



WIDA's XCPC Algorithm Template (II)

(自收集 jiangly 模板合集)

v8.7 2024.10.07

Author: jiangly · Tyrp: WIDA

目录

杂类	1
int128 库函数自定义	1
常用库函数重载	1
字符调整	2
二分算法	3
二分算法（整数域）	3
二分算法（实数域）	3
图与网络	5
强连通分量缩点（SCC）	5
割边与割边缩点（EBCC）	6
二分图最大权匹配（MaxAssignment 基于 KM）	7
一般图最大匹配（Graph 带花树算法）【久远】	9
TwoSat（2-Sat）	13
最大流（MaxFlow 新版）	13
费用流	15
费用流（MCFGraph 旧版）	15
费用流（MinCostFlow 新版）	17
树链剖分（HLD）	18
数论、几何、多项式	21
快速幂	21
基姆拉尔森公式	21
欧拉筛	22
莫比乌斯函数筛（莫比乌斯反演）	23
扩展欧几里得（exgcd）	24
欧拉函数	24
欧拉函数（求解单个数的欧拉函数）	24
欧拉函数（求解全部数的欧拉函数）	25
组合数	25
组合数（小范围预处理，逆元+杨辉三角）	25
组合数（Comb, with. ModIntBase）	26
素数测试与因式分解（Miller–Rabin & Pollard–Rho）	27
平面几何	28
平面几何（Point）	28
平面几何（with. complex）	34
立体几何（Point）	35
静态凸包	36
静态凸包（with. Point, 新版）	36
静态凸包（with. complex）	37
多项式	38
多项式（Poly, 旧版）	38
多项式（Poly, with. MInt & MLong）	42
多项式乘法	49
生成函数	51
生成函数（q-int）	51
生成函数（q-Binomial）	51
生成函数（Binomial 任意模数二项式）	52

自适应辛普森法 (Simpson)	54
矩阵 (Matrix)	54
高斯消元法 (gaussian elimination)【久远】	55
数据结构	57
树状数组 (Fenwick)	57
并查集	57
并查集 (DSU)	57
可撤销并查集 (DSU With Rollback)	58
线段树	59
线段树 (SegmentTree+Info 区间加+单点修改)	59
线段树 (SegmentTree 区间乘+单点加)	60
线段树 (SegmentTree+Info 初始赋值+单点修改+查找前驱后继)	61
线段树 (SegmentTree+Info+Merge 初始赋值+单点修改+区间合并)	63
懒标记线段树 (LazySegmentTree)	64
取模类	67
取模类 (Z 旧版)	67
取模类 (MLong & MInt 新版)	69
动态取模类 (ModIntBase)	72
状压 RMQ (RMQ)	74
Splay	75
其他平衡树	83
分数四则运算 (Frac)	88
线性基 (Basis)	89
高精度 (BigInt)	89
Link-Cut Tree	90
字符串	95
马拉车 (Manacher)	95
Z 函数	95
后缀数组	95
后缀数组 (SuffixArray 旧版)	95
后缀数组 (SA 及其应用 新版)	96
后缀自动机	98
后缀自动机 (SuffixAutomaton 旧版)	98
后缀自动机 (SAM 新版)	99
回文自动机 (PAM)	100
AC 自动机	101
AC 自动机 (AC 旧版)	101
AC 自动机 (AhoCorasick, with vector 新版)	102
AC 自动机 (AhoCorasick, with string 新版)	104
字符串哈希 (随机底模例题)	105
最长公共前缀 LCP (例题)	107
字典树 Trie	108
前缀函数 (KMP)	111

个人收集，仅供参考。如有需要，您可以通过以下渠道获取最新版本或与我取得联系

www.github.com/hh2048

WIDA, 2024.10.07

1 杂类

1.1 int128 库函数自定义

```

1 ostream &operator<<(ostream &os, i128 n) {
2     if (n == 0) {
3         return os << 0;
4     }
5     string s;
6     while (n > 0) {
7         s += char('0' + n % 10);
8         n /= 10;
9     }
10    reverse(s.begin(), s.end());
11    return os << s;
12 }
13 i128 toi128(const string &s) {
14     i128 n = 0;
15     for (auto c : s) {
16         n = n * 10 + (c - '0');
17     }
18     return n;
19 }
20 i128 sqrti128(i128 n) {
21     i128 lo = 0, hi = 1E16;
22     while (lo < hi) {
23         i128 x = (lo + hi + 1) / 2;
24         if (x * x <= n) {
25             lo = x;
26         } else {
27             hi = x - 1;
28         }
29     }
30     return lo;
31 }
32
33 i128 gcd(i128 a, i128 b) {
34     while (b) {
35         a %= b;
36         swap(a, b);
37     }
38     return a;
39 }
```

1.2 常用库函数重载

```

1 /** 上取整下取整 */
2 i64 ceilDiv(i64 n, i64 m) {
3     if (n >= 0) {
4         return (n + m - 1) / m;
5     } else {
6         return n / m;
7     }
8 }
9 i64 floorDiv(i64 n, i64 m) {
10    if (n >= 0) {
11        return n / m;
12    } else {
13        return (n - m + 1) / m;
14    }
}
```

```

15 }
16 /**
17 template<class T>
18 void chmax(T &a, T b) {
19     if (a < b) {
20         a = b;
21     }
22 }
23 /**
24 i128 gcd(i128 a, i128 b) {
25     return b ? gcd(b, a % b) : a;
26 }
27 /**
28 i64 sqrt(i64 n) {
29     i64 s = sqrt(n);
30     while (s * s > n) {
31         s--;
32     }
33     while ((s + 1) * (s + 1) <= n) {
34         s++;
35     }
36     return s;
37 }
38 /**
39 i64 get(i64 n) {
40     i64 u = sqrt(2.0L * n);
41     while (u * (u + 1) / 2 < n) {
42         u++;
43     }
44     while (u * (u - 1) / 2 + 1 > n) {
45         u--;
46     }
47     return u;
48 }
49 /**
50 int logi(int a, int b) {
51     int t = 0;
52     i64 v = 1;
53     while (v < b) {
54         v *= a;
55         t++;
56     }
57     return t;
58 }
59 int llog(int a, int b) {
60     if (a <= b) {
61         int l = logi(a, b);
62         return (l == 0 ? 0 : __lg(2 * l - 1));
63     }
64     int l = logi(b, a + 1) - 1;
65     assert(l > 0);
66     return -__lg(l);
67 }

```

1.3 字符调整

```

1 /**
2 void rev(string &s) {
3     int l = s.size();
4     for (int i = 1; i < l; i += 2) {
5         if (isupper(s[i])) {
6             s[i] = tolower(s[i]);

```

```

7     } else {
8         s[i] = toupper(s[i]);
9     }
10 }
11 }
12
13 int get(char c) {
14     int x;
15     if (islower(c)) {
16         x = c - 'a';
17     } else {
18         x = 26 + c - 'A';
19     }
20     return x;
21 }
```

1.4 二分算法

1.4.1 二分算法（整数域）

```

1  /** 二分算法（整数域）：前驱 */
2  int lo = 1, hi = 1E9;
3  while (lo < hi) {
4      int m = (lo + hi + 1) / 2;
5      if (check(m)) {
6          lo = m;
7      } else {
8          hi = m - 1;
9      }
10 }
11 cout << lo << "\n";
12 /** 二分算法（整数域）：后继 */
13 int lo = 1, hi = n;
14 while (lo < hi) {
15     int m = (lo + hi) / 2;
16     if (check(m)) {
17         hi = m;
18     } else {
19         lo = m + 1;
20     }
21 }
22 cout << lo << "\n";
```

1.4.2 二分算法（实数域）

```

1  /** 二分算法（实数域） */
2  auto check = [&](double t) {
3      // write
4  };
5
6  double lo = 0;
7  double hi = 1E12;
8  while (hi - lo > max(1.0, lo) * eps) {
9      double x = (lo + hi) / 2;
10     if (check(x)) {
11         hi = x;
12     } else {
13         lo = x;
14     }
15 }
```

```
17 cout << lo << "\n";
18
19 /** 二分算法（实数域） */
20 using i64 = long long;
21 using real = long double;
22
23 constexpr real eps = 1E-7;
24
25 auto get = [&](const auto &f) {
26     real lo = -1E4, hi = 1E4;
27     while (hi - lo > 3 * eps) {
28         real x1 = (lo + hi - eps) / 2;
29         real x2 = (lo + hi + eps) / 2;
30         if (f(x1) > f(x2)) {
31             lo = x1;
32         } else {
33             hi = x2;
34         }
35     }
36     return f((lo + hi) / 2);
37 };
38
39 cout << get([&](real px) {
40     return get([&](real py) {
41         // write
42     });
43 }) << "\n";
```

/END/

2 图与网络

2.1 强连通分量缩点 (SCC)

```

1 struct SCC {
2     int n;
3     vector<vector<int>> adj;
4     vector<int> stk;
5     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] = min(low[x], low[y]);
35             } else if (bel[y] == -1) {
36                 low[x] = 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     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     }

```

59 | };

2.2 割边与割边缩点 (EBCC)

```

1  set<pair<int, int>> E;
2
3  struct EBCC {
4      int n;
5      vector<vector<int>> adj;
6      vector<int> stk;
7      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] = min(low[x], low[y]);
42             } else if (bel[y] == -1 && dfn[y] < dfn[x]) {
43                 E.emplace(x, y);
44                 low[x] = 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     vector<int> work() {
60         dfs(0, -1);
61         return bel;
62     }
63
64     struct Graph {
65         int n;
66         vector<pair<int, int>> edges;
67         vector<int> siz;
68         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)

```

1 constexpr int inf = 1E7;
2 template<class T>
3 struct MaxAssignment {
4     public:
5         T solve(int nx, int ny, vector<vector<T>> a) {
6             assert(0 <= nx && nx <= ny);
7             assert(int(a.size()) == nx);
8             for (int i = 0; i < nx; ++i) {
9                 assert(int(a[i].size()) == ny);
10                for (auto x : a[i])
11                    assert(x >= 0);
12            }
13
14            auto update = [&](int x) {
15                for (int y = 0; y < ny; ++y) {
16                    if (lx[x] + ly[y] - a[x][y] < slack[y]) {
17                        slack[y] = lx[x] + ly[y] - a[x][y];
18                        slackx[y] = x;
19                    }
20                }
21            };
22
23            costs.resize(nx + 1);
24            costs[0] = 0;
25            lx.assign(nx, numeric_limits<T>::max());
26            ly.assign(ny, 0);
27            xy.assign(nx, -1);
28            yx.assign(ny, -1);
29            slackx.resize(ny);
30            for (int cur = 0; cur < nx; ++cur) {

```

```

31     queue<int> que;
32     visx.assign(nx, false);
33     visy.assign(ny, false);
34     slack.assign(ny, numeric_limits<T>::max());
35     p.assign(nx, -1);
36
37     for (int x = 0; x < nx; ++x) {
38         if (xy[x] == -1) {
39             que.push(x);
40             visx[x] = true;
41             update(x);
42         }
43     }
44
45     int ex, ey;
46     bool found = false;
47     while (!found) {
48         while (!que.empty() && !found) {
49             auto x = que.front();
50             que.pop();
51             for (int y = 0; y < ny; ++y) {
52                 if (a[x][y] == lx[x] + ly[y] && !visy[y]) {
53                     if (yx[y] == -1) {
54                         ex = x;
55                         ey = y;
56                         found = true;
57                         break;
58                     }
59                     que.push(yx[y]);
60                     p[yx[y]] = x;
61                     visy[y] = visx[yx[y]] = true;
62                     update(yx[y]);
63                 }
64             }
65         }
66         if (found)
67             break;
68
69         T delta = numeric_limits<T>::max();
70         for (int y = 0; y < ny; ++y)
71             if (!visy[y])
72                 delta = min(delta, slack[y]);
73         for (int x = 0; x < nx; ++x)
74             if (visx[x])
75                 lx[x] -= delta;
76         for (int y = 0; y < ny; ++y) {
77             if (visy[y]) {
78                 ly[y] += delta;
79             } else {
80                 slack[y] -= delta;
81             }
82         }
83         for (int y = 0; y < ny; ++y) {
84             if (!visy[y] && slack[y] == 0) {
85                 if (yx[y] == -1) {
86                     ex = slackx[y];
87                     ey = y;
88                     found = true;
89                     break;
90                 }
91                 que.push(yx[y]);
92                 p[yx[y]] = slackx[y];
93                 visy[y] = visx[yx[y]] = true;
94                 update(yx[y]);
95             }
96         }
97     }
98 }
```

```

95         }
96     }
97 }
98
99     costs[cur + 1] = costs[cur];
100    for (int x = ex, y = ey, ty; x != -1; x = p[x], y = ty) {
101        costs[cur + 1] += a[x][y];
102        if (xy[x] != -1)
103            costs[cur + 1] -= a[x][xy[x]];
104        ty = xy[x];
105        xy[x] = y;
106        yx[y] = x;
107    }
108 }
109 return costs[nx];
110 }
111 vector<int> assignment() {
112     return xy;
113 }
114 pair<vector<T>, vector<T>> labels() {
115     return make_pair(lx, ly);
116 }
117 vector<T> weights() {
118     return costs;
119 }
120 private:
121     vector<T> lx, ly, slack, costs;
122     vector<int> xy, yx, p, slackx;
123     vector<bool> visx, visy;
124 };

```

2.4 一般图最大匹配 (Graph 带花树算法) 【久远】

```

1  /** 一般图最大匹配 (Graph 带花树算法) */
2  struct Graph {
3      int n;
4      vector<vector<int>> e;
5      Graph(int n) : n(n), e(n) {}
6      void addEdge(int u, int v) {
7          e[u].push_back(v);
8          e[v].push_back(u);
9      }
10     vector<int> findMatching(int m, const auto &init) {
11         vector<int> match(n, -1), vis(n), link(n), f(n), dep(n);
12         for (auto [x, y] : init) {
13             match[x] = y;
14             match[y] = x;
15         }
16         // disjoint set union
17         auto find = [&](int u) {
18             while (f[u] != u)
19                 u = f[u] = f[f[u]];
20             return u;
21         };
22         auto lca = [&](int u, int v) {
23             u = find(u);
24             v = find(v);
25             while (u != v) {
26                 if (dep[u] < dep[v])
27                     swap(u, v);
28                 u = find(link[match[u]]);
29             }

```

```

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

```

```

91             match[x] = y;
92         }
93     }
94 }
95 }
96 }
97 for (int u = 0; u < m; ++u)
98     if (match[u] == -1)
99         augment(u);
100 return match;
101 }
102 };
103
104 /**
105  * 一般图最大匹配 (Graph 带花树算法) 【久远】 */
106 struct Graph {
107     int n;
108     vector<vector<int>> e;
109     Graph(int n) : n(n), e(n) {}
110     void addEdge(int u, int v) {
111         e[u].push_back(v);
112         e[v].push_back(u);
113     }
114     vector<int> findMatching() {
115         vector<int> match(n, -1), vis(n), link(n), f(n), dep(n);
116
117         // disjoint set union
118         auto find = [&](int u) {
119             while (f[u] != u)
120                 u = f[u] = f[f[u]];
121             return u;
122         };
123
124         auto lca = [&](int u, int v) {
125             u = find(u);
126             v = find(v);
127             while (u != v) {
128                 if (dep[u] < dep[v])
129                     swap(u, v);
130                 u = find(link[match[u]]);
131             }
132             return u;
133         };
134
135         queue<int> que;
136         auto blossom = [&](int u, int v, int p) {
137             while (find(u) != p) {
138                 link[u] = v;
139                 v = match[u];
140                 if (vis[v] == 0) {
141                     vis[v] = 1;
142                     que.push(v);
143                 }
144                 f[u] = f[v] = p;
145                 u = link[v];
146             }
147         };
148
149         // find an augmenting path starting from u and augment (if exist)
150         auto augment = [&](int u) {
151
152             while (!que.empty())
153                 que.pop();
154
155             iota(f.begin(), f.end(), 0);

```

```

155
156         // vis = 0 corresponds to inner vertices, vis = 1 corresponds to outer
157 vertices
158     fill(vis.begin(), vis.end(), -1);
159
160     que.push(u);
161     vis[u] = 1;
162     dep[u] = 0;
163
164     while (!que.empty()) {
165         int u = que.front();
166         que.pop();
167         for (auto v : e[u]) {
168             if (vis[v] == -1) {
169
170                 vis[v] = 0;
171                 link[v] = u;
172                 dep[v] = dep[u] + 1;
173
174                 // found an augmenting path
175                 if (match[v] == -1) {
176                     for (int x = v, y = u, temp; y != -1; x = temp, y =
177 == -1 ? -1 : link[x]) {
178                         temp = match[y];
179                         match[x] = y;
180                         match[y] = x;
181                     }
182                 }
183
184                 vis[match[v]] = 1;
185                 dep[match[v]] = dep[u] + 2;
186                 que.push(match[v]);
187
188             } else if (vis[v] == 1 && find(v) != find(u)) {
189                 // found a blossom
190                 int p = lca(u, v);
191                 blossom(u, v, p);
192                 blossom(v, u, p);
193             }
194         }
195     };
196
197
198     // find a maximal matching greedily (decrease constant)
199     auto greedy = [&]() {
200         for (int u = 0; u < n; ++u) {
201             if (match[u] != -1)
202                 continue;
203             for (auto v : e[u]) {
204                 if (match[v] == -1) {
205                     match[u] = v;
206                     match[v] = u;
207                     break;
208                 }
209             }
210         }
211     };
212
213     greedy();
214
215     for (int u = 0; u < n; ++u)
216         if (match[u] == -1)

```

```

217         augment(u);
218
219     return match;
220 }
221 };

```

2.5 TwoSat (2-Sat)

```

1 struct TwoSat {
2     int n;
3     vector<vector<int>> e;
4     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        vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 * n, -1);
12        vector<int> stk;
13        int now = 0, cnt = 0;
14        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] = min(low[u], low[v]);
21                } else if (id[v] == -1) {
22                    low[u] = 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    vector<bool> answer() { return ans; }
43 };

```

2.6 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  vector<_Edge> e;
12  vector<vector<int>> g;
13  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      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, 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, numeric_limits<T>::max());
81     }
82     return ans;
83 }
84
85 vector<bool> minCut() {
86     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 vector<Edge> edges() {
100    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 旧版)

