EOF
Universidad de la Amazonia, Colombia

Diego, Rafael e Jaiderbr

3 de septiembre de 2025

Indice

1. DataStructure

1.1. indexed set

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

template<class T> using T_set = tree<T, null_type, less<T>,
    rb_tree_tag, tree_order_statistics_node_update>;
template<class L> using T_multiset = tree<L, null_type, less_equal<L>,
    rb_tree_tag, tree_order_statistics_node_update>;

/*
st.find_by_order(k) - Retorna un iterador al k-esimo elemento, >= sz
    .end()
st.order_of_key(k)  - Retorna el numero de elementos estrictamente
    menores que k
st.lower_bound(k)   - Retorna un iterador al primer elemento >= k, >=
    sz .end()
st.upper_bound(k)   - Retorna un iterador al primer elemento > k, >=
    sz .end()
st.erase(k)         - Elimina el elemento k
st.erase_if([](int x) { return x % 2 == 0; }); - Elimina todos los
    elementos que cumplan la condicion

T_set<int> st;
T_multiset<int> mst;
*/
```

1.2. inversion count

```
// Computa el numero de inversiones para transformar
// l en r (si no es posible, retorna -1)

template<typename T> int inversion_count(vector<T> l, vector<T> r =
    {}) {
    if (!sz(r)) {
        r = l;
        sort(all(r));
    }
    int n = sz(l);
    vector<int> v(n), bit(n);
    vector<pair<T, int>> w;
    forn(i, n) w.pb({ r[i], i + 1 });
    sort(all(w));
    forn(i, n) {
        auto it = lower_bound(w.begin(), w.end(), make_pair(l[i],
            int(0)));
        if (it == w.end() or it->first != l[i]) return -1; // no da
        v[i] = it->second;
        it->second = -1;
    }

    int ans = 0;
    for (int i = n - 1; i >= 0; i--) {
        for (int j = v[i] - 1; j; j -= j & -j) ans += bit[j];
        for (int j = v[i]; j < n; j += j & -j) bit[j]++;
    }

    return ans;
}
```

1.3. Min queue deque

```
// para max negar...
template<class T> struct Queue {
    deque<pair<T, int>> q;

    void push(T x) {
        int ct = 1;
        while (sz(q) and x < q.front().first)
            ct += q.front().second, q.pop_front();
        q.emplace_front(x, ct);
    }
```

```
    void pop() {
        if (q.back().second > 1) q.back().second--;
        else q.pop_back();
    }
    T min() { return q.back().first; }
};
```

1.4. Min queue stack

```
// para max negar...
template<class T> struct Stack {
    stack<pair<T, T>> s;

    void push(T x) {
        if (!sz(s)) s.push({ x, x });
        else s.emplace(x, min(s.top().second, x));
    }
    T top() { return s.top().first; }
    T pop() {
        T ans = s.top().first;
        s.pop();
        return ans;
    }
    int size() { return sz(s); }
    T min() { return s.top().second; }
};
```

1.5. Dsu

```
struct dsu {
    vector<int> pad, tam;
    int size;

    dsu(int n) : pad(n), tam(n, 1), size(n) {
        iota(all(pad), 0);
    }

    void make() {
        pad.pb(sz(pad));
        tam.pb(1);
        size++;
    }

    int find(int v) {
        if (v == pad[v]) return v;
        return pad[v] = find(pad[v]);
    }

    void unite(int a, int b) {
        a = find(a);
        b = find(b);
        if (a != b) {
            if (tam[a] < tam[b]) swap(a, b);
            pad[b] = a;
            tam[a] += tam[b];
            size--;
        }
    }

    int same(int a, int b) {
        return find(a) == find(b);
    }

    int count(int v) {
        return tam[find(v)];
    }
};
```

1.6. Segment Tree ( )

```
struct node { int start, end, maxLen; };

struct STregularBracket {
    vector<node> seg;
    int size;

    STregularBracket(string S) {
        S = "0" + S;
        size = S.size();
        seg.resize(4 * size);
        build(1, 1, size - 1, S);
    }

    void build(int idx, int s, int e, string& S) {
        if (s == e) {
            if (S[s] == '(') seg[idx] = { 1, 0, 0 };
            else             seg[idx] = { 0, 1, 0 };
            return;
        }
        int m = (s + e) / 2;
```

```
    build(idx << 1, s, m, S);
    build(idx << 1 | 1, m + 1, e, S);
    pull(idx);
}

void pull(int idx) {
    node& L = seg[idx << 1], & R = seg[idx << 1 | 1], & P =
        seg[idx];
    P.start = R.start;
    P.end = L.end;
    P.maxLen = L.maxLen + R.maxLen;
    int pares = min(L.start, R.end);
    P.maxLen += pares * 2;
    int dif = L.start - R.end;
    if (dif > 0) P.start += dif;
    else         P.end -= dif;
}

node query(int idx, int s, int e, int l, int r) {
    if (l > e || s > r) return { 0, 0, 0 };
    if (s >= l && e <= r) return seg[idx];
    int m = (s + e) / 2;
    node p1 = query(idx << 1, s, m, l, r);
    node p2 = query(idx << 1 | 1, m + 1, e, l, r);
    node ans;
    ans.start = p2.start;
    ans.end = p1.end;
    ans.maxLen = p1.maxLen + p2.maxLen;
    int pares = min(p1.start, p2.end);
    ans.maxLen += pares * 2;
    int dif = p1.start - p2.end;
    if (dif > 0) ans.start += dif;
    else         ans.end -= dif;
    return ans;
}

void update(int idx, int s, int e, int pos, char val) {
    if (s == e) {
        if (val == '(') seg[idx] = { 1, 0, 0 };
        else            seg[idx] = { 0, 1, 0 };
        return;
    }
    int m = (s + e) / 2;
    if (pos <= m) update(idx << 1, s, m, pos, val);
    else          update(idx << 1 | 1, m + 1, e, pos, val);
    pull(idx);
}
```

```
    // [1, n]
    node query(int l, int r) { return query(1, 1, size - 1, l, r); }
    void update(int pos, char val) { update(1, 1, size - 1, pos, val);
        }
};
```

1.7. STable

```
struct STable {
    int n, K;
    vector<vector<int>> st;

    STable(const vector<int>& a) {
        n = sz(a);
        K = int(log2(n)) + 1;
        st.assign(n + 1, vector<int>(K));
        forn(i, n) st[i][0] = a[i];
        forn(j, K - 1)
            for (int i = 0; i + (1 << (j + 1)) <= n; ++i)
                st[i][j + 1] = oper(st[i][j], st[i + (1 << j)][j]);
    }

    int oper(int a, int b) { return __gcd(a, b); }

    int query(int l, int r) {
        int k = 31 - __builtin_clz(r - l + 1);
        return oper(st[l][k], st[r - (1 << k) + 1][k]);
    }
};
```

1.8. WaveletTree

```
struct WaveletTree {
    int lo, hi;
    WaveletTree* left = nullptr, * right = nullptr;
    vector<int> freq, pref;

    // Build from [from, to) with values in [x, y] x = min value, y =
        max value
    WaveletTree(vector<int>::iterator from, vector<int>::iterator to,
        int x, int y) : lo(x), hi(y) {
        if (from >= to) return;
        int mid = (lo + hi) >> 1;
```

```
    auto f = [mid](int v) { return v <= mid; };

    int sz = to - from;
    freq.reserve(sz + 1);
    freq.push_back(0);
    pref.reserve(sz + 1);
    pref.push_back(0);

    for (auto it = from; it != to; ++it) {
        freq.push_back(freq.back() + f(*it));
        pref.push_back(pref.back() + *it);
    }

    if (lo == hi) return;

    auto pivot = stable_partition(from, to, f);
    left = new WaveletTree(from, pivot, lo, mid);
    right = new WaveletTree(pivot, to, mid + 1, hi);
}

// k-th smallest in [l,r]
int kth(int l, int r, int k) {
    if (l > r) return 0;
    if (lo == hi) return lo;

    int lb = freq[l - 1], rb = freq[r];
    int inLeft = rb - lb;
    if (k <= inLeft) return left->kth(lb + 1, rb, k);
    else return right->kth(l - lb, r - rb, k - inLeft);
}

// number of elements == k in [l,r]
int eq(int l, int r, int k) {
    if (l > r || k < lo || k > hi) return 0;
    if (lo == hi) return r - l + 1;

    int lb = freq[l - 1], rb = freq[r];
    int mid = (lo + hi) >> 1;
    if (k <= mid) return left->eq(lb + 1, rb, k);
    else return right->eq(l - lb, r - rb, k);
}

// number of elements <= k in [l,r]
int le(int l, int r, int k) {
    if (l > r || k < lo) return 0;
    if (hi <= k) return r - l + 1;

    int lb = freq[l - 1], rb = freq[r];
    return left->le(lb + 1, rb, k) + right->le(l - lb, r - rb, k);
}

// number of elements < k in [l,r]
int lt(int l, int r, int k) {
    if (l > r || k <= lo) return 0;
    if (hi < k) return r - l + 1;

    int lb = freq[l - 1], rb = freq[r];
    return left->lt(lb + 1, rb, k) + right->lt(l - lb, r - rb, k);
}

// number of elements >= k in [l,r]
int ge(int l, int r, int k) {
    if (l > r || k > hi) return 0;
    if (k <= lo) return r - l + 1;

    int lb = freq[l - 1], rb = freq[r];
    return left->ge(lb + 1, rb, k) + right->ge(l - lb, r - rb, k);
}

// number of elements > k in [l,r]
int gt(int l, int r, int k) {
    if (l > r || k >= hi) return 0;
    if (k < lo) return r - l + 1;

    int lb = freq[l - 1], rb = freq[r];
    int mid = (lo + hi) >> 1;
    if (k < mid) return left->gt(lb + 1, rb, k) + right->count(l -
        lb, r - rb);
    else return right->gt(l - lb, r - rb, k);
}

// helper to count total in node
int count(int l, int r) {
    if (l > r) return 0;
    return r - l + 1;
}

// number of elements in [l,r] between [a,b]
int between(int l, int r, int a, int b) {
    return le(l, r, b) - lt(l, r, a);
}

// sum of elements <= k in [l,r]
int sum_le(int l, int r, int k) {
```

```
        if (l > r || k < lo) return 0;
        if (hi <= k) return pref[r] - pref[l - 1];

        int lb = freq[l - 1], rb = freq[r];
        return left->sum_le(lb + 1, rb, k) + right->sum_le(l - lb, r -
            rb, k);
    }
};
```

1.9. Mint

```
template <typename T, T m>
struct modint {
    T x;
    constexpr static T mod() { return m; }
    constexpr T val() const { return x; }
    constexpr modint() : x(0) {}
    modint(T x_) :x(x_% mod()) { if (x < 0) x += mod(); }
    modint& operator+=(modint b) { if ((x += b.x) >= mod()) x -=
        mod(); return *this; }
    modint& operator-=(modint b) { if ((x -= b.x) < 0) x += mod();
        return *this; }
    modint& operator*=(modint b) { x = (T)(x)*b.x % mod(); return
        *this; }
    modint pow(T e) const {
        modint r = 1, b = *this;
        while (e) {
            if (e & 1) r *= b;
            b *= b, e >>= 1;
        }
        return r;
    }
    modint inv() { return pow(mod() - 2); }
    modint& operator /=(modint b) { return *this *= b.pow(mod() - 2); }
    friend modint operator+ (modint a, modint b) { return a += b; }
    friend modint operator- (modint a, modint b) { return a -= b; }
    friend modint operator/ (modint a, modint b) { return a /= b; }
    friend modint operator* (modint a, modint b) { return a *= b; }
    friend bool   operator< (modint a, modint b) { return a.x < b.x; }
    friend bool   operator> (modint a, modint b) { return a.x > b.x; }
    friend bool   operator==(modint a, modint b) { return a.x == b.x; }
    friend bool   operator!=(modint a, modint b) { return a.x != b.x; }
    friend ostream& operator<<(ostream& os, const modint& a) { return
        os << a.val(); }
};
constexpr int mod = 1000000007;
using mint = modint<int, mod>;
```

1.10. Eval

```
template <typename T> struct eval {
    string s;
    int n;
    eval(string s) : s(s), n(sz(s)) {}

    stack <T> nums;
    stack <char> oper;

    int order(char op) {
        if (op < 0) return 3;
        if (op == '+' || op == '-') return 1;
        if (op == '*' || op == '/') return 2;
        return 0;
    }
    bool is_op(char c) { return c == '+' || c == '-' || c == '*' || c
        == '/'; }

    bool is_unary(char c) { return c == '+' || c == '-'; }

    T apply(T a, T b, char op) {
        if (op == '+') return a + b;
        if (op == '-') return a - b;
        if (op == '*') return a * b;
        if (op == '/') return a / b;
        return 0;
    }

    T go() {
        int op = oper.top(); oper.pop();
        if (op < 0) {
            T v = nums.top(); nums.pop();
            return apply(0, v, -op);
        }
        T v2 = nums.top(); nums.pop();
        T v1 = nums.top(); nums.pop();
        return apply(v1, v2, op);
    }

    T get() {
        bool ok = 1;
        forn(i, n) {
            if (s[i] == ' ') continue;
```

```
        if (s[i] == '(') oper.push('('), ok = 1;
        else if (s[i] == ')') {
            while (oper.top() != '(') nums.push(go());
            oper.pop(), ok = 0;
        }
        else if (is_op(s[i])) {
            char alt = s[i];
            if (ok && is_unary(alt)) alt = -alt;
            while (sz(oper) && ((alt >= 0 && order(oper.top()) >=
                order(alt)) || (alt < 0 && order(oper.top()) >
                order(alt)))) nums.push(go());
            oper.push(alt), ok = 1;
        }

        // else {
        //     int val = 0;
        //     while (i < n && isalnum(s[i])) val = val * 10 +
            s[i++] - '0';
        //     --i;
        //     nums.push(val), ok = 0;
        // }

        else {
            T val = 0;
            int dec = -1;
            while (i < n && (isdigit(s[i]) || s[i] == '.')) {
                if (s[i] == '.') dec = 0;
                else {
                    val = val * 10 + (s[i] - '0');
                    if (dec >= 0) ++dec;
                }
                ++i;
            }
            if (dec > 0) val /= pow(10, dec);
            --i;
            nums.push(val);
            ok = 0;
        }
    }

    while (sz(oper)) nums.push(go());

    return nums.top();
}

};
```

1.11. Fenwick Tree

```
template<typename T>
struct BIT {
    vector<T> ft;
    BIT(int n) : ft(n + 1) {}
    BIT(const vector<T>& a) : ft(sz(a) + 1) {
        forn(i, sz(a)) { upd(i + 1, a[i]); }
    }

    T qry(int i) {
        T ans = 0;
        for (; i; i -= i & -i) ans += ft[i];
        return ans;
    }

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

    void upd(int i, T v) {
        for (; i < sz(ft); i += i & -i) ft[i] += v;
    }
};
```

1.12. Segment Tree 2D

```
template<typename T>
struct STree {
    int n, m;
    T neutro = T(0);
    vector<vector<T>> st;

    STree(vector<vector<T>>& a) {
        n = sz(a);
        m = sz(a[0]);
        st = vector<vector<T>>(2 * n, vector<T>(2 * m, neutro));
        build(a);
    }

    inline T oper(T a, T b) { return a + b; }

    void build(vector<vector<T>>& a) {
        forn(i, n) forn(j, m) st[i + n][j + m] = a[i][j];
        forn(i, n) {
            for (int j = m - 1; j >= 1; --j) {
                st[i + n][j] = oper(st[i + n][j << 1], st[i + n][j << 1
                    | 1]);
```

```
            }
        }
        for (int i = n - 1; i >= 1; --i) {
            forn(j, 2 * m) {
                st[i][j] = oper(st[i << 1][j], st[i << 1 | 1][j]);
            }
        }
    }

    T qry(int x1, int y1, int x2, int y2) { // [x1, y1] [x2, y2]
        T ans = neutro;
        for (int i0 = x1 + n, i1 = x2 + n + 1; i0 < i1; i0 >>= 1, i1
            >>= 1) {
            int t[4], q = 0;
            if (i0 & 1) t[q++] = i0++;
            if (i1 & 1) t[q++] = --i1;
            forn(k, q)
                for (int j0 = y1 + m, j1 = y2 + m + 1; j0 < j1; j0 >>=
                    1, j1 >>= 1) {
                    if (j0 & 1) ans = oper(ans, st[t[k]][j0++]);
                    if (j1 & 1) ans = oper(ans, st[t[k]][--j1]);
                }
        }
        return ans;
    }

    void upd(int l, int r, T val) {
        st[l + n][r + m] = val;
        for (int j = r + m; j > 1; j >>= 1) {
            st[l + n][j >> 1] = oper(st[l + n][j], st[l + n][j ^ 1]);
        }
        for (int i = l + n; i > 1; i >>= 1) {
            for (int j = r + m; j; j >>= 1) {
                st[i >> 1][j] = oper(st[i][j], st[i ^ 1][j]);
            }
        }
    }
};
```

1.13. Segment Tree Iterative

```
template<typename T>
struct STree {
    vector<T> st;
    int n;
    T neutro = T(0);
    T oper(T a, T b) { return a + b; }
    STree(vector<T>& a) {
        n = sz(a);
        st.resize(n * 2);
        forn(i, n) st[n + i] = a[i];
        for (int i = n - 1; i >= 1; i -= 1) st[i] = oper(st[i << 1],
            st[i << 1 | 1]);
    }

    void upd(int p, T val) {
        for (st[p += n] = val; p > 1; p >>= 1) st[p >> 1] =
            oper(st[p], st[p ^ 1]);
    }

    T query(int l, int r) { //[l, r)
        T v = neutro;
        for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
            if (l & 1) v = oper(v, st[l++]);
            if (r & 1) v = oper(v, st[--r]);
        }
        return v;
    }
};
```

1.14. Segment Tree Lazy

```
template<typename T>
struct STree {
    int n; vector<T> st, lazy;
    T neutro = T(0);

    STree(int m) {
        n = m;
        st.resize(n * 4);
        lazy.resize(n * 4);
    }

    STree(vector<T>& a) {
        n = sz(a);
        st.resize(n * 4);
        lazy.resize(n * 4);
        build(1, 0, n - 1, a);
    }

    T oper(T a, T b) { return a + b; }
```

```
    void build(int v, int tl, int tr, vector<T>& a) {
        if (tl == tr) {
            st[v] = a[tl];
            return;
        }
        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, a);
        build(v * 2 + 1, tm + 1, tr, a);
        st[v] = oper(st[v * 2], st[v * 2 + 1]);
    }

    void push(int v, int tl, int tr) {
        if (!lazy[v]) return;
        st[v] += (tr - tl + 1) * lazy[v];
        if (tl != tr) {
            lazy[v * 2] += lazy[v];
            lazy[v * 2 + 1] += lazy[v];
        }
        lazy[v] = 0;
    }

    void upd(int v, int tl, int tr, int l, int r, T val) {
        push(v, tl, tr);
        if (tr < l || tl > r) return;
        if (tl >= l && tr <= r) {
            lazy[v] = val;
            push(v, tl, tr);
            return;
        }
        int tm = (tl + tr) / 2;
        upd(v * 2, tl, tm, l, r, val);
        upd(v * 2 + 1, tm + 1, tr, l, r, val);
        st[v] = oper(st[v * 2], st[v * 2 + 1]);
    }

    T query(int v, int tl, int tr, int l, int r) {
        push(v, tl, tr);
        if (tl > r || tr < l) return neutro;
        if (l <= tl && tr <= r) return st[v];
        int tm = (tl + tr) / 2;
        return oper(query(v * 2, tl, tm, l, r), query(v * 2 + 1, tm + 1
            , tr, l, r));
    }

    void upd(int l, int r, T val) { upd(1, 0, n - 1, l, r, val); }
    T query(int l, int r) { return query(1, 0, n - 1, l, r); }
};
```

1.15. Segment Tree

```
template<typename T>
struct STree {
    int n; vector<T> st;
    T neutro = T(0);

    STree(vector<T>& a) {
        n = sz(a);
        st.resize(n * 4);
        build(1, 0, n - 1, a);
    }

    T oper(T a, T b) { return max(a, b); }

    void build(int v, int tl, int tr, vector<T>& a) {
        if (tl == tr) {
            st[v] = a[tl];
            return;
        }
        int tm = (tr + tl) / 2;
        build(v * 2, tl, tm, a);
        build(v * 2 + 1, tm + 1, tr, a);
        st[v] = oper(st[v * 2], st[v * 2 + 1]);
    }

    T query(int v, int tl, int tr, int l, int r) {
        if (tl > r || tr < l) return neutro;
        if (l <= tl && tr <= r) return st[v];
        int tm = (tl + tr) / 2;
        return oper(query(v * 2, tl, tm, l, r), query(v * 2 + 1, tm + 1
            , tr, l, r));
    }

    void upd(int v, int tl, int tr, int pos, T val) {
        if (tl == tr) {
            st[v] = val;
            return;
        }
        int tm = (tr + tl) / 2;
        if (pos <= tm) upd(v * 2, tl, tm, pos, val);
        else upd(v * 2 + 1, tm + 1, tr, pos, val);
        st[v] = oper(st[v * 2], st[v * 2 + 1]);
    }
    void upd(int pos, T val) { upd(1, 0, n - 1, pos, val); }
    T query(int l, int r) { return query(1, 0, n - 1, l, r); }
};
```

1.16. SqrtBlocks

```
template<typename T>
struct SqrtBlocks {
    int n, blk_sz, blk_n;
    vector<T> st;
    vector<vector<T>> blocks;

    SqrtBlocks(vector<T>& a) {
        n = sz(a), st = a;
        blk_sz = sqrt(n) + 1, blk_n = (n + blk_sz - 1) / blk_sz;
        blocks.resize(blk_n);
        forn(i, n) blocks[i / blk_sz].pb(st[i]);
        forn(i, blk_n) sort(all(blocks[i]));
    }

    void update(int pos, int val) {
        int blk = pos / blk_sz;
        auto& b = blocks[blk];
        auto it = lower_bound(all(b), st[pos]);
        b.erase(it);
        b.insert(lower_bound(all(b), val), val);
        st[pos] = val;
    }
    // >
    T query_greater(int l, int r, int val) {
        T res = 0;
        int bl = l / blk_sz, br = r / blk_sz;
        if (bl == br) {
            forne(i, l, r + 1) res += (st[i] > val);
            return res;
        }
        int end_l = (bl + 1) * blk_sz;
        forne(i, l, end_l) res += (st[i] > val);
        forne(b, bl + 1, br) res += end(blocks[b]) -
            upper_bound(all(blocks[b]), val);
        int start_r = br * blk_sz;
        forne(i, start_r, r + 1) res += (st[i] > val);
        return res;
    }

    // >=
    T query_ge(int l, int r, int val) {
        T res = 0;
        int bl = l / blk_sz, br = r / blk_sz;
        if (bl == br) {
            forne(i, l, r + 1) res += (st[i] >= val);
            return res;
        }
        int end_l = (bl + 1) * blk_sz;
        forne(i, l, end_l) res += (st[i] >= val);
        forne(b, bl + 1, br) res += end(blocks[b]) -
            lower_bound(all(blocks[b]), val);
        int start_r = br * blk_sz;
        forne(i, start_r, r + 1) res += (st[i] >= val);
        return res;
    }

    // <
    T query_less(int l, int r, int val) {
        T res = 0;
        int bl = l / blk_sz, br = r / blk_sz;
        if (bl == br) {
            forne(i, l, r + 1) res += (st[i] < val);
            return res;
        }
        int end_l = (bl + 1) * blk_sz;
        forne(i, l, end_l) res += (st[i] < val);
        forne(b, bl + 1, br) res += lower_bound(all(blocks[b]), val) -
            begin(blocks[b]);
        int start_r = br * blk_sz;
        forne(i, start_r, r + 1) res += (st[i] < val);
        return res;
    }

    // <=
    T query_le(int l, int r, int val) {
        T res = 0;
        int bl = l / blk_sz, br = r / blk_sz;
        if (bl == br) {
            forne(i, l, r + 1) res += (st[i] <= val);
            return res;
        }
        int end_l = (bl + 1) * blk_sz;
        forne(i, l, end_l) res += (st[i] <= val);
        forne(b, bl + 1, br) res += upper_bound(all(blocks[b]), val) -
            begin(blocks[b]);
        int start_r = br * blk_sz;
        forne(i, start_r, r + 1) res += (st[i] <= val);
        return res;
    }

    // ==
    T query_equal(int l, int r, int val) {
```

```
        T res = 0;
        int bl = l / blk_sz, br = r / blk_sz;
        if (bl == br) {
            forne(i, l, r + 1) res += (st[i] == val);
            return res;
        }
        int end_l = (bl + 1) * blk_sz;
        forne(i, l, end_l) res += (st[i] == val);
        forne(b, bl + 1, br) res += upper_bound(all(blocks[b]), val) -
            lower_bound(all(blocks[b]), val);
        int start_r = br * blk_sz;
        forne(i, start_r, r + 1) res += (st[i] == val);
        return res;
    }

    // between [a,b]
    T query_between(int l, int r, int a, int b) {
        T res = 0;
        int bl = l / blk_sz, br = r / blk_sz;
        if (bl == br) {
            forne(i, l, r + 1) res += (st[i] >= a && st[i] <= b);
            return res;
        }
        int end_l = (bl + 1) * blk_sz;
        forne(i, l, end_l) res += (st[i] >= a && st[i] <= b);
        forne(bk, bl + 1, br) res += upper_bound(all(blocks[bk]), b) -
            lower_bound(all(blocks[bk]), a);
        int start_r = br * blk_sz;
        forne(i, start_r, r + 1) res += (st[i] >= a && st[i] <= b);
        return res;
    }
};
```

1.17. Coo Compress

```
template<typename T>
struct COO_COMPRESS {
    vector<T> nums;
    bool is_compress = true;

    int size() {
        if (!is_compress) compress();
        return sz(nums);
    }

    void clear() {
        nums.clear();
        is_compress = true;
    }

    void insert(T x) {
        nums.pb(x);
        is_compress = false;
    }

    void compress() {
        sort(all(nums));
        nums.resize(unique(all(nums)) - nums.begin());
        is_compress = true;
    }

    vector<T> compress_offline(vector<T> nums) {
        if (!sz(nums))return nums;
        vector<pair<T, int>> vvv;
        forn(i, sz(nums)) vvv.pb({ nums[i],i });
        sort(all(vvv));
        int cont = 0;
        T last = vvv[0].first;
        nums[vvv[0].second] = 0;
        forne(i, 1, sz(vvv)) {
            if (vvv[i].first != last) cont++, last = vvv[i].first;
            nums[vvv[i].second] = cont;
        }
        return nums;
    }

    int get(T x) {
        if (!is_compress) compress();
        int pos = lower_bound(all(nums), x) - nums.begin();
        assert(pos != sz(nums) && nums[pos] == x);
        return pos;
    }

    T iget(int x) {
        if (!is_compress) compress();
        assert(0 <= x && x < sz(nums));
        return nums[x];
    }
};
```

1.18. Mo's

```
void add(int x) {}
void del(int x) {}
int get_ans() {}

vector<int> mo(const vector<pair<int, int>>& q) {
  int l = 0, r = -1, blk = 350; // sqrt(n)
  vector<int> inx(sz(q)), ans(sz(q));
  auto K = [&](const pair<int, int>& x) -> pair<int, int> {
    return pair<int, int>(x.f / blk, x.s ^ -(x.f / blk & 1));
    };
  iota(all(inx), 0);
  sort(all(inx), [&](int a, int b) -> bool { return K(q[a]) < K(q[b]);
     });
  /*
  sort(all(inx), [&](int l, int r) {
      if (q[l].f / blk != q[r].f / blk) return q[l].f < q[r].f;
      if ((q[l].f / blk) % 2) return q[l].s > q[r].s;
      return q[l].s < q[r].s;
  });
  */
  for (int nxt : inx) {
    pair<int, int> it = q[nxt];
    while (r < it.s) add(++r);
    while (l > it.f) add(--l);
    while (r > it.s) del(r--);
    while (l < it.f) del(l++);
    ans[nxt] = get_ans();
  }
  return ans;
}
```

2. DP

2.1. Knapsack

