Team Notebook

National Institute of Technology Hamirpur

December 20, 2023

Contents

1 Algorithms

1.1 Mo's algorithm on trees

```
void MoAlgoOnTree() {
    Dfs(0, -1);
    vector<int> euler(tk);
    for (int i = 0; i < n; ++i) {
        euler[tin[i]] = i;
        euler[tout[i]] = i;
    }
    vector<int> l(q), r(q), qr(q), sp(q,
        -1);
    for (int i = 0; i < q; ++i) {
        if (tin[u[i]] > tin[v[i]]) swap(u[
            i], v[i]);
        int z = GetLCA(u[i], v[i]);
        sp[i] = z[i];
        if (z == u)
            l[i] = tin[u[i]], r[i] = tin[v
                [i]];
        else
            l[i] = tout[u[i]], r[i] = tin[
                v[i]];
        qr[i] = i;
    }
    sort(qr.begin(), qr.end(), [&](int i,
        int j) {
        if (l[i] / kB == l[j] / kB) return
            r[i] < r[j];
        return l[i] / kB < l[j] / kB;
    });
    vector<bool> used(n);
    // Add(v): add/remove v to/from the
        path based on used[v]
    for (int i = 0, tl = 0, tr = -1; i <
        q; ++i) {
        while (tl < l[qr[i]]) Add(euler[tl
            ++]);
        while (tl > l[qr[i]]) Add(euler[--
            tl]);
        while (tr > r[qr[i]]) Add(euler[tr
            --]);
        while (tr < r[qr[i]]) Add(euler[++
            tr]);
        // add/remove LCA(u, v) if
            necessary
    }
}
```

1.2 Mo's algorithm

```
// TODO: Find optimal block_size for
    compiler optimization
const int block_size = 450; // sqrt(N)
// TODO: Define constants needed for
    global variables (arrays etc. - The
    DS)
const int N = 200005;
const int MX = 1E6 + 5;
// TODO: initialize data structure, along
    with answer variable - can define
    get_answer() function also
ll a[N], cnt[MX], sum[N];
ll ans;

struct Query {
    int l, r, idx;
    Query() = default;
    Query(int L, int R, int i) : l(L), r(
        R), idx(i) {}
    pair<int, int> toPair() const {
        return make_pair(l / block_size,
            ((l / block_size) & 1) ? -r :
            +r);
    }
    bool operator<(const Query &other)
        const {
        return this->toPair() < other.
            toPair();
    }
};

// TODO: remove value at idx from data
    structure
void remove(int idx) {
    ll ocnt = cnt[a[idx]], ncnt = ocnt -
        1;
    ll old_ocnt = sum[ocnt], old_ncnt =
        sum[ncnt];
    sum[ocnt] -= a[idx];
```

```
    --cnt[a[idx]];   // reduce count
    sum[ncnt] += a[idx];
    ll new_ocnt = sum[ocnt], new_ncnt =
        sum[ncnt];

    ans += ocnt * ocnt * (new_ocnt -
        old_ocnt);
    ans += ncnt * ncnt * (new_ncnt -
        old_ncnt);
}

// TODO: add value at idx from data
    structure
void add(int idx) {
    ll ocnt = cnt[a[idx]], ncnt = ocnt +
        1;
    ll old_ocnt = sum[ocnt], old_ncnt =
        sum[ncnt];
    sum[ocnt] -= a[idx];
    ++cnt[a[idx]];   // increase count
    sum[ncnt] += a[idx];
    ll new_ocnt = sum[ocnt], new_ncnt =
        sum[ncnt];

    ans += ocnt * ocnt * (new_ocnt -
        old_ocnt);
    ans += ncnt * ncnt * (new_ncnt -
        old_ncnt);
}
```

```
vector<ll> mo_s_algorithm(vector<Query>
    queries) {
    vector<ll> answers(queries.size());
    sort(queries.begin(), queries.end());

    int cur_l = 0;
    int cur_r = -1;
    // invariant: data structure will
        always reflect the range [cur_l,
        cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        answers[q.idx] = ans;
    }
    return answers;
}
```

1.3 Next Greater Element

```
vector<ll> next_greater(vector<ll> &a) {
    ll n = ll(a.size());
    vector<ll> res(n, n);
    stack<ll> stk;
    for (ll i = n - 1; i >= 0; --i) {
        while (!stk.empty() && a[stk.top()
            ] <= a[i]) stk.pop();
        if (!stk.empty()) res[i] = stk.top
            ();
        stk.push(i);
    }
    return res;
}

vector<ll> prev_greater(vector<ll> &a) {
    ll n = ll(a.size());
    vector<ll> res(n, -1);
    stack<ll> stk;
    for (ll i = 0; i < n; ++i) {
        while (!stk.empty() && a[stk.top()
            ] <= a[i]) stk.pop();
        if (!stk.empty()) res[i] = stk.top
            ();
        stk.push(i);
    }
    return res;
}
```

1.4 Sqrt Decomposition

```
// fix for compiler optimization
const int block_size = 550; // sqrt(N)

struct Square_Root_Decomposition {
    int n, block_size;
    vector<ll> a;
    vector<vector<ll>> block;
    Square_Root_Decomposition(vector<ll>
        &v): a(v) {
        n = int(a.size());
        for (int i = 0; i < n; ++i) {
            if (block.empty() || int(block
                .back().size()) ==
                block_size) block.
                emplace_back();
            block.back().push_back(a[i]);
        }
        for (auto &bl : block) sort(bl.
            begin(), bl.end());
    }
    // O(block_size)
    void update(int idx, ll val) {
        // update block
        vector<ll> &bl = block[idx /
            block_size];
        int sz = int(bl.size()),
            pos = int(lower_bound(bl.begin
                (), bl.end(), a[idx]) - bl
                .begin());
        bl[pos] = val;
        while (pos + 1 < sz && bl[pos] >
            bl[pos + 1])
            swap(bl[pos], bl[pos + 1]),
                ++pos;
        while (pos >= 1 && bl[pos] < bl[
            pos - 1])
            swap(bl[pos], bl[pos - 1]),
                --pos;
        // update array
        a[idx] = val;
    }
    // O(block_size)
    int query(int l, int r, ll val) {
        int ans = 0, lb = l / block_size,
            rb = r / block_size;
        if (lb == rb) {
            for (int i = l; i <= r; ++i) {
                ans += (a[i] < val);
            }
        } else {
            for (int i = l; i < (lb + 1) *
                block_size; ++i) {
                ans += (a[i] < val);
            }
            for (int i = rb * block_size;
                i <= r; ++i) {
                ans += (a[i] < val);
            }
            for (int i = lb + 1; i < rb;
                ++i) {
                ans += int(lower_bound(
                    block[i].begin(),
                    block[i].end(), val) -
                    block[i].begin());
            }
        }
        return ans;
    }
};
```

