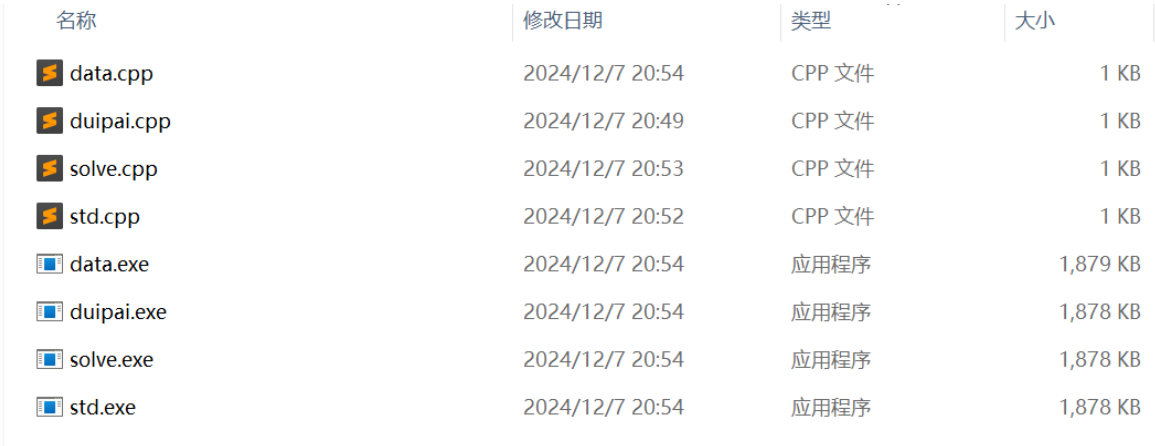
Basic

freopen

```
#ifdef LOCAL
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
#endif
```

对拍



名称	修改日期	类型	大小
data.cpp	2024/12/7 20:54	CPP 文件	1 KB
duipai.cpp	2024/12/7 20:49	CPP 文件	1 KB
solve.cpp	2024/12/7 20:53	CPP 文件	1 KB
std.cpp	2024/12/7 20:52	CPP 文件	1 KB
data.exe	2024/12/7 20:54	应用程序	1,879 KB
duipai.exe	2024/12/7 20:54	应用程序	1,878 KB
solve.exe	2024/12/7 20:54	应用程序	1,878 KB
std.exe	2024/12/7 20:54	应用程序	1,878 KB

data.cpp

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    srand(time(0));
    int a = rand() % 5;
    int b = rand() % 5;
    cout << a << ' ' << b << endl;

    return 0;
}
```

duipai.cpp

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int t = 0;
    while (true) {
        cout << "test: " << t++ << endl;
        system("data.exe > data.in");
        system("std.exe < data.in > std.out");
```

```
        system("solve.exe < data.in > solve.out");
        if (system("fc std.out solve.out > diff.log")) {
            cout << "WA\n";
            break;
        }
        cout << "AC\n";
    }

    return 0;
}
```

Discrete

```
template <typename T> struct Discrete {
    vector<T> p;

    Discrete() {}

    void add(T t) {
        p.push_back(t);
    }
    void init() {
        sort(p.begin(), p.end());
        p.resize(unique(p.begin(), p.end()) - p.begin());
    }
    int size() {
        return p.size();
    }
    int id(T t) {
        return lower_bound(p.begin(), p.end(), t) - p.begin();
    }
    T operator[](int id) {
        return p[id];
    }
    vector<T> &get() {
        return p;
    }
};
```

