目录

1 杂类

1.1 int128 库函数自定义

```cpp
ostream &operator<<(ostream &os, i128 n) {
    if (n == 0) {
        return os << 0;
    }
    string s;
    while (n > 0) {
        s += char('0' + n % 10);
        n /= 10;
    }
    reverse(s.begin(), s.end());
    return os << s;
}
i128 toi128(const string &s) {
    i128 n = 0;
    for (auto c : s) {
        n = n * 10 + (c - '0');
    }
    return n;
}
i128 sqrti128(i128 n) {
    i128 lo = 0, hi = 1E16;
    while (lo < hi) {
        i128 x = (lo + hi + 1) / 2;
        if (x * x <= n) {
            lo = x;
        } else {
            hi = x - 1;
        }
    }
    return lo;
}

i128 gcd(i128 a, i128 b) {
    while (b) {
        a %= b;
        swap(a, b);
    }
    return a;
}
```

1.2 常用库函数重载

```cpp
/**   上取整下取整   **/
i64 ceilDiv(i64 n, i64 m) {
    if (n >= 0) {
        return (n + m - 1) / m;
    } else {
        return n / m;
    }
}
i64 floorDiv(i64 n, i64 m) {
    if (n >= 0) {
        return n / m;
    } else {
        return (n - m + 1) / m;
    }
}
/**   最大值赋值   **/
template<class T>
void chmax(T &a, T b) {
    if (a < b) {
        a = b;
    }
}
/**   最大公约数   **/
i128 gcd(i128 a, i128 b) {
    return b ? gcd(b, a % b) : a;
}
/**   精确开平方   **/
i64 sqrt(i64 n) {
    i64 s = sqrt(n);
    while (s * s > n) {
        s--;
    }
    while ((s + 1) * (s + 1) <= n) {
        s++;
    }
    return s;
}
/**   精确开平方   **/
i64 get(i64 n) {
    i64 u = sqrt(2.0L * n);
    while (u * (u + 1) / 2 < n) {
        u++;
    }
    while (u * (u - 1) / 2 + 1 > n) {
        u--;
    }
    return u;
}
/**   求 Log   **/
int logi(int a, int b) {
    int t = 0;
    i64 v = 1;
    while (v < b) {
        v *= a;
        t++;
    }
    return t;
}
int llog(int a, int b) {
    if (a <= b) {
        int l = logi(a, b);
        return (l == 0 ? 0 : __lg(2 * l - 1));
    }
    int l = logi(b, a + 1) - 1;
    assert(l > 0);
    return -__lg(l);
}
```

1.3 字符调整

```
/**   大小写转换、获取字母序   **/
void rev(string &s) {
    int l = s.size();
    for (int i = 1; i < l; i += 2) {
        if (isupper(s[i])) {
            s[i] = tolower(s[i]);
```

```cpp
        } else {
            s[i] = toupper(s[i]);
        }
    }
}

int get(char c) {
    int x;
    if (islower(c)) {
        x = c - 'a';
    } else {
        x = 26 + c - 'A';
    }
    return x;
}
```

1.4 二分算法

1.4.1 二分算法（整数域）

```cpp
/**   二分算法（整数域）： 前驱   **/
int lo = 1, hi = 1E9;
while (lo < hi) {
    int m = (lo + hi + 1) / 2;
    if (check(m)) {
        lo = m;
    } else {
        hi = m - 1;
    }
}
cout << lo << "\n";
/**   二分算法（整数域）：后继   **/
int lo = 1, hi = n;
while (lo < hi) {
    int m = (lo + hi) / 2;
    if (check(m)) {
        hi = m;
    } else {
        lo = m + 1;
    }
}
cout << lo << "\n";
```

1.4.2 二分算法（实数域）

```cpp
/**   二分算法（实数域）   **/
auto check = [&](double t) {
    // write
};

double lo = 0;
double hi = 1E12;
while (hi - lo > max(1.0, lo) * eps) {
    double x = (lo + hi) / 2;
    if (check(x)) {
        hi = x;
    } else {
        lo = x;
    }
}

```

```cpp
cout << lo << "\n";

/**   二分算法（实数域）   **/
using i64 = long long;
using real = long double;

constexpr real eps = 1E-7;

auto get = [&](const auto &f) {
    real lo = -1E4, hi = 1E4;
    while (hi - lo > 3 * eps) {
        real x1 = (lo + hi - eps) / 2;
        real x2 = (lo + hi + eps) / 2;
        if (f(x1) > f(x2)) {
            lo = x1;
        } else {
            hi = x2;
        }
    }
    return f((lo + hi) / 2);
};

cout << get([&](real px) {
    return get([&](real py) {
        // write
    });
}) << "\n";
```

/END/

2 图与网络

2.1 强连通分量缩点（SCC）

```cpp
struct SCC {
    int n;
    vector<vector<int>> adj;
    vector<int> stk;
    vector<int> dfn, low, bel;
    int cur, cnt;

    SCC() {}
    SCC(int n) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        adj.assign(n, {});
        dfn.assign(n, -1);
        low.resize(n);
        bel.assign(n, -1);
        stk.clear();
        cur = cnt = 0;
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
    }

    void dfs(int x) {
        dfn[x] = low[x] = cur++;
        stk.push_back(x);

        for (auto y : adj[x]) {
            if (dfn[y] == -1) {
                dfs(y);
                low[x] = min(low[x], low[y]);
            } else if (bel[y] == -1) {
                low[x] = min(low[x], dfn[y]);
            }
        }

        if (dfn[x] == low[x]) {
            int y;
            do {
                y = stk.back();
                bel[y] = cnt;
                stk.pop_back();
            } while (y != x);
            cnt++;
        }
    }

    vector<int> work() {
        for (int i = 0; i < n; i++) {
            if (dfn[i] == -1) {
                dfs(i);
            }
        }
        return bel;
    }
};
```

2.2 割边与割边缩点（EBCC）

```
set<pair<int, int>> E;

struct EBCC {
    int n;
    vector<vector<int>> adj;
    vector<int> stk;
    vector<int> dfn, low, bel;
    int cur, cnt;

    EBCC() {}
    EBCC(int n) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        adj.assign(n, {});
        dfn.assign(n, -1);
        low.resize(n);
        bel.assign(n, -1);
        stk.clear();
        cur = cnt = 0;
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void dfs(int x, int p) {
        dfn[x] = low[x] = cur++;
        stk.push_back(x);

        for (auto y : adj[x]) {
            if (y == p) {
                continue;
            }
            if (dfn[y] == -1) {
                E.emplace(x, y);
                dfs(y, x);
                low[x] = min(low[x], low[y]);
            } else if (bel[y] == -1 && dfn[y] < dfn[x]) {
                E.emplace(x, y);
                low[x] = min(low[x], dfn[y]);
            }
        }

        if (dfn[x] == low[x]) {
            int y;
            do {
                y = stk.back();
                bel[y] = cnt;
                stk.pop_back();
            } while (y != x);
            cnt++;
        }
    }

```

```cpp
    vector<int> work() {
        dfs(0, -1);
        return bel;
    }

    struct Graph {
        int n;
        vector<pair<int, int>> edges;
        vector<int> siz;
        vector<int> cnte;
    };
    Graph compress() {
        Graph g;
        g.n = cnt;
        g.siz.resize(cnt);
        g.cnte.resize(cnt);
        for (int i = 0; i < n; i++) {
            g.siz[bel[i]]++;
            for (auto j : adj[i]) {
                if (bel[i] < bel[j]) {
                    g.edges.emplace_back(bel[i], bel[j]);
                } else if (i < j) {
                    g.cnte[bel[i]]++;
                }
            }
        }
        return g;
    }
};
```

2.3 二分图最大权匹配（MaxAssignment 基于KM）

```cpp
constexpr int inf = 1E7;
template<class T>
struct MaxAssignment {
    public:
        T solve(int nx, int ny, vector<vector<T>> a) {
            assert(0 <= nx && nx <= ny);
            assert(int(a.size()) == nx);
            for (int i = 0; i < nx; ++i) {
                assert(int(a[i].size()) == ny);
                for (auto x : a[i])
                    assert(x >= 0);
            }

            auto update = [&](int x) {
                for (int y = 0; y < ny; ++y) {
                    if (lx[x] + ly[y] - a[x][y] < slack[y]) {
                        slack[y] = lx[x] + ly[y] - a[x][y];
                        slackx[y] = x;
                    }
                }
            };

            costs.resize(nx + 1);
            costs[0] = 0;
            lx.assign(nx, numeric_limits<T>::max());
            ly.assign(ny, 0);
            xy.assign(nx, -1);
            yx.assign(ny, -1);
            slackx.resize(ny);
            for (int cur = 0; cur < nx; ++cur) {
```

```
            queue<int> que;
            visx.assign(nx, false);
            visy.assign(ny, false);
            slack.assign(ny, numeric_limits<T>::max());
            p.assign(nx, -1);

            for (int x = 0; x < nx; ++x) {
                if (xy[x] == -1) {
                    que.push(x);
                    visx[x] = true;
                    update(x);
                }
            }

            int ex, ey;
            bool found = false;
            while (!found) {
                while (!que.empty() && !found) {
                    auto x = que.front();
                    que.pop();
                    for (int y = 0; y < ny; ++y) {
                        if (a[x][y] == lx[x] + ly[y] && !visy[y]) {
                            if (yx[y] == -1) {
                                ex = x;
                                ey = y;
                                found = true;
                                break;
                            }
                            que.push(yx[y]);
                            p[yx[y]] = x;
                            visy[y] = visx[yx[y]] = true;
                            update(yx[y]);
                        }
                    }
                }
                if (found)
                    break;

                T delta = numeric_limits<T>::max();
                for (int y = 0; y < ny; ++y)
                    if (!visy[y])
                        delta = min(delta, slack[y]);
                for (int x = 0; x < nx; ++x)
                    if (visx[x])
                        lx[x] -= delta;
                for (int y = 0; y < ny; ++y) {
                    if (visy[y]) {
                        ly[y] += delta;
                    } else {
                        slack[y] -= delta;
                    }
                }
                for (int y = 0; y < ny; ++y) {
                    if (!visy[y] && slack[y] == 0) {
                        if (yx[y] == -1) {
                            ex = slackx[y];
                            ey = y;
                            found = true;
                            break;
                        }
                        que.push(yx[y]);
                        p[yx[y]] = slackx[y];
                        visy[y] = visx[yx[y]] = true;
                        update(yx[y]);
                        }
                    }
                }

                costs[cur + 1] = costs[cur];
                for (int x = ex, y = ey, ty; x != -1; x = p[x], y = ty) {
                    costs[cur + 1] += a[x][y];
                    if (xy[x] != -1)
                        costs[cur + 1] -= a[x][xy[x]];
                    ty = xy[x];
                    xy[x] = y;
                    yx[y] = x;
                }
            }
            return costs[nx];
        }
        vector<int> assignment() {
            return xy;
        }
        pair<vector<T>, vector<T>> labels() {
            return make_pair(lx, ly);
        }
        vector<T> weights() {
            return costs;
        }
    private:
        vector<T> lx, ly, slack, costs;
        vector<int> xy, yx, p, slackx;
        vector<bool> visx, visy;
};
```

2.4 一般图最大匹配（Graph 带花树算法）【久远】

```
/**   一般图最大匹配（Graph 带花树算法）   **/
struct Graph {
    int n;
    vector<vector<int>> e;
    Graph(int n) : n(n), e(n) {}
    void addEdge(int u, int v) {
        e[u].push_back(v);
        e[v].push_back(u);
    }
    vector<int> findMatching(int m, const auto &init) {
        vector<int> match(n, -1), vis(n), link(n), f(n), dep(n);
        for (auto [x, y] : init) {
            match[x] = y;
            match[y] = x;
        }
        // disjoint set union
        auto find = [&](int u) {
            while (f[u] != u)
                u = f[u] = f[f[u]];
            return u;
        };
        auto lca = [&](int u, int v) {
            u = find(u);
            v = find(v);
            while (u != v) {
                if (dep[u] < dep[v])
                    swap(u, v);
                u = find(link[match[u]]);
            }
```

```
      return u;
    };
    queue<int> que;
    auto blossom = [&](int u, int v, int p) {
      while (find(u) != p) {
        link[u] = v;
        v = match[u];
        if (vis[v] == 0) {
          vis[v] = 1;
          que.push(v);
        }
        f[u] = f[v] = p;
        u = link[v];
      }
    };
    // find an augmenting path starting from u and augment (if exist)
    auto augment = [&](int u) {
      while (!que.empty())
        que.pop();
      iota(f.begin(), f.end(), 0);
      // vis = 0 corresponds to inner vertices, vis = 1 corresponds to outer vertices
      fill(vis.begin(), vis.end(), -1);
      que.push(u);
      vis[u] = 1;
      dep[u] = 0;
      int y = -1;
      while (!que.empty()){
        int u = que.front();
        que.pop();
        if (u >= m) {
          y = u;
        }
        for (auto v : e[u]) {
          if (vis[v] == -1) {
            vis[v] = 0;
            link[v] = u;
            dep[v] = dep[u] + 1;
            // found an augmenting path
            if (match[v] == -1) {
              for (int x = v, y = u, temp; y != -1; x = temp, y = x == -1 ? -1 : link[x]) {
                temp = match[y];
                match[x] = y;
                match[y] = x;
              }
              return;
            }
            vis[match[v]] = 1;
            dep[match[v]] = dep[u] + 2;
            que.push(match[v]);
          } else if (vis[v] == 1 && find(v) != find(u)) {
            // found a blossom
            int p = lca(u, v);
            blossom(u, v, p);
            blossom(v, u, p);
          }
        }
      }
      if (y != -1) {
        for (int x = -1, temp; y != -1; x = temp, y = x == -1 ? -1 : link[x]) {
          temp = match[y];
          if (x != -1) {
```

```
                        match[x] = y;
                    }
                    match[y] = x;
                }
            }
        };
        for (int u = 0; u < m; ++u)
            if (match[u] == -1)
                augment(u);
        return match;
    }
};

/**   一般图最大匹配（Graph 带花树算法）【久远】   **/
struct Graph {
    int n;
    vector<vector<int>> e;
    Graph(int n) : n(n), e(n) {}
    void addEdge(int u, int v) {
        e[u].push_back(v);
        e[v].push_back(u);
    }
    vector<int> findMatching() {
        vector<int> match(n, -1), vis(n), link(n), f(n), dep(n);

        // disjoint set union
        auto find = [&](int u) {
            while (f[u] != u)
                u = f[u] = f[f[u]];
            return u;
        };

        auto lca = [&](int u, int v) {
            u = find(u);
            v = find(v);
            while (u != v) {
                if (dep[u] < dep[v])
                    swap(u, v);
                u = find(link[match[u]]);
            }
            return u;
        };

        queue<int> que;
        auto blossom = [&](int u, int v, int p) {
            while (find(u) != p) {
                link[u] = v;
                v = match[u];
                if (vis[v] == 0) {
                    vis[v] = 1;
                    que.push(v);
                }
                f[u] = f[v] = p;
                u = link[v];
            }
        };

        // find an augmenting path starting from u and augment (if exist)
        auto augment = [&](int u) {

            while (!que.empty())
                que.pop();

            iota(f.begin(), f.end(), 0);

            // vis = 0 corresponds to inner vertices, vis = 1 corresponds to outer vertices
            fill(vis.begin(), vis.end(), -1);

            que.push(u);
            vis[u] = 1;
            dep[u] = 0;

            while (!que.empty()){
                int u = que.front();
                que.pop();
                for (auto v : e[u]) {
                    if (vis[v] == -1) {

                        vis[v] = 0;
                        link[v] = u;
                        dep[v] = dep[u] + 1;

                        // found an augmenting path
                        if (match[v] == -1) {
                            for (int x = v, y = u, temp; y != -1; x = temp, y = x == -1 ? -1 : link[x]) {
                                temp = match[y];
                                match[x] = y;
                                match[y] = x;
                            }
                            return;
                        }

                        vis[match[v]] = 1;
                        dep[match[v]] = dep[u] + 2;
                        que.push(match[v]);

                    } else if (vis[v] == 1 && find(v) != find(u)) {
                        // found a blossom
                        int p = lca(u, v);
                        blossom(u, v, p);
                        blossom(v, u, p);
                    }
                }
            }

        };

        // find a maximal matching greedily (decrease constant)
        auto greedy = [&]() {
            for (int u = 0; u < n; ++u) {
                if (match[u] != -1)
                    continue;
                for (auto v : e[u]) {
                    if (match[v] == -1) {
                        match[u] = v;
                        match[v] = u;
                        break;
                    }
                }
            }
        };

        greedy();

        for (int u = 0; u < n; ++u)
            if (match[u] == -1)
```

```
                    augment(u);

            return match;
        }
};
```

2.5 TwoSat (2-Sat)

```
struct TwoSat {
    int n;
    vector<vector<int>> e;
    vector<bool> ans;
    TwoSat(int n) : n(n), e(2 * n), ans(n) {}
    void addClause(int u, bool f, int v, bool g) {
        e[2 * u + !f].push_back(2 * v + g);
        e[2 * v + !g].push_back(2 * u + f);
    }
    bool satisfiable() {
        vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 * n, -1);
        vector<int> stk;
        int now = 0, cnt = 0;
        function<void(int)> tarjan = [&](int u) {
            stk.push_back(u);
            dfn[u] = low[u] = now++;
            for (auto v : e[u]) {
                if (dfn[v] == -1) {
                    tarjan(v);
                    low[u] = min(low[u], low[v]);
                } else if (id[v] == -1) {
                    low[u] = min(low[u], dfn[v]);
                }
            }
            if (dfn[u] == low[u]) {
                int v;
                do {
                    v = stk.back();
                    stk.pop_back();
                    id[v] = cnt;
                } while (v != u);
                ++cnt;
            }
        };
        for (int i = 0; i < 2 * n; ++i) if (dfn[i] == -1) tarjan(i);
        for (int i = 0; i < n; ++i) {
            if (id[2 * i] == id[2 * i + 1]) return false;
            ans[i] = id[2 * i] > id[2 * i + 1];
        }
        return true;
    }
    vector<bool> answer() { return ans; }
};
```

2.6 最大流（MaxFlow 新版）

```
constexpr int inf = 1E9;
template<class T>
struct MaxFlow {
    struct _Edge {
        int to;
        T cap;
        _Edge(int to, T cap) : to(to), cap(cap) {}
    };

    int n;
    vector<_Edge> e;
    vector<vector<int>> g;
    vector<int> cur, h;

    MaxFlow() {}
    MaxFlow(int n) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        e.clear();
        g.assign(n, {});
        cur.resize(n);
        h.resize(n);
    }

    bool bfs(int s, int t) {
        h.assign(n, -1);
        queue<int> que;
        h[s] = 0;
        que.push(s);
        while (!que.empty()) {
            const int u = que.front();
            que.pop();
            for (int i : g[u]) {
                auto [v, c] = e[i];
                if (c > 0 && h[v] == -1) {
                    h[v] = h[u] + 1;
                    if (v == t) {
                        return true;
                    }
                    que.push(v);
                }
            }
        }
        return false;
    }

    T dfs(int u, int t, T f) {
        if (u == t) {
            return f;
        }
        auto r = f;
        for (int &i = cur[u]; i < int(g[u].size()); ++i) {
            const int j = g[u][i];
            auto [v, c] = e[j];
            if (c > 0 && h[v] == h[u] + 1) {
                auto a = dfs(v, t, min(r, c));
                e[j].cap -= a;
                e[j ^ 1].cap += a;
                r -= a;
                if (r == 0) {
                    return f;
                }
            }
        }
        return f - r;
    }
    void addEdge(int u, int v, T c) {
        g[u].push_back(e.size());
```

```cpp
        e.emplace_back(v, c);
        g[v].push_back(e.size());
        e.emplace_back(u, 0);
    }
    T flow(int s, int t) {
        T ans = 0;
        while (bfs(s, t)) {
            cur.assign(n, 0);
            ans += dfs(s, t, numeric_limits<T>::max());
        }
        return ans;
    }

    vector<bool> minCut() {
        vector<bool> c(n);
        for (int i = 0; i < n; i++) {
            c[i] = (h[i] != -1);
        }
        return c;
    }

    struct Edge {
        int from;
        int to;
        T cap;
        T flow;
    };
    vector<Edge> edges() {
        vector<Edge> a;
        for (int i = 0; i < e.size(); i += 2) {
            Edge x;
            x.from = e[i + 1].to;
            x.to = e[i].to;
            x.cap = e[i].cap + e[i + 1].cap;
            x.flow = e[i + 1].cap;
            a.push_back(x);
        }
        return a;
    }
};
```

