ICPC - Notebook

Contents

1 data structures

1.1 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>;
```

1.2 Segtree Rmq Lazy Max Update

```
struct SegmentTree
{
        int N;
        vll ns, lazy;
        SegmentTree (const vll &xs)
            : N (xs.size ()), ns (4 * N, 0), lazy (4 * N, 0)
        {
                for (size_t i = 0; i < xs.size (); ++i)
                        {
                                update (i, i, xs[i]);
                        }
        }
        void
        update (int a, int b, ll value)
        {
                update (1, 0, N - 1, a, b, value);
        }
        void
        update (int node, int L, int R, int a, int b, ll value)
        {
                if (lazy[node])
                        {
                                ns[node] = max (ns[node], lazy[node]);
                                if (L < R)
                                        {
                                                lazy[2 * node] = max (lazy[2 * node], lazy[node]);
                                                lazy[2 * node + 1] = max (lazy[2 * node + 1], lazy[node]);
                                        }
                                lazy[node] = 0;
                        }
                if (a > R or b < L)
                        return;
                if (a <= L and R <= b)
                        {
                                ns[node] = max (ns[node], value);
                                if (L < R)
                                        {
                                                lazy[2 * node] = max (value, lazy[2 * node]);
                                                lazy[2 * node + 1] = max (value, lazy[2 * node + 1]);
                                        }
                                return;
                        }
                update (2 * node, L, (L + R) / 2, a, b, value);
                update (2 * node + 1, (L + R) / 2 + 1, R, a, b, value);
                ns[node] = max (ns[node * 2], ns[node * 2 + 1]);
        }

        ll
        RMQ (int a, int b)
        {
                return RMQ (1, 0, N - 1, a, b);
        }
        ll
        RMQ (int node, int L, int R, int a, int b)
        {
                if (lazy[node])
                        {
                                ns[node] = max (ns[node], lazy[node]);
                                if (L < R)
                                        {
                                                lazy[node * 2] = max (lazy[node * 2], lazy[node]);
                                                lazy[node * 2 + 1] = max (lazy[node * 2 + 1], lazy[node]);
                                        }
                                lazy[node] = 0;
                        }

                if (a > R or b < L)
                        return 0;
                if (a <= L and R <= b)
                        return ns[node];
                ll x = RMQ (2 * node, L, (L + R) / 2, a, b);
                ll y = RMQ (2 * node + 1, (L + R) / 2 + 1, R, a, b);
                return max (x, y);
        }
};
```

1.3 Segtree Rmq Lazy Range

```
struct SegmentTree {
        int N;
        vll ns, lazy;
        SegmentTree(const vll &xs)
                : N(xs.size()), ns(4 * N, INT_MAX), lazy(4 * N, 0) {
                for (size_t i = 0; i < xs.size(); ++i) update(i, i, xs[i]);
        }
        void update(int a, int b, ll value) {
                update(1, 0, N - 1, a, b, value);
        }
        void update(int node, int L, int R, int a, int b, ll value) {
                if (lazy[node]) {
                        ns[node] = ns[node] == INT_MAX ? lazy[node]
                          : ns[node] + lazy[node];
                        if (L < R) {
                                lazy[2 * node] += lazy[node];
```

```
                                lazy[2 * node + 1] += lazy[node];
                        }
                        lazy[node] = 0;
                }
                if (a > R or b < L) return;
                if (a <= L and R <= b) {
                        ns[node] = ns[node] == INT_MAX ? value : ns[node] + value;
                        if (L < R) {
                                lazy[2 * node] += value;
                                lazy[2 * node + 1] += value;
                        }
                        return;
                }
                update(2 * node, L, (L + R) / 2, a, b, value);
                update(2 * node + 1, (L + R) / 2 + 1, R, a, b, value);
                ns[node] = min(ns[2 * node], ns[2 * node + 1]);
        }
        ll RMQ(int a, int b) { return RMQ(1, 0, N - 1, a, b); }
        ll RMQ(int node, int L, int R, int a, int b) {
                if (lazy[node]) {
                        ns[node] = ns[node] == INT_MAX ? lazy[node]
                          : ns[node] + lazy[node];
                        if (L < R) {
                                lazy[2 * node] += lazy[node];
                                lazy[2 * node + 1] += lazy[node];
                        }
                        lazy[node] = 0;
                }
                if (a > R or b < L) return INT_MAX;