```

1  /**
2   * 费用流 (MCFGraph 旧版)
3   * 下方为最小费用**最大流**模板，如需求解最小费用**可行流**，需要去除建边限制
4 */
5 struct MCFGraph {
6     struct Edge {
7         int v, c, f;
8         Edge(int v, int c, int f) : v(v), c(c), f(f) {}
9     };
10    const int n;
11    vector<Edge> e;
12    vector<vector<int>> g;
13    vector<i64> h, dis;
14    vector<int> pre;
15    bool dijkstra(int s, int t) {
16        dis.assign(n, numeric_limits<i64>::max());
17        pre.assign(n, -1);
18    }
19
20    void addEdge(int u, int v, int c, int f) {
21        e.push_back(Edge{u, v, c, f});
22        g[u].push_back(e.size() - 1);
23        g[v].push_back(e.size());
24    }
25
26    void removeEdge(int u, int v) {
27        auto it1 = std::find(g[u].begin(), g[u].end(), e.size());
28        auto it2 = std::find(g[v].begin(), g[v].end(), e.size());
29        if (it1 != g[u].end() & it2 != g[v].end()) {
30            g[u].erase(it1);
31            g[v].erase(it2);
32        }
33    }
34
35    void reverseEdge(int u, int v) {
36        auto it1 = std::find(g[u].begin(), g[u].end(), e.size());
37        auto it2 = std::find(g[v].begin(), g[v].end(), e.size());
38        if (it1 != g[u].end() & it2 != g[v].end()) {
39            g[u].erase(it1);
40            g[v].push_back(e.size());
41        }
42    }
43
44    void updateFlow(int u, int v, int f) {
45        auto it1 = std::find(g[u].begin(), g[u].end(), e.size());
46        auto it2 = std::find(g[v].begin(), g[v].end(), e.size());
47        if (it1 != g[u].end() & it2 != g[v].end()) {
48            e[e.size() - 1].f -= f;
49            e[e.size()].f += f;
50        }
51    }
52
53    void print() {
54        for (int i = 0; i < n; i++) {
55            cout << "v" << i << ": ";
56            for (int j = 0; j < g[i].size(); j++) {
57                cout << e[g[i][j]].v << " ";
58            }
59            cout << endl;
60        }
61    }
62
63    void printPath() {
64        for (int i = 0; i < n; i++) {
65            cout << "v" << i << ": ";
66            for (int j = 0; j < g[i].size(); j++) {
67                cout << e[g[i][j]].v << " ";
68            }
69            cout << endl;
70        }
71    }
72
73    void printFlow() {
74        for (int i = 0; i < n; i++) {
75            cout << "v" << i << ": ";
76            for (int j = 0; j < g[i].size(); j++) {
77                cout << e[g[i][j]].flow << " ";
78            }
79            cout << endl;
80        }
81    }
82
83    void printCapacity() {
84        for (int i = 0; i < n; i++) {
85            cout << "v" << i << ": ";
86            for (int j = 0; j < g[i].size(); j++) {
87                cout << e[g[i][j]].cap << " ";
88            }
89            cout << endl;
90        }
91    }
92
93    void printH() {
94        for (int i = 0; i < n; i++) {
95            cout << "v" << i << ": ";
96            for (int j = 0; j < g[i].size(); j++) {
97                cout << h[g[i][j]] << " ";
98            }
99            cout << endl;
100    }
101
102    void printDis() {
103        for (int i = 0; i < n; i++) {
104            cout << "v" << i << ": ";
105            for (int j = 0; j < g[i].size(); j++) {
106                cout << dis[g[i][j]] << " ";
107            }
108            cout << endl;
109        }
110    }
111
112    void printPre() {
113        for (int i = 0; i < n; i++) {
114            cout << "v" << i << ": ";
115            for (int j = 0; j < g[i].size(); j++) {
116                cout << pre[g[i][j]] << " ";
117            }
118            cout << endl;
119        }
120    }
121
122    void printE() {
123        for (int i = 0; i < n; i++) {
124            cout << "v" << i << ": ";
125            for (int j = 0; j < g[i].size(); j++) {
126                cout << e[g[i][j]].c << " ";
127            }
128            cout << endl;
129        }
130    }
131
132    void printF() {
133        for (int i = 0; i < n; i++) {
134            cout << "v" << i << ": ";
135            for (int j = 0; j < g[i].size(); j++) {
136                cout << e[g[i][j]].f << " ";
137            }
138            cout << endl;
139        }
140    }
141
142    void printG() {
143        for (int i = 0; i < n; i++) {
144            cout << "v" << i << ": ";
145            for (int j = 0; j < g[i].size(); j++) {
146                cout << g[i][j] << " ";
147            }
148            cout << endl;
149        }
150    }
151
152    void printH() {
153        for (int i = 0; i < n; i++) {
154            cout << "v" << i << ": ";
155            for (int j = 0; j < g[i].size(); j++) {
156                cout << h[g[i][j]] << " ";
157            }
158            cout << endl;
159        }
160    }
161
162    void printDis() {
163        for (int i = 0; i < n; i++) {
164            cout << "v" << i << ": ";
165            for (int j = 0; j < g[i].size(); j++) {
166                cout << dis[g[i][j]] << " ";
167            }
168            cout << endl;
169        }
170    }
171
172    void printPre() {
173        for (int i = 0; i < n; i++) {
174            cout << "v" << i << ": ";
175            for (int j = 0; j < g[i].size(); j++) {
176                cout << pre[g[i][j]] << " ";
177            }
178            cout << endl;
179        }
180    }
181
182    void printE() {
183        for (int i = 0; i < n; i++) {
184            cout << "v" << i << ": ";
185            for (int j = 0; j < g[i].size(); j++) {
186                cout << e[g[i][j]].c << " ";
187            }
188            cout << endl;
189        }
190    }
191
192    void printF() {
193        for (int i = 0; i < n; i++) {
194            cout << "v" << i << ": ";
195            for (int j = 0; j < g[i].size(); j++) {
196                cout << e[g[i][j]].f << " ";
197            }
198            cout << endl;
199        }
200    }
201
202    void printG() {
203        for (int i = 0; i < n; i++) {
204            cout << "v" << i << ": ";
205            for (int j = 0; j < g[i].size(); j++) {
206                cout << g[i][j] << " ";
207            }
208            cout << endl;
209        }
210    }
211
212    void printH() {
213        for (int i = 0; i < n; i++) {
214            cout << "v" << i << ": ";
215            for (int j = 0; j < g[i].size(); j++) {
216                cout << h[g[i][j]] << " ";
217            }
218            cout << endl;
219        }
220    }
221
222    void printDis() {
223        for (int i = 0; i < n; i++) {
224            cout << "v" << i << ": ";
225            for (int j = 0; j < g[i].size(); j++) {
226                cout << dis[g[i][j]] << " ";
227            }
228            cout << endl;
229        }
230    }
231
232    void printPre() {
233        for (int i = 0; i < n; i++) {
234            cout << "v" << i << ": ";
235            for (int j = 0; j < g[i].size(); j++) {
236                cout << pre[g[i][j]] << " ";
237            }
238            cout << endl;
239        }
240    }
241
242    void printE() {
243        for (int i = 0; i < n; i++) {
244            cout << "v" << i << ": ";
245            for (int j = 0; j < g[i].size(); j++) {
246                cout << e[g[i][j]].c << " ";
247            }
248            cout << endl;
249        }
250    }
251
252    void printF() {
253        for (int i = 0; i < n; i++) {
254            cout << "v" << i << ": ";
255            for (int j = 0; j < g[i].size(); j++) {
256                cout << e[g[i][j]].f << " ";
257            }
258            cout << endl;
259        }
260    }
261
262    void printG() {
263        for (int i = 0; i < n; i++) {
264            cout << "v" << i << ": ";
265            for (int j = 0; j < g[i].size(); j++) {
266                cout << g[i][j] << " ";
267            }
268            cout << endl;
269        }
270    }
271
272    void printH() {
273        for (int i = 0; i < n; i++) {
274            cout << "v" << i << ": ";
275            for (int j = 0; j < g[i].size(); j++) {
276                cout << h[g[i][j]] << " ";
277            }
278            cout << endl;
279        }
280    }
281
282    void printDis() {
283        for (int i = 0; i < n; i++) {
284            cout << "v" << i << ": ";
285            for (int j = 0; j < g[i].size(); j++) {
286                cout << dis[g[i][j]] << " ";
287            }
288            cout << endl;
289        }
290    }
291
292    void printPre() {
293        for (int i = 0; i < n; i++) {
294            cout << "v" << i << ": ";
295            for (int j = 0; j < g[i].size(); j++) {
296                cout << pre[g[i][j]] << " ";
297            }
298            cout << endl;
299        }
300    }
301
302    void printE() {
303        for (int i = 0; i < n; i++) {
304            cout << "v" << i << ": ";
305            for (int j = 0; j < g[i].size(); j++) {
306                cout << e[g[i][j]].c << " ";
307            }
308            cout << endl;
309        }
310    }
311
312    void printF() {
313        for (int i = 0; i < n; i++) {
314            cout << "v" << i << ": ";
315            for (int j = 0; j < g[i].size(); j++) {
316                cout << e[g[i][j]].f << " ";
317            }
318            cout << endl;
319        }
320    }
321
322    void printG() {
323        for (int i = 0; i < n; i++) {
324            cout << "v" << i << ": ";
325            for (int j = 0; j < g[i].size(); j++) {
326                cout << g[i][j] << " ";
327            }
328            cout << endl;
329        }
330    }
331
332    void printH() {
333        for (int i = 0; i < n; i++) {
334            cout << "v" << i << ": ";
335            for (int j = 0; j < g[i].size(); j++) {
336                cout << h[g[i][j]] << " ";
337            }
338            cout << endl;
339        }
340    }
341
342    void printDis() {
343        for (int i = 0; i < n; i++) {
344            cout << "v" << i << ": ";
345            for (int j = 0; j < g[i].size(); j++) {
346                cout << dis[g[i][j]] << " ";
347            }
348            cout << endl;
349        }
350    }
351
352    void printPre() {
353        for (int i = 0; i < n; i++) {
354            cout << "v" << i << ": ";
355            for (int j = 0; j < g[i].size(); j++) {
356                cout << pre[g[i][j]] << " ";
357            }
358            cout << endl;
359        }
360    }
361
362    void printE() {
363        for (int i = 0; i < n; i++) {
364            cout << "v" << i << ": ";
365            for (int j = 0; j < g[i].size(); j++) {
366                cout << e[g[i][j]].c << " ";
367            }
368            cout << endl;
369        }
370    }
371
372    void printF() {
373        for (int i = 0; i < n; i++) {
374            cout << "v" << i << ": ";
375            for (int j = 0; j < g[i].size(); j++) {
376                cout << e[g[i][j]].f << " ";
377            }
378            cout << endl;
379        }
380    }
381
382    void printG() {
383        for (int i = 0; i < n; i++) {
384            cout << "v" << i << ": ";
385            for (int j = 0; j < g[i].size(); j++) {
386                cout << g[i][j] << " ";
387            }
388            cout << endl;
389        }
390    }
391
392    void printH() {
393        for (int i = 0; i < n; i++) {
394            cout << "v" << i << ": ";
395            for (int j = 0; j < g[i].size(); j++) {
396                cout << h[g[i][j]] << " ";
397            }
398            cout << endl;
399        }
400    }
401
402    void printDis() {
403        for (int i = 0; i < n; i++) {
404            cout << "v" << i << ": ";
405            for (int j = 0; j < g[i].size(); j++) {
406                cout << dis[g[i][j]] << " ";
407            }
408            cout << endl;
409        }
410    }
411
412    void printPre() {
413        for (int i = 0; i < n; i++) {
414            cout << "v" << i << ": ";
415            for (int j = 0; j < g[i].size(); j++) {
416                cout << pre[g[i][j]] << " ";
417            }
418            cout << endl;
419        }
420    }
421
422    void printE() {
423        for (int i = 0; i < n; i++) {
424            cout << "v" << i << ": ";
425            for (int j = 0; j < g[i].size(); j++) {
426                cout << e[g[i][j]].c << " ";
427            }
428            cout << endl;
429        }
430    }
431
432    void printF() {
433        for (int i = 0; i < n; i++) {
434            cout << "v" << i << ": ";
435            for (int j = 0; j < g[i].size(); j++) {
436                cout << e[g[i][j]].f << " ";
437            }
438            cout << endl;
439        }
440    }
441
442    void printG() {
443        for (int i = 0; i < n; i++) {
444            cout << "v" << i << ": ";
445            for (int j = 0; j < g[i].size(); j++) {
446                cout << g[i][j] << " ";
447            }
448            cout << endl;
449        }
450    }
451
452    void printH() {
453        for (int i = 0; i < n; i++) {
454            cout << "v" << i << ": ";
455            for (int j = 0; j < g[i].size(); j++) {
456                cout << h[g[i][j]] << " ";
457            }
458            cout << endl;
459        }
460    }
461
462    void printDis() {
463        for (int i = 0; i < n; i++) {
464            cout << "v" << i << ": ";
465            for (int j = 0; j < g[i].size(); j++) {
466                cout << dis[g[i][j]] << " ";
467            }
468            cout << endl;
469        }
470    }
471
472    void printPre() {
473        for (int i = 0; i < n; i++) {
474            cout << "v" << i << ": ";
475            for (int j = 0; j < g[i].size(); j++) {
476                cout << pre[g[i][j]] << " ";
477            }
478            cout << endl;
479        }
480    }
481
482    void printE() {
483        for (int i = 0; i < n; i++) {
484            cout << "v" << i << ": ";
485            for (int j = 0; j < g[i].size(); j++) {
486                cout << e[g[i][j]].c << " ";
487            }
488            cout << endl;
489        }
490    }
491
492    void printF() {
493        for (int i = 0; i < n; i++) {
494            cout << "v" << i << ": ";
495            for (int j = 0; j < g[i].size(); j++) {
496                cout << e[g[i][j]].f << " ";
497            }
498            cout << endl;
499        }
500    }
501
502    void printG() {
503        for (int i = 0; i < n; i++) {
504            cout << "v" << i << ": ";
505            for (int j = 0; j < g[i].size(); j++) {
506                cout << g[i][j] << " ";
507            }
508            cout << endl;
509        }
510    }
511
512    void printH() {
513        for (int i = 0; i < n; i++) {
514            cout << "v" << i << ": ";
515            for (int j = 0; j < g[i].size(); j++) {
516                cout << h[g[i][j]] << " ";
517            }
518            cout << endl;
519        }
520    }
521
522    void printDis() {
523        for (int i = 0; i < n; i++) {
524            cout << "v" << i << ": ";
525            for (int j = 0; j < g[i].size(); j++) {
526                cout << dis[g[i][j]] << " ";
527            }
528            cout << endl;
529        }
530    }
531
532    void printPre() {
533        for (int i = 0; i < n; i++) {
534            cout << "v" << i << ": ";
535            for (int j = 0; j < g[i].size(); j++) {
536                cout << pre[g[i][j]] << " ";
537            }
538            cout << endl;
539        }
540    }
541
542    void printE() {
543        for (int i = 0; i < n; i++) {
544            cout << "v" << i << ": ";
545            for (int j = 0; j < g[i].size(); j++) {
546                cout << e[g[i][j]].c << " ";
547            }
548            cout << endl;
549        }
550    }
551
552    void printF() {
553        for (int i = 0; i < n; i++) {
554            cout << "v" << i << ": ";
555            for (int j = 0; j < g[i].size(); j++) {
556                cout << e[g[i][j]].f << " ";
557            }
558            cout << endl;
559        }
560    }
561
562    void printG() {
563        for (int i = 0; i < n; i++) {
564            cout << "v" << i << ": ";
565            for (int j = 0; j < g[i].size(); j++) {
566                cout << g[i][j] << " ";
567            }
568            cout << endl;
569        }
570    }
571
572    void printH() {
573        for (int i = 0; i < n; i++) {
574            cout << "v" << i << ": ";
575            for (int j = 0; j < g[i].size(); j++) {
576                cout << h[g[i][j]] << " ";
577            }
578            cout << endl;
579        }
580    }
581
582    void printDis() {
583        for (int i = 0; i < n; i++) {
584            cout << "v" << i << ": ";
585            for (int j = 0; j < g[i].size(); j++) {
586                cout << dis[g[i][j]] << " ";
587            }
588            cout << endl;
589        }
590    }
591
592    void printPre() {
593        for (int i = 0; i < n; i++) {
594            cout << "v" << i << ": ";
595            for (int j = 0; j < g[i].size(); j++) {
596                cout << pre[g[i][j]] << " ";
597            }
598            cout << endl;
599        }
600    }
601
602    void printE() {
603        for (int i = 0; i < n; i++) {
604            cout << "v" << i << ": ";
605            for (int j = 0; j < g[i].size(); j++) {
606                cout << e[g[i][j]].c << " ";
607            }
608            cout << endl;
609        }
610    }
611
612    void printF() {
613        for (int i = 0; i < n; i++) {
614            cout << "v" << i << ": ";
615            for (int j = 0; j < g[i].size(); j++) {
616                cout << e[g[i][j]].f << " ";
617            }
618            cout << endl;
619        }
620    }
621
622    void printG() {
623        for (int i = 0; i < n; i++) {
624            cout << "v" << i << ": ";
625            for (int j = 0; j < g[i].size(); j++) {
626                cout << g[i][j] << " ";
627            }
628            cout << endl;
629        }
630    }
631
632    void printH() {
633        for (int i = 0; i < n; i++) {
634            cout << "v" << i << ": ";
635            for (int j = 0; j < g[i].size(); j++) {
636                cout << h[g[i][j]] << " ";
637            }
638            cout << endl;
639        }
640    }
641
642    void printDis() {
643        for (int i = 0; i < n; i++) {
644            cout << "v" << i << ": ";
645            for (int j = 0; j < g[i].size(); j++) {
646                cout << dis[g[i][j]] << " ";
647            }
648            cout << endl;
649        }
650    }
651
652    void printPre() {
653        for (int i = 0; i < n; i++) {
654            cout << "v" << i << ": ";
655            for (int j = 0; j < g[i].size(); j++) {
656                cout << pre[g[i][j]] << " ";
657            }
658            cout << endl;
659        }
660    }
661
662    void printE() {
663        for (int i = 0; i < n; i++) {
664            cout << "v" << i << ": ";
665            for (int j = 0; j < g[i].size(); j++) {
666                cout << e[g[i][j]].c << " ";
667            }
668            cout << endl;
669        }
670    }
671
672    void printF() {
673        for (int i = 0; i < n; i++) {
674            cout << "v" << i << ": ";
675            for (int j = 0; j < g[i].size(); j++) {
676                cout << e[g[i][j]].f << " ";
677            }
678            cout << endl;
679        }
680    }
681
682    void printG() {
683        for (int i = 0; i < n; i++) {
684            cout << "v" << i << ": ";
685            for (int j = 0; j < g[i].size(); j++) {
686                cout << g[i][j] << " ";
687            }
688            cout << endl;
689        }
690    }
691
692    void printH() {
693        for (int i = 0; i < n; i++) {
694            cout << "v" << i << ": ";
695            for (int j = 0; j < g[i].size(); j++) {
696                cout << h[g[i][j]] << " ";
697            }
698            cout << endl;
699        }
700    }
701
702    void printDis() {
703        for (int i = 0; i < n; i++) {
704            cout << "v" << i << ": ";
705            for (int j = 0; j < g[i].size(); j++) {
706                cout << dis[g[i][j]] << " ";
707            }
708            cout << endl;
709        }
710    }
711
712    void printPre() {
713        for (int i = 0; i < n; i++) {
714            cout << "v" << i << ": ";
715            for (int j = 0; j < g[i].size(); j++) {
716                cout << pre[g[i][j]] << " ";
717            }
718            cout << endl;
719        }
720    }
721
722    void printE() {
723        for (int i = 0; i < n; i++) {
724            cout << "v" << i << ": ";
725            for (int j = 0; j < g[i].size(); j++) {
726                cout << e[g[i][j]].c << " ";
727            }
728            cout << endl;
729        }
730    }
731
732    void printF() {
733        for (int i = 0; i < n; i++) {
734            cout << "v" << i << ": ";
735            for (int j = 0; j < g[i].size(); j++) {
736                cout << e[g[i][j]].f << " ";
737            }
738            cout << endl;
739        }
740    }
741
742    void printG() {
743        for (int i = 0; i < n; i++) {
744            cout << "v" << i << ": ";
745            for (int j = 0; j < g[i].size(); j++) {
746                cout << g[i][j] << " ";
747            }
748            cout << endl;
749        }
750    }
751
752    void printH() {
753        for (int i = 0; i < n; i++) {
754            cout << "v" << i << ": ";
755            for (int j = 0; j < g[i].size(); j++) {
756                cout << h[g[i][j]] << " ";
757            }
758            cout << endl;
759        }
760    }
761
762    void printDis() {
763        for (int i = 0; i < n; i++) {
764            cout << "v" << i << ": ";
765            for (int j = 0; j < g[i].size(); j++) {
766                cout << dis[g[i][j]] << " ";
767            }
768            cout << endl;
769        }
770    }
771
772    void printPre() {
773        for (int i = 0; i < n; i++) {
774            cout << "v" << i << ": ";
775            for (int j = 0; j < g[i].size(); j++) {
776                cout << pre[g[i][j]] << " ";
777            }
778            cout << endl;
779        }
780    }
781
782    void printE() {
783        for (int i = 0; i < n; i++) {
784            cout << "v" << i << ": ";
785            for (int j = 0; j < g[i].size(); j++) {
786                cout << e[g[i][j]].c << " ";
787            }
788            cout << endl;
789        }
790    }
791
792    void printF() {
793        for (int i = 0; i < n; i++) {
794            cout << "v" << i << ": ";
795            for (int j = 0; j < g[i].size(); j++) {
796                cout << e[g[i][j]].f << " ";
797            }
798           
```