2.7 费用流

2.7.1 费用流（MCFGraph 旧版）

```cpp
/**   费用流（MCFGraph 旧版）
 *    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
**/
struct MCFGraph {
    struct Edge {
        int v, c, f;
        Edge(int v, int c, int f) : v(v), c(c), f(f) {}
    };
    const int n;
    vector<Edge> e;
    vector<vector<int>> g;
    vector<i64> h, dis;
    vector<int> pre;
    bool dijkstra(int s, int t) {
        dis.assign(n, numeric_limits<i64>::max());
        pre.assign(n, -1);
        priority_queue<pair<i64, int>, vector<pair<i64, int>>, greater<pair<i64, int>>> que;
        dis[s] = 0;
        que.emplace(0, s);
        while (!que.empty()) {
            i64 d = que.top().first;
            int u = que.top().second;
            que.pop();
            if (dis[u] < d) continue;
            for (int i : g[u]) {
                int v = e[i].v;
                int c = e[i].c;
                int f = e[i].f;
                if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
                    dis[v] = d + h[u] - h[v] + f;
                    pre[v] = i;
                    que.emplace(dis[v], v);
                }
            }
        }
        return dis[t] != numeric_limits<i64>::max();
    }
    MCFGraph(int n) : n(n), g(n) {}
    void addEdge(int u, int v, int c, int f) {
        // if (f < 0) {
            g[u].push_back(e.size());
            e.emplace_back(v, 0, f);
            g[v].push_back(e.size());
            e.emplace_back(u, c, -f);
        // } else {
        //     g[u].push_back(e.size());
        //     e.emplace_back(v, c, f);
        //     g[v].push_back(e.size());
        //     e.emplace_back(u, 0, -f);
        // }
    }
    pair<int, i64> flow(int s, int t) {
        int flow = 0;
        i64 cost = 0;
        h.assign(n, 0);
        while (dijkstra(s, t)) {
            for (int i = 0; i < n; ++i) h[i] += dis[i];
            int aug = numeric_limits<int>::max();
            for (int i = t; i != s; i = e[pre[i] ^ 1].v) aug = min(aug, e[pre[i]].c);
            for (int i = t; i != s; i = e[pre[i] ^ 1].v) {
                e[pre[i]].c -= aug;
                e[pre[i] ^ 1].c += aug;
            }
            flow += aug;
            cost += i64(aug) * h[t];
        }
        return make_pair(flow, cost);
    }
};
```

2.7.2 费用流（MinCostFlow 新版）

```cpp
template<class T>
struct MinCostFlow {
    struct _Edge {
        int to;
        T cap;
        T cost;
        _Edge(int to_, T cap_, T cost_) : to(to_), cap(cap_), cost(cost_) {}
    };
    int n;
    vector<_Edge> e;
    vector<vector<int>> g;
    vector<T> h, dis;
    vector<int> pre;
    bool dijkstra(int s, int t) {
        dis.assign(n, numeric_limits<T>::max());
        pre.assign(n, -1);
        priority_queue<pair<T, int>, vector<pair<T, int>>, greater<pair<T, int>>>
que;
        dis[s] = 0;
        que.emplace(0, s);
        while (!que.empty()) {
            T d = que.top().first;
            int u = que.top().second;
            que.pop();
            if (dis[u] != d) {
                continue;
            }
            for (int i : g[u]) {
                int v = e[i].to;
                T cap = e[i].cap;
                T cost = e[i].cost;
                if (cap > 0 && dis[v] > d + h[u] - h[v] + cost) {
                    dis[v] = d + h[u] - h[v] + cost;
                    pre[v] = i;
                    que.emplace(dis[v], v);
                }
            }
        }
        return dis[t] != numeric_limits<T>::max();
    }
    MinCostFlow() {}
    MinCostFlow(int n_) {
        init(n_);
    }
    void init(int n_) {
        n = n_;
        e.clear();
        g.assign(n, {});
    }
    void addEdge(int u, int v, T cap, T cost) {
        g[u].push_back(e.size());
        e.emplace_back(v, cap, cost);
        g[v].push_back(e.size());
        e.emplace_back(u, 0, -cost);
    }
    pair<T, T> flow(int s, int t) {
        T flow = 0;
        T cost = 0;
        h.assign(n, 0);
        while (dijkstra(s, t)) {
            for (int i = 0; i < n; ++i) {
                h[i] += dis[i];
            }
            T aug = numeric_limits<int>::max();
            for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
                aug = min(aug, e[pre[i]].cap);
            }
            for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
                e[pre[i]].cap -= aug;
                e[pre[i] ^ 1].cap += aug;
            }
            flow += aug;
            cost += aug * h[t];
        }
        return make_pair(flow, cost);
    }
    struct Edge {
        int from;
        int to;
        T cap;
        T cost;
        T flow;
    };
    vector<Edge> edges() {
        vector<Edge> a;
        for (int i = 0; i < e.size(); i += 2) {
            Edge x;
            x.from = e[i + 1].to;
            x.to = e[i].to;
            x.cap = e[i].cap + e[i + 1].cap;
            x.cost = e[i].cost;
            x.flow = e[i + 1].cap;
            a.push_back(x);
        }
        return a;
    }
};
```

2.8 树链剖分（HLD）

```
struct HLD {
    int n;
    vector<int> siz, top, dep, parent, in, out, seq;
    vector<vector<int>> adj;
    int cur;

    HLD() {}
    HLD(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        siz.resize(n);
        top.resize(n);
        dep.resize(n);
        parent.resize(n);
        in.resize(n);
        out.resize(n);
        seq.resize(n);
        cur = 0;
        adj.assign(n, {});
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void work(int root = 0) {
        top[root] = root;
        dep[root] = 0;
        parent[root] = -1;
        dfs1(root);
        dfs2(root);
    }
    void dfs1(int u) {
        if (parent[u] != -1) {
            adj[u].erase(find(adj[u].begin(), adj[u].end(), parent[u]));
        }

        siz[u] = 1;
        for (auto &v : adj[u]) {
            parent[v] = u;
            dep[v] = dep[u] + 1;
            dfs1(v);
            siz[u] += siz[v];
            if (siz[v] > siz[adj[u][0]]) {
                swap(v, adj[u][0]);
            }
        }
    }
    void dfs2(int u) {
        in[u] = cur++;
        seq[in[u]] = u;
        for (auto v : adj[u]) {
            top[v] = v == adj[u][0] ? top[u] : v;
            dfs2(v);
        }
        out[u] = cur;
    }
    int lca(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] > dep[top[v]]) {
                u = parent[top[u]];
            } else {
                v = parent[top[v]];
            }
        }
        return dep[u] < dep[v] ? u : v;
    }

    int dist(int u, int v) {
        return dep[u] + dep[v] - 2 * dep[lca(u, v)];
    }

    int jump(int u, int k) {
        if (dep[u] < k) {
            return -1;
        }

        int d = dep[u] - k;

        while (dep[top[u]] > d) {
            u = parent[top[u]];
        }

        return seq[in[u] - dep[u] + d];
    }

```

```cpp
    bool isAncester(int u, int v) {
        return in[u] <= in[v] && in[v] < out[u];
    }

    int rootedParent(int u, int v) {
        swap(u, v);
        if (u == v) {
            return u;
        }
        if (!isAncester(u, v)) {
            return parent[u];
        }
        auto it = upper_bound(adj[u].begin(), adj[u].end(), v, [&](int x, int y) {
            return in[x] < in[y];
        }) - 1;
        return *it;
    }

    int rootedSize(int u, int v) {
        if (u == v) {
            return n;
        }
        if (!isAncester(v, u)) {
            return siz[v];
        }
        return n - siz[rootedParent(u, v)];
    }

    int rootedLca(int a, int b, int c) {
        return lca(a, b) ^ lca(b, c) ^ lca(c, a);
    }
};
```

/END/

3 数论、几何、多项式

3.1 快速幂

```cpp
/**   快速幂 - 普通版   **/
int power(int a, i64 b, int p) {
    int res = 1;
    for (; b; b /= 2, a = 1LL * a * a % p) {
        if (b % 2) {
            res = 1LL * res * a % p;
        }
    }
    return res;
}
/**   快速幂 - 手写乘法   **/
i64 mul(i64 a, i64 b, i64 p) {
    i64 c = a * b - i64(1.0L * a * b / p) * p;
    c %= p;
    if (c < 0) {
        c += p;
    }
    return c;
}
i64 power(i64 a, i64 b, i64 p) {
    i64 res = 1;
    for (; b; b /= 2, a = mul(a, a, p)) {
        if (b % 2) {
            res = mul(res, a, p);
        }
    }
    return res;
}
```

3.2 基姆拉尔森公式

```cpp
const int d[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

bool isLeap(int y) {
    return y % 400 == 0 || (y % 4 == 0 && y % 100 != 0);
}

int daysInMonth(int y, int m) {
    return d[m - 1] + (isLeap(y) && m == 2);
}

int getDay(int y, int m, int d) {
    int ans = 0;
    for (int i = 1970; i < y; i++) {
        ans += 365 + isLeap(i);
    }
    for (int i = 1; i < m; i++) {
        ans += daysInMonth(y, i);
    }
    ans += d;
    return (ans + 2) % 7 + 1;
}
```

3.3 欧拉筛

```cpp
/**   欧拉筛   **/
vector<int> minp, primes;

void sieve(int n) {
    minp.assign(n + 1, 0);
    primes.clear();

    for (int i = 2; i <= n; i++) {
        if (minp[i] == 0) {
            minp[i] = i;
            primes.push_back(i);
        }

        for (auto p : primes) {
            if (i * p > n) {
                break;
            }
            minp[i * p] = p;
            if (p == minp[i]) {
                break;
            }
        }
    }
}

bool isprime(int n) {
    return minp[n] == n;
}

/**   欧拉筛   **/
void sieve(int n) {
    minp.assign(n + 1, 0);
    phi.assign(n + 1, 0);
    primes.clear();

    for (int i = 2; i <= n; i++) {
        if (minp[i] == 0) {
            minp[i] = i;
            phi[i] = i - 1;
            primes.push_back(i);
        }

        for (auto p : primes) {
            if (i * p > n) {
                break;
            }
            minp[i * p] = p;
            if (p == minp[i]) {
                phi[i * p] = phi[i] * p;
                break;
            }
            phi[i * p] = phi[i] * (p - 1);
        }
    }
    for (int i = 2; i <= n; i++) {
        phi[i] += phi[i - 1];
    }
}
```

3.4 莫比乌斯函数筛（莫比乌斯反演）

```cpp
unordered_map<int, Z> fMu;

vector<int> minp, primes, phi, mu;
vector<i64> sphi;

void sieve(int n) {
    minp.assign(n + 1, 0);
    phi.assign(n + 1, 0);
    sphi.assign(n + 1, 0);
    mu.assign(n + 1, 0);
    primes.clear();
    phi[1] = 1;
    mu[1] = 1;

    for (int i = 2; i <= n; i++) {
        if (minp[i] == 0) {
            minp[i] = i;
            phi[i] = i - 1;
            mu[i] = -1;
            primes.push_back(i);
        }

        for (auto p : primes) {
            if (i * p > n) {
                break;
            }
            minp[i * p] = p;
            if (p == minp[i]) {
                phi[i * p] = phi[i] * p;
                break;
            }
            phi[i * p] = phi[i] * (p - 1);
            mu[i * p] = -mu[i];
        }
    }

    for (int i = 1; i <= n; i++) {
        sphi[i] = sphi[i - 1] + phi[i];
        mu[i] += mu[i - 1];
    }
}

Z sumMu(int n) {
    if (n <= N) {
        return mu[n];
    }
    if (fMu.count(n)) {
        return fMu[n];
    }
    if (n == 0) {
        return 0;
    }
    Z ans = 1;
    for (int l = 2, r; l <= n; l = r + 1) {
        r = n / (n / l);
        ans -= (r - l + 1) * sumMu(n / l);
    }
    return ans;
}
```

3.5 扩展欧几里得（exgcd）

```cpp
/**   扩展欧几里得 ( exgcd )    **/
i64 exgcd(i64 a, i64 b, i64 &x, i64 &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    i64 g = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return g;
}
pair<i64, i64> sol(i64 a, i64 b, i64 m) { // ax + b = 0 (mod m)
    assert(m > 0);
    b *= -1;
    i64 x, y;
    i64 g = exgcd(a, m, x, y);
    if (g < 0) {
        g *= -1;
        x *= -1;
        y *= -1;
    }
    if (b % g != 0) {
        return {-1, -1};
    }
    x = x * (b / g) % (m / g);
    if (x < 0) {
        x += m / g;
    }
    return {x, m / g};
}

/**   扩展欧几里得 ( exgcd )    **/
array<i64, 3> exgcd(i64 a, i64 b) {
    if (!b) {
        return {a, 1, 0};
    }
    auto [g, x, y] = exgcd(b, a % b);
    return {g, y, x - a / b * y};
}
```

3.6 欧拉函数

3.6.1 欧拉函数（求解单个数的欧拉函数）

```cpp
int phi(int n) {
    int res = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0) {
                n /= i;
            }
            res = res / i * (i - 1);
        }
    }
    if (n > 1) {
        res = res / n * (n - 1);
    }
    return res;
}
```

3.6.2 欧拉函数（求解全部数的欧拉函数）

```cpp
constexpr int N = 1E7;
constexpr int P = 1000003;

bool isprime[N + 1];
int phi[N + 1];
vector<int> primes;

fill(isprime + 2, isprime + N + 1, true);
phi[1] = 1;
for (int i = 2; i <= N; i++) {
    if (isprime[i]) {
        primes.push_back(i);
        phi[i] = i - 1;
    }
    for (auto p : primes) {
        if (i * p > N) {
            break;
        }
        isprime[i * p] = false;
        if (i % p == 0) {
            phi[i * p] = phi[i] * p;
            break;
        }
        phi[i * p] = phi[i] * (p - 1);
    }
}
```

3.7 组合数

3.7.1 组合数（小范围预处理，逆元+杨辉三角）

```cpp
constexpr int P = 1000000007;
constexpr int L = 10000;

int fac[L + 1], invfac[L + 1];
int sumbinom[L + 1][7];

int binom(int n, int m) {
    if (n < m || m < 0) {
        return 0;
    }
    return 1LL * fac[n] * invfac[m] % P * invfac[n - m] % P;
}

int power(int a, int b) {
    int res = 1;
    for (; b; b /= 2, a = 1LL * a * a % P) {
        if (b % 2) {
            res = 1LL * res * a % P;
        }
    }
    return res;
}

int main() {
    fac[0] = 1;
    for (int i = 1; i <= L; i++) {
        fac[i] = 1LL * fac[i - 1] * i % P;
    }
    invfac[L] = power(fac[L], P - 2);
    for (int i = L; i; i--) {
        invfac[i - 1] = 1LL * invfac[i] * i % P;
    }

    sumbinom[0][0] = 1;
    for (int i = 1; i <= L; i++) {
        for (int j = 0; j < 7; j++) {
            sumbinom[i][j] = (sumbinom[i - 1][j] + sumbinom[i - 1][(j + 6) % 7]) % P;
        }
    }
}
```

3.7.2 组合数 (Comb, with. ModIntBase)

```cpp
struct Comb {
    int n;
    vector<Z> _fac;
    vector<Z> _invfac;
    vector<Z> _inv;

    Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
    Comb(int n) : Comb() {
        init(n);
    }

    void init(int m) {
        if (m <= n) return;
        _fac.resize(m + 1);
        _invfac.resize(m + 1);
        _inv.resize(m + 1);

        for (int i = n + 1; i <= m; i++) {
            _fac[i] = _fac[i - 1] * i;
        }
        _invfac[m] = _fac[m].inv();
        for (int i = m; i > n; i--) {
            _invfac[i - 1] = _invfac[i] * i;
            _inv[i] = _invfac[i] * _fac[i - 1];
        }
        n = m;
    }

    Z fac(int m) {
        if (m > n) init(2 * m);
        return _fac[m];
    }
    Z invfac(int m) {
        if (m > n) init(2 * m);
        return _invfac[m];
    }
    Z inv(int m) {
        if (m > n) init(2 * m);
        return _inv[m];
    }
    Z binom(int n, int m) {
        if (n < m || m < 0) return 0;
        return fac(n) * invfac(m) * invfac(n - m);
    }
} comb;
```

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

```cpp
i64 mul(i64 a, i64 b, i64 m) {
    return static_cast<__int128>(a) * b % m;
}
i64 power(i64 a, i64 b, i64 m) {
    i64 res = 1 % m;
    for (; b; b >>= 1, a = mul(a, a, m))
        if (b & 1)
            res = mul(res, a, m);
    return res;
}
bool isprime(i64 n) {
    if (n < 2)
        return false;
    static constexpr int A[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    int s = __builtin_ctzll(n - 1);
    i64 d = (n - 1) >> s;
    for (auto a : A) {
        if (a == n)
            return true;
        i64 x = power(a, d, n);
        if (x == 1 || x == n - 1)
            continue;
        bool ok = false;
        for (int i = 0; i < s - 1; ++i) {
            x = mul(x, x, n);
            if (x == n - 1) {
                ok = true;
                break;
            }
        }
        if (!ok)
            return false;
    }
    return true;
}
vector<i64> factorize(i64 n) {
    vector<i64> p;
    function<void(i64)> f = [&](i64 n) {
        if (n <= 10000) {
            for (int i = 2; i * i <= n; ++i)
                for (; n % i == 0; n /= i)
                    p.push_back(i);
            if (n > 1)
                p.push_back(n);
            return;
        }
        if (isprime(n)) {
            p.push_back(n);
            return;
        }
        auto g = [&](i64 x) {
            return (mul(x, x, n) + 1) % n;
        };
        i64 x0 = 2;
        while (true) {
            i64 x = x0;
            i64 y = x0;
            i64 d = 1;
            i64 power = 1, lam = 0;
            i64 v = 1;
            while (d == 1) {
                y = g(y);
                ++lam;
                v = mul(v, abs(x - y), n);
                if (lam % 127 == 0) {
                    d = gcd(v, n);
                    v = 1;
                }
                if (power == lam) {
                    x = y;
                    power *= 2;
                    lam = 0;
                    d = gcd(v, n);
                    v = 1;
                }
            }
            if (d != n) {
                f(d);
                f(n / d);
                return;
            }
            ++x0;
        }
    };
    f(n);
    sort(p.begin(), p.end());
    return p;
}
```

