);
80                 } else if (i < j) {
81                     g.cnte[bel[i]]++;
82                 }
83             }
84         }
85         return g;
86     }
87 };

```

2.3 二分图最大权匹配 (MaxAssignment 基于KM) 【久远】

```

1  template<class T>
2  struct MaxAssignment {
3      public:
4          T solve(int nx, int ny, std::vector<std::vector<T>> a) {
5              assert(0 <= nx && nx <= ny);
6              assert(int(a.size()) == nx);
7              for (int i = 0; i < nx; ++i) {
8                  assert(int(a[i].size()) == ny);
9                  for (auto x : a[i])
10                     assert(x >= 0);
11              }
12
13              auto update = [&](int x) {
14                  for (int y = 0; y < ny; ++y) {
15                      if (lx[x] + ly[y] - a[x][y] < slack[y]) {
16                          slack[y] = lx[x] + ly[y] - a[x][y];
17                          slackx[y] = x;
18                      }
19                  }
20              };
21
22              costs.resize(nx + 1);
23              costs[0] = 0;
24              lx.assign(nx, std::numeric_limits<T>::max());
25              ly.assign(ny, 0);
26              xy.assign(nx, -1);
27              yx.assign(ny, -1);
28              slackx.resize(ny);
29              for (int cur = 0; cur < nx; ++cur) {
30                  std::queue<int> que;

```

```

31 visx.assign(nx, false);
32 visy.assign(ny, false);
33 slack.assign(ny, std::numeric_limits<T>::max());
34 p.assign(nx, -1);
35
36 for (int x = 0; x < nx; ++x) {
37     if (xy[x] == -1) {
38         que.push(x);
39         visx[x] = true;
40         update(x);
41     }
42 }
43
44 int ex, ey;
45 bool found = false;
46 while (!found) {
47     while (!que.empty() && !found) {
48         auto x = que.front();
49         que.pop();
50         for (int y = 0; y < ny; ++y) {
51             if (a[x][y] == lx[x] + ly[y] && !visy[y]) {
52                 if (yx[y] == -1) {
53                     ex = x;
54                     ey = y;
55                     found = true;
56                     break;
57                 }
58                 que.push(yx[y]);
59                 p[yx[y]] = x;
60                 visy[y] = visx[yx[y]] = true;
61                 update(yx[y]);
62             }
63         }
64     }
65     if (found)
66         break;
67
68     T delta = std::numeric_limits<T>::max();
69     for (int y = 0; y < ny; ++y)
70         if (!visy[y])
71             delta = std::min(delta, slack[y]);
72     for (int x = 0; x < nx; ++x)
73         if (visx[x])
74             lx[x] -= delta;
75     for (int y = 0; y < ny; ++y) {
76         if (visy[y]) {
77             ly[y] += delta;
78         } else {
79             slack[y] -= delta;
80         }
81     }
82     for (int y = 0; y < ny; ++y) {
83         if (!visy[y] && slack[y] == 0) {
84             if (yx[y] == -1) {
85                 ex = slackx[y];
86                 ey = y;
87                 found = true;
88                 break;
89             }
90             que.push(yx[y]);
91             p[yx[y]] = slackx[y];
92             visy[y] = visx[yx[y]] = true;
93             update(yx[y]);
94         }

```



```

95         }
96     }
97
98     costs[cur + 1] = costs[cur];
99     for (int x = ex, y = ey, ty; x != -1; x = p[x], y = ty) {
100         costs[cur + 1] += a[x][y];
101         if (xy[x] != -1)
102             costs[cur + 1] -= a[x][xy[x]];
103         ty = xy[x];
104         xy[x] = y;
105         yx[y] = x;
106     }
107 }
108 return costs[nx];
109 }
110 std::vector<int> assignment() {
111     return xy;
112 }
113 std::pair<std::vector<T>, std::vector<T>> labels() {
114     return std::make_pair(lx, ly);
115 }
116 std::vector<T> weights() {
117     return costs;
118 }
119 private:
120     std::vector<T> lx, ly, slack, costs;
121     std::vector<int> xy, yx, p, slackx;
122     std::vector<bool> visx, visy;
123 };

```

2.4 一般图最大匹配 (Graph 带花树算法) 【久远】

```

1 struct Graph {
2     int n;
3     std::vector<std::vector<int>> e;
4     Graph(int n) : n(n), e(n) {}
5     void addEdge(int u, int v) {
6         e[u].push_back(v);
7         e[v].push_back(u);
8     }
9     std::vector<int> findMatching() {
10         std::vector<int> match(n, -1), vis(n), link(n), f(n), dep(n);
11
12         // disjoint set union
13         auto find = [&](int u) {
14             while (f[u] != u)
15                 u = f[u] = f[f[u]];
16             return u;
17         };
18
19         auto lca = [&](int u, int v) {
20             u = find(u);
21             v = find(v);
22             while (u != v) {
23                 if (dep[u] < dep[v])
24                     std::swap(u, v);
25                 u = find(link[match[u]]);
26             }
27             return u;
28         };
29
30         std::queue<int> que;

```

```

31     auto blossom = [&](int u, int v, int p) {
32         while (find(u) != p) {
33             link[u] = v;
34             v = match[u];
35             if (vis[v] == 0) {
36                 vis[v] = 1;
37                 que.push(v);
38             }
39             f[u] = f[v] = p;
40             u = link[v];
41         }
42     };
43
44     // find an augmenting path starting from u and augment (if exist)
45     auto augment = [&](int u) {
46
47         while (!que.empty())
48             que.pop();
49
50         std::iota(f.begin(), f.end(), 0);
51
52         // vis = 0 corresponds to inner vertices, vis = 1 corresponds to outer
53         vertices
54         std::fill(vis.begin(), vis.end(), -1);
55
56         que.push(u);
57         vis[u] = 1;
58         dep[u] = 0;
59
60         while (!que.empty()){
61             int u = que.front();
62             que.pop();
63             for (auto v : e[u]) {
64                 if (vis[v] == -1) {
65
66                     vis[v] = 0;
67                     link[v] = u;
68                     dep[v] = dep[u] + 1;
69
70                     // found an augmenting path
71                     if (match[v] == -1) {
72                         for (int x = v, y = u, temp; y != -1; x = temp, y = x
73                             == -1 ? -1 : link[x]) {
74                             temp = match[y];
75                             match[x] = y;
76                             match[y] = x;
77                         }
78                         return;
79                     }
80                     vis[match[v]] = 1;
81                     dep[match[v]] = dep[u] + 2;
82                     que.push(match[v]);
83                 } else if (vis[v] == 1 && find(v) != find(u)) {
84                     // found a blossom
85                     int p = lca(u, v);
86                     blossom(u, v, p);
87                     blossom(v, u, p);
88                 }
89             }
90         }
91     };
92

```

```

93
94 // find a maximal matching greedily (decrease constant)
95 auto greedy = [&]() {
96
97     for (int u = 0; u < n; ++u) {
98         if (match[u] != -1)
99             continue;
100         for (auto v : e[u]) {
101             if (match[v] == -1) {
102                 match[u] = v;
103                 match[v] = u;
104                 break;
105             }
106         }
107     }
108 };
109
110 greedy();
111
112 for (int u = 0; u < n; ++u)
113     if (match[u] == -1)
114         augment(u);
115
116 return match;
117 }
118 };

```

2.5 TwoSat (2-Sat)

```

1 struct TwoSat {
2     int n;
3     std::vector<std::vector<int>> e;
4     std::vector<bool> ans;
5     TwoSat(int n) : n(n), e(2 * n), ans(n) {}
6     void addClause(int u, bool f, int v, bool g) {
7         e[2 * u + !f].push_back(2 * v + g);
8         e[2 * v + !g].push_back(2 * u + f);
9     }
10    bool satisfiable() {
11        std::vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 * n, -1);
12        std::vector<int> stk;
13        int now = 0, cnt = 0;
14        std::function<void(int)> tarjan = [&](int u) {
15            stk.push_back(u);
16            dfn[u] = low[u] = now++;
17            for (auto v : e[u]) {
18                if (dfn[v] == -1) {
19                    tarjan(v);
20                    low[u] = std::min(low[u], low[v]);
21                } else if (id[v] == -1) {
22                    low[u] = std::min(low[u], dfn[v]);
23                }
24            }
25            if (dfn[u] == low[u]) {
26                int v;
27                do {
28                    v = stk.back();
29                    stk.pop_back();
30                    id[v] = cnt;
31                } while (v != u);
32                ++cnt;
33            }

```

```

34     };
35     for (int i = 0; i < 2 * n; ++i) if (dfn[i] == -1) tarjan(i);
36     for (int i = 0; i < n; ++i) {
37         if (id[2 * i] == id[2 * i + 1]) return false;
38         ans[i] = id[2 * i] > id[2 * i + 1];
39     }
40     return true;
41 }
42 std::vector<bool> answer() { return ans; }
43 };

```

2.6 最大流

2.6.1 最大流 (Flow 旧版其一, 整数应用)

```

1  template<class T>
2  struct Flow {
3      const int n;
4      struct Edge {
5          int to;
6          T cap;
7          Edge(int to, T cap) : to(to), cap(cap) {}
8      };
9      std::vector<Edge> e;
10     std::vector<std::vector<int>> g;
11     std::vector<int> cur, h;
12     Flow(int n) : n(n), g(n) {}
13
14     bool bfs(int s, int t) {
15         h.assign(n, -1);
16         std::queue<int> que;
17         h[s] = 0;
18         que.push(s);
19         while (!que.empty()) {
20             const int u = que.front();
21             que.pop();
22             for (int i : g[u]) {
23                 auto [v, c] = e[i];
24                 if (c > 0 && h[v] == -1) {
25                     h[v] = h[u] + 1;
26                     if (v == t) {
27                         return true;
28                     }
29                     que.push(v);
30                 }
31             }
32         }
33         return false;
34     }
35
36     T dfs(int u, int t, T f) {
37         if (u == t) {
38             return f;
39         }
40         auto r = f;
41         for (int &i = cur[u]; i < int(g[u].size()); ++i) {
42             const int j = g[u][i];
43             auto [v, c] = e[j];
44             if (c > 0 && h[v] == h[u] + 1) {
45                 auto a = dfs(v, t, std::min(r, c));
46                 e[j].cap -= a;
47                 e[j ^ 1].cap += a;

```

```

48         r -= a;
49         if (r == 0) {
50             return f;
51         }
52     }
53 }
54 return f - r;
55 }
56 void addEdge(int u, int v, T c) {
57     g[u].push_back(e.size());
58     e.emplace_back(v, c);
59     g[v].push_back(e.size());
60     e.emplace_back(u, 0);
61 }
62 T maxFlow(int s, int t) {
63     T ans = 0;
64     while (bfs(s, t)) {
65         cur.assign(n, 0);
66         ans += dfs(s, t, std::numeric_limits<T>::max());
67     }
68     return ans;
69 }
70 };

```

2.6.2 最大流 (Flow 旧版其二, 浮点数应用)

```

1  template<class T>
2  struct Flow {
3      const int n;
4      struct Edge {
5          int to;
6          T cap;
7          Edge(int to, T cap) : to(to), cap(cap) {}
8      };
9      std::vector<Edge> e;
10     std::vector<std::vector<int>> g;
11     std::vector<int> cur, h;
12     Flow(int n) : n(n), g(n) {}
13
14     bool bfs(int s, int t) {
15         h.assign(n, -1);
16         std::queue<int> que;
17         h[s] = 0;
18         que.push(s);
19         while (!que.empty()) {
20             const int u = que.front();
21             que.pop();
22             for (int i : g[u]) {
23                 auto [v, c] = e[i];
24                 if (c > 0 && h[v] == -1) {
25                     h[v] = h[u] + 1;
26                     if (v == t) {
27                         return true;
28                     }
29                     que.push(v);
30                 }
31             }
32         }
33         return false;
34     }
35
36     T dfs(int u, int t, T f) {

```

```

37     if (u == t) {
38         return f;
39     }
40     auto r = f;
41     double res = 0;
42     for (int &i = cur[u]; i < int(g[u].size()); ++i) {
43         const int j = g[u][i];
44         auto [v, c] = e[j];
45         if (c > 0 && h[v] == h[u] + 1) {
46             auto a = dfs(v, t, std::min(r, c));
47             res += a;
48             e[j].cap -= a;
49             e[j ^ 1].cap += a;
50             r -= a;
51             if (r == 0) {
52                 return f;
53             }
54         }
55     }
56     return res;
57 }
58 void addEdge(int u, int v, T c) {
59     g[u].push_back(e.size());
60     e.emplace_back(v, c);
61     g[v].push_back(e.size());
62     e.emplace_back(u, 0);
63 }
64 T maxFlow(int s, int t) {
65     T ans = 0;
66     while (bfs(s, t)) {
67         cur.assign(n, 0);
68         ans += dfs(s, t, 1E100);
69     }
70     return ans;
71 }
72 };

```

2.6.3 最大流 (MaxFlow 新版)

```

1  constexpr int inf = 1E9;
2  template<class T>
3  struct MaxFlow {
4      struct _Edge {
5          int to;
6          T cap;
7          _Edge(int to, T cap) : to(to), cap(cap) {}
8      };
9
10     int n;
11     std::vector<_Edge> e;
12     std::vector<std::vector<int>> g;
13     std::vector<int> cur, h;
14
15     MaxFlow() {}
16     MaxFlow(int n) {
17         init(n);
18     }
19
20     void init(int n) {
21         this->n = n;
22         e.clear();
23         g.assign(n, {});

```

```

24     cur.resize(n);
25     h.resize(n);
26 }
27
28 bool bfs(int s, int t) {
29     h.assign(n, -1);
30     std::queue<int> que;
31     h[s] = 0;
32     que.push(s);
33     while (!que.empty()) {
34         const int u = que.front();
35         que.pop();
36         for (int i : g[u]) {
37             auto [v, c] = e[i];
38             if (c > 0 && h[v] == -1) {
39                 h[v] = h[u] + 1;
40                 if (v == t) {
41                     return true;
42                 }
43                 que.push(v);
44             }
45         }
46     }
47     return false;
48 }
49
50 T dfs(int u, int t, T f) {
51     if (u == t) {
52         return f;
53     }
54     auto r = f;
55     for (int &i = cur[u]; i < int(g[u].size()); ++i) {
56         const int j = g[u][i];
57         auto [v, c] = e[j];
58         if (c > 0 && h[v] == h[u] + 1) {
59             auto a = dfs(v, t, std::min(r, c));
60             e[j].cap -= a;
61             e[j ^ 1].cap += a;
62             r -= a;
63             if (r == 0) {
64                 return f;
65             }
66         }
67     }
68     return f - r;
69 }
70 void addEdge(int u, int v, T c) {
71     g[u].push_back(e.size());
72     e.emplace_back(v, c);
73     g[v].push_back(e.size());
74     e.emplace_back(u, 0);
75 }
76 T flow(int s, int t) {
77     T ans = 0;
78     while (bfs(s, t)) {
79         cur.assign(n, 0);
80         ans += dfs(s, t, std::numeric_limits<T>::max());
81     }
82     return ans;
83 }
84
85 std::vector<bool> minCut() {
86     std::vector<bool> c(n);
87     for (int i = 0; i < n; i++) {

```

```

88         c[i] = (h[i] != -1);
89     }
90     return c;
91 }
92
93 struct Edge {
94     int from;
95     int to;
96     T cap;
97     T flow;
98 };
99 std::vector<Edge> edges() {
100     std::vector<Edge> a;
101     for (int i = 0; i < e.size(); i += 2) {
102         Edge x;
103         x.from = e[i + 1].to;
104         x.to = e[i].to;
105         x.cap = e[i].cap + e[i + 1].cap;
106         x.flow = e[i + 1].cap;
107         a.push_back(x);
108     }
109     return a;
110 }
111 };

```

2.7 费用流

2.7.1 费用流 (MCFGraph 最小费用可行流)

```

1  struct MCFGraph {
2      struct Edge {
3          int v, c, f;
4          Edge(int v, int c, int f) : v(v), c(c), f(f) {}
5      };
6      const int n;
7      std::vector<Edge> e;
8      std::vector<std::vector<int>>> g;
9      std::vector<i64> h, dis;
10     std::vector<int> pre;
11     bool dijkstra(int s, int t) {
12         dis.assign(n, std::numeric_limits<i64>::max());
13         pre.assign(n, -1);
14         std::priority_queue<std::pair<i64, int>, std::vector<std::pair<i64, int>>,
std::greater<std::pair<i64, int>>> que;
15         dis[s] = 0;
16         que.emplace(0, s);
17         while (!que.empty()) {
18             i64 d = que.top().first;
19             int u = que.top().second;
20             que.pop();
21             if (dis[u] < d) continue;
22             for (int i : g[u]) {
23                 int v = e[i].v;
24                 int c = e[i].c;
25                 int f = e[i].f;
26                 if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
27                     dis[v] = d + h[u] - h[v] + f;
28                     pre[v] = i;
29                     que.emplace(dis[v], v);
30                 }
31             }
32         }
33     }
34 }

```



```

33         return dis[t] != std::numeric_limits<i64>::max();
34     }
35     MCFGGraph(int n) : n(n), g(n) {}
36     void addEdge(int u, int v, int c, int f) {
37         if (f < 0) {
38             g[u].push_back(e.size());
39             e.emplace_back(v, 0, f);
40             g[v].push_back(e.size());
41             e.emplace_back(u, c, -f);
42         } else {
43             g[u].push_back(e.size());
44             e.emplace_back(v, c, f);
45             g[v].push_back(e.size());
46             e.emplace_back(u, 0, -f);
47         }
48     }
49     std::pair<int, i64> flow(int s, int t) {
50         int flow = 0;
51         i64 cost = 0;
52         h.assign(n, 0);
53         while (dijkstra(s, t)) {
54             for (int i = 0; i < n; ++i) h[i] += dis[i];
55             int aug = std::numeric_limits<int>::max();
56             for (int i = t; i != s; i = e[pre[i] ^ 1].v) aug = std::min(aug,
e[pre[i]].c);
57             for (int i = t; i != s; i = e[pre[i] ^ 1].v) {
58                 e[pre[i]].c -= aug;
59                 e[pre[i] ^ 1].c += aug;
60             }
61             flow += aug;
62             cost += i64(aug) * h[t];
63         }
64         return std::make_pair(flow, cost);
65     }
66 };

```

2.7.2 费用流 (MCFGGraph 最小费用最大流)

代码同上，但是需要注释掉建边限制。以下为参考：

