

The Reference

Contents

1 Data structures	2	5 Graphs	10	6.18 Lcm	18
1.1 Dsu	2	5.1 2 SAT (struct)	10	6.19 Modular Inverse Using Phi	18
1.2 Dsu (Python)	2	5.2 SCC (struct)	11	6.20 N Choose K Count	19
1.3 Ordered Set Gnu Pbds	2	5.3 SCC Nodes (kosajaru)	11	6.21 Permutation Count	19
1.4 Segtree Rmaxq Rmaxu	2	5.4 Bellman Ford	11	6.22 Polynomial	19
1.5 Segtree Rminq Pau	3	5.5 Check Bipartite	12	6.23 Power Sum	19
1.6 Segtree Rminq Rsu	3	5.6 Count SCC (kosajaru)	12	6.24 Sieve List Primes	19
1.7 Segtree Rsq Rsu	4	5.7 Dijkstra	12	7 Searching	19
1.8 Sparse Table Rminq	5	5.8 Floyd Warshall	12	7.1 Ternary Search Recursive	19
2 Dynamic programming	5	5.9 Kruskal (Python)	12	8 Strings	20
2.1 Edit Distance	5	5.10 Lowest Common Ancestor Sparse Table	13	8.1 Hash Range Query	20
2.2 Kadane	5	5.11 Topological Sorting	13	8.2 Longest Palindrome	20
2.3 Knapsack (value)	5	6 Math	14	8.3 Rabin Karp	20
2.4 Knapsack With Elements	5	6.1 Arithmetic Progression Sum	14	8.4 String Psum	21
2.5 Longest Increasing Sequence	6	6.2 Combinatorics With Repetitions	14	8.5 Suffix Automaton (complete)	21
2.6 Money Sum Bottom Up	6	6.3 Count Divisors Memo	14	8.6 Trie Naive	22
2.7 Tsp	6	6.4 Euler Phi	14	8.7 Z Function Get Occurence Positions	22
3 Extras	6	6.5 Factorial Factorization	15	9 Trees	23
3.1 Bigint	6	6.6 Factorial	15	9.1 Binary Lifting	23
3.2 Binary To Gray	9	6.7 Factorization (Pollard who)	15	9.2 Maximum Distances	23
3.3 Get Permutation Cicles	9	6.8 Factorization With Primes	16	9.3 Tree Diameter	24
4 Geometry	10	6.9 Factorization	16	10 Settings and macros	24
4.1 Point Template	10	6.10 Fast Fourier Transform	16	10.1 short-macro.cpp	24
		6.11 Fast Exp	16	10.2 .vimrc	24
		6.12 Gauss Elimination	16	10.3 degug.cpp	24
		6.13 Gcd Using Factorization	17	10.4 .bashrc	25
		6.14 Gcd	17	10.5 macro.cpp	26
		6.15 Integer Mod	17		
		6.16 Is Prime	18		
		6.17 Lcm Using Factorization	18		

1 Data structures

1.1 Dsu

```
struct DSU {
    vector<int> ps;
    vector<int> size;
    DSU(int N) : ps(N + 1), size(N + 1, 1) { iota(ps.begin(), ps.end(), 0); }
    int find_set(int x) { return ps[x] == x ? x : ps[x] = find_set(ps[x]); }
    bool same_set(int x, int y) { return find_set(x) == find_set(y); }
    void union_set(int x, int y) {
        if (same_set(x, y)) return;

        int px = find_set(x);
        int py = find_set(y);

        if (size[px] < size[py]) swap(px, py);

        ps[py] = px;
        size[px] += size[py];
    }
};
```

1.2 Dsu (Python)

```
class DSU:
    def __init__(self, n):
        self.n = n
        self.p = [x for x in range(0, n + 1)]
        self.size = [0 for i in range(0, n + 1)]

    def find_set(self, x): # log n
        if self.p[x] == x:
            return x
        else:
            self.p[x] = self.find_set(self.p[x])
            return self.p[x]

    def same_set(self, x, y): # log n
        return bool(self.find_set(x) == self.find_set(y))

    def union_set(self, x, y): # log n
        px = self.find_set(x)
        py = self.find_set(y)

        if px == py:
            return

        size_x = self.size[px]
        size_y = self.size[py]

        if size_x > size_y:
            self.p[py] = self.p[px]
            self.size[px] += self.size[py]
        else:
            self.p[px] = self.p[py]
            self.size[py] += self.size[px]
```

1.3 Ordered Set Gnu Pbds

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T>
// using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
// tree_order_statistics_node_update>;

// if you want to find the elements less or equal :p
using ordered_set = tree<T, null_type, less_equal<T>, rb_tree_tag,
                        tree_order_statistics_node_update>;
```

1.4 Segtree Rmaxq Rmaxu

```
template <typename T = ll>
struct SegTree {
    int N;
    T nu, nq;
    vector<T> st, lazy;
    SegTree(const vector<T> &xs)
        : N(len(xs)),
          nu(numeric_limits<T>::min()),
          nq(numeric_limits<T>::min()),
          st(4 * N + 1, nu),
          lazy(4 * N + 1, nu) {
        for (int i = 0; i < len(xs); ++i) update(i, i, xs[i]);
    }

    void update(int l, int r, T value) { update(1, 0, N - 1, l, r, value); }

    T query(int l, int r) { return query(1, 0, N - 1, l, r); }

    void update(int node, int nl, int nr, int ql, int qr, T v) {
        propagation(node, nl, nr);

        if (ql > nr or qr < nl) return;

        st[node] = max(st[node], v);
        if (ql <= nl and nr <= qr) {
            if (nl < nr) {
                lazy[left(node)] = max(lazy[left(node)], v);
                lazy[right(node)] = max(lazy[right(node)], v);
            }
            return;
        }
        update(left(node), nl, mid(nl, nr), ql, qr, v);
        update(right(node), mid(nl, nr) + 1, nr, ql, qr, v);

        st[node] = max(st[left(node)], st[right(node)]);
    }

    T query(int node, int nl, int nr, int ql, int qr) {
        propagation(node, nl, nr);

        if (ql > nr or qr < nl) return nq;

        if (ql <= nl and nr <= qr) return st[node];
```

```

    T x = query(left(node), nl, mid(nl, nr), ql, qr);
    T y = query(right(node), mid(nl, nr) + 1, nr, ql, qr);

    return max(x, y);
}

void propagation(int node, int nl, int nr) {
    if (lazy[node] != nu) {
        st[node] = max(st[node], lazy[node]);

        if (nl < nr) {
            lazy[left(node)] = max(lazy[left(node)], lazy[node]);
            lazy[right(node)] = max(lazy[right(node)], lazy[node]);
        }

        lazy[node] = nu;
    }
}

int left(int p) { return p << 1; }
int right(int p) { return (p << 1) + 1; }
int mid(int l, int r) { return (r - l) / 2 + 1; }
};

int main() {
    int n;
    cin >> n;
    vector<array<int, 3>> xs(n);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < 3; ++j) {
            cin >> xs[i][j];
        }
    }
    vi aux(n, 0);
    SegTree<int> st(aux);
    for (int i = 0; i < n; ++i) {
        int a = min(i + xs[i][1], n);
        int b = min(i + xs[i][2], n);
        st.update(i, i, st.query(i, i) + xs[i][0]);
        int cur = st.query(i, i);
        st.update(a, b, cur);
    }

    cout << st.query(0, n) << '\n';
}

```

1.5 Segtree Rminq Pau

```

template <typename T = ll>
struct SegTree {
    int n;
    T nu, nq;
    vector<T> st;
    SegTree(const vector<T> &v)
        : n(len(v)), nu(0), nq(numeric_limits<T>::max()), st(n * 4 + 1, nu) {
        for (int i = 0; i < n; ++i) update(i, v[i]);
    }
    void update(int p, T v) { update(1, 0, n - 1, p, v); }

```

```

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

    void update(int node, int nl, int nr, int p, T v) {
        if (p < nl or p > nr) return;

        if (nl == nr) {
            st[node] = v;
            return;
        }

        update(left(node), nl, mid(nl, nr), p, v);
        update(right(node), mid(nl, nr) + 1, nr, p, v);

        st[node] = min(st[left(node)], st[right(node)]);
    }

    T query(int node, int nl, int nr, int ql, int qr) {
        if (ql <= nl and qr >= nr) return st[node];
        if (nl > qr or nr < ql) return nq;
        if (nl == nr) return st[node];

        return min(query(left(node), nl, mid(nl, nr), ql, qr),
                    query(right(node), mid(nl, nr) + 1, nr, ql, qr));
    }

    int left(int p) { return p << 1; }
    int right(int p) { return (p << 1) + 1; }
    int mid(int l, int r) { return (r - l) / 2 + 1; }
};

```

1.6 Segtree Rminq Rsu

```

template <typename t = ll>
struct SegTree {
    int n;
    t nu;
    t nq;
    vector<t> st, lazy;
    SegTree(const vector<t> &xs)
        : n(len(xs)),
          nu(0),
          nq(numeric_limits<t>::max()),
          st(4 * n, nu),
          lazy(4 * n, nu) {
        for (int i = 0; i < len(xs); ++i) update(i, i, xs[i]);
    }

    SegTree(int n) : n(n), st(4 * n, nu), lazy(4 * n, nu) {}

    void update(int l, int r, ll value) { update(1, 0, n - 1, l, r, value); }

    t query(int l, int r) { return query(1, 0, n - 1, l, r); }

    void update(int node, int nl, int nr, int ql, int qr, ll v) {
        propagation(node, nl, nr);

        if (ql > nr or qr < nl) return;

```

```

    if (ql <= nl and nr <= qr) {
        st[node] += (nr - nl + 1) * v;

        if (nl < nr) {
            lazy[left(node)] += v;
            lazy[right(node)] += v;
        }

        return;
    }

    update(left(node), nl, mid(nl, nr), ql, qr, v);
    update(right(node), mid(nl, nr) + 1, nr, ql, qr, v);

    st[node] = min(st[left(node)], st[right(node)]);
}

t query(int node, int nl, int nr, int ql, int qr) {
    propagation(node, nl, nr);

    if (ql > nr or qr < nl) return nq;

    if (ql <= nl and nr <= qr) return st[node];

    t x = query(left(node), nl, mid(nl, nr), ql, qr);
    t y = query(right(node), mid(nl, nr) + 1, nr, ql, qr);

    return min(x, y);
}

void propagation(int node, int nl, int nr) {
    if (lazy[node]) {
        st[node] += lazy[node];

        if (nl < nr) {
            lazy[left(node)] += lazy[node];
            lazy[right(node)] += lazy[node];
        }

        lazy[node] = nu;
    }
}

int left(int p) { return p << 1; }
int right(int p) { return (p << 1) + 1; }
int mid(int l, int r) { return (r - l) / 2 + 1; }
};

```

1.7 Segtree Rsq Rsu