3.9 平面几何

3.9.1 平面几何 (Point)

```
template<class T>
struct Point {
    T x;
    T y;
    Point(const T &x_ = 0, const T &y_ = 0) : x(x_), y(y_) {}

    template<class U>
    operator Point<U>() {
        return Point<U>(U(x), U(y));
    }
    Point &operator+=(const Point &p) & {
        x += p.x;
        y += p.y;
        return *this;
    }
    Point &operator-=(const Point &p) & {
        x -= p.x;
        y -= p.y;
        return *this;
    }
    Point &operator*=(const T &v) & {
        x *= v;
        y *= v;
        return *this;
    }
    Point &operator/=(const T &v) & {
        x /= v;
        y /= v;
        return *this;
    }
```

```cpp
    Point operator-() const {
        return Point(-x, -y);
    }
    friend Point operator+(Point a, const Point &b) {
        return a += b;
    }
    friend Point operator-(Point a, const Point &b) {
        return a -= b;
    }
    friend Point operator*(Point a, const T &b) {
        return a *= b;
    }
    friend Point operator/(Point a, const T &b) {
        return a /= b;
    }
    friend Point operator*(const T &a, Point b) {
        return b *= a;
    }
    friend bool operator==(const Point &a, const Point &b) {
        return a.x == b.x && a.y == b.y;
    }
    friend istream &operator>>(istream &is, Point &p) {
        return is >> p.x >> p.y;
    }
    friend ostream &operator<<(ostream &os, const Point &p) {
        return os << "(" << p.x << ", " << p.y << ")";
    }
};

template<class T>
struct Line {
    Point<T> a;
    Point<T> b;
    Line(const Point<T> &a_ = Point<T>(), const Point<T> &b_ = Point<T>()) : a(a_), b(b_) {}
};

template<class T>
T dot(const Point<T> &a, const Point<T> &b) {
    return a.x * b.x + a.y * b.y;
}

template<class T>
T cross(const Point<T> &a, const Point<T> &b) {
    return a.x * b.y - a.y * b.x;
}

template<class T>
T square(const Point<T> &p) {
    return dot(p, p);
}

template<class T>
double length(const Point<T> &p) {
    return sqrt(square(p));
}

template<class T>
double length(const Line<T> &l) {
    return length(l.a - l.b);
}

template<class T>
Point<T> normalize(const Point<T> &p) {
    return p / length(p);
}

template<class T>
bool parallel(const Line<T> &l1, const Line<T> &l2) {
    return cross(l1.b - l1.a, l2.b - l2.a) == 0;
}

template<class T>
double distance(const Point<T> &a, const Point<T> &b) {
    return length(a - b);
}

template<class T>
double distancePL(const Point<T> &p, const Line<T> &l) {
    return abs(cross(l.a - l.b, l.a - p)) / length(l);
}

template<class T>
double distancePS(const Point<T> &p, const Line<T> &l) {
    if (dot(p - l.a, l.b - l.a) < 0) {
        return distance(p, l.a);
    }
    if (dot(p - l.b, l.a - l.b) < 0) {
        return distance(p, l.b);
    }
    return distancePL(p, l);
}

template<class T>
Point<T> rotate(const Point<T> &a) {
    return Point(-a.y, a.x);
}

template<class T>
int sgn(const Point<T> &a) {
    return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
}

template<class T>
bool pointOnLineLeft(const Point<T> &p, const Line<T> &l) {
    return cross(l.b - l.a, p - l.a) > 0;
}

template<class T>
Point<T> lineIntersection(const Line<T> &l1, const Line<T> &l2) {
    return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - 
l2.a, l1.a - l1.b));
}

template<class T>
bool pointOnSegment(const Point<T> &p, const Line<T> &l) {
    return cross(p - l.a, l.b - l.a) == 0 && min(l.a.x, l.b.x) <= p.x && p.x <= 
max(l.a.x, l.b.x)
        && min(l.a.y, l.b.y) <= p.y && p.y <= max(l.a.y, l.b.y);
}

template<class T>
bool pointInPolygon(const Point<T> &a, const vector<Point<T>> &p) {
    int n = p.size();
    for (int i = 0; i < n; i++) {
        if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
            return true;
        }
    }

    int t = 0;
    for (int i = 0; i < n; i++) {
        auto u = p[i];
        auto v = p[(i + 1) % n];
        if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
            t ^= 1;
        }
        if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
            t ^= 1;
        }
    }

    return t == 1;
}

// 0 : not intersect
// 1 : strictly intersect
// 2 : overlap
// 3 : intersect at endpoint
template<class T>
tuple<int, Point<T>, Point<T>> segmentIntersection(const Line<T> &l1, const Line<T>
&l2) {
    if (max(l1.a.x, l1.b.x) < min(l2.a.x, l2.b.x)) {
        return {0, Point<T>(), Point<T>()};
    }
    if (min(l1.a.x, l1.b.x) > max(l2.a.x, l2.b.x)) {
        return {0, Point<T>(), Point<T>()};
    }
    if (max(l1.a.y, l1.b.y) < min(l2.a.y, l2.b.y)) {
        return {0, Point<T>(), Point<T>()};
    }
    if (min(l1.a.y, l1.b.y) > max(l2.a.y, l2.b.y)) {
        return {0, Point<T>(), Point<T>()};
    }
    if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
        if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
            return {0, Point<T>(), Point<T>()};
        } else {
            auto maxx1 = max(l1.a.x, l1.b.x);
            auto minx1 = min(l1.a.x, l1.b.x);
            auto maxy1 = max(l1.a.y, l1.b.y);
            auto miny1 = min(l1.a.y, l1.b.y);
            auto maxx2 = max(l2.a.x, l2.b.x);
            auto minx2 = min(l2.a.x, l2.b.x);
            auto maxy2 = max(l2.a.y, l2.b.y);
            auto miny2 = min(l2.a.y, l2.b.y);
            Point<T> p1(max(minx1, minx2), max(miny1, miny2));
            Point<T> p2(min(maxx1, maxx2), min(maxy1, maxy2));
            if (!pointOnSegment(p1, l1)) {
                swap(p1.y, p2.y);
            }
            if (p1 == p2) {
                return {3, p1, p2};
            } else {
                return {2, p1, p2};
            }
        }
    }
    auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
    auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
    auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
    auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);

    if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0) ||
(cp3 < 0 && cp4 < 0)) {
        return {0, Point<T>(), Point<T>()};
    }

    Point p = lineIntersection(l1, l2);
    if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
        return {1, p, p};
    } else {
        return {3, p, p};
    }
}

template<class T>
double distanceSS(const Line<T> &l1, const Line<T> &l2) {
    if (get<0>(segmentIntersection(l1, l2)) != 0) {
        return 0.0;
    }
    return min({distancePS(l1.a, l2), distancePS(l1.b, l2), distancePS(l2.a, l1),
distancePS(l2.b, l1)});
}

template<class T>
bool segmentInPolygon(const Line<T> &l, const vector<Point<T>> &p) {
    int n = p.size();
    if (!pointInPolygon(l.a, p)) {
        return false;
    }
    if (!pointInPolygon(l.b, p)) {
        return false;
    }
    for (int i = 0; i < n; i++) {
        auto u = p[i];
        auto v = p[(i + 1) % n];
        auto w = p[(i + 2) % n];
        auto [t, p1, p2] = segmentIntersection(l, Line(u, v));

        if (t == 1) {
            return false;
        }
        if (t == 0) {
            continue;
        }
        if (t == 2) {
            if (pointOnSegment(v, l) && v != l.a && v != l.b) {
                if (cross(v - u, w - v) > 0) {
                    return false;
                }
            }
        } else {
            if (p1 != u && p1 != v) {
                if (pointOnLineLeft(l.a, Line(v, u))
                    || pointOnLineLeft(l.b, Line(v, u))) {
                    return false;
                }
            } else if (p1 == v) {
                if (l.a == v) {
                    if (pointOnLineLeft(u, l)) {
                        if (pointOnLineLeft(w, l)
                            && pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    } else {
                        if (pointOnLineLeft(w, l)
                            || pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    }
                } else if (l.b == v) {
                    if (pointOnLineLeft(u, Line(l.b, l.a))) {
                        if (pointOnLineLeft(w, Line(l.b, l.a))
                            && pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    } else {
                        if (pointOnLineLeft(w, Line(l.b, l.a))
                            || pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    }
                } else {
                    if (pointOnLineLeft(u, l)) {
                        if (pointOnLineLeft(w, Line(l.b, l.a))
                            || pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    } else {
                        if (pointOnLineLeft(w, l)
                            || pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    }
                }
            }
        }
    }
    return true;
}

template<class T>
vector<Point<T>> hp(vector<Line<T>> lines) {
    sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
        auto d1 = l1.b - l1.a;
        auto d2 = l2.b - l2.a;

        if (sgn(d1) != sgn(d2)) {
            return sgn(d1) == 1;
        }

        return cross(d1, d2) > 0;
    });

    deque<Line<T>> ls;
    deque<Point<T>> ps;
    for (auto l : lines) {
        if (ls.empty()) {
            ls.push_back(l);
            continue;
        }

        while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
            ps.pop_back();
            ls.pop_back();
        }

        while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
            ps.pop_front();
```

```cpp
                ls.pop_front();
            }

            if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
                if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {

                    if (!pointOnLineLeft(ls.back().a, l)) {
                        assert(ls.size() == 1);
                        ls[0] = l;
                    }
                    continue;
                }
                return {};
            }

            ps.push_back(lineIntersection(ls.back(), l));
            ls.push_back(l);
        }

        while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
            ps.pop_back();
            ls.pop_back();
        }
        if (ls.size() <= 2) {
            return {};
        }
        ps.push_back(lineIntersection(ls[0], ls.back()));

        return vector(ps.begin(), ps.end());
}

using real = long double;
using P = Point<real>;

constexpr real eps = 0;
```

3.9.2 平面几何（with. complex）

```cpp
using Point = complex<long double>;

#define x real
#define y imag

long double dot(const Point &a, const Point &b) {
    return (conj(a) * b).x();
}

long double cross(const Point &a, const Point &b) {
    return (conj(a) * b).y();
}

long double length(const Point &a) {
    return sqrt(dot(a, a));
}

long double dist(const Point &a, const Point &b) {
    return length(a - b);
}

long double get(const Point &a, const Point &b, const Point &c, const Point &d) {
    auto e = a + (b - a) * cross(c - a, d - a) / cross(b - a, d - c);
    return dist(d, e);
}
```

3.10 立体几何 (Point)

```cpp
using i64 = long long;
using real = double;

struct Point {
    real x = 0;
    real y = 0;
    real z = 0;
};

Point operator+(const Point &a, const Point &b) {
    return {a.x + b.x, a.y + b.y, a.z + b.z};
}

Point operator-(const Point &a, const Point &b) {
    return {a.x - b.x, a.y - b.y, a.z - b.z};
}

Point operator*(const Point &a, real b) {
    return {a.x * b, a.y * b, a.z * b};
}

Point operator/(const Point &a, real b) {
    return {a.x / b, a.y / b, a.z / b};
}

real length(const Point &a) {
    return hypot(a.x, a.y, a.z);
}

Point normalize(const Point &a) {
    real l = length(a);
    return {a.x / l, a.y / l, a.z / l};
}

real getAng(real a, real b, real c) {
    return acos((a * a + b * b - c * c) / 2 / a / b);
}

ostream &operator<<(ostream &os, const Point &a) {
    return os << "(" << a.x << ", " << a.y << ", " << a.z << ")";
}

real dot(const Point &a, const Point &b) {
    return a.x * b.x + a.y * b.y + a.z * b.z;
}

Point cross(const Point &a, const Point &b) {
    return {
        a.y * b.z - a.z * b.y,
        a.z * b.x - a.x * b.z,
        a.x * b.y - a.y * b.x
    };
}
```

3.11 静态凸包

3.11.1 静态凸包（with. Point，新版）

```cpp
struct Point {
    i64 x;
    i64 y;
    Point() : x{0}, y{0} {}
    Point(i64 x_, i64 y_) : x{x_}, y{y_} {}
};

i64 dot(Point a, Point b) {
    return a.x * b.x + a.y * b.y;
}

i64 cross(Point a, Point b) {
    return a.x * b.y - a.y * b.x;
}

Point operator+(Point a, Point b) {
    return Point(a.x + b.x, a.y + b.y);
}

Point operator-(Point a, Point b) {
    return Point(a.x - b.x, a.y - b.y);
}

auto getHull(vector<Point> p) {
    sort(p.begin(), p.end(),
        [&](auto a, auto b) {
            return a.x < b.x || (a.x == b.x && a.y < b.y);
        });

    vector<Point> hi, lo;
    for (auto p : p) {
        while (hi.size() > 1 && cross(hi.back() - hi[hi.size() - 2], p - hi.back()) >= 0) {
            hi.pop_back();
        }
        while (!hi.empty() && hi.back().x == p.x) {
            hi.pop_back();
        }
        hi.push_back(p);
        while (lo.size() > 1 && cross(lo.back() - lo[lo.size() - 2], p - lo.back()) <= 0) {
            lo.pop_back();
        }
        if (lo.empty() || lo.back().x < p.x) {
            lo.push_back(p);
        }
    }
    return make_pair(hi, lo);
}

const double inf = INFINITY;
```

3.11.2 静态凸包 (with. complex)

```cpp
using Point = complex<i64>;

#define x real
#define y imag

auto dot(const Point &a, const Point &b) {
    return (conj(a) * b).x();
}

auto cross(const Point &a, const Point &b) {
    return (conj(a) * b).y();
}

auto rot(const Point &p) {
    return Point(-p.y(), p.x());
}

auto complexHull(vector<Point> a) {
    sort(a.begin(), a.end(), [&](auto a, auto b) {
        if (a.x() != b.x()) {
            return a.x() < b.x();
        } else {
            return a.y() < b.y();
        }
    });

    vector<Point> l, h;

    for (auto p : a) {
        while (l.size() > 1 && cross(l.back() - l[l.size() - 2], p - l.back()) <= 0) {
            l.pop_back();
        }

        while (h.size() > 1 && cross(h.back() - h[h.size() - 2], p - h.back()) >= 0) {
            h.pop_back();
        }

        l.push_back(p);
        h.push_back(p);
    }

    reverse(h.begin(), h.end());

    h.insert(h.end(), l.begin() + 1, l.end() - 1);

    return h;
}

int sgn(Point p) {
    if (p.y() > 0 || (p.y() == 0 && p.x() < 0)) {
        return 0;
    } else {
        return 1;
    }
}
```

3.12 多项式

3.12.1 多项式（Poly，旧版）

```cpp
constexpr int C = 1024;
constexpr int P = 998244353;
vector<int> rev, roots{0, 1};
int power(int a, int b) {
    int res = 1;
    for (; b; b >>= 1, a = 1ll * a * a % P)
        if (b & 1)
            res = 1ll * res * a % P;
    return res;
}
void dft(vector<int> &a) {
    int n = a.size();
    if (int(rev.size()) != n) {
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; ++i)
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
    }
    for (int i = 0; i < n; ++i)
        if (rev[i] < i)
            swap(a[i], a[rev[i]]);
    if (int(roots.size()) < n) {
        int k = __builtin_ctz(roots.size());
        roots.resize(n);
        while ((1 << k) < n) {
            int e = power(3, (P - 1) >> (k + 1));
            for (int i = 1 << (k - 1); i < (1 << k); ++i) {
                roots[2 * i] = roots[i];
                roots[2 * i + 1] = 1ll * roots[i] * e % P;
            }
            ++k;
        }
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; ++j) {
                int u = a[i + j];
                int v = 1ll * a[i + j + k] * roots[k + j] % P;
                int x = u + v;
                if (x >= P)
                    x -= P;
                a[i + j] = x;
                x = u - v;
                if (x < 0)
                    x += P;
                a[i + j + k] = x;
            }
        }
    }
}
void idft(vector<int> &a) {
    int n = a.size();
    reverse(a.begin() + 1, a.end());
    dft(a);
    int inv = power(n, P - 2);
    for (int i = 0; i < n; ++i)
        a[i] = 1ll * a[i] * inv % P;
}
struct Poly {
    vector<int> a;
    Poly() {}
    Poly(int a0) {
        if (a0)
            a = {a0};
    }
    Poly(const vector<int> &a1) : a(a1) {
        while (!a.empty() && !a.back())
            a.pop_back();
    }
    int size() const {
        return a.size();
    }
    int operator[](int idx) const {
        if (idx < 0 || idx >= size())
            return 0;
        return a[idx];
    }
    Poly mulxk(int k) const {
        auto b = a;
        b.insert(b.begin(), k, 0);
        return Poly(b);
    }
    Poly modxk(int k) const {
        k = min(k, size());
        return Poly(vector<int>(a.begin(), a.begin() + k));
    }
    Poly divxk(int k) const {
        if (size() <= k)
            return Poly();
        return Poly(vector<int>(a.begin() + k, a.end()));
    }
    friend Poly operator+(const Poly a, const Poly &b) {
        vector<int> res(max(a.size(), b.size()));
        for (int i = 0; i < int(res.size()); ++i) {
            res[i] = a[i] + b[i];
            if (res[i] >= P)
                res[i] -= P;
        }
        return Poly(res);
    }
    friend Poly operator-(const Poly a, const Poly &b) {
        vector<int> res(max(a.size(), b.size()));
        for (int i = 0; i < int(res.size()); ++i) {
            res[i] = a[i] - b[i];
            if (res[i] < 0)
                res[i] += P;
        }
        return Poly(res);
    }
    friend Poly operator*(Poly a, Poly b) {
        int sz = 1, tot = a.size() + b.size() - 1;
        while (sz < tot)
            sz *= 2;
        a.a.resize(sz);
        b.a.resize(sz);
        dft(a.a);
        dft(b.a);
        for (int i = 0; i < sz; ++i)
            a.a[i] = 1ll * a[i] * b[i] % P;
        idft(a.a);
        return Poly(a.a);
    }
    Poly &operator+=(Poly b) {
        return (*this) = (*this) + b;
    }
    Poly &operator-=(Poly b) {
        return (*this) = (*this) - b;
    }
    Poly &operator*=(Poly b) {
        return (*this) = (*this) * b;
    }
    Poly deriv() const {
        if (a.empty())
            return Poly();
        vector<int> res(size() - 1);
        for (int i = 0; i < size() - 1; ++i)
            res[i] = 1ll * (i + 1) * a[i + 1] % P;
        return Poly(res);
    }
    Poly integr() const {
        if (a.empty())
            return Poly();
        vector<int> res(size() + 1);
        for (int i = 0; i < size(); ++i)
            res[i + 1] = 1ll * a[i] * power(i + 1, P - 2) % P;
        return Poly(res);
    }
    Poly inv(int m) const {
        Poly x(power(a[0], P - 2));
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (2 - modxk(k) * x)).modxk(k);
        }
        return x.modxk(m);
    }
    Poly log(int m) const {
        return (deriv() * inv(m)).integr().modxk(m);
    }
    Poly exp(int m) const {
        Poly x(1);
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (1 - x.log(k) + modxk(k))).modxk(k);
        }
        return x.modxk(m);
    }
    Poly sqrt(int m) const {
        Poly x(1);
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((P + 1) / 2);
        }
        return x.modxk(m);
    }
    Poly mulT(Poly b) const {
        if (b.size() == 0)
            return Poly();
        int n = b.size();
        reverse(b.a.begin(), b.a.end());
        return ((*this) * b).divxk(n - 1);
    }
    vector<int> eval(vector<int> x) const {
        if (size() == 0)
            return vector<int>(x.size(), 0);
```

```cpp
        const int n = max(int(x.size()), size());
        vector<Poly> q(4 * n);
        vector<int> ans(x.size());
        x.resize(n);
        function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                q[p] = vector<int>{1, (P - x[l]) % P};
            } else {
                int m = (l + r) / 2;
                build(2 * p, l, m);
                build(2 * p + 1, m, r);
                q[p] = q[2 * p] * q[2 * p + 1];
            }
        };
        build(1, 0, n);
        function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r,
const Poly &num) {
            if (r - l == 1) {
                if (l < int(ans.size()))
                    ans[l] = num[0];
            } else {
                int m = (l + r) / 2;
                work(2 * p, l, m, num.mulT(q[2 * p + 1]).modxk(m - l));
                work(2 * p + 1, m, r, num.mulT(q[2 * p]).modxk(r - m));
            }
        };
        work(1, 0, n, mulT(q[1].inv(n)));
        return ans;
    }
};
using i64 = long long;
void dft(vector<vector<int>> &a) {
    int n = a.size();
    for (auto &v : a) {
        dft(v);
    }
    for (int i = 0; i < int(a[0].size()); i++) {
        vector<int> v(n);
        for (int j = 0; j < n; j++) {
            v[j] = a[j][i];
        }
        dft(v);
        for (int j = 0; j < n; j++) {
            a[j][i] = v[j];
        }
    }
}
void idft(vector<vector<int>> &a) {
    int n = a.size();
    for (auto &v : a) {
        idft(v);
    }
    for (int i = 0; i < int(a[0].size()); i++) {
        vector<int> v(n);
        for (int j = 0; j < n; j++) {
            v[j] = a[j][i];
        }
        idft(v);
        for (int j = 0; j < n; j++) {
            a[j][i] = v[j];
        }
    }
}
auto inv(const vector<vector<int>> &a) {
    int m = 1;
    vector g(1, vector{Poly(a[0]).inv(C).a});
    while (m < C) {
        vector a0(4 * m, vector<int>(4 * C));
        for (int i = 0; i < 2 * m; i++) {
            for (int j = 0; j < C; j++) {
                a0[i][j] = a[i][j];
            }
        }
        dft(a0);
        g.resize(4 * m);
        for (auto &v : g) {
            v.resize(4 * C);
        }
        dft(g);
        for (int i = 0; i < 4 * m; i++) {
            for (int j = 0; j < 4 * C; j++) {
                g[i][j] = i64(g[i][j]) * (2 + i64(P - a0[i][j]) * g[i][j] % P) % P;
            }
        }
        idft(g);
        m *= 2;
        g.resize(m);
        for (auto &v : g) {
            v.resize(C);
        }
    }
    return g;
}
```