```

17     priority_queue<pair<i64, int>, vector<pair<i64, int>>, greater<pair<i64,
int>>> que;
18     dis[s] = 0;
19     que.emplace(0, s);
20     while (!que.empty()) {
21         i64 d = que.top().first;
22         int u = que.top().second;
23         que.pop();
24         if (dis[u] < d) continue;
25         for (int i : g[u]) {
26             int v = e[i].v;
27             int c = e[i].c;
28             int f = e[i].f;
29             if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
30                 dis[v] = d + h[u] - h[v] + f;
31                 pre[v] = i;
32                 que.emplace(dis[v], v);
33             }
34         }
35     }
36     return dis[t] != numeric_limits<i64>::max();
37 }
38 MCFGraph(int n) : n(n), g(n) {}
39 void addEdge(int u, int v, int c, int f) {
40     // if (f < 0) {
41     g[u].push_back(e.size());
42     e.emplace_back(v, 0, f);
43     g[v].push_back(e.size());
44     e.emplace_back(u, c, -f);
45     // } else {
46     //     g[u].push_back(e.size());
47     //     e.emplace_back(v, c, f);
48     //     g[v].push_back(e.size());
49     //     e.emplace_back(u, 0, -f);
50     // }
51 }
52 pair<int, i64> flow(int s, int t) {
53     int flow = 0;
54     i64 cost = 0;
55     h.assign(n, 0);
56     while (dijkstra(s, t)) {
57         for (int i = 0; i < n; ++i) h[i] += dis[i];
58         int aug = numeric_limits<int>::max();
59         for (int i = t; i != s; i = e[pre[i] ^ 1].v) aug = min(aug,
e[pre[i]].c);
60         for (int i = t; i != s; i = e[pre[i] ^ 1].v) {
61             e[pre[i]].c -= aug;
62             e[pre[i] ^ 1].c += aug;
63         }
64         flow += aug;
65         cost += i64(aug) * h[t];
66     }
67     return make_pair(flow, cost);
68 }
69 };
70
71

```

2.7.2 费用流 (MinCostFlow 新版)

```

1 template<class T>
2 struct MinCostFlow {
3     struct _Edge {
4         int to;
5         T cap;
6         T cost;
7         _Edge(int to_, T cap_, T cost_) : to(to_), cap(cap_), cost(cost_) {}
8     };
9     int n;
10    vector<_Edge> e;
11    vector<vector<int>> g;
12    vector<T> h, dis;
13    vector<int> pre;
14    bool dijkstra(int s, int t) {
15        dis.assign(n, numeric_limits<T>::max());
16        pre.assign(n, -1);
17        priority_queue<pair<T, int>, vector<pair<T, int>>, greater<pair<T, int>>>
que;
18        dis[s] = 0;
19        que.emplace(0, s);
20        while (!que.empty()) {
21            T d = que.top().first;
22            int u = que.top().second;
23            que.pop();
24            if (dis[u] != d) {
25                continue;
26            }
27            for (int i : g[u]) {
28                int v = e[i].to;
29                T cap = e[i].cap;
30                T cost = e[i].cost;
31                if (cap > 0 && dis[v] > d + h[u] - h[v] + cost) {
32                    dis[v] = d + h[u] - h[v] + cost;
33                    pre[v] = i;
34                    que.emplace(dis[v], v);
35                }
36            }
37        }
38        return dis[t] != numeric_limits<T>::max();
39    }
40    MinCostFlow() {}
41    MinCostFlow(int n_) {
42        init(n_);
43    }
44    void init(int n_) {
45        n = n_;
46        e.clear();
47        g.assign(n, {});
48    }
49    void addEdge(int u, int v, T cap, T cost) {
50        g[u].push_back(e.size());
51        e.emplace_back(v, cap, cost);
52        g[v].push_back(e.size());
53        e.emplace_back(u, 0, -cost);
54    }
55    pair<T, T> flow(int s, int t) {
56        T flow = 0;
57        T cost = 0;
58        h.assign(n, 0);
59        while (dijkstra(s, t)) {
60            for (int i = 0; i < n; ++i) {

```

```

61         h[i] += dis[i];
62     }
63     T aug = numeric_limits<int>::max();
64     for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
65         aug = min(aug, e[pre[i]].cap);
66     }
67     for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
68         e[pre[i]].cap -= aug;
69         e[pre[i] ^ 1].cap += aug;
70     }
71     flow += aug;
72     cost += aug * h[t];
73 }
74 return make_pair(flow, cost);
75 }
76 struct Edge {
77     int from;
78     int to;
79     T cap;
80     T cost;
81     T flow;
82 };
83 vector<Edge> edges() {
84     vector<Edge> a;
85     for (int i = 0; i < e.size(); i += 2) {
86         Edge x;
87         x.from = e[i + 1].to;
88         x.to = e[i].to;
89         x.cap = e[i].cap + e[i + 1].cap;
90         x.cost = e[i].cost;
91         x.flow = e[i + 1].cap;
92         a.push_back(x);
93     }
94     return a;
95 }
96 };

```

2.8 树链剖分 (HLD)

```

1 struct HLD {
2     int n;
3     vector<int> siz, top, dep, parent, in, out, seq;
4     vector<vector<int>> adj;
5     int cur;
6
7     HLD() {}
8     HLD(int n) {
9         init(n);
10    }
11    void init(int n) {
12        this->n = n;
13        siz.resize(n);
14        top.resize(n);
15        dep.resize(n);
16        parent.resize(n);
17        in.resize(n);
18        out.resize(n);
19        seq.resize(n);
20        cur = 0;
21        adj.assign(n, {});
22    }
23    void addEdge(int u, int v) {

```

```

24     adj[u].push_back(v);
25     adj[v].push_back(u);
26 }
27 void work(int root = 0) {
28     top[root] = root;
29     dep[root] = 0;
30     parent[root] = -1;
31     dfs1(root);
32     dfs2(root);
33 }
34 void dfs1(int u) {
35     if (parent[u] != -1) {
36         adj[u].erase(find(adj[u].begin(), adj[u].end(), parent[u]));
37     }
38
39     siz[u] = 1;
40     for (auto &v : adj[u]) {
41         parent[v] = u;
42         dep[v] = dep[u] + 1;
43         dfs1(v);
44         siz[u] += siz[v];
45         if (siz[v] > siz[adj[u][0]]) {
46             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 isAncester(int u, int v) {
89         return in[u] <= in[v] && in[v] < out[u];
90     }
91
92     int rootedParent(int u, int v) {
93         swap(u, v);
94         if (u == v) {
95             return u;
96         }
97         if (!isAncester(u, v)) {
98             return parent[u];
99         }
100        auto it = 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 (!isAncester(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  /** 快速幂 - 普通版 */
2  int power(int a, i64 b, int p) {
3      int res = 1;
4      for (; b; b /= 2, a = 1LL * a * a % p) {
5          if (b % 2) {
6              res = 1LL * res * a % p;
7          }
8      }
9      return res;
10 }
11 /**
12 i64 mul(i64 a, i64 b, i64 p) {
13     i64 c = a * b - i64(1.0L * a * b / p) * p;
14     c %= p;
15     if (c < 0) {
16         c += p;
17     }
18     return c;
19 }
20 i64 power(i64 a, i64 b, i64 p) {
21     i64 res = 1;
22     for (; b; b /= 2, a = mul(a, a, p)) {
23         if (b % 2) {
24             res = mul(res, a, p);
25         }
26     }
27     return res;
28 }
```

3.2 基姆拉尔森公式

```

1 const int d[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
2
3 bool isLeap(int y) {
4     return y % 400 == 0 || (y % 4 == 0 && y % 100 != 0);
5 }
6
7 int daysInMonth(int y, int m) {
8     return d[m - 1] + (isLeap(y) && m == 2);
9 }
10
11 int getDay(int y, int m, int d) {
12     int ans = 0;
13     for (int i = 1970; i < y; i++) {
14         ans += 365 + isLeap(i);
15     }
16     for (int i = 1; i < m; i++) {
17         ans += daysInMonth(y, i);
18     }
19     ans += d;
20     return (ans + 2) % 7 + 1;
21 }
```

3.3 欧拉筛

```

1  /** 欧拉筛 */
2  vector<int> minp, primes;
3
4  void sieve(int n) {
5      minp.assign(n + 1, 0);
6      primes.clear();
7
8      for (int i = 2; i <= n; i++) {
9          if (minp[i] == 0) {
10              minp[i] = i;
11              primes.push_back(i);
12          }
13
14          for (auto p : primes) {
15              if (i * p > n) {
16                  break;
17              }
18              minp[i * p] = p;
19              if (p == minp[i]) {
20                  break;
21              }
22          }
23      }
24  }
25
26  bool isprime(int n) {
27      return minp[n] == n;
28  }
29
30  /** 欧拉筛 */
31  void sieve(int n) {
32      minp.assign(n + 1, 0);
33      phi.assign(n + 1, 0);
34      primes.clear();
35
36      for (int i = 2; i <= n; i++) {
37          if (minp[i] == 0) {
38              minp[i] = i;
39              phi[i] = i - 1;
40              primes.push_back(i);
41          }
42
43          for (auto p : primes) {
44              if (i * p > n) {
45                  break;
46              }
47              minp[i * p] = p;
48              if (p == minp[i]) {
49                  phi[i * p] = phi[i] * p;
50                  break;
51              }
52              phi[i * p] = phi[i] * (p - 1);
53          }
54      }
55      for (int i = 2; i <= n; i++) {
56          phi[i] += phi[i - 1];
57      }
58  }

```

3.4 莫比乌斯函数筛（莫比乌斯反演）

```

1  unordered_map<int, Z> fMu;
2
3  vector<int> minp, primes, phi, mu;
4  vector<i64> sphi;
5
6  void sieve(int n) {
7      minp.assign(n + 1, 0);
8      phi.assign(n + 1, 0);
9      sphi.assign(n + 1, 0);
10     mu.assign(n + 1, 0);
11     primes.clear();
12     phi[1] = 1;
13     mu[1] = 1;
14
15     for (int i = 2; i <= n; i++) {
16         if (minp[i] == 0) {
17             minp[i] = i;
18             phi[i] = i - 1;
19             mu[i] = -1;
20             primes.push_back(i);
21         }
22
23         for (auto p : primes) {
24             if (i * p > n) {
25                 break;
26             }
27             minp[i * p] = p;
28             if (p == minp[i]) {
29                 phi[i * p] = phi[i] * p;
30                 break;
31             }
32             phi[i * p] = phi[i] * (p - 1);
33             mu[i * p] = -mu[i];
34         }
35     }
36
37     for (int i = 1; i <= n; i++) {
38         sphi[i] = sphi[i - 1] + phi[i];
39         mu[i] += mu[i - 1];
40     }
41 }
42
43 Z sumMu(int n) {
44     if (n <= N) {
45         return mu[n];
46     }
47     if (fMu.count(n)) {
48         return fMu[n];
49     }
50     if (n == 0) {
51         return 0;
52     }
53     Z ans = 1;
54     for (int l = 2, r; l <= n; l = r + 1) {
55         r = n / (n / l);
56         ans -= (r - l + 1) * sumMu(n / l);
57     }
58     return ans;
59 }
```

3.5 扩展欧几里得 (exgcd)

```

1  /** 扩展欧几里得 ( exgcd ) */
2  i64 exgcd(i64 a, i64 b, i64 &x, i64 &y) {
3      if (b == 0) {
4          x = 1;
5          y = 0;
6          return a;
7      }
8      i64 g = exgcd(b, a % b, y, x);
9      y -= a / b * x;
10     return g;
11 }
12 pair<i64, i64> sol(i64 a, i64 b, i64 m) { // ax + b = 0 (mod m)
13     assert(m > 0);
14     b *= -1;
15     i64 x, y;
16     i64 g = exgcd(a, m, x, y);
17     if (g < 0) {
18         g *= -1;
19         x *= -1;
20         y *= -1;
21     }
22     if (b % g != 0) {
23         return {-1, -1};
24     }
25     x = x * (b / g) % (m / g);
26     if (x < 0) {
27         x += m / g;
28     }
29     return {x, m / g};
30 }
31
32 /** 扩展欧几里得 ( exgcd ) */
33 array<i64, 3> exgcd(i64 a, i64 b) {
34     if (!b) {
35         return {a, 1, 0};
36     }
37     auto [g, x, y] = exgcd(b, a % b);
38     return {g, y, x - a / b * y};
39 }
```

3.6 欧拉函数

3.6.1 欧拉函数 (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 }
```

3.6.2 欧拉函数（求解全部数的欧拉函数）

```

1 constexpr int N = 1E7;
2 constexpr int P = 1000003;
3
4 bool isprime[N + 1];
5 int phi[N + 1];
6 vector<int> primes;
7
8 fill(isprime + 2, isprime + N + 1, true);
9 phi[1] = 1;
10 for (int i = 2; i <= N; i++) {
11     if (isprime[i]) {
12         primes.push_back(i);
13         phi[i] = i - 1;
14     }
15     for (auto p : primes) {
16         if (i * p > N) {
17             break;
18         }
19         isprime[i * p] = false;
20         if (i % p == 0) {
21             phi[i * p] = phi[i] * p;
22             break;
23         }
24         phi[i * p] = phi[i] * (p - 1);
25     }
26 }
```

3.7 组合数

3.7.1 组合数（小范围预处理，逆元+杨辉三角）

```

1 constexpr int P = 1000000007;
2 constexpr int L = 10000;
3
4 int fac[L + 1], invfac[L + 1];
5 int sumbinom[L + 1][7];
6
7 int binom(int n, int m) {
8     if (n < m || m < 0) {
9         return 0;
10    }
11    return 1LL * fac[n] * invfac[m] % P * invfac[n - m] % P;
12 }
13
14 int power(int a, int b) {
15     int res = 1;
16     for (; b; b /= 2, a = 1LL * a * a % P) {
17         if (b % 2) {
18             res = 1LL * res * a % P;
19         }
20     }
21     return res;
22 }
23
24 int main() {
25     fac[0] = 1;
26     for (int i = 1; i <= L; i++) {
27         fac[i] = 1LL * fac[i - 1] * i % P;
28     }
29     invfac[L] = power(fac[L], P - 2);
```

```

30     for (int i = L; i; i--) {
31         invfac[i - 1] = 1LL * invfac[i] * i % P;
32     }
33
34     sumbinom[0][0] = 1;
35     for (int i = 1; i <= L; i++) {
36         for (int j = 0; j < 7; j++) {
37             sumbinom[i][j] = (sumbinom[i - 1][j] + sumbinom[i - 1][(j + 6) % 7]) %
P;
38         }
39     }
40 }
```

3.7.2 组合数 (Comb, with. ModIntBase)

```

1 struct Comb {
2     int n;
3     vector<Z> _fac;
4     vector<Z> _invfac;
5     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        if (m <= n) return;
14        _fac.resize(m + 1);
15        _invfac.resize(m + 1);
16        _inv.resize(m + 1);
17
18        for (int i = n + 1; i <= m; i++) {
19            _fac[i] = _fac[i - 1] * i;
20        }
21        _invfac[m] = _fac[m].inv();
22        for (int i = m; i > n; i--) {
23            _invfac[i - 1] = _invfac[i] * i;
24            _inv[i] = _invfac[i] * _fac[i - 1];
25        }
26        n = m;
27    }
28
29    Z fac(int m) {
30        if (m > n) init(2 * m);
31        return _fac[m];
32    }
33    Z invfac(int m) {
34        if (m > n) init(2 * m);
35        return _invfac[m];
36    }
37    Z inv(int m) {
38        if (m > n) init(2 * m);
39        return _inv[m];
40    }
41    Z binom(int n, int m) {
42        if (n < m || m < 0) return 0;
43        return fac(n) * invfac(m) * invfac(n - m);
44    }
45 } comb;
```

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 vector<i64> factorize(i64 n) {
37     vector<i64> p;
38     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, abs(x - y), n);
65         if (lam % 127 == 0) {
66             d = gcd(v, n);
67             v = 1;
68         }
69         if (power == lam) {
70             x = y;
71             power *= 2;
72             lam = 0;
73             d = 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 sort(p.begin(), p.end());
87 return p;
88 }
```

3.9 平面几何

3.9.1 平面几何 (Point)

```

1 template<class T>
2 struct Point {
3     T x;
4     T y;
5     Point(const T &x_ = 0, const T &y_ = 0) : x(x_), y(y_) {}
6
7     template<class U>
8     operator Point<U>() {
9         return Point<U>(U(x), U(y));
10    }
11    Point &operator+=(const Point &p) & {
12        x += p.x;
13        y += p.y;
14        return *this;
15    }
16    Point &operator-=(const Point &p) & {
17        x -= p.x;
18        y -= p.y;
19        return *this;
20    }
21    Point &operator*=(const T &v) & {
22        x *= v;
23        y *= v;
24        return *this;
25    }
26    Point &operator/=(const T &v) & {
27        x /= v;
28        y /= v;
29        return *this;
30    }
}
```

```

31 Point operator-() const {
32     return Point(-x, -y);
33 }
34 friend Point operator+(Point a, const Point &b) {
35     return a += b;
36 }
37 friend Point operator-(Point a, const Point &b) {
38     return a -= b;
39 }
40 friend Point operator*(Point a, const T &b) {
41     return a *= b;
42 }
43 friend Point operator/(Point a, const T &b) {
44     return a /= b;
45 }
46 friend Point operator*(const T &a, Point b) {
47     return b *= a;
48 }
49 friend bool operator==(const Point &a, const Point &b) {
50     return a.x == b.x && a.y == b.y;
51 }
52 friend istream &operator>>(istream &is, Point &p) {
53     return is >> p.x >> p.y;
54 }
55 friend ostream &operator<<(ostream &os, const Point &p) {
56     return os << "(" << p.x << ", " << p.y << ")";
57 }
58 };
59
60 template<class T>
61 struct Line {
62     Point<T> a;
63     Point<T> b;
64     Line(const Point<T> &a_ = Point<T>(), const Point<T> &b_ = Point<T>()) : a(a_), b(b_) {}
65 };
66
67 template<class T>
68 T dot(const Point<T> &a, const Point<T> &b) {
69     return a.x * b.x + a.y * b.y;
70 }
71
72 template<class T>
73 T cross(const Point<T> &a, const Point<T> &b) {
74     return a.x * b.y - a.y * b.x;
75 }
76
77 template<class T>
78 T square(const Point<T> &p) {
79     return dot(p, p);
80 }
81
82 template<class T>
83 double length(const Point<T> &p) {
84     return sqrt(square(p));
85 }
86
87 template<class T>
88 double length(const Line<T> &l) {
89     return length(l.a - l.b);
90 }
91
92 template<class T>
93 Point<T> normalize(const Point<T> &p) {

```