```

template <typename T = ll>
struct SegTree {
    int N;
    vector<T> st, lazy;
    T nu = 0;
    T nq = 0;
    SegTree(const vector<T> &xs) : N(len(xs)), st(4 * N, nu), lazy(4 * N, nu) {
        for (int i = 0; i < len(xs); ++i) update(i, i, xs[i]);
    }
};

```

```

}

SegTree(int n) : N(n), st(4 * N, nu), lazy(4 * N, nu) {}

void update(int l, int r, ll value) { update(1, 0, N - 1, l, r, value); }

T query(int l, int r) { return query(1, 0, N - 1, l, r); }

void update(int node, int nl, int nr, int ql, int qr, ll v) {
    propagation(node, nl, nr);

    if (ql > nr or qr < nl) return;

    if (ql <= nl and nr <= qr) {
        st[node] += (nr - nl + 1) * v;

        if (nl < nr) {
            lazy[left(node)] += v;
            lazy[right(node)] += v;
        }

        return;
    }

    update(left(node), nl, mid(nl, nr), ql, qr, v);
    update(right(node), mid(nl, nr) + 1, nr, ql, qr, v);

    st[node] = st[left(node)] + st[right(node)];
}

T query(int node, int nl, int nr, int ql, int qr) {
    propagation(node, nl, nr);

    if (ql > nr or qr < nl) return nq;

    if (ql <= nl and nr <= qr) return st[node];

    T x = query(left(node), nl, mid(nl, nr), ql, qr);
    T y = query(right(node), mid(nl, nr) + 1, nr, ql, qr);

    return x + y;
}

void propagation(int node, int nl, int nr) {
    if (lazy[node]) {
        st[node] += (nr - nl + 1) * lazy[node];

        if (nl < nr) {
            lazy[left(node)] += lazy[node];
            lazy[right(node)] += lazy[node];
        }

        lazy[node] = nu;
    }
}

int left(int p) { return p << 1; }
int right(int p) { return (p << 1) + 1; }

```

```

    int mid(int l, int r) { return (r - l) / 2 + 1; }
};

```

1.8 Sparse Table Rminq

```

/*
    Sparse table implementation for rmq.
    build: O(NlogN)
    query: O(1)
*/
int fastlog2(ll x) {
    ull i = x;
    return i ? __builtin_clzll(1) - __builtin_clzll(i) : -1;
}
template <typename T>
class SparseTable {
public:
    int N;
    int K;
    vector<vector<T>> st;
    SparseTable(vector<T> vs)
        : N((int)vs.size()), K(fastlog2(N) + 1), st(K + 1, vector<T>(N + 1)) {
        copy(vs.begin(), vs.end(), st[0].begin());

        for (int i = 1; i <= K; ++i)
            for (int j = 0; j + (1 << i) <= N; ++j)
                st[i][j] = min(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
    }
    T RMQ(int l, int r) { // [l, r], 0 indexed
        int i = fastlog2(r - l + 1);
        return min(st[i][l], st[i][r - (1 << i) + 1]);
    }
};

```

2 Dynamic programming

2.1 Edit Distance

```

int edit_distance(const string &a, const string &b) {
    int n = a.size();
    int m = b.size();
    vector<vi> dp(n + 1, vi(m + 1, 0));

    int ADD = 1, DEL = 1, CHG = 1;
    for (int i = 0; i <= n; ++i) {
        dp[i][0] = i * DEL;
    }
    for (int i = 1; i <= m; ++i) {
        dp[0][i] = ADD * i;
    }

    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
            int add = dp[i][j - 1] + ADD;
            int del = dp[i - 1][j] + DEL;
            int chg = dp[i - 1][j - 1] + (a[i - 1] == b[j - 1] ? 0 : 1) * CHG;
            dp[i][j] = min({add, del, chg});
        }
    }
}

```

```

    }
}

return dp[n][m];
}

```

2.2 Kadane

```

/*
    * Find the maximum sum subarray in a given array.
    * O(N)
    * */
int kadane(const vi &as) {
    vi s(len(as));
    s[0] = as[0];

    for (int i = 1; i < len(as); ++i) s[i] = max(as[i], s[i - 1] + as[i]);

    return *max_element(all(s));
}

```

2.3 Knapsack (value)

```

const int MAXN{2010}, MAXM{2010};

ll st[MAXN][MAXM];

ll dp(int i, int m, int M, const vii &cs) {
    if (i < 0) return 0;

    if (st[i][m] != -1) return st[i][m];

    auto res = dp(i - 1, m, M, cs);
    auto [w, v] = cs[i];

    if (w <= m) res = max(res, dp(i - 1, m - w, M, cs) + v);

    st[i][m] = res;
    return res;
}

ll knapsack(int M, const vii &cs) {
    memset(st, -1, sizeof st);

    return dp((int)cs.size() - 1, M, M, cs);
}

```

2.4 Knapsack With Elements

```

const int MAXN{2010}, MAXM{2010};
ll st[MAXN][MAXM];
char ps[MAXN][MAXM];

pair<ll, vi> knapsack(int M, const vii &cs) {
    int N = len(cs) - 1;

    for (int i = 0; i <= N; ++i) st[i][0] = 0;
}

```

```

for (int m = 0; m <= M; ++m) st[0][m] = 0;

for (int i = 1; i <= N; ++i) {
    for (int m = 1; m <= M; ++m) {
        st[i][m] = st[i - 1][m];
        ps[i][m] = 0;
        auto [w, v] = cs[i];

        if (w <= m and st[i - 1][m - w] + v > st[i][m]) {
            st[i][m] = st[i - 1][m - w] + v;
            ps[i][m] = 1;
        }
    }
}

int m = M;
vi is;
for (int i = N; i >= 1; --i) {
    if (ps[i][m]) {
        is.push_back(i);
        m -= cs[i].first;
    }
}

reverse(all(is));

// max value, items
return {st[N][M], is};
}

```

2.5 Longest Increasing Sequence

```

int LIS(int N, const vector<int> &as) {
    vector<int> lis(N + 1, oo);
    lis[0] = -oo;

    auto ans = 0;

    for (int i = 0; i < N; ++i) {
        auto it = lower_bound(lis.begin(), lis.end(), as[i]);
        auto pos = (int)(it - lis.begin());

        ans = max(ans, pos);
        lis[pos] = as[i];
    }

    return ans;
}

```

2.6 Money Sum Bottom Up

```

/*
    find every possible sum using
    the given values only once.
*/
set<int> money_sum(const vi &xs) {
    using vc = vector<char>;

```

```

    using vvc = vector<vc>;
    int _m = accumulate(all(xs), 0);
    int _n = xs.size();
    vvc _dp(_n + 1, vc(_m + 1, 0));
    set<int> _ans;
    _dp[0][xs[0]] = 1;
    for (int i = 1; i < _n; ++i) {
        for (int j = 0; j <= _m; ++j) {
            if (j == 0 or _dp[i - 1][j]) {
                _dp[i][j + xs[i]] = 1;
                _dp[i][j] = 1;
            }
        }
    }

    for (int i = 0; i < _n; ++i)
        for (int j = 0; j <= _m; ++j)
            if (_dp[i][j]) _ans.insert(j);
    return _ans;
}

```

2.7 Tsp

```

using vi = vector<int>;
vector<vi> dist;
vector<vi> memo;
/* 0 ( N^2 * 2^N )*/
int tsp(int i, int mask, int N) {
    if (mask == (1 << N) - 1) return dist[i][0];
    if (memo[i][mask] != -1) return memo[i][mask];
    int ans = INT_MAX << 1;
    for (int j = 0; j < N; ++j) {
        if (mask & (1 << j)) continue;
        auto t = tsp(j, mask | (1 << j), N) + dist[i][j];
        ans = min(ans, t);
    }
    return memo[i][mask] = ans;
}

```

3 Extras

3.1 Bigint

```

const int maxn = 1e2 + 14, lg = 15;
const int base = 1000000000;
const int base_digits = 9;
struct bigint {
    vector<int> a;
    int sign;

    int size() {
        if (a.empty()) return 0;
        int ans = (a.size() - 1) * base_digits;
        int ca = a.back();
        while (ca) ans++, ca /= 10;
        return ans;
    }
}

```

```

bigint operator^(const bigint &v) {
    bigint ans = 1, a = *this, b = v;
    while (!b.isZero()) {
        if (b % 2) ans *= a;
        a *= a, b /= 2;
    }
    return ans;
}

string to_string() {
    stringstream ss;
    ss << *this;
    string s;
    ss >> s;
    return s;
}

int sumof() {
    string s = to_string();
    int ans = 0;
    for (auto c : s) ans += c - '0';
    return ans;
}

