

Contents

1. Contest	1
2. Data Structures	2
3. Graph	4
4. Number Theory	9
5. Strings	12

1. Contest

1.1. rng.h

```
mt19937_64
rng(chrono::steady_clock::now().time_since_epoch().count());

int random(int a, int b) {
    if (a > b)
        return 0;
    return a + rng() % (b - a + 1);
}
double random_double(double a, double b) {
    return a + (b - a) * (rng() / (double)rng.max());
}
```

1.2. template.h

```
#include <bits/stdc++.h>

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
typedef tree<int, null_type, less_equal<int>, rb_tree_tag,
            tree_order_statistics_node_update>
            ordered_multiset;
typedef tree<int, null_type, less<int>, rb_tree_tag,
            tree_order_statistics_node_update>
            ordered_set;
```

```
// find_by_order returns iterator to kth largest (0-indexed)
// order_of_key returns number of elements strictly less than
// given value -->
// basically index (0-indexed) for multiset, to erase use
// upper_bound.
// Upper_bound lower_bound exchange their roles
#define ONLINE_JUDGE
#ifndef ONLINE_JUDGE
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#else
#define db(x) ((void)0)
#define dbs(x) ((void)0)
#endif

#define int long long
#define fast()
\   ios_base::sync_with_stdio(0);
\   cin.tie(NULL);
\   cout.tie(NULL);
#define fr(i, a, b) for (int i = (a); i < (int)(b); ++i)
#define pb push_back
#define prDouble(x) cout << fixed << setprecision(10) << x
int M = 1e9 + 7;
#define all(x) x.begin(), x.end()
#define allr(x) x.rbegin(), x.rend()
#define sz(x) (int)x.size()
void solve() {}
signed main() {
    fast();
    int t = 1;
    cin >> t;
    while (t--) {
```

```

    solve();
}
return 0;
}

```

2. Data Structures

2.1. LazySegTree.h

```

template <typename T, typename U> struct seg_tree_lazy {
    int S, H;
    T zero;
    vector<T> value;
    U noop;
    vector<bool> dirty;
    vector<U> prop;
    seg_tree_lazy(int _S, T _zero = T(), U _noop = U()) {
        zero = _zero, noop = _noop;
        for (S = 1, H = 1; S < _S;)
            S *= 2, H++;
        value.resize(2 * S, zero);
        dirty.resize(2 * S, false);
        prop.resize(2 * S, noop);
    }
    void set_leaves(vector<T> &leaves) {
        copy(leaves.begin(), leaves.end(), value.begin() + S);
        for (int i = S - 1; i > 0; i--)
            value[i] = value[2 * i] + value[2 * i + 1];
    }
    void apply(int i, U &update) {
        value[i] = update(value[i]);
        if (i < S) {
            prop[i] = prop[i] + update;
            dirty[i] = true;
        }
    }
    void rebuild(int i) {

```

```

        for (int l = i / 2; l; l /= 2) {
            T combined = value[2 * l] + value[2 * l + 1];
            value[l] = prop[l](combined);
        }
    }
    void propagate(int i) {
        for (int h = H; h > 0; h--) {
            int l = i >> h;
            if (dirty[l]) {
                apply(2 * l, prop[l]);
                apply(2 * l + 1, prop[l]);
                prop[l] = noop;
                dirty[l] = false;
            }
        }
    }
    void upd(int i, int j, U update) {
        i += S, j += S;
        propagate(i), propagate(j);
        for (int l = i, r = j; l <= r; l /= 2, r /= 2) {
            if ((l & 1) == 1)
                apply(l++, update);
            if ((r & 1) == 0)
                apply(r--, update);
        }
        rebuild(i), rebuild(j);
    }
    T query(int i, int j) {
        i += S, j += S;
        propagate(i), propagate(j);
        T res_left = zero, res_right = zero;
        for (; i <= j; i /= 2, j /= 2) {
            if ((i & 1) == 1)
                res_left = res_left + value[i++];
            if ((j & 1) == 0)
                res_right = value[j--] + res_right;
        }
    }
}