3.12.2　多项式 (Poly, with. MInt & MLong)

```cpp
vector<int> rev;
template<int P>
vector<MInt<P>> roots{0, 1};

template<int P>
constexpr MInt<P> findPrimitiveRoot() {
    MInt<P> i = 2;
    int k = __builtin_ctz(P - 1);
    while (true) {
        if (power(i, (P - 1) / 2) != 1) {
            break;
        }
        i += 1;
    }
    return power(i, (P - 1) >> k);
}

template<int P>
constexpr MInt<P> primitiveRoot = findPrimitiveRoot<P>();

template<>
constexpr MInt<998244353> primitiveRoot<998244353> {31};

template<int P>
constexpr void dft(vector<MInt<P>> &a) {
    int n = a.size();

    if (int(rev.size()) != n) {
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; i++) {
                rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
            }
        }

    for (int i = 0; i < n; i++) {
        if (rev[i] < i) {
            swap(a[i], a[rev[i]]);
        }
    }
    if (roots<P>.size() < n) {
        int k = __builtin_ctz(roots<P>.size());
        roots<P>.resize(n);
        while ((1 << k) < n) {
            auto e = power(primitiveRoot<P>, 1 << (__builtin_ctz(P - 1) - k - 1));
            for (int i = 1 << (k - 1); i < (1 << k); i++) {
                roots<P>[2 * i] = roots<P>[i];
                roots<P>[2 * i + 1] = roots<P>[i] * e;
            }
            k++;
        }
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j++) {
                MInt<P> u = a[i + j];
                MInt<P> v = a[i + j + k] * roots<P>[k + j];
                a[i + j] = u + v;
                a[i + j + k] = u - v;
            }
        }
    }
}

template<int P>
constexpr void idft(vector<MInt<P>> &a) {
    int n = a.size();
    reverse(a.begin() + 1, a.end());
    dft(a);
    MInt<P> inv = (1 - P) / n;
    for (int i = 0; i < n; i++) {
        a[i] *= inv;
    }
}

template<int P = 998244353>
struct Poly : public vector<MInt<P>> {
    using Value = MInt<P>;

    Poly() : vector<Value>() {}
    explicit constexpr Poly(int n) : vector<Value>(n) {}

    explicit constexpr Poly(const vector<Value> &a) : vector<Value>(a) {}
    constexpr Poly(const initializer_list<Value> &a) : vector<Value>(a) {}

    template<class InputIt, class = _RequireInputIter<InputIt>>
    explicit constexpr Poly(InputIt first, InputIt last) : vector<Value>(first,
last) {}

    template<class F>
    explicit constexpr Poly(int n, F f) : vector<Value>(n) {
        for (int i = 0; i < n; i++) {
            (*this)[i] = f(i);
        }
    }

    constexpr Poly shift(int k) const {
        if (k >= 0) {
            auto b = *this;
            b.insert(b.begin(), k, 0);
            return b;
        } else if (this->size() <= -k) {
            return Poly();
        } else {
            return Poly(this->begin() + (-k), this->end());
        }
    }
    constexpr Poly trunc(int k) const {
        Poly f = *this;
        f.resize(k);
        return f;
    }
    constexpr friend Poly operator+(const Poly &a, const Poly &b) {
        Poly res(max(a.size(), b.size()));
        for (int i = 0; i < a.size(); i++) {
            res[i] += a[i];
        }
        for (int i = 0; i < b.size(); i++) {
            res[i] += b[i];
        }
        return res;
    }
    constexpr friend Poly operator-(const Poly &a, const Poly &b) {
        Poly res(max(a.size(), b.size()));
        for (int i = 0; i < a.size(); i++) {
            res[i] += a[i];
        }
        for (int i = 0; i < b.size(); i++) {
            res[i] -= b[i];
        }
        return res;
    }
    constexpr friend Poly operator-(const Poly &a) {
        vector<Value> res(a.size());
        for (int i = 0; i < int(res.size()); i++) {
            res[i] = -a[i];
        }
        return Poly(res);
    }
    constexpr friend Poly operator*(Poly a, Poly b) {
        if (a.size() == 0 || b.size() == 0) {
            return Poly();
        }
        if (a.size() < b.size()) {
            swap(a, b);
        }
        int n = 1, tot = a.size() + b.size() - 1;
        while (n < tot) {
            n *= 2;
        }
        if (((P - 1) & (n - 1)) != 0 || b.size() < 128) {
            Poly c(a.size() + b.size() - 1);
            for (int i = 0; i < a.size(); i++) {
                for (int j = 0; j < b.size(); j++) {
                    c[i + j] += a[i] * b[j];
                }
            }
            return c;
        }
        a.resize(n);
        b.resize(n);
        dft(a);
        dft(b);
        for (int i = 0; i < n; ++i) {
            a[i] *= b[i];
        }
        idft(a);
        a.resize(tot);
        return a;
    }
    constexpr friend Poly operator*(Value a, Poly b) {
        for (int i = 0; i < int(b.size()); i++) {
            b[i] *= a;
        }
        return b;
    }
    constexpr friend Poly operator*(Poly a, Value b) {
        for (int i = 0; i < int(a.size()); i++) {
            a[i] *= b;
        }
        return a;
    }
    constexpr friend Poly operator/(Poly a, Value b) {
        for (int i = 0; i < int(a.size()); i++) {
            a[i] /= b;
        }
        return a;
    }
    constexpr Poly &operator+=(Poly b) {
        return (*this) = (*this) + b;
    }
    constexpr Poly &operator-=(Poly b) {
        return (*this) = (*this) - b;
    }
    constexpr Poly &operator*=(Poly b) {
        return (*this) = (*this) * b;
    }
    constexpr Poly &operator*=(Value b) {
        return (*this) = (*this) * b;
    }
    constexpr Poly &operator/=(Value b) {
        return (*this) = (*this) / b;
    }
    constexpr Poly deriv() const {
        if (this->empty()) {
            return Poly();
        }
        Poly res(this->size() - 1);
        for (int i = 0; i < this->size() - 1; ++i) {
            res[i] = (i + 1) * (*this)[i + 1];
        }
        return res;
    }
    constexpr Poly integr() const {
        Poly res(this->size() + 1);
        for (int i = 0; i < this->size(); ++i) {
            res[i + 1] = (*this)[i] / (i + 1);
        }
        return res;
    }
    constexpr Poly inv(int m) const {
        Poly x{(*this)[0].inv()};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (Poly{2} - trunc(k) * x)).trunc(k);
        }
        return x.trunc(m);
    }
    constexpr Poly log(int m) const {
        return (deriv() * inv(m)).integr().trunc(m);
    }
    constexpr Poly exp(int m) const {
        Poly x{1};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (Poly{1} - x.log(k) + trunc(k))).trunc(k);
        }
        return x.trunc(m);
    }
    constexpr Poly pow(int k, int m) const {
        int i = 0;
        while (i < this->size() && (*this)[i] == 0) {
            i++;
        }
        if (i == this->size() || 1LL * i * k >= m) {
            return Poly(m);
        }
        Value v = (*this)[i];
        auto f = shift(-i) * v.inv();
        return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * power(v, k);
    }
    constexpr Poly sqrt(int m) const {
        Poly x{1};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x + (trunc(k) * x.inv(k)).trunc(k)) * CInv<2, P>;
        }
        return x.trunc(m);
    }
    constexpr Poly mulT(Poly b) const {
        if (b.size() == 0) {
            return Poly();
        }
        int n = b.size();
        reverse(b.begin(), b.end());
        return ((*this) * b).shift(-(n - 1));
    }
    constexpr vector<Value> eval(vector<Value> x) const {
        if (this->size() == 0) {
            return vector<Value>(x.size(), 0);
        }
        const int n = max(x.size(), this->size());
        vector<Poly> q(4 * n);
        vector<Value> ans(x.size());
        x.resize(n);
        function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                q[p] = Poly{1, -x[l]};
            } else {
                int m = (l + r) / 2;
                build(2 * p, l, m);
                build(2 * p + 1, m, r);
                q[p] = q[2 * p] * q[2 * p + 1];
            }
```

```cpp
        };
        build(1, 0, n);
        function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r, const Poly &num) {
            if (r - l == 1) {
                if (l < int(ans.size())) {
                    ans[l] = num[0];
                }
            } else {
                int m = (l + r) / 2;
                work(2 * p, l, m, num.mulT(q[2 * p + 1]).trunc(m - l));
                work(2 * p + 1, m, r, num.mulT(q[2 * p]).trunc(r - m));
            }
        };
        work(1, 0, n, mulT(q[1].inv(n)));
        return ans;
    }
};

template<int P = 998244353>
Poly<P> berlekampMassey(const Poly<P> &s) {
    Poly<P> c;
    Poly<P> oldC;
    int f = -1;
    for (int i = 0; i < s.size(); i++) {
        auto delta = s[i];
        for (int j = 1; j <= c.size(); j++) {
            delta -= c[j - 1] * s[i - j];
        }
        if (delta == 0) {
            continue;
        }
        if (f == -1) {
            c.resize(i + 1);
            f = i;
        } else {
            auto d = oldC;
            d *= -1;
            d.insert(d.begin(), 1);
            MInt<P> df1 = 0;
            for (int j = 1; j <= d.size(); j++) {
                df1 += d[j - 1] * s[f + 1 - j];
            }
            assert(df1 != 0);
            auto coef = delta / df1;
            d *= coef;
            Poly<P> zeros(i - f - 1);
            zeros.insert(zeros.end(), d.begin(), d.end());
            d = zeros;
            auto temp = c;
            c += d;
            if (i - temp.size() > f - oldC.size()) {
                oldC = temp;
                f = i;
            }
        }
    }
    c *= -1;
    c.insert(c.begin(), 1);
    return c;
}

template<int P = 998244353>
MInt<P> linearRecurrence(Poly<P> p, Poly<P> q, i64 n) {
    int m = q.size() - 1;
    while (n > 0) {
        auto newq = q;
        for (int i = 1; i <= m; i += 2) {
            newq[i] *= -1;
        }
        auto newp = p * newq;
        newq = q * newq;
        for (int i = 0; i < m; i++) {
            p[i] = newp[i * 2 + n % 2];
        }
        for (int i = 0; i <= m; i++) {
            q[i] = newq[i * 2];
        }
        n /= 2;
    }
    return p[0] / q[0];
}

struct Comb {
    int n;
    vector<Z> _fac;
    vector<Z> _invfac;
    vector<Z> _inv;

    Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
    Comb(int n) : Comb() {
        init(n);
    }

    void init(int m) {
        m = min(m, Z::getMod() - 1);
        if (m <= n) return;
        _fac.resize(m + 1);
        _invfac.resize(m + 1);
        _inv.resize(m + 1);

        for (int i = n + 1; i <= m; i++) {
            _fac[i] = _fac[i - 1] * i;
        }
        _invfac[m] = _fac[m].inv();
        for (int i = m; i > n; i--) {
            _invfac[i - 1] = _invfac[i] * i;
            _inv[i] = _invfac[i] * _fac[i - 1];
        }
        n = m;
    }

    Z fac(int m) {
        if (m > n) init(2 * m);
        return _fac[m];
    }
    Z invfac(int m) {
        if (m > n) init(2 * m);
        return _invfac[m];
    }
    Z inv(int m) {
        if (m > n) init(2 * m);
        return _inv[m];
    }
    Z binom(int n, int m) {
        if (n < m || m < 0) return 0;
        return fac(n) * invfac(m) * invfac(n - m);
    }
```

```
} comb;

Poly<P> get(int n, int m) {
    if (m == 0) {
        return Poly(n + 1);
    }
    if (m % 2 == 1) {
        auto f = get(n, m - 1);
        Z p = 1;
        for (int i = 0; i <= n; i++) {
            f[n - i] += comb.binom(n, i) * p;
            p *= m;
        }
        return f;
    }
    auto f = get(n, m / 2);
    auto fm = f;
    for (int i = 0; i <= n; i++) {
        fm[i] *= comb.fac(i);
    }
    Poly pw(n + 1);
    pw[0] = 1;
    for (int i = 1; i <= n; i++) {
        pw[i] = pw[i - 1] * (m / 2);
    }
    for (int i = 0; i <= n; i++) {
        pw[i] *= comb.invfac(i);
    }
    fm = fm.mulT(pw);
    for (int i = 0; i <= n; i++) {
        fm[i] *= comb.invfac(i);
    }
    return f + fm;
}
```

3.12.3 多项式乘法

```
constexpr int P = 998244353;

int power(int a, int b) {
    int res = 1;
    for (; b; b /= 2, a = 1LL * a * a % P) {
        if (b % 2) {
            res = 1LL * res * a % P;
        }
    }
    return res;
}

vector<int> rev, roots {0, 1};

void dft(vector<int> &a) {
    int n = a.size();
    if (int(rev.size()) != n) {
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; i++) {
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
        }
    }
    for (int i = 0; i < n; i++) {
        if (rev[i] < i) {
            swap(a[i], a[rev[i]]);
```

```cpp
            }
        }
        if (roots.size() < n) {
            int k = __builtin_ctz(roots.size());
            roots.resize(n);
            while ((1 << k) < n) {
                int e = power(31, 1 << (__builtin_ctz(P - 1) - k - 1));
                for (int i = 1 << (k - 1); i < (1 << k); i++) {
                    roots[2 * i] = roots[i];
                    roots[2 * i + 1] = 1LL * roots[i] * e % P;
                }
                k++;
            }
        }

        for (int k = 1; k < n; k *= 2) {
            for (int i = 0; i < n; i += 2 * k) {
                for (int j = 0; j < k; j++) {
                    int u = a[i + j];
                    int v = 1LL * a[i + j + k] * roots[k + j] % P;
                    a[i + j] = (u + v) % P;
                    a[i + j + k] = (u - v) % P;
                }
            }
        }
}

void idft(vector<int> &a) {
    int n = a.size();
    reverse(a.begin() + 1, a.end());
    dft(a);
    int inv = (1 - P) / n;
    for (int i = 0; i < n; i++) {
        a[i] = 1LL * a[i] * inv % P;
    }
}

vector<int> mul(vector<int> a, vector<int> b) {
    int n = 1, tot = a.size() + b.size() - 1;
    while (n < tot) {
        n *= 2;
    }
    if (tot < 128) {
        vector<int> c(a.size() + b.size() - 1);
        for (int i = 0; i < a.size(); i++) {
            for (int j = 0; j < b.size(); j++) {
                c[i + j] = (c[i + j] + 1LL * a[i] * b[j]) % P;
            }
        }
        return c;
    }
    a.resize(n);
    b.resize(n);
    dft(a);
    dft(b);
    for (int i = 0; i < n; i++) {
        a[i] = 1LL * a[i] * b[i] % P;
    }
    idft(a);
    a.resize(tot);
    return a;
}
```

3.13 生成函数

3.13.1 生成函数 (q-int)

```cpp
i64 power(i64 a, i64 b, i64 p) {
    i64 res = 1;
    for (; b; b /= 2, a = i128(a) * a % p) {
        if (b % 2) {
            res = i128(res) * a % p;
        }
    }
    return res;
}

pair<int, int> qint(int q, int n, int p) {
    q %= p;
    for (int x = 2; x * x <= n; x++) {
        if (n % x == 0) {
            auto [v1, e1] = qint(q, x, p);
            auto [v2, e2] = qint(power(q, x, p), n / x, p);
            return {1LL * v1 * v2 % p, e1 + e2};
        }
    }
    if (q == 1) {
        if (n == p) {
            return {0, 1};
        }
        return {n, 0};
    }
    // cerr << q << " " << n << " " << p << "\n";
    i64 v = 1 - power(q, n, 1LL * p * p);
    if (v < 0) {
        v += 1LL * p * p;
    }
    assert(v != 0);
    int inv = power(1 - q + p, p - 2, p);
    if (v % p == 0) {
        return {(v / p) * inv % p, 1};
    } else {
        return {v % p * inv % p, 0};
    }
}
```

3.13.2 生成函数 (q-Binomial)

```cpp
int power(int a, int b, int p) {
    int res = 1;
    for (; b; b /= 2, a = 1LL * a * a % p) {
        if (b % 2) {
            res = 1LL * res * a % p;
        }
    }
    return res;
}

int qint(int n, int q, int p) {
    return 1LL * (power(q, n, p) - 1) * power(q - 1, p - 2, p) % p;
}

int qBinomial(int n, int k, int q, int p) {
    if (q == 0) {
        return 1;
    }
    int r = 0;
    int x = 1;
    do {
        x = 1LL * x * q % p;
        r++;
    } while (x != 1);

    if (n / r > k / r + (n - k) / r) {
        return 0;
    }
    int num = 1, den = 1;
    for (int i = 1; i <= k % r; i++) {
        num = 1LL * num * qint(n % r - i + 1, q, p) % p;
        den = 1LL * den * qint(i, q, p) % p;
    }
    n /= r, k /= r;
    while (n > 0 || k > 0) {
        if (n % p < k % p) {
            return 0;
        }
        for (int i = 1; i <= k % p; i++) {
            num = 1LL * num * (n % p - i + 1) % p;
            den = 1LL * den * i % p;
        }
        n /= p, k /= p;
    }
    int ans = 1LL * num * power(den, p - 2, p) % p;
    return ans;
}
```

3.13.3 生成函数（Binomial 任意模数二项式）

```cpp
vector<pair<int, int>> factorize(int n) {
    vector<pair<int, int>> factors;
    for (int i = 2; static_cast<long long>(i) * i <= n; i++) {
        if (n % i == 0) {
            int t = 0;
            for (; n % i == 0; n /= i)
                ++t;
            factors.emplace_back(i, t);
        }
    }
    if (n > 1)
        factors.emplace_back(n, 1);
    return factors;
}
constexpr int power(int base, i64 exp) {
    int res = 1;
    for (; exp > 0; base *= base, exp /= 2) {
        if (exp % 2 == 1) {
            res *= base;
        }
    }
    return res;
}
constexpr int power(int base, i64 exp, int mod) {
    int res = 1 % mod;
    for (; exp > 0; base = 1LL * base * base % mod, exp /= 2) {
        if (exp % 2 == 1) {
            res = 1LL * res * base % mod;
        }
    }
    return res;
}
int inverse(int a, int m) {
    int g = m, r = a, x = 0, y = 1;
    while (r != 0) {
        int q = g / r;
        g %= r;
        swap(g, r);
        x -= q * y;
        swap(x, y);
    }
    return x < 0 ? x + m : x;
}
int solveModuloEquations(const vector<pair<int, int>> &e) {
    int m = 1;
    for (size_t i = 0; i < e.size(); i++) {
        m *= e[i].first;
    }
    int res = 0;
    for (size_t i = 0; i < e.size(); i++) {
        int p = e[i].first;
        res = (res + 1LL * e[i].second * (m / p) * inverse(m / p, p)) % m;
    }
    return res;
}
constexpr int N = 1E5;
class Binomial {
    const int mod;
private:
    const vector<pair<int, int>> factors;
    vector<int> pk;
    vector<vector<int>> prod;
    static constexpr i64 exponent(i64 n, int p) {
        i64 res = 0;
        for (n /= p; n > 0; n /= p) {
            res += n;
        }
        return res;
    }
    int product(i64 n, size_t i) {
        int res = 1;
        int p = factors[i].first;
        for (; n > 0; n /= p) {
            res = 1LL * res * power(prod[i].back(), n / pk[i], pk[i]) % pk[i] *
prod[i][n % pk[i]] % pk[i];
        }
        return res;
    }
public:
    Binomial(int mod) : mod(mod), factors(factorize(mod)) {
        pk.resize(factors.size());
        prod.resize(factors.size());
        for (size_t i = 0; i < factors.size(); i++) {
            int p = factors[i].first;
            int k = factors[i].second;
            pk[i] = power(p, k);
            prod[i].resize(min(N + 1, pk[i]));
            prod[i][0] = 1;
            for (int j = 1; j < prod[i].size(); j++) {
                if (j % p == 0) {
                    prod[i][j] = prod[i][j - 1];
                } else {
                    prod[i][j] = 1LL * prod[i][j - 1] * j % pk[i];
                }
            }
        }
    }
    int operator()(i64 n, i64 m) {
        if (n < m || m < 0) {
            return 0;
        }
        vector<pair<int, int>> ans(factors.size());
        for (int i = 0; i < factors.size(); i++) {
            int p = factors[i].first;
            int k = factors[i].second;
            int e = exponent(n, p) - exponent(m, p) - exponent(n - m, p);
            if (e >= k) {
                ans[i] = make_pair(pk[i], 0);
            } else {
                int pn = product(n, i);
                int pm = product(m, i);
                int pd = product(n - m, i);
                int res = 1LL * pn * inverse(pm, pk[i]) % pk[i] * inverse(pd, pk[i]) % pk[i] * power(p, e) % pk[i];
                ans[i] = make_pair(pk[i], res);
            }
        }
        return solveModuloEquations(ans);
    }
};
```