```

1 void addEdge(int u, int v, int c, int f) { // 可行流
2     if (f < 0) {
3         g[u].push_back(e.size());
4         e.emplace_back(v, 0, f);
5         g[v].push_back(e.size());
6         e.emplace_back(u, c, -f);
7     } else {
8         g[u].push_back(e.size());
9         e.emplace_back(v, c, f);
10        g[v].push_back(e.size());
11        e.emplace_back(u, 0, -f);
12    }
13 }

```

```

1 void addEdge(int u, int v, int c, int f) { // 最大流
2     g[u].push_back(e.size());
3     e.emplace_back(v, c, f);
4     g[v].push_back(e.size());
5     e.emplace_back(u, 0, -f);
6 }

```

2.8 树链剖分 (HLD)

```
1 struct HLD {
2     int n;
3     std::vector<int> siz, top, dep, parent, in, out, seq;
4     std::vector<std::vector<int>> adj;
5     int cur;
6
7     HLD() {}
8     HLD(int n) {
9         init(n);
10    }
11    void init(int n) {
12        this->n = n;
13        siz.resize(n);
14        top.resize(n);
15        dep.resize(n);
16        parent.resize(n);
17        in.resize(n);
18        out.resize(n);
19        seq.resize(n);
20        cur = 0;
21        adj.assign(n, {});
22    }
23    void addEdge(int u, int v) {
24        adj[u].push_back(v);
25        adj[v].push_back(u);
26    }
27    void work(int root = 0) {
28        top[root] = root;
29        dep[root] = 0;
30        parent[root] = -1;
31        dfs1(root);
32        dfs2(root);
33    }
34    void dfs1(int u) {
35        if (parent[u] != -1) {
36            adj[u].erase(std::find(adj[u].begin(), adj[u].end(), parent[u]));
37        }
38
39        siz[u] = 1;
40        for (auto &v : adj[u]) {
41            parent[v] = u;
42            dep[v] = dep[u] + 1;
43            dfs1(v);
44            siz[u] += siz[v];
45            if (siz[v] > siz[adj[u][0]]) {
46                std::swap(v, adj[u][0]);
47            }
48        }
49    }
50    void dfs2(int u) {
51        in[u] = cur++;
52        seq[in[u]] = u;
53        for (auto v : adj[u]) {
54            top[v] = v == adj[u][0] ? top[u] : v;
55            dfs2(v);
56        }
57        out[u] = cur;
58    }
59    int lca(int u, int v) {
60        while (top[u] != top[v]) {
61            if (dep[top[u]] > dep[top[v]]) {
```

```

62         u = parent[top[u]];
63     } else {
64         v = parent[top[v]];
65     }
66 }
67 return dep[u] < dep[v] ? u : v;
68 }
69
70 int dist(int u, int v) {
71     return dep[u] + dep[v] - 2 * dep[lca(u, v)];
72 }
73
74 int jump(int u, int k) {
75     if (dep[u] < k) {
76         return -1;
77     }
78
79     int d = dep[u] - k;
80
81     while (dep[top[u]] > d) {
82         u = parent[top[u]];
83     }
84
85     return seq[in[u] - dep[u] + d];
86 }
87
88 bool isAncestor(int u, int v) {
89     return in[u] <= in[v] && in[v] < out[u];
90 }
91
92 int rootedParent(int u, int v) {
93     std::swap(u, v);
94     if (u == v) {
95         return u;
96     }
97     if (!isAncestor(u, v)) {
98         return parent[u];
99     }
100     auto it = std::upper_bound(adj[u].begin(), adj[u].end(), v, [&](int x, int
y) {
101         return in[x] < in[y];
102     }) - 1;
103     return *it;
104 }
105
106 int rootedSize(int u, int v) {
107     if (u == v) {
108         return n;
109     }
110     if (!isAncestor(v, u)) {
111         return siz[v];
112     }
113     return n - siz[rootedParent(u, v)];
114 }
115
116 int rootedLca(int a, int b, int c) {
117     return lca(a, b) ^ lca(b, c) ^ lca(c, a);
118 }
119 };

```

/END/

3 数论、几何、多项式

3.1 快速幂

```
1 int power(int a, i64 b, int p) {
2     int res = 1;
3     for (; b; b /= 2, a = 1LL * a * a % p) {
4         if (b % 2) {
5             res = 1LL * res * a % p;
6         }
7     }
8     return res;
9 }
```

3.2 欧拉筛

```
1 std::vector<int> minp, primes;
2
3 void sieve(int n) {
4     minp.assign(n + 1, 0);
5     primes.clear();
6
7     for (int i = 2; i <= n; i++) {
8         if (minp[i] == 0) {
9             minp[i] = i;
10            primes.push_back(i);
11        }
12
13        for (auto p : primes) {
14            if (i * p > n) {
15                break;
16            }
17            minp[i * p] = p;
18            if (p == minp[i]) {
19                break;
20            }
21        }
22    }
23 }
```

3.3 莫比乌斯函数筛 (莫比乌斯函数/反演)

```
1 std::unordered_map<int, Z> fMu;
2
3 constexpr int N = 1E7;
4 std::vector<int> minp, primes;
5 std::vector<Z> mu;
6
7 void sieve(int n) {
8     minp.assign(n + 1, 0);
9     mu.resize(n);
10    primes.clear();
11
12    mu[1] = 1;
13    for (int i = 2; i <= n; i++) {
14        if (minp[i] == 0) {
15            mu[i] = -1;
16            minp[i] = i;
17            primes.push_back(i);
```

```

18     }
19
20     for (auto p : primes) {
21         if (i * p > n) {
22             break;
23         }
24         minp[i * p] = p;
25         if (p == minp[i]) {
26             break;
27         }
28         mu[i * p] = -mu[i];
29     }
30 }
31
32 for (int i = 1; i <= n; i++) {
33     mu[i] += mu[i - 1];
34 }
35 }
36
37
38 Z sumMu(int n) {
39     if (n <= N) {
40         return mu[n];
41     }
42     if (fMu.count(n)) {
43         return fMu[n];
44     }
45     if (n == 0) {
46         return 0;
47     }
48     Z ans = 1;
49     for (int l = 2, r; l <= n; l = r + 1) {
50         r = n / (n / l);
51         ans -= (r - l + 1) * sumMu(n / l);
52     }
53     return ans;
54 }
55
56 int main() {
57     std::ios::sync_with_stdio(false);
58     std::cin.tie(nullptr);
59
60     sieve(N);
61
62     int L, R;
63     std::cin >> L >> R;
64     L -= 1;
65
66     Z ans = 0;
67     for (int l = 1, r; l <= R; l = r + 1) {
68         r = R / (R / l);
69         if (l <= L) {
70             r = std::min(r, L / (L / l));
71         }
72
73         ans += (power(Z(2), R / l - L / l) - 1) * (sumMu(r) - sumMu(l - 1));
74     }
75
76     std::cout << ans << "\n";
77
78     return 0;
79 }

```

3.4 求解单个数的欧拉函数

```
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 }
```

3.5 扩展欧几里得 (exGCD)

```
1 int exgcd(int a, int b, int &x, int &y) {
2     if (!b) {
3         x = 1, y = 0;
4         return a;
5     }
6     int g = exgcd(b, a % b, y, x);
7     y -= a / b * x;
8     return g;
9 }
```

3.6 组合数 (Comb, with. MInt & MLong)

```
1 struct Comb {
2     int n;
3     std::vector<Z> _fac;
4     std::vector<Z> _invfac;
5     std::vector<Z> _inv;
6
7     Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
8     Comb(int n) : Comb() {
9         init(n);
10    }
11
12    void init(int m) {
13        m = std::min(m, Z::getMod() - 1);
14        if (m <= n) return;
15        _fac.resize(m + 1);
16        _invfac.resize(m + 1);
17        _inv.resize(m + 1);
18
19        for (int i = n + 1; i <= m; i++) {
20            _fac[i] = _fac[i - 1] * i;
21        }
22        _invfac[m] = _fac[m].inv();
23        for (int i = m; i > n; i--) {
24            _invfac[i - 1] = _invfac[i] * i;
25            _inv[i] = _invfac[i] * _fac[i - 1];
26        }
27        n = m;
28    }
29 }
```

```

29
30     Z fac(int m) {
31         if (m > n) init(2 * m);
32         return _fac[m];
33     }
34     Z invfac(int m) {
35         if (m > n) init(2 * m);
36         return _invfac[m];
37     }
38     Z inv(int m) {
39         if (m > n) init(2 * m);
40         return _inv[m];
41     }
42     Z binom(int n, int m) {
43         if (n < m || m < 0) return 0;
44         return fac(n) * invfac(m) * invfac(n - m);
45     }
46 } comb;

```

3.7 二项式 (Binomial 任意模数计算)

```

1  std::vector<std::pair<int, int>> factorize(int n) {
2      std::vector<std::pair<int, int>> factors;
3      for (int i = 2; static_cast<long long>(i) * i <= n; i++) {
4          if (n % i == 0) {
5              int t = 0;
6              for (; n % i == 0; n /= i)
7                  ++t;
8              factors.emplace_back(i, t);
9          }
10     }
11     if (n > 1)
12         factors.emplace_back(n, 1);
13     return factors;
14 }
15 constexpr int power(int base, i64 exp) {
16     int res = 1;
17     for (; exp > 0; base *= base, exp /= 2) {
18         if (exp % 2 == 1) {
19             res *= base;
20         }
21     }
22     return res;
23 }
24 constexpr int power(int base, i64 exp, int mod) {
25     int res = 1 % mod;
26     for (; exp > 0; base = 1LL * base * base % mod, exp /= 2) {
27         if (exp % 2 == 1) {
28             res = 1LL * res * base % mod;
29         }
30     }
31     return res;
32 }
33 int inverse(int a, int m) {
34     int g = m, r = a, x = 0, y = 1;
35     while (r != 0) {
36         int q = g / r;
37         g %= r;
38         std::swap(g, r);
39         x -= q * y;
40         std::swap(x, y);
41     }

```

```

42     return x < 0 ? x + m : x;
43 }
44 int solveModuloEquations(const std::vector<std::pair<int, int>> &e) {
45     int m = 1;
46     for (std::size_t i = 0; i < e.size(); i++) {
47         m *= e[i].first;
48     }
49     int res = 0;
50     for (std::size_t i = 0; i < e.size(); i++) {
51         int p = e[i].first;
52         res = (res + 1LL * e[i].second * (m / p) * inverse(m / p, p)) % m;
53     }
54     return res;
55 }
56 constexpr int N = 1E5;
57 class Binomial {
58     const int mod;
59 private:
60     const std::vector<std::pair<int, int>> factors;
61     std::vector<int> pk;
62     std::vector<std::vector<int>> prod;
63     static constexpr i64 exponent(i64 n, int p) {
64         i64 res = 0;
65         for (n /= p; n > 0; n /= p) {
66             res += n;
67         }
68         return res;
69     }
70     int product(i64 n, std::size_t i) {
71         int res = 1;
72         int p = factors[i].first;
73         for (; n > 0; n /= p) {
74             res = 1LL * res * power(prod[i].back(), n / pk[i], pk[i]) % pk[i] *
prod[i][n % pk[i]] % pk[i];
75         }
76         return res;
77     }
78 public:
79     Binomial(int mod) : mod(mod), factors(factorize(mod)) {
80         pk.resize(factors.size());
81         prod.resize(factors.size());
82         for (std::size_t i = 0; i < factors.size(); i++) {
83             int p = factors[i].first;
84             int k = factors[i].second;
85             pk[i] = power(p, k);
86             prod[i].resize(std::min(N + 1, pk[i]));
87             prod[i][0] = 1;
88             for (int j = 1; j < prod[i].size(); j++) {
89                 if (j % p == 0) {
90                     prod[i][j] = prod[i][j - 1];
91                 } else {
92                     prod[i][j] = 1LL * prod[i][j - 1] * j % pk[i];
93                 }
94             }
95         }
96     }
97     int operator()(i64 n, i64 m) {
98         if (n < m || m < 0) {
99             return 0;
100         }
101         std::vector<std::pair<int, int>> ans(factors.size());
102         for (int i = 0; i < factors.size(); i++) {
103             int p = factors[i].first;
104             int k = factors[i].second;

```



```

105         int e = exponent(n, p) - exponent(m, p) - exponent(n - m, p);
106         if (e >= k) {
107             ans[i] = std::make_pair(pk[i], 0);
108         } else {
109             int pn = product(n, i);
110             int pm = product(m, i);
111             int pd = product(n - m, i);
112             int res = 1LL * pn * inverse(pm, pk[i]) % pk[i] * inverse(pd,
pk[i]) % pk[i] * power(p, e) % pk[i];
113             ans[i] = std::make_pair(pk[i], res);
114         }
115     }
116     return solveModuloEquations(ans);
117 }
118 };

```

3.8 素数测试与因式分解 (Miller-Rabin & Pollard-Rho)

```

1  i64 mul(i64 a, i64 b, i64 m) {
2      return static_cast<__int128>(a) * b % m;
3  }
4  i64 power(i64 a, i64 b, i64 m) {
5      i64 res = 1 % m;
6      for (; b; b >>= 1, a = mul(a, a, m))
7          if (b & 1)
8              res = mul(res, a, m);
9      return res;
10 }
11 bool isprime(i64 n) {
12     if (n < 2)
13         return false;
14     static constexpr int A[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
15     int s = __builtin_ctzll(n - 1);
16     i64 d = (n - 1) >> s;
17     for (auto a : A) {
18         if (a == n)
19             return true;
20         i64 x = power(a, d, n);
21         if (x == 1 || x == n - 1)
22             continue;
23         bool ok = false;
24         for (int i = 0; i < s - 1; ++i) {
25             x = mul(x, x, n);
26             if (x == n - 1) {
27                 ok = true;
28                 break;
29             }
30         }
31         if (!ok)
32             return false;
33     }
34     return true;
35 }
36 std::vector<i64> factorize(i64 n) {
37     std::vector<i64> p;
38     std::function<void(i64)> f = [&](i64 n) {
39         if (n <= 10000) {
40             for (int i = 2; i * i <= n; ++i)
41                 for (; n % i == 0; n /= i)
42                     p.push_back(i);
43             if (n > 1)
44                 p.push_back(n);

```

```

45         return;
46     }
47     if (isprime(n)) {
48         p.push_back(n);
49         return;
50     }
51     auto g = [&](i64 x) {
52         return (mul(x, x, n) + 1) % n;
53     };
54     i64 x0 = 2;
55     while (true) {
56         i64 x = x0;
57         i64 y = x0;
58         i64 d = 1;
59         i64 power = 1, lam = 0;
60         i64 v = 1;
61         while (d == 1) {
62             y = g(y);
63             ++lam;
64             v = mul(v, std::abs(x - y), n);
65             if (lam % 127 == 0) {
66                 d = std::gcd(v, n);
67                 v = 1;
68             }
69             if (power == lam) {
70                 x = y;
71                 power *= 2;
72                 lam = 0;
73                 d = std::gcd(v, n);
74                 v = 1;
75             }
76         }
77         if (d != n) {
78             f(d);
79             f(n / d);
80             return;
81         }
82         ++x0;
83     }
84 };
85 f(n);
86 std::sort(p.begin(), p.end());
87 return p;
88 }

```

3.9 平面几何

```

1  template<class T>
2  struct Point {
3      T x;
4      T y;
5      Point(T x_ = 0, 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+=(Point p) & {
12         x += p.x;
13         y += p.y;
14         return *this;
15     }

```

```

16     Point &operator--(Point p) & {
17         x -= p.x;
18         y -= p.y;
19         return *this;
20     }
21     Point &operator*=(T v) & {
22         x *= v;
23         y *= v;
24         return *this;
25     }
26     Point operator-() const {
27         return Point(-x, -y);
28     }
29     friend Point operator+(Point a, Point b) {
30         return a += b;
31     }
32     friend Point operator-(Point a, Point b) {
33         return a -= b;
34     }
35     friend Point operator*(Point a, T b) {
36         return a *= b;
37     }
38     friend Point operator*(T a, Point b) {
39         return b *= a;
40     }
41     friend bool operator==(Point a, Point b) {
42         return a.x == b.x && a.y == b.y;
43     }
44     friend std::istream &operator>>(std::istream &is, Point &p) {
45         return is >> p.x >> p.y;
46     }
47     friend std::ostream &operator<<(std::ostream &os, Point p) {
48         return os << "(" << p.x << ", " << p.y << ")";
49     }
50 };
51
52 template<class T>
53 T dot(Point<T> a, Point<T> b) {
54     return a.x * b.x + a.y * b.y;
55 }
56
57 template<class T>
58 T cross(Point<T> a, Point<T> b) {
59     return a.x * b.y - a.y * b.x;
60 }
61
62 template<class T>
63 T square(Point<T> p) {
64     return dot(p, p);
65 }
66
67 template<class T>
68 double length(Point<T> p) {
69     return std::sqrt(double(square(p)));
70 }
71
72 long double length(Point<long double> p) {
73     return std::sqrt(square(p));
74 }
75
76 template<class T>
77 struct Line {
78     Point<T> a;
79     Point<T> b;

```