```
    int n, x; cin>>n>>x;
    vector<array<int,2>>arr(n);
    forn(i,n) cin>>arr[i][0];
    forn(i,n) cin>>arr[i][1];

    vector<vector<int>>dp(n+1,vector<int>(x+1,0));
    forne(i,1,n+1){
        forne(j,1,x+1){
            dp[i][j]=dp[i-1][j];
            if(j-arr[i-1][0]>=0){
                int libro=arr[i-1][1];
                int price=arr[i-1][0];
                dp[i][j]=max(dp[i][j], libro+dp[i-1][j-price]);
             }

        }
    }
    cout<<dp[n][x]<<endl;


const ll inf=1e18+7;

ll Knapsack(ll n, ll cty,  vector<ll>& W,vector<ll>& V) {
    ll sum=accumulate(all(V),0LL);
    vector<ll>dp(sum+1,inf);
    dp[0]=0;
    forn(i, n){
        for(int j = sum-V[i]; j >= 0; j--){
            dp[j+V[i]]= min(dp[j+V[i]], dp[j]+W[i]);
        }
    }
    ll ans=0;
    forn(i,sum+1){
        if(dp[i]<= cty) ans=max(ans,ll(i));
    }
    return ans;
}
```

2.2. Lis

```
int lis(vector<int>& a) {
    vector<int>dp;
    forn(i, sz(a)) {
        auto it = lower_bound(all(dp), a[i]);
        if (it != dp.end()) *it = a[i];
        else dp.pb(a[i]);
    }
    return sz(dp);
}

constexpr int INF = ((1ULL << 63) - 1) >> 32;
template<typename T> vector<T> lis(vector<T>& v) {
    int n = sz(v), m = -1;
```

```
    vector<T> d(n + 1, INF);
    vector<int> l(n);
    d[0] = -INF;

    forn(i, n) {
        // Para non-decreasing use upper_bound()
        int t = lower_bound(all(d), v[i]) - begin(d);
        d[t] = v[i], l[i] = t, m = max(m, t);
    }

    int p = n;
    vector<T> ans;
    while (p--) if (l[p] == m) {
        ans.pb(v[p]); m--;
    }
    reverse(all(ans));
    return ans;
}


int lis_2(int n, vector<int>& A, vector<int>& B) {

    vector<vector<int>>C(inf, vector<int>());
    reverse(all(A)); reverse(all(B));
    for (int i = n - 1; i >= 0; i--) C[B[i]].pb(i);

    STree<int> dp(n); // Stree de max, neutro = INT_MIN

    for (int j = 0; j < n; j++) {
        for (auto&& i : C[A[j]]) {
            int mx_pre = dp.query(0, i);
            if (mx_pre != INT_MIN) dp.upd(i, mx_pre + 1);
            else dp.upd(i, 1);
        }
    }
    return dp.query(0, n);
}
```

2.3. Divide and Conquer dp

```
/*
Divide and Conquer DP
Particiona o array en k subarrays
minimizando la suma de las queries
*/
ll dp[MAX][2];

void solve(int k, int l, int r, int lk, int rk) {
    if (l > r) return;
    int m = (l + r) / 2, p = -1;
    auto& ans = dp[m][k & 1] = LINF;
    for (int i = max(m, lk); i <= rk; i++) {
        ll at = dp[i + 1][~k & 1] + query(m, i);
        if (at < ans) ans = at, p = i;
    }
    solve(k, l, m - 1, lk, p), solve(k, m + 1, r, p, rk);
}

ll DC(int n, int k) {
    dp[n][0] = dp[n][1] = 0;
    for (int i = 0; i < n; i++) dp[i][0] = LINF;
    for (int i = 1; i <= k; i++) solve(i, 0, n - i, 0, n - i);
    return dp[0][k & 1];
}
```

2.4. Edit Distance

```
/*
The edit distance between two strings is the minimum number of
   operations required to transform one string into the other.
*/
{
    string a, b; cin >> a >> b;
    int n = sz(a), m = sz(b);
    vector<vector<int>>dp(n + 1, vector<int>(m + 1, inf));
    forne(i, 0, n + 1) dp[i][0] = i;
    forne(j, 0, m + 1) dp[0][j] = j;
    forne(i, 1, n + 1) {
        forne(j, 1, m + 1) {
            dp[i][j] = min({ dp[i][j - 1] + 1,dp[i - 1][j - 1] + (a[i
                - 1] != b[j - 1]),dp[i - 1][j] + 1 });
        }
    }
    cout << dp[n][m] << endl;
}

constexpr int INF = (1e18 - 1);

int edit_distance(const string& s, const string& t) {
    int n = sz(s), m = sz(t);
    vector<int> dp(m + 1);
```

```
    iota(all(dp), 0);

    forn(i, n) {
        vector<int> ndp(m + 1, INF);
        ndp[0] = i + 1;
        forn(j, m) {
            ndp[j + 1] = min({ ndp[j] + 1, dp[j + 1] + 1, dp[j] +
                (s[i] != t[j]) });
        }
        dp.swap(ndp);
    }

    return dp[m];
}
vector<string> construct_edit_distance(const string& s, const string&
    t) {

    int n = sz(s), m = sz(t);
    vector<vector<int>> dp(n + 1, vector<int>(m + 1, INF));

    forn(i, n + 1) dp[i][0] = i;
    forn(j, m + 1) dp[0][j] = j;

    forn(i, n) {
        forn(j, m) {
            dp[i + 1][j + 1] = min({ dp[i + 1][j] + 1, dp[i][j + 1] + 1
                , dp[i][j] + (s[i] != t[j]) });
        }
    }

    vector<string> left = { s }, right = { t };

    while (n > 0 || m > 0) {
        if (n > 0 && dp[n][m] == dp[n - 1][m] + 1) {
            n--;
            string str = left.back();
            str.erase(str.begin() + n);
            left.push_back(str);
        }
        else if (m > 0 && dp[n][m] == dp[n][m - 1] + 1) {
            m--;
            string str = right.back();
            str.erase(str.begin() + m);
            right.push_back(str);
        }
        else if (n > 0 && m > 0 && dp[n][m] == dp[n - 1][m - 1] + (s[n
            - 1] != t[m - 1])) {
            n--, m--;
            if (s[n] != t[m]) {
                string str = left.back();
                str[n] = t[m];
                left.push_back(str);
            }
        }
        else {
            assert(false);
        }
    }

    assert(left.back() == right.back());
    right.pop_back();

    while (!right.empty()) {
        left.push_back(right.back());
        right.pop_back();
    }

    return left;
}
```

2.5. groups

```
/*
Dado N pesos y un limite Q, se quiere saber el minimo numero
de grupos en los que se pueden dividir los pesos tal que la
suma de los pesos de cada grupo sea menor o igual a Q
n => sz(nums);
q => maximo peso
nums => vector con los pesos
*/
int calculate(int n, int q, vector<int>& nums) {
    pair<int, int> best[1 << n];
    best[0] = { 1,0 };
    forne(i, 1, 1 << n) {
        best[i] = { n + 1,0 };
        forn(j, n) {
            if (i & (1 << j)) {
                auto cur = best[i ^ (1 << j)];
                if (cur.s + nums[j] <= q) {
                    cur.s += nums[j];
                }
                else {
                    cur.f++;
```

```
                cur.s = nums[j];
            }
            best[i] = min(best[i], cur);
        }
    }
}
return best[(1 << n) - 1].f;
}


/*
Dado N pesos y un limite Q, se quiere saber el numero de grupos
consecutivos en los que se pueden dividir los pesos tal que la
suma de los pesos de cada grupo sea menor o igual a Q
n => sz(nums);
q => maximo peso
nums => vector con los pesos
*/

int get(int n, int q, vector<int>& nums) {
    sort(all(nums));
    int l = 0, r = n - 1, ans = n;
    while (l < r) {
        if (nums[l] + nums[r] <= q) ans--, l++, r--;
        else r--;
    }
    return ans;
}
```

2.6. Shortest Hamiltonian Path

```
/*
Shortest Hamiltonian Path
Resuelve problemas del tipo de encontrar el camino mas corto
que recorre todos los nodos de un grafo una sola vez.
*/
vector<vector<pair<int, int>>> ady;
int n, m, target;
const int N = 18;
const int MASK = 1 << N;
const int INF = int(1e7);
int dp[N][MASK];

int solve(int v, int mask) {
    if (mask == target) return 0;
    int& ans = dp[v][mask];
    if (ans != -1) return ans;
    ans = INF;
    for (auto& u : ady[v]) {
        if (!(mask & (1 << u.first))) {
            ans = min(ans, solve(u.first, mask | (1 << u.first)) +
                u.second);
        }
    }
    return ans;
}
int main() {
cin >> n >> m;
target = (1 << n) - 1;
ady.assign(n, {});
forn(i, m) {
    int v, u, w; cin >> v >> u >> w;
    v--, u--;
    ady[v].push_back({ u, w });
    ady[u].push_back({ v, w });
}
memset(dp, -1, sizeof dp);
cout << solve(0, 1) << endl;

cout << flush;
return 0;
}
```

2.7. Money Sums

```
//  find all money sums you can create using these coins.
    int n; cin >> n;
    vector<int>nums(n), sums;
    forn(i, n) cin >> nums[i];
    vector<vector<bool>>dp(mxN + 1, vector<bool>(n * mxS + 1));
    dp[0][0] = 1;
    forne(i, 1, n + 1) {
        forn(j, mxS * n + 1) {
            dp[i][j] = dp[i - 1][j];
            if (j - nums[i - 1] >= 0 && dp[i - 1][j - nums[i - 1]])
                dp[i][j] = 1;
        }
    }

    forn(i, mxS * n + 1) {
        if (i && dp[n][i]) sums.pb(i);
    }
```

```
    cout << sz(sums) << endl;
    forn(i, sz(sums)) cout << sums[i] << " \n"[i + 1 == sz(sums)];
    cout << endl;
```

2.8. Digit dp

```
// - Descripcion: Cuenta la cantidad de numeros entre [a, b] que no
   tienen digitos iguales seguidos
// - Complejidad: O(NUM_E * NUM_T)

const int MOD = 998244353;
int tam, NUM[55], dp[55][2][2][11];

int solve(int i, bool menor, bool ncero, int last) {
    if (i == tam) return 1;
    int& ans = dp[i][menor][ncero][last];
    if (ans != -1) return ans;
    ans = 0;
    forn(dig, 10) {
        if (dig == last && (ncero || dig)) continue;
        if (menor || dig <= NUM[i]) {
            ans = (ans + solve(i + 1, menor || dig < NUM[i], ncero ||
                dig, dig)) % MOD;
        }
    }
    return ans;
}

bool g(string s) {
    forn(i, sz(s) - 1) {
        if (s[i] == s[i + 1]) return false;
    }
    return true;
}

int build(string s) {
    tam = sz(s);
    forn(i, sz(s)) {
        NUM[i] = s[i] - '0';
    }
    memset(dp, -1, sizeof dp);
    return solve(0, false, false, 10);
}
void solve() {
    string l, r;
    while (cin >> l >> r) {
        cout << ((build(r) - build(l) + MOD) % MOD + g(l)) % MOD <<
            endl;
    }
}
```

2.9. LCS

```
constexpr int mxN = 105;
vector<vector<int>> dp(mxN, vector<int>(mxN, -1));
// n=sz(s), m=sz(p)
int cntsub(const string& s, const string& p, int n, int m) {
    if ((n == 0 && m == 0) || m == 0) return 1;
    if (n == 0) return 0;
    int& ans = dp[n][m];
    if (~ans) return ans;
    if (s[n - 1] == p[m - 1]) {
        return ans = cntsub(s, p, n - 1, m - 1) + cntsub(s, p, n - 1,
            m);
    }
    else {
        return ans = cntsub(s, p, n - 1, m);
    }
}

bool issub(const string& str, const string& sub) {
    int idx = 0;
    for (auto&& i : str) {
        if (idx < sz(sub) && i == sub[idx]) {
            idx++;
        }
    }
    return idx == sz(sub);
}

//quadratic_memory
int lcs(const string& s, const string& t) {
    int n = sz(s);
    int m = sz(t);
    vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));
    forn(i, n) {
        forn(j, m) {
            dp[i + 1][j + 1] = max({ dp[i + 1][j], dp[i][j + 1],
                dp[i][j] + (s[i] == t[j]) });
        }
    }
```

```
    return dp[n][m];
}

//best
int lcs(const string& s, const string& t) {
    int n = sz(s);
    int m = sz(t);
    vector<int> dp(m + 1, 0);
    forn(i, n) {
        vector<int> newdp(m + 1, 0);
        forn(j, m) {
            newdp[j + 1] = max({ newdp[j], dp[j + 1], dp[j] + (s[i] ==
                t[j]) });
        }
        dp.swap(newdp);
    }
    return dp[m];
}

//construct lcs
string clcs(const string& s, const string& t) {
    int n = sz(s);
    int m = sz(t);
    vector<int> dp(m + 1, 0);
    vector<vector<bool>> pre(n + 1, vector<bool>(m + 1, false));
    forn(i, n) {
        vector<int> newdp(m + 1, 0);
        forn(j, m) {
            newdp[j + 1] = max({ newdp[j], dp[j + 1], dp[j] + (s[i] ==
                t[j]) });
            pre[i + 1][j + 1] = newdp[j + 1] == newdp[j];
        }
        dp.swap(newdp);
    }
    int a = n, b = m;
    string common;
    while (a > 0 && b > 0) {
        if (s[a - 1] == t[b - 1]) {
            common += s[a - 1];
            a--; b--;
            continue;
        }
        if (pre[a][b]) b--;
        else a--;
    }
    reverse(all(common));
    return common;
}

//best: construct lcs with  Hirschberg Algorithm
string clcsh(const string_view& s, const string_view& t) {
    int n = sz(s), int m = sz(t);
    if (n == 0 || m == 0) return "";
    if (n == 1) return t.find(s[0]) == string::npos ? "" : string(1,
        s[0]);
    int mid = n >> 1;
    vector<int> dp_ff(m + 1, 0);
    vector<int> dp_ss(m + 1, 0);
    vector<int> newdp(m + 1, 0);

    forn(i, mid) {
        forn(j, m) newdp[j + 1] = max({ newdp[j], dp_ff[j + 1],
            dp_ff[j] + (s[i] == t[j]) });
        dp_ff.swap(newdp);
    }

    newdp.assign(m + 1, 0);
    for (int i = n - 1; i >= mid; i--) {
        for (int j = m - 1; j >= 0; j--) {
            newdp[j] = max({ newdp[j + 1], dp_ss[j], dp_ss[j + 1] +
                (s[i] == t[j]) });
        }
        dp_ss.swap(newdp);
    }
    int splt = 0;
    forne(j, 1, m + 1) {
        if (dp_ff[j] + dp_ss[j] > dp_ff[splt] + dp_ss[splt]) {
            splt = j;
        }
    }
    dp_ff.clear();
    dp_ss.clear();
    newdp.clear();
    return (clcsh(s.substr(0, mid), t.substr(0, splt)) +
        clcsh(s.substr(mid), t.substr(splt)));
}

// lcs con tolerncia de 1% de eliminaciones al inicio
int lcs(const string& s, const string& t) {
    int n = sz(s);
    int poda = (n * 1) / 100 + 1;
    int ans = 0;
    vector<vector<int>> dp(poda + 1, vector<int>(poda + 1, 0));
    forn(i, poda + 1) {
```

```
        forn(j, poda + 1) {
            while (i + dp[i][j] < n && j + dp[i][j] < n && s[i +
                dp[i][j]] == t[j + dp[i][j]]) dp[i][j]++;
            if (i + 1 <= poda) dp[i + 1][j] = max(dp[i + 1][j],
                dp[i][j]);
            if (j + 1 <= poda) dp[i][j + 1] = max(dp[i][j + 1],
                dp[i][j]);
            ans = max(ans, dp[i][j]);
        }
    }
    return ans;
}
```

2.10. Subsequences

```
struct mint {
    static constexpr int  m = 1e9 + 7;
    //static inline int  m = 998244353; //to change mod
    int x;
    mint() : x(0) {}
    mint(long long x_) :x(x_% m) { if (x < 0) x += m; }
    int val() { return x; }
    mint& operator+=(mint b) { if ((x += b.x) >= m) x -= m; return
        *this; }
    mint& operator-=(mint b) { if ((x -= b.x) < 0) x += m; return
        *this; }
    mint& operator*=(mint b) { x = (long long)(x)*b.x % m; return
        *this; }
    mint pow(long long e) const {
        mint r = 1, b = *this;
        while (e) {
            if (e & 1) r *= b;
            b *= b;
            e >>= 1;
        }
        return r;
    }
    mint inv() { return pow(m - 2); }
    mint& operator/=(mint b) { return *this *= b.pow(m - 2); }
    friend mint operator+(mint a, mint b) { return a += b; }
    friend mint operator-(mint a, mint b) { return a -= b; }
    friend mint operator/(mint a, mint b) { return a /= b; }
    friend mint operator*(mint a, mint b) { return a *= b; }
    friend bool operator<(mint a, mint b) { return a.x < b.x; }
    friend bool operator==(mint a, mint b) { return a.x == b.x; }
    friend bool operator!=(mint a, mint b) { return a.x != b.x; }
};
// Find the number of distinct subsequences of a given string.
// distinct subsequences ending at each of the 26 letters of the
    alphabet.
template<typename T>int distinctsub(const T& sub) {
    int n = sz(sub);
    vector<mint> dp(n + 1, 0);
    vector<int>last(26, -1);
    // vector<mint>end_count(26, 0);
    dp[0] = 1;
    forn(i, n) {
        dp[i + 1] += 2 * dp[i];
        // end_count[sub[i] - 'a'] += dp[i];
        if (~last[sub[i] - 'a']) {
            dp[i + 1] -= dp[last[sub[i] - 'a']];
            // end_count[sub[i] - 'a'] -= dp[last[sub[i] - 'a']];
        }
        last[sub[i] - 'a'] = i;
    }
    return dp[n].x - 1;
}

// find the number of distinct subsequences of a given string.
// number of distinct subsequences of each length from 1 to n
// number of distinct subsequences of size i -> dp[n][i]
template<typename T>int distinctsub(const T& sub) {
    int n = sz(sub);
    vector<vector<mint>> dp(n + 1, vector<mint>(n + 1, 0));
    dp[0][0] = 1;
    vector<int> last(26, -1);
    // vector<mint> end_count(26, 0);
    forn(i, n) {
        forn(j, i + 1) {
            dp[i + 1][j + 1] = dp[i][j];
            dp[i + 1][j] += dp[i][j];
            // end_count[sub[i] - 'a'] += dp[i][j].x;
        }
        if (~last[sub[i] - 'a']) {
            forn(j, i + 1) {
                dp[i + 1][j + 1] -= dp[last[sub[i] - 'a']][j];
                // end_count[sub[i] - 'a'] -= dp[last[sub[i] -
                    'a']][j].x;
            }
        }
        last[sub[i] - 'a'] = i;
    }
```

```
    mint ans = 0;
    forne(i, 1, n + 1) ans += dp[n][i];
    return ans.x;
}
```

3. Flows

3.1. Dinic

```
constexpr int INF = ((1ULL << 63) - 1) >> 1;

struct Dinic {
    const bool scaling = 0;     // con scaling -> O(nm log(MAXCAP)),
    int lim;                    // con constante alta
    struct edge {
        int to, cap, rev, flow;
        bool res;
        edge(int to_, int cap_, int rev_, bool res_) : to(to_),
            cap(cap_), rev(rev_), flow(0), res(res_) {}
    };

    vector<vector<edge>> g;
    vector<int> lev, beg;
    int64_t F;
    Dinic(int n) : g(n), F(0) {}

    void add(int a, int b, int c) {
        g[a].eb(b, c, sz(g[b]), 0);
        g[b].eb(a, 0, sz(g[a]) - 1, 1);
    }
    bool bfs(int s, int t) {
        lev = vector<int>(sz(g), -1); lev[s] = 0;
        beg = vector<int>(sz(g), 0);
        queue<int> q; q.push(s);
        while (sz(q)) {
            int u = q.front(); q.pop();
            for (auto& i : g[u]) {
                if (lev[i.to] != -1 or (i.flow == i.cap)) continue;
                if (scaling and i.cap - i.flow < lim) continue;
                lev[i.to] = lev[u] + 1;
                q.push(i.to);
            }
        }
        return lev[t] != -1;
    }
    int dfs(int v, int s, int f = INF) {
        if (!f or v == s) return f;
        for (int& i = beg[v]; i < sz(g[v]); i++) {
            auto& e = g[v][i];
            if (lev[e.to] != lev[v] + 1) continue;
            int foi = dfs(e.to, s, min(f, e.cap - e.flow));
            if (!foi) continue;
            e.flow += foi, g[e.to][e.rev].flow -= foi;
            return foi;
        }
        return 0;
    }

    int64_t max_flow(int s, int t) {
        for (lim = scaling ? (1 << 30) : 1; lim; lim /= 2)
            while (bfs(s, t)) while (int ff = dfs(s, t)) F += ff;
        return F;
    }
};

// Recupera las aristas del corte s-t
vector<pair<int, int>> get_cut(Dinic& g, int s, int t) {
    g.max_flow(s, t);
    vector<pair<int, int>> cut;
    vector<int> vis(sz(g.g), 0), st = { s };
    vis[s] = 1;
    while (sz(st)) {
        int u = st.back(); st.pop_back();
        for (auto e : g.g[u]) if (!vis[e.to] and e.flow < e.cap)
            vis[e.to] = 1, st.pb(e.to);
    }
    for (int i = 0; i < sz(g.g); i++) for (auto e : g.g[i])
        if (vis[i] and !vis[e.to] and !e.res) cut.eb(i, e.to);
    return cut;
}
```

3.2. Blossom

```
struct Blossom { // O(E * V^2)
    struct struct_edge { int v; struct_edge* n; };
    typedef struct_edge* edge;
    int n;
    struct_edge pool[MAXE]; // 2 * n * n;
    edge top;
    vector<edge> g;
```

```
queue<int> q;
vector<int> f, base, inq, inb, inp, match;
vector<vector<int>> ed;

Blossom(int n) :
    n(n), match(n, -1), g(n), top(pool),
    f(n), base(n), inq(n), inb(n), inp(n),
    ed(n, vector<int>(n)) {
}

void add_edge(int u, int v) {
    if (ed[u][v]) return;
    ed[u][v] = 1;
    top->v = v, top->n = g[u], g[u] = top++;
    top->v = u, top->n = g[v], g[v] = top++;
}

int get_lca(int root, int u, int v) {
    fill(all(inp), 0);
    while (1) {
        inp[u = base[u]] = 1;
        if (u == root) break;
        u = f[match[u]];
    }
    while (1) {
        if (inp[v = base[v]]) return v;
        else v = f[match[v]];
    }
}

void mark(int lca, int u) {
    while (base[u] != lca) {
        int v = match[u];
        inb[base[u]] = 1;
        inb[base[v]] = 1;
        u = f[v];
        if (base[u] != lca) f[u] = v;
    }
}

void blossom_contraction(int s, int u, int v) {
    int lca = get_lca(s, u, v);
    fill(all(inb), 0);
    mark(lca, u); mark(lca, v);
    if (base[u] != lca) f[u] = v;
    if (base[v] != lca) f[v] = u;
    forn(u, n) {
        if (inb[base[u]]) {
            base[u] = lca;
            if (!inq[u]) {
                inq[u] = 1;
                q.push(u);
            }
        }
    }
}

int bfs(int s) {
    fill(all(inq), 0);
    fill(all(f), -1);
    forn(i, n) base[i] = i;
    q = queue<int>();
    q.push(s);
    inq[s] = 1;
    while (sz(q)) {
        int u = q.front(); q.pop();
        for (edge e = g[u]; e; e = e->n) {
            int v = e->v;
            if (base[u] != base[v] && match[u] != v) {
                if ((v == s) || (match[v] != -1 && f[match[v]] !=
                    -1))
                    blossom_contraction(s, u, v);
                else if (f[v] == -1) {
                    f[v] = u;
                    if (match[v] == -1) return v;
                    else if (!inq[match[v]]) {
                        inq[match[v]] = 1;
                        q.push(match[v]);
                    }
                }
            }
        }
    }
    return -1;
}

int doit(int u) {
    if (u == -1) return 0;
    int v = f[u];
    doit(match[v]);
    match[v] = u; match[u] = v;
    return u != -1;
}
```

```
    int matching() {
        int ans = 0;
        forn(u, n)
            ans += (match[u] == -1) && doit(bfs(u));
        return ans;
    }

    // (i < net.match[i]) => means match
    vector<pair<int,int>> get_edges() {
        vector<pair<int,int>> ans;
        forn(u, n) if (u < match[u])
            ans.pb({ u, match[u] });
        return ans;
    }
};
```

3.3. Hopcroft Karp

```
struct mbm { // O(E * sqrt(V))
    int nl, nr, flow = 0;
    vector<vector<int>> g;
    vector<int> dist, mfl, mfr;

    mbm(int nl, int nr) :
        nl(nl), nr(nr), g(nl), mfl(nl, -1),
        mfr(nr, -1), dist(nl) {
    }

    void add(int u, int v) { g[u].pb(v); }

    void bfs() {
        queue<int> q;
        forn(u, nl)
            if (!~mfl[u]) q.push(u), dist[u] = 0;
            else dist[u] = -1;
        while (sz(q)) {
            int u = q.front();
            q.pop();
            for (auto& v : g[u])
                if (~mfr[v] && !~dist[mfr[v]]) {
                    dist[mfr[v]] = dist[u] + 1;
                    q.push(mfr[v]);
                }
        }
    }
    bool dfs(int u) {
        for (auto& v : g[u])
            if (!~mfr[v]) {
                mfl[u] = v, mfr[v] = u;
                return true;
            }
        for (auto& v : g[u])
            if (dist[mfr[v]] == dist[u] + 1 && dfs(mfr[v])) {
                mfl[u] = v, mfr[v] = u;
                return true;
            }
        return false;
    }

    int get_matching() {
        while (true) {
            bfs();
            int agt = 0;
            forn(u, nl)
                if (!~mfl[u]) agt += dfs(u);
            if (!agt) break;
            flow += agt;
        }
        return flow;
    }

    pair<vector<int>, vector<int>> MVC() {
        vector<int> L, R;
        forn(u, nl)
            if (!~dist[u]) L.pb(u);
            else if (~mfl[u]) R.pb(mfl[u]);
        return { L, R };
    }

    vector<pair<int,int>> get_edges() {
        vector<pair<int,int>> ans;
        forn(u, nl)
            if (mfl[u] != -1)
                ans.pb({ u, mfl[u] });
        return ans;
    }
};
```

3.4. Matching

```
struct mbm { // O(V * E)
```

```
    int l, r;
    vector<int> mat;
    vector<bool> vis;
    vector<vector<int>> g;

    mbm(int l, int r) : l(l), r(r), mat(r), vis(l), g(l) {}

    bool match(int v) {
        if (vis[v]) return false;
        vis[v] = true;
        for (int& u : g[v]) {
            if (mat[u] == -1 || match(mat[u])) {
                mat[u] = v;
                return true;
            }
        }
        return false;
    }

    vector<pair<int,int>> matching() {
        vector<pair<int,int>> ans;
        fill(all(mat), -1);
        forn(i, l) {
            fill(all(vis), false);
            match(i);
        }
        forn(i, r) if (~mat[i]) ans.pb({ mat[i], i });
        return ans;
    }
};
```

3.5. Maximum flow minimum cost