                if (a <= L and R <= b) return ns[node];
                ll x = RMQ(2 * node, L, (L + R) / 2, a, b);
                ll y = RMQ(2 * node + 1, (L + R) / 2 + 1, R, a, b);
                return min(x, y);
        }
};
```

1.4 Segtree Point Rmq

```
class SegTree {
   public:
    int n;
    vector<ll> st;
    SegTree(const vector<ll> &v) : n((int)v.size()), st(n * 4 + 1, LLONG_MAX) {
        for (int i = 0; i < n; ++i) update(i, v[i]);
    }
    void update(int p, ll v) { update(1, 0, n - 1, p, v); }
    ll RMQ(int l, int r) { return RMQ(1, 0, n - 1, l, r); }

   private:
    void update(int node, int l, int r, int p, ll v) {
        if (p < l or p > r) return;     // fora do intervalo.

        if (l == r) {
            st[node] = v;
            return;
        }

        int mid = l + (r - l) / 2;

        update(node * 2, l, mid, p, v);
        update(node * 2 + 1, mid + 1, r, p, v);

        st[node] = min(st[node * 2], st[node * 2 + 1]);
    }

    ll RMQ(int node, int L, int R, int l, int r) {
        if (l <= L and r >= R) return st[node];
        if (L > r or R < l) return LLONG_MAX;
        if (L == R) return st[node];

        int mid = L + (R - L) / 2;

        return min(RMQ(node * 2, L, mid, l, r),
                   RMQ(node * 2 + 1, mid + 1, R, l, r));
    }
};
```

1.5 Sparse-segment-tree

```
#include <bits/stdc++.h>
#pragma GCC optimize("O3")
#define FOR(i, x, y) for (int i = x; i < y; i++)
#define MOD 1000000007
typedef long long ll;
using namespace std;

struct Node {
  int sum, lazy, tl, tr, l, r;
  Node() : sum(0), lazy(0), l(-1), r(-1) {}
};

const int MAXN = 123456;
Node segtree[64 * MAXN];
int cnt = 2;
const ll v = 10;

void push_lazy(int node) {
  if (segtree[node].lazy) {
    segtree[node].sum = (segtree[node].tr - segtree[node].tl + 1) * v;
    int mid = (segtree[node].tl + segtree[node].tr) / 2;
    if (segtree[node].l == -1) {
      segtree[node].l = cnt++;
      segtree[segtree[node].l].tl = segtree[node].tl;
      segtree[segtree[node].l].tr = mid;
    }
    if (segtree[node].r == -1) {
      segtree[node].r = cnt++;
      segtree[segtree[node].r].tl = mid + 1;
      segtree[segtree[node].r].tr = segtree[node].tr;
    }
    segtree[segtree[node].l].lazy = segtree[segtree[node].r].lazy = v;
    segtree[node].lazy = 0;
  }
}

void update(int node, int l, int r) {
  push_lazy(node);
  if (l == segtree[node].tl && r == segtree[node].tr) {
    segtree[node].lazy = v;
    push_lazy(node);
  } else {
    int mid = (segtree[node].tl + segtree[node].tr) / 2;
    if (segtree[node].l == -1) {
      segtree[node].l = cnt++;
```