```

80     Line(Point<T> a_ = Point<T>(), Point<T> b_ = Point<T>()) : a(a_), b(b_) {}
81 };
82
83 template<class T>
84 Point<T> rotate(Point<T> a) {
85     return Point(-a.y, a.x);
86 }
87
88 template<class T>
89 int sgn(Point<T> a) {
90     return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
91 }
92
93 template<class T>
94 bool pointOnLineLeft(Point<T> p, Line<T> l) {
95     return cross(l.b - l.a, p - l.a) > 0;
96 }
97
98 template<class T>
99 Point<T> lineIntersection(Line<T> l1, Line<T> l2) {
100     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b -
101     l2.a, l1.a - l1.b));
102 }
103
104 template<class T>
105 bool pointOnSegment(Point<T> p, Line<T> l) {
106     return cross(p - l.a, l.b - l.a) == 0 && std::min(l.a.x, l.b.x) <= p.x && p.x
107     <= std::max(l.a.x, l.b.x)
108     && std::min(l.a.y, l.b.y) <= p.y && p.y <= std::max(l.a.y, l.b.y);
109 }
110
111 template<class T>
112 bool pointInPolygon(Point<T> a, std::vector<Point<T>> p) {
113     int n = p.size();
114     for (int i = 0; i < n; i++) {
115         if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
116             return true;
117         }
118     }
119
120     int t = 0;
121     for (int i = 0; i < n; i++) {
122         auto u = p[i];
123         auto v = p[(i + 1) % n];
124         if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
125             t ^= 1;
126         }
127         if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
128             t ^= 1;
129         }
130     }
131     return t == 1;
132 }
133
134 // 0 : not intersect
135 // 1 : strictly intersect
136 // 2 : overlap
137 // 3 : intersect at endpoint
138 template<class T>
139 std::tuple<int, Point<T>, Point<T>> segmentIntersection(Line<T> l1, Line<T> l2) {
140     if (std::max(l1.a.x, l1.b.x) < std::min(l2.a.x, l2.b.x)) {
141         return {0, Point<T>(), Point<T>()};
142     }

```

```

142     if (std::min(l1.a.x, l1.b.x) > std::max(l2.a.x, l2.b.x)) {
143         return {0, Point<T>(), Point<T>()};
144     }
145     if (std::max(l1.a.y, l1.b.y) < std::min(l2.a.y, l2.b.y)) {
146         return {0, Point<T>(), Point<T>()};
147     }
148     if (std::min(l1.a.y, l1.b.y) > std::max(l2.a.y, l2.b.y)) {
149         return {0, Point<T>(), Point<T>()};
150     }
151     if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
152         if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
153             return {0, Point<T>(), Point<T>()};
154         } else {
155             auto maxx1 = std::max(l1.a.x, l1.b.x);
156             auto minx1 = std::min(l1.a.x, l1.b.x);
157             auto maxy1 = std::max(l1.a.y, l1.b.y);
158             auto miny1 = std::min(l1.a.y, l1.b.y);
159             auto maxx2 = std::max(l2.a.x, l2.b.x);
160             auto minx2 = std::min(l2.a.x, l2.b.x);
161             auto maxy2 = std::max(l2.a.y, l2.b.y);
162             auto miny2 = std::min(l2.a.y, l2.b.y);
163             Point<T> p1(std::max(minx1, minx2), std::max(miny1, miny2));
164             Point<T> p2(std::min(maxx1, maxx2), std::min(maxy1, maxy2));
165             if (!pointOnSegment(p1, l1)) {
166                 std::swap(p1.y, p2.y);
167             }
168             if (p1 == p2) {
169                 return {3, p1, p2};
170             } else {
171                 return {2, p1, p2};
172             }
173         }
174     }
175     auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
176     auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
177     auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
178     auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);
179
180     if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0) ||
181         (cp3 < 0 && cp4 < 0)) {
182         return {0, Point<T>(), Point<T>()};
183     }
184
185     Point p = lineIntersection(l1, l2);
186     if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
187         return {1, p, p};
188     } else {
189         return {3, p, p};
190     }
191 }
192
193 template<class T>
194 bool segmentInPolygon(Line<T> l, std::vector<Point<T>> p) {
195     int n = p.size();
196     if (!pointInPolygon(l.a, p)) {
197         return false;
198     }
199     if (!pointInPolygon(l.b, p)) {
200         return false;
201     }
202     for (int i = 0; i < n; i++) {
203         auto u = p[i];
204         auto v = p[(i + 1) % n];
205         auto w = p[(i + 2) % n];

```

```

205     auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
206
207     if (t == 1) {
208         return false;
209     }
210     if (t == 0) {
211         continue;
212     }
213     if (t == 2) {
214         if (pointOnSegment(v, l) && v != l.a && v != l.b) {
215             if (cross(v - u, w - v) > 0) {
216                 return false;
217             }
218         }
219     } else {
220         if (p1 != u && p1 != v) {
221             if (pointOnLineLeft(l.a, Line(v, u))
222                 || pointOnLineLeft(l.b, Line(v, u))) {
223                 return false;
224             }
225         } else if (p1 == v) {
226             if (l.a == v) {
227                 if (pointOnLineLeft(u, l)) {
228                     if (pointOnLineLeft(w, l)
229                         && pointOnLineLeft(w, Line(u, v))) {
230                         return false;
231                     }
232                 } else {
233                     if (pointOnLineLeft(w, l)
234                         || pointOnLineLeft(w, Line(u, v))) {
235                         return false;
236                     }
237                 }
238             } else if (l.b == v) {
239                 if (pointOnLineLeft(u, Line(l.b, l.a))) {
240                     if (pointOnLineLeft(w, Line(l.b, l.a))
241                         && pointOnLineLeft(w, Line(u, v))) {
242                         return false;
243                     }
244                 } else {
245                     if (pointOnLineLeft(w, Line(l.b, l.a))
246                         || pointOnLineLeft(w, Line(u, v))) {
247                         return false;
248                     }
249                 }
250             } else {
251                 if (pointOnLineLeft(u, l)) {
252                     if (pointOnLineLeft(w, Line(l.b, l.a))
253                         || pointOnLineLeft(w, Line(u, v))) {
254                         return false;
255                     }
256                 } else {
257                     if (pointOnLineLeft(w, l)
258                         || pointOnLineLeft(w, Line(u, v))) {
259                         return false;
260                     }
261                 }
262             }
263         }
264     }
265 }
266 return true;
267 }
268

```

```

269 template<class T>
270 std::vector<Point<T>> hp(std::vector<Line<T>> lines) {
271     std::sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
272         auto d1 = l1.b - l1.a;
273         auto d2 = l2.b - l2.a;
274
275         if (sgn(d1) != sgn(d2)) {
276             return sgn(d1) == 1;
277         }
278
279         return cross(d1, d2) > 0;
280     });
281
282     std::deque<Line<T>> ls;
283     std::deque<Point<T>> ps;
284     for (auto l : lines) {
285         if (ls.empty()) {
286             ls.push_back(l);
287             continue;
288         }
289
290         while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
291             ps.pop_back();
292             ls.pop_back();
293         }
294
295         while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
296             ps.pop_front();
297             ls.pop_front();
298         }
299
300         if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
301             if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
302
303                 if (!pointOnLineLeft(ls.back().a, l)) {
304                     assert(ls.size() == 1);
305                     ls[0] = l;
306                 }
307                 continue;
308             }
309             return {};
310         }
311
312         ps.push_back(lineIntersection(ls.back(), l));
313         ls.push_back(l);
314     }
315
316     while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
317         ps.pop_back();
318         ls.pop_back();
319     }
320     if (ls.size() <= 2) {
321         return {};
322     }
323     ps.push_back(lineIntersection(ls[0], ls.back()));
324
325     return std::vector(ps.begin(), ps.end());
326 }

```

3.10 静态凸包

3.10.1 静态凸包 (with. Point)

```
1 struct Point {
2     i64 x;
3     i64 y;
4     Point(i64 x = 0, i64 y = 0) : x(x), y(y) {}
5 };
6
7 bool operator==(const Point &a, const Point &b) {
8     return a.x == b.x && a.y == b.y;
9 }
10
11 Point operator+(const Point &a, const Point &b) {
12     return Point(a.x + b.x, a.y + b.y);
13 }
14
15 Point operator-(const Point &a, const Point &b) {
16     return Point(a.x - b.x, a.y - b.y);
17 }
18
19 i64 dot(const Point &a, const Point &b) {
20     return a.x * b.x + a.y * b.y;
21 }
22
23 i64 cross(const Point &a, const Point &b) {
24     return a.x * b.y - a.y * b.x;
25 }
26
27 void norm(std::vector<Point> &h) {
28     int i = 0;
29     for (int j = 0; j < int(h.size()); j++) {
30         if (h[j].y < h[i].y || (h[j].y == h[i].y && h[j].x < h[i].x)) {
31             i = j;
32         }
33     }
34     std::rotate(h.begin(), h.begin() + i, h.end());
35 }
36
37 int sgn(const Point &a) {
38     return a.y > 0 || (a.y == 0 && a.x > 0) ? 0 : 1;
39 }
40
41 std::vector<Point> getHull(std::vector<Point> p) {
42     std::vector<Point> h, l;
43     std::sort(p.begin(), p.end(), [&](auto a, auto b) {
44         if (a.x != b.x) {
45             return a.x < b.x;
46         } else {
47             return a.y < b.y;
48         }
49     });
50     p.erase(std::unique(p.begin(), p.end()), p.end());
51     if (p.size() <= 1) {
52         return p;
53     }
54
55     for (auto a : p) {
56         while (h.size() > 1 && cross(a - h.back(), a - h[h.size() - 2]) <= 0) {
57             h.pop_back();
58         }
59         while (l.size() > 1 && cross(a - l.back(), a - l[l.size() - 2]) >= 0) {
```



```

60         l.pop_back();
61     }
62     l.push_back(a);
63     h.push_back(a);
64 }
65
66 l.pop_back();
67 std::reverse(h.begin(), h.end());
68 h.pop_back();
69 l.insert(l.end(), h.begin(), h.end());
70 return l;
71 }

```

3.10.2 静态凸包 (with. std::complex)

```

1  using Point = std::complex<i64>;
2
3  #define x real
4  #define y imag
5
6  auto dot(const Point &a, const Point &b) {
7      return (std::conj(a) * b).x();
8  }
9
10 auto cross(const Point &a, const Point &b) {
11     return (std::conj(a) * b).y();
12 }
13
14 auto rot(const Point &p) {
15     return Point(-p.y(), p.x());
16 }
17
18 auto complexHull(std::vector<Point> a) {
19     std::sort(a.begin(), a.end(), [&](auto a, auto b) {
20         if (a.x() != b.x()) {
21             return a.x() < b.x();
22         } else {
23             return a.y() < b.y();
24         }
25     });
26
27     std::vector<Point> l, h;
28
29     for (auto p : a) {
30         while (l.size() > 1 && cross(l.back() - l[l.size() - 2], p - l.back()) <= 0)
31         {
32             l.pop_back();
33         }
34         while (h.size() > 1 && cross(h.back() - h[h.size() - 2], p - h.back()) >= 0)
35         {
36             h.pop_back();
37         }
38         l.push_back(p);
39         h.push_back(p);
40     }
41
42     std::reverse(h.begin(), h.end());
43
44     h.insert(h.end(), l.begin() + 1, l.end() - 1);
45

```

```

46     return h;
47 }
48
49 int sgn(Point p) {
50     if (p.y() > 0 || (p.y() == 0 && p.x() < 0)) {
51         return 0;
52     } else {
53         return 1;
54     }
55 }

```

3.11 多项式相关

3.11.1 多项式相关 (Poly, with. Z)

```

1  std::vector<int> rev;
2  std::vector<Z> roots{0, 1};
3  void dft(std::vector<Z> &a) {
4      int n = a.size();
5
6      if (int(rev.size()) != n) {
7          int k = __builtin_ctz(n) - 1;
8          rev.resize(n);
9          for (int i = 0; i < n; i++) {
10             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
11         }
12     }
13
14     for (int i = 0; i < n; i++) {
15         if (rev[i] < i) {
16             std::swap(a[i], a[rev[i]]);
17         }
18     }
19     if (int(roots.size()) < n) {
20         int k = __builtin_ctz(roots.size());
21         roots.resize(n);
22         while ((1 << k) < n) {
23             Z e = power(Z(3), (P - 1) >> (k + 1));
24             for (int i = 1 << (k - 1); i < (1 << k); i++) {
25                 roots[2 * i] = roots[i];
26                 roots[2 * i + 1] = roots[i] * e;
27             }
28             k++;
29         }
30     }
31     for (int k = 1; k < n; k *= 2) {
32         for (int i = 0; i < n; i += 2 * k) {
33             for (int j = 0; j < k; j++) {
34                 Z u = a[i + j];
35                 Z v = a[i + j + k] * roots[k + j];
36                 a[i + j] = u + v;
37                 a[i + j + k] = u - v;
38             }
39         }
40     }
41 }
42 void idft(std::vector<Z> &a) {
43     int n = a.size();
44     std::reverse(a.begin() + 1, a.end());
45     dft(a);
46     Z inv = (1 - P) / n;
47     for (int i = 0; i < n; i++) {

```

```

48     a[i] *= inv;
49 }
50 }
51 struct Poly {
52     std::vector<Z> a;
53     Poly() {}
54     explicit Poly(int size, std::function<Z(int)> f = [](int) { return 0; }) :
a(size) {
55         for (int i = 0; i < size; i++) {
56             a[i] = f(i);
57         }
58     }
59     Poly(const std::vector<Z> &a) : a(a) {}
60     Poly(const std::initializer_list<Z> &a) : a(a) {}
61     int size() const {
62         return a.size();
63     }
64     void resize(int n) {
65         a.resize(n);
66     }
67     Z operator[](int idx) const {
68         if (idx < size()) {
69             return a[idx];
70         } else {
71             return 0;
72         }
73     }
74     Z &operator[](int idx) {
75         return a[idx];
76     }
77     Poly mulxk(int k) const {
78         auto b = a;
79         b.insert(b.begin(), k, 0);
80         return Poly(b);
81     }
82     Poly modxk(int k) const {
83         k = std::min(k, size());
84         return Poly(std::vector<Z>(a.begin(), a.begin() + k));
85     }
86     Poly divxk(int k) const {
87         if (size() <= k) {
88             return Poly();
89         }
90         return Poly(std::vector<Z>(a.begin() + k, a.end()));
91     }
92     friend Poly operator+(const Poly &a, const Poly &b) {
93         std::vector<Z> res(std::max(a.size(), b.size()));
94         for (int i = 0; i < int(res.size()); i++) {
95             res[i] = a[i] + b[i];
96         }
97         return Poly(res);
98     }
99     friend Poly operator-(const Poly &a, const Poly &b) {
100         std::vector<Z> res(std::max(a.size(), b.size()));
101         for (int i = 0; i < int(res.size()); i++) {
102             res[i] = a[i] - b[i];
103         }
104         return Poly(res);
105     }
106     friend Poly operator-(const Poly &a) {
107         std::vector<Z> res(a.size());
108         for (int i = 0; i < int(res.size()); i++) {
109             res[i] = -a[i];
110         }

```

```

111     return Poly(res);
112 }
113 friend Poly operator*(Poly a, Poly b) {
114     if (a.size() == 0 || b.size() == 0) {
115         return Poly();
116     }
117     if (a.size() < b.size()) {
118         std::swap(a, b);
119     }
120     if (b.size() < 128) {
121         Poly c(a.size() + b.size() - 1);
122         for (int i = 0; i < a.size(); i++) {
123             for (int j = 0; j < b.size(); j++) {
124                 c[i + j] += a[i] * b[j];
125             }
126         }
127         return c;
128     }
129     int sz = 1, tot = a.size() + b.size() - 1;
130     while (sz < tot) {
131         sz *= 2;
132     }
133     a.a.resize(sz);
134     b.a.resize(sz);
135     dft(a.a);
136     dft(b.a);
137     for (int i = 0; i < sz; ++i) {
138         a.a[i] = a[i] * b[i];
139     }
140     idft(a.a);
141     a.resize(tot);
142     return a;
143 }
144 friend Poly operator*(Z a, Poly b) {
145     for (int i = 0; i < int(b.size()); i++) {
146         b[i] *= a;
147     }
148     return b;
149 }
150 friend Poly operator*(Poly a, Z b) {
151     for (int i = 0; i < int(a.size()); i++) {
152         a[i] *= b;
153     }
154     return a;
155 }
156 Poly &operator+=(Poly b) {
157     return (*this) = (*this) + b;
158 }
159 Poly &operator-=(Poly b) {
160     return (*this) = (*this) - b;
161 }
162 Poly &operator*=(Poly b) {
163     return (*this) = (*this) * b;
164 }
165 Poly &operator*=(Z b) {
166     return (*this) = (*this) * b;
167 }
168 Poly deriv() const {
169     if (a.empty()) {
170         return Poly();
171     }
172     std::vector<Z> res(size() - 1);
173     for (int i = 0; i < size() - 1; ++i) {
174         res[i] = (i + 1) * a[i + 1];

```

```

175     }
176     return Poly(res);
177 }
178 Poly integr() const {
179     std::vector<Z> res(size() + 1);
180     for (int i = 0; i < size(); ++i) {
181         res[i + 1] = a[i] / (i + 1);
182     }
183     return Poly(res);
184 }
185 Poly inv(int m) const {
186     Poly x{a[0].inv()};
187     int k = 1;
188     while (k < m) {
189         k *= 2;
190         x = (x * (Poly{2} - modxk(k) * x)).modxk(k);
191     }
192     return x.modxk(m);
193 }
194 Poly log(int m) const {
195     return (deriv() * inv(m)).integr().modxk(m);
196 }
197 Poly exp(int m) const {
198     Poly x{1};
199     int k = 1;
200     while (k < m) {
201         k *= 2;
202         x = (x * (Poly{1} - x.log(k) + modxk(k))).modxk(k);
203     }
204     return x.modxk(m);
205 }
206 Poly pow(int k, int m) const {
207     int i = 0;
208     while (i < size() && a[i].val() == 0) {
209         i++;
210     }
211     if (i == size() || 1LL * i * k >= m) {
212         return Poly(std::vector<Z>(m));
213     }
214     Z v = a[i];
215     auto f = divxk(i) * v.inv();
216     return (f.log(m - i * k) * k).exp(m - i * k).mulxk(i * k) * power(v, k);
217 }
218 Poly sqrt(int m) const {
219     Poly x{1};
220     int k = 1;
221     while (k < m) {
222         k *= 2;
223         x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((P + 1) / 2);
224     }
225     return x.modxk(m);
226 }
227 Poly mult(Poly b) const {
228     if (b.size() == 0) {
229         return Poly();
230     }
231     int n = b.size();
232     std::reverse(b.a.begin(), b.a.end());
233     return ((*this) * b).divxk(n - 1);
234 }
235 std::vector<Z> eval(std::vector<Z> x) const {
236     if (size() == 0) {
237         return std::vector<Z>(x.size(), 0);
238     }

```