```
struct mcmf {
    const ll INF = LONG_LONG_MAX;
    struct Edge { int to, rev; ll flo, cap, cost; };
    int n;
    vector<ll> p, dist;
    vector<pair<int, int>> pre;
    vector<vector<Edge>> g;

    mcmf(int m) : n(m), p(n), dist(n), pre(n), g(n) {}

    void add_edge(int v, int u, ll cap, ll cost) {
        g[v].pb({ u, sz(g[u]), 0, cap, cost });
        g[u].pb({ v, sz(g[v]) - 1, 0, 0, -cost });
    }

    bool path(int s, int t) {
        dist.assign(n, INF);
        using T = pair<ll, int>;
        priority_queue<T, vector<T>, greater<T>> todo;
        todo.push({ dist[s] = 0, s });
        while (sz(todo)) {
            T x = todo.top(); todo.pop();
            if (x.f > dist[x.s]) continue;
            for (auto& e : g[x.s]) {
                if (e.flo < e.cap && dist[e.to] > x.f + e.cost +
                    p[x.s] - p[e.to]) {
                    dist[e.to] = x.f + e.cost + p[x.s] - p[e.to];
                    pre[e.to] = { x.s, e.rev };
                    todo.push({ dist[e.to], e.to });
                }
            }
        }
        return dist[t] != INF;
    }

    pair<ll, ll> calc(int s, int t) {
        forn(_, n) forn(i, n) for (auto& e : g[i])
            if (e.cap) p[e.to] = min(p[e.to], p[i] + e.cost);
        ll totFlow = 0, totCost = 0;
        while (path(s, t)) {
            forn(i, n) p[i] += dist[i];
            ll df = INF;
            for (int x = t; x != s; x = pre[x].f) {
                Edge& e = g[pre[x].f][g[x][pre[x].s].rev];
                df = min(df, e.cap - e.flo);
            }
            totFlow += df; totCost += (p[t] - p[s]) * df;
            for (int x = t; x != s; x = pre[x].f) {
                Edge& e = g[x][pre[x].s]; e.flo -= df;
                g[pre[x].f][e.rev].flo += df;
            }
        }
        return { totFlow, totCost };
    }
};
```

3.6. Hungarian

```
template<typename T>
```

```
struct Hungarian { // O(V^3)
    int n, m;
    const T inf = 1e18;
    vector<T> u, v; vector<int> p, way;
    vector<vector<T>> g;

    Hungarian(int n, int m) :
        n(n), m(m), g(n + 1, vector<T>(m + 1, inf - 1)),
        u(n + 1), v(m + 1), p(m + 1), way(m + 1) {
    }

    void set(int u, int v, T w) { g[u + 1][v + 1] = w; }

    T assign() {
        forne(i, 1, n + 1) {
            int j0 = 0; p[0] = i;
            vector<T> minv(m + 1, inf);
            vector<char> used(m + 1, false);
            do {
                used[j0] = true;
                int i0 = p[j0], j1; T delta = inf;
                forne(j, 1, m + 1) if (!used[j]) {
                    T cur = g[i0][j] - u[i0] - v[j];
                    if (cur < minv[j]) minv[j] = cur, way[j] = j0;
                    if (minv[j] < delta) delta = minv[j], j1 = j;
                }
                forn(j, m + 1)
                    if (used[j]) u[p[j]] += delta, v[j] -= delta;
                    else minv[j] -= delta;
                j0 = j1;
            } while (p[j0]);
            do {
                int j1 = way[j0]; p[j0] = p[j1]; j0 = j1;
            } while (j0);
        }
        return -v[0];
    }
};
```

4. Geometry

4.1. isfigure

```
#include <bits/stdc++.h>
using namespace std;
#define endl      '\n'
#define f         first
#define s         second
#define ins       insert
#define pb        push_back
#define eb        emplace_back
#define sz(x)     int((x).size())
#define all(x)    begin(x), end(x)
typedef long long ll;
typedef unsigned long long ull;
#define forn(i, n) for (int i = 0; i < n; ++i)
#define each(i, x) for (auto &&i : x)
#define forne(i,x,n) for (int i = x; i < n; ++i)
#define show(x) for (auto &&i : x) {cerr << i <<' ';} cerr<< endl;


#define LOCAL
void dbg_out() { cerr << ']' << endl; }
template<typename Head, typename... Tail>
void dbg_out(Head H, Tail... T) { cerr << H;if (sizeof...(T)) cerr <<
    ',' << ' '; dbg_out(T...); }
#ifdef LOCAL
#define dbg(...) cerr << '|' << __LINE__ << '|'<< '{' << #__VA_ARGS__
    << '}'<<':'<<' '<<'[', dbg_out(__VA_ARGS__)
#else
#define dbg(...)
#endif



typedef ll T;
struct pt {
    T x, y;
    pt() : x(0), y(0) {}
    pt(T _x, T _y) : x(_x), y(_y) {}
    pt operator+(pt p) { return { x + p.x, y + p.y }; }
    pt operator-(pt p) { return { x - p.x, y - p.y }; }
    pt operator*(T d) { return { x * d, y * d }; }
    pt operator/(T d) { return { x / d, y / d }; }
    bool operator==(pt b) { return x == b.x && y == b.y; }
    bool operator!=(pt b) { return x != b.x || y != b.y; }
    bool operator<(pt b) { return x == b.x ? y < b.y : x < b.x; }

    void read() {
        cin >> x >> y;
    }
```

```
};
const double PI = acos(-1);
double DEG_TO_RAD(double n) { return n * PI / 180.0; }
double RAD_TO_DEG(double n) { return n * 180.0 / PI; }
T sq(pt p) { return p.x * p.x + p.y * p.y; }
T cross(pt v, pt w) { return v.x * w.y - v.y * w.x; }
double abs(pt p) { return sqrt(sq(p)); }
T dot(pt v, pt w) { return v.x * w.x + v.y * w.y; }
T dis(pt a, pt b) { return sq(a - b); }
//Transformaciones
pt translate(pt v, pt p) { return p + v; }
pt scale(pt c, double factor, pt p) { return c + (p - c) * factor; }
pt rot(pt p, double ang) { return { p.x * cos(ang) - p.y * sin(ang),
    p.x * sin(ang) + p.y * cos(ang) }; }
pt perp(pt p) { return { -p.y, p.x }; }
T isParall(pt v, pt w) { return cross(v, w) == 0; }


// A square has four right angles and four sides with equal lengths.
bool isSquare(pt a, pt b, pt c, pt d) {
    T ab = dis(a, b);
    T bc = dis(b, c);
    T cd = dis(c, d);
    T ad = dis(a, d);
    return isParall(a - b, c - d) && isParall(a - d, b - c) && dot(b -
        a, d - a) == 0 && ab == bc && bc == cd && cd == ad;
}

// A rectangle has four right angles.
bool isRectangle(pt a, pt b, pt c, pt d) {
    return isParall(a - b, c - d) && isParall(a - d, b - c) && dot(b -
        a, d - a) == 0;
}

// A rhombus has four sides with equal lengths.
bool isRhombus(pt a, pt b, pt c, pt d) {
    T ab = dis(a, b);
    T bc = dis(b, c);
    T cd = dis(c, d);
    T ad = dis(a, d);
    return ab == bc && bc == cd && cd == ad;
}

// A parallelogram has two pairs of parallel sides.
bool isParallelogram(pt a, pt b, pt c, pt d) {
    return isParall(a - b, c - d) && isParall(a - d, b - c);
}
```

```
// A trapezium has one pair of parallel sides.
bool isTrapezium(pt a, pt b, pt c, pt d) {
    return isParall(a - b, c - d) || isParall(a - d, b - c);
}

// A kite has reflection symmetry across a diagonal.
bool isKite(pt a, pt b, pt c, pt d) {
    T ab = dis(a, b);
    T bc = dis(b, c);
    T cd = dis(c, d);
    T ad = dis(a, d);
    return (ab == bc && cd == ad) || (ab == ad && bc == cd);
}
int main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);

    pt a, b, c, d;
    a.read(); b.read(); c.read(); d.read();

    if (isSquare(a, b, c, d))              cout << "square" << endl;
    else if (isRectangle(a, b, c, d))      cout << "rectangle" << endl;
    else if (isRhombus(a, b, c, d))        cout << "rhombus" << endl;
    else if (isParallelogram(a, b, c, d)) cout << "parallelogram" <<
        endl;
    else if (isTrapezium(a, b, c, d))      cout << "trapezium" << endl;
    else if (isKite(a, b, c, d))           cout << "kite" << endl;
    else cout << "none" << endl;









    cout << flush;
    return 0;
}
```

4.2. sweep line

```
/*
```

```
O(nlogn)
Par de puntos cuya distancia es la mas corta
ans = idx de los puntos en el vector de puntos dado
best = la mejor distancia entre dos puntos (la mas corta)
*/

struct P {
    //double para puntos con decimales ej (x , y) -> (1.234, 2.341)
    double x, y;
    int id;

    //int x, y, id;
};

struct Cx {
    bool operator()(const P& a, const P& b) const {
        return a.x < b.x || (a.x == b.x && a.y < b.y);
    }
};
struct Cy {
    bool operator()(const P& a, const P& b) const {
        return a.y < b.y;
    }
};

int n;
vector<P> a, buf;
double best;
pair<int, int> ans = { -1, -1 };

inline void upd(const P& u, const P& v) {
    double dx = u.x - v.x, dy = u.y - v.y;
    double d = sqrt(dx * dx + dy * dy);
    if (d < best) { best = d; ans = { u.id, v.id }; }
}

void rec(int l, int r) {
    if (r - l <= 3) {
        for (int i = l; i < r; i++)
            for (int j = i + 1; j < r; j++)
                upd(a[i], a[j]);
        sort(a.begin() + l, a.begin() + r, Cy());
        return;
    }
    int m = (l + r) >> 1;
    //double para puntos con decimales ej (x , y) -> (1.234, 2.341)
    double midx = a[m].x;
    //int midx = a[m].x;
    rec(l, m);
    rec(m, r);
    merge(a.begin() + l, a.begin() + m, a.begin() + m, a.begin() + r,
        buf.begin(), Cy());
    copy(buf.begin(), buf.begin() + (r - l), a.begin() + l);
    int sz = 0;
    for (int i = l; i < r; i++) {
        if (fabs(double(a[i].x) - midx) < best) {
            for (int j = sz - 1, k = 0; j >= 0 && k < 8; --j, ++k) {
                if ((a[i].y - buf[j].y) >= best) break;
                upd(a[i], buf[j]);
            }
            buf[sz++] = a[i];
        }
    }
}

//test con enteros

void test() {
    cin >> n;
    a.resize(n);
    buf.resize(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i].x >> a[i].y;
        a[i].id = i;
    }
    sort(all(a), Cx());
    best = 1e18;
    rec(0, n);
    int i = ans.f, j = ans.s;
    if (i > j) {
        swap(i, j);
    }
    cout << i << ' ' << j << ' ' << fixed << setprecision(6) << best
        << '\n';
}

//test con decimales

void testd() {
    int n;
    while (cin >> n) {
        if (n == 0) {
            break;
        }
```

```
        a.resize(n);
        buf.resize(n);
        vector<pair< string, string >> tempA;
        for (int i = 0; i < n; i++) {
            string sx, sy; cin >> sx >> sy;
            P p;
            p.x = stod(sx);
            p.y = stod(sy);
            p.id = i;
            a[i] = p;
            tempA.pb({ sx, sy });
        }
        buf.assign(n, {});
        sort(all(a), Cx());
        best = 1e300;
        ans = { 0, 0 };
        rec(0, n);
        int idx1 = ans.f, idx2 = ans.s;
        cout << tempA[idx1].f << ' ' << tempA[idx1].s << ' ' <<
            tempA[idx2].f << ' ' << tempA[idx2].s << endl;
    }


}
```

4.3. Polygon

```
// Requiere pt y line
enum {IN, OUT, ON};
struct polygon {

    vector<pt> p;
    polygon(int64_t n) : p(n) {}
    int64_t top = -1, bottom = -1;

    ld area(bool s = 0) {
        ld ans = 0;
        for (int i = 0, n = p.size(); i < n; i++) ans += cross(p[i],
            p[(i+1)%n]);
        ans /= 2;
        return s ? ans : abs(ans);
    }

    double perimeter() {
        ld per = 0;
        for(int i = 0, n = p.size(); i < n; i++) per += abs(p[i] -
            p[(i+1)%n]);
        return per;
    }

    bool above(pt a, pt p) { return p.y >= a.y; }

    bool crosses_ray(pt a, pt p, pt q) { // pq crosses ray from a
        return (above(a, q)-above(a, p)) * orient(a, p, q) > 0;
    }

    bool inDisk(pt a, pt b, pt p) {return dot(a-p, b-p) <= 0;}
    bool onSegment(pt a, pt b, pt p){return !orient(a, b, p) &&
        inDisk(a,b,p);}

    int64_t in_polygon(pt a) {
        int64_t crosses = 0;
        for(int i = 0, n = p.size(); i < n; i++) {
            if(onSegment(p[i], p[(i+1)%n], a)) return ON; //sobre el
                borde
            crosses += crosses_ray(a, p[i], p[(i+1)%n]);
        }
        return (crosses & 1 ? IN : OUT);
    }

    void normalize() { /// polygon is CCW
        bottom = min_element(all(p)) - p.begin();
        vector<pt> tmp(p.begin() + bottom, p.end());
        tmp.insert(tmp.end(), p.begin(), p.begin() + bottom);
        p.swap(tmp); bottom = 0;
        top = max_element(all(p)) - p.begin();
    }

    int64_t in_convex(pt a) {
        assert(bottom == 0 && top != -1);
        if(a < p[0] || p[top] < a) return OUT;
        ld orientation = orient(p[0], p[top], a);
        if(!orientation) {
            if(a == p[0] || a == p[top]) return ON;
            return top == 1 || top + 1 == p.size() ? ON : IN;
        } else if (orientation < 0) {
            auto it = lower_bound(p.begin() + 1, p.begin() + top, a);
            ld d = orient(*prev(it), a, *it);
            return d < 0 ? IN : (d > 0 ? OUT: ON);
        } else {
            auto it = upper_bound(p.rbegin(), p.rend() - top - 1, a);
            ld d = orient(*it, a, it == p.rbegin() ? p[0] : *prev(it));
```

```
        return d < 0 ? IN : (d > 0 ? OUT: ON);
    }
}
polygon cut(pt a, pt b) { // cuts polygon on line ab
    line l(a, b);
    polygon new_polygon(0);
    for(int i = 0, n = p.size(); i < n; ++i) {
        pt c = p[i], d = p[(i+1)%n];
        ld abc = cross(b-a, c-a), abd = cross(b-a, d-a);
        if(abc >= 0) new_polygon.p.push_back(c);
        if(abc * abd < 0) {
            pt out; inter(l, line(c, d), out);
            new_polygon.p.push_back(out);
        }
    }
    return new_polygon;
}

void convex_hull() {
    sort(all(p));
    vector<pt> ch;
    ch.reserve(p.size()+1);
    for(int it = 0; it < 2; it++) {
        int64_t start = ch.size();
        for(auto &a : p) {
            // if colineal are needed, use < and remove repeated
               points
            while(ch.size() >= start+2 && orient(ch[ch.size()-2],
                ch.back(), a) <= 0)
                ch.pop_back();
            ch.push_back(a);
        }
        ch.pop_back();
        reverse(p.begin(), p.end());
    }
    if(ch.size() == 2 && ch[0] == ch[1]) ch.pop_back();
    // if colineal are needed, use this
    //if(sz(ch) > sz(p)) ch.resize(p.size());
    p.swap(ch);
}

vector<pair<int64_t, int64_t>> antipodal() {
    vector<pair<int64_t, int64_t>> ans;
    int64_t n = p.size();
    if(n == 2) ans.push_back({0, 1});
    if(n < 3) return ans;
    auto nxt = [&](int x) { return (x+1 == n ? 0 : x+1); };
    auto area2 = [&](pt a, pt b, pt c) { return cross(b-a, c-a); };
    int64_t _b = 0;
    while(abs(area2(p[n - 1], p[0], p[nxt(_b)])) > abs(area2(p[n -
        1], p[0], p[_b]))) ++_b;
    for(int b = _b, a = 0; b != 0 && a <= _b; ++a) {
        ans.push_back({a, b});
        while (abs(area2(p[a], p[nxt(a)], p[nxt(b)])) >
            abs(area2(p[a], p[nxt(a)], p[b]))) {
            b = nxt(b);
            if(a != _b || b != 0) ans.push_back({ a, b });
            else return ans;
        }
        if(abs(area2(p[a], p[nxt(a)], p[nxt(b)])) ==
            abs(area2(p[a], p[nxt(a)], p[b]))) {
            if(a != _b || b != n-1) ans.push_back({ a, nxt(b) });
            else ans.push_back({ nxt(a), b });
        }
    }
    return ans;
}

pt centroid() {
    pt c{0, 0};
    ld scale = 6. * area(true);
    for(int i = 0, n = p.size(); i < n; ++i) {
        int64_t j = (i+1 == n ? 0 : i+1);
        c = c + (p[i] + p[j]) * cross(p[i], p[j]);
    }
    return c * (1.0 / scale);
}

int64_t pick() {
    int64_t boundary = 0;
    for(int i = 0, n = p.size(); i < n; i++) {
        int64_t j = (i+1 == n ? 0 : i+1);
        boundary += __gcd((int64_t)abs(p[i].x - p[j].x),
            (int64_t)abs(p[i].y - p[j].y));
    }
    return area() + 1 - boundary/2;
}

pt& operator[] (int64_t i){ return p[i]; }
};

ld areaTriangle(pt a, pt b, pt c) {
    return abs(cross(b-a, c-a)) / 2.0;
}
```

4.4. Line

```
// Requiere struct pt
struct line {
    pt v; ld c;

    //vector v and offset c
    line(pt v, ld c) : v(v), c(c) {}
    //ax+by=c
    line(ld a, ld b, ld c) : v({b, -a}), c(c) {}
    line(pt p, pt q) : v(q - p), c(cross(v, p)) {}
    // - these work with ld = int

    ld side(pt p) {return cross(v, p)-c;}
    ld dist(pt p) {return abs(side(p)) / abs(v);}
    ld sqDist(pt p) {return side(p) * side(p) / (ld)(v.norm());}
    line perpThrough(pt p) {return {p, p + perp(v)};}

    //Para ordenar pts sobre la linea
    bool cmpProj(pt p, pt q) {
        return dot(v, p) < dot(v, q);
    }

    line translate(pt t) {return {v, c + cross(v, t)};}
    // - these require ld = double
    line shiftLeft(double dist) {return {v, c + dist*abs(v)};}
    pt proj(pt p) {return p - (perp(v) * side(p)) * (1.0/(v.norm()));}
    pt refl(pt p) {return p - (perp(v) * 2 * side(p)) *
        (1.0/(v.norm()));}
};

bool inter(line l1, line l2, pt &out) {
    ld d = cross(l1.v, l2.v);
    if (d == 0) return false;
    out = (l2.v * l1.c - l1.v * l2.c) * (1.0 / d); // requires
        floating-point coordinates
    return true;
}

line bisector(line l1, line l2, bool interior) {
    assert(cross(l1.v, l2.v) != 0); // l1 and l2 cannot be parallel!
    ld sign = interior ? 1 : -1;
    return {l2.v * (1.0 / abs(l2.v)) + l1.v * (1.0 / abs(l1.v)) *
        sign, l2.c/abs(l2.v) + l1.c/abs(l1.v) * sign};
}
```

4.5. Circle

```
//Requiere pt y line

pt circumCenter(pt a, pt b, pt c) {
    b = b - a, c = c - a; // consider coordinates relative to A
    assert(cross(b,c) != 0); // no circumcircle if A,B,C aligned
    return a + perp(b * c.norm() - c * b.norm()) * (1.0 /
        cross(b,c)/2.0);
}
// (x- x0)^2 + (y-y0)^2
// (x0 + r cos(ang), y0 + r sin(ang))

template <typename ld> int64_t sgn(ld x) {
    return (ld(0) < x) - (x < ld(0));
}

int64_t circleLine(pt o, ld r, line l, pair<pt,pt> &out) {
    ld h2 = r * r - l.sqDist(o);
    if (h2 >= 0) { // the line touches the circle
        pt p = l.proj(o); // point P
        pt h = l.v*sqrt(h2) * (1.0 / abs(l.v)); // vector parallel to
            l, of
        //length h
        out = {p-h, p+h};
    }
    return 1 + sgn(h2);
}

int64_t circleCircle(pt o1, ld r1, pt o2, ld r2, pair<pt,pt> &out) {
    pt d=o2-o1; ld d2= d.norm();
    if (d2 == 0) {assert(r1 != r2); return 0;} // concentric circles
    ld pd = (d2 + r1 * r1 - r2 * r2)/2; // = |O_1P| * d
    ld h2 = r1 * r1 - pd * pd / d2; // = h^2
    if (h2 >= 0) {
        pt p = o1 + (d * pd)*(1.0 / d2), h = perp(d) * sqrt(h2/d2);
        out = {p-h, p+h};
    }
    return 1 + sgn(h2);
}

int64_t tangents(pt o1, ld r1, pt o2, ld r2, bool inner,
    vector<pair<pt,pt>> &out) {
    if (inner) r2 = -r2;
    pt d = o2-o1;
    ld dr = r1 - r2, d2 = d.norm(), h2 = d2 - dr * dr;
    if (d2 == 0 || h2 < 0) {assert(h2 != 0); return 0;}
```

```
    for (ld sign : {-1, 1}) {
        pt v = (d * dr + perp(d) * sqrt(h2) * sign) * (1.0 / d2);
        out.push_back({o1 + v * r1, o2 + v * r2});
    }
    return 1 + (h2 > 0);
}
```

4.6. Point

```
typedef long double T;
const T PI = acos(-1.0);
struct pt {
    T x, y;
    pt() : x(0), y(0) {}
    pt(T _x, T _y) : x(_x), y(_y) {}
    pt& operator+=(const pt& a) { x += a.x; y += a.y; return *this; }
    pt& operator-=(const pt& a) { x -= a.x; y -= a.y; return *this; }
    pt& operator*=(T mult) { x *= mult; y *= mult; return *this; }
    pt operator+(const pt& a) const { return pt(*this) += a; }
    pt operator-(const pt& a) const { return pt(*this) -= a; }
    pt operator*(T mult) const { return pt(*this) *= mult; }
    bool operator==(const pt& a) const { return x == a.x && y == a.y; }
    bool operator!=(const pt& a) const { return !(*this == a); }
    bool operator<(const pt& a) { return x == a.x ? y < a.y : x < a.x;
        }
    pt operator-() const { return pt(-x, -y); }
    pt rotate90() const { return pt(-y, x); }
    T norm() const { return (T)x * x + (T)y * y; }
    T dist() const { return sqrt(T(norm())); }
    bool top_half() const { return y > 0 || (y == 0 && x > 0); }
    friend ostream& operator<<(ostream& os, const pt& p) { return os
        << '(' << p.x << ", " << p.y << ')'; }
    friend istream& operator>>(istream& is, pt& p) { return is >> p.x
        >> p.y; }
};

T DEG_TO_RAD(T n) { return n * PI / 180.0; }
T RAD_TO_DEG(T n) { return n * 180.0 / PI; }
T abs(pt p) { return sqrt(p.norm()); }
pt perp(pt p) { return { -p.y, p.x }; }

// Producto Cruz
T cross(const pt& a, const pt& b) { return (T)a.x * b.y - (T)b.x *
    a.y; }
// Producto Escalar -> a * b = b * a -> (ang * a) * b = ang * (a * b)
    -> (a + b) * c = a * c + b * c
T dot(const pt& a, const pt& b) { return (T)a.x * b.x + (T)a.y * b.y; }
pt rot(pt p, double ang) { return { p.x * cos(ang) - p.y * sin(ang),
    p.x * sin(ang) + p.y * cos(ang) }; }
bool isPerp(pt v, pt w) { return !dot(v, w); }
// colinear == 0, left > 0, right < 0
T orient(pt a, pt b, pt c) { return cross(b - a, c - a); }
//Angulo(b-a, c-a), de 0 a 180
T angle(pt v, pt w) {
    T cosTheta = dot(v, w) / abs(v) / abs(w);
    return acos(max(T(-1.0), min(T(1.0), cosTheta)));
}

//De 0 a 360
T angle_complete(pt a, pt b, pt c) {
    pt ab = { b.x - a.x, b.y - a.y };
    pt cb = { b.x - c.x, b.y - c.y };
    T rslt = atan2(ab.y, ab.x) - atan2(cb.y, cb.x);
    return fabs((rslt * 180.0) / PI);
}

//Si un pt se encuentra dentro del angulo ABC
bool inAngle(pt a, pt b, pt c, pt p) {
    assert(orient(a, b, c) != 0);
    if (orient(a, b, c) < 0) swap(b, c);
    return orient(a, b, p) >= 0 && orient(a, c, p) <= 0;
}

T orientedAngle(pt a, pt b, pt c) {
    if (orient(a, b, c) >= 0) return angle(b - a, c - a);
    return 2 * PI - angle(b - a, c - a);
}

// Si un poligono es convexo
bool isConvex(vector<pt> p) {
    bool hasPos = 0, hasNeg = 0;
    for (int i = 0, n = p.size(); i < n; i++) {
        int64_t o = orient(p[i], p[(i + 1) % n], p[(i + 2) % n]);
        if (o > 0) hasPos = 1;
        if (o < 0) hasNeg = 1;
    }
    return !(hasPos && hasNeg);
}

// Devuelve el doble del area formada por tres puntos de un triangulo.
    Positivo cuando a -> b -> c es un giro a la izquierda.
```

```
T area_signed_2x(const pt& a, const pt& b, const pt& c) { return
    cross(b - a, c - a); }

T distance_to_line(const pt& p, const pt& a, const pt& b) {
    assert(a != b);
    return T(abs(area_signed_2x(p, a, b))) / (a - b).dist();
}

T manhattan_dist(const pt& a, const pt& b) {
    return (T)abs(a.x - b.x) + abs(a.y - b.y);
}
T euclidean_dist(const pt& a, const pt& b) {
    return (a - b).dist();
}

T infinity_norm_dist(const pt& a, const pt& b) {
    return max(abs(a.x - b.x), abs(a.y - b.y));
}

// Ordenar en orden creciente de y, deshaciendo los empates en orden
    creciente de x.
bool yx_compare(const pt& a, const pt& b) {
    return make_pair(a.y, a.x) < make_pair(b.y, b.x);
}
```

5. Graph

5.1. cycle len

```
// constexpr int mxN = 2500 + 50;
constexpr int inf = 1e9 + 7;
vector<int>adj[mxN];
int n, m;
int cycle_len(int start) {
    int ans = inf;

    vector<int> dist(n, -1);
    queue<int> bfs;

    dist[start] = 0;
    bfs.push(start);

    while (!bfs.empty()) {
        int node = bfs.front();
        bfs.pop();
        for (int adj_node : adj[node]) {
            if (dist[adj_node] == -1) {
                dist[adj_node] = dist[node] + 1;
                bfs.push(adj_node);
            }
            else if (dist[adj_node] >= dist[node]) {
                ans = min(ans, 1 + dist[adj_node] + dist[node]);
            }
        }
    }
    return ans;
}
```

5.2. Topo Sort DFS

```
int n, m; cin >> n >> m;
vector<int> ady[n];
forn (i, m) {
  int v, u; cin >> v >> u;
  v--, u--;
  ady[v].pb(u);
}

vector<int> topo;
vector<bool> vis(n);

function<void(int)> dfs = [&](int v) {
  vis[v] = true;
  for (int &u : ady[v]) {
    if (!vis[u]) dfs(u);
  }
  topo.pb(v);
};

forn (i, n) if (!vis[i]) dfs(i);
```

5.3. Topo Sort BFS

```
int n, m; cin >> n >> m;
vector<int> ady[n];
vector<int> grado(n);
forn (i, m) {
    int v, u; cin >> v >> u;
```