```

```

    }
    return res_left + res_right;
}
};

struct node {
    int sum, width;
    node operator+(const node &n) {
        // change 1
        return {sum + n.sum, width + n.width};
    }
};

struct update {
    bool type; // 0 for add, 1 for reset
    int value;
    node operator()(const node &n) {
        // change 2
        if (type)
            return {n.width * value, n.width};
        else
            return {n.sum + n.width * value, n.width};
    }
    update operator+(const update &u) {
        // change 3
        if (u.type)
            return u;
        return {type, value + u.value};
    }
};

// Example Initialization
// seg_tree_lazy<node, update> lst(size, {0, 1}, {0, 0});
// lst.set_leaves(leaves);
// lst upd(l, r, {0, value});
// auto result = lst.query(l, r);

```

2.2. Mos.h

```

int BLOCK = DO_NOT_FORGET_TO_CHANGE_THIS;
struct Query {
    int l, r, id;
    Query(int _l, int _r, int _id) : l(_l), r(_r), id(_id) {}
    bool operator<(Query &o) {
        int mblock = l / BLOCK, oblock = o.l / BLOCK;
        return (mblock < oblock) or
               (mblock == oblock and mblock % 2 == 0 and r < o.r)
        or
               (mblock == oblock and mblock % 2 == 1 and r > o.r);
    };
};

void solve() {
    vector<Query> queries;
    queries.reserve(q);
    for (int i = 0; i < q; i++) {
        int l, r;
        cin >> l >> r;
        l--, r--;
        queries.emplace_back(l, r, i);
    }
    sort(all(queries));
    int ans = 0;
    auto add = [&](int v) {};
    auto rem = [&](int v) {};
    vector<int> out(q); // Change out type if necessary
    int cur_l = 0, cur_r = -1;
    for (auto &[l, r, id] : queries) {
        while (cur_l > l)
            add(--cur_l);
        while (cur_l < l)
            rem(cur_l++);
        while (cur_r < r)
            add(++cur_r);
        while (cur_r > r)
            rem(cur_r--);
        ans += out[id];
    }
}
```

```

        rem(cur_r--);
        out[id] = ans;
    }
}

```

2.3. SegTree.h

```

struct segtree {
    typedef int T;
    // for max segtree, set unit = INT_MIN and f(a,b) = max(a,b)
    static constexpr T unit = 0;      // identity for sum
    T f(T a, T b) { return a + b; } // (any associative fn)
    vector<T> s;
    int n;
    segtree(int n = 0, T def = unit) : s(2 * n, def), n(n) {}
    void update(int pos, T val) {
        for (s[pos += n] = val; pos /= 2;)
            s[pos] = f(s[pos * 2], s[pos * 2 + 1]);
    }
    T query(int b, int e) { // query [b, e)
        T ra = unit, rb = unit;
        for (b += n, e += n; b < e; b /= 2, e /= 2) {
            if (b % 2)
                ra = f(ra, s[b++]);
            if (e % 2)
                rb = f(s[--e], rb);
        }
        return f(ra, rb);
    }
};

```

2.4. range_update_tree.h

```

template <typename T, typename F> struct RangeUpdateTree {
    int n;
    vector<T> tree;
    T identity;
    F merge;

```

```

RangeUpdateTree(const vector<T> &arr, T id, F _m)
    : n((int)arr.size()), tree(2 * n), identity(id),
merge(_m) {
    for (int i = 0; i < n; i++)
        tree[n + i] = arr[i];
    for (int i = n - 1; i >= 1; i--)
        tree[i] = merge(tree[2 * i], tree[2 * i + 1]);
}
void update(int l, int r, T value) {
    assert(l >= 0 && r < n && l <= r);
    for (l += n, r += n; l <= r; l >= 1, r >= 1) {
        if (l & 1)
            tree[l] = merge(value, tree[l]), l++;
        if (!(r & 1))
            tree[r] = merge(value, tree[r]), r--;
    }
    if (l == r)
        tree[l] = merge(value, tree[l]);
}
T query(int v) {
    T res = tree[v += n];
    for (; v > 1; v >= 1)
        res = merge(res, tree[v > 1]);
    return res;
};
// ex: RangeUpdateTree<int, decltype(join)> v(vi (n, 1e9), 1e9,
join); use auto
// func for join