```

239     const int n = std::max(int(x.size()), size());
240     std::vector<Poly> q(4 * n);
241     std::vector<Z> ans(x.size());
242     x.resize(n);
243     std::function<void(int, int, int)> build = [&](int p, int l, int r) {
244         if (r - l == 1) {
245             q[p] = Poly{1, -x[l]};
246         } else {
247             int m = (l + r) / 2;
248             build(2 * p, l, m);
249             build(2 * p + 1, m, r);
250             q[p] = q[2 * p] * q[2 * p + 1];
251         }
252     };
253     build(1, 0, n);
254     std::function<void(int, int, int, const Poly &)> work = [&](int p, int l,
int r, const Poly &num) {
255         if (r - l == 1) {
256             if (l < int(ans.size())) {
257                 ans[l] = num[0];
258             }
259         } else {
260             int m = (l + r) / 2;
261             work(2 * p, l, m, num.mulT(q[2 * p + 1]).modxk(m - 1));
262             work(2 * p + 1, m, r, num.mulT(q[2 * p]).modxk(r - m));
263         }
264     };
265     work(1, 0, n, mulT(q[1].inv(n)));
266     return ans;
267 }
268 };

```

3.12 多项式相关 (Poly, with. MInt & MLong)

```

1  std::vector<int> rev;
2  template<int P>
3  std::vector<MInt<P>> roots{0, 1};
4
5  template<int P>
6  constexpr MInt<P> findPrimitiveRoot() {
7      MInt<P> i = 2;
8      int k = __builtin_ctz(P - 1);
9      while (true) {
10         if (power(i, (P - 1) / 2) != 1) {
11             break;
12         }
13         i += 1;
14     }
15     return power(i, (P - 1) >> k);
16 }
17
18 template<int P>
19 constexpr MInt<P> primitiveRoot = findPrimitiveRoot<P>();
20
21 template<>
22 constexpr MInt<998244353> primitiveRoot<998244353> {31};
23
24 template<int P>
25 constexpr void dft(std::vector<MInt<P>> &a) {
26     int n = a.size();
27
28     if (int(rev.size()) != n) {

```

```

29     int k = __builtin_ctz(n) - 1;
30     rev.resize(n);
31     for (int i = 0; i < n; i++) {
32         rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
33     }
34 }
35
36 for (int i = 0; i < n; i++) {
37     if (rev[i] < i) {
38         std::swap(a[i], a[rev[i]]);
39     }
40 }
41 if (roots<P>.size() < n) {
42     int k = __builtin_ctz(roots<P>.size());
43     roots<P>.resize(n);
44     while ((1 << k) < n) {
45         auto e = power(primitiveRoot<P>, 1 << (__builtin_ctz(P - 1) - k - 1));
46         for (int i = 1 << (k - 1); i < (1 << k); i++) {
47             roots<P>[2 * i] = roots<P>[i];
48             roots<P>[2 * i + 1] = roots<P>[i] * e;
49         }
50         k++;
51     }
52 }
53 for (int k = 1; k < n; k *= 2) {
54     for (int i = 0; i < n; i += 2 * k) {
55         for (int j = 0; j < k; j++) {
56             MInt<P> u = a[i + j];
57             MInt<P> v = a[i + j + k] * roots<P>[k + j];
58             a[i + j] = u + v;
59             a[i + j + k] = u - v;
60         }
61     }
62 }
63 }
64
65 template<int P>
66 constexpr void idft(std::vector<MInt<P>> &a) {
67     int n = a.size();
68     std::reverse(a.begin() + 1, a.end());
69     dft(a);
70     MInt<P> inv = (1 - P) / n;
71     for (int i = 0; i < n; i++) {
72         a[i] *= inv;
73     }
74 }
75
76 template<int P = 998244353>
77 struct Poly : public std::vector<MInt<P>> {
78     using Value = MInt<P>;
79
80     Poly() : std::vector<Value>() {}
81     explicit constexpr Poly(int n) : std::vector<Value>(n) {}
82
83     explicit constexpr Poly(const std::vector<Value> &a) : std::vector<Value>(a) {}
84     constexpr Poly(const std::initializer_list<Value> &a) : std::vector<Value>(a) {}
85 }
86
87 template<class InputIt, class = std::RequireInputIter<InputIt>>
88 explicit constexpr Poly(InputIt first, InputIt last) : std::vector<Value>
89 (first, last) {}
90
91 template<class F>
92 explicit constexpr Poly(int n, F f) : std::vector<Value>(n) {

```

```

91     for (int i = 0; i < n; i++) {
92         (*this)[i] = f(i);
93     }
94 }
95
96 constexpr Poly shift(int k) const {
97     if (k >= 0) {
98         auto b = *this;
99         b.insert(b.begin(), k, 0);
100        return b;
101    } else if (this->size() <= -k) {
102        return Poly();
103    } else {
104        return Poly(this->begin() + (-k), this->end());
105    }
106 }
107 constexpr Poly trunc(int k) const {
108     Poly f = *this;
109     f.resize(k);
110     return f;
111 }
112 constexpr friend Poly operator+(const Poly &a, const Poly &b) {
113     Poly res(std::max(a.size(), b.size()));
114     for (int i = 0; i < a.size(); i++) {
115         res[i] += a[i];
116     }
117     for (int i = 0; i < b.size(); i++) {
118         res[i] += b[i];
119     }
120     return res;
121 }
122 constexpr friend Poly operator-(const Poly &a, const Poly &b) {
123     Poly res(std::max(a.size(), b.size()));
124     for (int i = 0; i < a.size(); i++) {
125         res[i] += a[i];
126     }
127     for (int i = 0; i < b.size(); i++) {
128         res[i] -= b[i];
129     }
130     return res;
131 }
132 constexpr friend Poly operator-(const Poly &a) {
133     std::vector<Value> res(a.size());
134     for (int i = 0; i < int(res.size()); i++) {
135         res[i] = -a[i];
136     }
137     return Poly(res);
138 }
139 constexpr friend Poly operator*(Poly a, Poly b) {
140     if (a.size() == 0 || b.size() == 0) {
141         return Poly();
142     }
143     if (a.size() < b.size()) {
144         std::swap(a, b);
145     }
146     int n = 1, tot = a.size() + b.size() - 1;
147     while (n < tot) {
148         n *= 2;
149     }
150     if (((P - 1) & (n - 1)) != 0 || b.size() < 128) {
151         Poly c(a.size() + b.size() - 1);
152         for (int i = 0; i < a.size(); i++) {
153             for (int j = 0; j < b.size(); j++) {
154                 c[i + j] += a[i] * b[j];

```



```

155         }
156     }
157     return c;
158 }
159 a.resize(n);
160 b.resize(n);
161 dft(a);
162 dft(b);
163 for (int i = 0; i < n; ++i) {
164     a[i] *= b[i];
165 }
166 idft(a);
167 a.resize(tot);
168 return a;
169 }
170 constexpr friend Poly operator*(Value a, Poly b) {
171     for (int i = 0; i < int(b.size()); i++) {
172         b[i] *= a;
173     }
174     return b;
175 }
176 constexpr friend Poly operator*(Poly a, Value b) {
177     for (int i = 0; i < int(a.size()); i++) {
178         a[i] *= b;
179     }
180     return a;
181 }
182 constexpr friend Poly operator/(Poly a, Value b) {
183     for (int i = 0; i < int(a.size()); i++) {
184         a[i] /= b;
185     }
186     return a;
187 }
188 constexpr Poly &operator+=(Poly b) {
189     return (*this) = (*this) + b;
190 }
191 constexpr Poly &operator-=(Poly b) {
192     return (*this) = (*this) - b;
193 }
194 constexpr Poly &operator*=(Poly b) {
195     return (*this) = (*this) * b;
196 }
197 constexpr Poly &operator*=(Value b) {
198     return (*this) = (*this) * b;
199 }
200 constexpr Poly &operator/=(Value b) {
201     return (*this) = (*this) / b;
202 }
203 constexpr Poly deriv() const {
204     if (this->empty()) {
205         return Poly();
206     }
207     Poly res(this->size() - 1);
208     for (int i = 0; i < this->size() - 1; ++i) {
209         res[i] = (i + 1) * (*this)[i + 1];
210     }
211     return res;
212 }
213 constexpr Poly integr() const {
214     Poly res(this->size() + 1);
215     for (int i = 0; i < this->size(); ++i) {
216         res[i + 1] = (*this)[i] / (i + 1);
217     }
218     return res;

```

```

219     }
220     constexpr Poly inv(int m) const {
221         Poly x{(*this)[0].inv()};
222         int k = 1;
223         while (k < m) {
224             k *= 2;
225             x = (x * (Poly{2} - trunc(k) * x)).trunc(k);
226         }
227         return x.trunc(m);
228     }
229     constexpr Poly log(int m) const {
230         return (deriv() * inv(m)).integr().trunc(m);
231     }
232     constexpr Poly exp(int m) const {
233         Poly x{1};
234         int k = 1;
235         while (k < m) {
236             k *= 2;
237             x = (x * (Poly{1} - x.log(k) + trunc(k))).trunc(k);
238         }
239         return x.trunc(m);
240     }
241     constexpr Poly pow(int k, int m) const {
242         int i = 0;
243         while (i < this->size() && (*this)[i] == 0) {
244             i++;
245         }
246         if (i == this->size() || 1LL * i * k >= m) {
247             return Poly(m);
248         }
249         Value v = (*this)[i];
250         auto f = shift(-i) * v.inv();
251         return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * power(v, k);
252     }
253     constexpr Poly sqrt(int m) const {
254         Poly x{1};
255         int k = 1;
256         while (k < m) {
257             k *= 2;
258             x = (x + (trunc(k) * x.inv(k)).trunc(k)) * CInv<2, P>;
259         }
260         return x.trunc(m);
261     }
262     constexpr Poly mulT(Poly b) const {
263         if (b.size() == 0) {
264             return Poly();
265         }
266         int n = b.size();
267         std::reverse(b.begin(), b.end());
268         return ((*this) * b).shift(-(n - 1));
269     }
270     constexpr std::vector<Value> eval(std::vector<Value> x) const {
271         if (this->size() == 0) {
272             return std::vector<Value>(x.size(), 0);
273         }
274         const int n = std::max(x.size(), this->size());
275         std::vector<Poly> q(4 * n);
276         std::vector<Value> ans(x.size());
277         x.resize(n);
278         std::function<void(int, int, int)> build = [&](int p, int l, int r) {
279             if (r - l == 1) {
280                 q[p] = Poly{1, -x[l]};
281             } else {
282                 int m = (l + r) / 2;

```

```

283         build(2 * p, 1, m);
284         build(2 * p + 1, m, r);
285         q[p] = q[2 * p] * q[2 * p + 1];
286     }
287 };
288 build(1, 0, n);
289 std::function<void(int, int, int, const Poly &)> work = [&](int p, int l,
int r, const Poly &num) {
290     if (r - l == 1) {
291         if (l < int(ans.size())) {
292             ans[l] = num[0];
293         }
294     } else {
295         int m = (l + r) / 2;
296         work(2 * p, l, m, num.mulT(q[2 * p + 1]).resize(m - 1));
297         work(2 * p + 1, m, r, num.mulT(q[2 * p]).resize(r - m));
298     }
299 };
300 work(1, 0, n, mulT(q[1].inv(n)));
301 return ans;
302 }
303 };
304
305 template<int P = 998244353>
306 Poly<P> berlekampMassey(const Poly<P> &s) {
307     Poly<P> c;
308     Poly<P> oldC;
309     int f = -1;
310     for (int i = 0; i < s.size(); i++) {
311         auto delta = s[i];
312         for (int j = 1; j <= c.size(); j++) {
313             delta -= c[j - 1] * s[i - j];
314         }
315         if (delta == 0) {
316             continue;
317         }
318         if (f == -1) {
319             c.resize(i + 1);
320             f = i;
321         } else {
322             auto d = oldC;
323             d *= -1;
324             d.insert(d.begin(), 1);
325             MInt<P> df1 = 0;
326             for (int j = 1; j <= d.size(); j++) {
327                 df1 += d[j - 1] * s[f + 1 - j];
328             }
329             assert(df1 != 0);
330             auto coef = delta / df1;
331             d *= coef;
332             Poly<P> zeros(i - f - 1);
333             zeros.insert(zeros.end(), d.begin(), d.end());
334             d = zeros;
335             auto temp = c;
336             c += d;
337             if (i - temp.size() > f - oldC.size()) {
338                 oldC = temp;
339                 f = i;
340             }
341         }
342     }
343     c *= -1;
344     c.insert(c.begin(), 1);
345     return c;

```

```

346 }
347
348
349 template<int P = 998244353>
350 MInt<P> linearRecurrence(Poly<P> p, Poly<P> q, i64 n) {
351     int m = q.size() - 1;
352     while (n > 0) {
353         auto newq = q;
354         for (int i = 1; i <= m; i += 2) {
355             newq[i] *= -1;
356         }
357         auto newp = p * newq;
358         newq = q * newq;
359         for (int i = 0; i < m; i++) {
360             p[i] = newp[i * 2 + n % 2];
361         }
362         for (int i = 0; i <= m; i++) {
363             q[i] = newq[i * 2];
364         }
365         n /= 2;
366     }
367     return p[0] / q[0];
368 }
369
370 struct Comb {
371     int n;
372     std::vector<Z> _fac;
373     std::vector<Z> _invfac;
374     std::vector<Z> _inv;
375
376     Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
377     Comb(int n) : Comb() {
378         init(n);
379     }
380
381     void init(int m) {
382         m = std::min(m, Z::getMod() - 1);
383         if (m <= n) return;
384         _fac.resize(m + 1);
385         _invfac.resize(m + 1);
386         _inv.resize(m + 1);
387
388         for (int i = n + 1; i <= m; i++) {
389             _fac[i] = _fac[i - 1] * i;
390         }
391         _invfac[m] = _fac[m].inv();
392         for (int i = m; i > n; i--) {
393             _invfac[i - 1] = _invfac[i] * i;
394             _inv[i] = _invfac[i] * _fac[i - 1];
395         }
396         n = m;
397     }
398
399     Z fac(int m) {
400         if (m > n) init(2 * m);
401         return _fac[m];
402     }
403     Z invfac(int m) {
404         if (m > n) init(2 * m);
405         return _invfac[m];
406     }
407     Z inv(int m) {
408         if (m > n) init(2 * m);
409         return _inv[m];

```

```

410     }
411     Z binom(int n, int m) {
412         if (n < m || m < 0) return 0;
413         return fac(n) * invfac(m) * invfac(n - m);
414     }
415 } comb;
416
417 Poly<P> get(int n, int m) {
418     if (m == 0) {
419         return Poly(n + 1);
420     }
421     if (m % 2 == 1) {
422         auto f = get(n, m - 1);
423         Z p = 1;
424         for (int i = 0; i <= n; i++) {
425             f[n - i] += comb.binom(n, i) * p;
426             p *= m;
427         }
428         return f;
429     }
430     auto f = get(n, m / 2);
431     auto fm = f;
432     for (int i = 0; i <= n; i++) {
433         fm[i] *= comb.fac(i);
434     }
435     Poly pw(n + 1);
436     pw[0] = 1;
437     for (int i = 1; i <= n; i++) {
438         pw[i] = pw[i - 1] * (m / 2);
439     }
440     for (int i = 0; i <= n; i++) {
441         pw[i] *= comb.invfac(i);
442     }
443     fm = fm.mulT(pw);
444     for (int i = 0; i <= n; i++) {
445         fm[i] *= comb.invfac(i);
446     }
447     return f + fm;
448 }

```

/END/

4 数据结构

4.1 树状数组 (Fenwick 新版)

```
1  template <typename T>
2  struct Fenwick {
3      int n;
4      std::vector<T> a;
5
6      Fenwick(int n_ = 0) {
7          init(n_);
8      }
9
10     void init(int n_) {
11         n = n_;
12         a.assign(n, T{});
13     }
14
15     void add(int x, const T &v) {
16         for (int i = x + 1; i <= n; i += i & -i) {
17             a[i - 1] = a[i - 1] + v;
18         }
19     }
20
21     T sum(int x) {
22         T ans{};
23         for (int i = x; i > 0; i -= i & -i) {
24             ans = ans + a[i - 1];
25         }
26         return ans;
27     }
28
29     T rangeSum(int l, int r) {
30         return sum(r) - sum(l);
31     }
32
33     int select(const T &k) {
34         int x = 0;
35         T cur{};
36         for (int i = 1 << std::__lg(n); i; i /= 2) {
37             if (x + i <= n && cur + a[x + i - 1] <= k) {
38                 x += i;
39                 cur = cur + a[x - 1];
40             }
41         }
42         return x;
43     }
44 };
```

4.2 并查集 (DSU)

```
1  struct DSU {
2      std::vector<int> f, siz;
3
4      DSU() {}
5      DSU(int n) {
6          init(n);
7      }
8
9      void init(int n) {
```