```
    v--, u--;
    ady[v].pb(u);
    grado[u]++;
}

vector<int> topo;
queue<int> qu;

forn (i, n) if (!grado[i]) qu.push(i);

while (sz(qu)) {
  int v = qu.front();
  qu.pop();
  topo.pb(v);
  for (int &u : ady[v]) {
    if (--grado[u] == 0) {
      qu.push(u);
    }
  }
}
```

5.4. Kosaraju

```
int n, m; cin >> n >> m;

vector<int> ady[n], rady[n];
vector<int> grado(n);

forn(i, m) {
  int a, b; cin >> a >> b;
  a--, b--;
  ady[a].pb(b);
  rady[b].pb(a);
}

vector<int> order;
vector<bool> vis(n);
vector<vector<int>> comp;

function<void(int)> dfs1 = [&](int v) {
  vis[v] = true;
  for (int& u : ady[v]) {
    if (!vis[u]) {
      dfs1(u);
    }
  }
  order.pb(v);
};

forn(i, n) if (!vis[i]) dfs1(i);

vis.assign(n, false);

function<void(int)> dfs2 = [&](int v) {
  vis[v] = true;
  comp.back().pb(v);
  for (int& u : rady[v]) {
    if (!vis[u]) {
      dfs2(u);
    }
  }
};

for (int i = n - 1; i >= 0; --i) {
  if (!vis[order[i]]) {
    comp.pb({});
    dfs2(order[i]);
  }
}

forn(i, sz(comp)) {
  cout << "Component #" << i + 1 << ":";
  for (int& j : comp[i]) {
    cout << " " << j + 1;
  }
  cout << endl;
}
```

5.5. Floyd Warshall

```
int n; cin >> n;
int ady[n][n];
const int INF = int(1e9);

forn (i, n) {
  forn (j, n) {
    ady[i][j] = (i == j ? 0 : INF);
  }
}

forn (i, n) {
  int v, u, w; cin >> v >> u >> w;
```

```
  v--, u--;
  ady[v][u] = ady[u][v] = w;
}

forn (k, n) {
  forn (i, n) {
    forn (j, n) {
      ady[i][j] = min(ady[i][j], ady[i][k] + ady[k][j]);
    }
  }
}
```

5.6. ArtiBridges

```
struct ArtiBridges {
    int n, timer;
    vector<bool> vis, is_articulation;
    vector<int> tin, low;
    vector<pair<int,int>> bridges;

    ArtiBridges(int m) :
        n(m), timer(0), vis(n), tin(n, -1),
        low(n, -1), is_articulation(n) {
        forn(i, n) if (!vis[i]) dfs(i);
    }

    void dfs(int v, int p = -1) {
        vis[v] = true;
        tin[v] = low[v] = timer++;
        int children = 0;
        for (int& u : g[v]) {
            if (u == p) continue;
            if (vis[u]) {
                low[v] = min(low[v], tin[u]);
            }
            else {
                dfs(u, v);
                low[v] = min(low[v], low[u]);
                if (low[u] >= tin[v] && p != -1)
                    is_articulation[v] = true;
                ++children;

                if (low[u] > tin[v])
                    bridges.pb({ v, u });
            }
        }
        if (p == -1 && children > 1)
            is_articulation[v] = true;
    }
};
```

5.7. Biconnected Components

```
struct BiConn {
    int n, timer;
    vector<bool> vis;
    vector<int> tin, low;
    stack<pair<int,int>> stk;
    vector<vector<pair<int,int>>> bcc;

    BiConn(int m) :
        n(m), timer(0), vis(n), tin(n, -1),
        low(n, -1) {
        forn(i, n) if (!vis[i]) dfs(i);
    }

    void dfs(int v, int p = -1) {
        vis[v] = true;
        tin[v] = low[v] = timer++;
        for (int& u : g[v]) {
            if (u == p) continue;
            if (vis[u]) {
                low[v] = min(low[v], tin[u]);
                if (tin[u] < tin[v]) stk.push({ v, u });
            }
            else {
                stk.push({ v, u });
                dfs(u, v);
                low[v] = min(low[v], low[u]);
                if (low[u] >= tin[v]) {
                    vector<pair<int,int>> comp;
                    pair<int,int> edge;
                    do {
                        edge = stk.top(); stk.pop();
                        comp.pb(edge);
                    } while (edge != make_pair(v, u));
                    bcc.pb(comp);
                }
            }
        }
    }
};
```

5.8. Dijkstra

```
struct edge {
  int v; ll w;

  bool operator < (const edge &x) const {
    return x.w < w;
  }
};

vector<ll> dist(n, LONG_LONG_MAX);

auto dijkstra = [&](edge v) {
  priority_queue<edge> pq;
  pq.push(v);
  dist[v.v] = 0;
  while (sz(pq)) {
    v = pq.top();
    pq.pop();
    if (v.w > dist[v.v]) continue;
    for (edge &u : g[v.v]) {
      if (dist[u.v] > dist[v.v] + u.w) {
        dist[u.v] = dist[v.v] + u.w;
        pq.push({u.v, dist[u.v]});
      }
    }
  }
};
```

5.9. Kruskal

```
struct edge {
  int v, u, w;

  bool operator < (const edge &x) const {
    return w < x.w;
  }
};

vector<edge> edges;
int n, m; cin >> n >> m;
forn (i, m) {
  int v, u, w; cin >> v >> u >> w;
  v--, u--;
  edges.push_back({v, u, w});
}
sort(all(edges));
dsu UF(n);
int nodes = 0, mst = 0;
for (edge &i : edges) {
  if (!UF.same(i.v, i.u)) {
    mst += i.w;
    UF.unite(i.v, i.u);
    nodes++;
  }
  if (nodes == n - 1) break;
}
```

5.10. Prim

```
struct edge {
  int v, w;

  bool operator < (const edge &x) const {
    return w > x.w;
  }
};

int n, m; cin >> n >> m;
vector<edge> ady[n];
forn (i, m) {
  int v, u, w; cin >> v >> u >> w;
  v--, u--;
  ady[v].pb({u, w});
  ady[u].pb({v, w});
}

priority_queue<edge> pq;
bool vis[n];
memset(vis, false, sizeof vis);

vis[0] = true;
for (edge &i : ady[0]) if (!vis[i.v]) pq.push(i);

int mst = 0;
while (sz(pq)) {
  edge v = pq.top();
  pq.pop();

  if (!vis[v.v]) {
    mst += v.w;
```

```
    vis[v.v] = true;
    for (edge &i : ady[v.v]) {
      if (!vis[i.v]) {
        pq.push(i);
      }
    }
  }
}
```

5.11. Bellman Ford

```
struct Edge { int v, u; ll w; };

const ll INF = 1e18;
vector<Edge> edges;
vector<ll> d;
vector<int> p;

vector<int> BellmanFord(int n, int src = -1) {
  d.assign(n, ~src ? INF : 0);
  if (~src) d[src] = 0;
  p.assign(n, -1);
  int x = -1;
  forn (i, n) {
    x = -1;
    for (Edge &e : edges)
      if (d[e.v] < INF)
        if (d[e.u] > d[e.v] + e.w) {
          d[e.u] = max(-INF, d[e.v] + e.w);
          p[e.u] = e.v;
          x = e.u;
        }
  }
  if (x == -1) return {};
  forn (i, n) x = p[x];
  vector<int> path;
  for (int cur = x;; cur = p[cur]) {
    path.pb(cur);
    if (cur == x && sz(path) > 1)
      break;
  }
  reverse(all(path));
  return path;
}

vector<int> BellmanFord(int n, int s, int t) {
  d.assign(n, INF);
  d[s] = 0;
  p.assign(n, -1);
  while (1) {
    bool any = false;
    for (Edge &e : edges)
      if (d[e.v] < INF)
        if (d[e.u] > d[e.v] + e.w) {
          d[e.u] = d[e.v] + e.w;
          p[e.u] = e.v;
          any = true;
        }
    if (!any) break;
  }
  if (d[t] == INF) return {};
  vector<int> path;
  for (int cur = t; cur != -1; cur = p[cur])
    path.pb(cur);
  reverse(all(path));
  return path;
}
```

5.12. LCA Binary Lifting

```
struct LCA {
  int timer, l, n;
  vector<int> tin, tout;
  vector<vector<int>> up;

  LCA(int n, int root = 0) {
    timer = 0;
    this->n = n;
    tin.resize(n);
    tout.resize(n);
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(root, root);
  }

  void dfs(int v, int p) {
    tin[v] = ++timer;
    up[v][0] = p;
    forn (i, l) up[v][i + 1] = up[up[v][i]][i];
    for (int &u : g[v]) if (u != p) dfs(u, v);
    tout[v] = ++timer;
  }
```

```
  bool is_ancestor(int v, int u) {
    return tin[v] <= tin[u] && tout[v] >= tout[u];
  }

  int lca(int v, int u) {
    if (is_ancestor(v, u)) return v;
    if (is_ancestor(u, v)) return u;
    rforn (i, l)
      if (!is_ancestor(up[u][i], v))
        u = up[u][i];
    return up[u][0];
  }
};
```

5.13. LCA

```
struct LCA {
  vector<int> height, euler, first, segtree;
  int n;

  LCA(vector<vector<int>> &g, int root = 0) {
    n = sz(g);
    height.resize(n);
    first.resize(n);
    dfs(g, root, root);
    int m = sz(euler);
    segtree.resize(m * 4);
    build(1, 0, m - 1);
  }

  void dfs(vector<vector<int>> &g, int v, int p, int h = 0) {
    height[v] = h;
    first[v] = sz(euler);
    euler.pb(v);
    for (int &u : g[v]) {
      if (u == p) continue;
      dfs(g, u, v, h + 1);
      euler.pb(v);
    }
  }

  void build(int node, int b, int e) {
    if (b == e) {
      segtree[node] = euler[b];
    } else {
      int mid = (b + e) / 2;
      build(node << 1, b, mid);
      build(node << 1 | 1, mid + 1, e);
      int l = segtree[node << 1], r = segtree[node << 1 | 1];
      segtree[node] = (height[l] < height[r]) ? l : r;
    }
  }

  int query(int node, int b, int e, int L, int R) {
    if (b > R || e < L) return -1;
    if (b >= L && e <= R) return segtree[node];
    int mid = (b + e) >> 1;
    int left = query(node << 1, b, mid, L, R);
    int right = query(node << 1 | 1, mid + 1, e, L, R);
    if (left == -1) return right;
    if (right == -1) return left;
    return height[left] < height[right] ? left : right;
  }

  int lca(int u, int v) {
    int left = first[u], right = first[v];
    if (left > right) swap(left, right);
    return query(1, 0, sz(euler) - 1, left, right);
  }
};
```

5.14. Two Sat

```
struct sat {
    int n, tot;
    vector<vector<int>> g;
    vector<int> vis, comp, id, ans;
    stack<int> s;

    sat() {}
    sat(int n_) : n(n_), tot(n), g(2*n) {}

    int dfs(int i, int& t) {
        int lo = id[i] = t++;
        s.push(i), vis[i] = 2;
        for (int j : g[i]) {
            if (!vis[j]) lo = min(lo, dfs(j, t));
            else if (vis[j] == 2) lo = min(lo, id[j]);
        }
        if (lo == id[i]) while (1) {
            int u = s.top(); s.pop();
```

```
            vis[u] = 1, comp[u] = i;
            if ((u>>1) < n and ans[u>>1] == -1) ans[u>>1] = ~u&1;
            if (u == i) break;
        }
        return lo;
    }

    void add_impl(int x, int y) { // x -> y = !x ou y
        x = x >= 0 ? 2*x : -2*x-1;
        y = y >= 0 ? 2*y : -2*y-1;
        g[x].push_back(y);
        g[y^1].push_back(x^1);
    }
    void add_cl(int x, int y) { // x ou y
        add_impl(~x, y);
    }
    void add_xor(int x, int y) { // x xor y
        add_cl(x, y), add_cl(~x, ~y);
    }
    void add_eq(int x, int y) { // x = y
        add_xor(~x, y);
    }
    void add_true(int x) { // x = T
        add_impl(~x, x);
    }
    void at_most_one(vector<int> v) { // no max um verdadeiro
        g.resize(2*(tot+v.size()));
        for (int i = 0; i < v.size(); i++) {
            add_impl(tot+i, ~v[i]);
            if (i) {
                add_impl(tot+i, tot+i-1);
                add_impl(v[i], tot+i-1);
            }
        }
        tot += v.size();
    }

    pair<bool, vector<int>> solve() {
        ans = vector<int>(n, -1);
        int t = 0;
        vis = comp = id = vector<int>(2*tot, 0);
        for (int i = 0; i < 2*tot; i++) if (!vis[i]) dfs(i, t);
        for (int i = 0; i < tot; i++)
            if (comp[2*i] == comp[2*i+1]) return {false, {}};
        return {true, ans};
    }
};
```

5.15. Tarjan

```
struct Tarjan {
  vector<int> low, num, comp;
  stack<int> st;
  int n, scc, cont;
  const int INF = int(1e9);

  Tarjan(int n) {
    this->n = n;
    low.resize(n);
    num.assign(n, -1);
    comp.resize(n);
    scc = cont = 0;
  }

  void dfs(int v) {
    low[v] = num[v] = cont++;
    st.push(v);
    for (int &u : g[v]) {
      if (num[u] == -1) dfs(u);
      low[v] = min(low[v], low[u]);
    }
    if (low[v] == num[v]) {
      int u;
      do {
        u = st.top(); st.pop();
        low[u] = INF;
        comp[u] = scc;
      } while (u != v);
      scc++;
    }
  };

  void go() {
    forn (i, n)
      if (num[i] == -1) dfs(i);
  }
};
```

5.16. bipartite Graph

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

```
    vector<vector<T>> adj;
    vector<T> side;
    Graph(int size) {
        n = size;
        adj.resize(n);
        side.resize(n, -1);
    }

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

    bool is_bipartite() {
        bool check = true;
        queue<int> q;
        for (int edge = 0; edge < n; ++edge) {
            if (side[edge] == -1) {
                q.push(edge);
                side[edge] = 0;
                while (q.size()) {
                    int curr = q.front();
                    q.pop();
                    for (auto neig : adj[curr]) {
                        if (side[neig] == -1) {
                            side[neig] = (1 ^ side[curr]);
                            q.push(neig);
                        }
                        else {
                            check &= (side[neig] != side[curr]);
                        }
                    }
                }
            }
        }
        return check;
    }
};
```

5.17. nx-ny-8

```
vector<vector<char>>board;
vector<vector<bool>>vis;
int n, m;
// U,UR, R,RD,D,LD,L, UL
int dx[8] = { -1, -1, 0, 1, 1, 1, 0, -1 };
int dy[8] = { 0, 1, 1, 1, 0, -1, -1, -1 };

void init() {
    board.resize(n + 1, vector<char>(m + 1));
    vis.resize(n + 1, vector<bool>(m + 1, 0));
}

void back(int x, int y) {
    vis[x][y] = 1;
    forn(i, 8) {
        int nx = x + dx[i], ny = y + dy[i];
        if (nx >= 0 && nx < n && ny >= 0 && ny < m && board[nx][ny] !=
            '1' && !vis[nx][ny]) back(nx, ny);
    }
}
```

5.18. nx-ny-4

```
vector<vector<char>>board;
vector<vector<bool>>vis;
int n, m;
// R D L U
int dx[] = { 0, 1, 0, -1 };
int dy[] = { 1, 0, -1, 0 };
void init() {
    board.resize(n + 1, vector<char>(m + 1));
    vis.resize(n + 1, vector<bool>(m + 1, 0));
}

void back(int x, int y) {
    vis[x][y] = 1;
    forn(i, 4) {
        int nx = x + dx[i], ny = y + dy[i];
        if (nx >= 0 && nx < n && ny >= 0 && ny < m && board[nx][ny] !=
            '1' && !vis[nx][ny]) back(nx, ny);
    }
}
```

6. Math

6.1. TernarySearch

```cpp
double f(double x) {
    return x;
}
// ternary_search(0.0, posibleMaximo)
double ternary_search(double l, double r) {
    double eps = 1e-9;
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);
        double f2 = f(m2);
        // if (f1 > c) f1 = f2 minimizar;
        if (f1 < f2) l = m1;
        else r = m2;
    }

    // return l;
    return f(l);
}
```

6.2. Discrete root

```cpp
//find all x -> x^k = a mod n
vector<int> discrete_root(int k, int a, int n) {
    int g = primitive_root(n);
    int gk = binpow(g, k, n);
    int y = discrete_log(gk, a, n);
    int x = binpow(g, y, n);//first solution
    int phin = phi(n);
    int delta = phin / __gcd(k, phin);

    vector<int> v;
    for (int i = 0; i < n - 1; i += delta) {
        x = binpow(g, y + i, n);
        v.pb(x);
    }
    return v;
}
```

6.3. Pollards Rho

```cpp
#define int int64_t
int mul(int a, int b, int m) {
    int ret = a * b - int((long double)1 / m * a * b + 0.5) * m;
    return ret < 0 ? ret + m : ret;
}

int pow(int x, int y, int m) {
    if (!y) return 1;
    int ans = pow(mul(x, x, m), y / 2, m);
    return y % 2 ? mul(x, ans, m) : ans;
}

bool prime(int n) {
    if (n < 2) return 0;
    if (n <= 3) return 1;
    if (n % 2 == 0) return 0;
    int r = __builtin_ctzll(n - 1), d = n >> r;
    for (int a : {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) {
        int x = pow(a, d, n);
        if (x == 1 or x == n - 1 or a % n == 0) continue;
        for (int j = 0; j < r - 1; j++) {
            x = mul(x, x, n);
            if (x == n - 1) break;
        }
        if (x != n - 1) return 0;
    }
    return 1;
}

int rho(int n) {
    if (n == 1 or prime(n)) return n;
    auto f = [n](int x) {return mul(x, x, n) + 1;};

    int x = 0, y = 0, t = 30, prd = 2, x0 = 1, q;
    while (t % 40 != 0 or gcd(prd, n) == 1) {
        if (x == y) x = ++x0, y = f(x);
        q = mul(prd, abs(x - y), n);
        if (q != 0) prd = q;
        x = f(x), y = f(f(y)), t++;
    }
    return gcd(prd, n);
}

vector<int> fact(int n) {
    if (n == 1) return {};
    if (prime(n)) return { n };
    int d = rho(n);
```

```
    vector<int> l = fact(d), r = fact(n / d);
    l.insert(l.end(), r.begin(), r.end());
    return l;
}
```

6.4. Factorization Sieve

```
#define int int64_t
constexpr int MAX = 1e6;
int primediv[MAX]; //10^6
vector<int> primes;

void sieve() {
    forn(i, MAX) primediv[i] = i;
    int root = sqrt(MAX) + 1;
    forne(i, 2, MAX) {
        if (primediv[i] != i) continue;
        primes.pb(i);
        if (i > root) continue;
        for (int j = i * i; j < MAX; j += i) primediv[j] = i;
    }
}

map<int, int> factorize(int n) { //n <= 10^12
    map<int, int> factors;
    for (int i = 0; i < primes.size() && n >= MAX; ++i) {
        while (n % primes[i] == 0) {
            factors[primes[i]]++;
            n /= primes[i];
        }
    }
    if (n >= MAX) {
        factors[n]++;
        return factors;
    }
    while (n > 1) {
        factors[primediv[n]]++;
        n /= primediv[n];
    }
    return factors;
}
```

6.5. squares in a circle

```
#include <bits/stdc++.h>
using namespace std;
#define endl      '\n'
#define f         first
#define s         second
#define ins       insert
#define pb        push_back
#define eb        emplace_back
#define sz(x)     int((x).size())
#define all(x)    begin(x), end(x)
typedef long long ll;
typedef unsigned long long ull;
#define forn(i, n) for (int i = 0; i < n; ++i)
#define each(i, x) for (auto &&i : x)
#define forne(i,x,n) for (int i = x; i < n; ++i)
#define show(x) for (auto &&i : x) {cerr << i <<' ';} cerr<< endl;

void dbg_out() { cerr << ']' << endl; }
template<typename Head, typename... Tail>
void dbg_out(Head H, Tail... T) { cerr << H;if (sizeof...(T)) cerr <<
    ',' << ' '; dbg_out(T...); }
#ifdef LOCAL
#define dbg(...) cerr << '|' << __LINE__ << '|'<< '{' << #__VA_ARGS__
    << '}'<<':'<<' '<<'[', dbg_out(__VA_ARGS__)
#else
#define dbg(...)
#endif
ll get(ll mid){
    ll ans=0;
    for(ll i=1; i*i < mid; i++){
        ans+=4*floor(sqrt( mid-i*i));
    }

    return ans;

}

const int inf= 1e9+7;
int main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    int test=1;
    #ifdef LOCAL
        freopen("in.txt", "r", stdin);
        freopen("out.txt", "w", stdout);
        test=2;
    #endif
```

```
    while(test--){
        ll n; cin>>n;
        ll r=4*n, l=1;
        double pre= 1.0*inf;
        while(l<=r){
            ll mid=(l+r)>>1;
            if(get(mid)>n){
                r=mid-1;
                pre=min(pre,sqrt(mid));
            }else l=mid+1;
        }
        cout<<fixed<< setprecision(30)<<pre<<endl;


    }

    cout << flush;
    return 0;
}
```

6.6. Discrete Log

```
// Returns minimum x for which a ^ x % m = b % m, a and m are coprime.
int discrete_log_coprime(int a, int b, int m) {
    int n = sqrt(m) + 1, an = 1;
    unordered_map<int, int> mapa;

    forn(i, n) an = (an * a) % m;

    int aq = b, anp = 1;
    forn(q, n + 1) {
        mapa[aq] = q;
        aq = (aq * a) % m;
    }

    forabe(p, 1, n) {
        anp = (anp * an) % m;
        if (mapa.count(anp))
            return n * p - mapa[anp];
    }
    return -1;
}

int discrete_log_nocoprime(int a, int b, int m) {
    if (a == 0) return (b == 0) ? 1 : -1;
    a %= m;  b %= m;
    int k = 1, add = 0, g;
    while ((g = __gcd(a, m)) > 1) {
        if (b == k) return add;
        if (b % g) return -1;
        b /= g;  m /= g;  add++;
        k = (k * a / g) % m;
    }

    int n = sqrt(m) + 1, an = 1, aq = b, anp = k;
    unordered_map<int, int> mapa;

    forn(i, n) an = (an * a) % m;

    forn(q, n + 1) {
        mapa[aq] = q;
        aq = (aq * a) % m;
    }

    forne(p, 1, n + 1) {
        anp = (anp * an) % m;
        if (mapa.count(anp))
            return n * p - mapa[anp] + add;
    }
    return -1;
}
```

6.7. Chinese Remainder Theorem

```
//a === b1 % m1
//a === b2 % m2
ll CRT(ll b1, ll b2, ll m1, ll m2) {
    ll x, y;
    ll c = gcd(m1, m2, x, y);
    ll a = b2 * x * m1 + b1 * y * m2;
    ll mod = m1 * m2;
    a = a % mod;
    if (a < 0) a += mod;
    return a;
}
//ff -> b, ss -> m, all m are coprimes
int CRT_general(vector<ii> &c){
    int a = 0, M = 1;
    forn(i, sz(c))
        M *= c[i].ss;
    forn(i, sz(c))
```

```
        a = (a + modular_inverse(M/c[i].ss, c[i].ss)
             * c[i].ff * (M/c[i].ss)) % M;
    return a;
}
```

6.8. Primitive root

```
//find g -> (g^k = a mod m) for all a -> gcd(a, m)=1
int primitive_root(int m) {
    int phin = phi(m);
    map<int, int> factors = factorize(phin);

    forn(i, 1, m) {
        bool ok = true;
        for (auto it : factors) {
            ok = ok && binpow(i, phin / it.f, m) != 1;
            if (!ok) break;
        }
        if (ok) return i;
    }
    return -1;
}
```

6.9. polynomial

```
//zx^n+...+cx^2+bx+a
typedef int tp; // type of polynomial
template<class T = tp>
struct poly {  // poly<> : 1 variable, poly<poly<>>: 2 variables, etc.
    vector<T> c;
    T& operator[](int k) { return c[k]; }
    poly(vector<T>& c) :c(c) {}
    poly(initializer_list<T> c) :c(c) {}
    poly(int k) :c(k) {}
    poly() {}
    poly operator+(poly<T> o) {
        int m = c.size(), n = o.c.size();
        poly res(max(m, n));
        forn(i, m)res[i] = res[i] + c[i];
        forn(i, n)res[i] = res[i] + o.c[i];
        return res;
    }
    poly operator*(tp k) {
        poly res(c.size());
        forn(i, c.size())res[i] = c[i] * k;
        return res;
    }
    poly operator*(poly o) {
        int m = c.size(), n = o.c.size();
        poly res(m + n - 1);
        forn(i, m)forn(j, n)res[i + j] = res[i + j] + c[i] * o.c[j];
        return res;
    }
    poly operator-(poly<T> o) { return *this + (o * -1); }
    T operator()(tp v) {
        T sum(0);
        for (int i = c.size() - 1;i >= 0;--i)sum = sum * v + c[i];
        return sum;
    }
};
// example: p(x,y)=2*x^2+3*x*y-y+4
// poly<poly<>> p={{4,-1},{0,3},{2}}
// printf("%d\n",p(2)(3)) // 27 (p(2,3))
set<tp> roots(poly<> p) { // only for integer polynomials
    set<tp> r;
    while (!p.c.empty() && !p.c.back())p.c.pop_back();
    if (!p(0))r.insert(0);
    if (p.c.empty())return r;
    tp a0 = 0, an = abs(p[p.c.size() - 1]);
    for (int k = 0;!a0;a0 = abs(p[k++]));
    vector<tp> ps, qs;
    forne(i, 1, sqrt(a0) + 1)if (a0 % i == 0)ps.pb(i), ps.pb(a0 / i);
    forne(i, 1, sqrt(an) + 1)if (an % i == 0)qs.pb(i), qs.pb(an / i);
    for (auto pt : ps)for (auto qt : qs)if (pt % qt == 0) {
        tp x = pt / qt;
        if (!p(x))r.insert(x);
        if (!p(-x))r.insert(-x);
    }
    return r;
}
pair<poly<>, tp> ruffini(poly<> p, tp r) { // returns pair (result,rem)
    int n = p.c.size() - 1;
    vector<tp> b(n);
    b[n - 1] = p[n];
    for (int k = n - 2;k >= 0;--k)b[k] = p[k + 1] + r * b[k + 1];
    return { poly<>(b),p[0] + r * b[0] };
}
// only for double polynomials
pair<poly<>, poly<> > polydiv(poly<> p, poly<> q) { // returns pair
    (result,rem)
    int n = p.c.size() - q.c.size() + 1;
```

```
    vector<tp> b(n);
    for (int k = n - 1;k >= 0;--k) {
        b[k] = p.c.back() / q.c.back();
        forn(i, q.c.size())p[i + k] -= b[k] * q[i];
        p.c.pop_back();
    }
    while (!p.c.empty() && abs(p.c.back()) < EPS)p.c.pop_back();
    return { poly<>(b),p };
}
// only for double polynomials
poly<> interpolate(vector<tp> x, vector<tp> y) { //TODO TEST
    poly<> q = { 1 }, S = { 0 };
    for (tp a : x)q = poly<>({ -a,1 }) * q;
    forn(i, x.size()) {
        poly<> Li = ruffini(q, x[i]).ff;
        Li = Li * (1.0 / Li(x[i])); // change for int polynomials
        S = S + Li * y[i];
    }
    return S;
}
vector<ll> coef(vector<ll> roots, bool first = true) {
    int l = roots.size() + 1;
    vector<ll> c(10002, 0), m(10002, 0);
    c[0] = 1;
    forn(k, roots.size()) {
        forne(i, 1, l) m[i] = c[i] + c[i - 1] * roots[k];
        forne(i, 1, l) c[i] = m[i];
    }
    ll sign = first ? 1 : -1;
    forn(i, roots.size()) {
        sign *= -1LL;
        m[i + 1] *= sign;
    }
    return m;
}
inline ll modn(ll x) { x = x % mod; if (x < 0ll) x += mod; return x; }
//interpolate for consecutive values X and evaluate at K;
ll interpolateAndEvaluate(ll k, int inix, vector<ll>& y) {
    ll den = 1, num = 1;
    int len = inix + sz(y) - 1;

    forne(i, inix, len) {
        num = (num * (k - (i + 1))) % mod;
        den = (den * modn(-1ll * i)) % mod;
    }

    ll res = (y[0] * divmod(num, den)) % mod;

    forne(i, inix, len) {
        num = divmod(num, k - (i + 1));
        num = (num * (k - i)) % mod;
        den = divmod(den, modn(-1ll * (sz(y) - i)));
        den = (den * i) % mod;
        res = (res + (y[i] * divmod(num, den)) % mod) % mod;
    }
    return res;
}
```

6.10. LinearDiophantineEquations

