

Listings

1	CODE HASHES	1	39	Row Reduce	17
2	hash.sh	1	40	Solve Linear Equations MOD	17
3	GRAPHS	1	41	Matrix Inverse	18
4	Bridges and Cuts	1	42	Euler's Totient Phi Function	18
5	Centroid	2	43	MAX FLOW	18
6	Dijkstra	3	44	Dinic	18
7	Floyd Warshall	3	45	Hungarian	19
8	HLD	3	46	Min Cost Max Flow	19
9	Hopcroft Karp	4	47	MISC	20
10	LCA	4	48	Disjoint Set	20
11	SCC	5	49	PBDS	20
12	RANGE DATA STRUCTURES	5	50	Count Rectangles	20
13	Segment Tree	5	51	Longest Increasing Subsequence	21
14	Fenwick Tree	6	52	Monotonic Stack	21
15	Sparse Table	7	53	Safe Hash	21
16	Implicit Lazy Segment Tree	7			
17	Range Updates, Point Queries	8			
18	Kth Smallest	8			
19	Number Distinct Elements	8			
20	Buckets	9			
21	Persistent Lazy Segment Tree	10			
22	Mos Algorithm	11			
23	Merge Sort Tree	11			
24	STRINGS	12			
25	Suffix Array	12			
26	Longest Common Prefix Array	13			
27	Prefix Function	13			
28	KMP	13			
29	Trie	14			
30	Longest Common Prefix Query	14			
31	MATH	15			
32	BIN EXP MOD	15			
33	Fibonacci	15			
34	Matrix Mult and Pow	15			
35	N Choose K MOD	16			
36	Partitions	16			
37	Derangements	16			
38	Prime Sieve Mobius	17			

Listing 1: CODE HASHES

Listing 2: hash.sh

```
# Hashes a file, ignoring all whitespace, comments, and includes. Use for
# verifying that code was correctly typed.

# usage:
#   chmod u+x hash.sh
#   cat <file> | ./hash.sh
# or just copy this command:
#   cat <file> | sed -r '/^#(include|pragma)/d' | cpp -dD -P -fpreprocessed -MM | tr -d
#     ↳ '[:space:]' | md5sum | cut -c-6
sed -r '/^#(include|pragma)/d' | cpp -dD -P -fpreprocessed | tr -d '[:space:]' | md5sum
#     ↳ | cut -c-6
```

Listing 3: GRAPHS

Listing 4: Bridges and Cuts

```
//cat bridges_and_cuts.h | ./hash.sh
//985365
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/biconnected_components,
#     ↳ https://judge.yosupo.jp/problem/two_edge_connected_components
//with asserts checking correctness of isBridge and isCut

//O(n+m) time & space
//2 edge cc and bcc stuff doesn't depend on each other, so delete whatever is not needed
//handles multiple edges

//example initialization of 'adj':
//for (int i = 0; i < m; i++) {
//  int u, v;
//  cin >> u >> v;
//  u--, v--;
//  adj[u].emplace_back(v, i);
//  adj[v].emplace_back(u, i);
//}

struct info {
  //2 edge connected component stuff (e.g. components split by bridge edges)
  #     ↳ https://cp-algorithms.com/graph/bridge-searching.html
  int num2EdgeCCs;
  vector<bool> isBridge; //edge id -> true iff bridge edge
  vector<int> TwoEdgeCCID; //node -> ID of 2-edge component (which are labeled 0, 1,
#     ↳ ..., 'num2EdgeCCs'-1)

  //bi-connected component stuff (e.g. components split by cut/articulation nodes)
  #     ↳ https://cp-algorithms.com/graph/cutpoints.html
  int numBCCs;
  vector<bool> isCut; //node -> true iff cut node
  vector<int> bccID; //edge id -> ID of BCC (which are labeled 0, 1, ..., 'numBCCs'-1)
};

info bridge_and_cut(const vector<vector<pair<int/*neighbor*/, int/*edge id*/>>>&
#     ↳ adj/*undirected graph*/, int m/*number of edges*/) {
```

```
//stuff for both (always keep)
int n = adj.size(), timer = 1;
vector<int> tin(n, 0);
//2 edge CC stuff (delete if not needed)
int num2EdgeCCs = 0;
vector<bool> isBridge(m, false);
vector<int> TwoEdgeCCID(n), nodeStack;
//BCC stuff (delete if not needed)
int numBCCs = 0;
vector<bool> isCut(n, false);
vector<int> bccID(m), edgeStack;
auto dfs = [&](auto&& self, int v, int pId) -> int {
  int low = tin[v] = timer++;
  int deg = 0;
  nodeStack.push_back(v);
  for (auto [to, eId] : adj[v]) {
    if (eId == pId) continue;
    if (!tin[to]) {
      edgeStack.push_back(eId);
      int lowCh = self(self, to, eId);
      if (lowCh >= tin[v]) {
        isCut[v] = true;
        while (true) {
          int edge = edgeStack.back();
          edgeStack.pop_back();
          bccID[edge] = numBCCs;
          if (edge == eId) break;
        }
        numBCCs++;
        low = min(low, lowCh);
        deg++;
      } else if (tin[to] < tin[v]) {
        edgeStack.push_back(eId);
        low = min(low, tin[to]);
      }
    }
  }
  if (pId == -1) isCut[v] = (deg > 1);
  if (tin[v] == low) {
    if (pId != -1) isBridge[pId] = true;
    while (true) {
      int node = nodeStack.back();
      nodeStack.pop_back();
      TwoEdgeCCID[node] = num2EdgeCCs;
      if (node == v) break;
    }
    num2EdgeCCs++;
  }
  return low;
};
for (int i = 0; i < n; i++) {
  if (!tin[i])
    dfs(dfs, i, -1);
}
return {num2EdgeCCs, isBridge, TwoEdgeCCID, numBCCs, isCut, bccID};
}
```