2 Data Structures

2.1 BIT

```
template <class T>
class BIT {
  public:
    vector<T> tree;
    int n;

    BIT(int _n) : n(_n + 1) { tree.resize(
        n); }
    BIT(const vector<T> &a) : BIT(int(a.
        size()) + 1) {
        for (int i = 0; i < int(a.size());
            ++i) add(i, a[i]);
    }

    void add(int i, T delta) {
        ++i;
```

```
        while (i < n) {
            tree[i] += delta;
            i += (i & -i);
        }
    }

    T get(int i) {
        ++i;
        T sum{};
        while (i > 0) {
            sum += tree[i];
            i -= (i & -i);
        }
        return sum;
    }
    T get(int l, int r) { return get(r) -
        get(l - 1); }

    // finds first index where get(i) >=
        k
    int find_first(T k) { // O(log(n)) -
        Walking the BIT
        int i = 0;
        int mask = (1 << __lg(n - 1));
        while (mask != 0) {
            int t_i = i + mask; // the
                midpoint of the current
                interval
            mask >>= 1;         // halve
                the current interval
            if (t_i >= n)       // avoid
                overflow
                continue;
            if (k > tree[t_i]) {
                // if smaller, subtract
                    tree[t_i] and move up
                i = t_i;            // update
                    index
                k -= tree[t_i];     // update
                    the frequency for the
                    next iteration
            }
        }
        return i;
    };
};
```

2.2 DSU

```
struct DSU {
    int N;
    vector<int> par, siz;
    int components;
    DSU(int n) : N(n), par(N), siz(N, 1),
        components(N) {
        for (int i = 0; i < N; i++) par[i]
            = i;
    }
    int root(int X) {
        if (par[X] != X) par[X] = root(par
            [X]);
        return par[X];
    }
    bool same(int X, int Y) { return root(
        X) == root(Y); }
    void unite(int X, int Y) {
        X = root(X), Y = root(Y);
        if (X == Y) return;
        if (siz[Y] < siz[X]) swap(X, Y);
        par[X] = Y;
        siz[Y] += siz[X];
        siz[X] = 0;
        --components;
    }
    int get_size(int X) { return siz[root(
        X)]; }
};
```

2.3 Lazy Segment Tree 1

```
int n, q;
vector<ll> a, lazy;
vector<matrix> t;

void build(int in = 1, int s = 0, int e =
    n - 1) {
    if (s == e) {
        t[in] = fib(a[s]);
    } else {
```

```
        int mid = (s + e) >> 1;
        build(2 * in, s, mid);
        build(2 * in + 1, mid + 1, e);
        t[in] = t[2 * in] + t[2 * in + 1];
    }
}

void push(int in) {
    if (lazy[in] != 0) {
        lazy[2 * in] += lazy[in];
        lazy[2 * in + 1] += lazy[in];

        t[2 * in] = t[2 * in] * fib(lazy[
            in]);
        t[2 * in + 1] = t[2 * in + 1] *
            fib(lazy[in]);

        lazy[in] = 0;
    }
}

void update(int in, int s, int e, int qs,
    int qe, int val, matrix& toAdd) {
    if (qs > qe) {
        return;
    }

    if (s == qs && e == qe) {
        t[in] = t[in] * toAdd;
        lazy[in] += val;
    } else {
        push(in);
        int mid = (s + e) >> 1;
        update(2 * in, s, mid, qs, min(
            mid, qe), val, toAdd);
        update(2 * in + 1, mid + 1, e,
            max(mid + 1, qs), qe, val,
            toAdd);

        t[in] = t[2 * in] + t[2 * in + 1];
    }
}

int query(int in, int s, int e, int qs,
    int qe) {
    if (qs > qe) {
        return 0;
    }

    if (s == qs && e == qe) {
        return t[in].mat[0][1];
    }

    push(in);

    int mid = (s + e) >> 1;

    return (query(2 * in, s, mid, qs, min(
        mid, qe)) +
            query(2 * in + 1, mid + 1, e,
                max(mid + 1, qs), qe)) %
        MOD;
}
```

2.4 Lazy Segment Tree 2

```
void build(int a[], int v, int tl, int tr)
    {
    if (tl == tr) {
        t[v] = a[tl];
    } else {
        int tm = (tl + tr) / 2;
        build(a, v * 2, tl, tm);
        build(a, v * 2 + 1, tm + 1, tr);
        t[v] = 0;
    }
}

void push(int v) {
    t[v * 2] += lazy[v];
    lazy[v * 2] += lazy[v];
    t[v * 2 + 1] += lazy[v];
    lazy[v * 2 + 1] += lazy[v];
    lazy[v] = 0;
}

void update(int v, int tl, int tr, int l,
    int r, int addend) {
    if (l > r)
        return;
    if (l == tl && tr == r) {
        t[v] += addend;
```

```
        lazy[v] += addend;
    } else {
        push(v);
        int tm = (tl + tr) / 2;
        update(v * 2, tl, tm, l, min(r, tm), addend);
        update(v * 2 + 1, tm + 1, tr, max(l, tm + 1), r, addend);
        t[v] = max(t[v * 2], t[v * 2 + 1]);
    }
}

int query(int v, int tl, int tr, int l, int r) {
    if (l > r)
        return -INF;
    if (l == tl && tr == r)
        return t[v];
    push(v);
    int tm = (tl + tr) / 2;
    return max(query(v * 2, tl, tm, l, min(r, tm)),
               query(v * 2 + 1, tm + 1, tr, max(l, tm + 1), r));
}
```