```
bool find_any_solution(int a, int b, int c, int& x, int& y, int& g) {
    g = extEuclid(a, b, x, y);
    if (c % g)
        return false;
    x *= (c / g);
    y *= (c / g);
    return true;
}

void find_all_solutions(int a, int b, int c) {
    int x, y, g, x0, y0;
    if (!find_any_solution(a, b, c, x, y, g))
        return;
    forne(i, -10, 10) {
        x0 = x + i * (b / g);
        y0 = y - i * (a / g);
        printf("%d*%d + %d*%d = %d\n", a, x0, b, y0, a * x0 + b * y0);
    }
}
```

6.11. is prime

```
bool is_prime(int64_t n) {
    if (n < 2) return 0;
    for (int64_t p = 2; p * p <= n; p += p % 2 + 1) if (n % p == 0)
        return 0;
    return 1;
}
```

6.12. Miller Rabin

```
bool probably_prime(ll n, ll a, ll d, int s){
    ll x = binpow(a, d, n);
    if(x == 1 || x+1 == n) return true;
    forn(r, s){
        x = mulmod(x,x,n);
        if(x == 1) return false;
        if(x+1 == n) return true;
    }
    return false;
}

bool miller_rabin(ll n){//check (n is prime)?
    if(n < 2) return false;
    const int a[] = {2,3,5,7,11,13,17,19,23};
    int s = -1;
    ll d = n-1;
    while(!d&1) d >>= 1, s++;

    forn(i, 9){
        if(n == a[i]) return true;
        if(!probably_prime(n, a[i], d, s))
            return false;
    }
    return true;
}
```

6.13. Sieve

```
const int MAX = int(1e6);
bitset<MAX + 5> bs;
vector<int64_t> prime;

void sieve() {
    bs.set();
    bs[0] = bs[1] = 0;
    for (int i = 2; i <= MAX; i++) {
        if (bs[i]) {
            prime.pb(i);
            for (int j = i * i; j <= MAX; j += i) {
                bs[j] = 0;
            }
        }
    }
}
```

6.14. Propiedades del modulo

```
constexpr int MOD = 1e9 + 7;

int64_t binpow(int64_t a, int64_t b, int64_t m) {
    a %= m;
    int64_t res = 1;
    while (b > 0) {
        if (b & 1) res = res * a % m;
        a = a * a % m;
        b >>= 1;
    }
    return res;
}
inline int inv_mod(int a) { return binpow(a, MOD - 2, MOD); }
inline int add(int a, int b) { return ((a % MOD) + (b % MOD)) % MOD; }
inline int res(int a, int b) { return ((a % MOD) - (b % MOD) + MOD) %
   MOD; }
inline int mul(int a, int b) { return ((a % MOD) * (b % MOD)) % MOD; }
inline int divm(int a, int b) { return  mul(a, inv_mod(b)); }
```

6.15. ExtendedEuclid

```
int extEuclid(int a, int b, int &x, int &y){
    if(b == 0){
        x = 1;
        y = 0;
        return a;
    }
    int xi, yi;
    int g = extEuclid(b, a%b, xi, yi);
    x = yi;
    y = xi-yi*(a/b);
    return g;
}
```

6.16. Modular Inverse

```
int inv[MAXN];

void modular_inverse_range(int m) {
    inv[0] = 0;  inv[1] = 1;
    forne(i, 2, MAXN)
```

```
        inv[i] = (-(m / i) * inv[m % i] + m) % m;
}

int modular_inverse_binpow(int a, int m) {
    return binpow(a, phi(m) - 1, m);
}

int modular_inverse_extEuclid(int a, int m) {
    int x, y;
    int g = extEuclid(a, m, x, y);
    if (g != 1)
        return -1;
    x = (x % m + m) % m;
    return x;
}

vector<int> inversos(vector<int> a, int m) {
    vector<int> inv;
    int v = 1;
    forn(i, sz(a)) {
        inv.pb(v);
        v = (v * a[i]) % m;
    }

    int x, y;
    extEuclid(v, m, x, y);
    x = (x % m + m) % m;
    for (int i = sz(a) - 1; i >= 0; i--) {
        inv[i] = inv[i] * x;
        x = (x * a[i]) % m;
    }
    return inv;
}
```

6.17. Phi

```
int phi(int n){
    int result = n;
    for(int i=2; i*i<=n; ++i){
        if(n%i) continue;
        while(n%i == 0)
            n /= i;
        result -= result/i;
    }

    if(n > 1)
        result -= result/n;
    return result;
}

vi phi_1_to_n(int n){
    vector<int> phi;
    forne(i, n) phi.pb(i);

    for(int i = 2; i <= n; ++i){
        if(phi[i] != i) continue;

        for(int j = i; j <= n; j += i)
            phi[j] -= phi[j]/i;
    }
    return phi;
}

vi phi_1_to_n2(int n){
    vector<int> phi;
    forne(i, n) phi.pb(i-1);
    phi[1] = 1;

    for(int i = 2; i <= n; ++i){
        for(int j = i*2; j <= n; j += i)
            phi[j] -= phi[i];
    }
    return phi;
}
```

6.18. Pow

```
int64_t binpow(int64_t a, int64_t b, int64_t m) {
    a %= m;
    int64_t res = 1;
    while (b > 0) {
        if (b & 1) res = res * a % m;
        a = a * a % m;
        b >>= 1;
    }
    return res;
}

int64_t binpow(int64_t a, int64_t b) {
    int64_t res = 1;
    while (b > 0) {
        if (b & 1) res = res * a;
```

```
        a = a * a;
        b >>= 1;
    }
    return res;
}
```

6.19. floordiv ceildiv

```
int64_t floor_div(int64_t a, int64_t b) {
    return a / b - ((a ^ b) < 0 && a % b != 0);
}

int64_t ceil_div(int64_t a, int64_t b) {
    return a / b + ((a ^ b) > 0 && a % b != 0);
}
```

6.20. sqrt

```
ll int_sqrt (ll x) {
    ll ans = 0;
    for (ll k = 1LL << 30; k != 0; k /= 2)
        if ((ans + k) * (ans + k) <= x)
            ans += k;
    return ans;
}
```

6.21. dec and bin

```
string dec_to_bin(int64_t n) {
    bitset<32> bs(n);
    string s = bs.to_string();
    return s;
}
int64_t bin_to_dec(string s) {
    bitset<32> bs(s);
    int64_t n = bs.to_ullong();
    return n;
}

string dec_to_bin(int64_t n) {
    string s;
    while (n) {
        if (n & 1) s.pb('1');
        else s.pb('0');
        n >>= 1;
    }
    reverse(all(s));
    return s;
}
int64_t bin_to_dec(string s) {
    int64_t res = 0;
    for (auto&& i : s) {
        res <<= 1;
        res += i - '0';
    }
    return res;
}
```

6.22. Berlekamp massey

```
struct ber_ma{
    vi BM(vi &x){
        vi ls,cur; int lf,ld;
        forn(i,sz(x)){
            ll t=0;
            forn(j,sz(cur)) t=(t+x[i-j-1]*(ll)cur[j])%mod;
            if((t-x[i])%mod==0) continue;
            if(!sz(cur)){
                cur.resize(i+1);
                lf=i; ld=(t-x[i])%mod;
                continue;
            }
            ll k=-(x[i]-t)*inv(ld,mod);
            vi c(i-lf-1); c.pb(k);
            forn(j,sz(ls)) c.pb(-ls[j]*k%mod);
            if(sz(c)<sz(cur)) c.resize(sz(cur));
            forn(j,sz(cur)) c[j]=(c[j]+cur[j])%mod;
            if(i-lf+sz(ls)>=sz(cur)) ls=cur,lf=i,ld=(t-x[i])%mod;
            cur=c;
        }
        forn(i,sz(cur)) cur[i]=(cur[i]%mod+mod)%mod;
        return cur;
    }

    int m; //length of recurrence
    //a: first terms
    //h: relation
    vector<ll> a, h, t_, s, t;
```

```
    //calculate p*q mod f
    inline vector<ll> mull(vector<ll> p, vector<ll> q){
        forn(i,2*m) t_[i]=0;
        forn(i,m) if(p[i])
            forn(j,m)
                t_[i+j]=(t_[i+j]+p[i]*q[j])%mod;
        for(int i=2*m-1;i>=m;--i) if(t_[i])
            forn(j,m)
                t_[i-j-1]=(t_[i-j-1]+t_[i]*h[j])%mod;
        forn(i,m) p[i]=t_[i];
        return p;
    }
    inline ll calc(ll k){
    if(k < sz(a)) return a[k];
        forn(i,m) s[i]=t[i]=0;
        s[0]=1;
        if(m!=1) t[1]=1;
        else t[0]=h[0];

        while(k){
            if(k&1LL) s = mull(s,t);
            t = mull(t,t); k/=2;
        }
        ll su=0;
        forn(i,m) su=(su+s[i]*a[i])%mod;
        return (su%mod+mod)%mod;
    }
    ber_ma(vi &x){
        vi v = BM(x); m=sz(v);
        h.resize(m), a.resize(m), s.resize(m);
        t.resize(m), t_.resize(2*m);
        forn(i,m) h[i]=v[i],a[i]=x[i];
    }
};
```

6.23. fraction

```
struct fraction {
    int num, den;

    fraction(int num, int den) :num(num), den(den) {
        check_den();
        simplify();
    }
    void check_den() {
        if (den < 0) {
            num = -num;
            den = -den;
        }
    }
    void simplify() {
        int mcd = __gcd(abs(num), abs(den));
        num /= mcd;
        den /= mcd;
    }
    pair<int, int> x() { return { num,den }; }

    fraction operator + (const fraction& x) const {
        return fraction(num * x.den + den * x.num, den * x.den);
    }
    fraction operator - (const fraction& x) const {
        return fraction(num * x.den - den * x.num, den * x.den);
    }
    fraction operator * (const fraction& x) const {
        return fraction(num * x.num, den * x.den);
    }
    fraction operator / (const fraction& x) const {
        return fraction(num * x.den, den * x.num);
    }
    friend ostream& operator << (ostream& os, const fraction& x) {
        return os << x.num << " / " << x.den;
    }
};
```

6.24. Matrix

```
struct matrix {
  int n, m;
  vector<vector<int>> v;

  matrix(int n, int m, bool ones = false) : n(n), m(m), v(n,
      vector<int>(m)) {
    if (ones) forn(i, n) v[i][i] = 1;
  }

  matrix operator * (const matrix& o) {
    matrix ans(n, o.m);
    forn(i, n)
      forn(k, m) if (v[i][k])
      forn(j, o.m)
      ans[i][j] = (1ll * v[i][k] * o.v[k][j] + ans[i][j]) % MOD;
    return ans;
```

```
  }

  vector<int>& operator [] (int i) {
    return v[i];
  }
};

matrix binpow(matrix b, ll e) {
  matrix ans(b.n, b.m, true);
  while (e) {
    if (e & 1) ans = ans * b;
    b = b * b;
    e >>= 1;
  }
  return ans;
}
```

6.25. nCK

```
struct mint {
    static constexpr int  m = 1e9 + 7;
    //static inline int  m = 998244353; //to change mod
    int x;
    mint() : x(0) {}
    mint(long long x_) :x(x_% m) { if (x < 0) x += m; }
    int val() { return x; }
    mint& operator+=(mint b) { if ((x += b.x) >= m) x -= m; return
        *this; }
    mint& operator-=(mint b) { if ((x -= b.x) < 0) x += m; return
        *this; }
    mint& operator*=(mint b) { x = (long long)(x)*b.x % m; return
        *this; }
    mint pow(long long e) const {
        mint r = 1, b = *this;
        while (e) {
            if (e & 1) r *= b;
            b *= b;
            e >>= 1;
        }
        return r;
    }
    mint inv() { return pow(m - 2); }
    mint& operator/=(mint b) { return *this *= b.pow(m - 2); }
    friend mint operator+(mint a, mint b) { return a += b; }
    friend mint operator-(mint a, mint b) { return a -= b; }
    friend mint operator/(mint a, mint b) { return a /= b; }
    friend mint operator*(mint a, mint b) { return a *= b; }
    friend bool operator<(mint a, mint b) { return a.x < b.x; }
    friend bool operator==(mint a, mint b) { return a.x == b.x; }
    friend bool operator!=(mint a, mint b) { return a.x != b.x; }
};


const int mxN = 1e6 + 7; // max value of  N,K
mint fact[mxN];
mint inv_fact[mxN];
void init() {
    fact[0] = 1;
    forne(i, 1, mxN) fact[i] = fact[i - 1] * i;
    forn(i, mxN) inv_fact[i] = fact[i].inv();
}
// https://cp-algorithms.com/combinatorics/binomial-coefficients.html

mint nCk(ll n, ll k) {
    return (fact[n] * inv_fact[k]) * inv_fact[n - k];
}
```

6.26. Binomial

```
template<typename T>
struct Binomial {
    vector<T> Facto, inv_Facto;
    void extend(int m = -1) {
        int n = sz(Facto);
        if (m == -1) m = n * 2;
        if (n >= m) return;
        Facto.resize(m);
        inv_Facto.resize(m);
        for (int i = n; i < m; i++) Facto[i] = Facto[i - 1] * T(i);
        inv_Facto[m - 1] = T(1) / Facto[m - 1];
        for (int i = m - 1; i > n; i--) inv_Facto[i - 1] =
            inv_Facto[i] * T(i);
    }
    Binomial(int MAX = 0) {
        Facto.resize(1, T(1));
        inv_Facto.resize(1, T(1));
        extend(MAX + 1);
    }

    T fact(int i) {
        if (i < 0) return 0;
        while (int(sz(Facto)) <= i) extend();
```

```
        return Facto[i];
    }
    T invfact(int i) {
        if (i < 0) return 0;
        while (int(sz(inv_Facto)) <= i) extend();
        return inv_Facto[i];
    }
    T C(int a, int b) {
        if (a < b || b < 0) return 0;
        return fact(a) * invfact(b) * invfact(a - b);
    }
    T invC(int a, int b) {
        if (a < b || b < 0) return 0;
        return fact(b) * fact(a - b) * invfact(a);
    }
    T P(int a, int b) {
        if (a < b || b < 0) return 0;
        return fact(a) * invfact(a - b);
    }
    T inv(int a) {
        if (a < 0) return inv(-a) * T(-1);
        if (a == 0) return 1;
        return fact(a - 1) * invfact(a);
    }
    T Catalan(int n) {
        if (n < 0) return 0;
        return fact(2 * n) * invfact(n + 1) * invfact(n);
    }
    T narayana(int n, int k) {
        if (n <= 0 || n < k || k < 1) return 0;
        return C(n, k) * C(n, k - 1) * inv(n);
    }
    T Catalan_pow(int n, int d) {
        if (n < 0 || d < 0) return 0;
        if (d == 0) {
            if (n == 0) return 1;
            return 0;
        }
        return T(d) * inv(d + n) * C(2 * n + d - 1, n);
    }
    // return [x^a] 1/(1-x)^b
    T ruiseki(int a, int b) {
        if (a < 0 || b < 0) return 0;
        if (a == 0) {
            return 1;
        }
        return C(a + b - 1, b - 1);
    }
    // (a, b) -> (c, d)
    // always x + e >= y
    T mirror(int a, int b, int c, int d, int e = 0) {
        if (a + e < b || c + e < d) return 0;
        if (a > c || b > d) return 0;
        a += e;
        c += e;
        return C(c + d - a - b, c - a) - C(c + d - a - b, c - b + 1);
    }
    // return sum_{i = 0, ... , a} sum_{j = 0, ... , b} C(i + j, i)
    // return C(a + b + 2, a + 1) - 1;
    T gird_sum(int a, int b) {
        if (a < 0 || b < 0) return 0;
        return C(a + b + 2, a + 1) - 1;
    }
    // return sum_{i = a, ..., b - 1} sum_{j = c, ... , d - 1} C(i +
        j, i)
    // AGC 018 E
    T gird_sum_2(int a, int b, int c, int d) {
        if (a >= b || c >= d) return 0;
        a--, b--, c--, d--;
        return gird_sum(a, c) - gird_sum(a, d) - gird_sum(b, c) +
            gird_sum(b, d);
    }

    // the number of diagonal dissections of a convex n-gon into k+1
        regions.
    // OEIS A033282
    // AGC065D
    T diagonal(int n, int k) {
        if (n <= 2 || n - 3 < k || k < 0) return 0;
        return C(n - 3, k) * C(n + k - 1, k) * inv(k + 1);
    }
};

Binomial<mint> bin;
```

6.27. FFT

```
typedef complex<double> base;
const double PI = acos(-1);

struct FFT {
    vector<int> rev;
    FFT(){ }
```

```
void calc_rev(int n, int log_n){
    forn(i, n) {
        rev.pb(0);
        forn(j, log_n)
            if(i & (1<<j))
                rev[i] |= 1<<(log_n-1-j);
    }
}

void computeFFT(vector<base> &a, bool invert) {
    int n = (int) a.size();

    forn(i, n)
        if (i < rev[i])
            swap (a[i], a[rev[i]]);

    for(int len=2; len<=n; len<<=1) {
        double ang = 2*PI/len * (invert ? -1 : 1);
        base wlen (cos(ang), sin(ang));
        for (int i=0; i<n; i+=len) {
            base w(1);
            for (int j=0; j<len/2; ++j) {
                base u = a[i+j],  v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }
    if(invert)
        forn(i, n)
            a[i] /= n;
}

vector<int> multiply(vector<int> &a, vector<int> &b) {
    int n;  for(n = 1; n < sz(a) + sz(b); n <<= 1);
    calc_rev(n, round(log2(n)));

    vector<base> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    fa.resize(n);  fb.resize(n);

    computeFFT(fa, false),  computeFFT(fb, false);
    forn(i, n) fa[i] *= fb[i];
    computeFFT(fa, true);

    vector<int> res(n);
    forn(i, n) res[i] = int(fa[i].real() + 0.5);

    //Carries for integer multiplication
    /*int carry = 0;
    forn(i, n) {
        res[i] += carry;
        carry = res[i] / 10;
        res[i] %= 10;
    }*/
    return res;
}
};
```

6.28. FFT ll

```
typedef long double ld;
const ld PI = acos((ld)-1);

namespace FFT {
    struct com {
        ld x, y;

        com(ld _x = 0, ld _y = 0) : x(_x), y(_y) {}

        inline com operator + (const com& c) const {
            return com(x + c.x, y + c.y);
        }
        inline com operator - (const com& c) const {
            return com(x - c.x, y - c.y);
        }
        inline com operator * (const com& c) const {
            return com(x * c.x - y * c.y, x * c.y + y * c.x);
        }
        inline com conj() const {
            return com(x, -y);
        }
    };

    const static int maxk = 19, maxn = (1 << maxk) + 1;
    com ws[maxn];
    int dp[maxn];
    com rs[maxn];
    int n, k;
    int lastk = -1;

    void fft(com* a, bool torev = 0) {
```

```
    if (lastk != k) {
        lastk = k;
        dp[0] = 0;

        for (int i = 1, g = -1; i < n; ++i) {
            if (!(i & (i - 1))) {
                ++g;
            }
            dp[i] = dp[i ^ (1 << g)] ^ (1 << (k - 1 - g));
        }

        ws[1] = com(1, 0);
        forn(two, k - 1) {
            ld alf = PI / n * (1 << (k - 1 - two));
            com cur = com(cos(alf), sin(alf));

            int p2 = (1 << two), p3 = p2 * 2;
            forab(j, p2, p3) {
                ws[j * 2 + 1] = (ws[j * 2] = ws[j]) * cur;
            }
        }
    }
    forn(i, n) {
        if (i < dp[i]) {
            swap(a[i], a[dp[i]]);
        }
    }
    if (torev) {
        forn(i, n) {
            a[i].y = -a[i].y;
        }
    }
    for (int len = 1; len < n; len <<= 1) {
        for (int i = 0; i < n; i += len) {
            int wit = len;
            for (int it = 0, j = i + len; it < len; ++it, ++i,
                ++j) {
                com tmp = a[j] * ws[wit++];
                a[j] = a[i] - tmp;
                a[i] = a[i] + tmp;
            }
        }
    }
}

com a[maxn];
vector<ll> multiply(vector<ll>& _a, vector<ll>& _b) {
    int na = sz(_a), nb = sz(_b);

    for (k = 0, n = 1; n < na + nb - 1; n <<= 1, ++k);
    forn(i, n) {
        a[i] = com(i < na ? _a[i] : 0, i < nb ? _b[i] : 0);
    }
    fft(a);
    a[n] = a[0];
    forn(i, (n - i) + 1) {
        a[i] = (a[i] * a[i] - (a[n - i] * a[n - i]).conj()) *
            com(0, (ld)-1 / n / 4);
        a[n - i] = a[i].conj();
    }
    fft(a, 1);
    int res = 0;

    vector<ll> ans(n);
    forn(i, n) {
        ll val = (ll)round(a[i].x);
        ans[i] = val;//only for multiply poly

        /*if (val) {//only for multiply long integers
            while (res < i) {
                ans[res++] = 0;
            }
            ans[res++] = val;
        }*/
    }
    return ans;
}
};
```

6.29. Segmented Sieve

```
vector<int> prime; // sqrt(MAX R)

vector<ll> segmented_criba(ll l, ll r) {
    l = max(l, 2ll);
    vector<bool> vis(r - l + 1);
    for (int& pp : prime) {
        if ((ll)pp * pp > r) break;
        ll mn = (l + pp - 1) / pp;
        if (mn == 1ll) mn++;
        mn *= pp;
        for (ll i = mn; i <= r; i += pp) {
            vis[i - l] = true;
```

```
        }
    }
    vector<ll> ans;
    forn(i, sz(vis)) if (!vis[i]) ans.pb(l + i);
    return ans;
}
```

6.30. divisores

```
vector<int>div(int n) {
    vector<int> ans;
    for (int i = 1; i * i <= n;i++) {
        if (n % i == 0) {
            ans.pb(i);
            if (i != n / i) {
                ans.pb(n / i);
            }
        }
    }
    return ans;
}
```

7. Problems

7.1. Eval fractio

```
# 1/2+1/3 -> 5/6
import math
import sys
from fractions import Fraction
import re
input = sys.stdin.readline
write = sys.stdout.write
def print(x): write(str(x) + '\n')

def main():
    s = input().strip()
    while s != "":
        s_eval = re.sub(r'(\d+)/(\d+)', r'Fraction(\1,\2)', s)
        ans = eval(s_eval, {'Fraction': Fraction})
        x = ans
        y = f"{x.numerator}/{x.denominator}"
        print(y)
        s = input().strip()

if __name__ == '__main__':
    main()
```

7.2. dp+nck

```
#include <bits/stdc++.h>
using namespace std;
#define endl      '\n'
#define f         first
#define s         second
#define ins       insert
#define pb        push_back
#define eb        emplace_back
#define sz(x)     int((x).size())
#define all(x)    begin(x), end(x)
typedef long long ll;
typedef unsigned long long ull;
#define forn(i, n) for (int i = 0; i < n; ++i)
#define each(i, x) for (auto &&i : x)
#define forne(i,x,n) for (int i = x; i < n; ++i)
#define show(x) for (auto &&i : x) {cerr << i <<' ';} cerr<< endl;


#define LOCAL
void dbg_out() { cerr << ']' << endl; }
template<typename Head, typename... Tail>
void dbg_out(Head H, Tail... T) { cerr << H;if (sizeof...(T)) cerr <<
    ',' << ' '; dbg_out(T...); }
#ifdef LOCAL
#define dbg(...) cerr << '|' << __LINE__ << '|'<< '{' << #__VA_ARGS__
    << '}'<<':'<<' '<<'[', dbg_out(__VA_ARGS__)
#else
#define dbg(...)
#endif


//
    https://github.com/JaiderBR/CompetitiveProgramming/blob/main/Data%20St
static constexpr int mod = 1e9 + 7;
struct mint {
    static constexpr int  m = 1e9 + 7;
    int x;
    mint() : x(0) {}
```

```
    mint(long long x_) :x(x_ % m) { if (x < 0) x += m; }
    int val() { return x; }
    mint& operator+=(mint b) { if ((x += b.x) >= m) x -= m; return
        *this; }
    mint& operator-=(mint b) { if ((x -= b.x) < 0) x += m; return
        *this; }
    mint& operator*=(mint b) { x = (long long)(x)*b.x % m; return
        *this; }
    mint pow(long long e) const {
        mint r = 1, b = *this;
        while (e) {
            if (e & 1) r *= b;
            b *= b;
            e >>= 1;
        }
        return r;
    }
    mint inv() { return pow(m - 2); }
    mint& operator/=(mint b) { return *this *= b.pow(m - 2); }
    friend mint operator+(mint a, mint b) { return a += b; }
    friend mint operator-(mint a, mint b) { return a -= b; }
    friend mint operator/(mint a, mint b) { return a /= b; }
    friend mint operator*(mint a, mint b) { return a *= b; }
    friend bool operator==(mint a, mint b) { return a.x == b.x; }
    friend bool operator!=(mint a, mint b) { return a.x != b.x; }
};

const int mxN = 105;
mint dp[21][mxN][mxN][mxN];
mint factorial[mxN];
mint inverse_factorial[mxN];

void init() {
    factorial[0] = 1;
    forne(i, 1, mxN) factorial[i] = factorial[i - 1] * i;
    forn(i, mxN) inverse_factorial[i] = factorial[i].inv();
}
// https://cp-algorithms.com/combinatorics/binomial-coefficients.html
mint binomial_coefficient(ll n, ll k) {
    return (factorial[n] * inverse_factorial[k]) * inverse_factorial[n
        - k];
}

mint back(ll n, ll r, ll g, ll b) {
    if (r < 0 || g < 0 || b < 0) return 0;
    if (n == 0) return 1;
    if (dp[n][r][g][b] != -1) return dp[n][r][g][b];
    mint form_1 = 0, form_2 = 0, form_3 = 0;
    form_1 = back(n - 1, r - n, g, b) + back(n - 1, r, g - n, b) +
        back(n - 1, r, g, b - n);
    if (n % 2 == 0) {
        /*
        R G B (R G) - (R B) - (G B)
        */
        form_2 = binomial_coefficient(n, n / 2) * ((back(n - 1, r - n
            / 2, g - n / 2, b) + back(n - 1, r - n / 2, g, b - n / 2)
            + back(n - 1, r, g - n / 2, b - n / 2)));
    }

    if (n % 3 == 0) {
        /*
        n=3 bc_1=3 bc_2=2 = 6
        R G B - R B G - G B R -B G R - G R B - B R G
        */
        mint bc_1 = binomial_coefficient(n, n / 3);
        mint bc_2 = binomial_coefficient(2 * n / 3, n / 3);
        form_3 = (bc_1 * bc_2) * back(n - 1, r - n / 3, g - n / 3, b -
            n / 3);
    }
    return dp[n][r][g][b] = form_1 + form_2 + form_3;
}


int main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    init();
    ll n, r, g, b; cin >> n >> r >> g >> b;
    forn(i, n + 1) forn(j, r + 1) forn(k, g + 1) forn(l, b + 1)
        dp[i][j][k][l] = -1;
    // memset(dp, -1, sizeof dp);
    cout << back(n, r, g, b).val() << endl;

    cout << flush;
    return 0;
}
```

7.3. Conotruncado

```
#include <bits/stdc++.h>
using namespace std;
#define endl      '\n'
#define f         first
#define s         second
```