```

94     return p / length(p);
95 }
96
97 template<class T>
98 bool parallel(const Line<T> &l1, const Line<T> &l2) {
99     return cross(l1.b - l1.a, l2.b - l2.a) == 0;
100 }
101
102 template<class T>
103 double distance(const Point<T> &a, const Point<T> &b) {
104     return length(a - b);
105 }
106
107 template<class T>
108 double distancePL(const Point<T> &p, const Line<T> &l) {
109     return abs(cross(l.a - l.b, l.a - p)) / length(l);
110 }
111
112 template<class T>
113 double distancePS(const Point<T> &p, const Line<T> &l) {
114     if (dot(p - l.a, l.b - l.a) < 0) {
115         return distance(p, l.a);
116     }
117     if (dot(p - l.b, l.a - l.b) < 0) {
118         return distance(p, l.b);
119     }
120     return distancePL(p, l);
121 }
122
123 template<class T>
124 Point<T> rotate(const Point<T> &a) {
125     return Point(-a.y, a.x);
126 }
127
128 template<class T>
129 int sgn(const Point<T> &a) {
130     return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
131 }
132
133 template<class T>
134 bool pointOnLineLeft(const Point<T> &p, const Line<T> &l) {
135     return cross(l.b - l.a, p - l.a) > 0;
136 }
137
138 template<class T>
139 Point<T> lineIntersection(const Line<T> &l1, const Line<T> &l2) {
140     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b -
141 l2.a, l1.a - l1.b));
142
143 template<class T>
144 bool pointOnSegment(const Point<T> &p, const Line<T> &l) {
145     return cross(p - l.a, l.b - l.a) == 0 && min(l.a.x, l.b.x) <= p.x && p.x <=
146     max(l.a.x, l.b.x)
147     && min(l.a.y, l.b.y) <= p.y && p.y <= max(l.a.y, l.b.y);
148 }
149
150 template<class T>
151 bool pointInPolygon(const Point<T> &a, const vector<Point<T>> &p) {
152     int n = p.size();
153     for (int i = 0; i < n; i++) {
154         if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
155             return true;
156         }

```

```

156 }
157
158     int t = 0;
159     for (int i = 0; i < n; i++) {
160         auto u = p[i];
161         auto v = p[(i + 1) % n];
162         if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
163             t ^= 1;
164         }
165         if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
166             t ^= 1;
167         }
168     }
169
170     return t == 1;
171 }
172
173 // 0 : not intersect
174 // 1 : strictly intersect
175 // 2 : overlap
176 // 3 : intersect at endpoint
177 template<class T>
178 tuple<int, Point<T>, Point<T>> segmentIntersection(const Line<T> &l1, const Line<T>
&l2) {
179     if (max(l1.a.x, l1.b.x) < min(l2.a.x, l2.b.x)) {
180         return {0, Point<T>(), Point<T>()};
181     }
182     if (min(l1.a.x, l1.b.x) > max(l2.a.x, l2.b.x)) {
183         return {0, Point<T>(), Point<T>()};
184     }
185     if (max(l1.a.y, l1.b.y) < min(l2.a.y, l2.b.y)) {
186         return {0, Point<T>(), Point<T>()};
187     }
188     if (min(l1.a.y, l1.b.y) > max(l2.a.y, l2.b.y)) {
189         return {0, Point<T>(), Point<T>()};
190     }
191     if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
192         if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
193             return {0, Point<T>(), Point<T>()};
194         } else {
195             auto maxx1 = max(l1.a.x, l1.b.x);
196             auto minx1 = min(l1.a.x, l1.b.x);
197             auto maxy1 = max(l1.a.y, l1.b.y);
198             auto miny1 = min(l1.a.y, l1.b.y);
199             auto maxx2 = max(l2.a.x, l2.b.x);
200             auto minx2 = min(l2.a.x, l2.b.x);
201             auto maxy2 = max(l2.a.y, l2.b.y);
202             auto miny2 = min(l2.a.y, l2.b.y);
203             Point<T> p1(max(minx1, minx2), max(miny1, miny2));
204             Point<T> p2(min(maxx1, maxx2), min(maxy1, maxy2));
205             if (!pointOnSegment(p1, l1)) {
206                 swap(p1.y, p2.y);
207             }
208             if (p1 == p2) {
209                 return {3, p1, p2};
210             } else {
211                 return {2, p1, p2};
212             }
213         }
214     }
215     auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
216     auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
217     auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
218     auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);

```

```

219
220     if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0) ||
221     (cp3 < 0 && cp4 < 0)) {
222         return {0, Point<T>(), Point<T>()};
223     }
224
225     Point p = lineIntersection(l1, l2);
226     if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
227         return {1, p, p};
228     } else {
229         return {3, p, p};
230     }
231 }
232
233 template<class T>
234 double distanceSS(const Line<T> &l1, const Line<T> &l2) {
235     if (get<0>(segmentIntersection(l1, l2)) != 0) {
236         return 0.0;
237     }
238     return min({distancePS(l1.a, l2), distancePS(l1.b, l2), distancePS(l2.a, l1),
239     distancePS(l2.b, l1)});
240 }
241
242 template<class T>
243 bool segmentInPolygon(const Line<T> &l, const vector<Point<T>> &p) {
244     int n = p.size();
245     if (!pointInPolygon(l.a, p)) {
246         return false;
247     }
248     if (!pointInPolygon(l.b, p)) {
249         return false;
250     }
251     for (int i = 0; i < n; i++) {
252         auto u = p[i];
253         auto v = p[(i + 1) % n];
254         auto w = p[(i + 2) % n];
255         auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
256
257         if (t == 1) {
258             return false;
259         }
260         if (t == 0) {
261             continue;
262         }
263         if (t == 2) {
264             if (pointOnSegment(v, l) && v != l.a && v != l.b) {
265                 if (cross(v - u, w - v) > 0) {
266                     return false;
267                 }
268             }
269         } else {
270             if (p1 != u && p1 != v) {
271                 if (pointOnLineLeft(l.a, Line(v, u))
272                     || pointOnLineLeft(l.b, Line(v, u))) {
273                     return false;
274                 }
275             } else if (p1 == v) {
276                 if (l.a == v) {
277                     if (pointOnLineLeft(u, l)) {
278                         if (pointOnLineLeft(w, l)
279                             && pointOnLineLeft(w, Line(u, v))) {
280                             return false;
281                         }
282                     }
283                 }
284             }
285         }
286     }
287 }
```

```

281             if (pointOnLineLeft(w, 1)
282                 || pointOnLineLeft(w, Line(u, v))) {
283                 return false;
284             }
285         }
286     } else if (l.b == v) {
287         if (pointOnLineLeft(u, Line(l.b, l.a))) {
288             if (pointOnLineLeft(w, Line(l.b, l.a))
289                 && pointOnLineLeft(w, Line(u, v))) {
290                 return false;
291             }
292         } else {
293             if (pointOnLineLeft(w, Line(l.b, l.a))
294                 || pointOnLineLeft(w, Line(u, v))) {
295                 return false;
296             }
297         }
298     } else {
299         if (pointOnLineLeft(u, l)) {
300             if (pointOnLineLeft(w, Line(l.b, l.a))
301                 || pointOnLineLeft(w, Line(u, v))) {
302                 return false;
303             }
304         } else {
305             if (pointOnLineLeft(w, 1)
306                 || pointOnLineLeft(w, Line(u, v))) {
307                 return false;
308             }
309         }
310     }
311 }
312 }
313 }
314 return true;
315 }
316
317 template<class T>
318 vector<Point<T>> hp(vector<Line<T>> lines) {
319     sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
320         auto d1 = l1.b - l1.a;
321         auto d2 = l2.b - l2.a;
322
323         if (sgn(d1) != sgn(d2)) {
324             return sgn(d1) == 1;
325         }
326
327         return cross(d1, d2) > 0;
328     });
329
330     deque<Line<T>> ls;
331     deque<Point<T>> ps;
332     for (auto l : lines) {
333         if (ls.empty()) {
334             ls.push_back(l);
335             continue;
336         }
337
338         while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
339             ps.pop_back();
340             ls.pop_back();
341         }
342
343         while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
344             ps.pop_front();

```

```

345         ls.pop_front();
346     }
347
348     if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
349         if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
350
351             if (!pointOnLineLeft(ls.back().a, l)) {
352                 assert(ls.size() == 1);
353                 ls[0] = l;
354             }
355             continue;
356         }
357         return {};
358     }
359
360     ps.push_back(lineIntersection(ls.back(), l));
361     ls.push_back(l);
362 }
363
364 while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
365     ps.pop_back();
366     ls.pop_back();
367 }
368 if (ls.size() <= 2) {
369     return {};
370 }
371 ps.push_back(lineIntersection(ls[0], ls.back()));
372
373 return vector(ps.begin(), ps.end());
374 }
375
376 using real = long double;
377 using P = Point<real>;
378
379 constexpr real eps = 0;

```

3.9.2 平面几何 (with. complex)

```

1 using Point = complex<long double>;
2
3 #define x real
4 #define y imag
5
6 long double dot(const Point &a, const Point &b) {
7     return (conj(a) * b).x();
8 }
9
10 long double cross(const Point &a, const Point &b) {
11     return (conj(a) * b).y();
12 }
13
14 long double length(const Point &a) {
15     return sqrt(dot(a, a));
16 }
17
18 long double dist(const Point &a, const Point &b) {
19     return length(a - b);
20 }
21
22 long double get(const Point &a, const Point &b, const Point &c, const Point &d) {
23     auto e = a + (b - a) * cross(c - a, d - a) / cross(b - a, d - c);
24     return dist(d, e);

```

25 }

3.10 立体几何 (Point)

```

1  using i64 = long long;
2  using real = double;
3
4  struct Point {
5      real x = 0;
6      real y = 0;
7      real z = 0;
8  };
9
10 Point operator+(const Point &a, const Point &b) {
11     return {a.x + b.x, a.y + b.y, a.z + b.z};
12 }
13
14 Point operator-(const Point &a, const Point &b) {
15     return {a.x - b.x, a.y - b.y, a.z - b.z};
16 }
17
18 Point operator*(const Point &a, real b) {
19     return {a.x * b, a.y * b, a.z * b};
20 }
21
22 Point operator/(const Point &a, real b) {
23     return {a.x / b, a.y / b, a.z / b};
24 }
25
26 real length(const Point &a) {
27     return hypot(a.x, a.y, a.z);
28 }
29
30 Point normalize(const Point &a) {
31     real l = length(a);
32     return {a.x / l, a.y / l, a.z / l};
33 }
34
35 real getAng(real a, real b, real c) {
36     return acos((a * a + b * b - c * c) / 2 / a / b);
37 }
38
39 ostream &operator<<(ostream &os, const Point &a) {
40     return os << "(" << a.x << ", " << a.y << ", " << a.z << ")";
41 }
42
43 real dot(const Point &a, const Point &b) {
44     return a.x * b.x + a.y * b.y + a.z * b.z;
45 }
46
47 Point cross(const Point &a, const Point &b) {
48     return {
49         a.y * b.z - a.z * b.y,
50         a.z * b.x - a.x * b.z,
51         a.x * b.y - a.y * b.x
52     };
53 }

```

3.11 静态凸包

3.11.1 静态凸包 (with. Point, 新版)

```

1 struct Point {
2     i64 x;
3     i64 y;
4     Point() : x{0}, y{0} {}
5     Point(i64 x_, i64 y_) : x{x_}, y{y_} {}
6 };
7
8 i64 dot(Point a, Point b) {
9     return a.x * b.x + a.y * b.y;
10 }
11
12 i64 cross(Point a, Point b) {
13     return a.x * b.y - a.y * b.x;
14 }
15
16 Point operator+(Point a, Point b) {
17     return Point(a.x + b.x, a.y + b.y);
18 }
19
20 Point operator-(Point a, Point b) {
21     return Point(a.x - b.x, a.y - b.y);
22 }
23
24 auto getHull(vector<Point> p) {
25     sort(p.begin(), p.end(),
26          [&](auto a, auto b) {
27              return a.x < b.x || (a.x == b.x && a.y < b.y);
28          });
29
30     vector<Point> hi, lo;
31     for (auto p : p) {
32         while (hi.size() > 1 && cross(hi.back() - hi[hi.size() - 2], p - hi.back())
33             >= 0) {
34             hi.pop_back();
35         }
36         while (!hi.empty() && hi.back().x == p.x) {
37             hi.pop_back();
38         }
39         hi.push_back(p);
40         while (lo.size() > 1 && cross(lo.back() - lo[lo.size() - 2], p - lo.back())
41             <= 0) {
42             lo.pop_back();
43         }
44         if (lo.empty() || lo.back().x < p.x) {
45             lo.push_back(p);
46         }
47     }
48     return make_pair(hi, lo);
49 }
50 const double inf = INFINITY;

```

3.11.2 静态凸包 (with. complex)

```

1 using Point = complex<i64>;
2
3 #define x real
4 #define y imag
5
6 auto dot(const Point &a, const Point &b) {
7     return (conj(a) * b).x();
8 }
9
10 auto cross(const Point &a, const Point &b) {
11     return (conj(a) * b).y();
12 }
13
14 auto rot(const Point &p) {
15     return Point(-p.y(), p.x());
16 }
17
18 auto complexHull(vector<Point> a) {
19     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     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
35         while (h.size() > 1 && cross(h.back() - h[h.size() - 2], p - h.back()) >= 0)
36         {
37             h.pop_back();
38         }
39
40         l.push_back(p);
41         h.push_back(p);
42     }
43
44     reverse(h.begin(), h.end());
45
46     h.insert(h.end(), l.begin() + 1, l.end() - 1);
47
48     return h;
49 }
50
51 int sgn(Point p) {
52     if (p.y() > 0 || (p.y() == 0 && p.x() < 0)) {
53         return 0;
54     } else {
55         return 1;
56     }
57 }
```

3.12 多项式

3.12.1 多项式 (Poly, 旧版)

```

1 constexpr int C = 1024;
2 constexpr int P = 998244353;
3 vector<int> rev, roots{0, 1};
4 int power(int a, int b) {
5     int res = 1;
6     for (; b; b >>= 1, a = 111 * a * a % P)
7         if (b & 1)
8             res = 111 * res * a % P;
9     return res;
10 }
11 void dft(vector<int> &a) {
12     int n = a.size();
13     if (int(rev.size()) != n) {
14         int k = __builtin_ctz(n) - 1;
15         rev.resize(n);
16         for (int i = 0; i < n; ++i)
17             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
18     }
19     for (int i = 0; i < n; ++i)
20         if (rev[i] < i)
21             swap(a[i], a[rev[i]]);
22     if (int(roots.size()) < n) {
23         int k = __builtin_ctz(roots.size());
24         roots.resize(n);
25         while ((1 << k) < n) {
26             int e = power(3, (P - 1) >> (k + 1));
27             for (int i = 1 << (k - 1); i < (1 << k); ++i) {
28                 roots[2 * i] = roots[i];
29                 roots[2 * i + 1] = 111 * roots[i] * e % P;
30             }
31             ++k;
32         }
33     }
34     for (int k = 1; k < n; k *= 2) {
35         for (int i = 0; i < n; i += 2 * k) {
36             for (int j = 0; j < k; ++j) {
37                 int u = a[i + j];
38                 int v = 111 * a[i + j + k] * roots[k + j] % P;
39                 int x = u + v;
40                 if (x >= P)
41                     x -= P;
42                 a[i + j] = x;
43                 x = u - v;
44                 if (x < 0)
45                     x += P;
46                 a[i + j + k] = x;
47             }
48         }
49     }
50 }
51 void idft(vector<int> &a) {
52     int n = a.size();
53     reverse(a.begin() + 1, a.end());
54     dft(a);
55     int inv = power(n, P - 2);
56     for (int i = 0; i < n; ++i)
57         a[i] = 111 * a[i] * inv % P;
58 }
59 struct Poly {

```

```

60    vector<int> a;
61    Poly() {}
62    Poly(int a0) {
63        if (a0)
64            a = {a0};
65    }
66    Poly(const vector<int> &a1) : a(a1) {
67        while (!a.empty() && !a.back())
68            a.pop_back();
69    }
70    int size() const {
71        return a.size();
72    }
73    int operator[](int idx) const {
74        if (idx < 0 || idx >= size())
75            return 0;
76        return a[idx];
77    }
78    Poly mulxk(int k) const {
79        auto b = a;
80        b.insert(b.begin(), k, 0);
81        return Poly(b);
82    }
83    Poly modxk(int k) const {
84        k = min(k, size());
85        return Poly(vector<int>(a.begin(), a.begin() + k));
86    }
87    Poly divxk(int k) const {
88        if (size() <= k)
89            return Poly();
90        return Poly(vector<int>(a.begin() + k, a.end()));
91    }
92    friend Poly operator+(const Poly a, const Poly &b) {
93        vector<int> res(max(a.size(), b.size()));
94        for (int i = 0; i < int(res.size()); ++i) {
95            res[i] = a[i] + b[i];
96            if (res[i] >= P)
97                res[i] -= P;
98        }
99        return Poly(res);
100    }
101    friend Poly operator-(const Poly a, const Poly &b) {
102        vector<int> res(max(a.size(), b.size()));
103        for (int i = 0; i < int(res.size()); ++i) {
104            res[i] = a[i] - b[i];
105            if (res[i] < 0)
106                res[i] += P;
107        }
108        return Poly(res);
109    }
110    friend Poly operator*(Poly a, Poly b) {
111        int sz = 1, tot = a.size() + b.size() - 1;
112        while (sz < tot)
113            sz *= 2;
114        a.a.resize(sz);
115        b.a.resize(sz);
116        dft(a.a);
117        dft(b.a);
118        for (int i = 0; i < sz; ++i)
119            a.a[i] = 111 * a[i] * b[i] % P;
120        idft(a.a);
121        return Poly(a.a);
122    }
123    Poly &operator+=(Poly b) {

```

```

124     return (*this) = (*this) + b;
125 }
126 Poly &operator-=(Poly b) {
127     return (*this) = (*this) - b;
128 }
129 Poly &operator*=(Poly b) {
130     return (*this) = (*this) * b;
131 }
132 Poly deriv() const {
133     if (a.empty())
134         return Poly();
135     vector<int> res(size() - 1);
136     for (int i = 0; i < size() - 1; ++i)
137         res[i] = 111 * (i + 1) * a[i + 1] % P;
138     return Poly(res);
139 }
140 Poly integr() const {
141     if (a.empty())
142         return Poly();
143     vector<int> res(size() + 1);
144     for (int i = 0; i < size(); ++i)
145         res[i + 1] = 111 * a[i] * power(i + 1, P - 2) % P;
146     return Poly(res);
147 }
148 Poly inv(int m) const {
149     Poly x(power(a[0], P - 2));
150     int k = 1;
151     while (k < m) {
152         k *= 2;
153         x = (x * (2 - modxk(k) * x)).modxk(k);
154     }
155     return x.modxk(m);
156 }
157 Poly log(int m) const {
158     return (deriv() * inv(m)).integr().modxk(m);
159 }
160 Poly exp(int m) const {
161     Poly x(1);
162     int k = 1;
163     while (k < m) {
164         k *= 2;
165         x = (x * (1 - x.log(k) + modxk(k))).modxk(k);
166     }
167     return x.modxk(m);
168 }
169 Poly sqrt(int m) const {
170     Poly x(1);
171     int k = 1;
172     while (k < m) {
173         k *= 2;
174         x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((P + 1) / 2);
175     }
176     return x.modxk(m);
177 }
178 Poly mult(Poly b) const {
179     if (b.size() == 0)
180         return Poly();
181     int n = b.size();
182     reverse(b.a.begin(), b.a.end());
183     return ((*this) * b).divxk(n - 1);
184 }
185 vector<int> eval(vector<int> x) const {
186     if (size() == 0)
187         return vector<int>(x.size(), 0);

```