```

10     f.resize(n);
11     std::iota(f.begin(), f.end(), 0);
12     siz.assign(n, 1);
13 }
14
15 int find(int x) {
16     while (x != f[x]) {
17         x = f[x] = f[f[x]];
18     }
19     return x;
20 }
21
22 bool same(int x, int y) {
23     return find(x) == find(y);
24 }
25
26 bool merge(int x, int y) {
27     x = find(x);
28     y = find(y);
29     if (x == y) {
30         return false;
31     }
32     siz[x] += siz[y];
33     f[y] = x;
34     return true;
35 }
36
37 int size(int x) {
38     return siz[find(x)];
39 }
40 };

```

4.3 线段树

4.3.1 线段树 (SegmentTree 基础区间加乘)

```

1  struct SegmentTree {
2      int n;
3      std::vector<int> tag, sum;
4      SegmentTree(int n_) : n(n_), tag(4 * n, 1), sum(4 * n) {}
5
6      void pull(int p) {
7          sum[p] = (sum[2 * p] + sum[2 * p + 1]) % P;
8      }
9
10     void mul(int p, int v) {
11         tag[p] = 1LL * tag[p] * v % P;
12         sum[p] = 1LL * sum[p] * v % P;
13     }
14
15     void push(int p) {
16         mul(2 * p, tag[p]);
17         mul(2 * p + 1, tag[p]);
18         tag[p] = 1;
19     }
20
21     int query(int p, int l, int r, int x, int y) {
22         if (l >= y || r <= x) {
23             return 0;
24         }
25         if (l >= x && r <= y) {
26             return sum[p];

```

```

27     }
28     int m = (1 + r) / 2;
29     push(p);
30     return (query(2 * p, 1, m, x, y) + query(2 * p + 1, m, r, x, y)) % P;
31 }
32
33 int query(int x, int y) {
34     return query(1, 0, n, x, y);
35 }
36
37 void rangeMul(int p, int l, int r, int x, int y, int v) {
38     if (l >= y || r <= x) {
39         return;
40     }
41     if (l >= x && r <= y) {
42         mul(p, v);
43     }
44     int m = (1 + r) / 2;
45     push(p);
46     rangeMul(2 * p, 1, m, x, y, v);
47     rangeMul(2 * p + 1, m, r, x, y, v);
48     pull(p);
49 }
50
51 void rangeMul(int x, int y, int v) {
52     rangeMul(1, 0, n, x, y, v);
53 }
54
55 void add(int p, int l, int r, int x, int v) {
56     if (r - l == 1) {
57         sum[p] = (sum[p] + v) % P;
58         return;
59     }
60     int m = (1 + r) / 2;
61     push(p);
62     if (x < m) {
63         add(2 * p, 1, m, x, v);
64     } else {
65         add(2 * p + 1, m, r, x, v);
66     }
67     pull(p);
68 }
69
70 void add(int x, int v) {
71     add(1, 0, n, x, v);
72 }
73 };

```

4.3.2 线段树 (SegmentTree+Info 查找前驱后继)

```

1  template<class Info>
2  struct SegmentTree {
3      int n;
4      std::vector<Info> info;
5      SegmentTree() : n(0) {}
6      SegmentTree(int n_, Info v_ = Info()) {
7          init(n_, v_);
8      }
9      template<class T>
10     SegmentTree(std::vector<T> init_) {
11         init(init_);
12     }

```



```

13 void init(int n_, Info v_ = Info()) {
14     init(std::vector(n_, v_));
15 }
16 template<class T>
17 void init(std::vector<T> init_) {
18     n = init_.size();
19     info.assign(4 << std::__lg(n), Info());
20     std::function<void(int, int, int)> build = [&](int p, int l, int r) {
21         if (r - l == 1) {
22             info[p] = init_[l];
23             return;
24         }
25         int m = (l + r) / 2;
26         build(2 * p, l, m);
27         build(2 * p + 1, m, r);
28         pull(p);
29     };
30     build(1, 0, n);
31 }
32 void pull(int p) {
33     info[p] = info[2 * p] + info[2 * p + 1];
34 }
35 void modify(int p, int l, int r, int x, const Info &v) {
36     if (r - l == 1) {
37         info[p] = v;
38         return;
39     }
40     int m = (l + r) / 2;
41     if (x < m) {
42         modify(2 * p, l, m, x, v);
43     } else {
44         modify(2 * p + 1, m, r, x, v);
45     }
46     pull(p);
47 }
48 void modify(int p, const Info &v) {
49     modify(1, 0, n, p, v);
50 }
51 Info rangeQuery(int p, int l, int r, int x, int y) {
52     if (l >= y || r <= x) {
53         return Info();
54     }
55     if (l >= x && r <= y) {
56         return info[p];
57     }
58     int m = (l + r) / 2;
59     return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
60 }
61 Info rangeQuery(int l, int r) {
62     return rangeQuery(1, 0, n, l, r);
63 }
64 template<class F>
65 int findFirst(int p, int l, int r, int x, int y, F pred) {
66     if (l >= y || r <= x || !pred(info[p])) {
67         return -1;
68     }
69     if (r - l == 1) {
70         return l;
71     }
72     int m = (l + r) / 2;
73     int res = findFirst(2 * p, l, m, x, y, pred);
74     if (res == -1) {
75         res = findFirst(2 * p + 1, m, r, x, y, pred);
76     }

```

```

77     return res;
78 }
79 template<class F>
80 int findFirst(int l, int r, F pred) {
81     return findFirst(1, 0, n, l, r, pred);
82 }
83 template<class F>
84 int findLast(int p, int l, int r, int x, int y, F pred) {
85     if (l >= y || r <= x || !pred(info[p])) {
86         return -1;
87     }
88     if (r - l == 1) {
89         return l;
90     }
91     int m = (l + r) / 2;
92     int res = findLast(2 * p + 1, m, r, x, y, pred);
93     if (res == -1) {
94         res = findLast(2 * p, l, m, x, y, pred);
95     }
96     return res;
97 }
98 template<class F>
99 int findLast(int l, int r, F pred) {
100     return findLast(1, 0, n, l, r, pred);
101 }
102 };
103 struct Info {
104     int cnt = 0;
105     i64 sum = 0;
106     i64 ans = 0;
107 };
108 Info operator+(Info a, Info b) {
109     Info c;
110     c.cnt = a.cnt + b.cnt;
111     c.sum = a.sum + b.sum;
112     c.ans = a.ans + b.ans + a.cnt * b.sum - a.sum * b.cnt;
113     return c;
114 }

```

4.3.3 线段树 (SegmentTree+Info+Merge 区间合并)

```

1  template<class Info>
2  struct SegmentTree {
3      int n;
4      std::vector<Info> info;
5      SegmentTree() : n(0) {}
6      SegmentTree(int n_, Info v_ = Info()) {
7          init(n_, v_);
8      }
9      template<class T>
10     SegmentTree(std::vector<T> init_) {
11         init(init_);
12     }
13     void init(int n_, Info v_ = Info()) {
14         init(std::vector(n_, v_));
15     }
16     template<class T>
17     void init(std::vector<T> init_) {
18         n = init_.size();
19         info.assign(4 << std::lg(n), Info());
20         std::function<void(int, int, int)> build = [&](int p, int l, int r) {
21             if (r - l == 1) {

```

```

22         info[p] = init_[l];
23         return;
24     }
25     int m = (l + r) / 2;
26     build(2 * p, l, m);
27     build(2 * p + 1, m, r);
28     pull(p);
29 };
30 build(1, 0, n);
31 }
32 void pull(int p) {
33     info[p] = info[2 * p] + info[2 * p + 1];
34 }
35 void modify(int p, int l, int r, int x, const Info &v) {
36     if (r - l == 1) {
37         info[p] = v;
38         return;
39     }
40     int m = (l + r) / 2;
41     if (x < m) {
42         modify(2 * p, l, m, x, v);
43     } else {
44         modify(2 * p + 1, m, r, x, v);
45     }
46     pull(p);
47 }
48 void modify(int p, const Info &v) {
49     modify(1, 0, n, p, v);
50 }
51 Info rangeQuery(int p, int l, int r, int x, int y) {
52     if (l >= y || r <= x) {
53         return Info();
54     }
55     if (l >= x && r <= y) {
56         return info[p];
57     }
58     int m = (l + r) / 2;
59     return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
60 }
61 Info rangeQuery(int l, int r) {
62     return rangeQuery(1, 0, n, l, r);
63 }
64 template<class F>
65 int findFirst(int p, int l, int r, int x, int y, F pred) {
66     if (l >= y || r <= x || !pred(info[p])) {
67         return -1;
68     }
69     if (r - l == 1) {
70         return l;
71     }
72     int m = (l + r) / 2;
73     int res = findFirst(2 * p, l, m, x, y, pred);
74     if (res == -1) {
75         res = findFirst(2 * p + 1, m, r, x, y, pred);
76     }
77     return res;
78 }
79 template<class F>
80 int findFirst(int l, int r, F pred) {
81     return findFirst(1, 0, n, l, r, pred);
82 }
83 template<class F>
84 int findLast(int p, int l, int r, int x, int y, F pred) {
85     if (l >= y || r <= x || !pred(info[p])) {

```

```

86         return -1;
87     }
88     if (r - l == 1) {
89         return l;
90     }
91     int m = (l + r) / 2;
92     int res = findLast(2 * p + 1, m, r, x, y, pred);
93     if (res == -1) {
94         res = findLast(2 * p, l, m, x, y, pred);
95     }
96     return res;
97 }
98 template<class F>
99 int findLast(int l, int r, F pred) {
100     return findLast(1, 0, n, l, r, pred);
101 }
102 };
103
104 struct Info {
105     int x = 0;
106     int cnt = 0;
107 };
108
109 Info operator+(Info a, Info b) {
110     if (a.x == b.x) {
111         return {a.x, a.cnt + b.cnt};
112     } else if (a.cnt > b.cnt) {
113         return {a.x, a.cnt - b.cnt};
114     } else {
115         return {b.x, b.cnt - a.cnt};
116     }
117 }

```

4.4 懒标记线段树

4.4.1 懒标记线段树 (LazySegmentTree 基础区间修改)

```

1  template<class Info, class Tag>
2  struct LazySegmentTree {
3      const int n;
4      std::vector<Info> info;
5      std::vector<Tag> tag;
6      LazySegmentTree(int n) : n(n), info(4 << std::__lg(n)), tag(4 << std::__lg(n))
7  {}
8      LazySegmentTree(std::vector<Info> init) : LazySegmentTree(init.size()) {
9          std::function<void(int, int, int)> build = [&](int p, int l, int r) {
10              if (r - l == 1) {
11                  info[p] = init[l];
12                  return;
13              }
14              int m = (l + r) / 2;
15              build(2 * p, l, m);
16              build(2 * p + 1, m, r);
17              pull(p);
18          };
19          build(1, 0, n);
20      }
21      void pull(int p) {
22          info[p] = info[2 * p] + info[2 * p + 1];
23      }
24      void apply(int p, const Tag &v) {
25          info[p].apply(v);

```

```

25     tag[p].apply(v);
26 }
27 void push(int p) {
28     apply(2 * p, tag[p]);
29     apply(2 * p + 1, tag[p]);
30     tag[p] = Tag();
31 }
32 void modify(int p, int l, int r, int x, const Info &v) {
33     if (r - l == 1) {
34         info[p] = v;
35         return;
36     }
37     int m = (l + r) / 2;
38     push(p);
39     if (x < m) {
40         modify(2 * p, l, m, x, v);
41     } else {
42         modify(2 * p + 1, m, r, x, v);
43     }
44     pull(p);
45 }
46 void modify(int p, const Info &v) {
47     modify(1, 0, n, p, v);
48 }
49 Info rangeQuery(int p, int l, int r, int x, int y) {
50     if (l >= y || r <= x) {
51         return Info();
52     }
53     if (l >= x && r <= y) {
54         return info[p];
55     }
56     int m = (l + r) / 2;
57     push(p);
58     return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
59 }
60 Info rangeQuery(int l, int r) {
61     return rangeQuery(1, 0, n, l, r);
62 }
63 void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
64     if (l >= y || r <= x) {
65         return;
66     }
67     if (l >= x && r <= y) {
68         apply(p, v);
69         return;
70     }
71     int m = (l + r) / 2;
72     push(p);
73     rangeApply(2 * p, l, m, x, y, v);
74     rangeApply(2 * p + 1, m, r, x, y, v);
75     pull(p);
76 }
77 void rangeApply(int l, int r, const Tag &v) {
78     return rangeApply(1, 0, n, l, r, v);
79 }
80 void half(int p, int l, int r) {
81     if (info[p].act == 0) {
82         return;
83     }
84     if ((info[p].min + 1) / 2 == (info[p].max + 1) / 2) {
85         apply(p, {-(info[p].min + 1) / 2});
86         return;
87     }
88     int m = (l + r) / 2;

```

```

89     push(p);
90     half(2 * p, l, m);
91     half(2 * p + 1, m, r);
92     pull(p);
93 }
94 void half() {
95     half(1, 0, n);
96 }
97 };
98
99 constexpr i64 inf = 1E18;
100
101 struct Tag {
102     i64 add = 0;
103
104     void apply(Tag t) {
105         add += t.add;
106     }
107 };
108
109 struct Info {
110     i64 min = inf;
111     i64 max = -inf;
112     i64 sum = 0;
113     i64 act = 0;
114
115     void apply(Tag t) {
116         min += t.add;
117         max += t.add;
118         sum += act * t.add;
119     }
120 };
121
122 Info operator+(Info a, Info b) {
123     Info c;
124     c.min = std::min(a.min, b.min);
125     c.max = std::max(a.max, b.max);
126     c.sum = a.sum + b.sum;
127     c.act = a.act + b.act;
128     return c;
129 }

```

4.4.2 懒标记线段树 (LazySegmentTree 查找前驱后继)

```

1  template<class Info, class Tag>
2  struct LazySegmentTree {
3      int n;
4      std::vector<Info> info;
5      std::vector<Tag> tag;
6      LazySegmentTree() : n(0) {}
7      LazySegmentTree(int n_, Info v_ = Info()) {
8          init(n_, v_);
9      }
10     template<class T>
11     LazySegmentTree(std::vector<T> init_) {
12         init(init_);
13     }
14     void init(int n_, Info v_ = Info()) {
15         init(std::vector(n_, v_));
16     }
17     template<class T>
18     void init(std::vector<T> init_) {

```

```

19     n = init_.size();
20     info.assign(4 << std::__lg(n), Info());
21     tag.assign(4 << std::__lg(n), Tag());
22     std::function<void(int, int, int)> build = [&](int p, int l, int r) {
23         if (r - l == 1) {
24             info[p] = init_[l];
25             return;
26         }
27         int m = (l + r) / 2;
28         build(2 * p, l, m);
29         build(2 * p + 1, m, r);
30         pull(p);
31     };
32     build(1, 0, n);
33 }
34 void pull(int p) {
35     info[p] = info[2 * p] + info[2 * p + 1];
36 }
37 void apply(int p, const Tag &v) {
38     info[p].apply(v);
39     tag[p].apply(v);
40 }
41 void push(int p) {
42     apply(2 * p, tag[p]);
43     apply(2 * p + 1, tag[p]);
44     tag[p] = Tag();
45 }
46 void modify(int p, int l, int r, int x, const Info &v) {
47     if (r - l == 1) {
48         info[p] = v;
49         return;
50     }
51     int m = (l + r) / 2;
52     push(p);
53     if (x < m) {
54         modify(2 * p, l, m, x, v);
55     } else {
56         modify(2 * p + 1, m, r, x, v);
57     }
58     pull(p);
59 }
60 void modify(int p, const Info &v) {
61     modify(1, 0, n, p, v);
62 }
63 Info rangeQuery(int p, int l, int r, int x, int y) {
64     if (l >= y || r <= x) {
65         return Info();
66     }
67     if (l >= x && r <= y) {
68         return info[p];
69     }
70     int m = (l + r) / 2;
71     push(p);
72     return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
73 }
74 Info rangeQuery(int l, int r) {
75     return rangeQuery(1, 0, n, l, r);
76 }
77 void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
78     if (l >= y || r <= x) {
79         return;
80     }
81     if (l >= x && r <= y) {
82         apply(p, v);

```

```

83         return;
84     }
85     int m = (l + r) / 2;
86     push(p);
87     rangeApply(2 * p, l, m, x, y, v);
88     rangeApply(2 * p + 1, m, r, x, y, v);
89     pull(p);
90 }
91 void rangeApply(int l, int r, const Tag &v) {
92     return rangeApply(1, 0, n, l, r, v);
93 }
94 template<class F>
95 int findFirst(int p, int l, int r, int x, int y, F pred) {
96     if (l >= y || r <= x || !pred(info[p])) {
97         return -1;
98     }
99     if (r - l == 1) {
100         return l;
101     }
102     int m = (l + r) / 2;
103     push(p);
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, 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, F pred) {
116     if (l >= y || r <= x || !pred(info[p])) {
117         return -1;
118     }
119     if (r - l == 1) {
120         return l;
121     }
122     int m = (l + r) / 2;
123     push(p);
124     int res = findLast(2 * p + 1, m, r, x, y, pred);
125     if (res == -1) {
126         res = findLast(2 * p, l, m, x, y, pred);
127     }
128     return res;
129 }
130 template<class F>
131 int findLast(int l, int r, F pred) {
132     return findLast(1, 0, n, l, r, pred);
133 }
134 };
135
136 struct Tag {
137     i64 a = 0, b = 0;
138     void apply(Tag t) {
139         a = std::min(a, b + t.a);
140         b += t.b;
141     }
142 };
143
144 int k;
145
146 struct Info {

```



```

147     i64 x = 0;
148     void apply(Tag t) {
149         x += t.a;
150         if (x < 0) {
151             x = (x % k + k) % k;
152         }
153         x += t.b - t.a;
154     }
155 };
156 Info operator+(Info a, Info b) {
157     return {a.x + b.x};
158 }

```

4.4.3 懒标记线段树 (LazySegmentTree 二分修改)