Listing 5: Centroid

```
//cat centroid.h | ./hash.sh
```

```
//28d4e6
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/frequency_table_of_tree_distance
//with asserts checking depth of tree <= log2(n)

//returns array 'par' where 'par[i]' = parent of node 'i' in centroid tree
//'par[root]' is -1
//0-based nodes
//O(n log n)

//example usage:
// vector<int> parent = getCentroidTree(adj);
// vector<vector<int>> childs(n);
// int root;
// for (int i = 0; i < n; i++) {
//     if (parent[i] == -1)
//         root = i;
//     else
//         childs[parent[i]].push_back(i);
// }
vector<int> getCentroidTree(const vector<vector<int>>& adj /*unrooted tree*/) {
    int n = adj.size();
    vector<int> sizes(n);
    vector<bool> vis(n, false);
    auto dfsSz = [&](auto&& self, int node, int par) -> void {
        sizes[node] = 1;
        for (int to : adj[node]) {
            if (to != par && !vis[to]) {
                self(self, to, node);
                sizes[node] += sizes[to];
            }
        }
    };
    auto findCentroid = [&](int node) -> int {
        dfsSz(dfsSz, node, node);
        int sizeCap = sizes[node] / 2, par = -1;
        while (true) {
            bool found = false;
            for (int to : adj[node]) {
                if (to != par && !vis[to] && sizes[to] > sizeCap) {
                    found = true;
                    par = node;
                    node = to;
                    break;
                }
            }
            if (!found) return node;
        }
    };
    vector<int> parent(n);
    auto dfs = [&](auto&& self, int node, int par) -> void {
        node = findCentroid(node);
        parent[node] = par;
        vis[node] = true;
        for (int to : adj[node]) {
            if (!vis[to])
                self(self, to, node);
        }
    };
    dfs(dfs, 0, -1);
    return parent;
}
```

```
}
```

Listing 6: Dijkstra

```
//cat dijkstra.h | ./hash.sh
//27560a
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/shortest_path

//returns array 'len' where 'len[i]' = shortest path from node 'startNode' to node i
//For example len[startNode] will always = 0

const long long INF = 1e18;

vector<long long> dijkstra(const vector<vector<pair<int, long long>>>& adj /*directed or
↳ undirected, weighted graph*/, int startNode) {
    vector<long long> len(adj.size(), INF);
    len[startNode] = 0;
    set<pair<long long /*weight*/, int /*node*/>> q;
    q.insert({0LL, startNode});
    while (!q.empty()) {
        auto it = q.begin();
        int node = it->second;
        q.erase(it);
        for (auto [to, weight] : adj[node])
            if (len[to] > weight + len[node]) {
                q.erase({len[to], to});
                len[to] = weight + len[node];
                q.insert({len[to], to});
            }
    }
    return len;
}
```

Listing 7: Floyd Warshall

```
//cat floydWarshall.h | ./hash.sh
//84799a
#pragma once
//status: not tested

/**for directed graphs only** if you initialize len[i][i] to infinity, then
//afterward floyds, len[i][i] = length of shortest cycle including node 'i'
//
//another trick: change 'len' to 2d array of *bools* where len[i][j] = true if
//there exists an edge from i -> j in initial graph. Also do:
//'len[i][j] != len[i][k] & len[k][j]'
//Then after floyds, len[i][j] = true iff there's exists some path from node
//'i' to node 'j'
//
//Changing the order of for-loops to i-j-k (instead of the current k-i-j)
//results in min-plus matrix multiplication. If adjacency matrix is M, then
//after computing M^k (with binary exponentiation), M[i][j] = min length path
//from i to j with at most k edges.

for (int k = 0; k < n; k++)
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
```

```
len[i][j] = min(len[i][j], len[i][k] + len[k][j]);
```