```cpp
#define ins       insert
#define pb        push_back
#define eb        emplace_back
#define sz(x)     int((x).size())
#define all(x)    begin(x), end(x)
typedef long long ll;
typedef unsigned long long ull;
#define forn(i, n) for (int i = 0; i < n; ++i)
#define each(i, x) for (auto &&i : x)
#define forne(i,x,n) for (int i = x; i < n; ++i)
#define show(x) for (auto &&i : x) {cerr << i <<' ';} cerr<< endl;


void dbg_out() { cerr << ']' << endl; }
template<typename Head, typename... Tail>
void dbg_out(Head H, Tail... T) { cerr << H;if (sizeof...(T)) cerr <<
    ',' << ' '; dbg_out(T...); }
#ifdef LOCAL
#define dbg(...) cerr << '|' << __LINE__ << '|'<< '{' << #__VA_ARGS__
    << '}'<<':'<<' '<<'[', dbg_out(__VA_ARGS__)
#else
#define dbg(...)
#endif
#define int int64_t

const double PI = acos(-1);
const double eps = 1e-7;

// se requiere llenar 50% del volumen de un cono truncado
// con radio menor r, radio mayor R y altura h
// se requiere encontrar la altura a la que se debe llenar el cono
    truncado

void solve() {
    double r, R, h; cin >> r >> R >> h;

    auto get = [&](double r1, double r2, double h) {
        return (PI * h / 3.0) * (r1 * r1 + r1 * r2 + r2 * r2);
            //volumen cono truncado
        };

    double vol = get(r, R, h);
    double l = 0, hi = h;

    forn(_, 200) {
        double mid = (l + hi) / 2.0;
        double newmid = r + (R - r) * (mid / h); // Radio en la altura
            mid
        double volmid = get(r, newmid, mid);
        if (volmid < vol / 2.0) l = mid;
        else hi = mid;
    }

    cout << fixed << setprecision(9) << (l + hi) / 2.0 << endl;


}
main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);


    int testcase; cin >> testcase; while (testcase--) solve();









    cout << flush;
    return 0;
}
```

7.4. censor

```cpp
/*
Dado un string s y n strings t, se pide eliminar todas las ocurrencias
    de los strings t en s.
https://usaco.guide/adv/string-search/#problem-usaco-533
*/

#include <bits/stdc++.h>
using namespace std;
#define endl      '\n'
#define f         first
#define s         second
#define ins       insert
#define pb        push_back
#define eb        emplace_back
#define sz(x)     int((x).size())
#define all(x)    begin(x), end(x)
typedef long long ll;
```

```cpp
typedef unsigned long long ull;
#define forn(i, n) for (int i = 0; i < n; ++i)
#define each(i, x) for (auto &&i : x)
#define forne(i,x,n) for (int i = x; i < n; ++i)
#define show(x) for (auto &&i : x) {cout << i;} cerr<< endl;


void dbg_out() { cerr << ']' << endl; }
template<typename Head, typename... Tail>
void dbg_out(Head H, Tail... T) { cerr << H;if (sizeof...(T)) cerr <<
    ',' << ' '; dbg_out(T...); }
#ifdef LOCAL
#define dbg(...) cerr << '|' << __LINE__ << '|'<< '{' << #__VA_ARGS__
    << '}'<<':'<<' '<<'[', dbg_out(__VA_ARGS__)
#else
#define dbg(...)
#endif

/*
Usage:
        Good values c = 137, modbest=998244353, mod = 10^9 + 7, mod =
            1e18 + 9.
        If necessary to check too many pairs of hashes, use two
        different hashes.
        If hashing something other than english characters:
            - Don't have elements with value 0.
            - Use c > max element value.
*/

#define int int64_t
constexpr int mxN = 1e6 + 7;
vector<int>p(mxN);
void pre(int c, int mod) {
    p[0] = 1;
    for (int i = 0; i < mxN - 1; i++) {
        p[i + 1] = (c * p[i]) % mod;
    }
}

struct Hash {
    #warning llamar pre;
    ll c, mod;
    vector<int> h;
    Hash(const string s, const int c, const int mod) : c(c), mod(mod),
        h(sz(s) + 1) {
        h[0] = 0;
        for (int i = 0; i < sz(s); i++) {
            h[i + 1] = (c * h[i] + s[i]) % mod;
        }
    }
    // Returns hash of interval s[a ... b] (where 0 <= a <= b < sz(s))
    ll get(int a, int b) {
        return (h[b + 1] - ((h[a] * p[b - a + 1]) % mod) + mod) % mod;
    }
};

bool same(Hash& Ha, Hash& Hb, int l, int r) {
    int qa = Ha.get(l, r);
    int qb = Hb.get(sz(Hb.h) - 2 - r, sz(Hb.h) - 2 - l);
    return qa == qb;
}

const int mod = (1e9 + 7);
main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);

    pre(137, mod);
#ifndef LOCAL
    freopen("censor.in", "r", stdin);
    freopen("censor.out", "w", stdout);
#endif

    string s, t; cin >> s;

    int mm = (1ULL << 63) - 1, n;
    int mx = -((1ULL << 63) - 1);
    cin >> n;
    set<int>mp, tam;

    forn(i, n) {
        cin >> t;
        mm = min(mm, sz(t));
        mx = max(mx, sz(t));
        tam.ins(sz(t));
        Hash ht(t, 137, mod);
        mp.ins(ht.get(0, sz(t) - 1));
    }

    int c = 137;
    vector<int> h(sz(s) + 10);
    h[0] = 0;

    auto get = [&](int a, int b) -> int {
        return (h[b + 1] - ((h[a] * p[b - a + 1]) % mod) + mod) % mod;
```

```
        };
    int i = 0;
    vector<char>ans(sz(s) + 1);
    each(j, s) {
        ans[i] = j;
        h[i + 1] = (c * h[i] + j) % mod;
        if (i >= mm - 1) {
            each(l, tam) {
                if (i >= l - 1) {
                    int cur = get(i - l + 1, i);
                    if (i >= l - 1 && mp.count(cur)) {
                        i = i - l;
                    }
                }
                else break;
            }
        }
        i++;
    }

    forn(j, i) cout << ans[j];

    cout << flush;
    return 0;
}
```

7.5. F Less Than G

```
/*
Given two arrays a and b of n non-negative integers, count the number
    of good pairs
l,r (1<=l<=r<=n), satisfying F(l,r)<G(l,r)
Where F(l,r) is the sum of the square of numbers in the range [l,r]
And G(l,r) is the square of the bitwise OR of the range [l,r]
*/

main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);

    // brute();
    int n; cin >> n;
    vector<int> a(n), b(n), prefix(n + 6);
    set<int> pro[25];
    forn(i, n) cin >> a[i];
    forn(i, n) cin >> b[i];
    forn(init, 22) pro[init].insert(n);
    forn(i, n) prefix[i + 1] = prefix[i] + a[i] * a[i];
    prefix.erase(prefix.begin());
    forn(i, n) forn(j, 22) if (b[i] & (1 << j)) pro[j].insert(i);

    int ans = 0;
    forn(i, n) {
        int last = i, Or = b[i];
        while (last < n) {
            int best = n;
            forn(j, 22) {
                if (!(Or & (1 << j))) best = min(best,
                    *pro[j].upper_bound(last));
            }
            int l = last, r = best - 1, x = -1, need = Or * Or;
            while (l <= r) {
                int mid = (l + r) / 2;
                if (prefix[mid] - (i > 0 ? prefix[i - 1] : 0) >= need)
                    r = mid - 1;
                else l = mid + 1, x = mid;
            }
            if (~x) ans += x - last + 1;
            swap(last, best);
            if (last < n) Or |= b[last];
        }
    }
    cout << ans << endl;


    cout << flush;
    return 0;
}
```

7.6. graycode

```
/*
Genera una permutacion de 0 a 2^n-1, de modo que
dos posiciones adyacentes difieren en exactamente 1 bit
*/

vector<string> gray_code(int n) {
    vector<string> ret(1 << n);
    for (int i = 0; i < (1 << n); i++) {
        ret[i] = bitset<32>(i ^ (i >> 1)).to_string();
    }
    return ret;
}
```

7.7. Nested Circles

```
/*
You are given n circles numbered from 1 to n.
Each circle is defined by an integer center (xi,yi) and an integer
    radius ri.

Then we will ask you q questions. In each question,
we will give you an integer point (xi,yi),
and you have to find the number of circles that cover this point.
*/

void solve() {

    map<pair<int, int>, vector<array<int, 3>>> mp;
    map<pair<int, int>, int> mpans;

    int n, q, x, y, r; cin >> n >> q;

    forn(i, n) {
        cin >> x >> y >> r;
        mp[{int(x / 10), int(y / 10)}].pb({ x,y,r });
    }

    while (q--) {
        cin >> x >> y;
        int gX = int(x / 10), gY = int(y / 10);
        int ans = 0;
        if (!mpans.count({ x,y })) {
            forne(dx, -1, 2) {
                forne(dy, -1, 2) {
                    auto it = mp.find({ gX + dx,  gY + dy });
                    if (it != mp.end()) {
                        each(i, it->s) {
                            int xc = i[0], yc = i[1], rc = i[2];
                            if ((x - xc) * (x - xc) + (y - yc) * (y -
                                yc) <= rc * rc) ans++;
                        }
                    }
                }
            }
            cout << ans << endl;
            mpans[{x, y}] = ans;
        }
        else cout << mpans[{x, y}] << endl;

    }
}
```

7.8. Restoring the Expression

```
/*
12345168 = 123+45=168
199100 = 1+99=100
*/

#include <bits/stdc++.h>
using namespace std;
#define endl      '\n'
#define f         first
#define s         second
#define ins       insert
#define pb        push_back
#define eb        emplace_back
#define sz(x)     int((x).size())
#define all(x)    begin(x), end(x)
typedef long long ll;
typedef unsigned long long ull;
#define forn(i, n) for (int i = 0; i < n; ++i)
#define each(i, x) for (auto &&i : x)
#define forne(i,x,n) for (int i = x; i < n; ++i)
#define show(x) for (auto &&i : x) {cerr << i <<' ';} cerr<< endl;


void dbg_out() { cerr << ']' << endl; }
template<typename Head, typename... Tail>
void dbg_out(Head H, Tail... T) { cerr << H;if (sizeof...(T)) cerr <<
    ',' << ' '; dbg_out(T...); }
#ifdef LOCAL
#define dbg(...) cerr << '|' << __LINE__ << '|'<< '{' << #__VA_ARGS__
    << '}'<<':'<<' '<<'[', dbg_out(__VA_ARGS__)
#else
#define dbg(...)
#endif
#define int int64_t

constexpr int mxN = 1e6 + 7, mod = 998244353;
vector<int>p(mxN);
void pre(int c, int mod) {
    p[0] = 1;
    for (int i = 0; i < mxN - 1; i++) {
        p[i + 1] = (c * p[i]) % mod;
    }
```

```
}

struct Hash {
    #warning llamar pre;
    ll c, mod;
    vector<int> h;
    Hash(const string& s, const int c, const int mod) : c(c),
        mod(mod), h(sz(s) + 1) {
        h[0] = 0;
        for (int i = 0; i < sz(s); i++) {
            h[i + 1] = (c * h[i] + s[i] - '0') % mod;
        }
    }
    // Returns hash of interval s[a ... b] (where 0 <= a <= b < sz(s))
    ll get(int a, int b) {
        return (h[b + 1] - ((h[a] * p[b - a + 1]) % mod) + mod) % mod;
    }
};

bool same(Hash& Ha, Hash& Hb, int l, int r) {
    int qa = Ha.get(l, r);
    int qb = Hb.get(sz(Hb.h) - 2 - r, sz(Hb.h) - 2 - l);
    return qa == qb;
}

main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);

    pre(10, mod);

    string s; cin >> s;
    Hash ha(s, 10, mod);
    int n = sz(s);

    auto ok = [&](int i, int j) {
        i--, j--;
        if (i - 1 < 0) return;
        int a = ha.get(0, i - 1), b = ha.get(i, j - 1), c = ha.get(j,
            n - 1);
        if (((a + b) % mod) == c && ((s[i] != '0' ? 1 : i == j - 1))
            && ((s[j] != '0' ? 1 : j == n - 1))) {
            cout << string(begin(s), begin(s) + i) << "+" <<
                string(begin(s) + i, begin(s) + j) << "=" <<
                string(begin(s) + j, end(s)) << endl;
            exit(0);
        }
        };

    forne(i, n / 3, ((n / 2) + 1)) {
        ok(i, n - i + 1);
        ok(i + 1, n - i + 1);
        ok(n - i * 2 + 2, n - i + 1);
        ok(n - i * 2 + 1, n - i + 1);
    }




    cout << flush;
    return 0;
}
```

7.9. non overlapping substrings

```
/*
add:
    https://github.com/Jaiderbr/CompetitiveProgramming/blob/main/String/ah
You want to select as many non-overlapping substrings of S
as possible such that each selected substring is exactly equal to one
    of the M strings.

3
7 2
arwhwar
ar
w
7 1
arwhwar
ar
4 1
rrrr
rr

------
4
2
2


*/
```

```
void solve(){

    int n, m; cin >> n >> m;
    string s; cin >> s;
    vector<string> st(m);
    forn(i, m) cin >> st[i];

    aho_corasick aho(st);
    vector<pair<int, int>> intervals;
    int cur = 0;

    forn(i, n) {
        cur = aho.get_suffix_link(cur, s[i]);
        int dict_node = aho.nodes[cur].word_index < 0 ?
            aho.nodes[cur].dict : cur;

        while (dict_node >= 0) {
            int widx = aho.nodes[dict_node].word_index;
            int len = st[widx].size();
            intervals.pb({i - len + 1, i});
            dict_node = aho.nodes[dict_node].dict;
        }
    }

    sort(all(intervals), [](auto &a, auto &b) {
        if (a.s == b.s) return a.f < b.f;
        return a.s < b.s;
    });

    int ans = 0, last_end = -1;
    each(it, intervals) {
        if (it.f > last_end) {
            ans++;
            last_end = it.s;
        }
    }

    cout << ans << endl;
}
```

7.10. Maximum Product

```
// Find the number from the range [a,b] which has the maximum product
    of the digits.
pair<int, string> dp[20][2][2][2];
bool vis[20][2][2][2];


pair<int, string> back(string l, string r, int pos, int ta, int tb,
    int st) {
    if (pos == sz(l)) return { 1, "" };
    if (vis[pos][ta][tb][st]) return dp[pos][ta][tb][st];
    int sta = ta ? l[pos] - '0' : 0, end = tb ? r[pos] - '0' : 9, ans
        = -1;
    string s = "";
    forne(i, sta, end + 1) {
        int val = i;
        if (st == 0 && i == 0) val = 1;
        pair<int, string> alt = back(l, r, pos + 1, ta & (i == sta),
            tb & (i == end), st | i > 0);
        if (alt.f * val > ans) {
            ans = alt.f * val;
            if (i == 0 && st == 0) s = alt.s;
            else s = alt.s, s.pb('0' + i);
        }
    }

    vis[pos][ta][tb][st] = 1;
    return dp[pos][ta][tb][st] = { ans, s };
}

pair<int, string> solve(int a, int b) {
    string L = to_string(a), R = to_string(b);
    if (sz(L) < sz(R)) {
        reverse(all(L));
        L += string(sz(R) - sz(L), '0');
        reverse(all(L));
    }
    memset(vis, 0, sizeof(vis));
    pair<int, string> ans = back(L, R, 0, 1, 1, 0);
    reverse(all(ans.s));
    return { ans.f, ans.s };
}
```

7.11. circles touching radius

```
// Problema: hallar el radio maximo de los circulos externos que rodean
// un circulo interno de radio R, tocandolo y tocandose entre ellos

ld n, R; cin >> n >> R; // n = numero de circulos externos, R = radio
    circulo interno
```

```
ld l = 0, r = 1e9, eps = 1e-8; // inicializa binaria: l minimo, r
    grande, eps precision

while (r - l > eps) { // mientras el rango sea mayor que la precision

    ld mid = l + (r - l) / 2; // mid = radio candidato circulos
        externos

    ld sinv = sin(pi / n); // sin(pi/n) = factor trigonometrico para
        distancia entre centros

    ld d = 2 * (R + mid) * sinv; // d = distancia entre centros de dos
        circulos externos

    if (d >= 2 * mid) l = mid; // si se tocan o sobra espacio, mid
        valido, busca mayor
    else r = mid; // si no se tocan, radio muy grande, busca menor
}

cout << fixed << setprecision(7) << l << endl;
```

7.12. matrixexp

```
//https://codeforces.com/gym/104758/problem/B

#include <bits/stdc++.h>
using namespace std;
#define endl      '\n'
#define f         first
#define s         second
#define ins       insert
#define pb        push_back
#define eb        emplace_back
#define sz(x)     int((x).size())
#define all(x)    begin(x), end(x)
typedef long long ll;
#define int ll
typedef unsigned long long ull;
#define forn(i, n) for (int i = 0; i < n; ++i)
#define each(i, x) for (auto &&i : x)
#define forne(i,x,n) for (int i = x; i < n; ++i)
#define show(x) for (auto &&i : x) {cerr << i <<' ';} cerr<< endl;
void dbg_out() { cerr << ']' << endl; }
template<typename Head, typename... Tail>
void dbg_out(Head H, Tail... T) { cerr << H;if (sizeof...(T)) cerr <<
    ',' << ' '; dbg_out(T...); }
#ifdef LOCAL
#define dbg(...) cerr << '|' << __LINE__ << '|'<< '{' << #__VA_ARGS__
    << '}'<<':'<<' '<<'[', dbg_out(__VA_ARGS__)
#else
#define dbg(...)
#endif


const int MOD = 1e9 + 7;
struct matrix {
  int n, m;
  vector<vector<int>>v;

  matrix(int n, int m, bool ones = false) : n(n), m(m), v(n,
      vector<int>(m)) {
    if (ones) forn (i, n) v[i][i] = 1;
  }

  matrix operator * (const matrix &o) {
    matrix ans(n, o.m);
    forn (i, n)
      forn (k, m) if (v[i][k])
        forn (j, o.m)
          ans[i][j] = (v[i][k] * o.v[k][j] + ans[i][j]) % MOD;
    return ans;
  }

  vector<int> & operator [] (int i) {
    return v[i];
  }
};

matrix binpow(matrix b, int e) {
  matrix ans(b.n, b.m, true);
  while (e) {
    if (e & 1) ans = ans * b;
    b = b * b;
    e >>= 1;
  }
  return ans;
}
```

```
void solve(){

    matrix a(4, 4), b(4, 4);
    a[0][0] = 4;
    a[0][1] = (-1 + MOD) % MOD;
    a[0][2] = (-1 + MOD) % MOD;
    a[0][3] = (-1 + MOD) % MOD;
    forn(i, 3) a[i + 1][i] = 1;

    b[0][0] = 17;
    b[1][0] = 3;
    b[2][0] = 2;
    b[3][0] = 1;

    int n; cin >> n;
    if(n < 4){
        cout << b[3 - n][0] << endl;
    }else{
        matrix ans = binpow(a, n - 3) * b;
        cout << ans[0][0] << endl;
    }




}
 main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    int testcase=1;
    #ifdef LOCAL
        freopen("in.txt", "r", stdin);
        freopen("out.txt", "w", stdout);
        testcase=4;
    #endif

     //cin >> testcase;
     while (testcase--)solve();
    cout << flush;
    return 0;
}
```

7.13. Hard-Fibonacci

```
//https://vjudge.net/problem/SPOJ-FIBHARD

#include <bits/stdc++.h>
using namespace std;
#define endl      '\n'
#define f         first
#define s         second
#define ins       insert
#define pb        push_back
#define eb        emplace_back
#define sz(x)     int((x).size())
#define all(x)    begin(x), end(x)
typedef long long ll;
typedef unsigned long long ull;
#define forn(i, n) for (int i = 0; i < n; ++i)
#define each(i, x) for (auto &&i : x)
#define forne(i,x,n) for (int i = x; i < n; ++i)
#define show(x) for (auto &&i : x) {cerr << i <<' ';} cerr<< endl;


void dbg_out() { cerr << ']' << endl; }
template<typename Head, typename... Tail>
void dbg_out(Head H, Tail... T) { cerr << H;if (sizeof...(T)) cerr <<
    ',' << ' '; dbg_out(T...); }
#ifdef LOCAL
#define dbg(...) cerr << '|' << __LINE__ << '|'<< '{' << #__VA_ARGS__
    << '}'<<':'<<' '<<'[', dbg_out(__VA_ARGS__)
#else
#define dbg(...)
#endif
#define int int64_t

constexpr int MOD = 998244353;
constexpr int NMOD = 998244353 + 998244353 + 2;

struct matrix {
    int n, m;
    vector<vector<int>> v;
```

```
    matrix(int n, int m, bool ones = false) : n(n), m(m), v(n,
        vector<int>(m)) {
        if (ones) forn(i, n) v[i][i] = 1;
    }

    matrix operator * (const matrix& o) {
        matrix ans(n, o.m);
        forn(i, n)
            forn(k, m) if (v[i][k])
            forn(j, o.m)
            ans[i][j] = (1ll * v[i][k] * o.v[k][j] + ans[i][j]) % MOD;
        return ans;
    }

    vector<int>& operator [] (int i) {
        return v[i];
    }
};

matrix binpow(matrix b, ll e) {
    matrix ans(b.n, b.m, true);
    while (e) {
        if (e & 1) ans = ans * b;
        b = b * b;
        e >>= 1;
    }
    return ans;
}


void solve() {


    string s; cin >> s;

    int n = 0;
    forn(i, sz(s)) n = ((n * 10) % NMOD + (s[i] - '0')) % NMOD;

    matrix fib(2, 2);
    fib[0][0] = fib[0][1] = fib[1][0] = 1; fib[1][1] = 0;

    matrix res = binpow(fib, n);

    cout << res[0][1] << endl;
}
main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);


    int testcase; cin >> testcase; while (testcase--) solve();









    cout << flush;
    return 0;
}
```

7.14. Dueling Digits

```
const int mxN = 800 + 7, mxS = 14400 + 7, mod = 1e9 + 7;
int dp[mxN][mxS << 1];

int back(int pos, int addA) {
    if (pos == 0) return addA == mxS;
    int& ans = dp[pos][addA];
    if (~ans) return ans;
    ans = 0;
    forn(i, 10) {
        forn(j, 10) {
            if (i == j) continue;
            if ((i == 0 || j == 0) && pos == 1) continue;
            ans = (ans + back(pos - 1, addA + i - j)) % mod;
        }
    }
    return ans;
}

void solve() {

    int n; cin >> n;
    cout << back(n, mxS) << endl;
```

```
}
```

8. String

8.1. SThash

```
/*
query = or.query(l, r)  ror.query(n - 1 - r, n - 1 - l)
udp   = or.upd(pos, a)  ror.upd(n - 1 - pos, a);

LLAMAR PRECAL!!!!!!!!!!!!!!!!!!!!!!!!!

*/
const int MAX = 2e5 + 6;
const int MOD = 1e9 + 7;
const int BASE = 137;

int BP[MAX];
void precal() {
    BP[0] = 1;
    BP[1] = 1;
    for (int i = 1;i < MAX;i++) {
        BP[i] = (BP[i - 1] * BASE) % MOD;
    }
}
template<typename T>
struct SThash {
    struct node {
        int tam;
        int h;
        node() {}
    };
    int n;
    vector<node>tree;
    SThash(string& s) {
        n = sz(s);
        vector<T> a(n);
        tree.resize(n * 4);
        for (int i = 0;i < n;i++) a[i] = s[i];
        build(1, 0, n - 1, a);
    }
    node Merge(node a, node b) {
        node ret;
        ret.h = ((a.h * BP[b.tam]) + b.h) % MOD;
        ret.tam = a.tam + b.tam;
        return ret;
    }

    void build(int v, int tl, int tr, vector<T>& a) {
        if (tl == tr) {
            tree[v].h = a[tl];
            tree[v].tam = 1;
            return;
        }
        int mid = (tl + tr) >> 1;
        build(v * 2, tl, mid, a);
        build(v * 2 + 1, mid + 1, tr, a);
        tree[v] = Merge(tree[v * 2], tree[v * 2 + 1]);
    }


    void upd(int v, int tl, int tr, int id, int val) {
        if (tl > id or tr < id) return;
        if (tl == tr and tr == id) {
            tree[v].h = val;
            return;
        }
        int mid = (tl + tr) >> 1;
        upd(v * 2, tl, mid, id, val);
        upd(v * 2 + 1, mid + 1, tr, id, val);
        tree[v] = Merge(tree[v * 2], tree[v * 2 + 1]);
    }

    node query(int v, int tl, int tr, int l, int r) {
        if (tl >= l and tr <= r) return tree[v];
        int mid = (tl + tr) / 2;
        if (mid < l) return query(v + v + 1, mid + 1, tr, l, r);
        else if (mid >= r) return query(v + v, tl, mid, l, r);
        else return Merge(query(v + v, tl, mid, l, r), query(v + v + 1
            , mid + 1, tr, l, r));
    }
    void upd(int pos, int val) { upd(1, 0, n - 1, pos, val); }
    int query(int l, int r) { return query(1, 0, n - 1, l, r).h; }
};
```

8.2. HuffmanCoding

```cpp
struct Node {
    char data;
    int freq;
    Node* L, * R;
    Node(char data, int freq) : data(data), freq(freq), L(nullptr),
       R(nullptr) {}
};
struct Huffman {
    unordered_map<char, int> freqMap;
    unordered_map<char, string>hfCodes;
    string str;
    Node* root;
    Huffman(string& str) : str(str) {
        for (auto&& i : str) freqMap[i]++;
        root = build();
        createHF(root, "");
    }

    struct oper {
        bool operator()(const Node* L, const Node* R) const {
            return L->freq > R->freq;
        }
    };

    Node* build() {
        priority_queue<Node*, vector<Node*>, oper> pq;
        each(i, freqMap) {
            pq.push(new Node(i.f, i.s));
        }
        if (sz(pq) == 1) {
            Node* L = pq.top();
            pq.pop();
            Node* parent = new Node('\0', L->freq);
            parent->L = L;
            pq.push(parent);
        }
        while (sz(pq) > 1) {
            Node* L = pq.top();
            pq.pop();
            Node* R = pq.top();
            pq.pop();
            Node* parent = new Node('\0', L->freq + R->freq);
            parent->L = L;
            parent->R = R;
            pq.push(parent);
        }
        return pq.top();
    }

    void createHF(Node* root, string code) {
        if (root == nullptr) return;
        if (!root->L && !root->R) {
            hfCodes[root->data] = code;
        }
        createHF(root->L, code + "0");
        createHF(root->R, code + "1");
    }

    int LengthBinary() {
        int cnt = 0;
        for (auto&& i : str) cnt += sz(hfCodes[i]);
        return cnt;
    }
    void _print() {
        each(i, hfCodes) cout << i.f << ' ' << i.s << endl;
    }

};




int main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);



    string s; cin >> s;
    Huffman hf(s);
    cout << hf.LengthBinary() << endl;

    // hf._print();







    cout << flush;
    return 0;
}
```

8.3. palindrome range

```
vector<vector<int>>dp(mxN, vector<int>(mxN));
vector<vector<bool>>pal(mxN, vector<bool>(mxN));
string s; cin >> s;
int n = sz(s), q, l, r;

for (int i = n - 1; i >= 0; i--) {
    dp[i][i] = pal[i][i] = 1;
    for (int j = i + 1; j < n; j++) {
        pal[i][j] = (pal[i + 1][j - 1] || j - i == 1) & (s[i] ==
            s[j]);
        dp[i][j] = dp[i + 1][j] + dp[i][j - 1] - dp[i + 1][j - 1]
            + pal[i][j];
    }
}
cin >> q;
while (q--) {
    cin >> l >> r;
    cout << dp[l - 1][r - 1] << endl;
}
```

8.4. hashing-64

```
#define int int64_t
const int MOD = (1ll << 61) - 1;

int mulmod(int a, int b) {
    const static int L = (1ll << 30) - 1, _31 = (1ll << 31) - 1;
    int l1 = a & L, h1 = a >> 30, l2 = b & L, h2 = b >> 30;
    int m = l1 * h2 + l2 * h1, h = h1 * h2;
    int ans = l1 * l2 + (h >> 1) + ((h & 1) << 60) + (m >> 31) + ((m &
        _31) << 30) + 1;
    ans = (ans & MOD) + (ans >> 61), ans = (ans & MOD) + (ans >> 61);
    return ans - 1;
}

int rnd(int l, int r) {
    static std::mt19937
        rng(std::chrono::steady_clock::now().time_since_epoch().count());
    return std::uniform_int_distribution<int>(l, r)(rng);
}

struct Hash {
    static int P;
    vector<int> h, p;
    Hash(string& s) : h(sz(s)), p(sz(s)) {
        p[0] = 1, h[0] = s[0];
        forne(i, 1, sz(s)) p[i] = mulmod(p[i - 1], P), h[i] =
            (mulmod(h[i - 1], P) + s[i]) % MOD;
    }
    // Returns hash of interval s[a ... b] (where 0 <= a <= b < sz(s))
    int get(int l, int r) {
        int hash = h[r] - (l ? mulmod(h[l - 1], p[r - l + 1]) : 0);
        return hash < 0 ? hash + MOD : hash;
    }
};

int Hash::P = rnd(256, MOD - 1);
```

8.5. Z

```
/*
Dado una string s, devuelve un vector Z donde Z[i] representa el
    prefijo
de mayor longitud  de s, que tambien es prefijo del sufijo de s que
    inicia
en i.
01234567
aabzaaba "aab" es un prefijo de s y "aaba" es un sufijo de s, Z[4] = 3.