二维前缀和&查分

```
// 前缀和
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
        cin >> a[i][j];
        s[i][j] = s[i - 1][j] + s[i][j - 1] - s[i - 1][j - 1] + a[i][j];
    }
}
while (q--) {
    int x1, y1, x2, y2; cin >> x1 >> y1 >> x2 >> y2;
```

```
    cout << s[x2][y2] - s[x2][y1 - 1] - s[x1 - 1][y2] + s[x1 - 1][y1 - 1] <<
"\n";
}

// 差分
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
        cin >> a[i][j];
        b[i][j] = a[i][j] - a[i - 1][j] - a[i][j - 1] + a[i - 1][j - 1];
        // b[i][j] += a[i][j]; b[i + 1][j + 1] += a[i][j];
        // b[i][j + 1] -= a[i][j]; b[i + 1][j] -= a[i][j];
    }
}
while (q--) {
    int x1, y1, x2, y2, c; cin >> x1 >> y1 >> x2 >> y2 >> c;
    b[x1][y1] += c; b[x2 + 1][y2 + 1] += c;
    b[x2 + 1][y1] -= c; b[x1][y2 + 1] -= c;
}
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
        b[i][j] += b[i - 1][j] + b[i][j - 1] - b[i - 1][j - 1];
        cout << b[i][j] << " \n"[j == m];
    }
}
```

区间合并

```
void merge(vector<PII> &segs) {
    vector<PII> res;
    sort(segs.begin(), segs.end());
    int st = -2e9, ed = -2e9;
    for (auto seg : segs) {
        if (ed < seg.first) {
            if (st != -2e9) res.push_back({st, ed});
            st = seg.first, ed = seg.second;
        }
        else ed = max(ed, seg.second);
    }
    if (st != -2e9) res.push_back({st, ed});
    segs = res;
}
```

DataStructure

DSU

```
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;
        }
        // 启发式合并：
        if (siz[x] <= siz[y]) {
            swap(x, y);
        }
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }
    int size(int x) {
        return siz[find(x)];
    }
};
```

Fenwick

> idx0 :

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

```
    vector<T> a;

    Fenwick(int n_ = 0) {
        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 res{};
        for (int i = x; i > 0; i -= i & -i) {
            res = res + a[i - 1];
        }
        return res;
    }
    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;
    }
};
```

idx1 :

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

    Fenwick(int n_ = 0) {
        n = n_;
        a.assign(n + 1, T{});
    }

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

```
        }
        return res;
    }
    T query(int l, int r) {
        return sum(r) - sum(l - 1);
    }
};
```

SparesTable

> idx0 :

```
template <typename T, typename Compare = function<T(const T &, const T &)>>
struct SparseTable {
    int n;
    vector<vector<T>> ST;
    Compare comp;

    SparseTable(const vector<T> &v): SparseTable(v, [](const T &a, const T &b) {
        return max(a, b);
    }) {}
    SparseTable(const vector<T> &v, const Compare &f): comp(f) {
        n = v.size();
        ST.assign(n, vector<T>(__lg(n) + 1));
        for (int i = 0; i < n; i++) {
            ST[i][0] = v[i];
        }
        for (int j = 1; j <= __lg(n); j++) {
            for (int i = 0; i + (1 << j) - 1 < n; i++) {
                ST[i][j] = comp(ST[i][j - 1], ST[i + (1 << (j - 1))][j - 1]);
            }
        }
    }

    T query(int l, int r) {
        int j = __lg(r - l + 1);
        return comp(ST[l][j], ST[r - (1 << j) + 1][j]);
    }
};
```

> idx1 :

```
template <typename T, typename Compare = function<T(const T &, const T &)>>
struct SparseTable {
    int n;
    vector<vector<T>> ST;
    Compare comp;

    SparseTable(const vector<T> &v): SparseTable(v, [](const T &a, const T &b) {
        return max(a, b);
    }) {}
    SparseTable(const vector<T> &v, const Compare &f): comp(f) {
```

```
        n = v.size();
        ST.assign(n + 1, vector<T>(__lg(n) + 1));
        for (int i = 1; i <= n; i++) {
            ST[i][0] = v[i];
        }
        for (int j = 1; j <= __lg(n); j++) {
            for (int i = 1; i + (1 << j) - 1 <= n; i++) {
                ST[i][j] = comp(ST[i][j - 1], ST[i + (1 << (j - 1))][j - 1]);
            }
        }
    }

    T query(int l, int r) {
        int j = __lg(r - l + 1);
        return comp(ST[l][j], ST[r - (1 << j) + 1][j]);
    }
};
```