```

1  constexpr int inf = 1E9 + 1;
2  template<class Info, class Tag>
3  struct LazySegmentTree {
4      const int n;
5      std::vector<Info> info;
6      std::vector<Tag> tag;
7      LazySegmentTree(int n) : n(n), info(4 << std::__lg(n)), tag(4 << std::__lg(n))
8  {}
9      LazySegmentTree(std::vector<Info> init) : LazySegmentTree(init.size()) {
10         std::function<void(int, int, int)> build = [&](int p, int l, int r) {
11             if (r - l == 1) {
12                 info[p] = init[l];
13                 return;
14             }
15             int m = (l + r) / 2;
16             build(2 * p, l, m);
17             build(2 * p + 1, m, r);
18             pull(p);
19         };
20         build(1, 0, n);
21     }
22     void pull(int p) {
23         info[p] = info[2 * p] + info[2 * p + 1];
24     }
25     void apply(int p, const Tag &v) {
26         info[p].apply(v);
27         tag[p].apply(v);
28     }
29     void push(int p) {
30         apply(2 * p, tag[p]);
31         apply(2 * p + 1, tag[p]);
32         tag[p] = Tag();
33     }
34     void modify(int p, int l, int r, int x, const Info &v) {
35         if (r - l == 1) {
36             info[p] = v;
37             return;
38         }
39         int m = (l + r) / 2;
40         push(p);
41         if (x < m) {
42             modify(2 * p, l, m, x, v);
43         } else {
44             modify(2 * p + 1, m, r, x, v);
45         }
46         pull(p);
47     }
48 }

```

```

47 void modify(int p, const Info &v) {
48     modify(1, 0, n, p, v);
49 }
50 Info rangeQuery(int p, int l, int r, int x, int y) {
51     if (l >= y || r <= x) {
52         return Info();
53     }
54     if (l >= x && r <= y) {
55         return info[p];
56     }
57     int m = (l + r) / 2;
58     push(p);
59     return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
60 }
61 Info rangeQuery(int l, int r) {
62     return rangeQuery(1, 0, n, l, r);
63 }
64 void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
65     if (l >= y || r <= x) {
66         return;
67     }
68     if (l >= x && r <= y) {
69         apply(p, v);
70         return;
71     }
72     int m = (l + r) / 2;
73     push(p);
74     rangeApply(2 * p, l, m, x, y, v);
75     rangeApply(2 * p + 1, m, r, x, y, v);
76     pull(p);
77 }
78 void rangeApply(int l, int r, const Tag &v) {
79     return rangeApply(1, 0, n, l, r, v);
80 }
81 void maintainL(int p, int l, int r, int pre) {
82     if (info[p].difl > 0 && info[p].maxlowl < pre) {
83         return;
84     }
85     if (r - l == 1) {
86         info[p].max = info[p].maxlowl;
87         info[p].maxl = info[p].maxr = l;
88         info[p].maxlowl = info[p].maxlowr = -inf;
89         return;
90     }
91     int m = (l + r) / 2;
92     push(p);
93     maintainL(2 * p, l, m, pre);
94     pre = std::max(pre, info[2 * p].max);
95     maintainL(2 * p + 1, m, r, pre);
96     pull(p);
97 }
98 void maintainL() {
99     maintainL(1, 0, n, -1);
100 }
101 void maintainR(int p, int l, int r, int suf) {
102     if (info[p].difr > 0 && info[p].maxlowr < suf) {
103         return;
104     }
105     if (r - l == 1) {
106         info[p].max = info[p].maxlowl;
107         info[p].maxl = info[p].maxr = l;
108         info[p].maxlowl = info[p].maxlowr = -inf;
109         return;
110     }

```

```

111     int m = (1 + r) / 2;
112     push(p);
113     maintainR(2 * p + 1, m, r, suf);
114     suf = std::max(suf, info[2 * p + 1].max);
115     maintainR(2 * p, 1, m, suf);
116     pull(p);
117 }
118 void maintainR() {
119     maintainR(1, 0, n, -1);
120 }
121 };
122
123 struct Tag {
124     int add = 0;
125
126     void apply(Tag t) & {
127         add += t.add;
128     }
129 };
130
131 struct Info {
132     int max = -1;
133     int maxl = -1;
134     int maxr = -1;
135     int difl = inf;
136     int difr = inf;
137     int maxlowl = -inf;
138     int maxlowr = -inf;
139
140     void apply(Tag t) & {
141         if (max != -1) {
142             max += t.add;
143         }
144         difl += t.add;
145         difr += t.add;
146     }
147 };
148
149 Info operator+(Info a, Info b) {
150     Info c;
151     if (a.max > b.max) {
152         c.max = a.max;
153         c.maxl = a.maxl;
154         c.maxr = a.maxr;
155     } else if (a.max < b.max) {
156         c.max = b.max;
157         c.maxl = b.maxl;
158         c.maxr = b.maxr;
159     } else {
160         c.max = a.max;
161         c.maxl = a.maxl;
162         c.maxr = b.maxr;
163     }
164
165     c.difl = std::min(a.difl, b.difl);
166     c.difr = std::min(a.difr, b.difr);
167     if (a.max != -1) {
168         c.difl = std::min(c.difl, a.max - b.maxlowl);
169     }
170     if (b.max != -1) {
171         c.difr = std::min(c.difr, b.max - a.maxlowr);
172     }
173
174     if (a.max == -1) {

```

```

175         c.maxlowl = std::max(a.maxlowl, b.maxlowl);
176     } else {
177         c.maxlowl = a.maxlowl;
178     }
179     if (b.max == -1) {
180         c.maxlowr = std::max(a.maxlowr, b.maxlowr);
181     } else {
182         c.maxlowr = b.maxlowr;
183     }
184     return c;
185 }

```

4.5 取模类

4.5.1 取模类 (MLong & MInt)

```

1  constexpr int P = 998244353;
2  using i64 = long long;
3  // assume -P <= x < 2P
4  int norm(int x) {
5      if (x < 0) {
6          x += P;
7      }
8      if (x >= P) {
9          x -= P;
10     }
11     return x;
12 }
13 template<class T>
14 T power(T a, i64 b) {
15     T res = 1;
16     for (; b; b /= 2, a *= a) {
17         if (b % 2) {
18             res *= a;
19         }
20     }
21     return res;
22 }
23 struct Z {
24     int x;
25     Z(int x = 0) : x(norm(x)) {}
26     Z(i64 x) : x(norm(x % P)) {}
27     int val() const {
28         return x;
29     }
30     Z operator-() const {
31         return Z(norm(P - x));
32     }
33     Z inv() const {
34         assert(x != 0);
35         return power(*this, P - 2);
36     }
37     Z &operator*=(const Z &rhs) {
38         x = i64(x) * rhs.x % P;
39         return *this;
40     }
41     Z &operator+=(const Z &rhs) {
42         x = norm(x + rhs.x);
43         return *this;
44     }
45     Z &operator-=(const Z &rhs) {
46         x = norm(x - rhs.x);

```

```

47     return *this;
48 }
49 Z &operator/=(const Z &rhs) {
50     return *this *= rhs.inv();
51 }
52 friend Z operator*(const Z &lhs, const Z &rhs) {
53     Z res = lhs;
54     res *= rhs;
55     return res;
56 }
57 friend Z operator+(const Z &lhs, const Z &rhs) {
58     Z res = lhs;
59     res += rhs;
60     return res;
61 }
62 friend Z operator-(const Z &lhs, const Z &rhs) {
63     Z res = lhs;
64     res -= rhs;
65     return res;
66 }
67 friend Z operator/(const Z &lhs, const Z &rhs) {
68     Z res = lhs;
69     res /= rhs;
70     return res;
71 }
72 friend std::istream &operator>>(std::istream &is, Z &a) {
73     i64 v;
74     is >> v;
75     a = Z(v);
76     return is;
77 }
78 friend std::ostream &operator<<(std::ostream &os, const Z &a) {
79     return os << a.val();
80 }
81 };

```

4.5.2 取模类 (MLong & MInt 新版)

根据输入内容动态修改 MOD 的方法：`Z::setMod(p);`。

```

1  template<class T>
2  constexpr T power(T a, i64 b) {
3      T res = 1;
4      for (; b; b /= 2, a *= a) {
5          if (b % 2) {
6              res *= a;
7          }
8      }
9      return res;
10 }
11
12 constexpr i64 mul(i64 a, i64 b, i64 p) {
13     i64 res = a * b - i64(1.L * a * b / p) * p;
14     res %= p;
15     if (res < 0) {
16         res += p;
17     }
18     return res;
19 }
20 template<i64 P>
21 struct MLong {
22     i64 x;

```

```

23 constexpr MLong() : x{} {}
24 constexpr MLong(i64 x) : x{norm(x % getMod())} {}
25
26 static i64 Mod;
27 constexpr static i64 getMod() {
28     if (P > 0) {
29         return P;
30     } else {
31         return Mod;
32     }
33 }
34 constexpr static void setMod(i64 Mod_) {
35     Mod = Mod_;
36 }
37 constexpr i64 norm(i64 x) const {
38     if (x < 0) {
39         x += getMod();
40     }
41     if (x >= getMod()) {
42         x -= getMod();
43     }
44     return x;
45 }
46 constexpr i64 val() const {
47     return x;
48 }
49 explicit constexpr operator i64() const {
50     return x;
51 }
52 constexpr MLong operator-() const {
53     MLong res;
54     res.x = norm(getMod() - x);
55     return res;
56 }
57 constexpr MLong inv() const {
58     assert(x != 0);
59     return power(*this, getMod() - 2);
60 }
61 constexpr MLong &operator*=(MLong rhs) & {
62     x = mul(x, rhs.x, getMod());
63     return *this;
64 }
65 constexpr MLong &operator+=(MLong rhs) & {
66     x = norm(x + rhs.x);
67     return *this;
68 }
69 constexpr MLong &operator-=(MLong rhs) & {
70     x = norm(x - rhs.x);
71     return *this;
72 }
73 constexpr MLong &operator/=(MLong rhs) & {
74     return *this *= rhs.inv();
75 }
76 friend constexpr MLong operator*(MLong lhs, MLong rhs) {
77     MLong res = lhs;
78     res *= rhs;
79     return res;
80 }
81 friend constexpr MLong operator+(MLong lhs, MLong rhs) {
82     MLong res = lhs;
83     res += rhs;
84     return res;
85 }
86 friend constexpr MLong operator-(MLong lhs, MLong rhs) {

```

```

87     MLong res = lhs;
88     res -= rhs;
89     return res;
90 }
91 friend constexpr MLong operator/(MLong lhs, MLong rhs) {
92     MLong res = lhs;
93     res /= rhs;
94     return res;
95 }
96 friend constexpr std::istream &operator>>(std::istream &is, MLong &a) {
97     i64 v;
98     is >> v;
99     a = MLong(v);
100    return is;
101 }
102 friend constexpr std::ostream &operator<<(std::ostream &os, const MLong &a) {
103     return os << a.val();
104 }
105 friend constexpr bool operator==(MLong lhs, MLong rhs) {
106     return lhs.val() == rhs.val();
107 }
108 friend constexpr bool operator!=(MLong lhs, MLong rhs) {
109     return lhs.val() != rhs.val();
110 }
111 };
112
113 template<>
114 i64 MLong<0LL>::Mod = i64(1E18) + 9;
115
116 template<int P>
117 struct MInt {
118     int x;
119     constexpr MInt() : x{} {}
120     constexpr MInt(i64 x) : x{norm(x % getMod())} {}
121
122     static int Mod;
123     constexpr static int getMod() {
124         if (P > 0) {
125             return P;
126         } else {
127             return Mod;
128         }
129     }
130     constexpr static void setMod(int Mod_) {
131         Mod = Mod_;
132     }
133     constexpr int norm(int x) const {
134         if (x < 0) {
135             x += getMod();
136         }
137         if (x >= getMod()) {
138             x -= getMod();
139         }
140         return x;
141     }
142     constexpr int val() const {
143         return x;
144     }
145     explicit constexpr operator int() const {
146         return x;
147     }
148     constexpr MInt operator-() const {
149         MInt res;
150         res.x = norm(getMod() - x);

```

```

151         return res;
152     }
153     constexpr MInt inv() const {
154         assert(x != 0);
155         return power(*this, getMod() - 2);
156     }
157     constexpr MInt &operator*=(MInt rhs) & {
158         x = 1LL * x * rhs.x % getMod();
159         return *this;
160     }
161     constexpr MInt &operator+=(MInt rhs) & {
162         x = norm(x + rhs.x);
163         return *this;
164     }
165     constexpr MInt &operator-=(MInt rhs) & {
166         x = norm(x - rhs.x);
167         return *this;
168     }
169     constexpr MInt &operator/=(MInt rhs) & {
170         return *this *= rhs.inv();
171     }
172     friend constexpr MInt operator*(MInt lhs, MInt rhs) {
173         MInt res = lhs;
174         res *= rhs;
175         return res;
176     }
177     friend constexpr MInt operator+(MInt lhs, MInt rhs) {
178         MInt res = lhs;
179         res += rhs;
180         return res;
181     }
182     friend constexpr MInt operator-(MInt lhs, MInt rhs) {
183         MInt res = lhs;
184         res -= rhs;
185         return res;
186     }
187     friend constexpr MInt operator/(MInt lhs, MInt rhs) {
188         MInt res = lhs;
189         res /= rhs;
190         return res;
191     }
192     friend constexpr std::istream &operator>>(std::istream &is, MInt &a) {
193         i64 v;
194         is >> v;
195         a = MInt(v);
196         return is;
197     }
198     friend constexpr std::ostream &operator<<(std::ostream &os, const MInt &a) {
199         return os << a.val();
200     }
201     friend constexpr bool operator==(MInt lhs, MInt rhs) {
202         return lhs.val() == rhs.val();
203     }
204     friend constexpr bool operator!=(MInt lhs, MInt rhs) {
205         return lhs.val() != rhs.val();
206     }
207 };
208
209 template<>
210 int MInt<0>::Mod = 998244353;
211
212 template<int V, int P>
213 constexpr MInt<P> CInv = MInt<P>(V).inv();
214

```



```

215 constexpr int P = 1000000007;
216 using Z = MInt<P>;

```

4.6 状压RMQ (RMQ)

```

1  template<class T,
2      class Cmp = std::less<T>>
3  struct RMQ {
4      const Cmp cmp = Cmp();
5      static constexpr unsigned B = 64;
6      using u64 = unsigned long long;
7      int n;
8      std::vector<std::vector<T>> a;
9      std::vector<T> pre, suf, ini;
10     std::vector<u64> stk;
11     RMQ() {}
12     RMQ(const std::vector<T> &v) {
13         init(v);
14     }
15     void init(const std::vector<T> &v) {
16         n = v.size();
17         pre = suf = ini = v;
18         stk.resize(n);
19         if (!n) {
20             return;
21         }
22         const int M = (n - 1) / B + 1;
23         const int lg = std::__lg(M);
24         a.assign(lg + 1, std::vector<T>(M));
25         for (int i = 0; i < M; i++) {
26             a[0][i] = v[i * B];
27             for (int j = 1; j < B && i * B + j < n; j++) {
28                 a[0][i] = std::min(a[0][i], v[i * B + j], cmp);
29             }
30         }
31         for (int i = 1; i < n; i++) {
32             if (i % B) {
33                 pre[i] = std::min(pre[i], pre[i - 1], cmp);
34             }
35         }
36         for (int i = n - 2; i >= 0; i--) {
37             if (i % B != B - 1) {
38                 suf[i] = std::min(suf[i], suf[i + 1], cmp);
39             }
40         }
41         for (int j = 0; j < lg; j++) {
42             for (int i = 0; i + (2 << j) <= M; i++) {
43                 a[j + 1][i] = std::min(a[j][i], a[j][i + (1 << j)], cmp);
44             }
45         }
46         for (int i = 0; i < M; i++) {
47             const int l = i * B;
48             const int r = std::min(1U * n, l + B);
49             u64 s = 0;
50             for (int j = 1; j < r; j++) {
51                 while (s && cmp(v[j], v[std::__lg(s) + 1])) {
52                     s ^= 1ULL << std::__lg(s);
53                 }
54                 s |= 1ULL << (j - 1);
55                 stk[j] = s;
56             }
57         }

```

```

58     }
59     T operator()(int l, int r) {
60         if (l / B != (r - 1) / B) {
61             T ans = std::min(suf[l], pre[r - 1], cmp);
62             l = l / B + 1;
63             r = r / B;
64             if (l < r) {
65                 int k = std::__lg(r - l);
66                 ans = std::min({ans, a[k][l], a[k][r - (1 << k)]}, cmp);
67             }
68             return ans;
69         } else {
70             int x = B * (l / B);
71             return ini[__builtin_ctzll(stk[r - 1] >> (l - x)) + 1];
72         }
73     }
74 };

```

4.6.1 Splay

```

1  struct Node {
2      Node *l = nullptr;
3      Node *r = nullptr;
4      int cnt = 0;
5      i64 sum = 0;
6  };
7
8  Node *add(Node *t, int l, int r, int p, int v) {
9      Node *x = new Node;
10     if (t) {
11         *x = *t;
12     }
13     x->cnt += 1;
14     x->sum += v;
15     if (r - l == 1) {
16         return x;
17     }
18     int m = (l + r) / 2;
19     if (p < m) {
20         x->l = add(x->l, l, m, p, v);
21     } else {
22         x->r = add(x->r, m, r, p, v);
23     }
24     return x;
25 }
26
27 int find(Node *tl, Node *tr, int l, int r, int x) {
28     if (r <= x) {
29         return -1;
30     }
31     if (l >= x) {
32         int cnt = (tr ? tr->cnt : 0) - (tl ? tl->cnt : 0);
33         if (cnt == 0) {
34             return -1;
35         }
36         if (r - l == 1) {
37             return l;
38         }
39     }
40     int m = (l + r) / 2;
41     int res = find(tl ? tl->l : tl, tr ? tr->l : tr, l, m, x);
42     if (res == -1) {

```