2.5 Segment Tree

```
// can use decltype while initialising to make a little bit faster
template <class T, class op = function<T(const T &, const T &)>, class id = function<T()>>
class SegTree {
  public:
    SegTree() = default;
    SegTree(int n, op operation_, id identity_)
        : SegTree(vector<T>(n, identity_()), operation_, identity_) {}
    int ceil_pow2(int n) {
        int x = 0;
        while ((1U << x) < (unsigned int)(n)) x++;
        return x;
    }
    SegTree(const vector<T> &v, op operation_, id identity_)
        : operation(operation_), initialize(identity_), _n(int(v.size())) {
        height = ceil_pow2(_n);
        size = (1 << height);
        tree.resize(2 * size, initialize());
        for (int i = 0; i < _n; i++) tree[size + i] = v[i];
        for (int i = size - 1; i >= 1; i--) {
            calc(i);
        }
    }

    T _query(int node, int node_lo, int node_hi, int q_lo, int q_hi) {
        // if range is completely inside [q_lo, q_hi], then just return its ans
        if (q_lo <= node_lo && node_hi <= q_hi)
            return tree[node];
        if (node_hi < q_lo || q_hi < node_lo)
            return initialize(); // if disjoint ignore
        int last_in_left = (node_lo + node_hi) / 2;
        return operation(_query(2 * node, node_lo, last_in_left, q_lo, q_hi),
                         _query(2 * node + 1, last_in_left + 1, node_hi, q_lo, q_hi));
    }

    void _update(int node, int node_lo, int node_hi, int q_lo, int q_hi, T
```

```
    value) {
    // happens only once when leaf [id
        , id]
    if (q_lo <= node_lo && node_hi
        <= q_hi) {
        tree[node] = value;
        return;
    }
    // in disjoint just return
    if (node_hi < q_lo || q_hi <
        node_lo) return;
    int last_in_left = (node_lo +
        node_hi) / 2;
    _update(2 * node, node_lo,
        last_in_left , q_lo, q_hi,
        value);
    _update(2 * node + 1, last_in_left
        + 1, node_hi, q_lo, q_hi,
        value);

    // after updating now set, Post
        Call Area
    calc(node);
}

T _kth_order(int node, int node_lo,
    int node_hi, T k) {
    if (node_lo == node_hi) return
        node_lo;
    int last_in_left = (node_lo +
        node_hi) >> 1;
    if (tree[2 * node] >= k) return
        _kth_order(2 * node, node_lo,
        last_in_left , k);
    return _kth_order(2 * node + 1,
        last_in_left + 1, node_hi, k -
        tree[2 * node]);
}

T all_query() { return tree[1]; }
T query(int p) {
    assert(0 <= p && p < _n);
    return tree[p + size];
}
T query(int l, int r) {
    assert(0 <= l && l <= r && r < _n)
        ;
    return _query(1, 0, size - 1, l, r
        );
}
void update(int p, T x) {
    assert(0 <= p && p < _n);
    _update(1, 0, size - 1, p, p, x);
}
T kth_order(T k) {
    assert(k <= tree[1]);
    return _kth_order(1, 0, size - 1,
        k);
}

private:
vector<T> tree;
    void calc(int k) { tree[k] = operation
        (tree[2 * k], tree[2 * k + 1]); }
    op operation;
    id initialize ;
    int _n, size, height;
};
```

2.6 Sparse Table

```
template <class T, class U = function<T(
    const T &, const T &)>>
class Sparse_Table {
    ll N, K;
    vector<int> LOG;
    vector<vector<T>> st;
    U op;

    ll log2_floor(unsigned long long i) {
        return 63 - __builtin_clzll(i);
    }

  public:
    Sparse_Table() = default;
    Sparse_Table(const vector<T> &arr,
        const U &OP)
        : N(ll(arr.size())), K(log2_floor(
            N)), LOG(N + 1), st(N, vector
            <T>(K + 1)), op(OP) {
        LOG[1] = 0;
```

```
        for (ll i = 2; i <= N; i++) LOG[i
            ] = LOG[i / 2] + 1;
        for (ll i = 0; i < N; i++)
            st[i][0] = arr[i];
        for (ll j = 1; j <= K; j++)
            for (ll i = 0; i + (1 << j)
                <= N; i++)
                st[i][j] = op(st[i][j -
                    1], st[i + (1 << (j -
                    1))][j - 1]);
    }
    T query(ll L, ll R) {
        if (L > R) swap(L, R);
        ll j = LOG[R - L + 1];
        T res = op(st[L][j], st[R - (1
            << j) + 1][j]);
        return res;
    }
};
```

2.7 Trie

```
struct Node {
    Node* links[2];
    int val;
};

Node* naya() {
    Node* temp = new Node();
    temp->val = 0;
    temp->links[0] = NULL;
    temp->links[1] = NULL;
    return temp;
}

template <typename T>
class Trie {
    private:
    Node* root = naya();

    public:
    int BIT;
    Trie(int sz) {
        root = naya();
        BIT = sz;
    }

    void insert(T num) {
        Node* node = root;
        for (T i = BIT; i >= 0; i--) {
            T bit = (num >> i) & 1;
            node->val++;
            if (node->links[bit] == NULL
                ) {
                node->links[bit] = naya();
            }
            node = node->links[bit];
        }
        node->val++;
    }

    void remove(T num) {
        Node* node = root;
        for (T i = BIT; i >= 0; i--) {
            node->val--;
            T bit = (num >> i) & 1;
            node = node->links[bit];
        }
        node->val--;
    }

    T maxxor(int num) {
        T res = 0;
        Node* node = root;
        for (T i = BIT; i >= 0; i--) {
            T bit = (num >> i) & 1;
            if (node->links[!bit] && node
                ->links[!bit]->val) {
                res |= (static_cast<T>(1)
                    << i);
                node = node->links[!bit];
            } else if (node->links[bit]
                && node->links[bit]->val)
                node = node->links[bit];
        }
        return res;
    }

    T minxor(T num) {
        T res = 0;
        Node* node = root;
        for (T i = BIT; i >= 0; i--) {
```

```
            T bit = (num >> i) & 1;
            if (node->links[bit] && node
                ->links[bit]->val) {
                node = node->links[bit];
            } else if (node->links[!bit]
                && node->links[!bit]->val)
                {
                res |= ( static_cast <T>(1)
                    << i);
                node = node->links[!bit];
            }
        }
        return res;
    }
};
```

3 Geometry

3.1 point

```
template <typename T>
struct P {
    T x, y;
    P(T x = 0, T y = 0) : x(x), y(y) {}
    bool operator<(const P &p) const {
        return tie(x, y) < tie(p.x, p.y);
    }
    bool operator==(const P &p) const {
        return tie(x, y) == tie(p.x, p.y);
    }
    P operator-() const { return {-x, -y};
        }
    P operator+(P p) const { return {x + p
        .x, y + p.y}; }
    P operator-(P p) const { return {x - p
        .x, y - p.y}; }
    P operator*(T d) const { return {x * d
        , y * d}; }
    P operator/(T d) const { return {x / d
        , y / d}; }
    T dist2() const { return x * x + y * y
        ; }
    double len() const { return sqrt(dist2
        ()); }
    P unit() const { return *this / len();
        }
    friend T dot(P a, P b) { return a.x *
        b.x + a.y * b.y; }
    friend T cross(P a, P b) { return a.x
        * b.y - a.y * b.x; }
    friend T cross(P a, P b, P o) {
        return cross(a - o, b - o);
    }
};
using pt = P<ll>;
```