```
      segtree[segtree[node].l].tl = segtree[node].tl;
      segtree[segtree[node].l].tr = mid;
    }
    if (segtree[node].r == -1) {
      segtree[node].r = cnt++;
      segtree[segtree[node].r].tl = mid + 1;
      segtree[segtree[node].r].tr = segtree[node].tr;
    }

    if (l > mid)
      update(segtree[node].r, l, r);
    else if (r <= mid)
      update(segtree[node].l, l, r);
    else {
      update(segtree[node].l, l, mid);
      update(segtree[node].r, mid + 1, r);
    }

    push_lazy(segtree[node].l);
    push_lazy(segtree[node].r);
    segtree[node].sum =
      segtree[segtree[node].l].sum + segtree[segtree[
    node].r].sum;
  }
}

int query(int node, int l, int r) {
  push_lazy(node);
  if (l == segtree[node].tl && r == segtree[node].tr)
    return segtree[node].sum;
  else {
    int mid = (segtree[node].tl + segtree[node].tr) /
     2;
    if (segtree[node].l == -1) {
      segtree[node].l = cnt++;
      segtree[segtree[node].l].tl = segtree[node].tl;
      segtree[segtree[node].l].tr = mid;
    }
    if (segtree[node].r == -1) {
      segtree[node].r = cnt++;
      segtree[segtree[node].r].tl = mid + 1;
      segtree[segtree[node].r].tr = segtree[node].tr;
    }

    if (l > mid)
      return query(segtree[node].r, l, r);
    else if (r <= mid)
      return query(segtree[node].l, l, r);
    else
      return query(segtree[node].l, l, mid) +
             query(segtree[node].r, mid + 1, r);
  }
}
int main() {
  iostream::sync_with_stdio(false);
  cin.tie(0);
  int m;
  cout << "enter m: ";
  cin >> m;

  segtree[1].sum = 0;
  segtree[1].lazy = 0;
  segtree[1].tl = 1;
  segtree[1].tr = 1e9;

  int c = 0;
  FOR(_, 0, m) {
    int d, x, y;
    cout << "enter d, x, y:";
    cin >> d >> x >> y;
    if (d == 1) {
      c = query(1, x, y);
      cout << "c: ";
      cout << c << '\n';
    } else {
      update(1, x, y);
    }
  }
  return 0;
}
```

1.6 Segtree Rsq Lazy Range Sum

```
struct SegTree {
        int N;
        vector<ll> ns, lazy;

        SegTree(const vector<ll> &xs)
                : N(xs.size()), ns(4 * N, 0), lazy(4
    * N, 0) {
                for (size_t i = 0; i < xs.size(); ++i
    ) update(i, i, xs[i]);
        }

        void update(int a, int b, ll value) {
                update(1, 0, N - 1, a, b, value);
        }

        void update(int node, int L, int R, int a,
    int b, ll value) {
                // Lazy propagation
                if (lazy[node]) {
                        ns[node] += (R - L + 1) *
    lazy[node];

                        if (L < R)  // Se o ón ãno é
    uma folha, propaga
                        {
                                lazy[2 * node] +=
    lazy[node];
                                lazy[2 * node + 1] +=
     lazy[node];
                        }

                        lazy[node] = 0;
                }

                if (a > R or b < L) return;

                if (a <= L and R <= b) {
                        ns[node] += (R - L + 1) *
    value;

                        if (L < R) {
                                lazy[2 * node] +=
    value;
                                lazy[2 * node + 1] +=
     value;
                        }

                        return;
                }

                update(2 * node, L, (L + R) / 2, a, b
    , value);
                update(2 * node + 1, (L + R) / 2 + 1,
     R, a, b, value);

                ns[node] = ns[2 * node] + ns[2 * node
     + 1];
        }

        ll RSQ(int a, int b) { return RSQ(1, 0, N -
    1, a, b); }

        ll RSQ(int node, int L, int R, int a, int b)
    {
                if (lazy[node]) {
                        ns[node] += (R - L + 1) *
    lazy[node];

                        if (L < R) {
                                lazy[2 * node] +=
    lazy[node];
```

```
                                lazy[2 * node + 1] += lazy[node];
                        }

                        lazy[node] = 0;
                }

                if (a > R or b < L) return 0;

                if (a <= L and R <= b) return ns[node];

                ll x = RSQ(2 * node, L, (L + R) / 2, a, b);
                ll y = RSQ(2 * node + 1, (L + R) / 2 + 1, R, a, b);

                return x + y;
        }
};
```

1.7 Segtree Rxq Lazy Range Xor

```
struct SegTree {
        int N;
        vector<ll> ns, lazy;