3.14 自适应辛普森法（Simpson）

```
const double Pi = acos(-1.0);
constexpr double EPS = 1e-9;
double v, r, d;
double f(double x) {
    double s = sin(x);
    return 1 / v / (sqrt(s * s + 3) - s);
}
double simpson(double l, double r) {
    return (f(l) + 4 * f((l + r) / 2) + f(r)) * (r - l) / 6;
}
double integral(double l, double r, double eps, double st) {
    double mid = (l + r) / 2;
    double sl = simpson(l, mid);
    double sr = simpson(mid, r);
    if (abs(sl + sr - st) <= 15 * eps)
        return sl + sr + (sl + sr - st) / 15;
    return integral(l, mid, eps / 2, sl) + integral(mid, r, eps / 2, sr);
}
double integral(double l, double r) {
    return integral(l, r, EPS, simpson(l, r));
}
```

3.15 矩阵（Matrix）

```
using u64 = unsigned long long;
using Matrix = array<u64, 65>;

Matrix operator*(const Matrix &a, const Matrix &b) {
    Matrix c{};
    for (int i = 0; i <= 64; i++) {
        for (int j = 0; j <= 64; j++) {
            if (j == 64 ? i == 64 : (a[i] >> j & 1)) {
                c[i] ^= b[j];
            }
        }
    }
    return c;
}

u64 operator*(u64 a, const Matrix &b) {
    u64 c = 0;
    for (int i = 0; i <= 64; i++) {
        if (i == 64 || (a >> i & 1)) {
            c ^= b[i];
        }
    }
    return c;
}

Matrix readMatrix() {
    int m;
    cin >> m;

    Matrix f{};
    for (int i = 0; i < m; i++) {
        int s, o;
        u64 A;
        cin >> s >> o >> A;

        if (o == 0) {
            for (int j = 0; j < 64; j++) {
                if (A >> ((j + s) % 64) & 1) {
                    f[64] ^= 1ULL << ((j + s) % 64);
                } else {
                    f[j] ^= 1ULL << ((j + s) % 64);
                }
            }
        } else {
            for (int j = 0; j < 64; j++) {
                if (A >> ((j + s) % 64) & 1) {
                    f[j] ^= 1ULL << ((j + s) % 64);
                }
            }
        }
    }

    u64 B;
    cin >> B;
    f[64] ^= B;

    return f;
}
```

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

```
/**   高斯消元法 ( gaussian elimination ) 【久远】   **/
vector<int> operator*(const vector<int> &lhs, const vector<int> &rhs) {
    vector<int> res(lhs.size() + rhs.size() - 1);
    for (int i = 0; i < int(lhs.size()); ++i)
        for (int j = 0; j < int(rhs.size()); ++j)
            res[i + j] = (res[i + j] + 1ll * lhs[i] * rhs[j]) % P;
    return res;
}
vector<int> operator%(const vector<int> &lhs, const vector<int> &rhs) {
    auto res = lhs;
    int m = rhs.size() - 1;
    int inv = power(rhs.back(), P - 2);
    for (int i = res.size() - 1; i >= m; --i) {
        int x = 1ll * inv * res[i] % P;
        for (int j = 0; j < m; ++j)
            res[i - m + j] = (res[i - m + j] + 1ll * (P - x) * rhs[j]) % P;
    }
    if (int(res.size()) > m)
        res.resize(m);
    return res;
}
vector<int> gauss(vector<vector<int>> a, vector<int> b) {
    int n = a.size();
    for (int i = 0; i < n; ++i) {
        int r = i;
        while (a[r][i] == 0)
            ++r;
        swap(a[i], a[r]);
        swap(b[i], b[r]);
        int inv = power(a[i][i], P - 2);
        for (int j = i; j < n; ++j)
            a[i][j] = 1ll * a[i][j] * inv % P;
        b[i] = 1ll * b[i] * inv % P;
        for (int j = 0; j < n; ++j) {
            if (i == j)
                continue;
            int x = a[j][i];
            for (int k = i; k < n; ++k)
                a[j][k] = (a[j][k] + 1ll * (P - x) * a[i][k]) % P;
            b[j] = (b[j] + 1ll * (P - x) * b[i]) % P;
        }
    }
    return b;
}
/**   高斯消元法（gaussian elimination）【久远】   **/
vector<double> gauss(vector<vector<double>> a, vector<double> b) {
    int n = a.size();
    for (int i = 0; i < n; ++i) {
        double x = a[i][i];
        for (int j = i; j < n; ++j) a[i][j] /= x;
        b[i] /= x;
        for (int j = 0; j < n; ++j) {
            if (i == j) continue;
            x = a[j][i];
            for (int k = i; k < n; ++k) a[j][k] -= a[i][k] * x;
            b[j] -= b[i] * x;
        }
    }
    return b;
}
```

/END/

4 数据结构

4.1 树状数组（Fenwick）

```cpp
template <typename T>
struct Fenwick {
    int n;
    vector<T> a;

    Fenwick(int n_ = 0) {
        init(n_);
    }

    void init(int n_) {
        n = n_;
        a.assign(n, T{});
    }

    void add(int x, const T &v) {
        for (int i = x + 1; i <= n; i += i & -i) {
            a[i - 1] = a[i - 1] + v;
        }
    }

    T sum(int x) {
        T ans{};
        for (int i = x; i > 0; i -= i & -i) {
            ans = ans + a[i - 1];
        }
        return ans;
    }

    T rangeSum(int l, int r) {
        return sum(r) - sum(l);
    }

    int select(const T &k) {
        int x = 0;
        T cur{};
        for (int i = 1 << __lg(n); i; i /= 2) {
            if (x + i <= n && cur + a[x + i - 1] <= k) {
                x += i;
                cur = cur + a[x - 1];
            }
        }
        return x;
    }
};
```

4.2 并查集

4.2.1 并查集（DSU）

```cpp
struct DSU {
    vector<int> f, siz;

    DSU() {}
    DSU(int n) {
        init(n);
    }

    void init(int n) {
        f.resize(n);
        iota(f.begin(), f.end(), 0);
        siz.assign(n, 1);
    }

    int find(int x) {
        while (x != f[x]) {
            x = f[x] = f[f[x]];
        }
        return x;
    }

    bool same(int x, int y) {
        return find(x) == find(y);
    }

    bool merge(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) {
            return false;
        }
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }

    int size(int x) {
        return siz[find(x)];
    }
};
```

4.2.2 可撤销并查集（DSU With Rollback）

```cpp
struct DSU {
    vector<int> siz;
    vector<int> f;
    vector<array<int, 2>> his;

    DSU(int n) : siz(n + 1, 1), f(n + 1) {
        iota(f.begin(), f.end(), 0);
    }

    int find(int x) {
        while (f[x] != x) {
            x = f[x];
        }
        return x;
    }

    bool merge(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) {
            return false;
        }
        if (siz[x] < siz[y]) {
            swap(x, y);
        }
        his.push_back({x, y});
        siz[x] += siz[y];
        f[y] = x;
```

```cpp
            return true;
    }

    int time() {
        return his.size();
    }

    void revert(int tm) {
        while (his.size() > tm) {
            auto [x, y] = his.back();
            his.pop_back();
            f[y] = y;
            siz[x] -= siz[y];
        }
    }
};
```

4.3 线段树

4.3.1 线段树（SegmentTree+Info 区间加+单点修改）

```cpp
struct SegmentTree {
    int n;
    vector<int> tag;
    vector<Info> info;
    SegmentTree(int n_) : n(n_), tag(4 * n), info(4 * n) {}

    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }

    void add(int p, int v) {
        tag[p] += v;
        info[p].max += v;
    }

    void push(int p) {
        add(2 * p, tag[p]);
        add(2 * p + 1, tag[p]);
        tag[p] = 0;
    }

    Info query(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return {};
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        push(p);
        return query(2 * p, l, m, x, y) + query(2 * p + 1, m, r, x, y);
    }

    Info query(int x, int y) {
        return query(1, 0, n, x, y);
    }

    void rangeAdd(int p, int l, int r, int x, int y, int v) {
        if (l >= y || r <= x) {
            return;
        }
        if (l >= x && r <= y) {
            return add(p, v);
        }
        int m = (l + r) / 2;
        push(p);
        rangeAdd(2 * p, l, m, x, y, v);
        rangeAdd(2 * p + 1, m, r, x, y, v);
        pull(p);
    }

    void rangeAdd(int x, int y, int v) {
        rangeAdd(1, 0, n, x, y, v);
    }

    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }

    void modify(int x, const Info &v) {
        modify(1, 0, n, x, v);
    }
};
```

4.3.2 线段树（SegmentTree 区间乘+单点加）

```cpp
struct SegmentTree {
    int n;
    vector<int> tag, sum;
    SegmentTree(int n_) : n(n_), tag(4 * n, 1), sum(4 * n) {}

    void pull(int p) {
        sum[p] = (sum[2 * p] + sum[2 * p + 1]) % P;
    }

    void mul(int p, int v) {
        tag[p] = 1LL * tag[p] * v % P;
        sum[p] = 1LL * sum[p] * v % P;
    }

    void push(int p) {
        mul(2 * p, tag[p]);
        mul(2 * p + 1, tag[p]);
        tag[p] = 1;
    }

    int query(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return 0;
        }
        if (l >= x && r <= y) {
            return sum[p];
        }
        int m = (l + r) / 2;
        push(p);
        return (query(2 * p, l, m, x, y) + query(2 * p + 1, m, r, x, y)) % P;
    }

    int query(int x, int y) {
        return query(1, 0, n, x, y);
    }

    void rangeMul(int p, int l, int r, int x, int y, int v) {
        if (l >= y || r <= x) {
            return;
        }
        if (l >= x && r <= y) {
            return mul(p, v);
        }
        int m = (l + r) / 2;
        push(p);
        rangeMul(2 * p, l, m, x, y, v);
        rangeMul(2 * p + 1, m, r, x, y, v);
        pull(p);
    }

    void rangeMul(int x, int y, int v) {
        rangeMul(1, 0, n, x, y, v);
    }

    void add(int p, int l, int r, int x, int v) {
        if (r - l == 1) {
            sum[p] = (sum[p] + v) % P;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        if (x < m) {
            add(2 * p, l, m, x, v);
        } else {
            add(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }

    void add(int x, int v) {
        add(1, 0, n, x, v);
    }
};
```

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

```cpp
template<class Info> struct SegmentTree {
    int n;
    vector<Info> info;
    SegmentTree() : n(0) {}
    SegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    SegmentTree(vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(vector(n_, v_));
    }
    template<class T>
    void init(vector<T> init_) {
        n = init_.size();
        info.assign(4 << __lg(n), Info());
        function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F &&pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
```

```cpp
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F &&pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
    template<class F>
    int findLast(int p, int l, int r, int x, int y, F &&pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F &&pred) {
        return findLast(1, 0, n, l, r, pred);
    }
};
```

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

```cpp
template<class Info, class Merge = plus<Info>> struct SegmentTree {
    const int n;
    const Merge merge;
    vector<Info> info;
    SegmentTree(int n) : n(n), merge(Merge()), info(4 << __lg(n)) {}
    SegmentTree(vector<Info> init) : SegmentTree(init.size()) {
        function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = merge(info[2 * p], info[2 * p + 1]);
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        return merge(rangeQuery(2 * p, l, m, x, y), rangeQuery(2 * p + 1, m, r, x, y));
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
};
```

4.4 懒标记线段树 (LazySegmentTree)

```cpp
template<class Info, class Tag> struct LazySegmentTree {
    int n;
    vector<Info> info;
    vector<Tag> tag;
    LazySegmentTree() : n(0) {}
    LazySegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    LazySegmentTree(vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(vector(n_, v_));
    }
    template<class T>
    void init(vector<T> init_) {
        n = init_.size();
        info.assign(4 << __lg(n), Info());
        tag.assign(4 << __lg(n), Tag());
        function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void apply(int p, const Tag &v) {
        info[p].apply(v);
        tag[p].apply(v);
    }
    void push(int p) {
        apply(2 * p, tag[p]);
        apply(2 * p + 1, tag[p]);
        tag[p] = Tag();
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        push(p);
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
        if (l >= y || r <= x) {
            return;
        }
        if (l >= x && r <= y) {
            apply(p, v);
            return;
        }
        int m = (l + r) / 2;
        push(p);
        rangeApply(2 * p, l, m, x, y, v);
        rangeApply(2 * p + 1, m, r, x, y, v);
        pull(p);
    }
    void rangeApply(int l, int r, const Tag &v) {
        return rangeApply(1, 0, n, l, r, v);
    }
    void half(int p, int l, int r) {
        if (info[p].act == 0) {
            return;
        }
        if ((info[p].min + 1) / 2 == (info[p].max + 1) / 2) {
            apply(p, {-(info[p].min + 1) / 2});
            return;
        }
        int m = (l + r) / 2;
        push(p);
        half(2 * p, l, m);
        half(2 * p + 1, m, r);
        pull(p);
    }
    void half() {
        half(1, 0, n);
    }

    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F &&pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        push(p);
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F &&pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
    template<class F>
    int findLast(int p, int l, int r, int x, int y, F &&pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        push(p);
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F &&pred) {
        return findLast(1, 0, n, l, r, pred);
    }

    void maintainL(int p, int l, int r, int pre) {
        if (info[p].difl > 0 && info[p].maxlowl < pre) {
            return;
        }
        if (r - l == 1) {
            info[p].max = info[p].maxlowl;
            info[p].maxl = info[p].maxr = l;
```

```cpp
            info[p].maxlowl = info[p].maxlowr = -inf;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        maintainL(2 * p, l, m, pre);
        pre = max(pre, info[2 * p].max);
        maintainL(2 * p + 1, m, r, pre);
        pull(p);
    }
    void maintainL() {
        maintainL(1, 0, n, -1);
    }
    void maintainR(int p, int l, int r, int suf) {
        if (info[p].difr > 0 && info[p].maxlowr < suf) {
            return;
        }
        if (r - l == 1) {
            info[p].max = info[p].maxlowl;
            info[p].maxl = info[p].maxr = l;
            info[p].maxlowl = info[p].maxlowr = -inf;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        maintainR(2 * p + 1, m, r, suf);
        suf = max(suf, info[2 * p + 1].max);
        maintainR(2 * p, l, m, suf);
        pull(p);
    }
    void maintainR() {
        maintainR(1, 0, n, -1);
    }
};

struct Tag {
    int x = 0;
    void apply(const Tag &t) & {
        x = max(x, t.x);
    }
};

struct Info {
    int x = 0;
    void apply(const Tag &t) & {
        x = max(x, t.x);
    }
};

Info operator+(const Info &a, const Info &b) {
    return {max(a.x, b.x)};
}
```

4.5 取模类

4.5.1 取模类（Z 旧版）

```cpp
constexpr int P = 998244353;
// assume -P <= x < 2P
int norm(int x) {
    if (x < 0) {
        x += P;
    }
    if (x >= P) {
        x -= P;
    }
    return x;
}
template<class T>
T power(T a, i64 b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}
struct Z {
    int x;
    Z(int x = 0) : x(norm(x)) {}
    Z(i64 x) : x(norm(x % P)) {}
    int val() const {
        return x;
    }
    Z operator-() const {
        return Z(norm(P - x));
    }
    Z inv() const {
        assert(x != 0);
        return power(*this, P - 2);
    }
    Z &operator*=(const Z &rhs) {
        x = i64(x) * rhs.x % P;
        return *this;
    }
    Z &operator+=(const Z &rhs) {
        x = norm(x + rhs.x);
        return *this;
    }
    Z &operator-=(const Z &rhs) {
        x = norm(x - rhs.x);
        return *this;
    }
    Z &operator/=(const Z &rhs) {
        return *this *= rhs.inv();
    }
    friend Z operator*(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res *= rhs;
        return res;
    }
    friend Z operator+(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res += rhs;
        return res;
    }
    friend Z operator-(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res -= rhs;
        return res;
    }
    friend Z operator/(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res /= rhs;
        return res;
    }
    friend istream &operator>>(istream &is, Z &a) {
        i64 v;
        is >> v;
        a = Z(v);
        return is;
    }
    friend ostream &operator<<(ostream &os, const Z &a) {
        return os << a.val();
    }
};
```

4.5.2 取模类（MLong & MInt 新版）

```cpp
/**   取模类（MLong & MInt 新版）
 *    根据输入内容动态修改 MOD 的方法：Z::setMod(p) 。
**/
template<class T>
constexpr T power(T a, i64 b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}

constexpr i64 mul(i64 a, i64 b, i64 p) {
    i64 res = a * b - i64(1.L * a * b / p) * p;
    res %= p;
    if (res < 0) {
        res += p;
    }
    return res;
}
template<i64 P>
struct MLong {
    i64 x;
    constexpr MLong() : x{} {}
    constexpr MLong(i64 x) : x{norm(x % getMod())} {}

    static i64 Mod;
    constexpr static i64 getMod() {
        if (P > 0) {
            return P;
        } else {
            return Mod;
        }
    }
    constexpr static void setMod(i64 Mod_) {
        Mod = Mod_;
    }
    constexpr i64 norm(i64 x) const {
        if (x < 0) {
            x += getMod();
        }
        if (x >= getMod()) {
            x -= getMod();
        }
        return x;
    }
    constexpr i64 val() const {
        return x;
    }
    explicit constexpr operator i64() const {
        return x;
    }
    constexpr MLong operator-() const {
        MLong res;
        res.x = norm(getMod() - x);
        return res;
    }
    constexpr MLong inv() const {
        assert(x != 0);
        return power(*this, getMod() - 2);
    }
    constexpr MLong &operator*=(MLong rhs) & {
        x = mul(x, rhs.x, getMod());
        return *this;
    }
    constexpr MLong &operator+=(MLong rhs) & {
        x = norm(x + rhs.x);
        return *this;
    }
    constexpr MLong &operator-=(MLong rhs) & {
        x = norm(x - rhs.x);
        return *this;
    }
    constexpr MLong &operator/=(MLong rhs) & {
        return *this *= rhs.inv();
    }
    friend constexpr MLong operator*(MLong lhs, MLong rhs) {
        MLong res = lhs;
        res *= rhs;
        return res;
    }
    friend constexpr MLong operator+(MLong lhs, MLong rhs) {
        MLong res = lhs;
        res += rhs;
        return res;
    }
    friend constexpr MLong operator-(MLong lhs, MLong rhs) {
        MLong res = lhs;
        res -= rhs;
        return res;
    }
    friend constexpr MLong operator/(MLong lhs, MLong rhs) {
        MLong res = lhs;
        res /= rhs;
        return res;
    }
    friend constexpr istream &operator>>(istream &is, MLong &a) {
        i64 v;
        is >> v;
        a = MLong(v);
        return is;
    }
    friend constexpr ostream &operator<<(ostream &os, const MLong &a) {
        return os << a.val();
    }
    friend constexpr bool operator==(MLong lhs, MLong rhs) {
        return lhs.val() == rhs.val();
    }
    friend constexpr bool operator!=(MLong lhs, MLong rhs) {
        return lhs.val() != rhs.val();
    }
```

```cpp
};

template<>
i64 MLong<0LL>::Mod = i64(1E18) + 9;

template<int P>
struct MInt {
    int x;
    constexpr MInt() : x{} {}
    constexpr MInt(i64 x) : x{norm(x % getMod())} {}

    static int Mod;
    constexpr static int getMod() {
        if (P > 0) {
            return P;
        } else {
            return Mod;
        }
    }
    constexpr static void setMod(int Mod_) {
        Mod = Mod_;
    }
    constexpr int norm(int x) const {
        if (x < 0) {
            x += getMod();
        }
        if (x >= getMod()) {
            x -= getMod();
        }
        return x;
    }
    constexpr int val() const {
        return x;
    }
    explicit constexpr operator int() const {
        return x;
    }
    constexpr MInt operator-() const {
        MInt res;
        res.x = norm(getMod() - x);
        return res;
    }
    constexpr MInt inv() const {
        assert(x != 0);
        return power(*this, getMod() - 2);
    }
    constexpr MInt &operator*=(MInt rhs) & {
        x = 1LL * x * rhs.x % getMod();
        return *this;
    }
    constexpr MInt &operator+=(MInt rhs) & {
        x = norm(x + rhs.x);
        return *this;
    }
    constexpr MInt &operator-=(MInt rhs) & {
        x = norm(x - rhs.x);
        return *this;
    }
    constexpr MInt &operator/=(MInt rhs) & {
        return *this *= rhs.inv();
    }
    friend constexpr MInt operator*(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res *= rhs;
        return res;
    }
    friend constexpr MInt operator+(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res += rhs;
        return res;
    }
    friend constexpr MInt operator-(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res -= rhs;
        return res;
    }
    friend constexpr MInt operator/(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res /= rhs;
        return res;
    }
    friend constexpr istream &operator>>(istream &is, MInt &a) {
        i64 v;
        is >> v;
        a = MInt(v);
        return is;
    }
    friend constexpr ostream &operator<<(ostream &os, const MInt &a) {
        return os << a.val();
    }
    friend constexpr bool operator==(MInt lhs, MInt rhs) {
        return lhs.val() == rhs.val();
    }
    friend constexpr bool operator!=(MInt lhs, MInt rhs) {
        return lhs.val() != rhs.val();
    }
};

template<>
int MInt<0>::Mod = 998244353;

template<int V, int P>
constexpr MInt<P> CInv = MInt<P>(V).inv();

constexpr int P = 1000000007;
using Z = MInt<P>;
```