4 Graphs

4.1 Binary Lifting

```
template <typename T>
class binary_lifting {
  public:
    int n;
    vector<vector<T>> dp;
    vector<T> lev;
    vector<vector<T>> Tree;

    binary_lifting (const vector<vector<T
        >> &tree) {
        n = static_cast <int>(tree.size());
        dp.resize(n);
        lev.resize(n);
        Tree.resize(n);
        for (int i = 0; i < n; i++) {
            dp[i].resize(19);
            for (auto &v : tree[i]) {
                Tree[i].push_back(v);
            }
        }
    }

    void Levels(T u, T p) {
        for (auto &v : Tree[u]) {
            if (v == p) continue;
            lev[v] = lev[u] + 1;
            Levels(v, u);
        }
    }
```

```
void Lift(T u, T p) {
    dp[u][0] = p;
    for (int i = 1; i <= 18; i++) {
        if (dp[u][i - 1])
            dp[u][i] = dp[dp[u][i -
                1]][i - 1];
        else
            dp[u][i] = 0;
    }
    for (auto &v : Tree[u]) {
        if (v == p) continue;
        Lift(v, u);
    }
}

T get(T u, T jump) {
    for (int i = 18; i >= 0; i--) {
        if (jump == 0 || u == 0) break
            ;
        if (jump >= (1 << i)) {
            jump -= (1 << i);
            u = dp[u][i];
        }
    }
    return u;
}

T Query(T u, T v) {
    if (lev[u] < lev[v]) swap(u, v);
    u = get(u, lev[u] - lev[v]);
    if (u == v) return u;
        for (int i = 18; i >= 0; i--) {
            if (dp[u][i] != dp[v][i]) {
                u = dp[u][i];
                v = dp[v][i];
            }
        }
        return get(u, 1);
    }
};
```

4.2 Condensation Graph

```
vector<vector<int>> scc = Kosaraju(g, n);
DSU dsu(n);
for (auto &x : scc) {
    for (auto &y : x) {
        dsu.Merge(y, x[0]);
    }
}
vector<vector<int>> compressedGraph(n +
    1);
for (auto &x : edges) {
    int u = dsu.Find(x[0]), v = dsu.Find(x
        [1]);
    if (u == v) continue;
    compressedGraph[u].push_back(v);
}
```

4.3 Floyd Warshall

```
for (ll i = 0, u, v, wt; i < m; ++i) {
    cin >> u >> v >> wt, --u, --v;
    d[u][v] = wt;
    d[v][u] = wt;
}
for (ll i = 0; i < n; ++i) d[i][i] = 0;
// floyd warshall to calculate the
    distances
for (ll k = 0; k < n; ++k) {
    for (ll i = 0; i < n; ++i) {
        for (ll j = 0; j < n; ++j) {
            d[i][j] = min(d[i][j], d[i][k]
                + d[k][j]);
        }
    }
}
```

4.4 Get Centroid

```
// vis[i] = true if we have chosen i as
    the centroid of some
// component, equivalent to the fact that
    these vertices are not in
// remaining components and also block off
    the components from each
// other
vector<int> vis(n);
// par[i] = parent of i in the centroid
    tree
vector<int> par(n);
```

```
vector<int> sub(n);
// blocked vertex => size = 0
auto dfs_subtree = [&](const auto& self,
    int node, int p) -> int {
    if (vis[node]) return 0;
    sub[node] = 1;
    for (auto& child : g[node])
        if (child != p) sub[node] += self(
            self, child, node);
    return sub[node];
};

auto dfs_centroid = [&](const auto& self,
    int node, int p, int limit) -> int {
    for (auto& child : g[node])
        if (child != p && !vis[child] &&
            sub[child] > limit)
            return self(self, child, node,
                limit);
    return node;
};

// p -> parent in the centroid tree
auto dfs_create = [&](const auto& self,
    int node, int p) -> void {
    dfs_subtree(dfs_subtree, node, -1);
    int centroid = dfs_centroid(
        dfs_centroid, node, -1, sub[node]
        / 2);
    vis[centroid] = 1;
    par[centroid] = p;
    for (auto v : g[centroid])
        if (!vis[v]) self(self, v,
            centroid);
};

dfs_create(dfs_create, 0, -1);
```

4.5 Kosaraju (SCC)

```
vector<vector<int>> Kosaraju(vector<
    vector<int>> &graph, int n) {
    vector<int> order;
    vector<bool> vis(n + 1);
    vector<vector<int>> comps;

    auto dfs = [&](const auto &go, int u)
        -> void {
        vis[u] = true;
        for (auto &v : graph[u]) {
            if (vis[v]) continue;
            go(go, v);
        }
        order.push_back(u);
    };

    for (int i = 1; i <= n; i++) {
        if (!vis[i]) dfs(dfs, i);
    }
    vector<vector<int>> transpose(n + 1);
    for (int i = 1; i <= n; i++) {
        for (auto &e : graph[i]) transpose
            [e].push_back(i);
    }
    for (int i = 1; i <= n; i++) vis[i] =
        0;

    auto getscc = [&](const auto &go, int
        u, vector<int> &cur) -> void {
        vis[u] = true;
        cur.push_back(u);
        for (auto &v : transpose[u]) {
            if (vis[v]) continue;
            go(go, v, cur);
        }
    };

    while (order.size()) {
        int node = order.back();
        order.pop_back();
        if (vis[node]) continue;
        vector<int> cur;
        getscc(getscc, node, cur);
        comps.push_back(cur);
    }
    return comps;
}
```