Otra definicion: Dado un string s retorna un vector z donde z[i] es
    igual al mayor
numero de caracteres desde s[i] que coinciden con los caracteres desde
    s[0]

Complejidad: O(|n|)

*/
vector<int> z_function(string& s) {
    int n = s.size();
    vector<int> z(n);
    for (int i = 1, x = 0, y = 0; i < n; i++) {
        z[i] = max(0ll, min(z[i - x], y - i + 1));
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            x = i, y = i + z[i], z[i]++;
        }
    }
    return z;
}
```

8.6. paltree

```
/*
size() number of different palindrome substr
propagate() number of palindrome substr
lps longest palindrome substr {star, len}
*/

struct paltree {
    vector<vector<int>> t;
    int n, last, sz;
    vector<int> s, len, link, qt;
    pair<int, int> lps{ 0,0 };

    paltree(int N) {
        t.assign(N + 2, vector<int>(26, int()));
        s = len = link = qt = vector<int>(N + 2);
        s[0] = -1, link[0] = 1, len[0] = 0, link[1] = 1, len[1] = -1;
        sz = 2, last = 0, n = 1;
    }

    void add(char c) {
        s[n++] = c -= 'a';
        while (s[n - len[last] - 2] != c) last = link[last];
        if (!t[last][c]) {
            int prev = link[last];
            while (s[n - len[prev] - 2] != c) prev = link[prev];
            link[sz] = t[prev][c];
            len[sz] = len[last] + 2;
            t[last][c] = sz++;
            if (len[sz - 1] > lps.s) {
                lps = { n - len[sz - 1] - 1 ,len[sz - 1] };
            }
        }
        qt[last = t[last][c]]++;
    }
    int size() {
        return sz - 2;
    }

    ll propagate() {
        ll cnt = 0;
        for (int i = n; i > 1; i--) {
            qt[link[i]] += qt[i];
            cnt += qt[i];
        }
        return cnt;
    }

};
```

8.7. hashing

```
/*
Usage:
        Good values c = 137, modbest=998244353, mod = 10^9 + 7, mod =
            1e18 + 9.
        If necessary to check too many pairs of hashes, use two
        different hashes.
        If hashing something other than english characters:
            - Don't have elements with value 0.
            - Use c > max element value.
*/

//#define int int64_t

constexpr int mxN = 1e6 + 7;
vector<int>p(mxN);
void pre(const int c = 137, const int mod = 998244353) {
    p[0] = 1;
    forn(i, mxN - 1) p[i + 1] = (c * p[i]) % mod;
}

struct Hash {
    #warning llamar pre;
    int c, mod;
    vector<int> h;
    Hash(const string& s, const int c = 137, const int mod = 998244353
        ) : c(c), mod(mod), h(sz(s) + 1) {
        h[0] = 0;
        forn(i, sz(s)) h[i + 1] = (c * h[i] + s[i]) % mod;
    }
    // Returns hash of interval s[a ... b] (where 0 <= a <= b < sz(s))
    int get(int a, int b) {
        return (h[b + 1] - ((h[a] * p[b - a + 1]) % mod) + mod) % mod;
    }
};

bool same(Hash& Ha, Hash& Hb, int l, int r) {
    int qa = Ha.get(l, r);
    int qb = Hb.get(sz(Hb.h) - 2 - r, sz(Hb.h) - 2 - l);
    return qa == qb;
}
```

8.8. hash table

```
/*
hash_table
sirve para contar cuantas veces aparece un patron en un string
en un rango [l,r] en O(1) con O(n) de preprocesamiento
ejemplo:
string s;
a b a c a b a d a b a c a b a
string p;
b a
hash_table<int> h(s,p);
0 0 1 1 1 1 2 2 2 2 3 3 3 3 4

hash_table(string s, int m)
sirve para contar cuantas veces aparece un patron de longitud m en un
    string
modificar bulid() segun condicion

*/


template<typename T>
struct hash_table
{
    string s, p;
    int n, m;
    vector<T>prefix;
    hash_table(string s, string p) {
        this->s = s;
        this->p = p;
        this->n = sz(s);
        this->m = sz(p);
        prefix.resize(n + 5, 0);
        build();
    }
    hash_table(string s, int m) {
        this->s = s;
        this->n = sz(s);
        this->m = m;
        prefix.resize(n + 5, 0);
        build();
    }

    void build() {
        forn(i, n - m + 1) {
            int ok = 1;
            forn(j, m) {
                if (s[i + j] != p[j]) {
                    ok = 0;
                    break;
                }
            }
            prefix[i + 1] = prefix[i] + ok;
        }
    }
    int query(int l, int r) {
        if (r - l + 1 < m) return 0;
        return prefix[r - m + 1] - prefix[l - 1];
    }
};
```

8.9. ahobit

```
/*
ahobit: used to search for a pattern in a string
    - query(l,r): searches for how many times the pattern is repeated
       in the range [l,r]
    - numoc: number of occurrences of the pattern in the string
    - a: vector with the positions of the occurrences of the pattern
    - szp: size of the pattern
    - bs: bitset of the characters in the string
    - oc: bitset of the occurrences of the pattern
    - N: maximum size of the string
*/
struct ahobit {
    static constexpr int N = 1e5 + 9;
    bitset<N>bs[26], oc, _all;
    int szp;
    ahobit(const string& s) {
        for (int i = 0; i < sz(s); i++) bs[s[i] - 'a'][i] = 1, _all[i]
            = 1;
    }
    void add(const string& p) {
        // oc.set();
        oc = _all; szp = sz(p);
        for (int i = 0; i < sz(p); i++) oc &= (bs[p[i] - 'a'] >> i);
    }
    int num_occu() {
        return oc.count();
    }
    vector<int> pos_occu() {
        vector<int> a;
```

```
        int pos = oc._Find_first();
        a.clear(); a.pb(pos);
        pos = oc._Find_next(pos);
        while (pos < N) {
            a.pb(pos);
            pos = oc._Find_next(pos);
        }
        return a;
    }

    int query(int l, int r) {
        //1-indexed
        if (szp > r - l + 1) return 0;
        return (oc >> (l - 1)).count() - (oc >> (r - szp + 1)).count();
    }
};
```

8.10. suffixAutomaton

```
// codebreaker suffix automaton
struct suffixAutomaton {
    struct node {
        int len, link; bool end;
        map<char, int> next;
        int cnt; ll in, out, cntSubstrs;
    };

    vector<node> sa;
    //ocurrencias de estados, usar encontrar kth pequena lexico all
        strings
    vector<ll> cntState;
    int last; ll substrs = 0;

    suffixAutomaton() {}
    suffixAutomaton(string& s) {
        sa.reserve(sz(s) * 2);
        // cntState.reserve(sz(s)*2);
        last = add_node();
        sa[0].link = -1;
        sa[0].in = 1;
        for (char& c : s) add_char(c);
        for (int p = last; p; p = sa[p].link) sa[p].end = 1;
    }

    int add_node() { sa.pb({}); return sa.size() - 1; }

    void add_char(char c) {
        int u = add_node(), p = last;
        // cntState[u] = 1;
        sa[u].len = sa[last].len + 1;
        while (p != -1 && !sa[p].next.count(c)) {
            sa[p].next[c] = u;
            sa[u].in += sa[p].in;
            substrs += sa[p].in;
            p = sa[p].link;
        }
        if (p != -1) {
            int q = sa[p].next[c];
            if (sa[p].len + 1 != sa[q].len) {
                int clone = add_node();
                // cntState[clone] = 0;
                sa[clone] = sa[q];
                sa[clone].len = sa[p].len + 1;
                sa[clone].in = 0;
                sa[q].link = sa[u].link = clone;
                while (p != -1 && sa[p].next[c] == q) {
                    sa[p].next[c] = clone;
                    sa[q].in -= sa[p].in;
                    sa[clone].in += sa[p].in;
                    p = sa[p].link;
                }
            }
            else sa[u].link = q;
        }
        last = u;
    }
    //Cuenta la cantidad de ocurrencias de una cadena s
    int match_str(string& s) {
        int u = 0, n = sz(s);
        for (int i = 0; i < n; ++i) {
            if (!sa[u].next.count(s[i])) return 0;
            u = sa[u].next[s[i]];
        }
        return count_occ(u);
    }

    int count_occ(int u) {
        if (sa[u].cnt != 0) return sa[u].cnt;
        sa[u].cnt = sa[u].end;
        for (auto& v : sa[u].next)
            sa[u].cnt += count_occ(v.ss);
        return sa[u].cnt;
    }
```

```
//Calcular la cantidad de caminos que pertenecen al estado ti,
    desde ti hasta tn
ll count_paths(int u) {
    //Out cuenta la cantidad de caminos (cantidad de cadenas
        distintas)
    if (sa[u].out != 0) return sa[u].out; //sa[u].cntSubstrs != 0
        return sa[u].cntSubstrs
    for (auto& v : sa[u].next)
        sa[u].out += count_paths(v.ss); //sa[u].cntSubstrs +=
            count_paths(v.ss)
    return ++sa[u].out; //sa[u].cntSubstrs += cntState[u];
}

//kth subcadena mas pequena en orden lexicografico
//out para cadenas distintas, cntSubstrs para todas las cadenas
    llamar antes pre
string kth;
void dfs_kth(int u, ll& k) { //Antes llamar a count
    if (k == 0) return; // k < cntState[u] para todas las cadenas
    k--; // k -= cntState[u];
    for (auto& v : sa[u].next) {
        if (k < sa[v.ss].out) { //k < sa[v.ss].cntSubstrs
            kth += v.ff;
            return dfs_kth(v.ss, k);
        }
        k -= sa[v.ss].out; //k -= sa[v.ss],cntSubstrs
    }
}
//calcula la cantidad de ocurrencias de los estados
void pre() {
    vector<ii> v(sz(sa));
    forn(i, sz(sa)) v[i] = { sa[i].len, i };
    sort(all(v), greater<ii>());
    for (auto& it : v) {
        int u = it.ss;
        if (sa[u].link != -1)
            cntState[sa[u].link] += cntState[u];
    }
    cntState[0] = 1;
}
//longest common substring
int lcs(string& t) {
    int n = sz(t);
    int u = 0, l = 0, best = 0, bestPosition = 0;
    forn(i, n) {
        while (u && !sa[u].next.count(t[i])) {
            u = sa[u].link;
            l = sa[u].len;
        }
        if (sa[u].next.count(t[i])) u = sa[u].next[t[i]], l++;
        if (best < l) best = l, bestPosition = i;
    }
    return best;
}
vector<int> LCS, match;
void lcsMatch(string& t) {
    match.assign(sz(sa), 0); //usar pivote si toca resetear mucho
    int u = 0, l = 0;
    for (int i = 0; i < sz(t); ++i) {
        while (u && !sa[u].next.count(t[i])) {
            u = sa[u].link;
            l = sa[u].len;
        }
        if (sa[u].next.count(t[i])) u = sa[u].next[t[i]], l++;
        match[u] = max(match[u], l);
    }
    for (int i = sz(sa) - 1; i > 0; --i)
        match[i] = max(match[i], match[sa[i].link]);
    for (int i = 0; i < sz(sa); ++i)
        LCS[i] = min(LCS[i], match[i]);
}

//longest common substring de n cadenas
int lcs_n(vector<string>& t) {
    const int INF = 1e7;
    LCS.assign(sz(sa), INF);
    forn(i, sz(t)) lcsMatch(t[i]);
    return *max_element(all(LCS));
}

//longitud desde 1 hasta N, return v donde v[i] = num distintas
    substr de i longitud
vector<int> substringDistribution(int lenCadena) {
    vector<int> st(lenCadena + 5);
    forn(i, sz(sa)) {
        int l = sa[sa[i].link].len + 1; // l minlen subcadena que
            pertenece al conjunto sa[i]
        int r = sa[i].len; // r maxlen subcadena que pertenece al
            conjunto s[i]
        if (l > 0) st[l]++, st[r + 1]--;
    }
    forn(i, lenCadena + 1) st[i + 1] += st[i];
    return st;
```

```
    }
    //Devuelve V, V[i] = max ocurrencias para una subcadena de S de
        longitud i.
    void maxOcurrenciasLengths(int n) { //Llamar antes count_occ
        vector<int> ans(n + 1);
        forn(i, sz(sa)) ans[sa[i].len] = max(ans[sa[i].len],
            sa[i].cnt);
        forn(i, n) cout << ans[i + 1] << endl;
    }
    node& operator[](int i) { return sa[i]; }
};
```

8.11. rabin karp

```
// Dado un patron S y un texto T, se desea conocer los indices de las
    ocurrencias del patron S en el texto T.

vector<int> rabin_karp(string const& s, string const& t) {
    const int p = 31;
    const int m = 1e9 + 9;
    int S = sz(s), T = sz(t);

    vector<int64_t> p_pow(max(S, T)), h(T + 1, 0);
    p_pow[0] = 1;
    int64_t h_s = 0;
    forne(i, 1, sz(p_pow)) p_pow[i] = (p_pow[i - 1] * p) % m;
    forn(i, T) h[i + 1] = (h[i] + (t[i] - 'a' + 1) * p_pow[i]) % m;
    forn(i, S) h_s = (h_s + (s[i] - 'a' + 1) * p_pow[i]) % m;

    vector<int> occ;
    for (int i = 0; i + S - 1 < T; i++) {
        int64_t cur_h = (h[i + S] + m - h[i]) % m;
        if (cur_h == h_s * p_pow[i] % m) occ.pb(i);
    }
    return occ;
}
```

8.12. aho corasick

```
// Esta version de aho_corasick usa un bitmask de tamano ALPHABET, por
    lo que debe ser modificado para ALPHABET > 26.
// suff = el indice del nodo del sufijo estricto mas largo del nodo
    actual que tambien esta en el arbol.
// dict = el indice del nodo del sufijo estricto mas largo del nodo
    actual que esta en la lista de palabras.
// depth = profundidad normal del trie (la raiz es 0). Se puede
    eliminar para ahorrar memoria.
// word_index = el indice de la *primera* palabra que termina en este
    nodo. -1 si no hay ninguna.
// word_count = el numero total de palabras que terminan en este nodo.
    Usado en count_total_matches().
// first_child = el primer hijo de este nodo (los hijos son
    secuenciales debido al orden BFS), -1 si no hay ninguno.
// child_mask = la mascara de bits de las claves de los hijos
    disponibles desde este nodo. Si ALPHABET > 26, cambie el tipo.

const int INF = int(1e9) + 5;
template<char MIN_CHAR = 'a', typename mask_t = uint32_t>
struct aho_corasick {
    struct node {
        int suff = -1, dict = -1, depth = 0, word_index = -1,
            word_count = 0, first_child = -1;
        mask_t child_mask = 0;
        int get_child(char c) const {
            int bit = c - MIN_CHAR;
            if ((child_mask >> bit & 1) == 0) return -1;
            assert(first_child >= 0);
            return first_child + __builtin_popcount(child_mask &
                ((mask_t(1) << bit) - 1));
        }
    };

    vector<node> nodes;
    int W = 0;
    vector<int> word_location, word_indices_by_depth, defer;

    aho_corasick(const vector<string>& words = {}) { build(words); }

    // construir la adj list basada en los suffix parents. A menudo
        queremos realizar DP y/o consultas en este arbol.
    vector<vector<int>> build_suffix_adj() const {
        vector<vector<int>> adj(sz(nodes));
        forne(i, 1, sz(nodes)) adj[nodes[i].suff].push_back(i);
        return adj;
    }

    int get_or_add_child(int current, char c) {
        int bit = c - MIN_CHAR;
        if (nodes[current].child_mask >> bit & 1) return
            nodes[current].get_child(c);
```

```
    assert(nodes[current].child_mask >> bit == 0);
    int index = sz(nodes);
    nodes[current].child_mask |= mask_t(1) << bit;
    if (nodes[current].first_child < 0) nodes[current].first_child
        = index;
    nodes.emplace_back();
    nodes.back().depth = nodes[current].depth + 1;
    return index;
}

// return donde en el trie deberiamos terminar despues de comenzar
    en "location" y agregar el char "C".
// corre en el peor de los casos O(depth) pero se amortiza a O(1)
    en la mayoria de las situaciones.
int get_suffix_link(int location, char c) const {
    int child;
    while (location >= 0 && (child = nodes[location].get_child(c))
        < 0) location = nodes[location].suff;
    return location < 0 ? 0 : child;
}

void build(const vector<string>& words) {
    nodes = { node() };
    W = sz(words);
    vector<int> indices(W);
    iota(all(indices), 0);
    stable_sort(all(indices), [&](int a, int b) -> bool { return
        words[a] < words[b]; });
    word_location.assign(W, 0);
    vector<int> remaining = indices;
    int rem = W;

    for (int depth = 0; rem > 0; depth++) {
        int nrem = 0;
        forn(i, rem) {
            int word = remaining[i];
            int& location = word_location[word];
            if (depth >= int(words[word].size())) {
                if (nodes[location].word_index < 0)
                    nodes[location].word_index = word;
                nodes[location].word_count++;
            }
            else {
                location = get_or_add_child(location,
                    words[word][depth]);
                remaining[nrem++] = word;
            }
        }
        rem = nrem;
    }

    int max_depth = 0;
    defer.resize(W);

    forn(i, W) {
        max_depth = max(max_depth, int(words[i].size()));
        defer[i] = nodes[word_location[i]].word_index;
    }
    // crear una lista de indices de palabras en orden decreciente
        de profundidad, en tiempo lineal a traves de counting sort.
    word_indices_by_depth.resize(W);
    vector<int> depth_freq(max_depth + 1, 0);
    forn(i, W) depth_freq[words[i].size()]++;
    for (int i = max_depth - 1; i >= 0; i--)  depth_freq[i] +=
        depth_freq[i + 1];
    for (int i = W - 1; i >= 0; i--)
        word_indices_by_depth[--depth_freq[words[i].size()]] = i;

    // Solve suffix parents by traversing in order of depth (BFS
        order).

    forn(i, sz(nodes)) {
        mask_t child_mask = nodes[i].child_mask;
        while (child_mask != 0) {
            int bit = __builtin_ctzll(child_mask);
            char c = char(MIN_CHAR + bit);
            int index = nodes[i].get_child(c);
            child_mask ^= mask_t(1) << bit;
            // buscamos el suffix parent de index, que es el
                suffix parent de i que tiene un hijo c.
            int suffix_parent = get_suffix_link(nodes[i].suff, c);
            nodes[index].suff = suffix_parent;
            nodes[index].word_count +=
                nodes[suffix_parent].word_count;
            nodes[index].dict = nodes[suffix_parent].word_index < 0
                 ? nodes[suffix_parent].dict : suffix_parent;
        }
    }
}

// Counts the number of matches of each word in O(text length +
    num words).
vector<int> count_matches(const string& text) const {
    vector<int> matches(W, 0);
```

```
    int current = 0;
    for (char c : text) {
        current = get_suffix_link(current, c);
        int dict_node = nodes[current].word_index < 0 ?
            nodes[current].dict : current;
        if (dict_node >= 0) matches[nodes[dict_node].word_index]++;
    }

    // Iterate in decreasing order of depth.
    for (int word_index : word_indices_by_depth) {
        int location = word_location[word_index];
        int dict_node = nodes[location].dict;
        if (dict_node >= 0) matches[nodes[dict_node].word_index]
            += matches[word_index];
    }
    forn(i, W) matches[i] = matches[defer[i]];
    return matches;
}

// Finds the last index of the first occurrence of each word (INF
    if not present) in O(text length + num words).
vector<int> find_first_occurrence(const string& text) const {
    vector<int> first_occurrence(W, INF);
    int current = 0;
    forn(i, sz(text)) {
        char c = text[i];
        current = get_suffix_link(current, c);
        int dict_node = nodes[current].word_index < 0 ?
            nodes[current].dict : current;
        if (dict_node >= 0) {
            int word = nodes[dict_node].word_index;
            first_occurrence[word] = min(first_occurrence[word],
                i);
        }
    }

    // Iterate in decreasing order of depth.
    for (int word_index : word_indices_by_depth) {
        int location = word_location[word_index];
        int dict_node = nodes[location].dict;

        if (dict_node >= 0) {
            int word_parent = nodes[dict_node].word_index;
            first_occurrence[word_parent] =
                min(first_occurrence[word_parent],
                first_occurrence[word_index]);
        }
    }

    forn(i, W) first_occurrence[i] = first_occurrence[defer[i]];
    return first_occurrence;
}


vector<int> find_last_occurrence(const string& text) const {
    vector<int> first_occurrence(W, -INF);
    int current = 0;
    forn(i, sz(text)) {
        char c = text[i];
        current = get_suffix_link(current, c);
        int dict_node = nodes[current].word_index < 0 ?
            nodes[current].dict : current;
        if (dict_node >= 0) {
            int word = nodes[dict_node].word_index;
            first_occurrence[word] = max(first_occurrence[word],
                i);
        }
    }

    // Iterate in decreasing order of depth.
    for (int word_index : word_indices_by_depth) {
        int location = word_location[word_index];
        int dict_node = nodes[location].dict;

        if (dict_node >= 0) {
            int word_parent = nodes[dict_node].word_index;
            first_occurrence[word_parent] =
                max(first_occurrence[word_parent],
                first_occurrence[word_index]);
        }
    }
    forn(i, W) first_occurrence[i] = first_occurrence[defer[i]];
    return first_occurrence;
}

// Counts the number of matches over all words at each ending
    position in "text" in O(text length).
vector<int> count_matches_by_position(const string& text) const {
    vector<int> matches(sz(text));
    int current = 0;
    forn(i, sz(text)) {
        current = get_suffix_link(current, text[i]);
        matches[i] = nodes[current].word_count;
    }
```

```
        return matches;
    }

    // Counts the total number of matches of all words within "text"
       in O(text length).
    int64_t count_total_matches(const string& text) const {
        int64_t matches = 0;
        int current = 0;
        for (char c : text) {
            current = get_suffix_link(current, c);
            matches += nodes[current].word_count;
        }
        return matches;
    }
};
```

8.13. Manacher

```
// manacher receives a vector of T and returns the vector with the
   size of the palindromes
// ret[2*i] = size of the largest palindrome centered at i
// ret[2*i+1] = size of the largest palindrome centered at i and i+1
//
// Complexities:
// manacher - O(n)
// palindrome - <O(n), O(1)>
// pal_end - O(n)

template<typename T> vector<int> manacher(const T& s) {
    int l = 0, r = -1, n = s.size();
    vector<int> d1(n), d2(n);
    for (int i = 0; i < n; i++) {
        int k = i > r ? 1 : min(d1[l + r - i], r - i);
        while (i + k < n && i - k >= 0 && s[i + k] == s[i - k]) k++;
        d1[i] = k--;
        if (i + k > r) l = i - k, r = i + k;
    }
    l = 0, r = -1;
    for (int i = 0; i < n; i++) {
        int k = i > r ? 0 : min(d2[l + r - i + 1], r - i + 1); k++;
        while (i + k <= n && i - k >= 0 && s[i + k - 1] == s[i - k])
            k++;
        d2[i] = --k;
        if (i + k - 1 > r) l = i - k, r = i + k - 1;
    }
    vector<int> ret(2 * n - 1);
    for (int i = 0; i < n; i++) ret[2 * i] = 2 * d1[i] - 1;
    for (int i = 0; i < n - 1; i++) ret[2 * i + 1] = 2 * d2[i + 1];
    return ret;
}

// checks if string s[i..j] is palindrome
template<typename T> struct palindrome {
    vector<int> man;

    palindrome(const T& s) : man(manacher(s)) {}
    bool query(int i, int j) {
        return man[i + j] >= j - i + 1;
    }
};

// size of the largest palindrome ending in each position
template<typename T> vector<int> pal_end(const T& s) {
    vector<int> ret(s.size());
    palindrome<T> p(s);
    ret[0] = 1;
    for (int i = 1; i < s.size(); i++) {
        ret[i] = min(ret[i - 1] + 2, i + 1);
        while (!p.query(i - ret[i] + 1, i)) ret[i]--;
    }
    return ret;
}

void print_pals(const string s) {
    vector<int> man = manacher(s);
    int n = sz(s);
    forn(i, n) {
        for (int len = 1; len <= man[2 * i]; len += 2) {
            int start = i - (len - 1) / 2;
            cout << s.substr(start, len) << endl;
        }
        if (i < n - 1) {
            for (int len = 2; len <= man[2 * i + 1]; len += 2) {
                int start = i - (len - 2) / 2;
                cout << s.substr(start, len) << endl;
            }
        }
    }
}

//expansion
int odd(int d, int i, int n) {
```

```
    // d=(manacher[2 * i], i)
    int l = i - (d - 1) / 2;
    int r = i + (d - 1) / 2;

    while (l >= 0 && r < n) {
        //process
        l -= 1; r += 1;
    }
    return ((r - 1) - (l + 1) + 2) / 2;
}
int even(int d, int i, int n) {
    // d=(manacher[2 * i+1], i)
    if (i == n - 1) return 0;
    if (d == 0) d = 2;
    int l = i - d / 2 + 1;
    int r = i + d / 2;

    while (l >= 0 && r < n) {
        //process
        l -= 1; r += 1;
    }
    return ((r - 1) - (l + 1) + 2) / 2;
}


// largest palindrome
string manacher(const string& s) {
    if (sz(s) == 0) return "";

    string curr = "";
    for (auto&& i : s) {
        curr += i;
        curr += "#";
    }

    curr = "@#" + curr + "&";
    vector<ll> pali(sz(curr), 0);
    ll center = 0;
    ll R = 0;
    for (ll i = 1; i < sz(curr) - 1; i++) {
        if (i < R) pali[i] = min(pali[2 * center - i], R - i);
        while (curr[i + (pali[i] + 1)] == curr[i - (pali[i] + 1)])
            pali[i]++;
        if (i + pali[i] > R) {
            center = i;
            R = i + pali[i];
        }
    }
    ll HC = 0, CI = 0;
    for (ll i = 1; i < sz(curr) - 1; i++) {
        if (pali[i] > HC) {
            HC = pali[i];
            CI = i;
        }
    }
    string ans = "";
    if (HC <= 0) return string(1, s[0]);
    for (ll i = CI - HC + 1; i <= CI + HC - 1; i += 2) ans += curr[i];
    return ans;
}
```

8.14. trie