```

3. Graph

3.1. HLD.h

```

struct HLD {
    int n, timer = 0;
    vi top, tin, p, sub;

```

```

HLD(vvi &adj) : n(sz(adj)), top(n), tin(n), p(n, -1), sub(n,
1) {
    vi ord(n + 1);
    for (int i = 0, t = 0, v = ord[i]; i < n; v = ord[+i])
        for (auto &to : adj[v])
            if (to != p[v])
                p[to] = v, ord[+t] = to;
    for (int i = n - 1, v = ord[i]; i > 0; v = ord[-i])
        sub[p[v]] += sub[v];
    for (int v = 0; v < n; v++)
        if (sz(adj[v]))
            iter_swap(begin(adj[v]), max_element(all(adj[v]), [&
(int a, int b) {
    return make_pair(a != p[v], sub[a]) <
           make_pair(b != p[v], sub[b]);
})));
    function<void(int)> dfs = [&](int v) {
        tin[v] = timer++;
        for (auto &to : adj[v])
            if (to != p[v]) {
                top[to] = (to == adj[v][0] ? top[v] : to);
                dfs(to);
            }
    };
    dfs(0);
}
int lca(int u, int v) {
    return process(u, v, [](){}(int, int) {});
}
template <class B> int process(int a, int b, B op, bool
ignore_lca = false) {
    for (int v;; op(tin[v], tin[b]), b = p[v]) {
        if (tin[a] > tin[b])
            swap(a, b);
        if ((v = top[b]) == top[a])
            break;
    }
    if (int l = tin[a] + ignore_lca, r = tin[b]; l <= r)
        op(l, r);
    return a;
}
template <class B> void subtree(int v, B op, bool ignore_lca
= false) {
    if (sub[v] > 1 or !ignore_lca)
        op(tin[v] + ignore_lca, tin[v] + sub[v] - 1);
}
};



### 3.2. KthAnc.h


// O(log n) LCA with Kth anc
struct LCA {
    int n;
    vvi &adjLists;
    int lg;
    vvi up;
    vi depth;
    LCA(vvi &_adjLists, int root = 0) : n(sz(_adjLists)),
adjLists(_adjLists) {
        lg = 1;
        int pw = 1;
        while (pw <= n)
            pw <<= 1, lg++;
        // lg = 20
        up = vvi(n, vi(lg));
        depth.assign(n, -1);
        function<void(int, int)> parentDFS = [&](int from, int
parent) {
            depth[from] = depth[parent] + 1;
            up[from][0] = parent;
            for (auto to : adjLists[from]) {
                if (to == parent)
                    continue;

```

```

        parentDFS(to, from);
    }
};

parentDFS(root, root);
for (int j = 1; j < lg; j++) {
    for (int i = 0; i < n; i++) {
        up[i][j] = up[up[i][j - 1]][j - 1];
    }
}
int kthAnc(int v, int k) {
    int ret = v;
    int pw = 0;
    while (k) {
        if (k & 1)
            ret = up[ret][pw];
        k >>= 1;
        pw++;
    }
    return ret;
}
int lca(int u, int v) {
    if (depth[u] > depth[v])
        swap(u, v);
    v = kthAnc(v, depth[v] - depth[u]);
    if (u == v)
        return v;
    while (up[u][0] != up[v][0]) {
        int i = 0;
        for (; i < lg - 1; i++) {
            if (up[u][i + 1] == up[v][i + 1])
                break;
        }
        u = up[u][i], v = up[v][i];
    }
    return up[u][0];
}