        SegTree(const vector<ll> &xs)
                : N(xs.size()), ns(4 * N, 0), lazy(4 * N, 0) {
                for (size_t i = 0; i < xs.size(); ++i) update(i, i, xs[i]);
        }

        void update(int a, int b, ll value) {
                update(1, 0, N - 1, a, b, value);
        }

        void update(int node, int L, int R, int a, int b, ll value) {
                // Lazy propagation
                if (lazy[node]) {
                        ns[node] ^= lazy[node];

                        if (L < R)  // Se o ón ãno é uma folha, propaga
                        {
                                lazy[2 * node] ^= lazy[node];
                                lazy[2 * node + 1] ^= lazy[node];
                        }

                        lazy[node] = 0;
                }

                if (a > R or b < L) return;

                if (a <= L and R <= b) {
                        ns[node] ^= value;

                        if (L < R) {
                                lazy[2 * node] ^= value;
                                lazy[2 * node + 1] ^= value;
                        }

                        return;
                }

                update(2 * node, L, (L + R) / 2, a, b, value);
                update(2 * node + 1, (L + R) / 2 + 1, R, a, b, value);

                ns[node] = ns[2 * node] ^ ns[2 * node + 1];
        }

        ll rxq(int a, int b) { return RSQ(1, 0, N - 1, a, b); }

        ll rxq(int node, int L, int R, int a, int b) {
                if (lazy[node]) {
                        ns[node] ^= lazy[node];

                        if (L < R) {
                                lazy[2 * node] ^= lazy[node];
                                lazy[2 * node + 1] ^= lazy[node];
                        }

                        lazy[node] = 0;
                }

                if (a > R or b < L) return 0;

                if (a <= L and R <= b) return ns[node];

                ll x = rxq(2 * node, L, (L + R) / 2, a, b);
                ll y = rxq(2 * node + 1, (L + R) / 2 + 1, R, a, b);

                return x ^ y;
        }
};
```

1.8 Dsu

```
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]
```

1.9 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];
  }
};
```

1.10 Sparse Table Rmq

```
/*
        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 graphs

2.1 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};
}
```

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

    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);
    }
};
```

2.3 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;
}
```

2.4 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;
}
```

2.5 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(u);
                        }
                }
        }

        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);
        }
};
```

2.6 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;
}
```

2.7 Kruskal

```
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 te 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 verties.
    """
    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
```

2.8 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]);
                                }
                        }
```

```
                }
        }
};
```

2.9 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;
}
```

2.10 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;
}
```

3 extras

3.1 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;
}
```

3.2 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] * 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;
        }
};
```

3.3 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 dynamic programming

4.1 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];
}
```

4.2 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;
```

```
}
```

4.3 Knapsack Dp Values 01

```
const int MAX_N = 1001;
const int MAX_S = 100001;
array<array<int, MAX_S>, MAX_N> dp;
bool check[MAX_N][MAX_S];
pair<int, vi> knapsack(int S, const vector<pii> &xs) {
        int N = (int)xs.size();

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

        for (int m = 0; m <= S; ++m) dp[0][m] = 0;

        for (int i = 1; i <= N; ++i) {
                for (int m = 1; m <= S; ++m) {
                        dp[i][m] = dp[i - 1][m];
                        check[i][m] = false;

                        auto [w, v] = xs[i - 1];

                        if (w <= m and (dp[i - 1][m - w] + v) >= dp[i][m]) {
                                dp[i][m] = dp[i - 1][m - w] + v;
                                check[i][m] = true;
                        }
                }
        }

        int m = S;
        vi es;

        for (int i = N; i >= 1; --i) {
                if (check[i][m]) {
                        es.push_back(i);
                        m -= xs[i - 1].first;
                }
        }

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

        return {dp[N][S], es};
}
```