```
// T.count pref(s) number of strings that have a as a prefix
struct trie {
    vector<vector<int>> to;
    vector<int> end, pref;
    int sigma; char norm;
    int lcpsum = 0;
    trie(int sigma_ = 26, char norm_ = 'a') : sigma(sigma_),
        norm(norm_) {
        to = { vector<int>(sigma) };
        end = { 0 }, pref = { 0 };
    }
    void insert(string s) {
        int x = 0;
        for (auto c : s) {
            int& nxt = to[x][c - norm];
            if (!nxt) {
                nxt = to.size();
                to.pb(vector<int>(sigma));
                end.pb(0), pref.pb(0);
            }
            // else lcpsum += pref[nxt];
            x = nxt, pref[x]++;
        }
        end[x]++, pref[0]++;
    }
    void erase(string s) {
        int x = 0;
        for (char c : s) {
            int& nxt = to[x][c - norm];
            x = nxt, pref[x]--;
```

```
            if (!pref[x]) nxt = 0;
        }
        end[x]--, pref[0]--;
    }
    int find(string s) {
        int x = 0;
        for (auto c : s) {
            x = to[x][c - norm];
            if (!x) return -1;
        }
        return x;
    }

    int count_pref(string s) {
        int id = find(s);
        return id >= 0 ? pref[id] : 0;
    }

    string kth_word(int k, int x = 0, string s = "") {
        if (k <= end[x])  return s;
        k -= end[x];
        for (int i = 0; i < sigma; i++) {
            int nxt = to[x][i];
            if (!nxt) continue;
            if (k <= pref[nxt]) return kth_word(k, nxt, s + char(i +
                norm));
            k -= pref[nxt];
        }
        return "-1";
    }

};
```

8.15. suffixAutomaton popback

```
// tested: https://codeforces.com/gym/103185/problem/M
struct suffixAutomata {
    struct node {
        int len, link, cnt;
        int next[26];
    };

    vector<node> sa;
    vector<int> last, p1, p2, q1, qlink;
    int ans = 0; //number of distinct strings that occur at least
        twice as substrings of S
    string s;

    suffixAutomata(int mx_len) {
        sa.reserve(mx_len * 2);
        last.pb(add_node());
        sa[0].link = -1;
    }

    int add_node() { sa.pb({}); return sz(sa) - 1; }
    void add_char(char ch) {
        s.pb(ch);
        int c = ch - 'A';
        int u = add_node(), p = last.back();
        sa[u].len = sa[p].len + 1;
        while (p != -1 && !sa[p].next[c]) {
            sa[p].next[c] = u;
            p = sa[p].link;
        }
        p1.pb(p);
        if (p != -1) {
            int q = sa[p].next[c];
            q1.pb(q);
            if (sa[p].len + 1 != sa[q].len) {
                int clone = add_node();
                sa[clone] = sa[q];
                sa[clone].len = sa[p].len + 1;
                qlink.pb(sa[q].link);
                sa[q].link = sa[u].link = clone;
                while (p != -1 && sa[p].next[c] == q) {
                    sa[p].next[c] = clone;
                    p = sa[p].link;
                }
                p2.pb(p);
            }
            else sa[u].link = q;
            int v = sa[u].link;
            if (!sa[v].cnt) ans += sa[v].len - sa[sa[v].link].len;
            sa[v].cnt++;
        }
        last.pb(u);
    }

    void pop_back() {
        int c = s.back() - 'A'; s.pop_back();
        int u = last.back(); last.pop_back();
        int p = last.back();
        while (p != p1.back()) {
```

```
            sa[p].next[c] = 0;
            p = sa[p].link;
        }
        p1.pop_back();
        if (p != -1) {
            int v = sa[u].link;
            sa[v].cnt--;
            if (!sa[v].cnt) ans -= sa[v].len - sa[sa[v].link].len;
            int q = q1.back(); q1.pop_back();
            if (sa[p].len + 1 != sa[q].len) {
                sa[q].link = qlink.back(); qlink.pop_back();
                while (p != p2.back()) {
                    sa[p].next[c] = q;
                    p = sa[p].link;
                }
                p2.pop_back();
                sa.pop_back();
            }
        }
        sa.pop_back();
    }

    node& operator[](int i) { return sa[i]; }
};
```

8.16. Kmp

```
//Cuenta las ocurrencias del string p en el string s.

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

int kmp(string& s, string& p) {
    int n = s.size(), m = p.size(), cnt = 0;
    vector<int> pf = prefix_function(p);
    for (int i = 0, j = 0; i < n; i++) {
        while (j && s[i] != p[j]) j = pf[j - 1];
        if (s[i] == p[j]) j++;
        if (j == m) {
            cnt++;
            j = pf[j - 1];
        }
    }
    return cnt;
}
```

8.17. dynamic aho corasick

```
const int MX = 3 * (1e5), SIG = 26, LMX = 20;

struct aho_corasick {
    struct Node {
        Node* sig[SIG], * fail;
        int finish, cnt;
        Node() : fail(this), finish(0), cnt(0) {
            for (int i = 0; i < SIG; i++) sig[i] = this;
        }
        Node(Node* root) : fail(root), finish(0), cnt(0) {
            for (int i = 0; i < SIG; i++) sig[i] = root;
        }
    };

    Node* root;
    aho_corasick() { reset(); }
    void reset() { root = new Node; }

    void insert(string& s, int ind) {
        Node* u = root;

        for (char c : s) {
            c -= 'a';
            if (u->sig[c] == root) {
                u->sig[c] = new Node(root);
                u->sig[c]->finish = -1;
            }
            u = u->sig[c];
        }

        u->finish = ind;
        u->cnt++;
    }

    Node* getFail(Node* u, int c) {
```

```
        while (u != root && u->sig[c] == root) u = u->fail;
        return u->sig[c];
    }

    void build() {
        queue<Node*> q;

        for (int i = 0; i < SIG; i++) {
            if (root->sig[i] != root) q.push(root->sig[i]);
        }

        while (q.size()) {
            Node* u = q.front();
            q.pop();

            for (int i = 0; i < SIG; i++) {
                Node* v = u->sig[i];
                if (v != root) {
                    v->fail = getFail(u->fail, i);
                    v->cnt += v->fail->cnt;
                    q.push(v);
                }
            }
        }
    }

    int match(string& t) {
        Node* u = root;
        int res = 0;
        for (int i = 0; i < t.size(); i++) {
            char c = t[i] - 'a';
            if (u->sig[c] != root) u = u->sig[c];
            else u = getFail(u->fail, c);
            res += u->cnt;
        }
        return res;
    }
};

typedef vector<string*> vs;

struct dynamic_aho_corasick {
    aho_corasick ac[LMX];
    vs s[LMX];
    int exi;

    dynamic_aho_corasick() : exi(0) {}

    void insert(string& str) {
        int j = 0;
        while (exi & (1 << j)) j++;
        s[j].push_back(new string(str));

        for (int i = 0; i < j; i++) {
            for (string* t : s[i]) s[j].push_back(t);
            s[i].clear();
            ac[i].reset();
        }

        for (string* t : s[j]) ac[j].insert(*t, 1);
        ac[j].build();
        exi++;
    }

    int match(string& t) {
        int res = 0;

        for (int i = 0; i < LMX; i++)
            if (exi & (1 << i))
                res += ac[i].match(t);

        return res;
    }
};
```

8.18. suffixAutomaton1

```
constexpr int MAX = 1e5 + 7;

namespace sam {
    struct node {
        int len, link, cnt, fpos;
        bool acc;
        map<char, int> next;
    };
    int cur, sz;
    vector<node> sa(MAX * 2);

    void add(char c) {
        int at = cur;
        sa[cur].fpos = sa[sz].len = sa[cur].len + 1;
        sa[cur].fpos -= 1, cur = sz++;
```

```
    while (at != -1 && !sa[at].next.count(c)) sa[at].next[c] =
        cur, at = sa[at].link;
    if (at == -1) { sa[cur].link = 0; return; }
    int q = sa[at].next[c];
    if (sa[q].len == sa[at].len + 1) { sa[cur].link = q; return; }
    int qq = sz++;
    sa[qq].len = sa[at].len + 1, sa[qq].next = sa[q].next,
        sa[qq].link = sa[q].link;
    sa[qq].fpos = sa[q].fpos;
    while (at != -1 && sa[at].next[c] == q) sa[at].next[c] = qq,
        at = sa[at].link;
    sa[q].link = sa[cur].link = qq;
}

void build(string& s) {
    #warning "clear????";
    sa.assign(MAX * 2, node());
    cur = 0, sz = 0, sa[0].len = 0, sa[0].link = -1, sz++;
    for (auto& i : s) add(i);
    int at = cur;
    while (at) sa[at].acc = 1, at = sa[at].link;
}
int64_t distinct_substrings() {
    int ans = 0;
    for (int i = 1; i < sz; i++) ans += sa[i].len -
        sa[sa[i].link].len;
    return ans;
}
int longest_common_substring(string& S, string& T) {
    build(S);
    int at = 0, l = 0, ans = 0, pos = -1;
    for (int i = 0; i < sz(T); i++) {
        while (at && !sa[at].next.count(T[i])) at = sa[at].link, l
            = sa[at].len;
        if (sa[at].next.count(T[i])) at = sa[at].next[T[i]], l++;
        else at = 0, l = 0;
        if (l > ans) ans = l, pos = i;
    }
    return ans;
    // return T.substr(pos - ans + 1, ans);
}
vector<int> LCS, match;
void lcsMatch(string& t) {
    match.assign(MAX, 0);
    int u = 0, l = 0;
    for (int i = 0; i < sz(t); ++i) {
        while (u && !sa[u].next.count(t[i])) u = sa[u].link, l =
            sa[u].len;
        if (sa[u].next.count(t[i])) u = sa[u].next[t[i]], l++;
        match[u] = max(match[u], l);
    }
    for (int i = MAX - 1; i > 0; --i) match[i] = max(match[i],
        match[sa[i].link]);
    for (int i = 0; i < MAX; ++i) LCS[i] = min(LCS[i], match[i]);
}

int lcs_n(vector<string>& t) {
    const int INF = 1e7;
    LCS.assign(MAX, INF);
    forn(i, sz(t)) lcsMatch(t[i]);
    return *max_element(all(LCS));
}

int isSubstr(string& s) {
    int at = 0;
    for (auto& i : s) {
        if (!sa[at].next.count(i)) return 0;
        at = sa[at].next[i];
    }
    return at;
}

int count_occ(int u) {
    if (sa[u].cnt != 0) return sa[u].cnt;
    sa[u].cnt = sa[u].acc;
    for (auto& v : sa[u].next) sa[u].cnt += count_occ(v.s);
    return sa[u].cnt;
}

int pos_occ(string& s) {
    int x = sam::isSubstr(s);
    return x ? (abs(sam::sa[x].fpos - sz(s)) + 1) : -1;
}

int dp[2 * MAX];
int paths(int i) {
    auto& x = dp[i];
    if (x) return x;
    x = 1;
    for (char j = 'a';j <= 'z';j++) {
        if (sa[i].next.count(j)) x += paths(sa[i].next[j]);
    }
    return x;
}
```

```
    void kth_substring(int k, int at = 0) { // k=1 : menor substring
        lexicog.
        for (int i = 0; i < 26; i++) if (k && sa[at].next.count(i +
            'a')) {
            if (paths(sa[at].next[i + 'a']) >= k) {
                cout << char(i + 'a');
                kth_substring(k - 1, sa[at].next[i + 'a']);
                return;
            }
            k -= paths(sa[at].next[i + 'a']);
        }
    }
};
```

8.19. Min-Max-SuffixCyclic

```
Dado un string s devuelve el indice donde comienza la rotacion
    lexicograficamente menor de s.

int minimum_expression(string s) { //Factorizacion de lyndon
    s = s+s; // si no se concatena devuelve el indice del sufijo menor
    int len = s.size(), i = 0, j = 1, k = 0;
    while (i+k < len && j+k < len) {
        if (s[i+k] == s[j+k]) k++;
        else if (s[i+k] > s[j+k]) i = i+k+1, k = 0; // cambiar por <
            para maximum
        else j = j+k+1, k = 0;
        if (i == j) j++;
    }
    return min(i, j);
}


/*
max_suffix: retorna el inicio del sufijo lexicograficamente mayor
min_suffix: retorna el inicio del sufijo lexicograficamente menor
max_cyclic_shift: retorna el inicio del shift ciclico
    lexicograficamente mayor
min_cyclic_shift: retorna el inicio del shift ciclico
    lexicograficamente menor
*/
template<typename T> int max_suffix(T s, bool mi = false) {
    s.push_back(*min_element(s.begin(), s.end()) - 1);
    int ans = 0;
    for (int i = 1; i < s.size(); i++) {
        int j = 0;
        while (ans + j < i && s[i + j] == s[ans + j]) j++;
        if (s[i + j] > s[ans + j]) {
            if (!mi or i != s.size() - 2) ans = i;
        }
        else if (j) i += j - 1;
    }
    return ans;
}

template<typename T> int min_suffix(T s) {
    for (auto& i : s) i *= -1;
    s.push_back(*max_element(s.begin(), s.end()) + 1);
    return max_suffix(s, true);
}

template<typename T> int max_cyclic_shift(T s) {
    int n = s.size();
    for (int i = 0; i < n; i++) s.push_back(s[i]);
    return max_suffix(s);
}

template<typename T> int min_cyclic_shift(T s) {
    for (auto& i : s) i *= -1;
    return max_cyclic_shift(s);
}
```

8.20. Splitear

```
template<typename T>
T split(string& in) {
    T result;
    regex pattern("[^a-zA-Z]|paraagregarmas | |");
    in = regex_replace(in, pattern, " ");
    transform(all(in), in.begin(), ::toupper);
    istringstream iss(in);
    string token;
    if constexpr (is_same<T, vector<string>>::value) while (iss >>
        token) result.pb(token);
    else if constexpr (is_same<T, string>::value) {
        result = ""; while (iss >> token) result += token;
    }
    return result;
}
```

8.21. Suffix Array 1

```
vector<int> suffix_array(string s) {
    s += "$";
    int MAX = 260, n = sz(s), N = max(n, MAX);
    vector<int> sa(n), ra(n);
    for (int i = 0; i < n; i++) sa[i] = i, ra[i] = s[i];
    for (int k = 0; k < n; k ? k *= 2 : k++) {
        vector<int> nsa(sa), nra(n), cnt(N);
        for (int i = 0; i < n; i++) nsa[i] = (nsa[i] - k + n) % n,
            cnt[ra[i]]++;
        for (int i = 1; i < N; i++) cnt[i] += cnt[i - 1];
        for (int i = n - 1; i + 1; i--) sa[--cnt[ra[nsa[i]]]] = nsa[i];
        for (int i = 1, r = 0; i < n; i++) nra[sa[i]] = r += ra[sa[i]]
            != ra[sa[i - 1]] || ra[(sa[i] + k) % n] != ra[(sa[i - 1] +
            k) % n];
        ra = nra;
        if (ra[sa[n - 1]] == n - 1) break;
    }
    return vector<int>(sa.begin() + 1, sa.end());
}

vector<int> kasai(string s, vector<int> sa) {
    int n = sz(s), k = 0;
    vector<int> ra(n + 1), lcp(n);
    for (int i = 0; i < n; i++) ra[sa[i]] = i;
    for (int i = 0; i < n; i++, k -= !!k) {
        if (ra[i] == n - 1) { k = 0; continue; }
        int j = sa[ra[i] + 1];
        while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
        lcp[ra[i]] = k;
    }
    return lcp;
}
/*
find the number of occurrences of the string t in the string s
*/
int find_str(string& s, string& t, vector<int>& sa) {
    int n = sz(s);
    if (sz(t) > n) return 0;
    int L = 0, R = n - 1;
    int nL, nR;
    for (int i = 0; i < sz(t); i++) {
        int l = L, r = R + 1;
        while (l < r) {
            int m = (l + r) / 2;
            if (i + sa[m] >= n || s[i + sa[m]] < t[i]) l = m + 1;
            else r = m;
        }
        if (l == R + 1 || s[i + sa[l]] > t[i]) return 0;
        nL = l, l = L, r = R + 1;

        while (l < r) {
            int m = (l + r) / 2;
            if (i + sa[m] >= n || s[i + sa[m]] <= t[i]) l = m + 1;
            else r = m;
        }
        l--;
        nR = l, L = nL, R = nR;
    }
    return (nL <= nR ? nR - nL + 1 : 0);
}
/*
find the longest common substring what
appear in the string s at least least twice
*/
string lcs(vector<int>& sa, vector<int>& ka, string& s) {
    int idx = max_element(all(ka)) - begin(ka);
    return (ka[idx] > 0 ? s.substr(sa[idx], ka[idx]) : "-1");
}
/*
Find the longest common substring of two given strings s and t
create a new string s + '#' + t
compute the suffix array of the new string
compute the LCP array of the new string
pos_t = (i_s ? sa[i + 1] - (n + 1) : sa[i] - (n + 1));
*/
string find_lcs(string& s, string& t, vector<int>& sa, vector<int>&
    lcp) {
    int best = 0, n = sz(s), pos = INT_MAX;
    for (int i = 0; i < sz(lcp) - 1; i++) {
        bool i_s = (0 <= sa[i] && sa[i] <= n - 1);
        bool j_s = (0 <= sa[i + 1] && sa[i + 1] <= n - 1);
        if (i_s != j_s && best < lcp[i]) {
            best = lcp[i];
            pos = min(sa[i], sa[i + 1]);
        }
    }
    return pos == INT_MAX ? "" : s.substr(pos, best);
}
vector<int>substr_begin_by_letter(const string& s, const vector<int>&
    sa, const vector<int>& lcp) {
    vector<int>abc(26);
    int n = sz(s);
```

```
    forn(i, n) abc[s[sa[i]] - 'a'] += n - sa[i] - lcp[i];
    return abc;
}
int dis_substr(const string& s, const vector<int>& sa, const
    vector<int>& lcp) {
    int n = sz(s), ans = 0;
    forn(i, n) ans += n - sa[i] - lcp[i];
    return ans;
}
```

9. Utilities

9.1. cmd

```
"C:\w64devkit\bin\gdb.exe" !.exe
"C:\w64devkit\bin\g++.exe" -g !.cpp -o !.exe

g++ -o A A.cpp
./A

A < in.txt
A < in.txt > op.txt
```

9.2. Custom Hash

```
#define PI acos(-1)
struct chash {
    // any random-ish large odd number will do
    const uint64_t C = uint64_t(4e18 * PI) + 71;
    const uint32_t RANDOM =
        chrono::steady_clock::now().time_since_epoch().count();
    size_t operator()(uint64_t x) const { return __builtin_bswap64((x
        ^ RANDOM) * C); }
};

template <class K, class V> using u_map = unordered_map<K, V, chash>;
template <class K> using u_set = unordered_set<K, chash>;
```

9.3. template

```
#include <bits/stdc++.h>
using namespace std;
#define endl      '\n'
#define f         first
#define s         second
#define ins       insert
#define pb        push_back
#define eb        emplace_back
#define sz(x)     int((x).size())
#define all(x)    begin(x), end(x)
typedef long long ll;
typedef unsigned long long ull;
#define forn(i, n) for (int i = 0; i < n; ++i)
#define each(i, x) for (auto &&i : x)
#define forne(i,x,n) for (int i = x; i < n; ++i)
#define show(x) for (auto &&i : x) {cerr << i <<' ';} cerr<< endl;


void dbg_out() { cerr << ']' << endl; }
template<typename Head, typename... Tail>
void dbg_out(Head H, Tail... T) { cerr << H;if (sizeof...(T)) cerr <<
    ',' << ' '; dbg_out(T...); }
#ifdef LOCAL
#define dbg(...) cerr << '|' << __LINE__ << '|'<< '{' << #__VA_ARGS__
    << '}'<<':'<<' '<<'[', dbg_out(__VA_ARGS__)
#else
#define dbg(...)
#endif
#define int int64_t


signed main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
#ifdef LOCAL
    freopen("in", "r", stdin);
    freopen("out", "w", stdout);
    freopen("err", "w", stderr);
#endif




    cout << flush;
    return 0;
}
```

9.4. Pragma

```
#pragma GCC
    optimize("Ofast,unroll-loops,no-stack-protector,fast-math,inline")
#pragma GCC
    target("sse,sse2,sse3,ssse3,sse4,popcnt,lzcnt,mmx,abm,avx,avx2,bmi,bmi2,fma")
```

9.5. nodes STree

```
/*
segment whit the maximum sum
add to segment tree the node struct
T neutro = T();
T oper(T a, T b) {node::get(a, b);}
Check the base ans
*/
constexpr int inf = (1e18);
struct node {
    int lt, rt, sum, ans;
    node() : lt(-inf), rt(-inf), sum(0), ans(-inf) {}
    node(int x) : sum(x) {
        lt = rt = ans = (x);
    }
    static node get(node& a, node& b) {
        node res;
        res.sum = a.sum + b.sum;
        res.lt = max(a.lt, a.sum + b.lt);
        res.rt = max(b.rt, b.sum + a.rt);
        res.ans = max({ a.ans, b.ans, a.rt + b.lt });
        return res;
    }
};

/*
    max(al,al+1,...,ar)-min(al,al+1,...,ar)-(r-l),
*/
constexpr int inf = (1e18);
struct node {
    int len, mxl, mxr, mnl, mnr, ans;
    node() : len(0), mxl(-inf), mxr(-inf), mnl(inf), mnr(inf), ans(0)
        {}
    node(int pos, int val) : len(1), mxl(val + pos), mxr(val - pos),
        mnl(val - pos), mnr(val + pos), ans(0) {}

    static node get(node& a, node& b) {
        node res;
        res.len = a.len + b.len;
        res.mxl = max(a.mxl, b.mxl);
        res.mxr = max(a.mxr, b.mxr);
        res.mnl = min(a.mnl, b.mnl);
        res.mnr = min(a.mnr, b.mnr);
        res.ans = max({ a.ans, b.ans, a.mxl - b.mnr, b.mxr - a.mnl });
        return res;
    }
};
```

9.6. util

```
__builtin_popcount(x);  // Cuenta el numero de bits '1' en la
    representacion binaria de x.
__builtin_parity(x);    // Devuelve 1 si el numero de bits '1' en la
    representacion binaria de x es impar, 0 si es par.
__builtin_clz(x);       // Cuenta el numero de bits en '0' a la
    izquierda, desde el bit mas significativo hasta el primer '1'.
__builtin_ctz(x);       // Cuenta el numero de bits en '0' a la
    derecha, desde el bit menos significativo hasta el primer '1'.
__builtin_ffs(x);       // Encuentra la posicion del primer bit en '1'
    (contando desde 1, desde el bit menos significativo).
__lg(x);                // Devuelve el logaritmo en base 2
n & ~(1 << (x - 1));    // Apaga el m-esimo bit de n (bit 1 si m=1 es
    el menos significativo), Si m=1, apaga el bit menos significativo.
x& (-x);                // Aisla el bit menos significativo en '1' de
    x (devuelve el bit mas bajo en '1' de x).
~x & (x + 1);           // Aisla el bit menos significativo en '0' de
    x.
x | (x + 1);            // Enciende el bit menos significativo en '0'
    de x.
x& (x - 1);             // Apaga el bit menos significativo en '1' de
    x.
-~n;                    // Suma 1 a n.
~- n;                   // Resta 1 a n.
x && (!(x& (x - 1)));  // Comprueba si x es una potencia de 2.

//Rotar una matriz 90 grados
vector<vector<int>> rotar(vector<vector<int>>& a) {
  int n = sz(a), m = sz(a[0]);
  vector<vector<int>> v(m, vector<int>(n));
  forn(i, n) {
    forn(j, m) {
      v[j][n - 1 - i] = a[i][j];
    }
```

```
  }
  return v;
}

//123456789101112131415161718... what is the digit at position n?
char digit_at_pos(int n) {
  n--; // 0 index
  int len = 9, mm = 1;
  forne(i, 1, 32) { // change 32 to 64 if needed
    if (n < len) {
      int num = n / i + mm, pos = n % i;
      return to_string(num)[pos];
    }
    n -= len, mm *= 10, len = 9 * mm * (i + 1);
  }
}

// sum of even or odd numbers from l to r
auto eve = [&](int l, int r) { return ((r / 2) * ((r / 2) + 1)) - (((l
    - 1) / 2) * (((l - 1) / 2) + 1));};
auto odd = [&](int l, int r) { return (r * (r + 1) / 2) - ((l - 1) *
    ((l - 1) + 1) / 2) - eve(l, r); };

// > need
auto upper_bound = [&](int need) ->int {
  int l = -1, r = n;
  while (r - l > 1) {
    int mid = l + (r - l) / 2;
    if (nums[mid] <= need) l = mid;
    else r = mid;
  }
  return l;
  };
// >= need
auto lower_bound = [&](int need) ->int {
  int l = -1, r = n;
  while (r - l > 1) {
    int mid = l + (r - l) / 2;
    if (nums[mid] < need) l = mid;
    else r = mid;
  }
  return r;
  };
// == need
auto search = [&](int need) ->bool {
  int l = -1, r = n;
  while (r - l > 1) {
    int mid = l + (r - l) / 2;
    if (nums[mid] < need) l = mid;
    else r = mid;
  }
  return nums[r] == need;
  };

// xor sum from 0 to x
int xorsum(int x) {
  if (x % 4 == 0) return x;
  else if (x % 4 == 2) return x + 1;
  else if (x % 4 == 1) return 1;
  else return 0;
};

/* resultado de & en el rango [l, r] */
ll rangeAND(ll l, ll r) {
  ll ans = 0;
  rforn(i, 63) {
    if ((l & (1ll << i)) != (r & (1ll << i))) break;
    ans |= (l & (1ll << i));
  }
  return ans;
}
```

9.7. Plantillap

```
import math
import sys
input = sys.stdin.readline
write = sys.stdout.write
def fast_print(x): write(str(x) + '\n')

def main():

    fast_print("Hello!!")

if __name__ == '__main__':
    main()
```

9.8. Stres

```
import subprocess
```

```python
def run_command(command, input_data=None):
    process = subprocess.Popen(command, stdin=subprocess.PIPE,
        stdout=subprocess.PIPE, text=True)
    stdout, _ = process.communicate(input_data)
    return stdout.strip()

def compile_cpp(source_file, output_file):
    compile_command = ["g++", source_file, "-o", output_file, "-O2",
        "-std=c++11"]
    result = subprocess.run(compile_command)
    return result.returncode == 0

compile_cpp("brute.cpp", "brute")
compile_cpp("main.cpp", "main")
compile_cpp("gen.cpp", "gen")

for i in range(100000):
    testcase = run_command(["./gen"])
    brute_output = run_command(["./brute"], testcase)
    main_output = run_command(["./main"], testcase)
    if brute_output != main_output:
        print("Testcase:\n", testcase)
        print("Output brute:\n", brute_output)
        print("Output sol:\n", main_output)
        break
    else :
        print("Testcase", i, "OK")
```

9.9. random

```cpp
int rnd(int l, int r) {
    static std::mt19937
        rng(std::chrono::steady_clock::now().time_since_epoch().count());
    return std::uniform_int_distribution<int>(l, r)(rng);
}
```

9.10. int128

```cpp
using lint = __int128;

istream& operator>>(istream &in, lint &x) {
    string s; in >> s;
    x = 0; bool neg = 0; int i = 0;
    if (s[0] == '-') neg = 1, i = 1;
    for (; i < sz(s); i++) x = x * 10 + (s[i] - '0');
    if (neg) x *= -1;
    return in;
}

ostream& operator<<(ostream &out, lint x) {
    if (x == 0) return out << "0";
    if (x < 0) out << '-', x = -x;
    string s;
    while (x) s += '0' + x % 10, x /= 10;
    reverse(all(s));
    return out << s;
}
```