4.5.3 动态取模类（ModIntBase）

```cpp
template<typename T>
constexpr T power(T a, u64 b) {
    T res {1};
    for (; b != 0; b /= 2, a *= a) {
        if (b % 2 == 1) {
            res *= a;
        }
    }
    return res;
}

template<u32 P>
constexpr u32 mulMod(u32 a, u32 b) {
    return 1ULL * a * b % P;
}

template<u64 P>
constexpr u64 mulMod(u64 a, u64 b) {
    u64 res = a * b - u64(1.L * a * b / P - 0.5L) * P;
    res %= P;
    return res;
}

template<typename U, U P>
requires unsigned_integral<U>
struct ModIntBase {
public:
    constexpr ModIntBase() : x {0} {}

    template<typename T>
    requires integral<T>
    constexpr ModIntBase(T x_) : x {norm(x_ % T {P})} {}

    constexpr static U norm(U x) {
        if ((x >> (8 * sizeof(U) - 1) & 1) == 1) {
            x += P;
        }
        if (x >= P) {
            x -= P;
        }
        return x;
    }

    constexpr U val() const {
        return x;
    }

    constexpr ModIntBase operator-() const {
        ModIntBase res;
        res.x = norm(P - x);
        return res;
    }

    constexpr ModIntBase inv() const {
        return power(*this, P - 2);
    }

    constexpr ModIntBase &operator*=(const ModIntBase &rhs) & {
        x = mulMod<P>(x, rhs.val());
        return *this;
    }

    constexpr ModIntBase &operator+=(const ModIntBase &rhs) & {
        x = norm(x + rhs.x);
        return *this;
    }

    constexpr ModIntBase &operator-=(const ModIntBase &rhs) & {
        x = norm(x - rhs.x);
        return *this;
    }

    constexpr ModIntBase &operator/=(const ModIntBase &rhs) & {
        return *this *= rhs.inv();
    }

    friend constexpr ModIntBase operator*(ModIntBase lhs, const ModIntBase &rhs) {
        lhs *= rhs;
        return lhs;
    }

    friend constexpr ModIntBase operator+(ModIntBase lhs, const ModIntBase &rhs) {
        lhs += rhs;
        return lhs;
    }

    friend constexpr ModIntBase operator-(ModIntBase lhs, const ModIntBase &rhs) {
        lhs -= rhs;
        return lhs;
    }

    friend constexpr ModIntBase operator/(ModIntBase lhs, const ModIntBase &rhs) {
        lhs /= rhs;
        return lhs;
    }

    friend constexpr ostream &operator<<(ostream &os, const ModIntBase &a) {
        return os << a.val();
    }

    friend constexpr bool operator==(ModIntBase lhs, ModIntBase rhs) {
        return lhs.val() == rhs.val();
    }

    friend constexpr bool operator!=(ModIntBase lhs, ModIntBase rhs) {
        return lhs.val() != rhs.val();
    }

    friend constexpr bool operator<(ModIntBase lhs, ModIntBase rhs) {
        return lhs.val() < rhs.val();
    }

private:
    U x;
};

template<u32 P>
using ModInt = ModIntBase<u32, P>;

template<u64 P>
using ModInt64 = ModIntBase<u64, P>;

constexpr u32 P = 998244353;
using Z = ModInt<P>;
```

4.6 状压RMQ (RMQ)

```
template<class T, class Cmp = less<T>> struct RMQ {
    const Cmp cmp = Cmp();
    static constexpr unsigned B = 64;
    using u64 = unsigned long long;
    int n;
    vector<vector<T>> a;
    vector<T> pre, suf, ini;
    vector<u64> stk;
    RMQ() {}
    RMQ(const vector<T> &v) {
        init(v);
    }
    void init(const vector<T> &v) {
        n = v.size();
        pre = suf = ini = v;
        stk.resize(n);
        if (!n) {
                return;
            }
        const int M = (n - 1) / B + 1;
        const int lg = __lg(M);
        a.assign(lg + 1, vector<T>(M));
        for (int i = 0; i < M; i++) {
            a[0][i] = v[i * B];
            for (int j = 1; j < B && i * B + j < n; j++) {
                a[0][i] = min(a[0][i], v[i * B + j], cmp);
            }
        }
        for (int i = 1; i < n; i++) {
            if (i % B) {
                pre[i] = min(pre[i], pre[i - 1], cmp);
            }
        }
        for (int i = n - 2; i >= 0; i--) {
            if (i % B != B - 1) {
                suf[i] = min(suf[i], suf[i + 1], cmp);
            }
        }
        for (int j = 0; j < lg; j++) {
            for (int i = 0; i + (2 << j) <= M; i++) {
                a[j + 1][i] = min(a[j][i], a[j][i + (1 << j)], cmp);
            }
        }
        for (int i = 0; i < M; i++) {
            const int l = i * B;
            const int r = min(1U * n, l + B);
            u64 s = 0;
            for (int j = l; j < r; j++) {
                while (s && cmp(v[j], v[__lg(s) + l])) {
                    s ^= 1ULL << __lg(s);
                }
                s |= 1ULL << (j - l);
                stk[j] = s;
            }
        }
    }
    T operator()(int l, int r) {
        if (l / B != (r - 1) / B) {
            T ans = min(suf[l], pre[r - 1], cmp);
            l = l / B + 1;
            r = r / B;
            if (l < r) {
                int k = __lg(r - l);
                ans = min({ans, a[k][l], a[k][r - (1 << k)]}, cmp);
            }
            return ans;
        } else {
            int x = B * (l / B);
            return ini[__builtin_ctzll(stk[r - 1] >> (l - x)) + l];
        }
    }
};
```

4.7 Splay

```
struct Node {
    Node *l = nullptr;
    Node *r = nullptr;
    int cnt = 0;
```

```cpp
    i64 sum = 0;
};

Node *add(Node *t, int l, int r, int p, int v) {
    Node *x = new Node;
    if (t) {
        *x = *t;
    }
    x->cnt += 1;
    x->sum += v;
    if (r - l == 1) {
        return x;
    }
    int m = (l + r) / 2;
    if (p < m) {
        x->l = add(x->l, l, m, p, v);
    } else {
        x->r = add(x->r, m, r, p, v);
    }
    return x;
}

int find(Node *tl, Node *tr, int l, int r, int x) {
    if (r <= x) {
        return -1;
    }
    if (l >= x) {
        int cnt = (tr ? tr->cnt : 0) - (tl ? tl->cnt : 0);
        if (cnt == 0) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
    }
    int m = (l + r) / 2;
    int res = find(tl ? tl->l : tl, tr ? tr->l : tr, l, m, x);
    if (res == -1) {
        res = find(tl ? tl->r : tl, tr ? tr->r : tr, m, r, x);
    }
    return res;
}

pair<int, i64> get(Node *t, int l, int r, int x, int y) {
    if (l >= y || r <= x || !t) {
        return {0, 0LL};
    }
    if (l >= x && r <= y) {
        return {t->cnt, t->sum};
    }
    int m = (l + r) / 2;
    auto [cl, sl] = get(t->l, l, m, x, y);
    auto [cr, sr] = get(t->r, m, r, x, y);
    return {cl + cr, sl + sr};
}

struct Tree {
    int add = 0;
    int val = 0;
    int id = 0;
    Tree *ch[2] = {};
    Tree *p = nullptr;
};

```

```cpp
int pos(Tree *t) {
    return t->p->ch[1] == t;
}

void add(Tree *t, int v) {
    t->val += v;
    t->add += v;
}

void push(Tree *t) {
    if (t->ch[0]) {
        add(t->ch[0], t->add);
    }
    if (t->ch[1]) {
        add(t->ch[1], t->add);
    }
    t->add = 0;
}

void rotate(Tree *t) {
    Tree *q = t->p;
    int x = !pos(t);
    q->ch[!x] = t->ch[x];
    if (t->ch[x]) t->ch[x]->p = q;
    t->p = q->p;
    if (q->p) q->p->ch[pos(q)] = t;
    t->ch[x] = q;
    q->p = t;
}

void splay(Tree *t) {
    vector<Tree *> s;
    for (Tree *i = t; i->p; i = i->p) s.push_back(i->p);
    while (!s.empty()) {
        push(s.back());
        s.pop_back();
    }
    push(t);
    while (t->p) {
        if (t->p->p) {
            if (pos(t) == pos(t->p)) rotate(t->p);
            else rotate(t);
        }
        rotate(t);
    }
}

void insert(Tree *&t, Tree *x, Tree *p = nullptr) {
    if (!t) {
        t = x;
        x->p = p;
        return;
    }

    push(t);
    if (x->val < t->val) {
        insert(t->ch[0], x, t);
    } else {
        insert(t->ch[1], x, t);
    }
}

void dfs(Tree *t) {
    if (!t) {
        return;
    }
    push(t);
    dfs(t->ch[0]);
    cerr << t->val << " ";
    dfs(t->ch[1]);
}

pair<Tree *, Tree *> split(Tree *t, int x) {
    if (!t) {
        return {t, t};
    }
    Tree *v = nullptr;
    Tree *j = t;
    for (Tree *i = t; i; ) {
        push(i);
        j = i;
        if (i->val >= x) {
            v = i;
            i = i->ch[0];
        } else {
            i = i->ch[1];
        }
    }

    splay(j);
    if (!v) {
        return {j, nullptr};
    }

    splay(v);

    Tree *u = v->ch[0];
    if (u) {
        v->ch[0] = u->p = nullptr;
    }
    // cerr << "split " << x << "\n";
    // dfs(u);
    // cerr << "\n";
    // dfs(v);
    // cerr << "\n";
    return {u, v};
}

Tree *merge(Tree *l, Tree *r) {
    if (!l) {
        return r;
    }
    if (!r) {
        return l;
    }
    Tree *i = l;
    while (i->ch[1]) {
        i = i->ch[1];
    }
    splay(i);
    i->ch[1] = r;
    r->p = i;
    return i;
}
```

```cpp
struct Matrix : array<array<i64, 4>, 4> {
    Matrix(i64 v = 0) {
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                (*this)[i][j] = (i == j ? v : inf);
            }
        }
    }
};

Matrix operator*(const Matrix &a, const Matrix &b) {
    Matrix c(inf);
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 4; k++) {
                c[i][k] = min(c[i][k], a[i][j] + b[j][k]);
            }
        }
        c[i][3] = min(c[i][3], a[i][3]);
    }
    c[3][3] = 0;
    return c;
}

struct Node {
    Node *ch[2], *p;
    i64 sumg = 0;
    i64 sumh = 0;
    i64 sumb = 0;
    i64 g = 0;
    i64 h = 0;
    i64 b = 0;
    Matrix mat;
    Matrix prd;
    array<i64, 4> ans{};
    Node() : ch{nullptr, nullptr}, p(nullptr) {}

    void update() {
        mat = Matrix(inf);
        mat[0][0] = b + h - g + sumg;
        mat[1][1] = mat[1][2] = mat[1][3] = h + sumh;
        mat[2][0] = mat[2][1] = mat[2][2] = mat[2][3] = b + h + sumb;
        mat[3][3] = 0;
    }
};
void push(Node *t) {

}
void pull(Node *t) {
    t->prd = (t->ch[0] ? t->ch[0]->prd : Matrix()) * t->mat * (t->ch[1] ? t->ch[1]-
>prd : Matrix());
}
bool isroot(Node *t) {
    return t->p == nullptr || (t->p->ch[0] != t && t->p->ch[1] != t);
}
int pos(Node *t) {
    return t->p->ch[1] == t;
}
void pushAll(Node *t) {
    if (!isroot(t)) {
        pushAll(t->p);
    }
    push(t);
}
void rotate(Node *t) {
    Node *q = t->p;
    int x = !pos(t);
    q->ch[!x] = t->ch[x];
    if (t->ch[x]) {
        t->ch[x]->p = q;
    }
    t->p = q->p;
    if (!isroot(q)) {
        q->p->ch[pos(q)] = t;
    }
    t->ch[x] = q;
    q->p = t;
    pull(q);
}
void splay(Node *t) {
    pushAll(t);
    while (!isroot(t)) {
        if (!isroot(t->p)) {
            if (pos(t) == pos(t->p)) {
                rotate(t->p);
            } else {
                rotate(t);
            }
        }
        rotate(t);
    }
    pull(t);
}

array<i64, 4> get(Node *t) {
    array<i64, 4> ans;
    ans.fill(inf);
    ans[3] = 0;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 4; j++) {
            ans[i] = min(ans[i], t->prd[i][j]);
        }
    }
    return ans;
}

void access(Node *t) {
    array<i64, 4> old{};
    for (Node *i = t, *q = nullptr; i; q = i, i = i->p) {
        splay(i);
        if (i->ch[1]) {
            auto res = get(i->ch[1]);
            i->sumg += res[0];
            i->sumh += min({res[1], res[2], res[3]});
            i->sumb += min({res[0], res[1], res[2], res[3]});
        }
        i->ch[1] = q;
        i->sumg -= old[0];
        i->sumh -= min({old[1], old[2], old[3]});
        i->sumb -= min({old[0], old[1], old[2], old[3]});
        old = get(i);
        i->update();
        pull(i);
    }
    splay(t);
}
```

```cpp
constexpr int D = 27;
struct Info {
    int up[D][2] {};
    int down[D][2] {};
    int t = 0;
    i64 ans = 0;
};

Info operator+(const Info &a, const Info &b) {
    Info c;
    c.t = a.t ^ b.t;
    c.ans = a.ans + b.ans;
    for (int i = 0; i < D; i++) {
        for (int j = 0; j < 2; j++) {
            c.ans += (1LL << i) * a.down[i][j] * b.up[i][j ^ 1];
            c.up[i][j] += a.up[i][j] + b.up[i][j ^ (a.t >> i & 1)];
            c.down[i][j] += b.down[i][j] + a.down[i][j ^ (b.t >> i & 1)];
        }
    }
    return c;
}
struct Node {
    Node *ch[2], *p;
    Info val;
    Info tot;
    int cnt[D][2];
    i64 pair[D][2];
    i64 sum;
    Node() : ch{nullptr, nullptr}, p(nullptr), cnt {}, pair {}, sum {} {}
};
void pull(Node *t) {
    t->tot = (t->ch[0] ? t->ch[0]->tot : Info {}) + t->val + (t->ch[1] ? t->ch[1]-
>tot : Info {});
}
bool isroot(Node *t) {
    return t->p == nullptr || (t->p->ch[0] != t && t->p->ch[1] != t);
}
int pos(Node *t) {
    return t->p->ch[1] == t;
}
void rotate(Node *t) {
    Node *q = t->p;
    int x = !pos(t);
    q->ch[!x] = t->ch[x];
    if (t->ch[x]) {
        t->ch[x]->p = q;
    }
    t->p = q->p;
    if (!isroot(q)) {
        q->p->ch[pos(q)] = t;
    }
    t->ch[x] = q;
    q->p = t;
    pull(q);
}
void update(Node *t) {
    t->val.ans = t->val.t + t->sum;
    for (int i = 0; i < D; i++) {
        t->val.ans += (1LL << i) * t->pair[i][t->val.t >> i & 1];
        for (int j = 0; j < 2; j++) {
            t->val.up[i][j] = t->cnt[i][j ^ (t->val.t >> i & 1)];
            t->val.down[i][j] = t->cnt[i][j ^ (t->val.t >> i & 1)];
        }
        t->val.up[i][t->val.t >> i & 1]++;
        t->val.down[i][t->val.t >> i & 1]++;
    }
```

```cpp
    pull(t);
}
void splay(Node *t) {
    while (!isroot(t)) {
        if (!isroot(t->p)) {
            if (pos(t) == pos(t->p)) {
                rotate(t->p);
            } else {
                rotate(t);
            }
        }
        rotate(t);
    }
    pull(t);
}
void add(Node *t, Info s) {
    for (int i = 0; i < D; i++) {
        for (int x = 0; x < 2; x++) {
            t->pair[i][x] += s.up[i][1 ^ x];
            for (int j = 0; j < 2; j++) {
                t->pair[i][x] += t->cnt[i][j] * s.up[i][j ^ 1 ^ x];
            }
        }
        for (int j = 0; j < 2; j++) {
            t->cnt[i][j] += s.up[i][j];
        }
    }
    t->sum += s.ans;
}
void del(Node *t, Info s) {
    t->sum -= s.ans;
    for (int i = 0; i < D; i++) {
        for (int j = 0; j < 2; j++) {
            t->cnt[i][j] -= s.up[i][j];
        }
        for (int x = 0; x < 2; x++) {
            for (int j = 0; j < 2; j++) {
                t->pair[i][x] -= t->cnt[i][j] * s.up[i][j ^ 1 ^ x];
            }
            t->pair[i][x] -= s.up[i][1 ^ x];
        }
    }
}
void access(Node *t, int v) {
    Info lst;
    for (Node *i = t, *q = nullptr; i; q = i, i = i->p) {
        splay(i);
        if (i->ch[1]) {
            add(i, i->ch[1]->tot);
        }
        i->ch[1] = q;
        if (q) {
            del(i, lst);
        } else {
            i->val.t = v;
        }
        lst = i->tot;
        update(i);
    }
    splay(t);
}
```