SegTree

```
template <class info> struct segtree {
#define ls u << 1
#define rs u << 1 | 1
    struct node {
        int l, r;
        info data;
    };

    int n; vector<node> tr;
    segtree(): n(0) {};
    template<typename T> segtree(int n, T v): segtree(vector<T>(n, v)) {}
    template<typename T> segtree(vector<T> a): n(a.size()), tr(n * 4) {
        a.resize(n + 1);
        for (int i = n; i >= 1; i--) a[i] = a[i - 1];
        function<void(int, int, int)> build = [&](int u, int l, int r) -> void {
            tr[u].l = l;
            tr[u].r = r;
            if (l == r) {
                tr[u].data = info(a[l]);
                return;
            }
            int mid = (l + r) / 2;
            build(ls, l, mid);
            build(rs, mid + 1, r);
            pushup(u);
        };
        build(1, 1, n);
    }

    void pushup(int u) {
        tr[u].data = tr[ls].data + tr[rs].data;
    }
```

```
    void set(int u, int pos, const info &v) {
        if (tr[u].l == tr[u].r) {
            tr[u].data = v;
            return;
        }
        int mid = (tr[u].l + tr[u].r) / 2;
        if (pos <= mid) set(ls, pos, v);
        else set(rs, pos, v);
        pushup(u);
    }

    void set(int pos, const info &v) {
        set(1, pos, v);
    }

    info qry(int u, int l, int r) {
        if (l <= tr[u].l && tr[u].r <= r) {
            return tr[u].data;
        }
        int mid = (tr[u].l + tr[u].r) / 2;
        if (r <= mid) return qry(ls, l, r);
        if (l >= mid + 1) return qry(rs, l, r);
        return qry(ls, l, r) + qry(rs, l, r);
    }

    info qry(int l, int r) {
        return qry(1, l, r);
    }

#undef ls
#undef rs
};

struct info {

};
info operator+(const info &a, const info &b) {

}
```

LazySegTree

```
template <class info, class tag> struct lazysegtree {
#define ls u << 1
#define rs u << 1 | 1
    struct node {
        int l, r;
        info data;
        tag tg;
    };

    int n; vector<node> tr;
    lazysegtree(): n(0) {};
```

```
    template<typename T> lazysegtree(int n, T v): lazysegtree(vector<T>(n, v)) {}
    template<typename T> lazysegtree(vector<T> a): n(a.size()), tr(n * 4) {
        a.resize(n + 1);
        for (int i = n; i >= 1; i--) a[i] = a[i - 1];
        function<void(int, int, int)> build = [&](int u, int l, int r) -> void {
            tr[u].l = l;
            tr[u].r = r;
            if (l == r) {
                tr[u].data = info(a[l]);
                return;
            }
            int mid = (l + r) / 2;
            build(ls, l, mid);
            build(rs, mid + 1, r);
            pushup(u);
        };
        build(1, 1, n);
    }

    void pushup(int u) {
        tr[u].data = tr[ls].data + tr[rs].data;
    }

    void apply(int u, const tag &v) {
        tr[u].data.apply(v);
        tr[u].tg.apply(v);
    }

    void pushdown(int u) {
        apply(ls, tr[u].tg);
        apply(rs, tr[u].tg);
        tr[u].tg = tag();
    }

    void set(int u, int pos, const info &v) {
        if (tr[u].l == tr[u].r) {
            tr[u].data = v;
            return;
        }
        int mid = (tr[u].l + tr[u].r) / 2;
        pushdown(u);
        if (pos <= mid) set(ls, pos, v);
        else set(rs, pos, v);
        pushup(u);
    }

    void set(int pos, const info &v) {
        set(1, pos, v);
    }

    void modify(int u, int l, int r, const tag &v) {
        if (l <= tr[u].l && tr[u].r <= r) {
            apply(u, v);
            return;
        }
        int mid = (tr[u].l + tr[u].r) / 2;
```

```
        pushdown(u);
        if (l <= mid) modify(ls, l, r, v);
        if (r >= mid + 1) modify(rs, l, r, v);
        pushup(u);
    }

    void modify(int l, int r, const tag &v) {
        modify(1, l, r, v);
    }

    info qry(int u, int l, int r) {
        if (l <= tr[u].l && tr[u].r <= r) {
            return tr[u].data;
        }
        int mid = (tr[u].l + tr[u].r) / 2;
        pushdown(u);
        if (r <= mid) return qry(ls, l, r);
        if (l >= mid + 1) return qry(rs, l, r);
        return qry(ls, l, r) + qry(rs, l, r);
    }

    info qry(int l, int r) {
        return qry(1, l, r);
    }

#undef ls
#undef rs
};

struct tag {

};
struct info {

};
info operator+(const info &a, const info &b) {

}
```