/*</arpa>*/
bigint() : sign(1) {}

bigint(long long v) { *this = v; }

bigint(const string &s) { read(s); }

void operator=(const bigint &v) {
    sign = v.sign;
    a = v.a;
}

void operator=(long long v) {
    sign = 1;
    a.clear();
    if (v < 0) sign = -1, v = -v;
    for (; v > 0; v = v / base) a.push_back(v % base);
}

bigint operator+(const bigint &v) const {
    if (sign == v.sign) {
        bigint res = v;

        for (int i = 0, carry = 0; i < (int)max(a.size(), v.a.size()) || carry; ++i) {
            if (i == (int)res.a.size()) res.a.push_back(0);
            res.a[i] += carry + (i < (int)a.size() ? a[i] : 0);
            carry = res.a[i] >= base;
            if (carry) res.a[i] -= base;
        }
        return res;
    }
    return *this - (-v);
}

bigint operator-(const bigint &v) const {
    if (sign == v.sign) {

```

```

        if (abs() >= v.abs()) {
            bigint res = *this;
            for (int i = 0, carry = 0; i < (int)v.a.size() || carry; ++i) {
                res.a[i] -= carry + (i < (int)v.a.size() ? v.a[i] : 0);
                carry = res.a[i] < 0;
                if (carry) res.a[i] += base;
            }
            res.trim();
            return res;
        }
        return -(v - *this);
    }
}

return *this + (-v);
}

void operator*=(int v) {
    if (v < 0) sign = -sign, v = -v;
    for (int i = 0, carry = 0; i < (int)a.size() || carry; ++i) {
        if (i == (int)a.size()) a.push_back(0);
        long long cur = a[i] * (long long)v + carry;
        carry = (int)(cur / base);
        a[i] = (int)(cur % base);
        // asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) :
        // "A"(cur), "c"(base));
    }
    trim();
}

bigint operator*(int v) const {
    bigint res = *this;
    res *= v;
    return res;
}

void operator*=(long long v) {
    if (v < 0) sign = -sign, v = -v;
    if (v > base) {
        *this = *this * (v / base) * base + *this * (v % base);
        return;
    }
    for (int i = 0, carry = 0; i < (int)a.size() || carry; ++i) {
        if (i == (int)a.size()) a.push_back(0);
        long long cur = a[i] * (long long)v + carry;
        carry = (int)(cur / base);
        a[i] = (int)(cur % base);
        // asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) :
        // "A"(cur), "c"(base));
    }
    trim();
}

bigint operator*(long long v) const {
    bigint res = *this;
    res *= v;
    return res;
}

friend pair<bigint, bigint> divmod(const bigint &a1, const bigint &b1) {

```

```

int norm = base / (b1.a.back() + 1);
bigint a = a1.abs() * norm;
bigint b = b1.abs() * norm;
bigint q, r;
q.a.resize(a.a.size());

for (int i = a.a.size() - 1; i >= 0; i--) {
    r *= base;
    r += a.a[i];
    int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()];
    int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a.size() - 1];
    int d = ((long long)base * s1 + s2) / b.a.back();
    r -= b * d;
    while (r < 0) r += b, --d;
    q.a[i] = d;
}

q.sign = a1.sign * b1.sign;
r.sign = a1.sign;
q.trim();
r.trim();
return make_pair(q, r / norm);
}

bigint operator/(const bigint &v) const { return divmod(*this, v).first; }

bigint operator%(const bigint &v) const { return divmod(*this, v).second; }

void operator/=(int v) {
    if (v < 0) sign = -sign, v = -v;
    for (int i = (int)a.size() - 1, rem = 0; i >= 0; --i) {
        long long cur = a[i] + rem * (long long)base;
        a[i] = (int)(cur / v);
        rem = (int)(cur % v);
    }
    trim();
}

bigint operator/(int v) const {
    bigint res = *this;
    res /= v;
    return res;
}

int operator%(int v) const {
    if (v < 0) v = -v;
    int m = 0;
    for (int i = a.size() - 1; i >= 0; --i)
        m = (a[i] + m * (long long)base) % v;
    return m * sign;
}

void operator+=(const bigint &v) { *this = *this + v; }
void operator-=(const bigint &v) { *this = *this - v; }
void operator*=(const bigint &v) { *this = *this * v; }
void operator/=(const bigint &v) { *this = *this / v; }

bool operator<(const bigint &v) const {

```

```

    if (sign != v.sign) return sign < v.sign;
    if (a.size() != v.a.size()) return a.size() * sign < v.a.size() * v.sign;
    for (int i = a.size() - 1; i >= 0; i--)
        if (a[i] != v.a[i]) return a[i] * sign < v.a[i] * v.sign;
    return false;
}

bool operator>(const bigint &v) const { return v < *this; }
bool operator<=(const bigint &v) const { return !(v < *this); }
bool operator>=(const bigint &v) const { return !(*this < v); }
bool operator==(const bigint &v) const {
    return !(*this < v) && !(v < *this);
}
bool operator!=(const bigint &v) const { return *this < v || v < *this; }

void trim() {
    while (!a.empty() && !a.back()) a.pop_back();
    if (a.empty()) sign = 1;
}

bool isZero() const { return a.empty() || (a.size() == 1 && !a[0]); }

bigint operator-() const {
    bigint res = *this;
    res.sign = -sign;
    return res;
}

bigint abs() const {
    bigint res = *this;
    res.sign *= res.sign;
    return res;
}

long long longValue() const {
    long long res = 0;
    for (int i = a.size() - 1; i >= 0; i--) res = res * base + a[i];
    return res * sign;
}

friend bigint gcd(const bigint &a, const bigint &b) {
    return b.isZero() ? a : gcd(b, a % b);
}

friend bigint lcm(const bigint &a, const bigint &b) {
    return a / gcd(a, b) * b;
}

void read(const string &s) {
    sign = 1;
    a.clear();
    int pos = 0;
    while (pos < (int)s.size() && (s[pos] == '-' || s[pos] == '+')) {
        if (s[pos] == '-') sign = -sign;
        ++pos;
    }
    for (int i = s.size() - 1; i >= pos; i -= base_digits) {
        int x = 0;
        for (int j = max(pos, i - base_digits + 1); j <= i; j++)

```



```

        x = x * 10 + s[j] - '0';
        a.push_back(x);
    }
    trim();
}

friend istream &operator>>(istream &stream, bigint &v) {
    string s;
    stream >> s;
    v.read(s);
    return stream;
}

friend ostream &operator<<(ostream &stream, const bigint &v) {
    if (v.sign == -1) stream << '-';
    stream << (v.a.empty() ? 0 : v.a.back());
    for (int i = (int)v.a.size() - 2; i >= 0; --i)
        stream << setw(base_digits) << setfill('0') << v.a[i];
    return stream;
}

static vector<int> convert_base(const vector<int> &a, int old_digits,
                                int new_digits) {
    vector<long long> p(max(old_digits, new_digits) + 1);
    p[0] = 1;
    for (int i = 1; i < (int)p.size(); i++) p[i] = p[i - 1] * 10;
    vector<int> res;
    long long cur = 0;
    int cur_digits = 0;
    for (int i = 0; i < (int)a.size(); i++) {
        cur += a[i] * p[cur_digits];
        cur_digits += old_digits;
        while (cur_digits >= new_digits) {
            res.push_back(int(cur % p[new_digits]));
            cur /= p[new_digits];
            cur_digits -= new_digits;
        }
    }
    res.push_back((int)cur);
    while (!res.empty() && !res.back()) res.pop_back();
    return res;
}

typedef vector<long long> vll;

static vll karatsubaMultiply(const vll &a, const vll &b) {
    int n = a.size();
    vll res(n + n);
    if (n <= 32) {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) res[i + j] += a[i] * b[j];
        return res;
    }

    int k = n >> 1;
    vll a1(a.begin(), a.begin() + k);
    vll a2(a.begin() + k, a.end());
    vll b1(b.begin(), b.begin() + k);

```

```

        vll b2(b.begin() + k, b.end());

    vll a1b1 = karatsubaMultiply(a1, b1);
    vll a2b2 = karatsubaMultiply(a2, b2);

    for (int i = 0; i < k; i++) a2[i] += a1[i];
    for (int i = 0; i < k; i++) b2[i] += b1[i];

    vll r = karatsubaMultiply(a2, b2);
    for (int i = 0; i < (int)a1b1.size(); i++) r[i] -= a1b1[i];
    for (int i = 0; i < (int)a2b2.size(); i++) r[i] -= a2b2[i];

    for (int i = 0; i < (int)r.size(); i++) res[i + k] += r[i];
    for (int i = 0; i < (int)a1b1.size(); i++) res[i] += a1b1[i];
    for (int i = 0; i < (int)a2b2.size(); i++) res[i + n] += a2b2[i];
    return res;
}

bigint operator*(const bigint &v) const {
    vector<int> a6 = convert_base(this->a, base_digits, 6);
    vector<int> b6 = convert_base(v.a, base_digits, 6);
    vll a(a6.begin(), a6.end());
    vll b(b6.begin(), b6.end());
    while (a.size() < b.size()) a.push_back(0);
    while (b.size() < a.size()) b.push_back(0);
    while (a.size() & (a.size() - 1)) a.push_back(0), b.push_back(0);
    vll c = karatsubaMultiply(a, b);
    bigint res;
    res.sign = sign * v.sign;
    for (int i = 0, carry = 0; i < (int)c.size(); i++) {
        long long cur = c[i] + carry;
        res.a.push_back((int)(cur % 1000000));
        carry = (int)(cur / 1000000);
    }
    res.a = convert_base(res.a, 6, base_digits);
    res.trim();
    return res;
}
};

```

3.2 Binary To Gray

```

string binToGray(string bin) {
    string gray(bin.size(), '0');
    int n = bin.size() - 1;
    gray[0] = bin[0];
    for (int i = 1; i <= n; i++) {
        gray[i] = '0' + (bin[i - 1] == '1') ^ (bin[i] == '1');
    }
    return gray;
}

```

3.3 Get Permutation Cicles

```

/*
 * receives a permutation [0, n-1]
 * returns a vector of cicles
 * for example: [ 1, 0, 3, 4, 2] -> [[0, 1], [2, 3, 4]]
 */

```

```

* */
vector<vll> getPermutationCicles(const vll &ps) {
    ll n = len(ps);
    vector<char> visited(n);
    vector<vll> cicles;
    for (int i = 0; i < n; ++i) {
        if (visited[i]) continue;

        vll cicle;
        ll pos = i;
        while (!visited[pos]) {
            cicle.pb(pos);
            visited[pos] = true;
            pos = ps[pos];
        }

        cicles.push_back(vll(all(cicle)));
    }
    return cicles;
}

```

4 Geometry

4.1 Point Template