```

188     const int n = max(int(x.size()), size());
189     vector<Poly> q(4 * n);
190     vector<int> ans(x.size());
191     x.resize(n);
192     function<void(int, int, int)> build = [&](int p, int l, int r) {
193         if (r - l == 1) {
194             q[p] = vector<int>{1, (P - x[1]) % P};
195         } else {
196             int m = (l + r) / 2;
197             build(2 * p, l, m);
198             build(2 * p + 1, m, r);
199             q[p] = q[2 * p] * q[2 * p + 1];
200         }
201     };
202     build(1, 0, n);
203     function<void(int, int, const Poly &)> work = [&](int p, int l, int r,
204     const Poly &num) {
205         if (r - l == 1) {
206             if (l < int(ans.size()))
207                 ans[l] = num[0];
208         } else {
209             int m = (l + r) / 2;
210             work(2 * p, l, m, num.mult(q[2 * p + 1]).modxk(m - 1));
211             work(2 * p + 1, m, r, num.mult(q[2 * p]).modxk(r - m));
212         }
213     };
214     work(1, 0, n, mult(q[1].inv(n)));
215     return ans;
216 }
217 using i64 = long long;
218 void dft(vector<vector<int>> &a) {
219     int n = a.size();
220     for (auto &v : a) {
221         dft(v);
222     }
223     for (int i = 0; i < int(a[0].size()); i++) {
224         vector<int> v(n);
225         for (int j = 0; j < n; j++) {
226             v[j] = a[j][i];
227         }
228         dft(v);
229         for (int j = 0; j < n; j++) {
230             a[j][i] = v[j];
231         }
232     }
233 }
234 void idft(vector<vector<int>> &a) {
235     int n = a.size();
236     for (auto &v : a) {
237         idft(v);
238     }
239     for (int i = 0; i < int(a[0].size()); i++) {
240         vector<int> v(n);
241         for (int j = 0; j < n; j++) {
242             v[j] = a[j][i];
243         }
244         idft(v);
245         for (int j = 0; j < n; j++) {
246             a[j][i] = v[j];
247         }
248     }
249 }
250 auto inv(const vector<vector<int>> &a) {

```

```

251 int m = 1;
252 vector g(1, vector{Poly(a[0]).inv(C).a});
253 while (m < C) {
254     vector a0(4 * m, vector<int>(4 * C));
255     for (int i = 0; i < 2 * m; i++) {
256         for (int j = 0; j < C; j++) {
257             a0[i][j] = a[i][j];
258         }
259     }
260     dft(a0);
261     g.resize(4 * m);
262     for (auto &v : g) {
263         v.resize(4 * C);
264     }
265     dft(g);
266     for (int i = 0; i < 4 * m; i++) {
267         for (int j = 0; j < 4 * C; j++) {
268             g[i][j] = i64(g[i][j]) * (2 + i64(P - a0[i][j]) * g[i][j] % P) % P;
269         }
270     }
271     idft(g);
272     m *= 2;
273     g.resize(m);
274     for (auto &v : g) {
275         v.resize(C);
276     }
277 }
278 return g;
279 }
```

3.12.2 多项式 (Poly, with. MInt & MLong)

```

1 vector<int> rev;
2 template<int P>
3 vector<MIInt<P>> roots{0, 1};
4
5 template<int P>
6 constexpr MIInt<P> findPrimitiveRoot() {
7     MIInt<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 MIInt<P> primitiveRoot = findPrimitiveRoot<P>();
20
21 template<>
22 constexpr MIInt<998244353> primitiveRoot<998244353> {31};
23
24 template<int P>
25 constexpr void dft(vector<MIInt<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         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(vector<MInt<P>> &a) {
67     int n = a.size();
68     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 vector<MInt<P>> {
78     using Value = MInt<P>;
79
80     Poly() : vector<Value>() {}
81     explicit constexpr Poly(int n) : vector<Value>(n) {}
82
83     explicit constexpr Poly(const vector<Value> &a) : vector<Value>(a) {}
84     constexpr Poly(const initializer_list<Value> &a) : vector<Value>(a) {}
85
86     template<class InputIt, class = _RequireInputIter<InputIt>>
87     explicit constexpr Poly(InputIt first, InputIt last) : vector<Value>(first,
88     last) {}
89
90     template<class F>
91     explicit constexpr Poly(int n, F f) : vector<Value>(n) {
92         for (int i = 0; i < n; i++) {
93             (*this)[i] = f(i);
94         }
95     }
96 }
```

```

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
108    constexpr Poly trunc(int k) const {
109        Poly f = *this;
110        f.resize(k);
111        return f;
112    }
113    constexpr friend Poly operator+(const Poly &a, const Poly &b) {
114        Poly res(max(a.size(), b.size()));
115        for (int i = 0; i < a.size(); i++) {
116            res[i] += a[i];
117        }
118        for (int i = 0; i < b.size(); i++) {
119            res[i] += b[i];
120        }
121        return res;
122    }
123    constexpr friend Poly operator-(const Poly &a, const Poly &b) {
124        Poly res(max(a.size(), b.size()));
125        for (int i = 0; i < a.size(); i++) {
126            res[i] += a[i];
127        }
128        for (int i = 0; i < b.size(); i++) {
129            res[i] -= b[i];
130        }
131        return res;
132    }
133    constexpr friend Poly operator-(const Poly &a) {
134        vector<Value> res(a.size());
135        for (int i = 0; i < int(res.size()); i++) {
136            res[i] = -a[i];
137        }
138        return Poly(res);
139    }
140    constexpr friend Poly operator*(Poly a, Poly b) {
141        if (a.size() == 0 || b.size() == 0) {
142            return Poly();
143        }
144        if (a.size() < b.size()) {
145            swap(a, b);
146        }
147        int n = 1, tot = a.size() + b.size() - 1;
148        while (n < tot) {
149            n *= 2;
150        }
151        if (((P - 1) & (n - 1)) != 0 || b.size() < 128) {
152            Poly c(a.size() + b.size() - 1);
153            for (int i = 0; i < a.size(); i++) {
154                for (int j = 0; j < b.size(); j++) {
155                    c[i + j] += a[i] * b[j];
156                }
157            }
158            return c;
159        }

```

```

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     reverse(b.begin(), b.end());
268     return ((*this) * b).shift(-(n - 1));
269 }
270 constexpr vector<Value> eval(vector<Value> x) const {
271     if (this->size() == 0) {
272         return vector<Value>(x.size(), 0);
273     }
274     const int n = max(x.size(), this->size());
275     vector<Poly> q(4 * n);
276     vector<Value> ans(x.size());
277     x.resize(n);
278     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, l, m);
284             build(2 * p + 1, m, r);
285             q[p] = q[2 * p] * q[2 * p + 1];
286         }
287     };
288 }
```

```

287     };
288     build(1, 0, n);
289     function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r,
290     const Poly &num) {
291         if (r - l == 1) {
292             if (l < ans.size()) {
293                 ans[l] = num[0];
294             }
295         } else {
296             int m = (l + r) / 2;
297             work(2 * p, l, m, num.mult(q[2 * p + 1]).trunc(m - 1));
298             work(2 * p + 1, m, r, num.mult(q[2 * p]).trunc(r - m));
299         }
300     };
301     work(1, 0, n, mult(q[1].inv(n)));
302     return ans;
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             MIInt<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 template<int P = 998244353>
349 MIInt<P> linearRecurrence(Poly<P> p, Poly<P> q, i64 n) {

```

```

350     int m = q.size() - 1;
351     while (n > 0) {
352         auto newq = q;
353         for (int i = 1; i <= m; i += 2) {
354             newq[i] *= -1;
355         }
356         auto newp = p * newq;
357         newq = q * newq;
358         for (int i = 0; i < m; i++) {
359             p[i] = newp[i * 2 + n % 2];
360         }
361         for (int i = 0; i <= m; i++) {
362             q[i] = newq[i * 2];
363         }
364         n /= 2;
365     }
366     return p[0] / q[0];
367 }
368
369 struct Comb {
370     int n;
371     vector<Z> _fac;
372     vector<Z> _invfac;
373     vector<Z> _inv;
374
375     Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
376     Comb(int n) : Comb() {
377         init(n);
378     }
379
380     void init(int m) {
381         m = min(m, Z::getMod() - 1);
382         if (m <= n) return;
383         _fac.resize(m + 1);
384         _invfac.resize(m + 1);
385         _inv.resize(m + 1);
386
387         for (int i = n + 1; i <= m; i++) {
388             _fac[i] = _fac[i - 1] * i;
389         }
390         _invfac[m] = _fac[m].inv();
391         for (int i = m; i > n; i--) {
392             _invfac[i - 1] = _invfac[i] * i;
393             _inv[i] = _invfac[i] * _fac[i - 1];
394         }
395         n = m;
396     }
397
398     Z fac(int m) {
399         if (m > n) init(2 * m);
400         return _fac[m];
401     }
402     Z invfac(int m) {
403         if (m > n) init(2 * m);
404         return _invfac[m];
405     }
406     Z inv(int m) {
407         if (m > n) init(2 * m);
408         return _inv[m];
409     }
410     Z binom(int n, int m) {
411         if (n < m || m < 0) return 0;
412         return fac(n) * invfac(m) * invfac(n - m);
413     }

```

```

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

3.12.3 多项式乘法

```

1 constexpr int P = 998244353;
2
3 int power(int a, int b) {
4     int res = 1;
5     for (; b; b /= 2, a = 1LL * a * a % P) {
6         if (b % 2) {
7             res = 1LL * res * a % P;
8         }
9     }
10    return res;
11 }
12
13 vector<int> rev, roots {0, 1};
14
15 void dft(vector<int> &a) {
16     int n = a.size();
17     if (int(rev.size()) != n) {
18         int k = __builtin_ctz(n) - 1;
19         rev.resize(n);
20         for (int i = 0; i < n; i++) {
21             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
22         }
23     }
24     for (int i = 0; i < n; i++) {
25         if (rev[i] < i) {
26             swap(a[i], a[rev[i]]);
```

```

27     }
28 }
29 if (roots.size() < n) {
30     int k = __builtin_ctz(roots.size());
31     roots.resize(n);
32     while ((1 << k) < n) {
33         int e = power(31, 1 << (__builtin_ctz(P - 1) - k - 1));
34         for (int i = 1 << (k - 1); i < (1 << k); i++) {
35             roots[2 * i] = roots[i];
36             roots[2 * i + 1] = 1LL * roots[i] * e % P;
37         }
38         k++;
39     }
40 }
41
42 for (int k = 1; k < n; k *= 2) {
43     for (int i = 0; i < n; i += 2 * k) {
44         for (int j = 0; j < k; j++) {
45             int u = a[i + j];
46             int v = 1LL * a[i + j + k] * roots[k + j] % P;
47             a[i + j] = (u + v) % P;
48             a[i + j + k] = (u - v) % P;
49         }
50     }
51 }
52 }
53
54 void idft(vector<int> &a) {
55     int n = a.size();
56     reverse(a.begin() + 1, a.end());
57     dft(a);
58     int inv = (1 - P) / n;
59     for (int i = 0; i < n; i++) {
60         a[i] = 1LL * a[i] * inv % P;
61     }
62 }
63
64 vector<int> mul(vector<int> a, vector<int> b) {
65     int n = 1, tot = a.size() + b.size() - 1;
66     while (n < tot) {
67         n *= 2;
68     }
69     if (tot < 128) {
70         vector<int> c(a.size() + b.size() - 1);
71         for (int i = 0; i < a.size(); i++) {
72             for (int j = 0; j < b.size(); j++) {
73                 c[i + j] = (c[i + j] + 1LL * a[i] * b[j]) % P;
74             }
75         }
76         return c;
77     }
78     a.resize(n);
79     b.resize(n);
80     dft(a);
81     dft(b);
82     for (int i = 0; i < n; i++) {
83         a[i] = 1LL * a[i] * b[i] % P;
84     }
85     idft(a);
86     a.resize(tot);
87     return a;
88 }

```

3.13 生成函数

3.13.1 生成函数 (q-int)

```

1 i64 power(i64 a, i64 b, i64 p) {
2     i64 res = 1;
3     for (; b; b /= 2, a = i128(a) * a % p) {
4         if (b % 2) {
5             res = i128(res) * a % p;
6         }
7     }
8     return res;
9 }
10
11 pair<int, int> qint(int q, int n, int p) {
12     q %= p;
13     for (int x = 2; x * x <= n; x++) {
14         if (n % x == 0) {
15             auto [v1, e1] = qint(q, x, p);
16             auto [v2, e2] = qint(power(q, x, p), n / x, p);
17             return {1LL * v1 * v2 % p, e1 + e2};
18         }
19     }
20     if (q == 1) {
21         if (n == p) {
22             return {0, 1};
23         }
24         return {n, 0};
25     }
26     // cerr << q << " " << n << " " << p << "\n";
27     i64 v = 1 - power(q, n, 1LL * p * p);
28     if (v < 0) {
29         v += 1LL * p * p;
30     }
31     assert(v != 0);
32     int inv = power(1 - q + p, p - 2, p);
33     if (v % p == 0) {
34         return {(v / p) * inv % p, 1};
35     } else {
36         return {v % p * inv % p, 0};
37     }
38 }
```

3.13.2 生成函数 (q-Binomial)

```

1 int power(int a, int 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 }
10
11 int qint(int n, int q, int p) {
12     return 1LL * (power(q, n, p) - 1) * power(q - 1, p - 2, p) % p;
13 }
14
15 int qBinomial(int n, int k, int q, int p) {
16     if (q == 0) {
17         return 1;
```

```

18 }
19 int r = 0;
20 int x = 1;
21 do {
22     x = 1LL * x * q % p;
23     r++;
24 } while (x != 1);
25
26 if (n / r > k / r + (n - k) / r) {
27     return 0;
28 }
29 int num = 1, den = 1;
30 for (int i = 1; i <= k % r; i++) {
31     num = 1LL * num * qint(n % r - i + 1, q, p) % p;
32     den = 1LL * den * qint(i, q, p) % p;
33 }
34 n /= r, k /= r;
35 while (n > 0 || k > 0) {
36     if (n % p < k % p) {
37         return 0;
38     }
39     for (int i = 1; i <= k % p; i++) {
40         num = 1LL * num * (n % p - i + 1) % p;
41         den = 1LL * den * i % p;
42     }
43     n /= p, k /= p;
44 }
45 int ans = 1LL * num * power(den, p - 2, p) % p;
46 return ans;
47 }

```

3.13.3 生成函数 (Binomial 任意模数二项式)

```

1 vector<pair<int, int>> factorize(int n) {
2     vector<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         swap(g, r);
39         x -= q * y;
40         swap(x, y);
41     }
42     return x < 0 ? x + m : x;
43 }
44 int solveModuloEquations(const vector<pair<int, int>> &e) {
45     int m = 1;
46     for (size_t i = 0; i < e.size(); i++) {
47         m *= e[i].first;
48     }
49     int res = 0;
50     for (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 vector<pair<int, int>> factors;
61     vector<int> pk;
62     vector<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, 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] *
75             prod[i][n % pk[i]] % pk[i];
76         }
77         return res;
78     }
79 public:
80     Binomial(int mod) : mod(mod), factors(factorize(mod)) {
81         pk.resize(factors.size());
82         prod.resize(factors.size());
83         for (size_t i = 0; i < factors.size(); i++) {
84             int p = factors[i].first;
85             int k = factors[i].second;
86             pk[i] = power(p, k);
87             prod[i].resize(min(N + 1, pk[i]));
88             prod[i][0] = 1;
89             for (int j = 1; j < prod[i].size(); j++) {
90                 if (j % p == 0) {
91                     prod[i][j] = prod[i][j - 1];
92                 } else {
93                     prod[i][j] = 1LL * prod[i][j - 1] * j % pk[i];
94                 }
95             }
96         }
97     }
98 }
```

```

94     }
95 }
96 }
97 int operator()(i64 n, i64 m) {
98     if (n < m || m < 0) {
99         return 0;
100    }
101   vector<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] = 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,
113             pk[i]) % pk[i] * power(p, e) % pk[i];
114           ans[i] = make_pair(pk[i], res);
115       }
116   }
117   return solveModuloEquations(ans);
118 }
119 };

```

3.14 自适应辛普森法 (Simpson)

```

1 const double Pi = acos(-1.0);
2 constexpr double EPS = 1e-9;
3 double v, r, d;
4 double f(double x) {
5     double s = sin(x);
6     return 1 / v / (sqrt(s * s + 3) - s);
7 }
8 double simpson(double l, double r) {
9     return (f(l) + 4 * f((l + r) / 2) + f(r)) * (r - l) / 6;
10 }
11 double integral(double l, double r, double eps, double st) {
12     double mid = (l + r) / 2;
13     double sl = simpson(l, mid);
14     double sr = simpson(mid, r);
15     if (abs(sl + sr - st) <= 15 * eps)
16         return sl + sr + (sl + sr - st) / 15;
17     return integral(l, mid, eps / 2, sl) + integral(mid, r, eps / 2, sr);
18 }
19 double integral(double l, double r) {
20     return integral(l, r, EPS, simpson(l, r));
21 }

```

3.15 矩阵 (Matrix)

```

1 using u64 = unsigned long long;
2 using Matrix = array<u64, 65>;
3
4 Matrix operator*(const Matrix &a, const Matrix &b) {
5     Matrix c{};
6     for (int i = 0; i <= 64; i++) {
7         for (int j = 0; j <= 64; j++) {
8             if (j == 64 ? i == 64 : (a[i] >> j & 1)) {

```

```

9         c[i] ^= b[j];
10    }
11}
12}
13return c;
14}
15
16u64 operator*(u64 a, const Matrix &b) {
17    u64 c = 0;
18    for (int i = 0; i <= 64; i++) {
19        if (i == 64 || (a >> i & 1)) {
20            c ^= b[i];
21        }
22    }
23    return c;
24}
25
26Matrix readMatrix() {
27    int m;
28    cin >> m;
29
30    Matrix f{};
31    for (int i = 0; i < m; i++) {
32        int s, o;
33        u64 A;
34        cin >> s >> o >> A;
35
36        if (o == 0) {
37            for (int j = 0; j < 64; j++) {
38                if (A >> ((j + s) % 64) & 1) {
39                    f[64] ^= 1ULL << ((j + s) % 64);
40                } else {
41                    f[j] ^= 1ULL << ((j + s) % 64);
42                }
43            }
44        } else {
45            for (int j = 0; j < 64; j++) {
46                if (A >> ((j + s) % 64) & 1) {
47                    f[j] ^= 1ULL << ((j + s) % 64);
48                }
49            }
50        }
51    }
52
53    u64 B;
54    cin >> B;
55    f[64] ^= B;
56
57    return f;
58}

```

3.16 高斯消元法 (gaussian elimination) 【久远】

```

1  /** 高斯消元法 ( gaussian elimination ) 【久远】 */
2  vector<int> operator*(const vector<int> &lhs, const vector<int> &rhs) {
3      vector<int> res(lhs.size() + rhs.size() - 1);
4      for (int i = 0; i < int(lhs.size()); ++i)
5          for (int j = 0; j < int(rhs.size()); ++j)
6              res[i + j] = (res[i + j] + 111 * lhs[i] * rhs[j]) % P;
7      return res;
8  }
9  vector<int> operator%(const vector<int> &lhs, const vector<int> &rhs) {