4.6 LCA

```
template <typename T>
```

```
struct RMQ {
    int n, levels;
    vector<T> values;
    vector<vector<int>> range_low;

    int min_index(int a, int b) const {
        return values[b] < values[a] ? b :
         a; }

    void build(const vector<T> &_values)
        {
        values = _values;
        n = int(values.size());
        build();
    }

    void build() {
        levels = 32 - __builtin_clz(n);
        range_low.resize(levels);

        for (int k = 0; k < levels; k++)
            range_low[k].resize(n - (1
                << k) + 1);
        for (int i = 0; i < n; i++)
            range_low[0][i] = i;
        for (int k = 1; k < levels; k++)
            for (int i = 0; i <= n - (1
                << k); i++)
                range_low[k][i] =
                    min_index(
                        range_low[k - 1][i],
                            range_low[k - 1][i
                             + (1 << (k - 1))
                            ]);
    }

    int rmq_index(int a, int b) const {
        assert(a < b);
        int level = 31 - __builtin_clz(b -
            a);
        return min_index(range_low[level][
            a],
                        range_low[level][
                            b - (1 <<
                            level)]);
    }

    int rmq_value(int a, int b) const {
        return values[rmq_index(a, b)]; }
};

struct LCA {
    int n;
    vector<vector<int>> adj;
    vector<int> depth;
    vector<int> euler, first_occurrence;
    vector<int> tour_start, tour_end;
    RMQ<int> rmq;

    LCA(int _n = 0) : n(_n), adj(n), depth
        (n), first_occurrence(n),
        tour_start(n), tour_end(n) {}

    void add_edge(int a, int b) {
        adj[a].push_back(b);
        adj[b].push_back(a);
    }

    int tour;
    void dfs(int node, int parent = -1) {
        depth[node] = ((parent == -1) ? 0
            : depth[parent] + 1);
        first_occurrence[node] = int(euler
            .size());
        euler.push_back(node);
        tour_start[node] = tour++;

        for (int neighbor : adj[node])
            if (neighbor != parent) {
                dfs(neighbor, node);
                euler.push_back(node);
            }
        tour_end[node] = tour;
    }

    void build(int root = 0) {
        tour = 0;
        dfs(root);
        assert((int)euler.size() == 2 * n
            - 1);
        vector<int> euler_depths;
        for (int node : euler)
```

```
            euler_depths.push_back(depth[
                node]);
        rmq.build(euler_depths);
    }

    int get_lca(int a, int b) const {
        a = first_occurrence[a];
        b = first_occurrence[b];
        if (a > b) swap(a, b);
        return euler[rmq.rmq_index(a, b +
            1)];
    }

    int dist(int a, int b) const {
        return depth[a] + depth[b] - 2 *
            depth[get_lca(a, b)];
    }

    bool is_ancestor(int a, int b) const {
        return tour_start[a] <= tour_start
            [b] && tour_start[b] <
            tour_end[a];
    }

    bool on_path(int x, int a, int b)
        const {
        return (is_ancestor(x, a) ||
            is_ancestor(x, b)) &&
            is_ancestor(get_lca(a, b), x);
    }
};
```

4.7 Toposort

```
auto toposort = [&]() {
    vector<bool> visited(n + 1, false);
    stack<ll> st;

    function<void(ll)> dfs = [&](ll node)
        {
        visited[node] = true;

        for (auto &child : g[node]) {
            if (!visited[child]) {
                dfs(child);
            }
        }

        st.push(node);
    };

    for (ll i = 0; i < n; ++i) {
        if (!visited[i]) {
            dfs(i);
        }
    }

    vector<ll> ans;
    while (!st.empty()) {
        ans.push_back(st.top());
        st.pop();
    }
    return ans;
};
```

5 Maths

5.1 Combinatorics

```
const ll N = 2'00'000;
const ll MOD = 1e9 + 7; // 998244353;
ll M(ll x) { return ((x % MOD) + MOD)
    % MOD; }
ll add(ll x, ll y) { return (M(x) + M(y))
    % MOD; }
ll mul(ll x, ll y) { return (M(x) * M(y))
    % MOD; }
array<ll, N + 1> fact;
array<ll, N + 1> inv;
ll modpow(ll x, ll y, ll m) {
    if (y == 0) return 1 % m;
    ll u = modpow(x, y / 2, m);
    u = (u * u) % m;
    if (y % 2 == 1) u = (u * x) % m;
    return u;
}
// a**(p - 1) % p == 1
// a*(a**(p - 2)) % p == 1
// a**(p - 2) % p == 1 / a
ll inverse(ll a, ll m = MOD) { return
    modpow(a, m - 2, m); } //Fermats
```

```
    little theorem
void init () {
    fact [0] = 1;
    for (ll i = 1; i <= N; i++) fact [i] =
        (fact [i - 1] * i) % MOD;
    inv [N] = inverse (fact [N]);
    for (ll i = N - 1; i >= 0; --i) inv [i]
        = mul(inv[i + 1], (i + 1));

    /*
    fact [0] = inv [0] = invf [0] = fact [1] =
        inv [1] = invf [1] = 1;
    for (ll i = 2; i <= N; i++) {
        fact [i] = mul(fact [i - 1], i);
        inv [i] = M(-mul((MOD / i), inv[
            MOD % i]));
        invf [i] = mul(invf [i - 1], inv [i])
            ;
    }
    */
}

ll C(ll a, ll b) {
    if (a < b) return 0;
    return mul(mul(fact [a], inv [b]), inv [a
        - b]);
}

ll P(ll a, ll b) {
    if (a < b) return 0;
    return mul(fact [a], inv [a - b]);
}
```

5.2 Euler's totient function

```
void phi_1_to_n (int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;

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

5.3 Extended GCD

```
template <typename T>
pair<T, T> extended_eucledian(T a, T b) {
    // [a * x + b * y = g]
    // [b * x_ + (a % b) y_ = g]    => [b
        * x_ + (a - (a / b) * b) y_ = g]
        => [a * y_ + b * (x_ + (a / b) y_
        ) = g]
    // so x = y_ and y = x_ - (a / b) y_
    // base case g * 1 + 0 = g    =>
        x__ = 1 and y__ = 0
    if (a < b) swap(a, b);
    if (b == 0) return {1, 0};
    pair<T, T> mp = extended_eucledian(b,
        a % b);
    T x_ = mp. first ;
    T y_ = mp.second;
    T x = y_;
    T y = x_ - (a / b) * y_;
    return {x, y};
}

template <typename T>
pair<T, T> linear_diophantine (T a, T b, T
    c) {
    // a * x + b * y = c
    // as a and b are multiples of g, so c
        will also be a multiple of g,
        where g = gcd(a, b)
    // a * x + b * y = (k * g)
    // a * (x / k) + b * (y / k) = g
    // a * x_ + b * y_ = g
    T k = c / __gcd(a, b);
    pair<T, T> cur = extended_eucledian (a,
        b);
    T x_ = cur. first , y_ = cur.second;
    T x = k * x_;
    T y = k * y_;
    return {x, y};
}
```

5.4 Factors

```cpp
const ll MAXN = 1e14;
const ll N = ll(sqrt(MAXN)) + 10;
vector<ll> spf(N + 1);
vector<ll> pr;
void fill() {
    for (ll i = 2; i <= N; ++i) {
        if (spf[i] == 0) spf[i] = i, pr.push_back(i);
        for (ll j = 0; j < (ll)pr.size() && pr[j] <= spf[i] && i * pr[j] <= N; ++j)
            spf[i * pr[j]] = pr[j];
    }
}