```

const ld EPS = 1e-6;

typedef ld T;
bool eq(T a, T b) { return abs(a - b) <= EPS; }
struct point {
    T x, y;
    int id;
    point(T x = 0, T y = 0) : x(x), y(y) {}
    point operator+(const point &o) const { return {x + o.x, y + o.y}; }
    point operator-(const point &o) const { return {x - o.x, y - o.y}; }
    point operator*(T t) const { return {x * t, y * t}; }
    point operator/(T t) const { return {x / t, y / t}; }
    T operator*(const point &o) const {
        return x * o.x + y * o.y;
    } // dot product
    T operator^(const point &o) const {
        return x * o.y - y * o.x;
    } // cross product
};

ld dist(point a, point b) {
    point d = a - b;
    return sqrt(d * d);
}

```

5 Graphs

5.1 2 SAT (struct)

```

struct SAT2 {

```

```

    ll n;
    vll2d adj, adj_t;
    vc used;
    vll order, comp;
    vc assignment;
    bool solvable;
    SAT2(ll _n)
        : n(2 * _n),
          adj(n),
          adj_t(n),
          used(n),
          order(n),
          comp(n, -1),
          assignment(n / 2) {}
    void dfs1(int v) {
        used[v] = true;
        for (int u : adj[v]) {
            if (!used[u]) dfs1(u);
        }
        order.push_back(v);
    }

    void dfs2(int v, int c1) {
        comp[v] = c1;
        for (int u : adj_t[v]) {
            if (comp[u] == -1) dfs2(u, c1);
        }
    }

    bool solve_2SAT() {
        // find and label each SCC
        for (int i = 0; i < n; ++i) {
            if (!used[i]) dfs1(i);
        }
        reverse(all(order));
        ll j = 0;
        for (auto &v : order) {
            if (comp[v] == -1) dfs2(v, j++);
        }

        assignment.assign(n / 2, false);
        for (int i = 0; i < n; i += 2) {
            // x and !x belong to the same SCC
            if (comp[i] == comp[i + 1]) {
                solvable = false;
                return false;
            }

            assignment[i / 2] = comp[i] > comp[i + 1];
        }
        solvable = true;
        return true;
    }

    void add_disjunction(int a, bool na, int b, bool nb) {
        a = (2 * a) ^ na;
        b = (2 * b) ^ nb;
        int neg_a = a ^ 1;

```

```

    int neg_b = b ^ 1;
    adj[neg_a].push_back(b);
    adj[neg_b].push_back(a);
    adj_t[b].push_back(neg_a);
    adj_t[a].push_back(neg_b);
}
};

```

5.2 SCC (struct)

```

struct SCC {
    ll N;
    vll2d adj, tadj;
    vll todo, comps, comp;
    vector<set<ll>> sccadj;
    vchar vis;
    SCC(ll _N) : N(_N), adj(_N), tadj(_N), comp(_N, -1), sccadj(_N), vis(_N) {}

    void add_edge(ll x, ll y) { adj[x].eb(y), tadj[y].eb(x); }

    void dfs(ll x) {
        vis[x] = 1;
        for (auto &y : adj[x])
            if (!vis[y]) dfs(y);
        todo.pb(x);
    }
    void dfs2(ll x, ll v) {
        comp[x] = v;
        for (auto &y : tadj[x])
            if (comp[y] == -1) dfs2(y, v);
    }
    void gen() {
        for (ll i = 0; i < N; ++i)
            if (!vis[i]) dfs(i);
        reverse(all(todo));
        for (auto &x : todo)
            if (comp[x] == -1) {
                dfs2(x, x);
                comps.pb(x);
            }
    }

    void genSCCGraph() {
        for (ll i = 0; i < N; ++i) {
            for (auto &j : adj[i]) {
                if (comp[i] != comp[j]) {
                    sccadj[comp[i]].insert(comp[j]);
                }
            }
        }
    }
};

```

5.3 SCC Nodes (kosajaru)

```

/*
 * O(n+m)

```

```

* Returns a pair <a, b>
*     a: number of SCCs
*     b: vector of size n, where b[i] is the SCC id of node i
* */
void dfs(ll u, vchar &visited, const vll2d &g, vll &scc, bool buildScc, ll id,
        vll &sccid) {
    visited[u] = true;
    sccid[u] = id;
    for (auto &v : g[u])
        if (!visited[v]) dfs(v, visited, g, scc, buildScc, id, sccid);

    // if it's the first pass, add the node to the scc
    if (buildScc) scc.eb(u);
}

pair<ll, vll> kosajaru(vll2d &g) {
    ll n = len(g);
    vll scc;
    vchar vis(n);
    vll sccid(n);
    for (ll i = 0; i < n; i++)
        if (!vis[i]) dfs(i, vis, g, scc, true, 0, sccid);

    // build the transposed graph
    vll2d gt(n);
    for (int i = 0; i < n; ++i)
        for (auto &v : g[i]) gt[v].eb(i);

    // run the dfs on the previous scc order
    ll id = 1;
    vis.assign(n, false);
    for (ll i = len(scc) - 1; i >= 0; i--)
        if (!vis[scc[i]]) {
            dfs(scc[i], vis, gt, scc, false, id++, sccid);
        }
    return {id - 1, sccid};
}

```

5.4 Bellman Ford

```

bool bellman_ford(const vector<vector<pair<int, ll>>> &g, int s,
                 vector<ll> &dist) {
    int n = (int)g.size();
    dist.assign(n, LLONG_MAX);

    vector<int> count(n);
    vector<char> in_queue(n);
    queue<int> q;

    dist[s] = 0;
    q.push(s);
    in_queue[s] = true;

    while (not q.empty()) {
        int cur = q.front();
        q.pop();
        in_queue[cur] = false;

```

```

    for (auto [to, w] : g[cur]) {
        if (dist[cur] + w < dist[to]) {
            dist[to] = dist[cur] + w;
            if (not in_queue[to]) {
                q.push(to);
                in_queue[to] = true;
                count[to]++;
                if (count[to] > n) return false;
            }
        }
    }
}

return true;
}

```

5.5 Check Bipartite

```

// O(V)
bool checkBipartite(const ll n, const vector<vll> &adj) {
    ll s = 0;
    queue<ll> q;
    q.push(s);
    vll color(n, INF);
    color[s] = 0;
    bool isBipartite = true;
    while (!q.empty() && isBipartite) {
        ll u = q.front();
        q.pop();
        for (auto &v : adj[u]) {
            if (color[v] == INF) {
                color[v] = 1 - color[u];
                q.push(v);
            } else if (color[v] == color[u]) {
                return false;
            }
        }
    }
    return true;
}

```

5.6 Count SCC (kosajaru)

```

void dfs(ll u, vchar &visited, const vll2d &g, vll &scc, bool buildScc) {
    visited[u] = true;
    for (auto &v : g[u])
        if (!visited[v]) dfs(v, visited, g, scc, buildScc);

    // if it's the first pass, add the node to the scc
    if (buildScc) scc.eb(u);
}

ll kosajaru(vll2d &g) {
    ll n = len(g);
    vll scc;
    vchar vis(n);
    for (ll i = 0; i < n; i++)
        if (!vis[i]) dfs(i, vis, g, scc, true);
}

```

```

// build the transposed graph
vll2d gt(n);
for (int i = 0; i < n; ++i)
    for (auto &v : g[i]) gt[v].eb(i);

// run the dfs on the previous scc order
ll scccnt = 0;
vis.assign(n, false);
for (ll i = len(scc) - 1; i >= 0; i--)
    if (!vis[scc[i]]) dfs(scc[i], vis, gt, scc, false), scccnt++;
return scccnt;
}

```

5.7 Dijkstra

```

ll __inf = LLONG_MAX >> 5;
vll dijkstra(const vector<vector<pll>> &g, ll n) {
    priority_queue<pll, vector<pll>, greater<pll>> pq;
    vll dist(n, __inf);
    vector<char> vis(n);
    pq.emplace(0, 0);
    dist[0] = 0;
    while (!pq.empty()) {
        auto [d1, v] = pq.top();
        pq.pop();
        if (vis[v]) continue;
        vis[v] = true;

        for (auto [d2, u] : g[v]) {
            if (dist[u] > d1 + d2) {
                dist[u] = d1 + d2;
                pq.emplace(dist[u], u);
            }
        }
    }
    return dist;
}

```

5.8 Floyd Warshall

```

vector<vll> floyd_warshall(const vector<vll> &adj, ll n) {
    auto dist = adj;

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            for (int k = 0; k < n; ++k) {
                dist[j][k] = min(dist[j][k], dist[j][i] + dist[i][k]);
            }
        }
    }
    return dist;
}

```

5.9 Kruskal (Python)

```

class DSU:

```

```

def __init__(self, n):
    self.n = n
    self.p = [x for x in range(0, n + 1)]
    self.size = [0 for i in range(0, n + 1)]

def find_set(self, x):
    if self.p[x] == x:
        return x
    else:
        self.p[x] = self.find_set(self.p[x])
        return self.p[x]

def same_set(self, x, y):
    return bool(self.find_set(x) == self.find_set(y))

def union_set(self, x, y):
    px = self.find_set(x)
    py = self.find_set(y)

    if px == py:
        return

    size_x = self.size[px]
    size_y = self.size[py]

    if size_x > size_y:
        self.p[py] = self.p[px]
        self.size[px] += self.size[py]
    else:
        self.p[px] = self.p[py]
        self.size[py] += self.size[px]

def kruskal(gv, n):
    """
    Receives the list of edges as a list of tuple in the form:
        d, u, v
        d: distance between u and v
    And also n as the total of vertices.
    """
    dsu = DSU(n)

    c = 0
    for e in gv:
        d, u, v = e
        if not dsu.same_set(u, v):
            c += d
            dsu.union_set(u, v)

    return c

```

5.10 Lowest Common Ancestor Sparse Table

```

int fastlog2(ll x) {
    ull i = x;
    return i ? __builtin_clzll(1) - __builtin_clzll(i) : -1;
}
template <typename T>

```