```

```

10     auto res = lhs;
11     int m = rhs.size() - 1;
12     int inv = power(rhs.back(), P - 2);
13     for (int i = res.size() - 1; i >= m; --i) {
14         int x = 111 * inv * res[i] % P;
15         for (int j = 0; j < m; ++j)
16             res[i - m + j] = (res[i - m + j] + 111 * (P - x) * rhs[j]) % P;
17     }
18     if (int(res.size()) > m)
19         res.resize(m);
20     return res;
21 }
22 vector<int> gauss(vector<vector<int>> a, vector<int> b) {
23     int n = a.size();
24     for (int i = 0; i < n; ++i) {
25         int r = i;
26         while (a[r][i] == 0)
27             ++r;
28         swap(a[i], a[r]);
29         swap(b[i], b[r]);
30         int inv = power(a[i][i], P - 2);
31         for (int j = i; j < n; ++j)
32             a[i][j] = 111 * a[i][j] * inv % P;
33         b[i] = 111 * b[i] * inv % P;
34         for (int j = 0; j < n; ++j) {
35             if (i == j)
36                 continue;
37             int x = a[j][i];
38             for (int k = i; k < n; ++k)
39                 a[j][k] = (a[j][k] + 111 * (P - x) * a[i][k]) % P;
40             b[j] = (b[j] + 111 * (P - x) * b[i]) % P;
41         }
42     }
43     return b;
44 }
45 /** 高斯消元法 (gaussian elimination) 【久远】 */
46 vector<double> gauss(vector<vector<double>> a, vector<double> b) {
47     int n = a.size();
48     for (int i = 0; i < n; ++i) {
49         double x = a[i][i];
50         for (int j = i; j < n; ++j) a[i][j] /= x;
51         b[i] /= x;
52         for (int j = 0; j < n; ++j) {
53             if (i == j) continue;
54             x = a[j][i];
55             for (int k = i; k < n; ++k) a[j][k] -= a[i][k] * x;
56             b[j] -= b[i] * x;
57         }
58     }
59     return b;
60 }

```

/END/

4 数据结构

4.1 树状数组 (Fenwick)

```

1 template <typename T>
2 struct Fenwick {
3     int n;
4     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 << __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

4.2.1 并查集 (DSU)

```

1 struct DSU {
2     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        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};
```

4.2.2 可撤销并查集 (DSU With Rollback)

```
1 struct DSU {
2     vector<int> siz;
3     vector<int> f;
4     vector<array<int, 2>> his;
5
6     DSU(int n) : siz(n + 1, 1), f(n + 1) {
7         iota(f.begin(), f.end(), 0);
8     }
9
10    int find(int x) {
11        while (f[x] != x) {
12            x = f[x];
13        }
14        return x;
15    }
16
17    bool merge(int x, int y) {
18        x = find(x);
19        y = find(y);
20        if (x == y) {
21            return false;
22        }
23        if (siz[x] < siz[y]) {
24            swap(x, y);
25        }
26        his.push_back({x, y});
27        siz[x] += siz[y];
28        f[y] = x;
29    }
30}
```

```

29         return true;
30     }
31
32     int time() {
33         return his.size();
34     }
35
36     void revert(int tm) {
37         while (his.size() > tm) {
38             auto [x, y] = his.back();
39             his.pop_back();
40             f[y] = y;
41             siz[x] -= siz[y];
42         }
43     }
44 };

```

4.3 线段树

4.3.1 线段树 (SegmentTree+Info 区间加+单点修改)

```

1 struct SegmentTree {
2     int n;
3     vector<int> tag;
4     vector<Info> info;
5     SegmentTree(int n_) : n(n_), tag(4 * n), info(4 * n) {}
6
7     void pull(int p) {
8         info[p] = info[2 * p] + info[2 * p + 1];
9     }
10
11    void add(int p, int v) {
12        tag[p] += v;
13        info[p].max += v;
14    }
15
16    void push(int p) {
17        add(2 * p, tag[p]);
18        add(2 * p + 1, tag[p]);
19        tag[p] = 0;
20    }
21
22    Info query(int p, int l, int r, int x, int y) {
23        if (l >= y || r <= x) {
24            return {};
25        }
26        if (l >= x && r <= y) {
27            return info[p];
28        }
29        int m = (l + r) / 2;
30        push(p);
31        return query(2 * p, l, m, x, y) + query(2 * p + 1, m, r, x, y);
32    }
33
34    Info query(int x, int y) {
35        return query(1, 0, n, x, y);
36    }
37
38    void rangeAdd(int p, int l, int r, int x, int y, int v) {
39        if (l >= y || r <= x) {
40            return;
41        }

```

```

42     if (l >= x && r <= y) {
43         return add(p, v);
44     }
45     int m = (l + r) / 2;
46     push(p);
47     rangeAdd(2 * p, l, m, x, y, v);
48     rangeAdd(2 * p + 1, m, r, x, y, v);
49     pull(p);
50 }
51
52 void rangeAdd(int x, int y, int v) {
53     rangeAdd(1, 0, n, x, y, v);
54 }
55
56 void modify(int p, int l, int r, int x, const Info &v) {
57     if (r - l == 1) {
58         info[p] = v;
59         return;
60     }
61     int m = (l + r) / 2;
62     push(p);
63     if (x < m) {
64         modify(2 * p, l, m, x, v);
65     } else {
66         modify(2 * p + 1, m, r, x, v);
67     }
68     pull(p);
69 }
70
71 void modify(int x, const Info &v) {
72     modify(1, 0, n, x, v);
73 }
74 };

```

4.3.2 线段树 (SegmentTree 区间乘+单点加)

```

1 struct SegmentTree {
2     int n;
3     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 = (l + r) / 2;
29     push(p);
30     return (query(2 * p, l, 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         return mul(p, v);
43     }
44     int m = (l + r) / 2;
45     push(p);
46     rangeMul(2 * p, l, 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 = (l + r) / 2;
61     push(p);
62     if (x < m) {
63         add(2 * p, l, 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 };

```

4.3.3 线段树 (SegmentTree+Info 初始赋值+单点修改+查找前驱后继)

```

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

```

```

15  template<class T>
16  void init(vector<T> init_) {
17      n = init_.size();
18      info.assign(4 << __lg(n), Info());
19      function<void(int, int, int)> build = [&](int p, int l, int r) {
20          if (r - l == 1) {
21              info[p] = init_[1];
22              return;
23          }
24          int m = (l + r) / 2;
25          build(2 * p, l, m);
26          build(2 * p + 1, m, r);
27          pull(p);
28      };
29      build(1, 0, n);
30  }
31  void pull(int p) {
32      info[p] = info[2 * p] + info[2 * p + 1];
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      if (x < m) {
41          modify(2 * p, l, m, x, v);
42      } else {
43          modify(2 * p + 1, m, r, x, v);
44      }
45      pull(p);
46  }
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      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  template<class F>
64  int findFirst(int p, int l, int r, int x, int y, F &&pred) {
65      if (l >= y || r <= x) {
66          return -1;
67      }
68      if (l >= x && r <= y && !pred(info[p])) {
69          return -1;
70      }
71      if (r - l == 1) {
72          return 1;
73      }
74      int m = (l + r) / 2;
75      int res = findFirst(2 * p, l, m, x, y, pred);
76      if (res == -1) {
77          res = findFirst(2 * p + 1, m, r, x, y, pred);
78      }

```

```

79     return res;
80 }
81 template<class F>
82     int findFirst(int l, int r, F &&pred) {
83     return findFirst(1, 0, n, l, r, pred);
84 }
85 template<class F>
86     int findLast(int p, int l, int r, int x, int y, F &&pred) {
87     if (l >= y || r <= x) {
88         return -1;
89     }
90     if (l >= x && r <= y && !pred(info[p])) {
91         return -1;
92     }
93     if (r - l == 1) {
94         return l;
95     }
96     int m = (l + r) / 2;
97     int res = findLast(2 * p + 1, m, r, x, y, pred);
98     if (res == -1) {
99         res = findLast(2 * p, l, m, x, y, pred);
100    }
101   return res;
102 }
103 template<class F>
104     int findLast(int l, int r, F &&pred) {
105     return findLast(1, 0, n, l, r, pred);
106 }
107 };

```

4.3.4 线段树 (SegmentTree+Info+Merge 初始赋值+单点修改+区间合并)

```

1 template<class Info, class Merge = plus<Info>> struct SegmentTree {
2     const int n;
3     const Merge merge;
4     vector<Info> info;
5     SegmentTree(int n) : n(n), merge(Merge()), info(4 << __lg(n)) {}
6     SegmentTree(vector<Info> init) : SegmentTree(init.size()) {
7         function<void(int, int, int)> build = [&](int p, int l, int r) {
8             if (r - l == 1) {
9                 info[p] = init[l];
10                return;
11            }
12            int m = (l + r) / 2;
13            build(2 * p, l, m);
14            build(2 * p + 1, m, r);
15            pull(p);
16        };
17        build(1, 0, n);
18    }
19    void pull(int p) {
20        info[p] = merge(info[2 * p], info[2 * p + 1]);
21    }
22    void modify(int p, int l, int r, int x, const Info &v) {
23        if (r - l == 1) {
24            info[p] = v;
25            return;
26        }
27        int m = (l + r) / 2;
28        if (x < m) {
29            modify(2 * p, l, m, x, v);

```

```

30     } else {
31         modify(2 * p + 1, m, r, x, v);
32     }
33     pull(p);
34 }
35 void modify(int p, const Info &v) {
36     modify(1, 0, n, p, v);
37 }
38 Info rangeQuery(int p, int l, int r, int x, int y) {
39     if (l >= y || r <= x) {
40         return Info();
41     }
42     if (l >= x && r <= y) {
43         return info[p];
44     }
45     int m = (l + r) / 2;
46     return merge(rangeQuery(2 * p, l, m, x, y), rangeQuery(2 * p + 1, m, r, x,
y));
47 }
48 Info rangeQuery(int l, int r) {
49     return rangeQuery(1, 0, n, l, r);
50 }
51 };

```

4.4 懒标记线段树 (LazySegmentTree)

```

1 template<class Info, class Tag> struct LazySegmentTree {
2     int n;
3     vector<Info> info;
4     vector<Tag> tag;
5     LazySegmentTree() : n(0) {}
6     LazySegmentTree(int n_, Info v_ = Info()) {
7         init(n_, v_);
8     }
9     template<class T>
10    LazySegmentTree(vector<T> init_) {
11        init(init_);
12    }
13    void init(int n_, Info v_ = Info()) {
14        init(vector(n_, v_));
15    }
16    template<class T>
17    void init(vector<T> init_) {
18        n = init_.size();
19        info.assign(4 << __lg(n), Info());
20        tag.assign(4 << __lg(n), Tag());
21        function<void(int, int, int)> build = [&](int p, int l, int r) {
22            if (r - l == 1) {
23                info[p] = init_[l];
24                return;
25            }
26            int m = (l + r) / 2;
27            build(2 * p, l, m);
28            build(2 * p + 1, m, r);
29            pull(p);
30        };
31        build(1, 0, n);
32    }
33    void pull(int p) {
34        info[p] = info[2 * p] + info[2 * p + 1];
35    }
36    void apply(int p, const Tag &v) {

```

```

37         info[p].apply(v);
38         tag[p].apply(v);
39     }
40     void push(int p) {
41         apply(2 * p, tag[p]);
42         apply(2 * p + 1, tag[p]);
43         tag[p] = Tag();
44     }
45     void modify(int p, int l, int r, int x, const Info &v) {
46         if (r - l == 1) {
47             info[p] = v;
48             return;
49         }
50         int m = (l + r) / 2;
51         push(p);
52         if (x < m) {
53             modify(2 * p, l, m, x, v);
54         } else {
55             modify(2 * p + 1, m, r, x, v);
56         }
57         pull(p);
58     }
59     void modify(int p, const Info &v) {
60         modify(1, 0, n, p, v);
61     }
62     Info rangeQuery(int p, int l, int r, int x, int y) {
63         if (l >= y || r <= x) {
64             return Info();
65         }
66         if (l >= x && r <= y) {
67             return info[p];
68         }
69         int m = (l + r) / 2;
70         push(p);
71         return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
72     }
73     Info rangeQuery(int l, int r) {
74         return rangeQuery(1, 0, n, l, r);
75     }
76     void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
77         if (l >= y || r <= x) {
78             return;
79         }
80         if (l >= x && r <= y) {
81             apply(p, v);
82             return;
83         }
84         int m = (l + r) / 2;
85         push(p);
86         rangeApply(2 * p, l, m, x, y, v);
87         rangeApply(2 * p + 1, m, r, x, y, v);
88         pull(p);
89     }
90     void rangeApply(int l, int r, const Tag &v) {
91         return rangeApply(1, 0, n, l, r, v);
92     }
93     void half(int p, int l, int r) {
94         if (info[p].act == 0) {
95             return;
96         }
97         if ((info[p].min + 1) / 2 == (info[p].max + 1) / 2) {
98             apply(p, {-(info[p].min + 1) / 2});
99             return;
100        }

```

```

101     int m = (l + r) / 2;
102     push(p);
103     half(2 * p, l, m);
104     half(2 * p + 1, m, r);
105     pull(p);
106 }
107 void half() {
108     half(1, 0, n);
109 }
110
111 template<class F>
112 int findFirst(int p, int l, int r, int x, int y, F &&pred) {
113     if (l >= y || r <= x) {
114         return -1;
115     }
116     if (l >= x && r <= y && !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 = findFirst(2 * p, l, m, x, y, pred);
125     if (res == -1) {
126         res = findFirst(2 * p + 1, m, r, x, y, pred);
127     }
128     return res;
129 }
130 template<class F>
131 int findFirst(int l, int r, F &&pred) {
132     return findFirst(1, 0, n, l, r, pred);
133 }
134 template<class F>
135 int findLast(int p, int l, int r, int x, int y, F &&pred) {
136     if (l >= y || r <= x) {
137         return -1;
138     }
139     if (l >= x && r <= y && !pred(info[p])) {
140         return -1;
141     }
142     if (r - l == 1) {
143         return l;
144     }
145     int m = (l + r) / 2;
146     push(p);
147     int res = findLast(2 * p + 1, m, r, x, y, pred);
148     if (res == -1) {
149         res = findLast(2 * p, l, m, x, y, pred);
150     }
151     return res;
152 }
153 template<class F>
154 int findLast(int l, int r, F &&pred) {
155     return findLast(1, 0, n, l, r, pred);
156 }
157
158 void maintainL(int p, int l, int r, int pre) {
159     if (info[p].difl > 0 && info[p].maxlowl < pre) {
160         return;
161     }
162     if (r - l == 1) {
163         info[p].max = info[p].maxlowl;
164         info[p].maxl = info[p].maxr = l;

```

```

165         info[p].maxlowl = info[p].maxlowr = -inf;
166         return;
167     }
168     int m = (l + r) / 2;
169     push(p);
170     maintainL(2 * p, l, m, pre);
171     pre = max(pre, info[2 * p].max);
172     maintainL(2 * p + 1, m, r, pre);
173     pull(p);
174 }
175 void maintainL() {
176     maintainL(1, 0, n, -1);
177 }
178 void maintainR(int p, int l, int r, int suf) {
179     if (info[p].difr > 0 && info[p].maxlowr < suf) {
180         return;
181     }
182     if (r - l == 1) {
183         info[p].max = info[p].maxlowl;
184         info[p].maxl = info[p].maxr = 1;
185         info[p].maxlowl = info[p].maxlowr = -inf;
186         return;
187     }
188     int m = (l + r) / 2;
189     push(p);
190     maintainR(2 * p + 1, m, r, suf);
191     suf = max(suf, info[2 * p + 1].max);
192     maintainR(2 * p, l, m, suf);
193     pull(p);
194 }
195 void maintainR() {
196     maintainR(1, 0, n, -1);
197 }
198 };
199
200 struct Tag {
201     int x = 0;
202     void apply(const Tag &t) & {
203         x = max(x, t.x);
204     }
205 };
206
207 struct Info {
208     int x = 0;
209     void apply(const Tag &t) & {
210         x = max(x, t.x);
211     }
212 };
213
214 Info operator+(const Info &a, const Info &b) {
215     return {max(a.x, b.x)};
216 }

```

4.5 取模类

4.5.1 取模类 (z 旧版)

```

1 constexpr int P = 998244353;
2 // assume -P <= x < 2P
3 int norm(int x) {
4     if (x < 0) {
5         x += P;

```

```

6      }
7      if (x >= P) {
8          x -= P;
9      }
10     return x;
11 }
12 template<class T>
13 T power(T a, i64 b) {
14     T res = 1;
15     for (; b; b /= 2, a *= a) {
16         if (b % 2) {
17             res *= a;
18         }
19     }
20     return res;
21 }
22 struct Z {
23     int x;
24     Z(int x = 0) : x(norm(x)) {}
25     Z(i64 x) : x(norm(x % P)) {}
26     int val() const {
27         return x;
28     }
29     Z operator-() const {
30         return Z(norm(P - x));
31     }
32     Z inv() const {
33         assert(x != 0);
34         return power(*this, P - 2);
35     }
36     Z &operator*=(const Z &rhs) {
37         x = i64(x) * rhs.x % P;
38         return *this;
39     }
40     Z &operator+=(const Z &rhs) {
41         x = norm(x + rhs.x);
42         return *this;
43     }
44     Z &operator-=(const Z &rhs) {
45         x = norm(x - rhs.x);
46         return *this;
47     }
48     Z &operator/=(const Z &rhs) {
49         return *this *= rhs.inv();
50     }
51     friend Z operator*(const Z &lhs, const Z &rhs) {
52         Z res = lhs;
53         res *= rhs;
54         return res;
55     }
56     friend Z operator+(const Z &lhs, const Z &rhs) {
57         Z res = lhs;
58         res += rhs;
59         return res;
60     }
61     friend Z operator-(const Z &lhs, const Z &rhs) {
62         Z res = lhs;
63         res -= rhs;
64         return res;
65     }
66     friend Z operator/(const Z &lhs, const Z &rhs) {
67         Z res = lhs;
68         res /= rhs;
69         return res;

```

```

70 }
71 friend istream &operator>>(istream &is, Z &a) {
72     i64 v;
73     is >> v;
74     a = Z(v);
75     return is;
76 }
77 friend ostream &operator<<(ostream &os, const Z &a) {
78     return os << a.val();
79 }
80 };

```

4.5.2 取模类 (MLong & MInt 新版)

```

1  /** 取模类 (MLong & MInt 新版)
2   * 根据输入内容动态修改 MOD 的方法 : z::setMod(p) 。
3   */
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14
15 constexpr i64 mul(i64 a, i64 b, i64 p) {
16     i64 res = a * b - i64(1.L * a * b / p) * p;
17     res %= p;
18     if (res < 0) {
19         res += p;
20     }
21     return res;
22 }
23 template<i64 P>
24 struct MLong {
25     i64 x;
26     constexpr MLong() : x{} {}
27     constexpr MLong(i64 x) : x{norm(x % getMod())} {}
28
29     static i64 Mod;
30     constexpr static i64 getMod() {
31         if (P > 0) {
32             return P;
33         } else {
34             return Mod;
35         }
36     }
37     constexpr static void setMod(i64 Mod_) {
38         Mod = Mod_;
39     }
40     constexpr i64 norm(i64 x) const {
41         if (x < 0) {
42             x += getMod();
43         }
44         if (x >= getMod()) {
45             x -= getMod();
46         }
47         return x;
48     }
49     constexpr i64 val() const {

```