Listing 8: HLD

```
//cat hld.h | ./hash.sh
//ce8f71
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/lca,
    ↪ https://judge.yosupo.jp/problem/vertex_add_path_sum,
    ↪ https://judge.yosupo.jp/problem/vertex_add_subtree_sum

//source: https://codeforces.com/blog/entry/53170
//assumes a single tree, 1-based nodes is possible by passing in 'root' in range [1, n]

struct hld {
    int n;
    vector<int> Size, par, timeIn, Next;
    hld(vector<vector<int>>& adj /*single unrooted tree*/, int root) :
        n(adj.size()), Size(n, 1), par(n, root), timeIn(n), Next(n, root) {
        dfs1(root, adj);
        int Time = 0;
        dfs2(root, adj, Time);
    }
    void dfs1(int node, vector<vector<int>>& adj) {
        for (int& to : adj[node]) {
            if (to == par[node]) continue;
            par[to] = node;
            dfs1(to, adj);
            Size[node] += Size[to];
            if (Size[to] > Size[adj[node][0]] || adj[node][0] == par[node])
                swap(to, adj[node][0]);
        }
    }
    void dfs2(int node, const vector<vector<int>>& adj, int& Time) {
        timeIn[node] = Time++;
        for (int to : adj[node]) {
            if (to == par[node]) continue;
            Next[to] = (Time == timeIn[node] + 1 ? Next[node] : to);
            dfs2(to, adj, Time);
        }
    }
    // Returns intervals (of timeIn's) corresponding to the path between u and v, not
    ↪ necessarily in order
    // This can answer queries for "is some node 'x' on some path" by checking if the
    ↪ timeIn[x] is in any of these intervals
    vector<pair<int, int>> path(int u, int v) const {
        vector<pair<int, int>> res;
        for (; v = par[Next[v]] {
            if (timeIn[v] < timeIn[u]) swap(u, v);
            if (timeIn[Next[v]] <= timeIn[u]) {
                res.emplace_back(timeIn[u], timeIn[v]);
                return res;
            }
            res.emplace_back(timeIn[Next[v]], timeIn[v]);
        }
    }
    // Returns interval (of timeIn's) corresponding to the subtree of node i
    // This can answer queries for "is some node 'x' in some other node's subtree" by
    ↪ checking if timeIn[x] is in this interval
    pair<int, int> subtree(int i) const {
```

```
        return {timeIn[i], timeIn[i] + Size[i] - 1};
    }
    // Returns lca of nodes u and v
    int lca(int u, int v) const {
        for (; v = par[Next[v]] {
            if (timeIn[v] < timeIn[u]) swap(u, v);
            if (timeIn[Next[v]] <= timeIn[u]) return u;
        }
    }
};
```

Listing 9: Hopcroft Karp

```
//cat hopcroftKarp.h | ./hash.sh
//e32052
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/bipartitematching
//with asserts checking correctness of min vertex cover

//Modified from
    ↪ https://github.com/foreverbell/acm-icpc-cheat-sheet/blob/master/src/graph-algorithms/
//Worst case O(E*sqrt(V)) but faster in practice

struct match {
    /** of edges in matching (which = size of min vertex cover by König's theorem)
    int sizeOfMatching;
    //an arbitrary max matching is found. For this matching:
    //if ml[nodeLeft] == -1:
    //    'nodeLeft' is not in matching
    //else:
    //    the edge 'nodeLeft' <=> ml[nodeLeft] is in the matching
    //
    //similarly for mr with edge mr[nodeRight] <=> nodeRight in matching if
    ↪ mr[nodeRight] != -1
    //matchings stored in ml and mr are the same matching
    //provides way to check if any node is in matching
    vector<int> ml, mr;
    //an arbitrary min vertex cover is found. For this MVC: leftMVC['left node'] is true
    ↪ iff 'left node' is in the min vertex cover (same for rightMVC)
    //if leftMVC['left node'] is false, then 'left node' is in the corresponding maximal
    ↪ independent set
    vector<bool> leftMVC, rightMVC;
};

//Think of the bipartite graph as having a left side (with size lSz) and a right side
    ↪ (with size rSz).
//Nodes on left side are indexed 0,1,...,lSz-1
//Nodes on right side are indexed 0,1,...,rSz-1
//
//'adj' is like a directed adjacency list containing edges from left side -> right side:
//To initialize 'adj': For every edge nodeLeft <=> nodeRight, do:
    ↪ adj[nodeLeft].push_back(nodeRight)
match hopcroftKarp(const vector<vector<int>>& adj /*bipartite graph*/, int rSz /*number of
    ↪ nodes on right side*/) {
    int sizeOfMatching = 0, lSz = adj.size();
    vector<int> ml(lSz, -1), mr(rSz, -1);
    while (true) {
        queue<int> q;
        vector<int> level(lSz, -1);
        for (int i = 0; i < lSz; i++) {
```

```
        if (ml[i] == -1) level[i] = 0, q.push(i);
    }
    bool found = false;
    vector<bool> leftMVC(lSz, true), rightMVC(rSz, false);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        leftMVC[u] = false;
        for (int x : adj[u]) {
            rightMVC[x] = true;
            int v = mr[x];
            found |= v == -1;
            if (v != -1 && level[v] < 0) {
                level[v] = level[u] + 1;
                q.push(v);
            }
        }
    }
    if (!found) return {sizeofMatching, ml, mr, leftMVC, rightMVC};
    auto dfs = [&](auto&& self, int u) -> bool {
        for (int x : adj[u]) {
            int v = mr[x];
            if (v == -1 || (level[u] + 1 == level[v] && self(self, v))) {
                ml[u] = x;
                mr[x] = u;
                return true;
            }
        }
        level[u] = 1e9; //acts as visited array
        return false;
    };
    for (int i = 0; i < lSz; i++)
        sizeofMatching += (ml[i] == -1 && dfs(dfs, i));
}
```