```

class SparseTable {
public:
    int N;
    int K;
    vector<vector<T>> st;
    SparseTable(vector<T> vs)
        : N((int)vs.size()), K(fastlog2(N) + 1), st(K + 1, vector<T>(N + 1)) {
        copy(vs.begin(), vs.end(), st[0].begin());

        for (int i = 1; i <= K; ++i)
            for (int j = 0; j + (1 << i) <= N; ++j)
                st[i][j] = min(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
    }
    SparseTable() {}
    T RMQ(int l, int r) {
        int i = fastlog2(r - l + 1);
        return min(st[i][l], st[i][r - (1 << i) + 1]);
    }
};

class LCA {
public:
    int p;
    int n;
    vi first;
    vector<char> visited;
    vi vertices;
    vi height;
    SparseTable<int> st;

    LCA(const vector<vi> &g)
        : p(0), n((int)g.size()), first(n + 1), visited(n + 1, 0), height(n + 1) {
        build_dfs(g, 1, 1);
        st = SparseTable<int>(vertices);
    }

    void build_dfs(const vector<vi> &g, int u, int hi) {
        visited[u] = true;
        height[u] = hi;
        first[u] = vertices.size();
        vertices.push_back(u);
        for (auto uv : g[u]) {
            if (!visited[uv]) {
                build_dfs(g, uv, hi + 1);
                vertices.push_back(uv);
            }
        }
    }

    int lca(int a, int b) {
        int l = min(first[a], first[b]);
        int r = max(first[a], first[b]);
        return st.RMQ(l, r);
    }
};

```

5.11 Topological Sorting

/*

```

* O(V)
* assumes:
*     * vertices have index [0, n-1]
* if is a DAG:
*     * returns a topological sorting
* else:
*     * returns an empty vector
* */
enum class state { not_visited, processing, done };
bool dfs(const vector<vll> &adj, ll s, vector<state> &states, vll &order) {
    states[s] = state::processing;
    for (auto &v : adj[s]) {
        if (states[v] == state::not_visited) {
            if (not dfs(adj, v, states, order)) return false;
        } else if (states[v] == state::processing)
            return false;
    }
    states[s] = state::done;
    order.pb(s);
    return true;
}
vll topologicalSorting(const vector<vll> &adj) {
    ll n = len(adj);
    vll order;
    vector<state> states(n, state::not_visited);
    for (int i = 0; i < n; ++i) {
        if (states[i] == state::not_visited) {
            if (not dfs(adj, i, states, order)) return {};
        }
    }
    reverse(all(order));
    return order;
}

```

6 Math

6.1 Arithmetic Progression Sum

```

/*
* s: first term
* d: common difference
* n: number of terms
*/
ll arithmeticProgressionSum(ll s, ll d, ll n) {
    return (s + (s + d * (n - 1))) * n / 2ll;
}

```

6.2 Combinatorics With Repetitions

```

void combinations_with_repetition(int n, int k,
                                   function<void(const vector<int> &)> process)
{
    vector<int> v(k, 1);
    int pos = k - 1;

    while (true) {
        process(v);

```

```

        v[pos]++;

        while (pos > 0 and v[pos] > n) {
            --pos;
            v[pos]++;
        }

        if (pos == 0 and v[pos] > n) break;

        for (int i = pos + 1; i < k; ++i) v[i] = v[pos];

        pos = k - 1;
    }
}

```

6.3 Count Divisors Memo

```

const ll mod = 1073741824;
const ll maxd = 100 * 100 * 100 + 1;
vector<ll> memo(maxd, -1);
ll countdivisors(ll x) {
    ll ox = x;
    ll ans = 1;
    for (ll i = 2; i <= x; ++i) {
        if (memo[x] != -1) {
            ans *= memo[x];
            break;
        }
        ll count = 0;
        while (x and x % i == 0) {
            x /= i;
            count++;
        }
        ans *= (count + 1);
    }
    memo[ox] = ans;
    return ans;
}

```

6.4 Euler Phi

```

const ll MAXN = 1e5;
vll list_primes(ll n) { // Nlog * log N
    vll ps;
    bitset<MAXN> sieve;
    sieve.set();
    sieve.reset(1);
    for (ll i = 2; i <= n; ++i) {
        if (sieve[i]) ps.push_back(i);
        for (ll j = i * 2; j <= n; j += i) {
            sieve.reset(j);
        }
    }
    return ps;
}

vector<pll> factorization(ll n, const vll &primes) {

```

```

vector<pll> ans;
for (auto &p : primes) {
    if (n == 1) break;
    ll cnt = 0;
    while (n % p == 0) {
        cnt++;
        n /= p;
    }
    if (cnt) ans.emplace_back(p, cnt);
}
return ans;
}

ll phi(ll n, vector<pll> factors) {
    if (n == 1) return 1;
    ll ans = n;

    for (auto [p, k] : factors) {
        ans /= p;
        ans *= (p - 1);
    }

    return ans;
}

```

6.5 Factorial Factorization

```

// O(logN) greater k that p^k | n
ll E(ll n, ll p) {
    ll k = 0, b = p;
    while (b <= n) {
        k += n / b;
        b *= p;
    }
    return k;
}

// list every prime until MAXN O(Nlog * log N)
const ll MAXN = 1e5;
vll list_primes(ll n) {
    vll ps;
    bitset<MAXN> sieve;
    sieve.set();
    sieve.reset(1);
    for (ll i = 2; i <= n; ++i) {
        if (sieve[i]) ps.push_back(i);
        for (ll j = i * 2; j <= n; j += i) sieve.reset(j);
    }
    return ps;
}

// O(pi(N)*logN)
map<ll, ll> factorial_factorization(ll n, const vll &primes) {
    map<ll, ll> fs;
    for (const auto &p : primes) {
        if (p > n) break;
        fs[p] = E(n, p);
    }
}

```

```

    return fs;
}

```

6.6 Factorial

```

const ll MAX = 18;
vll fv(MAX, -1);
ll factorial(ll n) {
    if (fv[n] != -1) return fv[n];
    if (n == 0) return 1;
    return n * factorial(n - 1);
}

```

6.7 Factorization (Pollard who)

```

/*
 * Factorizes a number into its prime factors.
 * time: O(n^(1/4) * log(n))
 * memory: just to store the prime factors
 */

ll mul(ll a, ll b, ll m) {
    ll ret = a * b - (ll)((ld)1 / m * a * b + 0.5) * m;
    return ret < 0 ? ret + m : ret;
}

ll pow(ll a, ll b, ll m) {
    ll ans = 1;
    for (; b > 0; b /= 2ll, a = mul(a, a, m)) {
        if (b % 2ll == 1) ans = mul(ans, a, m);
    }
    return ans;
}

bool prime(ll n) {
    if (n < 2) return 0;
    if (n <= 3) return 1;
    if (n % 2 == 0) return 0;

    ll r = __builtin_ctzll(n - 1), d = n >> r;
    for (int a : {2, 325, 9375, 28178, 450775, 9780504, 795265022}) {
        ll 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;
}

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

    ll 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);
}

vll fact(ll n) {
    if (n == 1) return {};
    if (prime(n)) return {n};
    ll d = rho(n);
    vll l = fact(d), r = fact(n / d);
    l.insert(l.end(), r.begin(), r.end());
    return l;
}

```

6.8 Factorization With Primes

```

// Nlog * log N
const ll MAXN = 1e5;
vll list_primes(ll n) {
    vll ps;
    bitset<MAXN> sieve;
    sieve.set();
    sieve.reset(1);
    for (ll i = 2; i <= n; ++i) {
        if (sieve[i]) ps.push_back(i);
        for (ll j = i * 2; j <= n; j += i) sieve.reset(j);
    }
    return ps;
}

// O(pi(sqrt(n)))
map<ll, ll> factorization(ll n, const vll &primes) {
    map<ll, ll> ans;
    for (auto p : primes) {
        if (p * p > n) break;
        ll count = 0;
        for (; n % p == 0; count++, n /= p)
            ;
        if (count) ans[p] = count;
    }
    return ans;
}

```

6.9 Factorization

```

// O(sqrt(n))
map<ll, ll> factorization(ll n) {
    map<ll, ll> ans;
    for (ll i = 2; i * i <= n; i++) {
        ll count = 0;
        for (; n % i == 0; count++, n /= i)
            ;
        if (count) ans[i] = count;
    }
}

```

```

if (n > 1) ans[n]++;
return ans;
}

```

6.10 Fast Fourier Transform

```

template <bool invert = false>
void fft(vector<complex<double>>& xs) {
    int N = (int)xs.size();

    if (N == 1) return;

    vector<complex<double>> es(N / 2), os(N / 2);

    for (int i = 0; i < N / 2; ++i) es[i] = xs[2 * i];

    for (int i = 0; i < N / 2; ++i) os[i] = xs[2 * i + 1];

    fft<invert>(es);
    fft<invert>(os);

    auto signal = (invert ? 1 : -1);
    auto theta = 2 * signal * acos(-1) / N;
    complex<double> S{1}, S1{cos(theta), sin(theta)};

    for (int i = 0; i < N / 2; ++i) {
        xs[i] = (es[i] + S * os[i]);
        xs[i] /= (invert ? 2 : 1);

        xs[i + N / 2] = (es[i] - S * os[i]);
        xs[i + N / 2] /= (invert ? 2 : 1);

        S *= S1;
    }
}

```

6.11 Fast Exp

```

/*
Fast exponentiation algorithm,
compute a^n in O(log(n))
*/
ll fexp(ll a, int n) {
    if (n == 0) return 1;
    if (n == 1) return a;
    ll x = fexp(a, n / 2);
    return x * x * (n & 1 ? a : 1);
}

```

6.12 Gauss Elimination

```

template <size_t Dim>
struct GaussianElimination {
    vector<ll> basis;
    size_t size;