```

43     res = find(tl ? tl->r : tl, tr ? tr->r : tr, m, r, x);
44 }
45 return res;
46 }
47
48 std::pair<int, i64> get(Node *t, int l, int r, int x, int y) {
49     if (l >= y || r <= x || !t) {
50         return {0, 0LL};
51     }
52     if (l >= x && r <= y) {
53         return {t->cnt, t->sum};
54     }
55     int m = (l + r) / 2;
56     auto [cl, sl] = get(t->l, l, m, x, y);
57     auto [cr, sr] = get(t->r, m, r, x, y);
58     return {cl + cr, sl + sr};
59 }
60
61 struct Tree {
62     int add = 0;
63     int val = 0;
64     int id = 0;
65     Tree *ch[2] = {};
66     Tree *p = nullptr;
67 };
68
69 int pos(Tree *t) {
70     return t->p->ch[1] == t;
71 }
72
73 void add(Tree *t, int v) {
74     t->val += v;
75     t->add += v;
76 }
77
78 void push(Tree *t) {
79     if (t->ch[0]) {
80         add(t->ch[0], t->add);
81     }
82     if (t->ch[1]) {
83         add(t->ch[1], t->add);
84     }
85     t->add = 0;
86 }
87
88 void rotate(Tree *t) {
89     Tree *q = t->p;
90     int x = !pos(t);
91     q->ch[!x] = t->ch[x];
92     if (t->ch[x]) t->ch[x]->p = q;
93     t->p = q->p;
94     if (q->p) q->p->ch[pos(q)] = t;
95     t->ch[x] = q;
96     q->p = t;
97 }
98
99 void splay(Tree *t) {
100     std::vector<Tree *> s;
101     for (Tree *i = t; i->p; i = i->p) s.push_back(i->p);
102     while (!s.empty()) {
103         push(s.back());
104         s.pop_back();
105     }
106     push(t);

```

```

107     while (t->p) {
108         if (t->p->p) {
109             if (pos(t) == pos(t->p)) rotate(t->p);
110             else rotate(t);
111         }
112         rotate(t);
113     }
114 }
115
116 void insert(Tree *&t, Tree *x, Tree *p = nullptr) {
117     if (!t) {
118         t = x;
119         x->p = p;
120         return;
121     }
122
123     push(t);
124     if (x->val < t->val) {
125         insert(t->ch[0], x, t);
126     } else {
127         insert(t->ch[1], x, t);
128     }
129 }
130
131 void dfs(Tree *t) {
132     if (!t) {
133         return;
134     }
135     push(t);
136     dfs(t->ch[0]);
137     std::cerr << t->val << " ";
138     dfs(t->ch[1]);
139 }
140
141 std::pair<Tree *, Tree *> split(Tree *t, int x) {
142     if (!t) {
143         return {t, t};
144     }
145     Tree *v = nullptr;
146     Tree *j = t;
147     for (Tree *i = t; i; ) {
148         push(i);
149         j = i;
150         if (i->val >= x) {
151             v = i;
152             i = i->ch[0];
153         } else {
154             i = i->ch[1];
155         }
156     }
157
158     splay(j);
159     if (!v) {
160         return {j, nullptr};
161     }
162
163     splay(v);
164
165     Tree *u = v->ch[0];
166     if (u) {
167         v->ch[0] = u->p = nullptr;
168     }
169     // std::cerr << "split " << x << "\n";
170     // dfs(u);

```

```

171 // std::cerr << "\n";
172 // dfs(v);
173 // std::cerr << "\n";
174 return {u, v};
175 }
176
177 Tree *merge(Tree *l, Tree *r) {
178     if (!l) {
179         return r;
180     }
181     if (!r) {
182         return l;
183     }
184     Tree *i = l;
185     while (i->ch[1]) {
186         i = i->ch[1];
187     }
188     splay(i);
189     i->ch[1] = r;
190     r->p = i;
191     return i;
192 }

```

```

1 struct Node {
2     Node *ch[2], *p;
3     bool rev;
4     int siz = 1;
5     Node() : ch{nullptr, nullptr}, p(nullptr), rev(false) {}
6 };
7 void reverse(Node *t) {
8     if (t) {
9         std::swap(t->ch[0], t->ch[1]);
10        t->rev ^= 1;
11    }
12 }
13 void push(Node *t) {
14     if (t->rev) {
15         reverse(t->ch[0]);
16         reverse(t->ch[1]);
17         t->rev = false;
18     }
19 }
20 void pull(Node *t) {
21     t->siz = (t->ch[0] ? t->ch[0]->siz : 0) + 1 + (t->ch[1] ? t->ch[1]->siz : 0);
22 }
23 bool isroot(Node *t) {
24     return t->p == nullptr || (t->p->ch[0] != t && t->p->ch[1] != t);
25 }
26 int pos(Node *t) {
27     return t->p->ch[1] == t;
28 }
29 void pushAll(Node *t) {
30     if (!isroot(t)) {
31         pushAll(t->p);
32     }
33     push(t);
34 }
35 void rotate(Node *t) {
36     Node *q = t->p;
37     int x = !pos(t);
38     q->ch[!x] = t->ch[x];
39     if (t->ch[x]) {
40         t->ch[x]->p = q;

```

```

41     }
42     t->p = q->p;
43     if (!isroot(q)) {
44         q->p->ch[pos(q)] = t;
45     }
46     t->ch[x] = q;
47     q->p = t;
48     pull(q);
49 }
50 void splay(Node *t) {
51     pushAll(t);
52     while (!isroot(t)) {
53         if (!isroot(t->p)) {
54             if (pos(t) == pos(t->p)) {
55                 rotate(t->p);
56             } else {
57                 rotate(t);
58             }
59         }
60         rotate(t);
61     }
62     pull(t);
63 }
64 void access(Node *t) {
65     for (Node *i = t, *q = nullptr; i; q = i, i = i->p) {
66         splay(i);
67         i->ch[1] = q;
68         pull(i);
69     }
70     splay(t);
71 }
72 void makeroot(Node *t) {
73     access(t);
74     reverse(t);
75 }
76 void link(Node *x, Node *y) {
77     makeroot(x);
78     x->p = y;
79 }
80 void split(Node *x, Node *y) {
81     makeroot(x);
82     access(y);
83 }
84 void cut(Node *x, Node *y) {
85     split(x, y);
86     x->p = y->ch[0] = nullptr;
87     pull(y);
88 }
89 int dist(Node *x, Node *y) {
90     split(x, y);
91     return y->siz - 1;
92 }

```

```

1  struct Matrix : std::array<std::array<i64, 4>, 4> {
2      Matrix(i64 v = 0) {
3          for (int i = 0; i < 4; i++) {
4              for (int j = 0; j < 4; j++) {
5                  (*this)[i][j] = (i == j ? v : inf);
6              }
7          }
8      }
9  };
10

```

```

11 Matrix operator*(const Matrix &a, const Matrix &b) {
12     Matrix c(inf);
13     for (int i = 0; i < 3; i++) {
14         for (int j = 0; j < 3; j++) {
15             for (int k = 0; k < 4; k++) {
16                 c[i][k] = std::min(c[i][k], a[i][j] + b[j][k]);
17             }
18         }
19         c[i][3] = std::min(c[i][3], a[i][3]);
20     }
21     c[3][3] = 0;
22     return c;
23 }
24
25 struct Node {
26     Node *ch[2], *p;
27     i64 sumg = 0;
28     i64 sumh = 0;
29     i64 sumb = 0;
30     i64 g = 0;
31     i64 h = 0;
32     i64 b = 0;
33     Matrix mat;
34     Matrix prd;
35     std::array<i64, 4> ans{};
36     Node() : ch{nullptr, nullptr}, p(nullptr) {}
37
38     void update() {
39         mat = Matrix(inf);
40         mat[0][0] = b + h - g + sumg;
41         mat[1][1] = mat[1][2] = mat[1][3] = h + sumh;
42         mat[2][0] = mat[2][1] = mat[2][2] = mat[2][3] = b + h + sumb;
43         mat[3][3] = 0;
44     }
45 };
46 void push(Node *t) {
47
48 }
49 void pull(Node *t) {
50     t->prd = (t->ch[0] ? t->ch[0]->prd : Matrix()) * t->mat * (t->ch[1] ? t->ch[1]-
51 >prd : Matrix());
52 }
53 bool isroot(Node *t) {
54     return t->p == nullptr || (t->p->ch[0] != t && t->p->ch[1] != t);
55 }
56 int pos(Node *t) {
57     return t->p->ch[1] == t;
58 }
59 void pushAll(Node *t) {
60     if (!isroot(t)) {
61         pushAll(t->p);
62     }
63     push(t);
64 }
65 void rotate(Node *t) {
66     Node *q = t->p;
67     int x = !pos(t);
68     q->ch[!x] = t->ch[x];
69     if (t->ch[x]) {
70         t->ch[x]->p = q;
71     }
72     t->p = q->p;
73     if (!isroot(q)) {
74         q->p->ch[pos(q)] = t;

```

```

74     }
75     t->ch[x] = q;
76     q->p = t;
77     pull(q);
78 }
79 void splay(Node *t) {
80     pushAll(t);
81     while (!isroot(t)) {
82         if (!isroot(t->p)) {
83             if (pos(t) == pos(t->p)) {
84                 rotate(t->p);
85             } else {
86                 rotate(t);
87             }
88         }
89         rotate(t);
90     }
91     pull(t);
92 }
93
94 std::array<i64, 4> get(Node *t) {
95     std::array<i64, 4> ans;
96     ans.fill(inf);
97     ans[3] = 0;
98     for (int i = 0; i < 3; i++) {
99         for (int j = 0; j < 4; j++) {
100             ans[i] = std::min(ans[i], t->prd[i][j]);
101         }
102     }
103     return ans;
104 }
105
106 void access(Node *t) {
107     std::array<i64, 4> old{};
108     for (Node *i = t, *q = nullptr; i; q = i, i = i->p) {
109         splay(i);
110         if (i->ch[1]) {
111             auto res = get(i->ch[1]);
112             i->sumg += res[0];
113             i->sumh += std::min({res[1], res[2], res[3]});
114             i->sumb += std::min({res[0], res[1], res[2], res[3]});
115         }
116         i->ch[1] = q;
117         i->sumg -= old[0];
118         i->sumh -= std::min({old[1], old[2], old[3]});
119         i->sumb -= std::min({old[0], old[1], old[2], old[3]});
120         old = get(i);
121         i->update();
122         pull(i);
123     }
124     splay(t);
125 }

```

4.7 其他平衡树

```

1 struct Node {
2     Node *l = nullptr;
3     Node *r = nullptr;
4     int sum = 0;
5     int sumodd = 0;
6
7     Node(Node *t) {

```



```

8         if (t) {
9             *this = *t;
10        }
11    }
12};
13
14Node *add(Node *t, int l, int r, int x, int v) {
15    t = new Node(t);
16    t->sum += v;
17    t->sumodd += (x % 2) * v;
18    if (r - l == 1) {
19        return t;
20    }
21    int m = (l + r) / 2;
22    if (x < m) {
23        t->l = add(t->l, l, m, x, v);
24    } else {
25        t->r = add(t->r, m, r, x, v);
26    }
27    return t;
28}
29
30int query1(Node *t1, Node *t2, int l, int r, int k) {
31    if (r - l == 1) {
32        return 1;
33    }
34    int m = (l + r) / 2;
35    int odd = (t1 && t1->r ? t1->r->sumodd : 0) - (t2 && t2->r ? t2->r->sumodd : 0);
36    int cnt = (t1 && t1->r ? t1->r->sum : 0) - (t2 && t2->r ? t2->r->sum : 0);
37    if (odd > 0 || cnt > k) {
38        return query1(t1 ? t1->r : t1, t2 ? t2->r : t2, m, r, k);
39    } else {
40        return query1(t1 ? t1->l : t1, t2 ? t2->l : t2, l, m, k - cnt);
41    }
42}
43
44std::array<int, 3> query2(Node *t1, Node *t2, int l, int r, int k) {
45    if (r - l == 1) {
46        int cnt = (t1 ? t1->sumodd : 0) - (t2 ? t2->sumodd : 0);
47        return {1, cnt, k};
48    }
49    int m = (l + r) / 2;
50    int cnt = (t1 && t1->r ? t1->r->sumodd : 0) - (t2 && t2->r ? t2->r->sumodd : 0);
51    if (cnt > k) {
52        return query2(t1 ? t1->r : t1, t2 ? t2->r : t2, m, r, k);
53    } else {
54        return query2(t1 ? t1->l : t1, t2 ? t2->l : t2, l, m, k - cnt);
55    }
56}

```

```

1 struct Node {
2     Node *l = nullptr;
3     Node *r = nullptr;
4     int cnt = 0;
5 };
6
7 Node *add(Node *t, int l, int r, int x) {
8     if (t) {
9         t = new Node(*t);
10    } else {
11        t = new Node;
12    }
13    t->cnt += 1;

```

```

14     if (r - l == 1) {
15         return t;
16     }
17     int m = (l + r) / 2;
18     if (x < m) {
19         t->l = add(t->l, l, m, x);
20     } else {
21         t->r = add(t->r, m, r, x);
22     }
23     return t;
24 }
25
26 int query(Node *t1, Node *t2, int l, int r, int x) {
27     int cnt = (t2 ? t2->cnt : 0) - (t1 ? t1->cnt : 0);
28     if (cnt == 0 || l >= x) {
29         return -1;
30     }
31     if (r - l == 1) {
32         return l;
33     }
34     int m = (l + r) / 2;
35     int res = query(t1 ? t1->r : t1, t2 ? t2->r : t2, m, r, x);
36     if (res == -1) {
37         res = query(t1 ? t1->l : t1, t2 ? t2->l : t2, l, m, x);
38     }
39     return res;
40 }

```

```

1  struct Info {
2      int imp = 0;
3      int id = 0;
4  };
5
6  Info operator+(Info a, Info b) {
7      return {std::max(a.imp, b.imp), 0};
8  }
9
10 struct Node {
11     int w = rng();
12     Info info;
13     Info sum;
14     int siz = 1;
15     Node *l = nullptr;
16     Node *r = nullptr;
17 };
18
19 void pull(Node *t) {
20     t->sum = t->info;
21     t->siz = 1;
22     if (t->l) {
23         t->sum = t->l->sum + t->sum;
24         t->siz += t->l->siz;
25     }
26     if (t->r) {
27         t->sum = t->sum + t->r->sum;
28         t->siz += t->r->siz;
29     }
30 }
31
32 std::pair<Node *, Node *> splitAt(Node *t, int p) {
33     if (!t) {
34         return {t, t};
35     }

```

```

36     if (p <= (t->l ? t->l->siz : 0)) {
37         auto [l, r] = splitAt(t->l, p);
38         t->l = r;
39         pull(t);
40         return {l, t};
41     } else {
42         auto [l, r] = splitAt(t->r, p - 1 - (t->l ? t->l->siz : 0));
43         t->r = l;
44         pull(t);
45         return {t, r};
46     }
47 }
48
49 void insertAt(Node *&t, int p, Node *x) {
50     if (!t) {
51         t = x;
52         return;
53     }
54     if (x->w < t->w) {
55         auto [l, r] = splitAt(t, p);
56         t = x;
57         t->l = l;
58         t->r = r;
59         pull(t);
60         return;
61     }
62     if (p <= (t->l ? t->l->siz : 0)) {
63         insertAt(t->l, p, x);
64     } else {
65         insertAt(t->r, p - 1 - (t->l ? t->l->siz : 0), x);
66     }
67     pull(t);
68 }
69
70 Node *merge(Node *a, Node *b) {
71     if (!a) {
72         return b;
73     }
74     if (!b) {
75         return a;
76     }
77
78     if (a->w < b->w) {
79         a->r = merge(a->r, b);
80         pull(a);
81         return a;
82     } else {
83         b->l = merge(a, b->l);
84         pull(b);
85         return b;
86     }
87 }
88
89 int query(Node *t, int v) {
90     if (!t) {
91         return 0;
92     }
93     if (t->sum.imp < v) {
94         return t->siz;
95     }
96     int res = query(t->r, v);
97     if (res != (t->r ? t->r->siz : 0)) {
98         return res;
99     }

```

```

100     if (t->info.imp > v) {
101         return res;
102     }
103     return res + 1 + query(t->l, v);
104 }
105
106 void dfs(Node *t) {
107     if (!t) {
108         return;
109     }
110     dfs(t->l);
111     std::cout << t->info.id << " ";
112     dfs(t->r);
113 }

```

```

1  struct Node {
2      Node *l = nullptr;
3      Node *r = nullptr;
4      int cnt = 0;
5      int cntnew = 0;
6  };
7
8  Node *add(int l, int r, int x, int isnew) {
9      Node *t = new Node;
10     t->cnt = 1;
11     t->cntnew = isnew;
12     if (r - l == 1) {
13         return t;
14     }
15     int m = (l + r) / 2;
16     if (x < m) {
17         t->l = add(l, m, x, isnew);
18     } else {
19         t->r = add(m, r, x, isnew);
20     }
21     return t;
22 }
23
24 struct Info {
25     Node *t = nullptr;
26     int psum = 0;
27     bool rev = false;
28 };
29
30 void pull(Node *t) {
31     t->cnt = (t->l ? t->l->cnt : 0) + (t->r ? t->r->cnt : 0);
32     t->cntnew = (t->l ? t->l->cntnew : 0) + (t->r ? t->r->cntnew : 0);
33 }
34
35 std::pair<Node *, Node *> split(Node *t, int l, int r, int x, bool rev) {
36     if (!t) {
37         return {t, t};
38     }
39     if (x == 0) {
40         return {nullptr, t};
41     }
42     if (x == t->cnt) {
43         return {t, nullptr};
44     }
45     if (r - l == 1) {
46         Node *t2 = new Node;
47         t2->cnt = t->cnt - x;
48         t->cnt = x;

```