Listing 10: LCA

```
//cat lca.h | ./hash.sh
//1b8562
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/lca

//https://codeforces.com/blog/entry/74847
//assumes a single tree, 1-based nodes is possible by passing in 'root' in range [1, n]

struct lca {
    int n;
    vector<int> jmp, jmpEdges, par, depth;
    vector<long long> dist;

    lca(const vector<vector<pair<int, long long>>>& adj, int root) :
        n(adj.size()), jmp(n, root), jmpEdges(n, 0), par(n, root), depth(n, 0), dist(n,
        ↪ 0LL) {
        dfs(root, adj);
    }

    void dfs(int node, const vector<vector<pair<int, long long>>>& adj) {
        for (auto [ch, w] : adj[node]) {
            if (ch == par[node]) continue;
```

```
        par[ch] = node;
        depth[ch] = 1 + depth[node];
        dist[ch] = w + dist[node];
        if (depth[node] > 0 && jmpEdges[node] == jmpEdges[jmp[node]])
            jmp[ch] = jmp[jmp[node]], jmpEdges[ch] = 2 * jmpEdges[node] + 1;
        else
            jmp[ch] = node, jmpEdges[ch] = 1;
        dfs(ch, adj);
    }
}

//traverse up k edges in O(log(k)). So with k=1 this returns 'node's parent
int kthPar(int node, int k) const {
    k = min(k, depth[node]);
    while (k > 0) {
        if (jmpEdges[node] <= k) {
            k -= jmpEdges[node];
            node = jmp[node];
        } else {
            k--;
            node = par[node];
        }
    }
    return node;
}

int getLca(int x, int y) const {
    if (depth[x] < depth[y]) swap(x, y);
    x = kthPar(x, depth[x] - depth[y]);
    while (x != y) {
        if (jmp[x] != jmp[y])
            x = jmp[x], y = jmp[y];
        else
            x = par[x], y = par[y];
    }
    return x;
}

int distEdges(int x, int y) const {
    return depth[x] + depth[y] - 2 * depth[getLca(x, y)];
}

long long distWeight(int x, int y) const {
    return dist[x] + dist[y] - 2 * dist[getLca(x, y)];
}
};
```

Listing 11: SCC

```
//cat scc.h | ./hash.sh
//6dae87
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/scc

struct sccInfo {
    int numSCCs;
    //scc's are labeled 0,1,...,'numSCCs-1'
    //sccId[i] is the id of the scc containing node 'i'
    //for each edge i -> j: sccId[i] >= sccId[j]
    vector<int> sccId;
```

```
};

sccInfo getSCCs(const vector<vector<int>>& adj /*directed, unweighted graph*/) {
    int n = adj.size(), timer = 1, numSCCs = 0;
    vector<int> tin(n, 0), sccId(n, -1), nodeStack;
    auto dfs = [&](auto&& self, int v) -> int {
        int low = tin[v] = timer++;
        nodeStack.push_back(v);
        for (int to : adj[v]) {
            if (sccId[to] < 0)
                low = min(low, tin[to] ? tin[to] : self(self, to));
        }
        if (tin[v] == low) {
            while (true) {
                int node = nodeStack.back();
                nodeStack.pop_back();
                sccId[node] = numSCCs;
                if (node == v) break;
            }
            numSCCs++;
        }
        return low;
    };
    for (int i = 0; i < n; i++) {
        if (!tin[i])
            dfs(dfs, i);
    }
    return {numSCCs, sccId};
}
```

Listing 12: RANGE DATA STRUCTURES

Listing 13: Segment Tree

```
//cat segTree.h | ./hash.sh
//a9b5a9
#pragma once
//stress tests: tests/stress-tests/range_data_structures/segTree.cpp

const long long inf = 1e18;

struct segTree {
    struct Node {
        long long sum, mx, mn;
        long long lazy;
        int l, r;

        int len() const {
            return r - l + 1;
        }
        //returns 1 + (# of nodes in left child's subtree)
        //https://cp-algorithms.com/data_structures/segment_tree.html#memory-efficient-implementation
        int rCh() const {
            return ((r - 1) & ~1) + 2;
        }
    };
};

vector<Node> tree;
```

```
//There's no constructor 'segTree(int size)' because how to initialize l,r in nodes
↳ without calling build?
//the whole point of 'segTree(int size)' was to be simpler by not calling build
segTree(const vector<long long>& arr) : tree(2 * (int) arr.size() - 1) {
    build(arr, 0, 0, (int) arr.size() - 1);
}

void build(const vector<long long>& arr, int v, int tl, int tr) {
    if (tl == tr) {
        tree[v] = {
            arr[tl],
            arr[tl],
            arr[tl],
            0,
            tl,
            tr
        };
    } else {
        int tm = tl + (tr - tl) / 2;
        build(arr, v + 1, tl, tm);
        build(arr, v + 2 * (tm - tl + 1), tm + 1, tr);
        tree[v] = combine(tree[v + 1], tree[v + 2 * (tm - tl + 1)]);
    }
}

static Node combine(const Node& L, const Node& R) {
    return {
        L.sum + R.sum,
        max(L.mx, R.mx),
        min(L.mn, R.mn),
        0,
        L.l,
        R.r
    };
}

//what happens when 'add' is applied to every index in range [tree[v].l, tree[v].r]?
void apply(int v, long long add) {
    tree[v].sum += tree[v].len() * add;
    tree[v].mx += add;
    tree[v].mn += add;
    if (tree[v].len() > 1) {
        tree[v + 1].lazy += add;
        tree[v + tree[v].rCh()].lazy += add;
    }
}

void push(int v) {
    if (tree[v].lazy) {
        apply(v, tree[v].lazy);
        tree[v].lazy = 0;
    }
}

//update range [l,r] with 'add'
void update(int l, int r, long long add) {
    update(0, l, r, add);
}

void update(int v, int l, int r, long long add) {
    push(v);
    if (tree[v].r < l || r < tree[v].l)
```

```
        return;
    if (l <= tree[v].l && tree[v].r <= r)
        return apply(v, add);
    update(v + 1, l, r, add);
    update(v + tree[v].rCh(), l, r, add);
    tree[v] = combine(tree[v + 1], tree[v + tree[v].rCh()]);
}

//range [l,r]
Node query(int l, int r) {
    return query(0, l, r);
}

Node query(int v, int l, int r) {
    if (tree[v].r < l || r < tree[v].l)
        return {0, -inf, inf, 0, 0, 0};
    push(v);
    if (l <= tree[v].l && tree[v].r <= r)
        return tree[v];
    return combine(query(v + 1, l, r),
        query(v + tree[v].rCh(), l, r));
}

};
```