    GaussianElimination() : basis(Dim + 1), size(0) {}
}

```



```

void insert(ll x) {
    for (ll i = Dim; i >= 0; i--) {
        if ((x & 1ll << i) == 0) continue;

        if (!basis[i]) {
            basis[i] = x;
            size++;
            break;
        }

        x ^= basis[i];
    }
}

void normalize() {
    for (ll i = Dim; i >= 0; i--)
        for (ll j = i - 1; j >= 0; j--)
            if (basis[i] & 1ll << j) basis[i] ^= basis[j];
}

bool check(ll x) {
    for (ll i = Dim; i >= 0; i--) {
        if ((x & 1ll << i) == 0) continue;

        if (!basis[i]) return false;

        x ^= basis[i];
    }

    return true;
}

auto operator[](ll k) { return at(k); }

ll at(ll k) {
    ll ans = 0;
    ll total = 1ll << size;
    for (ll i = Dim; ~i; i--) {
        if (!basis[i]) continue;

        ll mid = total >> 1ll;
        if ((mid < k and (ans & 1ll << i) == 0) ||
            (k <= mid and (ans & 1ll << i)))
            ans ^= basis[i];

        if (mid < k) k -= mid;

        total >>= 1ll;
    }
    return ans;
}

ll at_normalized(ll k) {
    ll ans = 0;
    k--;
    for (size_t i = 0; i <= Dim; i++) {
        if (!basis[i]) continue;

```

```

        if (k & 1) ans ^= basis[i];
        k >>= 1;
    }
    return ans;
}
};

```

6.13 Gcd Using Factorization

```

// O(sqrt(n))
map<ll, ll> factorization(ll n) {
    map<ll, ll> ans;
    for (ll i = 2; i * i <= n; i++) {
        ll count = 0;
        for (; n % i == 0; count++, n /= i)
            ;
        if (count) ans[i] = count;
    }
    if (n > 1) ans[n]++;
    return ans;
}

ll gcd_with_factorization(ll a, ll b) {
    map<ll, ll> fa = factorization(a);
    map<ll, ll> fb = factorization(b);
    ll ans = 1;
    for (auto fai : fa) {
        ll k = min(fai.second, fb[fai.first]);
        while (k--) ans *= fai.first;
    }
    return ans;
}

```

6.14 Gcd

```

ll gcd(ll a, ll b) { return b ? gcd(b, a % b) : a; }

```

6.15 Integer Mod

```

const ll INF = 1e18;
const ll mod = 998244353;
template <ll MOD = mod>
struct Modular {
    ll value;
    static const ll MOD_value = MOD;

    Modular(ll v = 0) {
        value = v % MOD;
        if (value < 0) value += MOD;
    }

    Modular(ll a, ll b) : value(0) {
        *this += a;
        *this /= b;
    }

    Modular& operator+=(Modular const& b) {
        value += b.value;

```

```

    if (value >= MOD) value -= MOD;
    return *this;
}
Modular& operator--(Modular const& b) {
    value -= b.value;
    if (value < 0) value += MOD;
    return *this;
}
Modular& operator*=(Modular const& b) {
    value = (ll)value * b.value % MOD;
    return *this;
}

friend Modular mexp(Modular a, ll e) {
    Modular res = 1;
    while (e) {
        if (e & 1) res *= a;
        a *= a;
        e >>= 1;
    }
    return res;
}
friend Modular inverse(Modular a) { return mexp(a, MOD - 2); }

Modular& operator/=(Modular const& b) { return *this *= inverse(b); }
friend Modular operator+(Modular a, Modular const b) { return a += b; }
Modular operator++(int) { return this->value = (this->value + 1) % MOD; }
Modular operator++() { return this->value = (this->value + 1) % MOD; }
friend Modular operator-(Modular a, Modular const b) { return a -= b; }
friend Modular operator-(Modular const a) { return 0 - a; }
Modular operator--(int) {
    return this->value = (this->value - 1 + MOD) % MOD;
}

Modular operator--() { return this->value = (this->value - 1 + MOD) % MOD; }
friend Modular operator*(Modular a, Modular const b) { return a *= b; }
friend Modular operator/(Modular a, Modular const b) { return a /= b; }
friend std::ostream& operator<<(std::ostream& os, Modular const& a) {
    return os << a.value;
}
}
friend bool operator==(Modular const& a, Modular const& b) {
    return a.value == b.value;
}
}
friend bool operator!=(Modular const& a, Modular const& b) {
    return a.value != b.value;
}
}
};

```

6.16 Is Prime

```

bool isprime(ll n) { // 0(sqrt(n))
    if (n < 2) return false;
    if (n == 2) return true;
    if (n % 2 == 0) return false;
    for (ll i = 3; i * i < n; i += 2)
        if (n % i == 0) return false;
    return true;
}

```

6.17 Lcm Using Factorization

```

map<ll, ll> factorization(ll n) {
    map<ll, ll> ans;
    for (ll i = 2; i * i <= n; i++) {
        ll count = 0;
        for (; n % i == 0; count++, n /= i)
            ;
        if (count) ans[i] = count;
    }
    if (n > 1) ans[n]++;
    return ans;
}

ll lcm_with_factorization(ll a, ll b) {
    map<ll, ll> fa = factorization(a);
    map<ll, ll> fb = factorization(b);
    ll ans = 1;
    for (auto fai : fa) {
        ll k = max(fai.second, fb[fai.first]);
        while (k--) ans *= fai.first;
    }
    return ans;
}

```

6.18 Lcm

```

ll gcd(ll a, ll b) { return b ? gcd(b, a % b) : a; }
ll lcm(ll a, ll b) { return a / gcd(a, b) * b; }

```

6.19 Modular Inverse Using Phi

```

map<ll, ll> factorization(ll n) {
    map<ll, ll> ans;
    for (ll i = 2; i * i <= n; i++) {
        ll count = 0;
        for (; n % i == 0; count++, n /= i)
            ;
        if (count) ans[i] = count;
    }
    if (n > 1) ans[n]++;
    return ans;
}

ll phi(ll n) {
    if (n == 1) return 1;

    auto fs = factorization(n);
    auto res = n;

    for (auto [p, k] : fs) {
        res /= p;
        res *= (p - 1);
    }

    return res;
}

```

```

ll fexp(ll a, ll n, ll mod) {
    if (n == 0) return 1;
    if (n == 1) return a;
    ll x = fexp(a, n / 2, mod);
    return x * x * (n & 1 ? a : 1) % mod;
}

ll inv(ll a, ll mod) { return fexp(a, phi(mod) - 1, mod); }

```

6.20 N Choose K Count

```

/*
 * O(nm) time, O(m) space
 * equal to n choose k
 * */
ll binom(ll n, ll k) {
    if (k > n) return 0;
    vll dp(k + 1, 0);
    dp[0] = 1;
    for (ll i = 1; i <= n; i++)
        for (ll j = k; j > 0; j--) dp[j] = dp[j] + dp[j - 1];
    return dp[k];
}

```

6.21 Permutation Count

```

const ll MAX = 18;
vll fv(MAX, -1);
ll factorial(ll n) {
    if (fv[n] != -1) return fv[n];
    if (n == 0) return 1;
    return n * factorial(n - 1);
}

```

```

template <typename T>
ll permutation_count(vector<T> xs) {
    map<T, ll> h;
    for (auto xi : xs) h[xi]++;
    ll ans = factorial((ll)xs.size());
    dbg(ans);
    for (auto [v, cnt] : h) {
        dbg(cnt);
        ans /= cnt;
    }

    return ans;
}

```

6.22 Polynomial

```

using polynomial = vector<ll>;
int degree(const polynomial &xs) { return xs.size() - 1; }
ll horner_evaluate(const polynomial &xs, ll x) {
    ll ans = 0;
    ll n = degree(xs);
    for (int i = n; i >= 0; --i) {

```

```

        ans *= x;
        ans += xs[i];
    }
    return ans;
}

polynomial operator+(const polynomial &a, const polynomial &b) {
    int n = degree(a);
    int m = degree(b);
    polynomial r(max(n, m) + 1, 0);

    for (int i = 0; i <= n; ++i) r[i] += a[i];
    for (int j = 0; j <= m; ++j) r[j] += b[j];
    while (!r.empty() and r.back() == 0) r.pop_back();
    if (r.empty()) r.push_back(0);
    return r;
}

polynomial operator*(const polynomial &p, const polynomial &q) {
    int n = degree(p);
    int m = degree(q);
    polynomial r(n + m + 1, 0);
    for (int i = 0; i <= n; ++i)
        for (int j = 0; j <= m; ++j) r[i + j] += (p[i] * q[j]);
    return r;
}

```

6.23 Power Sum

```

// calculates K^0 + K^1 ... + K^n
ll fastpow(ll a, int n) {
    if (n == 1) return a;
    ll x = fastpow(a, n / 2);
    return x * x * (n & 1 ? a : 1);
}

ll powersum(ll n, ll k) { return (fastpow(n, k + 1) - 1) / (n - 1); }

```

6.24 Sieve List Primes

```

// lsit every prime until MAXN
const ll MAXN = 1e5;
vll list_primes(ll n) { // Nlog * log N
    vll ps;
    bitset<MAXN> sieve;
    sieve.set();
    sieve.reset(1);
    for (ll i = 2; i <= n; ++i) {
        if (sieve[i]) ps.push_back(i);
        for (ll j = i * 2; j <= n; j += i) {
            sieve.reset(j);
        }
    }
    return ps;
}

```

7 Searching

7.1 Ternary Search Recursive

```

const double eps = 1e-6;

// IT MUST BE AN UNIMODAL FUNCTION
double f(int x) { return x * x + 2 * x + 4; }

double ternary_search(double l, double r) {
    if (fabs(f(l) - f(r)) < eps) return f((l + (r - l) / 2.0));

    auto third = (r - l) / 3.0;
    auto m1 = l + third;
    auto m2 = r - third;

    // change the signal to find the maximum point.
    return m1 < m2 ? ternary_search(m1, r) : ternary_search(l, m2);
}

```

8 Strings

8.1 Hash Range Query