// as sieve can run upto 1e7 easily in 1 sec, we can factorise upto 1e14
vector<ll> factorise(const ll &n) {
    vector<ll> res;
    ll temp = n;
    // if element is greater than current prime then break
    for (ll i = 0; i < ll(pr.size()); ++i) {
        if (temp < pr[i]) break;
        while (temp % pr[i] == 0) {
            res.push_back(pr[i]);
            temp /= pr[i];
        }
    }
    if (temp > 1) res.push_back(temp);
    return res;
}

vector<pair<ll, ll>> factorise_pair(const ll &n) {
    vector<pair<ll, ll>> C;
    ll temp = n, cnt = 0;
    // if element is greater than current prime then break
    for (ll i = 0; i < ll(pr.size()); ++i) {
        cnt = 0;
        if (temp < pr[i]) break;
        while (temp % pr[i] == 0) {
            cnt++;
            temp /= pr[i];
        }
        if (cnt) C.push_back({pr[i], cnt});
    }
    if (temp > 1) C.push_back({temp, 1});
    return C;
}

/*
n = (p1 ** a1) * (p2 ** a2) ..
count_of_divisors = (a1 + 1) * (a2 + 1) ..
sum_of_divisors = (1 + p1 + p1^2 + p1^3.. ) * ( ... ) ...
                = (p1^(a1 + 1) - 1)/(p1 - 1) ...
*/

// Returns the number of divisors of n
ll count_of_divisors(const ll &n) {
    vector<pair<ll, ll>> C = factorise_pair(n);
    ll P = 1;
    for (auto &[x, y] : C) P *= (y + 1);
    return P;
}

// Returns the sum of divisors of n
ll sum_of_divisors(const ll &n) {
    vector<pair<ll, ll>> C = factorise_pair(n);
    ll P = 1;
    for (auto &[x, y] : C) {
        ll t = 1, cnt = y + 1;
        while (cnt--) t *= x;
        P *= ((t - 1) / (x - 1));
    }
    return P;
}

// O(1) if n <= N and O(sz(pr)) if n >= N && n <= MAXN
bool isPrime(const ll &n) {
    assert(n <= MAXN);
    if (n <= N) return (spf[n] == n);
```

```cpp
    if (binary_search(pr.begin(), pr.end(), n)) return true;
    for (ll i = 0; i < ll(pr.size()); ++i)
        if (n % pr[i] == 0) return false;
    return true;
}
```

5.5 Linear Sieve

```cpp
class Linear_Sieve {
  public:
    int n;
    vector<int> factor;
    vector<int> primes;

    Linear_Sieve(int N) {
        n = N;
        factor.resize(n + 1);
    }

    void build() {
        for (i64 i = 2; i <= n; ++i) {
            if (factor[i] == 0) {
                factor[i] = i;
                primes.push_back(i);
            }
            for (i64 j = 0; i * primes[j] <= n; ++j) {
                factor[i * primes[j]] = primes[j];
                if (primes[j] == factor[i]) {
                    break;
                }
            }
        }
    }
};
```

5.6 Matrix

```cpp
// https://codeforces.com/gym/447639/problem/G

struct Matrix {
    static const ll M = MOD;
    static const ll SQMOD = M * M;
    static ll const N = 10;
    ll mat[N][N];
    ll n, m;
    Matrix(ll _n = N, ll _m = N, ll val = 0) : n(_n), m(_m) {
        for (ll i = 0; i < n; ++i)
            for (ll j = 0; j < m; ++j)
                mat[i][j] = val;
    }
    Matrix(const vector<vector<ll>> &&other) {
        n = ll(other.size());
        m = ll(other[0].size());
        for (ll i = 0; i < n; ++i)
            for (ll j = 0; j < m; ++j)
                mat[i][j] = other[i][j];
    }
    Matrix &operator=(const vector<vector<ll>> &&other) {
        return *this = Matrix(forward<decltype(other)>(other));
    }
    ll *operator[](ll r) { return mat[r]; }
    const ll *operator[](ll r) const { return mat[r]; }
    static Matrix unit(ll n) {
        Matrix res(n, n);
        for (ll i = 0; i < n; i++) res[i][i] = 1;
        return res;
    }
    Matrix &operator+=(const Matrix &rhs) {
        assert(n == rhs.n && m == rhs.m);
        for (ll i = 0; i < n; ++i)
            for (ll j = 0; j < m; ++j) {
                mat[i][j] += rhs[i][j];
                if (mat[i][j] >= M) mat[i][j] -= M;
            }
        return *this;
    }
```

```
Matrix operator+(const Matrix &rhs)
    const {
    Matrix lhs(*this);
    return lhs += rhs;
}
friend Matrix operator*(const Matrix
    &A, const Matrix &B) {
    assert(A.m == B.n);
    Matrix res(A.n, B.m);
    for (ll i = 0; i < res.n; i++)
        for (ll j = 0; j < res.m; j++)
            {
            ll sum = 0LL;
            for (ll k = 0; k < A.m; k
                ++) {
                sum += A[i][k] * B[k][
                    j];
                if (sum >= SQMOD)
                    sum -= SQMOD;
            }
            res[i][j] = (sum % M);
        }
    return res;
}
friend Matrix power(Matrix base, long
    long ex) {
    assert(base.n == base.m);
    Matrix res = Matrix::unit(base.n);
    while (ex > 0) {
        if (ex & 1) res = res * base;
        base = base * base;
        ex >>= 1;
    }
    return res;
}
friend string to_string(const Matrix
    &a) {
    string res = "\n";
    for (ll i = 0; i < a.n; ++i) {
        res += '{';
        for (ll j = 0; j < a.m; ++j) {
            res += std::to_string(a.
                mat[i][j]);
            if (j != a.m - 1) res += "
                , ";
        }
        res += "}\n";
    }
    res.append("\n");
    return res;
}
Matrix &operator*=(const Matrix &rhs)
    { return *this = *this * rhs; }
};

void test() {
    /*
        [fn fn-1 fn-2 gn gn-1 gn-2] = [fn
            -1 fn-2 fn-3 gn-1 gn-2 gn-3] *
            T
        [fn fn-1 fn-2 gn gn-1 gn-2] = [f2
            f1 f0 g2 g1 g0] * T^(n - 2)
    */
    ll a1, b1, c1, d1, a2, b2, c2, d2, f0,
        f1, f2, g0, g1, g2, q;
    cin >> a1 >> b1 >> c1 >> d1 >> a2
        >> b2 >> c2 >> d2 >> f0 >> f1
        >> f2 >> g0 >> g1 >> g2 >> q;
    Matrix Base = {{{f2, f1, f0, g2, g1,
        g0}}};
    Matrix Transform = {{{a1, 1, 0, 0, 0,
        0},
                         {b1, 0, 1, c2, 0,
                          0},
                         {0, 0, 0, d2, 0,
                          0},
                         {0, 0, 0, a2, 1,
                          0},
                         {c1, 0, 0, b2, 0,
                          1},
                         {d1, 0, 0, 0, 0,
                          0}}};
    while (q--) {
        ll n;
        cin >> n;
        auto Answer = Base * power(
            Transform, n - 2);
        ll fn = Answer[0][0], gn = Answer
            [0][3];
        cout << (fn + gn) % MOD << '\n'
            ;
    }
}
```

5.7 Miller Rabin