```

50         return x;
51     }
52     explicit constexpr operator i64() const {
53         return x;
54     }
55     constexpr MLong operator-() const {
56         MLong res;
57         res.x = norm(getMod() - x);
58         return res;
59     }
60     constexpr MLong inv() const {
61         assert(x != 0);
62         return power(*this, getMod() - 2);
63     }
64     constexpr MLong &operator*=(MLong rhs) & {
65         x = mul(x, rhs.x, getMod());
66         return *this;
67     }
68     constexpr MLong &operator+=(MLong rhs) & {
69         x = norm(x + rhs.x);
70         return *this;
71     }
72     constexpr MLong &operator-=(MLong rhs) & {
73         x = norm(x - rhs.x);
74         return *this;
75     }
76     constexpr MLong &operator/=(MLong rhs) & {
77         return *this *= rhs.inv();
78     }
79     friend constexpr MLong operator*(MLong lhs, MLong rhs) {
80         MLong res = lhs;
81         res *= rhs;
82         return res;
83     }
84     friend constexpr MLong operator+(MLong lhs, MLong rhs) {
85         MLong res = lhs;
86         res += rhs;
87         return res;
88     }
89     friend constexpr MLong operator-(MLong lhs, MLong rhs) {
90         MLong res = lhs;
91         res -= rhs;
92         return res;
93     }
94     friend constexpr MLong operator/(MLong lhs, MLong rhs) {
95         MLong res = lhs;
96         res /= rhs;
97         return res;
98     }
99     friend constexpr istream &operator>>(istream &is, MLong &a) {
100        i64 v;
101        is >> v;
102        a = MLong(v);
103        return is;
104    }
105    friend constexpr ostream &operator<<(ostream &os, const MLong &a) {
106        return os << a.val();
107    }
108    friend constexpr bool operator==(MLong lhs, MLong rhs) {
109        return lhs.val() == rhs.val();
110    }
111    friend constexpr bool operator!=(MLong lhs, MLong rhs) {
112        return lhs.val() != rhs.val();
113    }

```

```

114 };
115
116 template<int>
117 i64 MLong<0LL>::Mod = i64(1E18) + 9;
118
119 template<int P>
120 struct MInt {
121     int x;
122     constexpr MInt() : x{} {}
123     constexpr MInt(i64 x) : x{norm(x % getMod())} {}
124
125     static int Mod;
126     constexpr static int getMod() {
127         if (P > 0) {
128             return P;
129         } else {
130             return Mod;
131         }
132     }
133     constexpr void setMod(int Mod_) {
134         Mod = Mod_;
135     }
136     constexpr int norm(int x) const {
137         if (x < 0) {
138             x += getMod();
139         }
140         if (x >= getMod()) {
141             x -= getMod();
142         }
143         return x;
144     }
145     constexpr int val() const {
146         return x;
147     }
148     explicit constexpr operator int() const {
149         return x;
150     }
151     constexpr MInt operator-() const {
152         MInt res;
153         res.x = norm(getMod() - x);
154         return res;
155     }
156     constexpr MInt inv() const {
157         assert(x != 0);
158         return power(*this, getMod() - 2);
159     }
160     constexpr MInt &operator*=(MInt rhs) & {
161         x = 1LL * x * rhs.x % getMod();
162         return *this;
163     }
164     constexpr MInt &operator+=(MInt rhs) & {
165         x = norm(x + rhs.x);
166         return *this;
167     }
168     constexpr MInt &operator-=(MInt rhs) & {
169         x = norm(x - rhs.x);
170         return *this;
171     }
172     constexpr MInt &operator/=(MInt rhs) & {
173         return *this *= rhs.inv();
174     }
175     friend constexpr MInt operator*(MInt lhs, MInt rhs) {
176         MInt res = lhs;
177         res *= rhs;

```

```

178     return res;
179 }
180 friend constexpr MInt operator+(MInt lhs, MInt rhs) {
181     MInt res = lhs;
182     res += rhs;
183     return res;
184 }
185 friend constexpr MInt operator-(MInt lhs, MInt rhs) {
186     MInt res = lhs;
187     res -= rhs;
188     return res;
189 }
190 friend constexpr MInt operator/(MInt lhs, MInt rhs) {
191     MInt res = lhs;
192     res /= rhs;
193     return res;
194 }
195 friend constexpr istream &operator>>(istream &is, MInt &a) {
196     i64 v;
197     is >> v;
198     a = MInt(v);
199     return is;
200 }
201 friend constexpr ostream &operator<<(ostream &os, const MInt &a) {
202     return os << a.val();
203 }
204 friend constexpr bool operator==(MInt lhs, MInt rhs) {
205     return lhs.val() == rhs.val();
206 }
207 friend constexpr bool operator!=(MInt lhs, MInt rhs) {
208     return lhs.val() != rhs.val();
209 }
210 };
211
212 template<>
213 int MInt<0>::Mod = 998244353;
214
215 template<int V, int P>
216 constexpr MInt<P> CInv = MInt<P>(V).inv();
217
218 constexpr int P = 1000000007;
219 using Z = MInt<P>;

```

4.5.3 动态取模类 (ModIntBase)

```

1 template<typename T>
2 constexpr T power(T a, u64 b) {
3     T res {1};
4     for (; b != 0; b /= 2, a *= a) {
5         if (b % 2 == 1) {
6             res *= a;
7         }
8     }
9     return res;
10 }
11
12 template<u32 P>
13 constexpr u32 mulMod(u32 a, u32 b) {
14     return 1ULL * a * b % P;
15 }
16
17 template<u64 P>
18 constexpr u64 mulMod(u64 a, u64 b) {

```

```

19     u64 res = a * b - u64(1.L * a * b / P - 0.5L) * P;
20     res %= P;
21     return res;
22 }
23
24 template<typename U, U P>
25 requires unsigned_integral<U>
26 struct ModIntBase {
27 public:
28     constexpr ModIntBase() : x {0} {}
29
30     template<typename T>
31     requires integral<T>
32     constexpr ModIntBase(T x_) : x {norm(x_ % T {P})} {}
33
34     constexpr static U norm(U x) {
35         if ((x >> (8 * sizeof(U) - 1) & 1) == 1) {
36             x += P;
37         }
38         if (x >= P) {
39             x -= P;
40         }
41         return x;
42     }
43
44     constexpr U val() const {
45         return x;
46     }
47
48     constexpr ModIntBase operator-() const {
49         ModIntBase res;
50         res.x = norm(P - x);
51         return res;
52     }
53
54     constexpr ModIntBase inv() const {
55         return power(*this, P - 2);
56     }
57
58     constexpr ModIntBase &operator*=(const ModIntBase &rhs) & {
59         x = mulMod<P>(x, rhs.val());
60         return *this;
61     }
62
63     constexpr ModIntBase &operator+=(const ModIntBase &rhs) & {
64         x = norm(x + rhs.x);
65         return *this;
66     }
67
68     constexpr ModIntBase &operator-=(const ModIntBase &rhs) & {
69         x = norm(x - rhs.x);
70         return *this;
71     }
72
73     constexpr ModIntBase &operator/=(const ModIntBase &rhs) & {
74         return *this *= rhs.inv();
75     }
76
77     friend constexpr ModIntBase operator*(ModIntBase lhs, const ModIntBase &rhs) {
78         lhs *= rhs;
79         return lhs;
80     }
81
82     friend constexpr ModIntBase operator+(ModIntBase lhs, const ModIntBase &rhs) {

```

```

83     lhs += rhs;
84     return lhs;
85 }
86
87 friend constexpr ModIntBase operator-(ModIntBase lhs, const ModIntBase &rhs) {
88     lhs -= rhs;
89     return lhs;
90 }
91
92 friend constexpr ModIntBase operator/(ModIntBase lhs, const ModIntBase &rhs) {
93     lhs /= rhs;
94     return lhs;
95 }
96
97 friend constexpr ostream &operator<<(ostream &os, const ModIntBase &a) {
98     return os << a.val();
99 }
100
101 friend constexpr bool operator==(ModIntBase lhs, ModIntBase rhs) {
102     return lhs.val() == rhs.val();
103 }
104
105 friend constexpr bool operator!=(ModIntBase lhs, ModIntBase rhs) {
106     return lhs.val() != rhs.val();
107 }
108
109 friend constexpr bool operator<(ModIntBase lhs, ModIntBase rhs) {
110     return lhs.val() < rhs.val();
111 }
112
113 private:
114     U x;
115 };
116
117 template<u32 P>
118 using ModInt = ModIntBase<u32, P>;
119
120 template<u64 P>
121 using ModInt64 = ModIntBase<u64, P>;
122
123 constexpr u32 P = 998244353;
124 using Z = ModInt<P>;

```

4.6 状压RMQ (RMQ)

```

1 template<class T, class Cmp = less<T>> struct RMQ {
2     const Cmp cmp = Cmp();
3     static constexpr unsigned B = 64;
4     using u64 = unsigned long long;
5     int n;
6     vector<vector<T>> a;
7     vector<T> pre, suf, ini;
8     vector<u64> stk;
9     RMQ() {}
10    RMQ(const vector<T> &v) {
11        init(v);
12    }
13    void init(const vector<T> &v) {
14        n = v.size();
15        pre = suf = ini = v;
16        stk.resize(n);
17        if (!n) {

```

```

18     return;
19 }
20 const int M = (n - 1) / B + 1;
21 const int lg = __lg(M);
22 a.assign(lg + 1, vector<T>(M));
23 for (int i = 0; i < M; i++) {
24     a[0][i] = v[i * B];
25     for (int j = 1; j < B && i * B + j < n; j++) {
26         a[0][i] = min(a[0][i], v[i * B + j], cmp);
27     }
28 }
29 for (int i = 1; i < n; i++) {
30     if (i % B) {
31         pre[i] = min(pre[i], pre[i - 1], cmp);
32     }
33 }
34 for (int i = n - 2; i >= 0; i--) {
35     if (i % B != B - 1) {
36         suf[i] = min(suf[i], suf[i + 1], cmp);
37     }
38 }
39 for (int j = 0; j < lg; j++) {
40     for (int i = 0; i + (2 << j) <= M; i++) {
41         a[j + 1][i] = min(a[j][i], a[j][i + (1 << j)], cmp);
42     }
43 }
44 for (int i = 0; i < M; i++) {
45     const int l = i * B;
46     const int r = min(1U * n, l + B);
47     u64 s = 0;
48     for (int j = l; j < r; j++) {
49         while (s && cmp(v[j], v[__lg(s) + 1])) {
50             s ^= 1ULL << __lg(s);
51         }
52         s |= 1ULL << (j - l);
53         stk[j] = s;
54     }
55 }
56 }
57 T operator()(int l, int r) {
58     if (l / B != (r - 1) / B) {
59         T ans = min(suf[l], pre[r - 1], cmp);
60         l = l / B + 1;
61         r = r / B;
62         if (l < r) {
63             int k = __lg(r - l);
64             ans = min({ans, a[k][l], a[k][r - (1 << k)]}, cmp);
65         }
66         return ans;
67     } else {
68         int x = B * (l / B);
69         return ini[__builtin_ctzll(stk[r - 1] >> (l - x)) + 1];
70     }
71 }
72 };

```

4.7 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 1;
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 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    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 cerr << t->val << " ";
138 dfs(t->ch[1]);
139 }
140
141 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     splay(j);
158     if (!v) {
159         return {j, nullptr};
160     }
161
162     splay(v);
163
164     Tree *u = v->ch[0];
165     if (u) {
166         v->ch[0] = u->p = nullptr;
167     }
168     // cerr << "split " << x << "\n";
169     // dfs(u);
170     // cerr << "\n";
171     // dfs(v);
172     // cerr << "\n";
173     return {u, v};
174 }
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 Matrix : array<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] = min(c[i][k], a[i][j] + b[j][k]);
17             }
18         }
19         c[i][3] = 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     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]->prd : Matrix());
51 }
52 bool isroot(Node *t) {
53     return t->p == nullptr || (t->p->ch[0] != t && t->p->ch[1] != t);
54 }
55 int pos(Node *t) {
56     return t->p->ch[1] == t;
57 }
58 void pushAll(Node *t) {
59     if (!isroot(t)) {
60         pushAll(t->p);
61     }
62     push(t);
63 }
64 void rotate(Node *t) {
65     Node *q = t->p;

```

```

66     int x = !pos(t);
67     q->ch[!x] = t->ch[x];
68     if (t->ch[x]) {
69         t->ch[x]->p = q;
70     }
71     t->p = q->p;
72     if (!isroot(q)) {
73         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 array<i64, 4> get(Node *t) {
95     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] = min(ans[i], t->prd[i][j]);
101         }
102     }
103     return ans;
104 }
105
106 void access(Node *t) {
107     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 += min({res[1], res[2], res[3]});
114             i->sumb += min({res[0], res[1], res[2], res[3]});
115         }
116         i->ch[1] = q;
117         i->sumg -= old[0];
118         i->sumh -= min({old[1], old[2], old[3]});
119         i->sumb -= min({old[0], old[1], old[2], old[3]});
120         old = get(i);
121         i->update();
122         pull(i);
123     }
124     splay(t);
125 }

```

```

1 constexpr int D = 27;
2 struct Info {

```

```

3   int up[D][2] {};
4   int down[D][2] {};
5   int t = 0;
6   i64 ans = 0;
7 };
8
9 Info operator+(const Info &a, const Info &b) {
10    Info c;
11    c.t = a.t ^ b.t;
12    c.ans = a.ans + b.ans;
13    for (int i = 0; i < D; i++) {
14        for (int j = 0; j < 2; j++) {
15            c.ans += (1LL << i) * a.down[i][j] * b.up[i][j ^ 1];
16            c.up[i][j] += a.up[i][j] + b.up[i][j ^ (a.t >> i & 1)];
17            c.down[i][j] += b.down[i][j] + a.down[i][j ^ (b.t >> i & 1)];
18        }
19    }
20    return c;
21 }
22 struct Node {
23     Node *ch[2], *p;
24     Info val;
25     Info tot;
26     int cnt[D][2];
27     i64 pair[D][2];
28     i64 sum;
29     Node() : ch{nullptr, nullptr}, p(nullptr), cnt {}, pair {}, sum {} {}
30 };
31 void pull(Node *t) {
32     t->tot = (t->ch[0] ? t->ch[0]->tot : Info {}) + t->val + (t->ch[1]-
33     >tot : Info {});
34 }
35 bool isroot(Node *t) {
36     return t->p == nullptr || (t->p->ch[0] != t && t->p->ch[1] != t);
37 }
38 int pos(Node *t) {
39     return t->p->ch[1] == t;
40 }
41 void rotate(Node *t) {
42     Node *q = t->p;
43     int x = !pos(t);
44     q->ch[!x] = t->ch[x];
45     if (t->ch[x]) {
46         t->ch[x]->p = q;
47     }
48     t->p = q->p;
49     if (!isroot(q)) {
50         q->p->ch[pos(q)] = t;
51     }
52     t->ch[x] = q;
53     q->p = t;
54     pull(q);
55 }
56 void update(Node *t) {
57     t->val.ans = t->val.t + t->sum;
58     for (int i = 0; i < D; i++) {
59         t->val.ans += (1LL << i) * t->pair[i][t->val.t >> i & 1];
60         for (int j = 0; j < 2; j++) {
61             t->val.up[i][j] = t->cnt[i][j ^ (t->val.t >> i & 1)];
62             t->val.down[i][j] = t->cnt[i][j ^ (t->val.t >> i & 1)];
63         }
64         t->val.up[i][t->val.t >> i & 1]++;
65         t->val.down[i][t->val.t >> i & 1]++;
66     }
67 }

```

```

66     pull(t);
67 }
68 void splay(Node *t) {
69     while (!isroot(t)) {
70         if (!isroot(t->p)) {
71             if (pos(t) == pos(t->p)) {
72                 rotate(t->p);
73             } else {
74                 rotate(t);
75             }
76         }
77         rotate(t);
78     }
79     pull(t);
80 }
81 void add(Node *t, Info s) {
82     for (int i = 0; i < D; i++) {
83         for (int x = 0; x < 2; x++) {
84             t->pair[i][x] += s.up[i][1 ^ x];
85             for (int j = 0; j < 2; j++) {
86                 t->pair[i][x] += t->cnt[i][j] * s.up[i][j ^ 1 ^ x];
87             }
88         }
89         for (int j = 0; j < 2; j++) {
90             t->cnt[i][j] += s.up[i][j];
91         }
92     }
93     t->sum += s.ans;
94 }
95 void del(Node *t, Info s) {
96     t->sum -= s.ans;
97     for (int i = 0; i < D; i++) {
98         for (int j = 0; j < 2; j++) {
99             t->cnt[i][j] -= s.up[i][j];
100        }
101        for (int x = 0; x < 2; x++) {
102            for (int j = 0; j < 2; j++) {
103                t->pair[i][x] -= t->cnt[i][j] * s.up[i][j ^ 1 ^ x];
104            }
105            t->pair[i][x] -= s.up[i][1 ^ x];
106        }
107    }
108 }
109 void access(Node *t, int v) {
110     Info lst;
111     for (Node *i = t, *q = nullptr; i; q = i, i = i->p) {
112         splay(i);
113         if (i->ch[1]) {
114             add(i, i->ch[1]->tot);
115         }
116         i->ch[1] = q;
117         if (q) {
118             del(i, lst);
119         } else {
120             i->val.t = v;
121         }
122         lst = i->tot;
123         update(i);
124     }
125     splay(t);
126 }

```

4.8 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
14 Node *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
30 int 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
44 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 {l, 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 1;
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 {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 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     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 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             tie(t->l, t2->l) = split(t->l, l, m, x, rev);
56             t2->r = t->r;
57             t->r = nullptr;
58         } else {
59             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             tie(t->r, t2->r) = split(t->r, m, r, x, rev);
64             t2->l = t->l;
65             t->l = nullptr;
66         } else {
67             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 }
```

4.9 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 ostream &operator<<(ostream &os, Frac x) {
74     T g = 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 };

```

4.10 线性基 (Basis)

```

1 struct Basis {
2     int a[20] {};
3     int t[20] {};
4
5     Basis() {
6         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                    swap(a[i], x);
14                    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 };

```

4.11 高精度 (BigInt)

```

1 constexpr int N = 1000;
2
3 struct BigInt {
4     int a[N];
5     BigInt(int x = 0) : a{} {
6         for (int i = 0; x; i++) {

```

```

7         a[i] = x % 10;
8         x /= 10;
9     }
10    }
11    BigInt &operator*=(int x) {
12        for (int i = 0; i < N; i++) {
13            a[i] *= x;
14        }
15        for (int i = 0; i < N - 1; i++) {
16            a[i + 1] += a[i] / 10;
17            a[i] %= 10;
18        }
19        return *this;
20    }
21    BigInt &operator/=(int x) {
22        for (int i = N - 1; i >= 0; i--) {
23            if (i) {
24                a[i - 1] += a[i] % x * 10;
25            }
26            a[i] /= x;
27        }
28        return *this;
29    }
30    BigInt &operator+=(const BigInt &x) {
31        for (int i = 0; i < N; i++) {
32            a[i] += x.a[i];
33            if (a[i] >= 10) {
34                a[i + 1] += 1;
35                a[i] -= 10;
36            }
37        }
38        return *this;
39    }
40};
41
42 ostream &operator<<(ostream &o, const BigInt &a) {
43     int t = N - 1;
44     while (a.a[t] == 0) {
45         t--;
46     }
47     for (int i = t; i >= 0; i--) {
48         o << a.a[i];
49     }
50     return o;
51 }

```

4.12 Link-Cut Tree

```

1 namespace SegT {
2     int tag[8 * N];
3     int64_t wsum[8 * N], sum[8 * N];
4     void add(int p, int l, int r, int v) {
5         sum[p] += v * (r - l);
6         wsum[p] += 111 * v * (r - l) * (l + r + 1) / 2;
7         tag[p] += v;
8     }
9     void push(int p, int l, int r) {
10        int m = (l + r) / 2;
11        add(2 * p, l, m, tag[p]);
12        add(2 * p + 1, m, r, tag[p]);
13        tag[p] = 0;
14    }