```

struct Hash {
    const ll P = 31;
    int n;
    string s;
    vll h, hi, p;
    Hash() {}
    Hash(string s) : s(s), n(s.size()), h(n), hi(n), p(n) {
        for (int i = 0; i < n; i++) p[i] = (i ? P * p[i - 1] : 1) % MOD;
        for (int i = 0; i < n; i++) h[i] = (s[i] + (i ? h[i - 1] : 0) * P) % MOD;
        for (int i = n - 1; i >= 0; i--)
            hi[i] = (s[i] + (i + 1 < n ? hi[i + 1] : 0) * P) % MOD;
    }
    ll query(int l, int r) {
        ll hash = (h[r] - (l ? h[l - 1] * p[r - l + 1] % MOD : 0));
        return hash < 0 ? hash + MOD : hash;
    }
    ll query_inv(int l, int r) {
        ll hash = (hi[l] - (r + 1 < n ? hi[r + 1] * p[r - l + 1] % MOD : 0));
        return hash < 0 ? hash + MOD : hash;
    }
};

```

8.2 Longest Palindrome

```

string longest_palindrome(const string &s) {
    int n = (int)s.size();
    vector<array<int, 2>> dp(n);

    pii odd(0, -1), even(0, -1);
    pii ans;
    for (int i = 0; i < n; i++) {
        int k = 0;
        if (i > odd.second)
            k = 1;
        else
            k = min(dp[odd.first + odd.second - i][0], odd.second - i + 1);
        while (i - k >= 0 and i + k < n and s[i - k] == s[i + k]) k++;
    }
}

```

```

dp[i][0] = k--;
if (i + k > odd.second) odd = {i - k, i + k};
if (2 * dp[i][0] - 1 > ans.second) ans = {i - k, 2 * dp[i][0] - 1};

k = 0;
if (i <= even.second)
    k = min(dp[even.first + even.second - i + 1][1], even.second - i + 1);
while (i - k - 1 >= 0 and i + k < n and s[i - k - 1] == s[i + k]) k++;
dp[i][1] = k--;
if (i + k > even.second) even = {i - k - 1, i + k};
if (2 * dp[i][1] > ans.second) ans = {i - k - 1, 2 * dp[i][1]};
}
return s.substr(ans.first, ans.second);
}

```

8.3 Rabin Karp

```

size_t rabin_karp(const string &s, const string &p) {
    if (s.size() < p.size()) return 0;

    auto n = s.size(), m = p.size();
    const ll p1 = 31, p2 = 29, q1 = 1e9 + 7, q2 = 1e9 + 9;
    const ll p1_1 = fpow(p1, q1 - 2, q1), p1_2 = fpow(p1, m - 1, q1);
    const ll p2_1 = fpow(p2, q2 - 2, q2), p2_2 = fpow(p2, m - 1, q2);

    pair<ll, ll> hs, hp;
    for (int i = (int)m - 1; ~i; --i) {
        hs.first = (hs.first * p1) % q1;
        hs.first = (hs.first + (s[i] - 'a' + 1)) % q1;
        hs.second = (hs.second * p2) % q2;
        hs.second = (hs.second + (s[i] - 'a' + 1)) % q2;

        hp.first = (hp.first * p1) % q1;
        hp.first = (hp.first + (p[i] - 'a' + 1)) % q1;
        hp.second = (hp.second * p2) % q2;
        hp.second = (hp.second + (p[i] - 'a' + 1)) % q2;
    }

    size_t occ = 0;
    for (size_t i = 0; i < n - m; i++) {
        occ += (hs == hp);

        int fi = s[i] - 'a' + 1;
        int fm = p[i + m] - 'a' + 1;

        hs.first = (hs.first - fi + q1) % q1;
        hs.first = (hs.first * p1_1) % q1;
        hs.first = (hs.first + fm * p1_2) % q1;
        hs.second = (hs.second - fi + q2) % q2;
        hs.second = (hs.second * p2_1) % q2;
        hs.second = (hs.second + fm * p2_2) % q2;
    }
    occ += hs == hp;

    return occ;
}

```

8.4 String Psum

```
struct strPsum {
    ll n;
    ll k;
    vector<vll> psum;
    strPsum(const string &s) : n(s.size()), k(100), psum(k, vll(n + 1)) {
        for (ll i = 1; i <= n; ++i) {
            for (ll j = 0; j < k; ++j) {
                psum[j][i] = psum[j][i - 1];
            }
            psum[s[i - 1]][i]++;
        }
    }

    ll qtd(ll l, ll r, char c) { // [0,n-1]
        return psum[c][r + 1] - psum[c][l];
    }
}
```

8.5 Suffix Automaton (complete)

```
struct state {
    int len, link, cnt, firstpos;
    // this can be optimized using a vector with the alphabet size
    map<char, int> next;
    vi inv_link;
};

struct SuffixAutomaton {
    vector<state> st;
    int sz = 0;
    int last;
    vc cloned;

    SuffixAutomaton(const string &s, int maxlen)
        : st(maxlen * 2), cloned(maxlen * 2) {
        st[0].len = 0;
        st[0].link = -1;
        sz++;
        last = 0;
        for (auto &c : s) add_char(c);

        // precompute for count occurrences
        for (int i = 1; i < sz; i++) {
            st[i].cnt = !cloned[i];
        }
        vector<pair<state, int>> aux;
        for (int i = 0; i < sz; i++) {
            aux.push_back({st[i], i});
        }

        sort(all(aux), [](const pair<state, int> &a, const pair<state, int> &b) {
            return a.fst.len > b.fst.len;
        });

        for (auto &[stt, id] : aux) {
            if (stt.link != -1) {
                st[stt.link].cnt += st[id].cnt;
            }
        }
    }
}
```

```
    }
}

// for find every occurende position
for (int v = 1; v < sz; v++) {
    st[st[v].link].inv_link.push_back(v);
}

}

void add_char(char c) {
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    st[cur].firstpos = st[cur].len - 1;
    int p = last;
    // follow the suffix link until find a transition to c
    while (p != -1 and !st[p].next.count(c)) {
        st[p].next[c] = cur;
        p = st[p].link;
    }
    // there was no transition to c so create and leave
    if (p == -1) {
        st[cur].link = 0;
        last = cur;
        return;
    }

    int q = st[p].next[c];
    if (st[p].len + 1 == st[q].len) {
        st[cur].link = q;
    } else {
        int clone = sz++;
        cloned[clone] = true;
        st[clone].len = st[p].len + 1;
        st[clone].next = st[q].next;
        st[clone].link = st[q].link;
        st[clone].firstpos = st[q].firstpos;
        while (p != -1 and st[p].next[c] == q) {
            st[p].next[c] = clone;
            p = st[p].link;
        }
        st[q].link = st[cur].link = clone;
    }
    last = cur;
}

bool checkOccurrence(const string &t) { // 0(len(t))
    int cur = 0;
    for (auto &c : t) {
        if (!st[cur].next.count(c)) return false;
        cur = st[cur].next[c];
    }
    return true;
}

ll totalSubstrings() { // distinct, 0(len(s))
    ll tot = 0;
    for (int i = 1; i < sz; i++) {
        tot += st[i].len - st[st[i].link].len;
    }
}
```

```

    return tot;
}

// count occurrences of a given string t
int countOccurrences(const string &t) {
    int cur = 0;
    for (auto &c : t) {
        if (!st[cur].next.count(c)) return 0;
        cur = st[cur].next[c];
    }
    return st[cur].cnt;
}

// find the first index where t appears a substring 0(len(t))
int firstOccurrence(const string &t) {
    int cur = 0;
    for (auto c : t) {
        if (!st[cur].next.count(c)) return -1;
        cur = st[cur].next[c];
    }
    return st[cur].firstpos - len(t) + 1;
}

vi everyOccurrence(const string &t) {
    int cur = 0;
    for (auto c : t) {
        if (!st[cur].next.count(c)) return {};
        cur = st[cur].next[c];
    }
    vi ans;
    getEveryOccurrence(cur, len(t), ans);
    return ans;
}

void getEveryOccurrence(int v, int P_length, vi &ans) {
    if (!cloned[v]) ans.pb(st[v].firstpos - P_length + 1);
    for (int u : st[v].inv_link) getEveryOccurrence(u, P_length, ans);
}

};

```

8.6 Trie Naive

```

// time: O(n^2) memory: O(n^2)
using Node = map<char, int>;
using vi = vector<int>;
using Trie = vector<Node>;

```

```

Trie build(const string &s) {
    int n = (int)s.size();
    Trie trie(1);
    string suffix;

    for (int i = n - 1; i >= 0; --i) {
        suffix = s.substr(i) + '#';

        int v = 0; // root
        for (auto c : suffix) {
            if (c == '#') { // makrs the poistion of an occurrence

```

```

                trie[v][c] = i;
                break;
            }
            if (trie[v][c])
                v = trie[v][c];
            else {
                trie.push_back({});
                trie[v][c] = trie.size() - 1;
                v = trie.size() - 1;
            }
        }
    }
    return trie;
}

vi search(Trie &trie, string s) {
    int p = 0;
    vi occ;
    for (auto &c : s) {
        p = trie[p][c];
        if (!p) return occ;
    }

    queue<int> q;
    q.push(0);
    while (!q.empty()) {
        auto cur = q.front();
        q.pop();
        for (auto [c, v] : trie[cur]) {
            if (c == '#')
                occ.push_back(v);
            else
                q.push(v);
        }
    }
    return occ;
}

ll distinct_substr(const Trie &trie) {
    ll cnt = 0;
    queue<int> q;
    q.push(0);
    while (!q.empty()) {
        auto u = q.front();
        q.pop();

        for (auto [c, v] : trie[u]) {
            if (c != '#') {
                cnt++;
                q.push(v);
            }
        }
    }
    return cnt;
}

```

8.7 Z Function Get Occurence Positions

```

/*
 * ans[i] = a position where p matchs
 * with s perfectly starting
 * 0(len(s)+len(p))
 * */
vi getOccPos(string &s, string &p) {
    // Z-function
    char delim = '#';
    string t{p + delim + s};
    vi zs(len(t));

    int l = 0, r = 0;
    for (int i = 1; i < len(t); i++) {
        if (i <= r) zs[i] = min(zs[i - 1], r - i + 1);
        while (zs[i] + i < len(t) and t[zs[i]] == t[i + zs[i]]) zs[i]++;
        if (r < i + zs[i] - 1) l = i, r = i + zs[i] - 1;
    }