Listing 14: Fenwick Tree

```
//cat fenwickTree.h | ./hash.sh
//3d4557
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/point_add_range_sum,
    ↪ https://judge.yosupo.jp/problem/vertex_add_path_sum,
    ↪ https://judge.yosupo.jp/problem/vertex_add_subtree_sum,
    ↪ https://judge.yosupo.jp/problem/predecessor_problem

template<class T>
struct fenwickTree {
    vector<T> bit;
    fenwickTree(int n) : bit(n, 0) {}
    fenwickTree(const vector<T>& a) : bit(a.size()) {
        if (a.empty()) return;
        bit[0] = a[0];
        for (int i = 1; i < (int) a.size(); i++)
            bit[i] = bit[i - 1] + a[i];
        for (int i = (int) a.size() - 1; i > 0; i--) {
            int lower_i = (i & (i + 1)) - 1;
            if (lower_i >= 0)
                bit[i] -= bit[lower_i];
        }
    }
    void update(int idx, const T& d) {
        for (; idx < (int) bit.size(); idx = idx | (idx + 1))
            bit[idx] += d;
    }
    T sum(int r) const {
        T ret = 0;
        for (; r >= 0; r = (r & (r + 1)) - 1)
            ret += bit[r];
        return ret;
    }
    T sum(int l, int r) const {
        return sum(r) - sum(l - 1);
    }
};
```

```
    }
    //Returns min pos such that sum of [0, pos] >= sum
    //Returns bit.size() if no sum is >= sum, or -1 if empty sum is.
    //Doesn't work with negatives (since it's greedy), counterexample: array: {1, -1},
        ↪ sum: 1, this returns 2, but should return 0
    int lower_bound(T sum) const {
        if (sum <= 0) return -1;
        int pos = 0;
        for (int pw = 1 << (31 - __builtin_clz(bit.size() | 1)); pw; pw >>= 1) {
            if (pos + pw <= (int)bit.size() && bit[pos + pw - 1] < sum)
                pos += pw, sum -= bit[pos - 1];
        }
        return pos;
    }
};
```

Listing 15: Sparse Table

```
//cat sparseTable.h | ./hash.sh
//912bbe
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/staticrmq,
    ↪ https://judge.yosupo.jp/problem/zaigalgorithm,
    ↪ https://judge.yosupo.jp/problem/enumerate_palindromes

//usage:
// vector<long long> arr;
// ...
// sparseTable<long long> st(arr, [](auto x, auto y) { return min(x,y); });
//
//to also get index of min element, do:
// sparseTable<pair<long long,int>> st(arr, [](auto x, auto y) { return min(x,y); });
//and initialize '.second's to index. If there are multiple indexes of min element,
//it'll return the smallest (left-most) one
template <class T>
struct sparseTable {
    vector<vector<T>> dp;
    function<T(const T&, const T&)> func;
    sparseTable(const vector<T>& arr, const function<T(const T&, const T&)>& _func) :
        ↪ dp(1, arr), func(_func) {
        int n = arr.size();
        for (int pw = 1, k = 1; pw * 2 <= n; pw *= 2, k++) {
            dp.emplace_back(n - pw * 2 + 1);
            for (int j = 0; j < (int)dp[k].size(); j++)
                dp[k][j] = func(dp[k - 1][j], dp[k - 1][j + pw]);
        }
    }
    //inclusive range [l, r]
    T query(int l, int r) const {
        int lg = 31 - __builtin_clz(r - l + 1);
        return func(dp[lg][l], dp[lg][r - (1 << lg) + 1]);
    }
};
```