```

49     return {t, t2};
50 }
51 Node *t2 = new Node;
52 int m = (l + r) / 2;
53 if (!rev) {
54     if (t->l && x <= t->l->cnt) {
55         std::tie(t->l, t2->l) = split(t->l, l, m, x, rev);
56         t2->r = t->r;
57         t->r = nullptr;
58     } else {
59         std::tie(t->r, t2->r) = split(t->r, m, r, x - (t->l ? t->l->cnt : 0),
rev);
60     }
61 } else {
62     if (t->r && x <= t->r->cnt) {
63         std::tie(t->r, t2->r) = split(t->r, m, r, x, rev);
64         t2->l = t->l;
65         t->l = nullptr;
66     } else {
67         std::tie(t->l, t2->l) = split(t->l, l, m, x - (t->r ? t->r->cnt : 0),
rev);
68     }
69 }
70 pull(t);
71 pull(t2);
72 return {t, t2};
73 }
74
75 Node *merge(Node *t1, Node *t2, int l, int r) {
76     if (!t1) {
77         return t2;
78     }
79     if (!t2) {
80         return t1;
81     }
82     if (r - l == 1) {
83         t1->cnt += t2->cnt;
84         t1->cntnew += t2->cntnew;
85         delete t2;
86         return t1;
87     }
88     int m = (l + r) / 2;
89     t1->l = merge(t1->l, t2->l, l, m);
90     t1->r = merge(t1->r, t2->r, m, r);
91     delete t2;
92     pull(t1);
93     return t1;
94 }

```

4.8 分数四则运算 (Frac)

```

1  template<class T>
2  struct Frac {
3      T num;
4      T den;
5      Frac(T num_, T den_) : num(num_), den(den_) {
6          if (den < 0) {
7              den = -den;
8              num = -num;
9          }
10     }
11     Frac() : Frac(0, 1) {}

```

```

12     Frac(T num_) : Frac(num_, 1) {}
13     explicit operator double() const {
14         return 1. * num / den;
15     }
16     Frac &operator+=(const Frac &rhs) {
17         num = num * rhs.den + rhs.num * den;
18         den *= rhs.den;
19         return *this;
20     }
21     Frac &operator-=(const Frac &rhs) {
22         num = num * rhs.den - rhs.num * den;
23         den *= rhs.den;
24         return *this;
25     }
26     Frac &operator*=(const Frac &rhs) {
27         num *= rhs.num;
28         den *= rhs.den;
29         return *this;
30     }
31     Frac &operator/=(const Frac &rhs) {
32         num *= rhs.den;
33         den *= rhs.num;
34         if (den < 0) {
35             num = -num;
36             den = -den;
37         }
38         return *this;
39     }
40     friend Frac operator+(Frac lhs, const Frac &rhs) {
41         return lhs += rhs;
42     }
43     friend Frac operator-(Frac lhs, const Frac &rhs) {
44         return lhs -= rhs;
45     }
46     friend Frac operator*(Frac lhs, const Frac &rhs) {
47         return lhs *= rhs;
48     }
49     friend Frac operator/(Frac lhs, const Frac &rhs) {
50         return lhs /= rhs;
51     }
52     friend Frac operator-(const Frac &a) {
53         return Frac(-a.num, a.den);
54     }
55     friend bool operator==(const Frac &lhs, const Frac &rhs) {
56         return lhs.num * rhs.den == rhs.num * lhs.den;
57     }
58     friend bool operator!=(const Frac &lhs, const Frac &rhs) {
59         return lhs.num * rhs.den != rhs.num * lhs.den;
60     }
61     friend bool operator<(const Frac &lhs, const Frac &rhs) {
62         return lhs.num * rhs.den < rhs.num * lhs.den;
63     }
64     friend bool operator>(const Frac &lhs, const Frac &rhs) {
65         return lhs.num * rhs.den > rhs.num * lhs.den;
66     }
67     friend bool operator<=(const Frac &lhs, const Frac &rhs) {
68         return lhs.num * rhs.den <= rhs.num * lhs.den;
69     }
70     friend bool operator>=(const Frac &lhs, const Frac &rhs) {
71         return lhs.num * rhs.den >= rhs.num * lhs.den;
72     }
73     friend std::ostream &operator<<(std::ostream &os, Frac x) {
74         T g = std::gcd(x.num, x.den);
75         if (x.den == g) {

```

```

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

```

4.9 线性基 (Basis)

```

1  struct Basis {
2      int a[20] {};
3      int t[20] {};
4
5      Basis() {
6          std::fill(t, t + 20, -1);
7      }
8
9      void add(int x, int y = 1E9) {
10         for (int i = 0; i < 20; i++) {
11             if (x >> i & 1) {
12                 if (y > t[i]) {
13                     std::swap(a[i], x);
14                     std::swap(t[i], y);
15                 }
16                 x ^= a[i];
17             }
18         }
19     }
20
21     bool query(int x, int y = 0) {
22         for (int i = 0; i < 20; i++) {
23             if ((x >> i & 1) && t[i] >= y) {
24                 x ^= a[i];
25             }
26         }
27         return x == 0;
28     }
29 };

```

/END/

5 字符串

5.1 马拉车 (Manacher 新版)

```
1 std::vector<int> manacher(std::vector<int> s) {
2     std::vector<int> t{0};
3     for (auto c : s) {
4         t.push_back(c);
5         t.push_back(0);
6     }
7     int n = t.size();
8     std::vector<int> r(n);
9     for (int i = 0, j = 0; i < n; i++) {
10        if (2 * j - i >= 0 && j + r[j] > i) {
11            r[i] = std::min(r[2 * j - i], j + r[j] - i);
12        }
13        while (i - r[i] >= 0 && i + r[i] < n && t[i - r[i]] == t[i + r[i]]) {
14            r[i] += 1;
15        }
16        if (i + r[i] > j + r[j]) {
17            j = i;
18        }
19    }
20    return r;
21 }
```

5.2 z函数

```
1 std::vector<int> zFunction(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 }
```

5.3 后缀数组 (SA)

```
1 struct SuffixArray {
2     int n;
3     std::vector<int> sa, rk, lc;
4     SuffixArray(const std::string &s) {
5         n = s.length();
6         sa.resize(n);
7         lc.resize(n - 1);
8         rk.resize(n);
9         std::iota(sa.begin(), sa.end(), 0);
10        std::sort(sa.begin(), sa.end(), [&](int a, int b) {return s[a] < s[b];});
11        rk[sa[0]] = 0;
12        for (int i = 1; i < n; ++i)
13            rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
14    }
15 }
```



```

14     int k = 1;
15     std::vector<int> tmp, cnt(n);
16     tmp.reserve(n);
17     while (rk[sa[n - 1]] < n - 1) {
18         tmp.clear();
19         for (int i = 0; i < k; ++i)
20             tmp.push_back(n - k + i);
21         for (auto i : sa)
22             if (i >= k)
23                 tmp.push_back(i - k);
24         std::fill(cnt.begin(), cnt.end(), 0);
25         for (int i = 0; i < n; ++i)
26             ++cnt[rk[i]];
27         for (int i = 1; i < n; ++i)
28             cnt[i] += cnt[i - 1];
29         for (int i = n - 1; i >= 0; --i)
30             sa[--cnt[rk[tmp[i]]]] = tmp[i];
31         std::swap(rk, tmp);
32         rk[sa[0]] = 0;
33         for (int i = 1; i < n; ++i)
34             rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i - 1]] < tmp[sa[i]] || sa[i -
1] + k == n || tmp[sa[i - 1] + k] < tmp[sa[i] + k]);
35         k *= 2;
36     }
37     for (int i = 0, j = 0; i < n; ++i) {
38         if (rk[i] == 0) {
39             j = 0;
40         } else {
41             for (j -= j > 0; i + j < n && sa[rk[i] - 1] + j < n && s[i + j] ==
s[sa[rk[i] - 1] + j]; )
42                 ++j;
43             lc[rk[i] - 1] = j;
44         }
45     }
46 }
47 };

```

5.4 后缀自动机

5.4.1 后缀自动机 (SuffixAutomaton 旧版)

```

1  struct SuffixAutomaton {
2      static constexpr int ALPHABET_SIZE = 26, N = 5e5;
3      struct Node {
4          int len;
5          int link;
6          int next[ALPHABET_SIZE];
7          Node() : len(0), link(0), next{} {}
8      } t[2 * N];
9      int cntNodes;
10     SuffixAutomaton() {
11         cntNodes = 1;
12         std::fill(t[0].next, t[0].next + ALPHABET_SIZE, 1);
13         t[0].len = -1;
14     }
15     int extend(int p, int c) {
16         if (t[p].next[c]) {
17             int q = t[p].next[c];
18             if (t[q].len == t[p].len + 1)
19                 return q;
20             int r = ++cntNodes;
21             t[r].len = t[p].len + 1;

```

```

22         t[r].link = t[q].link;
23         std::copy(t[q].next, t[q].next + ALPHABET_SIZE, t[r].next);
24         t[q].link = r;
25         while (t[p].next[c] == q) {
26             t[p].next[c] = r;
27             p = t[p].link;
28         }
29         return r;
30     }
31     int cur = ++cntNodes;
32     t[cur].len = t[p].len + 1;
33     while (!t[p].next[c]) {
34         t[p].next[c] = cur;
35         p = t[p].link;
36     }
37     t[cur].link = extend(p, c);
38     return cur;
39 }
40 };

```

5.4.2 后缀自动机 (SAM 新版)

```

1  struct SAM {
2      static constexpr int ALPHABET_SIZE = 26;
3      struct Node {
4          int len;
5          int link;
6          std::array<int, ALPHABET_SIZE> next;
7          Node() : len{}, link{}, next{} {}
8      };
9      std::vector<Node> t;
10     SAM() {
11         init();
12     }
13     void init() {
14         t.assign(2, Node());
15         t[0].next.fill(1);
16         t[0].len = -1;
17     }
18     int newNode() {
19         t.emplace_back();
20         return t.size() - 1;
21     }
22     int extend(int p, int c) {
23         if (t[p].next[c]) {
24             int q = t[p].next[c];
25             if (t[q].len == t[p].len + 1) {
26                 return q;
27             }
28             int r = newNode();
29             t[r].len = t[p].len + 1;
30             t[r].link = t[q].link;
31             t[r].next = t[q].next;
32             t[q].link = r;
33             while (t[p].next[c] == q) {
34                 t[p].next[c] = r;
35                 p = t[p].link;
36             }
37             return r;
38         }
39         int cur = newNode();
40         t[cur].len = t[p].len + 1;

```

```

41     while (!t[p].next[c]) {
42         t[p].next[c] = cur;
43         p = t[p].link;
44     }
45     t[cur].link = extend(p, c);
46     return cur;
47 }
48 int extend(int p, char c, char offset = 'a') {
49     return extend(p, c - offset);
50 }
51
52 int next(int p, int x) {
53     return t[p].next[x];
54 }
55
56 int next(int p, char c, char offset = 'a') {
57     return next(p, c - 'a');
58 }
59
60 int link(int p) {
61     return t[p].link;
62 }
63
64 int len(int p) {
65     return t[p].len;
66 }
67
68 int size() {
69     return t.size();
70 }
71 };

```

5.5 回文自动机 (PAM)

```

1  struct PAM {
2      static constexpr int ALPHABET_SIZE = 28;
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
27     bool add(char c, char offset = 'a') {
28         int pos = s.size();

```

```

29     s += c;
30     int let = c - offset;
31     int cur = suff, curlen = 0;
32
33     while (true) {
34         curlen = t[cur].len;
35         if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])
36             break;
37         cur = t[cur].link;
38     }
39     if (t[cur].next[let]) {
40         suff = t[cur].next[let];
41         return false;
42     }
43
44     int num = newNode();
45     suff = num;
46     t[num].len = t[cur].len + 2;
47     t[cur].next[let] = num;
48
49     if (t[num].len == 1) {
50         t[num].link = 1;
51         t[num].cnt = 1;
52         return true;
53     }
54
55     while (true) {
56         cur = t[cur].link;
57         curlen = t[cur].len;
58         if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
59             t[num].link = t[cur].next[let];
60             break;
61         }
62     }
63
64     t[num].cnt = 1 + t[t[num].link].cnt;
65
66     return true;
67 }
68 };
69
70 PAM pam;

```

5.6 AC自动机

5.6.1 AC自动机 (AC 旧版)

```

1  constexpr int N = 3e5 + 30, A = 26;
2
3  struct Node {
4      int fail;
5      int sum;
6      int next[A];
7      Node() : fail(-1), sum(0) {
8          std::memset(next, -1, sizeof(next));
9      }
10 } node[N];
11
12 int cnt = 0;
13 int bin[N];
14 int nBin = 0;
15

```

```

16 int newNode() {
17     int p = nBin > 0 ? bin[--nBin] : cnt++;
18     node[p] = Node();
19     return p;
20 }
21
22 struct AC {
23     std::vector<int> x;
24     AC(AC &&a) : x(std::move(a.x)) {}
25     AC(std::vector<std::string> s, std::vector<int> w) {
26         x = {newNode(), newNode()};
27         std::fill(node[x[0]].next, node[x[0]].next + A, x[1]);
28         node[x[1]].fail = x[0];
29
30         for (int i = 0; i < int(s.size()); i++) {
31             int p = x[1];
32             for (int j = 0; j < int(s[i].length()); j++) {
33                 int c = s[i][j] - 'a';
34                 if (node[p].next[c] == -1) {
35                     int u = newNode();
36                     x.push_back(u);
37                     node[p].next[c] = u;
38                 }
39                 p = node[p].next[c];
40             }
41             node[p].sum += w[i];
42         }
43
44         std::queue<int> que;
45         que.push(x[1]);
46         while (!que.empty()) {
47             int u = que.front();
48             que.pop();
49             node[u].sum += node[node[u].fail].sum;
50             for (int c = 0; c < A; c++) {
51                 if (node[u].next[c] == -1) {
52                     node[u].next[c] = node[node[u].fail].next[c];
53                 } else {
54                     node[node[u].next[c]].fail = node[node[u].fail].next[c];
55                     que.push(node[u].next[c]);
56                 }
57             }
58         }
59     }
60     ~AC() {
61         for (auto p : x) {
62             bin[nBin++] = p;
63         }
64     }
65     i64 query(const std::string &s) const {
66         i64 ans = 0;
67         int p = x[1];
68         for (int i = 0; i < int(s.length()); i++) {
69             int c = s[i] - 'a';
70             p = node[p].next[c];
71             ans += node[p].sum;
72         }
73         return ans;
74     }
75 };

```

5.6.2 AC自动机 (AhoCorasick 新新版)

```
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 };

```

5.7 随机生成模底 字符串哈希（例题）

```

1  #include <bits/stdc++.h>
2
3  using i64 = long long;
4
5  bool isprime(int n) {
6      if (n <= 1) {
7          return false;
8      }
9      for (int i = 2; i * i <= n; i++) {
10         if (n % i == 0) {
11             return false;
12         }
13     }
14     return true;
15 }
16
17 int findPrime(int n) {
18     while (!isprime(n)) {
19         n++;
20     }
21     return n;
22 }
23
24 using Hash = std::array<int, 2>;
25
26 int main() {
27     std::ios::sync_with_stdio(false);
28     std::cin.tie(nullptr);
29
30     std::mt19937 rng(std::chrono::steady_clock::now().time_since_epoch().count());
31
32     const int P = findPrime(rng() % 900000000 + 100000000);
33
34     std::string s, x;
35     std::cin >> s >> x;
36
37     int n = s.length();
38     int m = x.length();
39
40     std::vector<int> h(n + 1), p(n + 1);
41     for (int i = 0; i < n; i++) {
42         h[i + 1] = (10LL * h[i] + s[i] - '0') % P;
43     }
44     p[0] = 1;
45     for (int i = 0; i < n; i++) {
46         p[i + 1] = 10LL * p[i] % P;

```

```

47     }
48
49     auto get = [&](int l, int r) {
50         return (h[r] + 1LL * (P - h[l]) * p[r - l]) % P;
51     };
52
53     int px = 0;
54     for (auto c : x) {
55         px = (10LL * px + c - '0') % P;
56     }
57
58     for (int i = 0; i <= n - 2 * (m - 1); i++) {
59         if ((get(i, i + m - 1) + get(i + m - 1, i + 2 * m - 2)) % P == px) {
60             std::cout << i + 1 << " " << i + m - 1 << "\n";
61             std::cout << i + m << " " << i + 2 * m - 2 << "\n";
62             return 0;
63         }
64     }
65
66     std::vector<int> z(m + 1), f(n + 1);
67     z[0] = m;
68
69     for (int i = 1, j = -1; i < m; i++) {
70         if (j != -1) {
71             z[i] = std::max(0, std::min(j + z[j] - i, z[i - j]));
72         }
73         while (z[i] + i < m && x[z[i]] == x[z[i] + i]) {
74             z[i]++;
75         }
76         if (j == -1 || i + z[i] > j + z[j]) {
77             j = i;
78         }
79     }
80     for (int i = 0, j = -1; i < n; i++) {
81         if (j != -1) {
82             f[i] = std::max(0, std::min(j + f[j] - i, z[i - j]));
83         }
84         while (f[i] + i < n && f[i] < m && x[f[i]] == s[f[i] + i]) {
85             f[i]++;
86         }
87         if (j == -1 || i + f[i] > j + f[j]) {
88             j = i;
89         }
90     }
91
92     for (int i = 0; i + m <= n; i++) {
93         int l = std::min(m, f[i]);
94
95         for (auto j : { m - 1, m - 1 - 1 }) {
96             if (j <= 0) {
97                 continue;
98             }
99             if (j <= i && (get(i - j, i) + get(i, i + m)) % P == px) {
100                 std::cout << i - j + 1 << " " << i << "\n";
101                 std::cout << i + 1 << " " << i + m << "\n";
102                 return 0;
103             }
104             if (i + m + j <= n && (get(i, i + m) + get(i + m, i + m + j)) % P ==
px) {
105                 std::cout << i + 1 << " " << i + m << "\n";
106                 std::cout << i + m + 1 << " " << i + m + j << "\n";
107                 return 0;
108             }
109         }

```



```
110     }  
111  
112     return 0;  
113 }
```

Edited by ***Wida***
Version: 8.5 (2024.06.01)

github.com/hh2048
www.cnblogs.com/WIDA