```

```

    };
    int dist(int u, int v) { return depth[u] + depth[v] - 2 * depth[lca(u, v)]; }
};



### 3.3. LCA.h


// O(1) LCA
struct LCA {
    int T = 0;
    vi st, path, ret;
    vi en, d;
    RMQ<int> rmq;
    LCA(vector<vi> &C)
        : st(sz(C)), en(sz(C)), d(sz(C)), rmq((dfs(C, 0, -1),
ret)) {}
    void dfs(vvi &adj, int v, int par) {
        st[v] = T++;
        for (auto to : adj[v])
            if (to != par) {
                path.pb(v), ret.pb(st[v]);
                d[to] = d[v] + 1;
                dfs(adj, to, v);
            }
        en[v] = T - 1;
    }
    bool anc(int p, int c) { return st[p] <= st[c] and en[p] >=
en[c]; }
    int lca(int a, int b) {
        if (a == b)
            return a;
        tie(a, b) = minmax(st[a], st[b]);
        return path[rmq.query(a, b - 1)];
    }
    int dist(int a, int b) { return d[a] + d[b] - 2 * d[lca(a,
b)]; }
};

```

3.4. bellman.h

```
bool bellman_ford(int n, int src, vector<vector<pair<int, int>>& adj,
                   vector<long long> &dist) {
    const long long INF = 1e18;
    dist.assign(n, INF);
    dist[src] = 0;
    for (int i = 0; i < n - 1; i++) {
        for (int u = 0; u < n; u++) {
            if (dist[u] == INF)
                continue;
            for (auto &p : adj[u]) {
                int v = p.first, w = p.second;
                if (dist[u] + w < dist[v])
                    dist[v] = dist[u] + w;
            }
        }
    }
    for (int u = 0; u < n; u++) {
        if (dist[u] == INF)
            continue;
        for (auto &p : adj[u]) {
            int v = p.first, w = p.second;
            if (dist[u] + w < dist[v])
                return false;
        }
    }
    return true;
}
```

3.5. bridges.h

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
```

```
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p)
            continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
        }
    }
}

void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}

// ARTICULATION POINTS:
int n;
vector<vector<int>> adj;
vector<bool> visited;
vector<int> tin, low;
int timer;
```

```

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

void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}

// arya bridges
void findBridges_dfs(int u, int p, int &time,
vector<vector<int>> &adj,
vector<int> &disc, vector<int> &low,
vector<pair<int, int>> &bridges) {
    disc[u] = low[u] = time++;
    for (int v : adj[u]) {
        if (v == p)
            continue;
        if (disc[v] != -1) {
            low[u] = min(low[u], disc[v]);
        } else {
            findBridges_dfs(v, u, time, adj, disc, low, bridges);
            low[u] = min(low[u], low[v]);
            if (low[v] > disc[u])
                bridges.push_back({u, v});
        }
    }
}

vector<pair<int, int>> findBridges(int n, vector<vector<int>>
&adj) {
    vector<int> disc(n, -1), low(n, -1);
    vector<pair<int, int>> bridges;
    int time = 0;

    for (int i = 0; i < n; ++i) {
        if (disc[i] == -1) {
            findBridges_dfs(i, -1, time, adj, disc, low, bridges);
        }
    }
    return bridges;
}

3.6. dijkstra.h
const int INF = 1000000000;
vector<vector<pair<int, int>>> adj;

```

```

void dijkstra(int s, vector<int> &d, vector<int> &p) {
    int n = adj.size();
    d.assign(n, INF);
    p.assign(n, -1);
    vector<bool> u(n, false);

    d[s] = 0;
    for (int i = 0; i < n; i++) {
        int v = -1;
        for (int j = 0; j < n; j++) {
            if (!u[j] && (v == -1 || d[j] < d[v]))
                v = j;
        }

        if (d[v] == INF)
            break;

        u[v] = true;
        for (auto edge : adj[v]) {
            int to = edge.first;
            int len = edge.second;

            if (d[v] + len < d[to]) {
                d[to] = d[v] + len;
                p[to] = v;
            }
        }
    }
}

```

3.7. floyd_washall.h