Listing 16: Implicit Lazy Segment Tree

```
//cat implicitSegTree.h | ./hash.sh
//c19ef1
#pragma once
```

```
//stress tests: tests/stress-tests/range_data_structures/implicitSegTree.cpp

//see TODO for lines of code which usually need to change (not a complete list)

const int N = 1.5e7; //TODO

struct Node {
    long long val; //could represent max, sum, etc
    long long lazy;
    int lCh, rCh; // children, indexes into 'tree', -1 for null
} tree[N];

struct implicitSegTree {

    int NEW_NODE, rootL, rootR; // [rootL, rootR] defines range of root node; handles
    ↪ negatives

    implicitSegTree(int l, int r) : NEW_NODE(0), rootL(l), rootR(r) {
        tree[NEW_NODE++] = {0, 0, -1, -1}; //TODO
    }

    static long long combine(long long val_l, long long val_r) {
        return val_l + val_r; //TODO
    }

    void apply(int v, int tl, int tr, long long add) {
        tree[v].val += (tr - tl + 1) * add; //TODO
        if (tl != tr) {
            tree[tree[v].lCh].lazy += add; //TODO
            tree[tree[v].rCh].lazy += add;
        }
    }

    void push(int v, int tl, int tr) {
        if (tl != tr && tree[v].lCh == -1) {
            assert(NEW_NODE + 1 < N);
            tree[v].lCh = NEW_NODE;
            tree[NEW_NODE++] = {0, 0, -1, -1}; //TODO
            tree[v].rCh = NEW_NODE;
            tree[NEW_NODE++] = {0, 0, -1, -1};
        }
        if (tree[v].lazy) {
            apply(v, tl, tr, tree[v].lazy);
            tree[v].lazy = 0;
        }
    }

    //update range [l,r] with 'add'
    void update(int l, int r, long long add) {
        update(0, rootL, rootR, l, r, add);
    }

    void update(int v, int tl, int tr, int l, int r, long long add) {
        push(v, tl, tr);
        if (tr < l || r < tl)
            return;
        if (l <= tl && tr <= r)
            return apply(v, tl, tr, add);
        int tm = tl + (tr - tl) / 2;
        update(tree[v].lCh, tl, tm, l, r, add);
        update(tree[v].rCh, tm + 1, tr, l, r, add);
        tree[v].val = combine(tree[tree[v].lCh].val, tree[tree[v].rCh].val);
    }
};
```

```
    }

    //query range [l,r]
    //for more complicated query which doesn't allocate new nodes, see:
    //https://github.com/lrviddeckis/Programming-Team-Code/blob/dc659297850440b65af2550a8342cc...
    long long query(int l, int r) {
        return query(0, rootL, rootR, l, r);
    }

    long long query(int v, int tl, int tr, int l, int r) {
        if (tr < l || r < tl)
            return 0; //TODO
        push(v, tl, tr);
        if (l <= tl && tr <= r)
            return tree[v].val;
        int tm = tl + (tr - tl) / 2;
        return combine(query(tree[v].lCh, tl, tm, l, r),
            query(tree[v].rCh, tm + 1, tr, l, r));
    }
};
```

Listing 17: Range Updates, Point Queries

```
//cat fenwickInv.h | ./hash.sh
//6009e6
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/vertex_add_subtree_sum,
    ↪ https://judge.yosupo.jp/problem/point_add_range_sum

#include "../fenwickTree.h"

template<class T>
struct fenwickInv {
    fenwickTree<T> ft;
    fenwickInv(int n) : ft(n) {}
    fenwickInv(const vector<T>& arr) : ft(init(arr)) {}
    fenwickTree<T> init(vector<T> arr /*intentional pass by value*/) const {
        for (int i = (int) arr.size() - 1; i >= 1; i--)
            arr[i] -= arr[i - 1];
        return fenwickTree<T>(arr);
    }
    //add 'add' to inclusive range [l, r]
    void update(int l, int r, const T& add) {
        ft.update(l, add);
        if (r + 1 < (int) ft.bit.size())
            ft.update(r + 1, -add);
    }
    //get value at index 'idx'
    T query(int idx) const {
        return ft.sum(idx);
    }
};
```

Listing 18: Kth Smallest

```
//cat kth_smallest.h | ./hash.sh
//7fa26d
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/range_kth_smallest
```


<pre>//modified from ↳ https://cp-algorithms.com/data_structures/segment_tree.html#preserving-the-history-of-history-versions-to-segment-tree struct kth_smallest { struct Node { int sum; int lCh, rCh; //children, indexes into 'tree' }; int mn, mx; vector<int> roots; deque<Node> tree; kth_smallest(const vector<int>& arr) : mn(INT_MAX), mx(INT_MIN), roots(arr.size() + ↳ 1, 0) { tree.push_back({0, 0, 0}); //acts as null for (int val : arr) mn = min(mn, val), mx = max(mx, val); for (int i = 0; i < (int)arr.size(); i++) roots[i + 1] = update(roots[i], mn, mx, arr[i]); } int update(int v, int tl, int tr, int idx) { if (tl == tr) { tree.push_back({tree[v].sum + 1, 0, 0}); return tree.size() - 1; } int tm = tl + (tr - tl) / 2; int lCh = tree[v].lCh; int rCh = tree[v].rCh; if (idx <= tm) lCh = update(lCh, tl, tm, idx); else rCh = update(rCh, tm + 1, tr, idx); tree.push_back({tree[lCh].sum + tree[rCh].sum, lCh, rCh}); return tree.size() - 1; } /* find (k+1)th smallest number among arr[l], arr[l+1], ..., arr[r] * k is 0-based, so query(l,r,0) returns the min */ int query(int l, int r, int k) const { assert(0 <= k && k < r - l + 1); //note this condition implies L <= R assert(0 <= l && r + 1 < (int)roots.size()); return query(roots[l], roots[r + 1], mn, mx, k); } int query(int vl, int vr, int tl, int tr, int k) const { if (tl == tr) return tl; int tm = tl + (tr - tl) / 2; int left_count = tree[tree[vr].lCh].sum - tree[tree[vl].lCh].sum; if (left_count > k) return query(tree[vl].lCh, tree[vr].lCh, tl, tm, k); return query(tree[vl].rCh, tree[vr].rCh, tm + 1, tr, k - left_count); } };</pre>	<pre>#pragma once //uses persistent segment tree ↳ https://cp-algorithms.com/data_structures/distinct_query.cpp //modified from ↳ https://cp-algorithms.com/data_structures/segment_tree.html#preserving-the-history-of-history-versions-to-segment-tree //works with negatives //O(n log n) time and space struct distinct_query { struct Node { int sum; int lCh, rCh; //children, indexes into 'tree' }; int sz; vector<int> roots; deque<Node> tree; distinct_query(const vector<int>& arr) : sz(arr.size() + 1), roots(sz, 0) { tree.push_back({0, 0, 0}); //acts as null map<int, int> lastIdx; for (int i = 0; i < (int)arr.size(); i++) { roots[i + 1] = update(roots[i], 0, sz - 1, lastIdx[arr[i]]); lastIdx[arr[i]] = i + 1; } } int update(int v, int tl, int tr, int idx) { if (tl == tr) { tree.push_back({tree[v].sum + 1, 0, 0}); return tree.size() - 1; } int tm = (tl + tr) / 2; int lCh = tree[v].lCh; int rCh = tree[v].rCh; if (idx <= tm) lCh = update(lCh, tl, tm, idx); else rCh = update(rCh, tm + 1, tr, idx); tree.push_back({tree[lCh].sum + tree[rCh].sum, lCh, rCh}); return tree.size() - 1; } //returns number of distinct elements in range [l,r] int query(int l, int r) const { return query(roots[l], roots[r + 1], 0, sz - 1, l + 1); } int query(int vl, int vr, int tl, int tr, int idx) const { if (tree[vr].sum == 0 idx <= tl) return 0; if (tr < idx) return tree[vr].sum - tree[vl].sum; int tm = (tl + tr) / 2; return query(tree[vl].lCh, tree[vr].lCh, tl, tm, idx) + query(tree[vl].rCh, tree[vr].rCh, tm + 1, tr, idx); } };</pre>
---	--