4.8 其他平衡树

```cpp
struct Node {
    Node *l = nullptr;
    Node *r = nullptr;
    int sum = 0;
    int sumodd = 0;

    Node(Node *t) {
        if (t) {
            *this = *t;
        }
    }
};

Node *add(Node *t, int l, int r, int x, int v) {
    t = new Node(t);
    t->sum += v;
    t->sumodd += (x % 2) * v;
    if (r - l == 1) {
        return t;
    }
    int m = (l + r) / 2;
    if (x < m) {
        t->l = add(t->l, l, m, x, v);
    } else {
        t->r = add(t->r, m, r, x, v);
    }
    return t;
}

int query1(Node *t1, Node *t2, int l, int r, int k) {
    if (r - l == 1) {
        return l;
    }
    int m = (l + r) / 2;
    int odd = (t1 && t1->r ? t1->r->sumodd : 0) - (t2 && t2->r ? t2->r->sumodd : 0);
    int cnt = (t1 && t1->r ? t1->r->sum : 0) - (t2 && t2->r ? t2->r->sum : 0);
    if (odd > 0 || cnt > k) {
        return query1(t1 ? t1->r : t1, t2 ? t2->r : t2, m, r, k);
    } else {
        return query1(t1 ? t1->l : t1, t2 ? t2->l : t2, l, m, k - cnt);
    }
}

array<int, 3> query2(Node *t1, Node *t2, int l, int r, int k) {
    if (r - l == 1) {
        int cnt = (t1 ? t1->sumodd : 0) - (t2 ? t2->sumodd : 0);
        return {l, cnt, k};
    }
    int m = (l + r) / 2;
    int cnt = (t1 && t1->r ? t1->r->sumodd : 0) - (t2 && t2->r ? t2->r->sumodd : 0);
    if (cnt > k) {
        return query2(t1 ? t1->r : t1, t2 ? t2->r : t2, m, r, k);
    } else {
        return query2(t1 ? t1->l : t1, t2 ? t2->l : t2, l, m, k - cnt);
    }
}
```

```cpp
struct Node {
    Node *l = nullptr;
    Node *r = nullptr;
    int cnt = 0;
};

Node *add(Node *t, int l, int r, int x) {
    if (t) {
        t = new Node(*t);
    } else {
        t = new Node;
    }
    t->cnt += 1;
    if (r - l == 1) {
        return t;
    }
    int m = (l + r) / 2;
    if (x < m) {
        t->l = add(t->l, l, m, x);
    } else {
        t->r = add(t->r, m, r, x);
    }
    return t;
}

int query(Node *t1, Node *t2, int l, int r, int x) {
    int cnt = (t2 ? t2->cnt : 0) - (t1 ? t1->cnt : 0);
    if (cnt == 0 || l >= x) {
        return -1;
    }
    if (r - l == 1) {
        return l;
    }
    int m = (l + r) / 2;
    int res = query(t1 ? t1->r : t1, t2 ? t2->r : t2, m, r, x);
    if (res == -1) {
        res = query(t1 ? t1->l : t1, t2 ? t2->l : t2, l, m, x);
    }
    return res;
}
```

```
struct Info {
    int imp = 0;
    int id = 0;
};

Info operator+(Info a, Info b) {
    return {max(a.imp, b.imp), 0};
}

struct Node {
    int w = rng();
    Info info;
    Info sum;
    int siz = 1;
    Node *l = nullptr;
    Node *r = nullptr;
};

void pull(Node *t) {
    t->sum = t->info;
    t->siz = 1;
    if (t->l) {
        t->sum = t->l->sum + t->sum;
        t->siz += t->l->siz;
    }
```

```cpp
    if (t->r) {
        t->sum = t->sum + t->r->sum;
        t->siz += t->r->siz;
    }
}

pair<Node *, Node *> splitAt(Node *t, int p) {
    if (!t) {
        return {t, t};
    }
    if (p <= (t->l ? t->l->siz : 0)) {
        auto [l, r] = splitAt(t->l, p);
        t->l = r;
        pull(t);
        return {l, t};
    } else {
        auto [l, r] = splitAt(t->r, p - 1 - (t->l ? t->l->siz : 0));
        t->r = l;
        pull(t);
        return {t, r};
    }
}

void insertAt(Node *&t, int p, Node *x) {
    if (!t) {
        t = x;
        return;
    }
    if (x->w < t->w) {
        auto [l, r] = splitAt(t, p);
        t = x;
        t->l = l;
        t->r = r;
        pull(t);
        return;
    }
    if (p <= (t->l ? t->l->siz : 0)) {
        insertAt(t->l, p, x);
    } else {
        insertAt(t->r, p - 1 - (t->l ? t->l->siz : 0), x);
    }
    pull(t);
}

Node *merge(Node *a, Node *b) {
    if (!a) {
        return b;
    }
    if (!b) {
        return a;
    }

    if (a->w < b->w) {
        a->r = merge(a->r, b);
        pull(a);
        return a;
    } else {
        b->l = merge(a, b->l);
        pull(b);
        return b;
    }
}

int query(Node *t, int v) {
    if (!t) {
        return 0;
    }
    if (t->sum.imp < v) {
        return t->siz;
    }
    int res = query(t->r, v);
    if (res != (t->r ? t->r->siz : 0)) {
        return res;
    }
    if (t->info.imp > v) {
        return res;
    }
    return res + 1 + query(t->l, v);
}

void dfs(Node *t) {
    if (!t) {
        return;
    }
    dfs(t->l);
    cout << t->info.id << " ";
    dfs(t->r);
}
```

```
struct Node {
    Node *l = nullptr;
    Node *r = nullptr;
    int cnt = 0;
    int cntnew = 0;
};

Node *add(int l, int r, int x, int isnew) {
    Node *t = new Node;
    t->cnt = 1;
    t->cntnew = isnew;
    if (r - l == 1) {
        return t;
    }
    int m = (l + r) / 2;
    if (x < m) {
        t->l = add(l, m, x, isnew);
    } else {
        t->r = add(m, r, x, isnew);
    }
    return t;
}

struct Info {
    Node *t = nullptr;
    int psum = 0;
    bool rev = false;
};

void pull(Node *t) {
    t->cnt = (t->l ? t->l->cnt : 0) + (t->r ? t->r->cnt : 0);
    t->cntnew = (t->l ? t->l->cntnew : 0) + (t->r ? t->r->cntnew : 0);
}

pair<Node *, Node *> split(Node *t, int l, int r, int x, bool rev) {
    if (!t) {
        return {t, t};
    }
    if (x == 0) {
        return {nullptr, t};
    }
    if (x == t->cnt) {
        return {t, nullptr};
    }
    if (r - l == 1) {
        Node *t2 = new Node;
        t2->cnt = t->cnt - x;
        t->cnt = x;
        return {t, t2};
    }
    Node *t2 = new Node;
    int m = (l + r) / 2;
    if (!rev) {
        if (t->l && x <= t->l->cnt) {
            tie(t->l, t2->l) = split(t->l, l, m, x, rev);
            t2->r = t->r;
            t->r = nullptr;
        } else {
            tie(t->r, t2->r) = split(t->r, m, r, x - (t->l ? t->l->cnt : 0), rev);
        }
    } else {
        if (t->r && x <= t->r->cnt) {
            tie(t->r, t2->r) = split(t->r, m, r, x, rev);
            t2->l = t->l;
            t->l = nullptr;
        } else {
            tie(t->l, t2->l) = split(t->l, l, m, x - (t->r ? t->r->cnt : 0), rev);
        }
    }
    pull(t);
    pull(t2);
    return {t, t2};
}

Node *merge(Node *t1, Node *t2, int l, int r) {
    if (!t1) {
        return t2;
    }
    if (!t2) {
        return t1;
    }
    if (r - l == 1) {
        t1->cnt += t2->cnt;
        t1->cntnew += t2->cntnew;
        delete t2;
        return t1;
    }
    int m = (l + r) / 2;
    t1->l = merge(t1->l, t2->l, l, m);
    t1->r = merge(t1->r, t2->r, m, r);
    delete t2;
    pull(t1);
    return t1;
}
```

4.9 分数四则运算（Frac）

```cpp
template<class T>
struct Frac {
    T num;
    T den;
    Frac(T num_, T den_) : num(num_), den(den_) {
        if (den < 0) {
            den = -den;
            num = -num;
        }
    }
    Frac() : Frac(0, 1) {}
    Frac(T num_) : Frac(num_, 1) {}
    explicit operator double() const {
        return 1. * num / den;
    }
    Frac &operator+=(const Frac &rhs) {
        num = num * rhs.den + rhs.num * den;
        den *= rhs.den;
        return *this;
    }
    Frac &operator-=(const Frac &rhs) {
        num = num * rhs.den - rhs.num * den;
        den *= rhs.den;
        return *this;
    }
    Frac &operator*=(const Frac &rhs) {
        num *= rhs.num;
        den *= rhs.den;
        return *this;
    }
    Frac &operator/=(const Frac &rhs) {
        num *= rhs.den;
        den *= rhs.num;
        if (den < 0) {
            num = -num;
            den = -den;
        }
        return *this;
    }
    friend Frac operator+(Frac lhs, const Frac &rhs) {
        return lhs += rhs;
    }
    friend Frac operator-(Frac lhs, const Frac &rhs) {
        return lhs -= rhs;
    }
    friend Frac operator*(Frac lhs, const Frac &rhs) {
        return lhs *= rhs;
    }
    friend Frac operator/(Frac lhs, const Frac &rhs) {
        return lhs /= rhs;
    }
    friend Frac operator-(const Frac &a) {
        return Frac(-a.num, a.den);
    }
    friend bool operator==(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den == rhs.num * lhs.den;
    }
    friend bool operator!=(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den != rhs.num * lhs.den;
    }
    friend bool operator<(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den < rhs.num * lhs.den;
    }
    friend bool operator>(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den > rhs.num * lhs.den;
    }
    friend bool operator<=(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den <= rhs.num * lhs.den;
    }
    friend bool operator>=(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den >= rhs.num * lhs.den;
    }
    friend ostream &operator<<(ostream &os, Frac x) {
        T g = gcd(x.num, x.den);
        if (x.den == g) {
            return os << x.num / g;
        } else {
            return os << x.num / g << "/" << x.den / g;
        }
    }
};
```

4.10 线性基 (Basis)

```
struct Basis {
    int a[20] {};
    int t[20] {};

    Basis() {
        fill(t, t + 20, -1);
    }

    void add(int x, int y = 1E9) {
        for (int i = 0; i < 20; i++) {
            if (x >> i & 1) {
                if (y > t[i]) {
                    swap(a[i], x);
                    swap(t[i], y);
                }
                x ^= a[i];
            }
        }
    }

    bool query(int x, int y = 0) {
        for (int i = 0; i < 20; i++) {
            if ((x >> i & 1) && t[i] >= y) {
                x ^= a[i];
            }
        }
        return x == 0;
    }
};
```

4.11 高精度 (BigInt)

```
constexpr int N = 1000;

struct BigInt {
    int a[N];
    BigInt(int x = 0) : a{} {
        for (int i = 0; x; i++) {
            a[i] = x % 10;
            x /= 10;
        }
    }
    BigInt &operator*=(int x) {
        for (int i = 0; i < N; i++) {
            a[i] *= x;
        }
        for (int i = 0; i < N - 1; i++) {
            a[i + 1] += a[i] / 10;
            a[i] %= 10;
        }
        return *this;
    }
    BigInt &operator/=(int x) {
        for (int i = N - 1; i >= 0; i--) {
            if (i) {
                a[i - 1] += a[i] % x * 10;
            }
            a[i] /= x;
        }
        return *this;
    }
    BigInt &operator+=(const BigInt &x) {
        for (int i = 0; i < N; i++) {
            a[i] += x.a[i];
            if (a[i] >= 10) {
                a[i + 1] += 1;
                a[i] -= 10;
            }
        }
        return *this;
    }
};

ostream &operator<<(ostream &o, const BigInt &a) {
    int t = N - 1;
    while (a.a[t] == 0) {
        t--;
    }
    for (int i = t; i >= 0; i--) {
        o << a.a[i];
    }
    return o;
}
```

4.12 Link-Cut Tree

```
namespace SegT {
    int tag[8 * N];
    int64_t wsum[8 * N], sum[8 * N];
    void add(int p, int l, int r, int v) {
        sum[p] += v * (r - l);
        wsum[p] += 1ll * v * (r - l) * (l + r + 1) / 2;
        tag[p] += v;
    }
    void push(int p, int l, int r) {
        int m = (l + r) / 2;
        add(2 * p, l, m, tag[p]);
        add(2 * p + 1, m, r, tag[p]);
        tag[p] = 0;
    }
    void pull(int p) {
        sum[p] = sum[2 * p] + sum[2 * p + 1];
        wsum[p] = wsum[2 * p] + wsum[2 * p + 1];
    }
    void rangeAdd(int p, int l, int r, int x, int y, int v) {
        if (l >= y || r <= x)
            return;
        if (l >= x && r <= y)
            return add(p, l, r, v);
        push(p, l, r);
        int m = (l + r) / 2;
        rangeAdd(2 * p, l, m, x, y, v);
        rangeAdd(2 * p + 1, m, r, x, y, v);
        pull(p);
    }
    int64_t query(int p, int l, int r, int x) {
        if (l >= x)
            return sum[p] * x;
        if (r <= x)
            return wsum[p];
        int m = (l + r) / 2;
        push(p, l, r);
        return query(2 * p, l, m, x) + query(2 * p + 1, m, r, x);
    }
    int get(int p, int l, int r, int x) {
        if (r - l == 1)
            return sum[p];
        int m = (l + r) / 2;
        push(p, l, r);
        if (x < m) {
            return get(2 * p, l, m, x);
        } else {
            return get(2 * p + 1, m, r, x);
        }
    }
}
namespace LCT {
    int ch[2 * N][2], p[2 * N], endp[2 * N], mn[2 * N], mx[2 * N];
    bool isroot(int t) {
        return ch[p[t]][0] != t && ch[p[t]][1] != t;
    }
    bool pos(int t) {
        return ch[p[t]][1] == t;
    }
    void pull(int t) {
        mn[t] = max(0, ch[t][0] ? mn[ch[t][0]] : SAM::len[SAM::link[t]]);
        mx[t] = ch[t][1] ? mx[ch[t][1]] : SAM::len[t];
    }
    void rotate(int t) {
        int k = !pos(t);
        int q = p[t];
        ch[q][!k] = ch[t][k];
        if (ch[t][k])
            p[ch[t][k]] = q;
        p[t] = p[q];
        if (isroot(q)) {
            endp[t] = endp[q];
        } else {
            ch[p[q]][pos(q)] = t;
        }
        ch[t][k] = q;
        p[q] = t;
        pull(q);
    }
    void splay(int t) {
        while (!isroot(t)) {
            int q = p[t];
            if (!isroot(q))
                rotate(pos(t) == pos(q) ? q : t);
            rotate(t);
        }
        pull(t);
    }
    void access(int t, int len) {
        for (int i = t, u = 0; i; u = i, i = p[i]) {
            splay(i);
            if (ch[i][1])
                endp[ch[i][1]] = endp[i];
            ch[i][1] = 0;
            pull(i);
            if (u)
                SegT::rangeAdd(1, 0, n, endp[i] - mx[i], endp[i] - mn[i], -1);
            ch[i][1] = u;
            pull(i);
        }
        splay(t);
        endp[t] = len;
        SegT::rangeAdd(1, 0, n, len - mx[t], len - mn[t], 1);
    }
    void cut(int t) {
        splay(t);
        if (ch[t][0]) {
            endp[ch[t][0]] = endp[t];
            p[ch[t][0]] = p[t];
            p[t] = 0;
            ch[t][0] = 0;
            pull(t);
        } else {
            p[t] = 0;
        }
    }
    void link(int t, int x) {
        p[t] = x;
    }
}
```

```
struct Node {
    Node *ch[2], *p;
    bool rev;
    int siz = 1;
    Node() : ch{nullptr, nullptr}, p(nullptr), rev(false) {}
};
void reverse(Node *t) {
    if (t) {
        swap(t->ch[0], t->ch[1]);
        t->rev ^= 1;
    }
}
void push(Node *t) {
    if (t->rev) {
        reverse(t->ch[0]);
        reverse(t->ch[1]);
        t->rev = false;
    }
}
void pull(Node *t) {
    t->siz = (t->ch[0] ? t->ch[0]->siz : 0) + 1 + (t->ch[1] ? t->ch[1]->siz : 0);
```

```cpp
}
bool isroot(Node *t) {
    return t->p == nullptr || (t->p->ch[0] != t && t->p->ch[1] != t);
}
int pos(Node *t) {
    return t->p->ch[1] == t;
}
void pushAll(Node *t) {
    if (!isroot(t)) {
        pushAll(t->p);
    }
    push(t);
}
void rotate(Node *t) {
    Node *q = t->p;
    int x = !pos(t);
    q->ch[!x] = t->ch[x];
    if (t->ch[x]) {
        t->ch[x]->p = q;
    }
    t->p = q->p;
    if (!isroot(q)) {
        q->p->ch[pos(q)] = t;
    }
    t->ch[x] = q;
    q->p = t;
    pull(q);
}
void splay(Node *t) {
    pushAll(t);
    while (!isroot(t)) {
        if (!isroot(t->p)) {
            if (pos(t) == pos(t->p)) {
                rotate(t->p);
            } else {
                rotate(t);
            }
        }
        rotate(t);
    }
    pull(t);
}
void access(Node *t) {
    for (Node *i = t, *q = nullptr; i; q = i, i = i->p) {
        splay(i);
        i->ch[1] = q;
        pull(i);
    }
    splay(t);
}
void makeroot(Node *t) {
    access(t);
    reverse(t);
}
void link(Node *x, Node *y) {
    makeroot(x);
    x->p = y;
}
void split(Node *x, Node *y) {
    makeroot(x);
    access(y);
}
void cut(Node *x, Node *y) {
    split(x, y);
    x->p = y->ch[0] = nullptr;
    pull(y);
}
int dist(Node *x, Node *y) {
    split(x, y);
    return y->siz - 1;
}
```

/END/

5　字符串

5.1　马拉车（Manacher）

```cpp
vector<int> manacher(string s) {
    string t = "#";
    for (auto c : s) {
        t += c;
        t += '#';
    }
    int n = t.size();
    vector<int> r(n);
    for (int i = 0, j = 0; i < n; i++) {
        if (2 * j - i >= 0 && j + r[j] > i) {
            r[i] = min(r[2 * j - i], j + r[j] - i);
        }
        while (i - r[i] >= 0 && i + r[i] < n && t[i - r[i]] == t[i + r[i]]) {
            r[i] += 1;
        }
        if (i + r[i] > j + r[j]) {
            j = i;
        }
    }
    return r;
}

```

5.2　Z函数

```cpp
vector<int> Z(string s) {
    int n = s.size();
    vector<int> z(n + 1);
    z[0] = n;
    for (int i = 1, j = 1; i < n; i++) {
        z[i] = max(0, min(j + z[j] - i, z[i - j]));
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }
        if (i + z[i] > j + z[j]) {
            j = i;
        }
    }
    return z;
}
```

5.3　后缀数组

5.3.1　后缀数组（SuffixArray 旧版）

```cpp
struct SuffixArray {
    int n;
    vector<int> sa, rk, lc;
    SuffixArray(const string &s) {
        n = s.length();
        sa.resize(n);
        lc.resize(n - 1);
        rk.resize(n);
        iota(sa.begin(), sa.end(), 0);
        sort(sa.begin(), sa.end(), [&](int a, int b) {return s[a] < s[b];});
        rk[sa[0]] = 0;
        for (int i = 1; i < n; ++i)
            rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
        int k = 1;
        vector<int> tmp, cnt(n);
        tmp.reserve(n);
        while (rk[sa[n - 1]] < n - 1) {
            tmp.clear();
            for (int i = 0; i < k; ++i)
                tmp.push_back(n - k + i);
            for (auto i : sa)
                if (i >= k)
                    tmp.push_back(i - k);
            fill(cnt.begin(), cnt.end(), 0);
            for (int i = 0; i < n; ++i)
                ++cnt[rk[i]];
            for (int i = 1; i < n; ++i)
                cnt[i] += cnt[i - 1];
            for (int i = n - 1; i >= 0; --i)
                sa[--cnt[rk[tmp[i]]]] = tmp[i];
            swap(rk, tmp);
            rk[sa[0]] = 0;
            for (int i = 1; i < n; ++i)
                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]);
            k *= 2;
        }
        for (int i = 0, j = 0; i < n; ++i) {
            if (rk[i] == 0) {
                j = 0;
            } else {
                for (j -= j > 0; i + j < n && sa[rk[i] - 1] + j < n && s[i + j] == s[sa[rk[i] - 1] + j]; )
                    ++j;
                lc[rk[i] - 1] = j;
            }
        }
    }
};
```