```

```

15 void pull(int p) {
16     sum[p] = sum[2 * p] + sum[2 * p + 1];
17     wsum[p] = wsum[2 * p] + wsum[2 * p + 1];
18 }
19 void rangeAdd(int p, int l, int r, int x, int y, int v) {
20     if (l >= y || r <= x)
21         return;
22     if (l >= x && r <= y)
23         return add(p, l, r, v);
24     push(p, l, r);
25     int m = (l + r) / 2;
26     rangeAdd(2 * p, l, m, x, y, v);
27     rangeAdd(2 * p + 1, m, r, x, y, v);
28     pull(p);
29 }
30 int64_t query(int p, int l, int r, int x) {
31     if (l >= x)
32         return sum[p] * x;
33     if (r <= x)
34         return wsum[p];
35     int m = (l + r) / 2;
36     push(p, l, r);
37     return query(2 * p, l, m, x) + query(2 * p + 1, m, r, x);
38 }
39 int get(int p, int l, int r, int x) {
40     if (r - l == 1)
41         return sum[p];
42     int m = (l + r) / 2;
43     push(p, l, r);
44     if (x < m) {
45         return get(2 * p, l, m, x);
46     } else {
47         return get(2 * p + 1, m, r, x);
48     }
49 }
50 }
51 namespace LCT {
52     int ch[2 * N][2], p[2 * N], endp[2 * N], mn[2 * N], mx[2 * N];
53     bool isroot(int t) {
54         return ch[p[t]][0] != t && ch[p[t]][1] != t;
55     }
56     bool pos(int t) {
57         return ch[p[t]][1] == t;
58     }
59     void pull(int t) {
60         mn[t] = max(0, ch[t][0] ? mn[ch[t][0]] : SAM::len[SAM::link[t]]);
61         mx[t] = ch[t][1] ? mx[ch[t][1]] : SAM::len[t];
62     }
63     void rotate(int t) {
64         int k = !pos(t);
65         int q = p[t];
66         ch[q][!k] = ch[t][k];
67         if (ch[t][k])
68             p[ch[t][k]] = q;
69         p[t] = p[q];
70         if (isroot(q)) {
71             endp[t] = endp[q];
72         } else {
73             ch[p[q]][pos(q)] = t;
74         }
75         ch[t][k] = q;
76         p[q] = t;
77         pull(q);
78     }

```

```

79 void splay(int t) {
80     while (!isroot(t)) {
81         int q = p[t];
82         if (!isroot(q))
83             rotate(pos(t) == pos(q) ? q : t);
84         rotate(t);
85     }
86     pull(t);
87 }
88 void access(int t, int len) {
89     for (int i = t, u = 0; i; u = i, i = p[i]) {
90         splay(i);
91         if (ch[i][1])
92             endp[ch[i][1]] = endp[i];
93         ch[i][1] = 0;
94         pull(i);
95         if (u)
96             SegT::rangeAdd(1, 0, n, endp[i] - mx[i], endp[i] - mn[i], -1);
97         ch[i][1] = u;
98         pull(i);
99     }
100    splay(t);
101    endp[t] = len;
102    SegT::rangeAdd(1, 0, n, len - mx[t], len - mn[t], 1);
103 }
104 void cut(int t) {
105     splay(t);
106     if (ch[t][0]) {
107         endp[ch[t][0]] = endp[t];
108         p[ch[t][0]] = p[t];
109         p[t] = 0;
110         ch[t][0] = 0;
111         pull(t);
112     } else {
113         p[t] = 0;
114     }
115 }
116 void link(int t, int x) {
117     p[t] = x;
118 }
119 }
```

```

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         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 makeroott(Node *t) {
73     access(t);
74     reverse(t);
75 }
76 void link(Node *x, Node *y) {
77     makeroott(x);
78     x->p = y;
79 }
80 void split(Node *x, Node *y) {
81     makeroott(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 }
```

/END/

5 字符串

5.1 马拉车 (Manacher)

```

1 vector<int> manacher(string s) {
2     string t = "#";
3     for (auto c : s) {
4         t += c;
5         t += '#';
6     }
7     int n = t.size();
8     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] = 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 }
22 }
```

5.2 Z函数

```

1 vector<int> Z(string s) {
2     int n = s.size();
3     vector<int> z(n + 1);
4     z[0] = n;
5     for (int i = 1, j = 1; i < n; i++) {
6         z[i] = max(0, 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

5.3.1 后缀数组 (SuffixArray 旧版)

```

1 struct SuffixArray {
2     int n;
3     vector<int> sa, rk, lc;
4     SuffixArray(const string &s) {
5         n = s.length();
6         sa.resize(n);
7         lc.resize(n - 1);
8         rk.resize(n);
9         iota(sa.begin(), sa.end(), 0);
10        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     int k = 1;
15     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         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         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 };

```

5.3.2 后缀数组 (SA及其应用 新版)

```

1 struct SA {
2     int n;
3     vector<int> sa, rk, lc;
4
5     SA(string s) {
6         n = s.size();
7         sa.resize(n);
8         lc.resize(n - 1);
9         rk.resize(n);
10        iota(sa.begin(), sa.end(), 0);
11        sort(sa.begin(), sa.end(),
12              [&](int a, int b) {
13                  return s[a] < s[b];
14              });
15        rk[sa[0]] = 0;
16        for (int i = 1; i < n; i++) {
17            rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
18        }
19        int k = 1;
20        vector<int> tmp, cnt(n);
21        tmp.reserve(n);

```

```

22     while (rk[sa[n - 1]] < n - 1) {
23         tmp.clear();
24         for (int i = 0; i < k; i++) {
25             tmp.push_back(n - k + i);
26         }
27         for (auto i : sa) {
28             if (i >= k) {
29                 tmp.push_back(i - k);
30             }
31         }
32         fill(cnt.begin(), cnt.end(), 0);
33         for (int i = 0; i < n; i++) {
34             cnt[rk[i]]++;
35         }
36         for (int i = 1; i < n; i++) {
37             cnt[i] += cnt[i - 1];
38         }
39         for (int i = n - 1; i >= 0; i--) {
40             sa[--cnt[rk[tmp[i]]]] = tmp[i];
41         }
42         swap(rk, tmp);
43         rk[sa[0]] = 0;
44         for (int i = 1; i < n; i++) {
45             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]);
46         }
47         k *= 2;
48     }
49     for (int i = 0, j = 0; i < n; i++) {
50         if (rk[i] == 0) {
51             j = 0;
52         } else {
53             for (j -= j > 0; i + j < n && sa[rk[i] - 1] + j < n && s[i + j] == s[sa[rk[i] - 1] + j]; ) {
54                 j++;
55             }
56             lc[rk[i] - 1] = j;
57         }
58     }
59 }
60 };
61
62 void solve() {
63     constexpr int K = 21;
64     vector st(K, vector<int>(l - 1));
65     st[0] = lc;
66     for (int j = 0; j < K - 1; j++) {
67         for (int i = 0; i + (2 << j) <= l - 1; i++) {
68             st[j + 1][i] = min(st[j][i], st[j][i + (1 << j)]);
69         }
70     }
71
72     auto rmq = [&](int l, int r) {
73         int k = __lg(r - l);
74         return min(st[k][l], st[k][r - (1 << k)]);
75     };
76
77     auto lcp = [&](int i, int j) {
78         if (i == j || i == n || j == n) {
79             return min(n - i, n - j);
80         }
81         int a = rk[i];
82         int b = rk[j];
83         if (a > b) {

```

```

84         swap(a, b);
85     }
86     return min({n - i, n - j, rmq(a, b)});
87 };
88
89 auto lcs = [&](int i, int j) {
90     if (i == j || i == 0 || j == 0) {
91         return min(i, j);
92     }
93     int a = rk[n + n - i];
94     int b = rk[n + n - j];
95     if (a > b) {
96         swap(a, b);
97     }
98     return min({i, j, rmq(a, b)});
99 };
100 }
```

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        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            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         array<int, ALPHABET_SIZE> next;
7         Node() : len{}, link{}, next{} {}
8     };
9     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 = 26;
3     struct Node {
4         int len;
5         int link;
6         int cnt;
7         array<int, ALPHABET_SIZE> next;
8         Node() : len{}, link{}, cnt{}, next{} {}
9     };
10    vector<Node> t;
11    int suff;
12    string s;
13    PAM() {
14        init();
15    }
16    void init() {
17        t.assign(2, Node());
18        t[0].len = -1;
19        suff = 1;
20        s.clear();
21    }
22    int newNode() {
23        t.emplace_back();
24        return t.size() - 1;
25    }
26    bool add(char c) {
27        int pos = s.size();
28        s += c;
29        int let = c - 'a';
30        int cur = suff, curlen = 0;
31        while (true) {
32            curlen = t[cur].len;
33            if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
34                break;
35            }
36            cur = t[cur].link;
37        }
38        if (t[cur].next[let]) {
39            suff = t[cur].next[let];
40            return false;
41        }
42        int num = newNode();
43        suff = num;
44        t[num].len = t[cur].len + 2;
45        t[cur].next[let] = num;
46        if (t[num].len == 1) {
47            t[num].link = 1;
48            t[num].cnt = 1;
49            return true;

```

```

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

```

5.6 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         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     vector<int> x;
24     AC(AC &&a) : x(move(a.x)) {}
25     AC(vector<string> s, vector<int> w) {
26         x = {newNode(), newNode()};
27         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 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 ~AC() {
60     for (auto p : x) {
61         bin[nBin++] = p;
62     }
63 }
64 i64 query(const string &s) const {
65     i64 ans = 0;
66     int p = x[1];
67     for (int i = 0; i < int(s.length()); i++) {
68         int c = s[i] - 'a';
69         p = node[p].next[c];
70         ans += node[p].sum;
71     }
72     return ans;
73 }
74 }
75 };

```

5.6.2 AC自动机 (AhoCorasick, with vector 新版)

```

1 struct AhoCorasick {
2     static constexpr int ALPHABET = 26;
3     struct Node {
4         int len;
5         int link;
6         array<int, ALPHABET> next;
7         Node() : link{}, next{} {}
8     };
9
10    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 vector<int> &a) {
28     int p = 1;
29     for (auto x : a) {
30         if (t[p].next[x] == 0) {
31             t[p].next[x] = newNode();
32             t[t[p].next[x]].len = t[p].len + 1;
33         }
34         p = t[p].next[x];
35     }
36     return p;
37 }
38
39 int add(const string &a, char offset = 'a') {
40     vector<int> b(a.size());
41     for (int i = 0; i < a.size(); i++) {
42         b[i] = a[i] - offset;
43     }
44     return add(b);
45 }
46
47 void work() {
48     queue<int> q;
49     q.push(1);
50
51     while (!q.empty()) {
52         int x = q.front();
53         q.pop();
54
55         for (int i = 0; i < ALPHABET; i++) {
56             if (t[x].next[i] == 0) {
57                 t[x].next[i] = t[t[x].link].next[i];
58             } else {
59                 t[t[x].next[i]].link = t[t[x].link].next[i];
60                 q.push(t[x].next[i]);
61             }
62         }
63     }
64 }
65
66 int next(int p, int x) {
67     return t[p].next[x];
68 }
69
70 int next(int p, char c, char offset = 'a') {
71     return next(p, c - 'a');
72 }
73
74 int link(int p) {
75     return t[p].link;
76 }
77
78 int len(int p) {
79     return t[p].len;
80 }
81

```

```

82     int size() {
83         return t.size();
84     }
85 };

```

5.6.3 AC自动机 (AhoCorasick, with string 新版)

```

1 struct AhoCorasick {
2     static constexpr int ALPHABET = 26;
3     struct Node {
4         int len;
5         int link;
6         array<int, ALPHABET> next;
7         Node() : len{0}, link{0}, next{} {}
8     };
9
10    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 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        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    }

```

```

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 字符串哈希（随机底模例题）

```

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 = array<int, 2>;
25
26 int main() {
27     ios::sync_with_stdio(false);
28     cin.tie(nullptr);
29
30     mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
31
32     const int P = findPrime(rng() % 900000000 + 100000000);
33
34     string s, x;
35     cin >> s >> x;
36
37     int n = s.length();
38     int m = x.length();
39
40     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 - 1]) % 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             cout << i + 1 << " " << i + m - 1 << "\n";
61             cout << i + m << " " << i + 2 * m - 2 << "\n";
62             return 0;
63         }
64     }
65
66     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] = max(0, 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] = max(0, 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 = min(m, f[i]);
94
95         for (auto j : {m - l, m - l - 1}) {
96             if (j <= 0) {
97                 continue;
98             }
99             if (j <= i && (get(i - j, i) + get(i, i + m)) % P == px) {
100                 cout << i - j + 1 << " " << i << "\n";
101                 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         cout << i + 1 << " " << i + m << "\n";
106         cout << i + m + 1 << " " << i + m + j << "\n";
107         return 0;
108     }
109 }
110 }
111 return 0;
112 }
113 }
```

5.8 最长公共前缀 LCP (例题)

```

1 constexpr int L = 2E6 + 10;
2
3 int len[L];
4 int lnk[L];
5 int nxt[L][26];
6
7 int f[L];
8 int tot = 1;
9
10 vector<int> adj[L];
11
12 int extend(int p, int c) {
13     if (nxt[p][c]) {
14         int q = nxt[p][c];
15         if (len[q] == len[p] + 1) {
16             return q;
17         }
18         int r = ++tot;
19         len[r] = len[p] + 1;
20         lnk[r] = lnk[q];
21         copy(nxt[q], nxt[q] + 26, nxt[r]);
22         lnk[q] = r;
23         while (nxt[p][c] == q) {
24             nxt[p][c] = r;
25             p = lnk[p];
26         }
27         return r;
28     }
29     int cur = ++tot;
30     len[cur] = len[p] + 1;
31     while (!nxt[p][c]) {
32         nxt[p][c] = cur;
33         p = lnk[p];
34     }
35     lnk[cur] = extend(p, c);
36     return cur;
37 }
38
39 int main() {
40     ios::sync_with_stdio(false);
41     cin.tie(nullptr);
42
43     fill(nxt[0], nxt[0] + 26, 1);
44     len[0] = -1;
45
46     int N;
47     cin >> N;
48
49     vector<string> S(N);
50     for (int i = 0; i < N; i++) {
```

```

51     cin >> S[i];
52     int p = 1;
53     for (auto c : S[i]) {
54         p = extend(p, c - 'a');
55         if (f[p] != -1) {
56             if (f[p] == 0) {
57                 f[p] = i + 1;
58             } else if (f[p] != i + 1) {
59                 f[p] = -1;
60             }
61         }
62     }
63 }
64
65 for (int i = 1; i <= tot; i++) {
66     adj[lnk[i]].push_back(i);
67 }
68 }
```

5.9 字典树 Trie

```

1 constexpr i64 inf = 1E18;
2
3 constexpr int N = 1E6 + 10;
4
5 int trie[N][26];
6 int tot;
7
8 int newNode() {
9     tot++;
10    fill(trie[tot], trie[tot] + 26, 0);
11    val[tot] = inf;
12    return tot;
13 }
14
15 void solve() {
16     /* init
17     tot = 0;
18     newNode();
19
20     /* insert
21     for (int i = 0; i < N; i++) {
22         int p = 1;
23         int l = S[i].size();
24         for (int j = 0; j < l; j++) {
25             int x = S[i][j] - 'a';
26             if (!trie[p][x]) {
27                 trie[p][x] = newNode();
28             }
29             p = trie[p][x];
30             /* 处理
31             /* val[p] = min(val[p], l + K + f[(K - (l - j - 1) % K) % K]);
32         }
33     }
34
35     /* query
36     for (int i = 0; i < L; i++) {
37         int p = 1;
38         for (int j = i; j < L; j++) {
39             int x = T[j] - 'a';
40             p = trie[p][x];
41             if (!p) {
```

```

42         continue;
43     }
44     /* 处理
45     /* dp[j + 1] = min(dp[j + 1], dp[i] + val[p]);
46   }
47 }
48 }
```

```

1 int tot;
2 int trie[N][2];
3 int f[N];
4
5 int newNode() {
6     int x = ++tot;
7     trie[x][0] = trie[x][1] = 0;
8     f[x] = inf;
9     return x;
10}
11void add(int x, int i) {
12    int p = 1;
13    for (int j = 29; j >= 0; j--) {
14        int &q = trie[p][x >> j & 1];
15        if (q == 0) {
16            q = newNode();
17        }
18        p = q;
19        f[p] = min(f[p], i);
20    }
21}
22
23int query(int a, int b) {
24    int ans1 = inf, ans2 = inf;
25    int p = 1;
26    for (int i = 29; i >= 0; i--) {
27        int d = a >> i & 1;
28        int e = b >> i & 1;
29        if (e) {
30            ans1 = min(ans1, f[trie[p][d]]);
31        } else {
32            ans2 = min(ans2, f[trie[p][d ^ 1]]);
33        }
34        p = trie[p][e ^ d];
35    }
36    ans1 = min(ans1, f[p]);
37    ans2 = min(ans2, f[p]);
38    if (ans1 == inf || ans2 == inf) {
39        return -1;
40    }
41    return max({1, ans1, ans2});
42}
```

```

1 int trie[N][2];
2 int cnt[N][2];
3
4 int tot = 0;
5 int newNode() {
6     int x = ++tot;
7     trie[x][0] = trie[x][1] = 0;
8     cnt[x][0] = cnt[x][1] = 0;
9     return x;
10}
11
```

```

12 void add(int x, int d, int t = 1) {
13     int p = 1;
14     cnt[p][d] += t;
15     for (int i = 29; i >= 0; i--) {
16         int u = x >> i & 1;
17         if (!trie[p][u]) {
18             trie[p][u] = newNode();
19         }
20         p = trie[p][u];
21         cnt[p][d] += t;
22     }
23 }
24
25 int query(int x, int d) {
26     int p = 1;
27     if (!cnt[p][d]) {
28         return 0;
29     }
30     int ans = 0;
31     for (int i = 29; i >= 0; i--) {
32         int u = x >> i & 1;
33         if (cnt[trie[p][u ^ 1]][d]) {
34             ans |= 1 << i;
35             p = trie[p][u ^ 1];
36         } else {
37             p = trie[p][u];
38         }
39     }
40     return ans;
41 }
```

```

1 constexpr int N = 1E7;
2 constexpr int inf = 1E9;
3 int tot;
4 int trie[N][2];
5 int f[N];
6
7 int newNode() {
8     int x = ++tot;
9     trie[x][0] = trie[x][1] = 0;
10    f[x] = inf;
11    return x;
12 }
13 void add(int x, int i) {
14     int p = 1;
15     for (int j = 29; j >= 0; j--) {
16         int &q = trie[p][x >> j & 1];
17         if (q == 0) {
18             q = newNode();
19         }
20         p = q;
21         f[p] = min(f[p], i);
22     }
23 }
24
25 int query(int a, int b) {
26     int ans1 = inf, ans2 = inf;
27     int p = 1;
28     for (int i = 29; i >= 0; i--) {
29         int d = a >> i & 1;
30         int e = b >> i & 1;
31         if (e) {
32             ans1 = min(ans1, f[trie[p][d]]);
```

```

33     } else {
34         ans2 = min(ans2, f[trie[p][d ^ 1]]);
35     }
36     p = trie[p][e ^ d];
37 }
38 ans1 = min(ans1, f[p]);
39 ans2 = min(ans2, f[p]);
40 if (ans1 == inf || ans2 == inf) {
41     return -1;
42 }
43 return max({1, ans1, ans2});
44 }
```

5.10 前缀函数 (KMP)

```

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

/END/



Author by *Jiangly*
Tyrp by *Wida*

Version: 1.8.7 (2024.10.07)

github.com/hh2048
cnblogs.com/WIDA