Listing 19: Number Distinct Elements

```
//cat distinct_query.h | ./hash.sh
//7160c1
```

Listing 20: Buckets

```
//cat buckets.h | ./hash.sh
//db53a3
#pragma once
//stress tests: tests/stress-tests/range_data_structures/buckets.cpp

//this code isn't the best. It's meant as a rough start for sqrt-decomposition, and to
↪ be modified
//doesn't handle overflow
struct buckets {
    const int BUCKET_SIZE = 50; //TODO: change - small value for testing

    struct bucket {
        int sumLazy = 0;
        int sumBucket = 0;
        int l, r; //inclusive range of bucket
        int len() const {
            return r - l + 1;
        }
    };

    vector<int> values;
    vector<bucket> _buckets;

    buckets(const vector<int>& initial) : values(initial) {
        int numBuckets = ((int) values.size() + BUCKET_SIZE - 1) / BUCKET_SIZE;
        _buckets.resize(numBuckets);
        for (int i = 0; i < numBuckets; i++) {
            _buckets[i].sumLazy = 0;
            _buckets[i].sumBucket = 0;
            _buckets[i].l = i * BUCKET_SIZE;
            _buckets[i].r = min((i + 1) * BUCKET_SIZE, (int) values.size()) - 1;
            for (int j = _buckets[i].l; j <= _buckets[i].r; j++)
                _buckets[i].sumBucket += values[j];
        }

        void pushLazy(int bIdx) {
            bucket& b = _buckets[bIdx];
            if (!b.sumLazy) return;
            for (int i = b.l; i <= b.r; i++)
                values[i] += b.sumLazy;
            b.sumLazy = 0;
        }

        //update range [L,R]
        void update(int L, int R, int diff) {
            int startBucket = L / BUCKET_SIZE;
            int endBucket = R / BUCKET_SIZE;
            if (startBucket == endBucket) { //range contained in same bucket case
                for (int i = L; i <= R; i++) {
                    values[i] += diff;
                    _buckets[startBucket].sumBucket += diff;
                }
                return;
            }
            for (int bIdx : {
                startBucket, endBucket
            }) { //handle "endpoint" buckets
                bucket& b = _buckets[bIdx];
                for (int i = max(b.l, L); i <= min(b.r, R); i++) {
                    values[i] += diff;
                    b.sumBucket += diff;
                }
            }
        }
    };
};
```

```
        for (int i = max(b.l, L); i <= min(b.r, R); i++) {
            values[i] += diff;
            b.sumBucket += diff;
        }
    }
    for (int i = startBucket + 1; i < endBucket; i++) { //handle all n/B buckets
        ↪ in middle
        bucket& b = _buckets[i];
        b.sumLazy += diff;
        b.sumBucket += b.len() * diff;
    }
}

//sum of range [L,R]
int query(int L, int R) {
    int startBucket = L / BUCKET_SIZE;
    int endBucket = R / BUCKET_SIZE;
    if (startBucket == endBucket) { //range contained in same bucket case
        pushLazy(startBucket);
        int sum = 0;
        for (int i = L; i <= R; i++)
            sum += values[i];
        return sum;
    }
    int sum = 0;
    for (int bIdx : {
        startBucket, endBucket
    }) { //handle "endpoint" buckets
        bucket& b = _buckets[bIdx];
        pushLazy(bIdx);
        for (int i = max(b.l, L); i <= min(b.r, R); i++)
            sum += values[i];
    }
    for (int i = startBucket + 1; i < endBucket; i++) //handle all n/B buckets in
        ↪ middle
        sum += _buckets[i].sumBucket;
    return sum;
}
};
```