Tire

```
struct Tire {
    int idx;
    vector<array<int, 26>> nxt;
    vector<int> isend;
    Tire() {}
    Tire(int n) {
        init(n);
    }
    void init(int n) {
        idx = 0;
        nxt.assign(n, {});
        isend.assign(n, {});
```

```
    }
    void insert(string s) {
        int n = s.size();
        int now = 0;
        for (int i = 0; i < n; i++) {
            int x = s[i] - 'a';
            if (!nxt[now][x]) nxt[now][x] = ++idx;
            now = nxt[now][x];
        }
        isend[now] = true;
    }
    bool qry(string s) {
        int n = s.size();
        int now = 0;
        for (int i = 0; i < n; i++) {
            int x = s[i] - 'a';
            if (!nxt[now][x]) return false;
            now = nxt[now][x];
        }
        return isend[now];
    }
    // 字典序输出字符串
    array<int, 105> a;
    void dfs(int now, int depth) {
        if (isend[now]) {
            for (int i = 0; i < depth; i++) {
                cout << (char)(a[i] + 'a');
            }
            cout << "\n";
        }
        for (int i = 0; i < 26; i++) {
            if (nxt[now][i]) {
                a[depth] = i;
                dfs(nxt[now][i], depth + 1);
            }
        }
    }
    void dfs() {
        dfs(0, 0);
    }
};
```

Graph

Dijkstra

```
struct edge {
    int v, w;
};
vector<edge> adj[150005];
int dist[150005];
bool st[150005];

void dijstra() {
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;

    priority_queue<pii, vector<pii>, greater<pii>> q;
    q.emplace(0, 1);

    while (!q.empty()) {
        auto [d, u] = q.top(); q.pop();
        if (st[u]) continue;
        st[u] = true;
        for (auto [v, w] : adj[u]) {
            if (d + w < dist[v]) {
                dist[v] = d + w;
                q.emplace(dist[v], v);
            }
        }
    }
}
// Judge: (dist[n] == 0x3f3f3f3f ? -1 : dist[n])
```

Floyd

```
void floyd() {
    for (int k = 1; k <= n; k ++ )
        for (int i = 1; i <= n; i ++ )
            for (int j = 1; j <= n; j ++ )
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
}
// Init:
for (int i = 1; i <= n; i ++ ) for (int j = 1; j <= n; j ++ )
    if (i == j) d[i][j] = 0;
    else d[i][j] = INF;
// Judge:
d[a][b] if d[a][b] <= INF / 2 else "possbile"
```

TopoSort

```
int n, m; cin >> n >> m;
vector<vector<int>> e(n);
vector<int> deg(n);
for (int i = 0; i < m; i++) {
    int a, b; cin >> a >> b;
    a --, b --;
    e[a].push_back(b);
    deg[b]++;
}
int hh = 0, tt = -1;
for (int i = 0; i < n; i++) {
    if (!deg[i]) {
        q[++tt] = i;
    }
}
while (hh <= tt) {
    int u = q[hh++];
    for (auto x : e[u]) {
        if (--deg[x] == 0) {
            q[++tt] = x;
        }
    }
}
if (tt == n - 1) {
    for (int i = 0; i < n; i++) {
        cout << q[i] + 1 << " \n"[i == n - 1];
    }
} else {
    cout << -1 << endl;
}
```