4.4 Tsp

```
using vi = vector<int>;
vector<vi> dist;
vector<vi> memo;
/* O ( 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;
}
```

5 trees

5.1 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;
}
```

5.2 Heavy-light-decomposition

```
// iagorrr ;)
#include <bits/stdc++.h>
using namespace std;
#ifdef DEBUG
#include "debug.cpp"
#else
#define dbg(...) 666
#endif
#define endl '\n'
#define fastio                          \
    ios_base::sync_with_stdio(false); \
    cin.tie(0);                         \
    cout.tie(0);
#define rep(i, l, r) for (int i = (l); i < (r); i++)
#define len(__x) (ll) __x.size()
using ll = long long;
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 INV(xxxx) \
    for (auto &xxx : xxxx) cin >> xxx;
#define PRINTV(___x) \
    for_each(all(___x), [](ll &____x) { cout << ____x << ' '; }), cout << '\n';
#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)
#define rsz(___x, ___n) resize(___x, ___n)

const ll INF = 1e18;

struct HLD {
    int n;
    vi sizes;
    vi2d g;
    vi groups;
    vi heavy;
    HLD(int n) : n(n), sizes(n + 1), g(n + 1), groups(n + 1), heavy(n + 1) {}
    void get_sizes(int u, int p) {
        int sz = 1;
        int bigc = -1;
        for (auto &v : g[u])
            if (v != p) {
                get_sizes(p, u);
                if (bigc == -1 or sizes[bigc] < sizes[v])
                    bigc = v, heavy[u] = v;
                sz += sizes[v];
            }
        sizes[u] = sz;
    }
    void decompose(int u, int p) {
        groups[u] = p;
        for (auto &v : g[u])
            if (v != p) {
                if (v == heavy[u])
                    decompose(v, p);
                else
                    decompose(v, v);
            }
    }
};
void run() {}
int32_t main(void) {
    fastio;
    int t;
    t = 1;
    // cin >> t;
    while (t--) run();
}
```

5.3 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;
}
```

6 searching

6.1 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);
}
```

7 math

7.1 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); }
```

7.2 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;
}
```

7.3 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);
}
```

7.4 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;
}
```

7.5 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];
}
```

7.6 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;
}
```

7.7 Is-prime

```
bool isprime(ll n) {  // O(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;
}
```

7.8 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);
}
```

7.9 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;
}
```

7.10 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;
}
```

7.11 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;
}
```

7.12 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;
        }
};
```

7.13 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;
}
```

7.14 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; }
```

7.15 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;
}

// lsit 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;
}
```

7.16 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;
}
```

7.17 Modular-inverse-using-phi

```cpp
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); }
```

7.18 Factorization

```cpp
// 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;
}
```

7.19 Gcd

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

7.20 Combinatorics With Repetitions

```cpp
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;
        }
}
```

8 strings

8.1 Rabin-karp

```cpp
vi rabin_karp(string const &s, string const &t) {
        ll p = 31;
        ll m = 1e9 + 9;
        int S = s.size(), T = t.size();

        vll p_pow(max(S, T));
        p_pow[0] = 1;
        for (int i = 1; i < (int)p_pow.size(); i++)
                p_pow[i] = (p_pow[i - 1] * p) % m;

        vll h(T + 1, 0);
        for (int i = 0; i < T; i++)
                h[i + 1] = (h[i] + (t[i] - 'a' + 1) * p_pow[i]) % m;
        ll h_s = 0;
        for (int i = 0; i < S; i++)
                h_s = (h_s + (s[i] - 'a' + 1) * p_pow[i]) % m;

        vi occurences;
        for (int i = 0; i + S - 1 < T; i++) {
                ll cur_h = (h[i + S] + m - h[i]) % m;
                // IT DON'T CONSIDERE CONLISIONS !
                if (cur_h == h_s * p_pow[i] % m)
                        occurences.push_back(i);
        }
        return occurences;
}
```

8.2 Trie-naive

```cpp
//  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 occurence
                                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;
}
```

8.3 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];
        }
};
```