    // Iterate over the results of Z-function to get ranges
    vi ans;
    int start = len(p) + 1 + 1 - 1;
    for (int i = start; i < len(zs); i++) {
        if (zs[i] == len(p)) {
            int l = i - start;
            ans.emplace_back(l);
        }
    }
    return ans;
}

```

9 Trees

9.1 Binary Lifting

```

/*
 * far[h][i] = the node that 2^h far from node i
 * sometimes is useful invert the order of loops
 * time : O(nlogn)
 * */
const int maxlog = 20;
int far[maxlog + 1][n + 1];
int n;
for (int h = 1; h <= maxlog; h++) {
    for (int i = 1; i <= n; i++) {
        far[h][i] = far[h - 1][far[h - 1][i]];
    }
}

```

9.2 Maximum Distances

```

/*
 * Returns the maximum distance from every node to any other node in the tree.
 * */
pll mostDistantFrom(const vector<vll> &adj, ll n, ll root) {
    // 0 indexed
    ll mostDistantNode = root;
    ll nodeDistance = 0;

```

```

queue<pll> q;
vector<char> vis(n);
q.emplace(root, 0);
vis[root] = true;
while (!q.empty()) {
    auto [node, dist] = q.front();
    q.pop();
    if (dist > nodeDistance) {
        nodeDistance = dist;
        mostDistantNode = node;
    }
    for (auto u : adj[node]) {
        if (!vis[u]) {
            vis[u] = true;
            q.emplace(u, dist + 1);
        }
    }
}
return {mostDistantNode, nodeDistance};
}

```

```

ll twoNodesDist(const vector<vll> &adj, ll n, ll a, ll b) {
    queue<pll> q;
    vector<char> vis(n);
    q.emplace(a, 0);
    while (!q.empty()) {
        auto [node, dist] = q.front();
        q.pop();
        if (node == b) return dist;
        for (auto u : adj[node]) {
            if (!vis[u]) {
                vis[u] = true;
                q.emplace(u, dist + 1);
            }
        }
    }
    return -1;
}

```

```

tuple<ll, ll, ll> tree_diameter(const vector<vll> &adj, ll n) {
    // returns two points of the diameter and the diameter itself
    auto [node1, dist1] = mostDistantFrom(adj, n, 0);
    auto [node2, dist2] = mostDistantFrom(adj, n, node1);
    auto diameter = twoNodesDist(adj, n, node1, node2);
    return make_tuple(node1, node2, diameter);
}

```

```

vll everyDistanceFromNode(const vector<vll> &adj, ll n, ll root) {
    // Single Source Shortest Path, from a given root
    queue<pair<ll, ll>> q;
    vll ans(n, -1);
    ans[root] = 0;
    q.emplace(root, 0);
    while (!q.empty()) {
        auto [u, d] = q.front();
        q.pop();

        for (auto w : adj[u]) {

```

```

        if (ans[w] != -1) continue;
        ans[w] = d + 1;
        q.emplace(w, d + 1);
    }
}
return ans;
}

vll maxDistances(const vector<vll> &adj, ll n) {
    auto [node1, node2, diameter] = tree_diameter(adj, n);
    auto distances1 = everyDistanceFromNode(adj, n, node1);
    auto distances2 = everyDistanceFromNode(adj, n, node2);
    vll ans(n);
    for (int i = 0; i < n; ++i) ans[i] = max(distances1[i], distances2[i]);
    return ans;
}

```

9.3 Tree Diameter

```

pll mostDistantFrom(const vector<vll> &adj, ll n, ll root) {
    // 0 indexed
    ll mostDistantNode = root;
    ll nodeDistance = 0;
    queue<pll> q;
    vector<char> vis(n);
    q.emplace(root, 0);
    vis[root] = true;
    while (!q.empty()) {
        auto [node, dist] = q.front();
        q.pop();
        if (dist > nodeDistance) {
            nodeDistance = dist;
            mostDistantNode = node;
        }
        for (auto u : adj[node]) {
            if (!vis[u]) {
                vis[u] = true;
                q.emplace(u, dist + 1);
            }
        }
    }
    return {mostDistantNode, nodeDistance};
}

ll twoNodesDist(const vector<vll> &adj, ll n, ll a, ll b) {
    // 0 indexed
    queue<pll> q;
    vector<char> vis(n);
    q.emplace(a, 0);
    while (!q.empty()) {
        auto [node, dist] = q.front();
        q.pop();
        if (node == b) {
            return dist;
        }
        for (auto u : adj[node]) {
            if (!vis[u]) {
                vis[u] = true;
                q.emplace(u, dist + 1);
            }
        }
    }
}

```

```

    }
}
return -1;
}

ll tree_diameter(const vector<vll> &adj, ll n) {
    // 0 indexed !!!
    auto [node1, dist1] = mostDistantFrom(adj, n, 0);
    auto [node2, dist2] = mostDistantFrom(adj, n, node1);
    auto diameter = twoNodesDist(adj, n, node1, node2);
    return diameter;
}

```

10 Settings and macros

10.1 short-macro.cpp

```

#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
#define fastio \
    ios_base::sync_with_stdio(false); \
    cin.tie(0); \
    cout.tie(0);
#define len(_x) (int) _x.size()
using ll = long long;
using pii = pair<int, int>;
#define all(a) a.begin(), a.end()

void run() {}
int32_t main(void) {
    fastio;
    int t;
    t = 1;
    // cin >> t;
    while (t--) run();
}

```

10.2 .vimrc

```

set ts=4 sw=4 sta nu rnu sc cindent
set bg=dark ruler clipboard=unnamed,unnamedplus, timeoutlen=100
colorscheme default

nnoremap <C-j> :botright belowright term bash <CR>
syntax on

```

10.3 degug.cpp

```

#include <bits/stdc++.h>
using namespace std;
/***** Debug Code *****/
template <typename T>
concept Printable = requires(T t) {
    { std::cout << t } -> std::same_as<std::ostream &>;
};

```



```

};
template <Printable T>
void __print(const T &x) {
    cerr << x;
}
template <size_t T>
void __print(const bitset<T> &x) {
    cerr << x;
}
template <typename A, typename B>
void __print(const pair<A, B> &p);
template <typename... A>
void __print(const tuple<A...> &t);
template <typename T>
void __print(stack<T> s);
template <typename T>
void __print(queue<T> q);
template <typename T, typename... U>
void __print(priority_queue<T, U...> q);
template <typename A>
void __print(const A &x) {
    bool first = true;
    cerr << '{';
    for (const auto &i : x) {
        cerr << (first ? "" : ","), __print(i);
        first = false;
    }
    cerr << '}';
}
template <typename A, typename B>
void __print(const pair<A, B> &p) {
    cerr << '(';
    __print(p.first);
    cerr << ',';
    __print(p.second);
    cerr << ')';
}
template <typename... A>
void __print(const tuple<A...> &t) {
    bool first = true;
    cerr << '(';
    apply(
        [&first](const auto &...args) {
            ((cerr << (first ? "" : ","), __print(args), first = false), ...);
        },
        t);
    cerr << ')';
}
template <typename T>
void __print(stack<T> s) {
    vector<T> debugVector;
    while (!s.empty()) {
        T t = s.top();
        debugVector.push_back(t);
        s.pop();
    }
    reverse(debugVector.begin(), debugVector.end());
    __print(debugVector);
}

```

```

}
template <typename T>
void __print(queue<T> q) {
    vector<T> debugVector;
    while (!q.empty()) {
        T t = q.front();
        debugVector.push_back(t);
        q.pop();
    }
    __print(debugVector);
}
template <typename T, typename... U>
void __print(priority_queue<T, U...> q) {
    vector<T> debugVector;
    while (!q.empty()) {
        T t = q.top();
        debugVector.push_back(t);
        q.pop();
    }
    __print(debugVector);
}
void _print() { cerr << "]\n"; }
template <typename Head, typename... Tail>
void _print(const Head &H, const Tail &...T) {
    __print(H);
    if (sizeof...(T)) cerr << ", ";
    _print(T...);
}

#define dbg(x...) \
    cerr << "[" << #x << "]" = ["; \
    _print(x)

```

10.4 .bashrc

```

cpp() {
    echo ">> COMPILING <<" 1>&2
    g++ -std=c++17 \
        -O2 \
        -g \
        -g3 \
        -Wextra \
        -Wshadow \
        -Wformat=2 \
        -Wconversion \
        -fsanitize=address,undefined \
        -fno-sanitize-recover \
        -Wfatal-errors \

    if [ $? -ne 0 ]; then
        echo ">> FAILED <<" 1>&2
        return 1
    fi
    echo ">> DONE << " 1>&2
    time ./a.out ${@:2}
}

prepare() {

```

```

for i in {a..z}
do
    cp macro.cpp $i.cpp
    touch $i.py
done

for i in {1..10}
do
    touch in${i}
    touch out${i}
    touch ans${i}
done
}

```

10.5 macro.cpp

```

#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
#define fastio \
    ios_base::sync_with_stdio(false); \
    cin.tie(0); \
    cout.tie(0);
#define len(__x) (int) __x.size()
using ll = long long;
using ld = long double;

```

```

using vll = vector<ll>;
using pll = pair<ll, ll>;
using vll2d = vector<vll>;
using vi = vector<int>;
using vi2d = vector<vi>;
using pii = pair<int, int>;
using vii = vector<pii>;
using vc = vector<char>;
#define all(a) a.begin(), a.end()
#define snd second
#define fst first
#define pb(___x) push_back(___x)
#define mp(___a, ___b) make_pair(___a, ___b)
#define eb(___x) emplace_back(___x)

const ll INF = 1e18;

void run() {}
int32_t main(void) {
    fastio;
    int t;
    t = 1;
    // cin >> t;
    while (t--) run();
}

```