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];
    }

    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);
    }
};
```

String

Hash

```
// 单模hash

const int P = 13331;
const i64 hash_mod = 1610612741;

    string s; s = ' ' + s;
    vector<int> h(n + 1), p(n + 1);
    p[0] = 1;
    for (int i = 1; i <= n; i++) {
        p[i] = p[i - 1] * P % hash_mod;
        h[i] = (h[i - 1] * P + s[i]) % hash_mod;
    }

    auto get = [&](int l, int r, vector<int> &h) -> i64 {
        return ((h[r] - h[l - 1] * p[r - l + 1]) % mod + mod) % mod;
    };
```

StringHash

```
std::mt19937 rng(std::chrono::steady_clock::now().time_since_epoch().count());
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;
}
template <int N> struct StringHash {
    static array<int, N> hash_mod;
    static array<int, N> base;
    vector<array<int, N>> p, h;
    StringHash() {};
    StringHash(const string &s) {
        int n = s.size();
        p.resize(n);
        h.resize(n);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < N; j++) {
                p[i][j] = 1LL * (i == 0 ? 1LL : p[i - 1][j]) * base[j] %
hash_mod[j];
                h[i][j] = (1LL * (i == 0 ? 0LL : h[i - 1][j]) * base[j] + s[i]) %
hash_mod[j];
            }
```

```
        }
    }
    array<int, N> get(int l, int r) {
        assert(r >= l);
        array<int, N> res{};
        for (int i = 0; i < N; i++) {
            res[i] = (h[r][i] - 1LL * (l == 0 ? 0LL : h[l - 1][i]) * p[r - l][i]
% hash_mod[i] + hash_mod[i]) % hash_mod[i];
        }
        return res;
    }
    bool same(int l1, int r1, int l2, int r2) {
        auto res1 = get(l1, r1);
        auto res2 = get(l2, r2);
        for (int i = 0; i < N; i++) {
            if (res1[i] != res2[i]) return false;
        }
        return true;
    }
};
constexpr int HN = 2;
template<> array<int, HN> StringHash<HN>::hash_mod = {
    findPrime(rng() % 900000000 + 100000000),
    findPrime(rng() % 900000000 + 100000000)
};
template<> array<int, HN> StringHash<HN>::base = {13331, 131};
using Hashing = StringHash<HN>;
```

Manacher

```
// Max Palindrome String: s.substr((i - p[i]) / 2, p[i] - 1) <=> [(i - p[i]) / 2,
(i - p[i]) / 2 + p[i] - 1 - 1];
vector<int> manacher(string s) {
    string t = "$#";
    for (auto c : s) {
        t += c;
        t += '#';
    }
    t += "^";
    int n = t.size();
    vector<int> r(n);
    int mid = 0, Rmax = 0;
    for (int i = 0; i < n; i++) {
        if (i < Rmax) r[i] = min(r[2 * mid - i], Rmax - i);
        while (i - r[i] >= 0 && i + r[i] < n && t[i - r[i]] == t[i + r[i]])
r[i]++;
        if (i + r[i] > Rmax) {
            mid = i;
            Rmax = i + r[i];
        }
    }
    return r;
```