```

const long long INF = (long long)1e18;
bool floyd_marshall(int n, vector<vector<long long>> &dist) {
    // initialise dist with edge weights, INF if no edge exists
    for (int k = 0; k < n; k++)
        for (int i = 0; i < n; i++)

```

```

            for (int j = 0; j < n; j++)
                if (dist[i][k] < INF && dist[k][j] < INF)
                    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k]
[j]);

            for (int i = 0; i < n; i++)
                if (dist[i][i] < 0)
                    return true;
            return false;
        }
    }
}

```

4. Number Theory

4.1. ModularArithmetic.h

```

int add(int x, int y, int m = M) {
    int ret = (x + y) % m;
    if (ret < 0)
        ret += m;
    return ret;
}

int mult(int x, int y, int m = M) {
    int ret = (x * y) % m;
    if (ret < 0)
        ret += m;
    return ret;
}

int pw(int a, int b, int m = M) {
    int ret = 1;
    int p = a;
    while (b) {
        if (b & 1)
            ret = mult(ret, p, m);
        b >>= 1;
        p = mult(p, p, m);
    }
    return ret;
}

```

```

}

#define LL int
const long long mod = 1e9 + 7;

int euclid(int a, int b, int &x, int &y) {
    if (!b)
        return x = 1, y = 0, a;
    int d = euclid(b, a % b, y, x);
    return y -= a / b * x, d;
}

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

LL mod_mul(LL a, LL b) {
    a = a % mod;
    b = b % mod;
    return (((a * b) % mod) + mod) % mod;
}

LL mod_add(LL a, LL b) {
    a = a % mod;
    b = b % mod;
    return (((a + b) % mod) + mod) % mod;
}

const int MX = 5e5 + 1;

vector<int> inv(MX + 1), fci(MX + 1), fc(MX + 1);
const int Mod = 1e9 + 7;

void Inverses() {
    inv[1] = 1;
    for (int i = 2; i <= MX; i++) {
        inv[i] = Mod - Mod / i * inv[Mod % i] % Mod;
    }
}

void Factorials() {
    fc[0] = fc[1] = 1;
    for (int i = 2; i <= MX; i++) {
        fc[i] = fc[i - 1] * i % Mod;
    }
}

void InverseFactorials() {
    Inverses();
    Factorials();
    fci[1] = fci[0] = 1;
    for (int i = 2; i <= MX; i++) {
        fci[i] = fci[i - 1] * inv[i] % Mod;
    }
}

int nck(int num, int k) {
    if (num < 0) {
        return 0;
    }
    if (k < 0) {
        return 0;
    }
    if (num < k) {
        return 0;
    } else {

```

```

        return fc[num] * fci[k] % Mod * fci[num - k] % Mod;
    }

int BinExpItermod(int a, int b) {
    int ans = 1;
    while (b > 0) {
        if (b & 1) {
            ans = (ans * a) % mod;
        }
        a = (a * a) % mod;
        b = b >> 1;
    }
    return ans;
}

```

4.2. bit_bns.h

```

// --- Bit Binary Search in o(log(n)) ---
const int M = 20 const int N = 1 << M

int
lower_bound(int val) {

int ans = 0, sum = 0;
for (int i = M - 1; i >= 0; i--) {
    int x = ans + (1 << i);
    if (sum + bit[x] < val)
        ans = x, sum += bit[x];
}

return ans + 1;
}

```

4.3. matrix_expo.h

```

int **matrixmul(int **matrix1, int **matrix2) {
    int **matrix3 = new int *[2];
    for (int i = 0; i < 2; i++)

```