5.3.2 后缀数组（SA及其应用 新版）

```
struct SA {
    int n;
    vector<int> sa, rk, lc;

    SA(string s) {
        n = s.size();
        sa.resize(n);
        lc.resize(n - 1);
        rk.resize(n);
        iota(sa.begin(), sa.end(), 0);
        sort(sa.begin(), sa.end(),
            [&](int a, int b) {
                return s[a] < s[b];
            });
        rk[sa[0]] = 0;
        for (int i = 1; i < n; i++) {
            rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
        }
        int k = 1;
        vector<int> tmp, cnt(n);
        tmp.reserve(n);
```

```cpp
        while (rk[sa[n - 1]] < n - 1) {
            tmp.clear();
            for (int i = 0; i < k; i++) {
                tmp.push_back(n - k + i);
            }
            for (auto i : sa) {
                if (i >= k) {
                    tmp.push_back(i - k);
                }
            }
            fill(cnt.begin(), cnt.end(), 0);
            for (int i = 0; i < n; i++) {
                cnt[rk[i]]++;
            }
            for (int i = 1; i < n; i++) {
                cnt[i] += cnt[i - 1];
            }
            for (int i = n - 1; i >= 0; i--) {
                sa[--cnt[rk[tmp[i]]]] = tmp[i];
            }
            swap(rk, tmp);
            rk[sa[0]] = 0;
            for (int i = 1; i < n; i++) {
                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]);
            }
            k *= 2;
        }
        for (int i = 0, j = 0; i < n; i++) {
            if (rk[i] == 0) {
                j = 0;
            } else {
                for (j -= j > 0; i + j < n && sa[rk[i] - 1] + j < n && s[i + j] == s[sa[rk[i] - 1] + j]; ) {
                    j++;
                }
                lc[rk[i] - 1] = j;
            }
        }
    }
};

void solve() {
    constexpr int K = 21;
    vector st(K, vector<int>(l - 1));
    st[0] = lc;
    for (int j = 0; j < K - 1; j++) {
        for (int i = 0; i + (2 << j) <= l - 1; i++) {
            st[j + 1][i] = min(st[j][i], st[j][i + (1 << j)]);
        }
    }

    auto rmq = [&](int l, int r) {
        int k = __lg(r - l);
        return min(st[k][l], st[k][r - (1 << k)]);
    };

    auto lcp = [&](int i, int j) {
        if (i == j || i == n || j == n) {
            return min(n - i, n - j);
        }
        int a = rk[i];
        int b = rk[j];
        if (a > b) {
            swap(a, b);
        }
        return min({n - i, n - j, rmq(a, b)});
    };

    auto lcs = [&](int i, int j) {
        if (i == j || i == 0 || j == 0) {
            return min(i, j);
        }
        int a = rk[n + n - i];
        int b = rk[n + n - j];
        if (a > b) {
            swap(a, b);
        }
        return min({i, j, rmq(a, b)});
    };
}
```

5.4 后缀自动机

5.4.1 后缀自动机（SuffixAutomaton 旧版）

```
struct SuffixAutomaton {
    static constexpr int ALPHABET_SIZE = 26, N = 5e5;
    struct Node {
        int len;
        int link;
        int next[ALPHABET_SIZE];
        Node() : len(0), link(0), next{} {}
    } t[2 * N];
    int cntNodes;
    SuffixAutomaton() {
        cntNodes = 1;
        fill(t[0].next, t[0].next + ALPHABET_SIZE, 1);
        t[0].len = -1;
    }
    int extend(int p, int c) {
        if (t[p].next[c]) {
            int q = t[p].next[c];
            if (t[q].len == t[p].len + 1)
                return q;
            int r = ++cntNodes;
            t[r].len = t[p].len + 1;
            t[r].link = t[q].link;
            copy(t[q].next, t[q].next + ALPHABET_SIZE, t[r].next);
            t[q].link = r;
            while (t[p].next[c] == q) {
                t[p].next[c] = r;
                p = t[p].link;
            }
            return r;
        }
        int cur = ++cntNodes;
        t[cur].len = t[p].len + 1;
        while (!t[p].next[c]) {
            t[p].next[c] = cur;
            p = t[p].link;
        }
        t[cur].link = extend(p, c);
        return cur;
    }
};
```

5.4.2 后缀自动机（SAM 新版）

```cpp
struct SAM {
    static constexpr int ALPHABET_SIZE = 26;
    struct Node {
        int len;
        int link;
        array<int, ALPHABET_SIZE> next;
        Node() : len{}, link{}, next{} {}
    };
    vector<Node> t;
    SAM() {
        init();
    }
    void init() {
        t.assign(2, Node());
        t[0].next.fill(1);
        t[0].len = -1;
    }
    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }
    int extend(int p, int c) {
        if (t[p].next[c]) {
            int q = t[p].next[c];
            if (t[q].len == t[p].len + 1) {
                return q;
            }
            int r = newNode();
            t[r].len = t[p].len + 1;
            t[r].link = t[q].link;
            t[r].next = t[q].next;
            t[q].link = r;
            while (t[p].next[c] == q) {
                t[p].next[c] = r;
                p = t[p].link;
            }
            return r;
        }
        int cur = newNode();
        t[cur].len = t[p].len + 1;
        while (!t[p].next[c]) {
            t[p].next[c] = cur;
            p = t[p].link;
        }
        t[cur].link = extend(p, c);
        return cur;
    }
    int extend(int p, char c, char offset = 'a') {
        return extend(p, c - offset);
    }

    int next(int p, int x) {
        return t[p].next[x];
    }

    int next(int p, char c, char offset = 'a') {
        return next(p, c - 'a');
    }

    int link(int p) {
        return t[p].link;
    }

    int len(int p) {
        return t[p].len;
    }

    int size() {
        return t.size();
    }
};
```

5.5 回文自动机（PAM）

```cpp
struct PAM {
    static constexpr int ALPHABET_SIZE = 26;
    struct Node {
        int len;
        int link;
        int cnt;
        array<int, ALPHABET_SIZE> next;
        Node() : len{}, link{}, cnt{}, next{} {}
    };
    vector<Node> t;
    int suff;
    string s;
    PAM() {
        init();
    }
    void init() {
        t.assign(2, Node());
        t[0].len = -1;
        suff = 1;
        s.clear();
    }
    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }
    bool add(char c) {
        int pos = s.size();
        s += c;
        int let = c - 'a';
        int cur = suff, curlen = 0;
        while (true) {
            curlen = t[cur].len;
            if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
                break;
            }
            cur = t[cur].link;
        }
        if (t[cur].next[let]) {
            suff = t[cur].next[let];
            return false;
        }
        int num = newNode();
        suff = num;
        t[num].len = t[cur].len + 2;
        t[cur].next[let] = num;
        if (t[num].len == 1) {
            t[num].link = 1;
            t[num].cnt = 1;
            return true;
        }
        while (true) {
            cur = t[cur].link;
            curlen = t[cur].len;
            if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
                t[num].link = t[cur].next[let];
                break;
            }
        }
        t[num].cnt = 1 + t[t[num].link].cnt;
        return true;
    }
    int next(int p, int x) {
        return t[p].next[x];
    }
    int link(int p) {
        return t[p].link;
    }
    int len(int p) {
        return t[p].len;
    }
    int size() {
        return t.size();
    }
};
```

5.6 AC自动机

5.6.1 AC自动机（AC 旧版）

```
constexpr int N = 3e5 + 30, A = 26;

struct Node {
    int fail;
    int sum;
    int next[A];
    Node() : fail(-1), sum(0) {
        memset(next, -1, sizeof(next));
    }
} node[N];

int cnt = 0;
int bin[N];
int nBin = 0;

int newNode() {
    int p = nBin > 0 ? bin[--nBin] : cnt++;
    node[p] = Node();
    return p;
}

struct AC {
    vector<int> x;
    AC(AC &&a) : x(move(a.x)) {}
    AC(vector<string> s, vector<int> w) {
        x = {newNode(), newNode()};
        fill(node[x[0]].next, node[x[0]].next + A, x[1]);
        node[x[1]].fail = x[0];

        for (int i = 0; i < int(s.size()); i++) {
            int p = x[1];
            for (int j = 0; j < int(s[i].length()); j++) {
                int c = s[i][j] - 'a';
                if (node[p].next[c] == -1) {
                    int u = newNode();
                    x.push_back(u);
                    node[p].next[c] = u;
                }
                p = node[p].next[c];
            }
            node[p].sum += w[i];
        }

        queue<int> que;
        que.push(x[1]);
        while (!que.empty()) {
            int u = que.front();
            que.pop();
            node[u].sum += node[node[u].fail].sum;
            for (int c = 0; c < A; c++) {
                if (node[u].next[c] == -1) {
                    node[u].next[c] = node[node[u].fail].next[c];
                } else {
                    node[node[u].next[c]].fail = node[node[u].fail].next[c];
                    que.push(node[u].next[c]);
                }
            }
        }
    }
    ~AC() {
        for (auto p : x) {
            bin[nBin++] = p;
        }
    }
    i64 query(const string &s) const {
        i64 ans = 0;
        int p = x[1];
        for (int i = 0; i < int(s.length()); i++)  {
            int c = s[i] - 'a';
            p = node[p].next[c];
            ans += node[p].sum;
        }
        return ans;
    }
};
```

5.6.2 AC自动机（AhoCorasick, with vector 新版）

```
struct AhoCorasick {
    static constexpr int ALPHABET = 26;
    struct Node {
        int len;
        int link;
        array<int, ALPHABET> next;
        Node() : link{}, next{} {}
    };

    vector<Node> t;

    AhoCorasick() {
        init();
    }

    void init() {
        t.assign(2, Node());
        t[0].next.fill(1);
        t[0].len = -1;
    }

    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }

    int add(const vector<int> &a) {
        int p = 1;
        for (auto x : a) {
            if (t[p].next[x] == 0) {
                t[p].next[x] = newNode();
                t[t[p].next[x]].len = t[p].len + 1;
            }
            p = t[p].next[x];
        }
        return p;
    }

    int add(const string &a, char offset = 'a') {
        vector<int> b(a.size());
        for (int i = 0; i < a.size(); i++) {
            b[i] = a[i] - offset;
        }
        return add(b);
    }

    void work() {
        queue<int> q;
        q.push(1);

        while (!q.empty()) {
            int x = q.front();
            q.pop();

            for (int i = 0; i < ALPHABET; i++) {
                if (t[x].next[i] == 0) {
                    t[x].next[i] = t[t[x].link].next[i];
                } else {
                    t[t[x].next[i]].link = t[t[x].link].next[i];
                    q.push(t[x].next[i]);
                }
            }
        }
    }

    int next(int p, int x) {
        return t[p].next[x];
    }

    int next(int p, char c, char offset = 'a') {
        return next(p, c - 'a');
    }

    int link(int p) {
        return t[p].link;
    }

    int len(int p) {
        return t[p].len;
    }

    int size() {
        return t.size();
    }
};
```

5.6.3 AC自动机（AhoCorasick, with string 新版）

```
struct AhoCorasick {
    static constexpr int ALPHABET = 26;
    struct Node {
        int len;
        int link;
        array<int, ALPHABET> next;
        Node() : len{0}, link{0}, next{} {}
    };

    vector<Node> t;

    AhoCorasick() {
        init();
    }

    void init() {
        t.assign(2, Node());
        t[0].next.fill(1);
        t[0].len = -1;
    }

    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }

    int add(const string &a) {
        int p = 1;
        for (auto c : a) {
            int x = c - 'a';
            if (t[p].next[x] == 0) {
                t[p].next[x] = newNode();
                t[t[p].next[x]].len = t[p].len + 1;
            }
            p = t[p].next[x];
        }
        return p;
    }

    void work() {
        queue<int> q;
        q.push(1);

        while (!q.empty()) {
            int x = q.front();
            q.pop();

            for (int i = 0; i < ALPHABET; i++) {
                if (t[x].next[i] == 0) {
                    t[x].next[i] = t[t[x].link].next[i];
                } else {
                    t[t[x].next[i]].link = t[t[x].link].next[i];
                    q.push(t[x].next[i]);
                }
            }
        }
    }

    int next(int p, int x) {
        return t[p].next[x];
    }

    int link(int p) {
        return t[p].link;
    }

    int len(int p) {
        return t[p].len;
    }

    int size() {
        return t.size();
    }
};
```

5.7 字符串哈希（随机底模例题）

```
#include <bits/stdc++.h>

using i64 = long long;

bool isprime(int n) {
    if (n <= 1) {
        return false;
    }
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}

int findPrime(int n) {
    while (!isprime(n)) {
        n++;
    }
    return n;
}

using Hash = array<int, 2>;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

    const int P = findPrime(rng() % 900000000 + 100000000);

    string s, x;
    cin >> s >> x;

    int n = s.length();
    int m = x.length();

    vector<int> h(n + 1), p(n + 1);
    for (int i = 0; i < n; i++) {
```

```cpp
        h[i + 1] = (10LL * h[i] + s[i] - '0') % P;
    }
    p[0] = 1;
    for (int i = 0; i < n; i++) {
        p[i + 1] = 10LL * p[i] % P;
    }

    auto get = [&](int l, int r) {
        return (h[r] + 1LL * (P - h[l]) * p[r - l]) % P;
    };

    int px = 0;
    for (auto c : x) {
        px = (10LL * px + c - '0') % P;
    }

    for (int i = 0; i <= n - 2 * (m - 1); i++) {
        if ((get(i, i + m - 1) + get(i + m - 1, i + 2 * m - 2)) % P == px) {
            cout << i + 1 << " " << i + m - 1 << "\n";
            cout << i + m << " " << i + 2 * m - 2 << "\n";
            return 0;
        }
    }

    vector<int> z(m + 1), f(n + 1);
    z[0] = m;

    for (int i = 1, j = -1; i < m; i++) {
        if (j != -1) {
            z[i] = max(0, min(j + z[j] - i, z[i - j]));
        }
        while (z[i] + i < m && x[z[i]] == x[z[i] + i]) {
            z[i]++;
        }
        if (j == -1 || i + z[i] > j + z[j]) {
            j = i;
        }
    }
    for (int i = 0, j = -1; i < n; i++) {
        if (j != -1) {
            f[i] = max(0, min(j + f[j] - i, z[i - j]));
        }
        while (f[i] + i < n && f[i] < m && x[f[i]] == s[f[i] + i]) {
            f[i]++;
        }
        if (j == -1 || i + f[i] > j + f[j]) {
            j = i;
        }
    }

    for (int i = 0; i + m <= n; i++) {
        int l = min(m, f[i]);

        for (auto j : { m - l, m - l - 1 }) {
            if (j <= 0) {
                continue;
            }
            if (j <= i && (get(i - j, i) + get(i, i + m)) % P == px) {
                cout << i - j + 1 << " " << i << "\n";
                cout << i + 1 << " " << i + m << "\n";
                return 0;
            }
            if (i + m + j <= n && (get(i, i + m) + get(i + m, i + m + j)) % P ==
px) {
```

```
                cout << i + 1 << " " << i + m << "\n";
                cout << i + m + 1 << " " << i + m + j << "\n";
                return 0;
            }
        }
    }

    return 0;
}
```

5.8 最长公共前缀 LCP（例题）

```
constexpr int L = 2E6 + 10;

int len[L];
int lnk[L];
int nxt[L][26];

int f[L];
int tot = 1;

vector<int> adj[L];

int extend(int p, int c) {
    if (nxt[p][c]) {
        int q = nxt[p][c];
        if (len[q] == len[p] + 1) {
            return q;
        }
        int r = ++tot;
        len[r] = len[p] + 1;
        lnk[r] = lnk[q];
        copy(nxt[q], nxt[q] + 26, nxt[r]);
        lnk[q] = r;
        while (nxt[p][c] == q) {
            nxt[p][c] = r;
            p = lnk[p];
        }
        return r;
    }
    int cur = ++tot;
    len[cur] = len[p] + 1;
    while (!nxt[p][c]) {
        nxt[p][c] = cur;
        p = lnk[p];
    }
    lnk[cur] = extend(p, c);
    return cur;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    fill(nxt[0], nxt[0] + 26, 1);
    len[0] = -1;

    int N;
    cin >> N;

    vector<string> S(N);
    for (int i = 0; i < N; i++) {
```

```cpp
        cin >> S[i];
        int p = 1;
        for (auto c : S[i]) {
            p = extend(p, c - 'a');
            if (f[p] != -1) {
                if (f[p] == 0) {
                    f[p] = i + 1;
                } else if (f[p] != i + 1) {
                    f[p] = -1;
                }
            }
        }
    }

    for (int i = 1; i <= tot; i++) {
        adj[lnk[i]].push_back(i);
    }
}
```

5.9 字典树 Trie

```cpp
constexpr i64 inf = 1E18;

constexpr int N = 1E6 + 10;

int trie[N][26];
int tot;

int newNode() {
    tot++;
    fill(trie[tot], trie[tot] + 26, 0);
    val[tot] = inf;
    return tot;
}

void solve() {
    //* init
    tot = 0;
    newNode();

    //* insert
    for (int i = 0; i < N; i++) {
        int p = 1;
        int l = S[i].size();
        for (int j = 0; j < l; j++) {
            int x = S[i][j] - 'a';
            if (!trie[p][x]) {
                trie[p][x] = newNode();
            }
            p = trie[p][x];
            //* 处理
            //* val[p] = min(val[p], l + K + f[(K - (l - j - 1) % K) % K]);
        }
    }

    //* query
    for (int i = 0; i < L; i++) {
        int p = 1;
        for (int j = i; j < L; j++) {
            int x = T[j] - 'a';
            p = trie[p][x];
            if (!p) {
                continue;
            }
            //* 处理
            //* dp[j + 1] = min(dp[j + 1], dp[i] + val[p]);
        }
    }
}
```

```
int tot;
int trie[N][2];
int f[N];

int newNode() {
    int x = ++tot;
    trie[x][0] = trie[x][1] = 0;
    f[x] = inf;
    return x;
}
void add(int x, int i) {
    int p = 1;
    for (int j = 29; j >= 0; j--) {
        int &q = trie[p][x >> j & 1];
        if (q == 0) {
            q = newNode();
        }
        p = q;
        f[p] = min(f[p], i);
    }
}

int query(int a, int b) {
    int ans1 = inf, ans2 = inf;
    int p = 1;
    for (int i = 29; i >= 0; i--) {
        int d = a >> i & 1;
        int e = b >> i & 1;
        if (e) {
            ans1 = min(ans1, f[trie[p][d]]);
        } else {
            ans2 = min(ans2, f[trie[p][d ^ 1]]);
        }
        p = trie[p][e ^ d];
    }
    ans1 = min(ans1, f[p]);
    ans2 = min(ans2, f[p]);
    if (ans1 == inf || ans2 == inf) {
        return -1;
    }
    return max({1, ans1, ans2});
}
```

```
int trie[N][2];
int cnt[N][2];

int tot = 0;
int newNode() {
    int x = ++tot;
    trie[x][0] = trie[x][1] = 0;
    cnt[x][0] = cnt[x][1] = 0;
    return x;
}

```

```cpp
void add(int x, int d, int t = 1) {
    int p = 1;
    cnt[p][d] += t;
    for (int i = 29; i >= 0; i--) {
        int u = x >> i & 1;
        if (!trie[p][u]) {
            trie[p][u] = newNode();
        }
        p = trie[p][u];
        cnt[p][d] += t;
    }
}

int query(int x, int d) {
    int p = 1;
    if (!cnt[p][d]) {
        return 0;
    }
    int ans = 0;
    for (int i = 29; i >= 0; i--) {
        int u = x >> i & 1;
        if (cnt[trie[p][u ^ 1]][d]) {
            ans |= 1 << i;
            p = trie[p][u ^ 1];
        } else {
            p = trie[p][u];
        }
    }
    return ans;
}
```

```cpp
constexpr int N = 1E7;
constexpr int inf = 1E9;
int tot;
int trie[N][2];
int f[N];

int newNode() {
    int x = ++tot;
    trie[x][0] = trie[x][1] = 0;
    f[x] = inf;
    return x;
}
void add(int x, int i) {
    int p = 1;
    for (int j = 29; j >= 0; j--) {
        int &q = trie[p][x >> j & 1];
        if (q == 0) {
            q = newNode();
        }
        p = q;
        f[p] = min(f[p], i);
    }
}

int query(int a, int b) {
    int ans1 = inf, ans2 = inf;
    int p = 1;
    for (int i = 29; i >= 0; i--) {
        int d = a >> i & 1;
        int e = b >> i & 1;
        if (e) {
            ans1 = min(ans1, f[trie[p][d]]);
            } else {
                ans2 = min(ans2, f[trie[p][d ^ 1]]);
            }
            p = trie[p][e ^ d];
        }
        ans1 = min(ans1, f[p]);
        ans2 = min(ans2, f[p]);
        if (ans1 == inf || ans2 == inf) {
            return -1;
        }
        return max({1, ans1, ans2});
}
```

5.10 前缀函数（KMP）

```
vector<int> kmp(string s) {
    int n = s.size();
    vector<int> f(n + 1);
    for (int i = 1, j = 0; i < n; i++) {
        while (j && s[i] != s[j]) {
            j = f[j];
        }
        j += (s[i] == s[j]);
        f[i + 1] = j;
    }
    return f;
}
```

/END/

[illegible]

Author by ***Jiangly***
Tyrp by ***Wida***

Version: 1.8.7 (2024.10.07)

github.com/hh2048
cnblogs.com/WIDA