```
}
```

Z_Function

```
vector<int> Z(string s) { // (suf of s) & s
    int n = s.size();
    vector<int> z(n);
    z[0] = n; //rule:z[0] = n
    for (int i = 1, l = 0, r = 0; i < n; i++) {
        if (i <= r) z[i] = min(z[i - l], r - i + 1); //s[i, r] = s[i-l, r-l]
        while (i + z[i] < n && s[i + z[i]] == s[z[i]]) z[i]++; //force
        if (i + z[i] - 1 > r) { //update[l, r]
            l = i;
            r = i + z[i] - 1;
        }
    }
    return z;
}

vector<int> P(string s, string t) { // lcp: (suff of s) & t
    auto z = Z(t); // t's pref, so Z(t), careful:TLE
    int n = s.size(), m = t.size();
    vector<int> p(n);
    for (int i = 0, l = 0, r = 0; i < n; i++) {
        if (i != 0 && i <= r) p[i] = min(z[i - l], r - i + 1); // s[i, r] = t[i-
l, r-l]
        while (i + p[i] < n && p[i] < m && s[i + p[i]] == t[p[i]]) p[i]++;
        if (i + p[i] - 1 > r) {
            l = i;
            r = i + p[i] - 1;
        }
    }
    return p;
}
```

```
template <class T> struct z_function {
    const T s; /// start-hash
    int n;
    vector<int> z;

    z_function(const T &s) : s(s), n(s.size()), z(n) {
        z[0] = n;
        int l = 0, r = 0;
        for (int i = 1; i < n; ++i) {
            z[i] = max(0, min(z[i - l], r - i));
            while (i + z[i] < n && s[z[i]] == s[i + z[i]]) z[i]++;
            if (i + z[i] > r) {
                l = i;
                r = i + z[i];
            }
```

```
        }
    } /// end-hash

    vector<int> cal(const T &t) { /// start-hash
        int m = t.size();
        vector<int> res(m);
        int l = 0, r = 0;
        for (int i = 0; i < m; ++i) {
            res[i] = max(0, min(i - l < n ? z[i - l] : 0, r - i));
            while (i + res[i] < m && s[res[i]] == t[i + res[i]]) res[i]++;
            if (i + res[i] > r) {
                l = i;
                r = i + res[i];
            }
        }
        return res;
    } /// end-hash
};
```

SuffixArray

```
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;
            }
        }
    }
};
```

Math

ksm

```
int ksm(int a, int k, int p) {
    int res = 1;
    while (k) {
        if (k & 1) res = (i64)res * a % p;
        a = (i64)a * a % p;
        k >>= 1;
    }
    return res;
}
```

Sieve

```
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;
            }
        }
    }
}
```

分解质因数

```
void divide(int n) {
    map<int, int> mp;
    for (int i = 2; i <= n / i; i++) {
        while (n % i == 0) {
            n /= i;
            mp[i] ++;
        }
    }
    if (n > 1) mp[n] ++;
}
```

Comb

```
struct Comb {
    int n;
    vector<Mint> fac;
    vector<Mint> inv;
    vector<Mint> invfac;

    Comb(int n): fac(n + 1), inv(n + 1), invfac(n + 1) {
        fac[0] = inv[0] = invfac[0] = 1;
        for (int i = 1; i < n; i++) {
            fac[i] = fac[i - 1] * i;
            invfac[i] = invfac[i - 1] * Mint(i).inv();
        }
    }

    Mint operator()(int n, int m) {
        if (m > n || m < 0 || n < 0) return 0;
        return fac[n] * invfac[m] * invfac[n - m];
    }
};
Comb C((int)1E6 + 10);
```

Comb_Lucas

```
struct Comb {
    int ksm(int a, int k, int p) {
        int res = 1;
        while (k) {
            if (k & 1) res = (i64)res * a % p;
            k >>= 1;
            a = (i64)a * a % p;
        }
        return res;
    }
    int inv(int a, int p) {
```

```
        return ksm(a, p - 2, p);
    }
    int C(int n, int m, int p) {
        int res = 1;
        for (int i = 1, j = n; i <= m; i++, j--) {
            res = (i64)res * j % p;
            res = (i64)res * inv(i, p) % p;
        }
        return res;
    }
    int Lucas(i64 n, i64 m, int p) {
        if (n < p && m < p) {
            return C(n, m, p);
        }
        return (i64)C(n % p, m % p, p) * Lucas(n / p, m / p, p) % p;
    }
} comb;
```

Mint