```
using u64 = uint64_t;
using u128 = __uint128_t;

u64 binpower(u64 base, u64 e, u64 mod) {
    u64 result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (u128) result * base
                % mod;
        base = (u128)base * base % mod;
        e >>= 1;
    }
    return result;
}

bool check_composite(u64 n, u64 a, u64 d,
    int s) {
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (u128)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
}
bool MillerRabin(u64 n) { // returns true
     if n is prime, else returns false.
    if (n < 2)
        return false;

    int r = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;
    }

    for (int a : {2, 3, 5, 7, 11, 13, 17,
        19, 23, 29, 31, 37}) {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true;
}
```

5.8 Mint

```
// https://github.com/naman1601/cp-
    templates/blob/main/mint.sublime-snippet
template <int Modulus = MOD>
struct Mint {
    int value;
    Mint(long long v = 0) {
        value = int(v % ll(Modulus));
        if (value < 0) value += Modulus;
    }
    Mint(long long a, long long b) : value
        (0) {
        *this += a;
        *this /= b;
    }
    friend string to_string(const Mint& a)
        { return to_string(a.value); }
    Mint& operator+=(Mint const& b) {
        value = ((value + b.value) %
            Modulus + Modulus) % Modulus;
        return *this;
    }
    Mint& operator-=(Mint const& b) {
        value = ((value - b.value) %
            Modulus + Modulus) % Modulus;
        return *this;
    }
    Mint& operator*=(Mint const& b) {
        value = (int((value * 1LL * b.
            value) % Modulus) + Modulus)
            % Modulus;
        return *this;
    }

    Mint mexp(Mint a, long long e) {
        Mint res = 1;
        while (e) {
```

```
            if (e & 1) res *= a;
            a *= a;
            e >>= 1;
        }
        return res;
    }

    Mint inverse(Mint a) { return mexp(a,
        Modulus - 2); }
    Mint& operator/=(Mint const& b) {
        return *this *= inverse(b); }
    friend Mint operator+(Mint a, Mint
        const b) { return a += b; }
    friend Mint operator-(Mint a, Mint
        const b) { return a -= b; }
    friend Mint operator-(Mint const a) {
        return 0 - a; }
    friend Mint operator*(Mint a, Mint
        const b) { return a *= b; }
    friend Mint operator/(Mint a, Mint
        const b) { return a /= b; }
    friend istream& operator>>(istream&
        istream, Mint& a) {
        long long v;
        istream >> v;
        a = v;
        return istream;
    }
    friend ostream& operator<<(ostream&
        ostream, Mint const& a) { return
        ostream << a.value; }
    friend bool operator==(Mint const& a,
        Mint const& b) { return a.value ==
        b.value; }
    friend bool operator!=(Mint const& a,
        Mint const& b) { return a.value !=
        b.value; }
};

using mint = Mint<MOD>;
const ll MAXN = 3'00'000;
vector<mint> fact(MAXN, 1), invf(MAXN,
    1);

void init() {
    for (ll i = 2; i < MAXN; i++) fact[i]
        = fact[i - 1] * i;
    invf[MAXN - 1] = 1 / fact[MAXN - 1];
    for (ll i = MAXN - 2; i >= 2; i--)
        invf[i] = invf[i + 1] * (i + 1);
}

mint C(ll a, ll b) {
    if (a < b) return mint();
    return fact[a] * invf[b] * invf[a - b
        ];
}

mint P(ll a, ll b) {
    if (a < b) return mint();
    return fact[a] * invf[a - b];
}
```

5.9 Prime Factorise

```
const ll N = 10000000;
array<ll, N + 1> spf;
vector<ll> pr;
void fill() {
    for (ll i = 2; i <= N; ++i) {
        if (spf[i] == 0) {
            spf[i] = i;
            pr.push_back(i);
        }
        for (ll j = 0; j < (ll)pr.size()
            && pr[j] <= spf[i] && i * pr[j
            ] <= N; ++j)
            spf[i * pr[j]] = pr[j];
    }
}

// Returns a vector containing all the
    prime factors of n (25 --> 5, 5)
vector<ll> prime_factorisation(ll n) {
    vector<ll> ans;
    while (n != 1) {
        ans.push_back(spf[n]);
        n /= spf[n];
    }
    return ans;
}
```

5.10 Small NCR

```
ll ncr(ll n, ll r) {
    ll p = 1, k = 1;
    if (n - r < r)
        r = n - r;
    if (r != 0) {
        while (r) {
            p *= n;
            k *= r;
            long long m = __gcd(p, k);
            p /= m;
            k /= m;
            n--;
            r--;
        }
    } else
        p = 1;
    return p;
}
```

6 Misc

6.1 Bitset

```
/*
Remember _Find_first() and _Find_next() ex
    - ist, and run in O(N/W), where W is
    word size of machine.
*/
#include <bits/stdc++.h>
using namespace std;

int main() {
    bitset<8> b(10);
    cout << b.to_string() << '\n';

    bitset<8> c("101");
    cout << c.to_ullong() << '\n';

    bitset<8> d = b | c;
    d.flip(0);
    d.count();
}
```

6.2 Coordinate Compress

```
template <class T>
vector<int> coordinateCompress(const
    vector<T> &a) {
    int n = int(a.size());

    vector<pair<T, int>> v(n);
    for (int i = 0; i < n; ++i) v[i] = {a[
        i], i};
    sort(v.begin(), v.end());
    vector<int> res(n);
    int curr = 0;
    for (int i = 0; i < n; ++i) {
        if (i == 0 || v[i].first != v[i -
            1].first) {
            ++curr;
        }
        res[v[i].second] = curr;
    }
    return res;
}
```

6.3 Enumerate submasks of mask

```
// O(3^n)
for (int m = 0; m < (1 << n); ++m)
    for (int s = m; s; s = (s - 1) & m)
//    ... s and m ...
```

6.4 Hash Pair