```

        matrix3[i] = new int[2];

    matrix3[0][0] =
        (matrix1[0][0] * matrix2[0][0]) + (matrix1[0][1] *
matrix2[1][0]);
    matrix3[0][1] =
        (matrix1[0][0] * matrix2[0][1]) + (matrix1[0][1] *
matrix2[1][1]);
    matrix3[1][0] =
        (matrix1[1][0] * matrix2[0][0]) + (matrix1[1][1] *
matrix2[1][0]);
    matrix3[1][1] =
        (matrix1[1][0] * matrix2[0][1]) + (matrix1[1][1] *
matrix2[1][1]);

    matrix3[0][0] %= M;
    matrix3[1][0] %= M;
    matrix3[0][1] %= M;
    matrix3[1][1] %= M;

    return matrix3;
}

int **matrixexpo(int **matrix, int n, int **ans) {
    while (n > 0) {
        if (n % 2 == 1)
            ans = matrixmul(ans, matrix);
        matrix = matrixmul(matrix, matrix);
        n /= 2;
    }
    return ans;
}

```

4.4. some_dp.h

```

// LIS
int lis(vector<int> const &a) {
    int n = a.size();

```

```

const int INF = 1e9;
vector<int> d(n + 1, INF);
d[0] = -INF;

for (int i = 0; i < n; i++) {
    int l = upper_bound(d.begin(), d.end(), a[i]) - d.begin();
    if (d[l - 1] < a[i] && a[i] < d[l])
        d[l] = a[i];
}

int ans = 0;
for (int l = 0; l <= n; l++) {
    if (d[l] < INF)
        ans = l;
}
return ans;
}
// or segtree lol

```

4.5. spf.h

```

int MX = 1e7 + 1;
vi spf(MX + 1, INT32_MAX);
vector<int> is_prime(MX + 1, 1);
void sieve(int n = MX) {
    is_prime[0] = is_prime[1] = 0;
    int cnt = 1;
    for (int i = 2; i <= n; i++) {
        if (is_prime[i]) {
            for (int j = i * i; j <= n; j += i) {
                is_prime[j] = 0;
                spf[j] = min(i, spf[j]);
            }
            is_prime[i] = cnt;
            cnt++;
        }
    }
}

```

```

    return;
}
```

5. Strings

5.1. Trie.h

```

class TrieNode {
public:
    unordered_map<char, TrieNode *> children;
    bool isEndOfWord;

    TrieNode() : isEndOfWord(false) {}
};

class Trie {
private:
    TrieNode *root;

public:
    Trie() { root = new TrieNode(); }
    void insert(const string &word) {
        TrieNode *node = root;
        for (char ch : word) {
            if (node->children.find(ch) == node->children.end()) {
                node->children[ch] = new TrieNode();
            }
            node = node->children[ch];
        }
        node->isEndOfWord = true;
    }
    bool search(const string &word) {
        TrieNode *node = root;
        for (char ch : word) {
            if (node->children.find(ch) == node->children.end()) {
                return false;
            }
            node = node->children[ch];
        }
    }
}

```

```

    }
    return node->isEndOfWord;
}
bool startsWith(const string &prefix) {
    TrieNode *node = root;
    for (char ch : prefix) {
        if (node->children.find(ch) == node->children.end()) {
            return false;
        }
        node = node->children[ch];
    }
    return true;
};

```

5.2. hash.h

```

template <int MOD, int P> struct RH {
    // using H1 = RH<1000000007, 91138233>;
    // using H2 = RH<1000000009, 97266353>;
    vector<long long> h, p;
    RH(const string &s) {
        int n = s.size();
        h.resize(n + 1, 0);
        p.resize(n + 1, 0);
        p[0] = 1;
        for (int i = 0; i < n; i++) {
            h[i + 1] = (h[i] * P + s[i]) % MOD;
            p[i + 1] = p[i] * P % MOD;
        }
    }
    long long get(int l, int r) { // [l,r]
        long long res = (h[r + 1] - h[l] * p[r - l + 1]) % MOD;
        return res < 0 ? res + MOD : res;
    }
};

```

5.3. kmp.h

```

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

vector<int> KMP(string text, string pattern) {
    string s = pattern + "#" + text;
    vector<int> pi = prefix_function(s);
    vector<int> matches;
    int p = pattern.length();

    for (int i = 0; i < s.length(); i++) {
        if (pi[i] == p) {
            int match_pos = i - 2 * p;
            matches.push_back(match_pos); // 0-based index in text
        }
    }
    return matches;
}

```