```
template <unsigned M_> struct ModInt {
    static constexpr unsigned M = M_;
    unsigned x;

    constexpr ModInt() = default;
    constexpr ModInt(unsigned x_) : x(x_ % M) {}
    constexpr ModInt(unsigned long long x_) : x(x_ % M) {}
    constexpr ModInt(int x_) : x(((x_ %= static_cast<int>(M)) < 0) ? (x_ +
static_cast<int>(M)) : x_) {}
    constexpr ModInt(long long x_) : x(((x_ %= static_cast<long long>(M)) < 0) ?
(x_ + static_cast<long long>(M)) : x_) {}

    constexpr ModInt operator++() {
        (*this) += 1; return *this;
    }
    constexpr ModInt operator--() {
        (*this) -= 1; return *this;
    }
    constexpr ModInt operator++(int) {
        const ModInt temp = *this; ++(*this);
        return temp;
    }
    constexpr ModInt operator--(int) {
        const ModInt temp = *this; --(*this);
        return temp;
    }
    ModInt &operator+=(const ModInt &a) {
        x = ((x += a.x) >= M) ? (x - M) : x;
        return *this;
    }
    ModInt &operator-=(const ModInt &a) {
        x = ((x -= a.x) >= M) ? (x + M) : x;
        return *this;
```

```
    }
    ModInt &operator*=(const ModInt &a) {
        x = (static_cast<unsigned long long>(x) * a.x) % M;
        return *this;
    }
    ModInt &operator/=(const ModInt &a) {
        return (*this *= a.inv());
    }
    ModInt pow(long long e) const {
        if (e < 0) return inv().pow(-e);
        ModInt a = *this, res = 1U;
        for (; e; e >>= 1, a *= a) { if (e & 1) res *= a; }
        return res;
    }
    ModInt inv() const {
        unsigned a = M, b = x; int y = 0, z = 1;
        for (; b; ) {
            const unsigned q = a / b;
            const unsigned c = a - q * b;
            a = b; b = c;
            const int w = y - static_cast<int>(q) * z;
            y = z; z = w;
        }
        assert(a == 1U);
        return ModInt(y);
    }
    ModInt operator+() const {
        return *this;
    }
    ModInt operator-() const {
        ModInt a; a.x = x ? (M - x) : 0U;
        return a;
    }
    ModInt operator+(const ModInt &a) const {
        return (ModInt(*this) += a);
    }
    ModInt operator-(const ModInt &a) const {
        return (ModInt(*this) -= a);
    }
    ModInt operator*(const ModInt &a) const {
        return (ModInt(*this) *= a);
    }
    ModInt operator/(const ModInt &a) const {
        return (ModInt(*this) /= a);
    }
    template <class T>
    friend ModInt operator+(T a, const ModInt &b) {
        return (ModInt(a) += b);
    }
    template <class T>
    friend ModInt operator-(T a, const ModInt &b) {
        return (ModInt(a) -= b);
    }
    template <class T>
    friend ModInt operator*(T a, const ModInt &b) {
        return (ModInt(a) *= b);
```

```
    }
    template <class T>
    friend ModInt operator/(T a, const ModInt &b) {
        return (ModInt(a) /= b);
    }
    explicit operator bool() const {
        return x;
    }
    bool operator==(const ModInt &a) const {
        return (x == a.x);
    }
    bool operator!=(const ModInt &a) const {
        return (x != a.x);
    }
    bool operator<(const ModInt &a) const {
        return (x < a.x);
    }
    bool operator>(const ModInt &a) const {
        return (x > a.x);
    }
    friend std::ostream &operator<<(std::ostream &os, const ModInt &a) {
        return os << a.x;
    }
    friend std::istream &operator>>(std::istream &is, ModInt &a) {
        return is >> a.x;
    }
};
constexpr int Mod = 1e9 + 7;
// constexpr int Mod = 998244353;
using mint = ModInt<Mod>;
```