```
struct hash_pair {
    template <class T1, class T2>
    size_t operator()(const pair<T1, T2
        >& p) const{
        auto hash1 = hash<T1>{}(p.first);
        auto hash2 = hash<T2>{}(p.second
            );
        return hash1 ^ hash2;
    }
};
```

6.5 PBDS

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T>
using o_set = tree<T, null_type, less<T>,
    rb_tree_tag,
    tree_order_statistics_node_update>;
template <typename T>
using o_multiset = tree<T, null_type,
    less_equal<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
// member functions :
// 1. order_of_key(k) : number of elements
    strictly lesser than k
// 2. find_by_order(k) : k-th element in
    the set
template <class key, class value, class
    cmp = std::less<key>>
using o_map = __gnu_pbds::tree<key, value,
    cmp, __gnu_pbds::rb_tree_tag,
    __gnu_pbds::
    tree_order_statistics_node_update>;
```

6.6 Pragmas

```
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,
    popcnt")
```

6.7 Safe Unordered Map

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x)
        {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0
            xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0
            x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    size_t operator()(uint64_t x) const {
        static const uint64_t
            FIXED_RANDOM = chrono::
            steady_clock::now().
            time_since_epoch().count();
        return splitmix64(x +
            FIXED_RANDOM);
    }
};
template <typename T1, typename T2> //
    Key should be integer type
using safe_map = unordered_map<T1, T2,
    custom_hash>;
```

7 Stress Testing

7.1 Test Generator

```
mt19937 rng(uint32_t(chrono::steady_clock
    ::now().time_since_epoch().count()));
#define uid(a, b) uniform_int_distribution
    <ll>(a, b)(rng)

string alphabet = "
    abcdefghijklmnopqrstuvwxyz";
string randomString(ll length) {
    string res = "";
    for (ll i = 0; i < length; ++i) {
        ll choice = uid(0, int(alphabet.
            length()) - 1);
        res += alphabet[choice];
    }
    return res;
}
```

7.2 built[dot]sh

```
#!/usr/bin/env bash

g++ -std=c++20 -Wshadow -Wextra -Wall -
    Wl,-ld_classic -Wconversion $1.cpp -o
    $1
```

7.3 stress[dot]sh

```
#!/usr/bin/env bash

# exit immediately on error
```

```
set -e

if [ "$1" == "-h" ]; then
  echo "Usage: stress [solA - wrong] [solB
      ] [gen] [numTests]"
  echo "Runs solutionA and solutionB
      against test cases output by
      generator and outputs a test on
      which they give different results"
  exit 0
fi

build.sh $1
build.sh $2
build.sh $3

for ((testNum=0;testNum<$4;testNum++));
    do
  ./$3 > input
  ./$2 < input > outSlow
  ./$1 < input > outWrong
  diff outSlow outWrong > /dev/null ||
      break
  echo "Passed Test $testNum"
done

if !(diff outWrong outSlow > /dev/null);
    then
  echo ""
  echo "WA on following test:"
  cat input
  echo "Your answer:"
  cat outWrong
  echo "Correct answer:"
  cat outSlow
  echo ""
  exit
fi

echo ""
echo Passed $4 tests
```

8 Strings

8.1 KMP and Z

```
class Matching {
  public:
    vector<int> LPS, Z, PAL;
    int n;

    Matching(int _n) : n(_n) {
        LPS.resize(n);
        Z.resize(n), PAL.resize(n);
        for (int i = 0; i < n; i++) LPS[i]
            = Z[i] = PAL[i] = 0;
    }

    void kmp(string s) {
        int i = 1, len = 0;
        while (i < n) {
            if (s[i] == s[len]) {
                len++;
                LPS[i] = len;
                i++;
            } else {
                if (len > 0)
                    len = LPS[len - 1];
                else
                    LPS[i] = 0, i++;
            }
        }
    }

    void Z_(string s) {
        int L = 0, R = 0;
        for (int i = 1; i < s.size(); i++)
            {
            if (i <= R) Z[i] = min(R - i +
                1, Z[i - L]);
            while (i + Z[i] < s.size()
                && s[Z[i]] == s[i + Z[i]])
                Z[i]++;
            if (i + Z[i] - 1 > R) L = i,
                R = i + Z[i] - 1;
        }
    }

    void Manacher(string s) {
        int r = 0, c = 0;
        for (int i = 1; i < n; i++) {
            int mirror = c - (i - c);
```

```cpp
                if (i < r) PAL[i] = min(PAL[mirror], r - i);
                while (i + PAL[i] + 1 < s.size() && i - PAL[i] - 1 >= 0 && s[i + PAL[i] + 1] == s[i - PAL[i] - 1]) PAL[i]++;
                if (i + PAL[i] > r) c = i, r = i + PAL[i];
            }
        }
};
```

8.2 Manachers Algorithm

```cpp
// https://www.youtube.com/watch?v=06QIlUBLTz4 and https://cp-algorithms.com/string/manacher.html
// TC: O(n)
vector<int> manacher(const string &s) {
    if (s.empty()) return {};

    string t = "@";
    for (auto &ch : s) {
        t += '#';
        t += ch;
    }
    t += "#$";

    int n = int(t.size());
    vector<int> p(n, 0);
    int mirror_index = 0, right_border = 0;
    for (int i = 1; i <= n - 2; ++i) {
        int opposite_index = mirror_index - (i - mirror_index);

        // kickstart but cannot go past right border
        if (i < right_border)
            p[i] = min(p[opposite_index], right_border - i);

        while (t[i + p[i] + 1] == t[i - p[i] - 1])
            ++p[i];

        if (i + p[i] > right_border) {
            // update border and mirror
            mirror_index = i;
            right_border = i + p[i];
        }
    }

    return vector<int>(begin(p) + 2, end(p) - 2);
}
```

8.3 Rolling Hash

```cpp
int count_unique_substrings(string const& s) {
    int n = s.size();

    const int p = 31;
    const int m = 1e9 + 9;
    vector<long long> p_pow(n);
    p_pow[0] = 1;
    for (int i = 1; i < n; i++)
        p_pow[i] = (p_pow[i - 1] * p) % m;

    vector<long long> h(n + 1, 0);
    for (int i = 0; i < n; i++)
        h[i + 1] = (h[i] + (s[i] - 'a' + 1) * p_pow[i]) % m;

    int cnt = 0;
    for (int l = 1; l <= n; l++) {
        unordered_set<long long> hs;
        for (int i = 0; i <= n - l; i++) {
            long long cur_h = (h[i + l] + m - h[i]) % m;
            cur_h = (cur_h * p_pow[n - i - 1]) % m;
            hs.insert(cur_h);
        }
        cnt += hs.size();
    }
    return cnt;
}
```