Listing 21: Persistent Lazy Segment Tree

```
//cat persistentLazySegTree.h | ./hash.sh
//87eace
#pragma once
//status: not tested

struct persistentLazySegTree {

    struct Node {
        int lCh, rCh; //children, indexes into 'tree'
        int sum;
        bool lazyTog;
    };

    int sz;
    deque<Node> tree;
    vector<int> roots;

    //implicit
```

```
persistentLazySegTree(int _sz) : sz(_sz) {
    tree.push_back({0, 0, 0, 0}); //acts as null
    roots.push_back(0);
}

void push(int v, int tl, int tr) {
    if (tl != tr) {
        tree.push_back(tree[tree[v].lCh]);
        tree[v].lCh = tree.size() - 1;
        tree.push_back(tree[tree[v].rCh]);
        tree[v].rCh = tree.size() - 1;
    }
    if (tree[v].lazyTog) {
        tree[v].sum = (tr - tl + 1) - tree[v].sum;
        tree[v].lazyTog = false;
        if (tl != tr) {
            tree[tree[v].lCh].lazyTog ^= 1;
            tree[tree[v].rCh].lazyTog ^= 1;
        }
    }
}

void set(int idx, int new_val) {
    tree.push_back(tree[roots.back()]); //allocate top down
    roots.push_back(tree.size() - 1);
    set(roots.back(), 0, sz - 1, idx, new_val);
}

void set(int v, int tl, int tr, int idx, int new_val) {
    push(v, tl, tr);
    if (tr < idx || idx < tl)
        return;
    if (idx <= tl && tr <= idx) {
        tree[v].sum = new_val;
        return;
    }
    int tm = (tl + tr) / 2;
    int lCh = tree[v].lCh;
    int rCh = tree[v].rCh;
    set(lCh, tl, tm, idx, new_val);
    set(rCh, tm + 1, tr, idx, new_val);
    tree[v].sum = tree[lCh].sum + tree[rCh].sum;
}

void toggleRange(int l, int r) {
    tree.push_back(tree[roots.back()]); //allocate top down
    roots.push_back(tree.size() - 1);
    toggleRange(roots.back(), 0, sz - 1, l, r);
}

void toggleRange(int v, int tl, int tr, int l, int r) {
    push(v, tl, tr);
    if (tr < l || r < tl)
        return;
    int lCh = tree[v].lCh;
    int rCh = tree[v].rCh;
    if (l <= tl && tr <= r) {
        tree[v].sum = (tr - tl + 1) - tree[v].sum;
        if (tl != tr) {
            tree[lCh].lazyTog ^= 1;
            tree[rCh].lazyTog ^= 1;
        }
    }
    return;
}
```

```

    }
    int tm = (tl + tr) / 2;
    toggleRange(lCh, tl, tm, l, r);
    toggleRange(rCh, tm + 1, tr, l, r);
    tree[v].sum = tree[lCh].sum + tree[rCh].sum;
}

//let's use implementation trick described here
//↪ https://codeforces.com/blog/entry/72626
//so that we don't have to propagate lazy vals and thus we don't have to allocate
//↪ new nodes
int query(int l, int r) const {
    int version = roots.size() - 1;
    int root = roots[version];
    return query(root, 0, sz - 1, l, r, tree[root].lazyTog);
}

int query(int v, int tl, int tr, int l, int r, bool tog) const {
    if (v == 0 || tr < l || r < tl)
        return 0;
    if (l <= tl && tr <= r) {
        int sum = tree[v].sum;
        if (tree[v].lazyTog) sum = (tr - tl + 1) - sum;
        return sum;
    }
    int tm = (tl + tr) / 2;
    tog ^= tree[v].lazyTog;
    return query(tree[v].lCh, tl, tm, l, r, tog) +
           query(tree[v].rCh, tm + 1, tr, l, r, tog);
}
};
```

Listing 22: Mos Algorithm

```
//cat MosAlg.h | ./hash.sh
//122ecb
//status: not tested

#include <bits/stdc++.h>
using namespace std;

const int Max = 1e6 + 2;
int block, answer[Max], answerToQuery;

struct query {
    int l, r, index;
};

bool cmp(query x, query y) {
    if (x.l / block == y.l / block) return x.r < y.r;
    return x.l < y.l;
}

void add(int pos) {
}

void remove(int pos) {
}

int main() {
    int q;
    cin >> q;
    vector<query> queries(q);
```

```
for (int i = 0; i < q; i++) {
    cin >> queries[i].l >> queries[i].r;
    queries[i].index = i;
    answer[i] = 0;
}
sort(queries.begin(), queries.end(), cmp);
int left = 0, right = 0; //store inclusive ranges, start at [0,0]
add(0);
answerToQuery = 0;
for (auto& q : queries) {
    while (left > q.l) {
        left--;
        add(left);
    }
    while (right < q.r) {
        right++;
        add(right);
    }
    while (left < q.l) {
        remove(left);
        left++;
    }
    while (right > q.r) {
        remove(right);
        right--;
    }
    answer[q.index] = answerToQuery;
}
for (int i = 0; i < q; i++) cout << answer[i] << '\n';
return 0;
}
```

Listing 23: Merge Sort Tree

```
//cat mergeSortTree.h | ./hash.sh
//62339b
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/static_range_frequency,
    ↪ https://judge.yosupo.jp/problem/range_kth_smallest

//For point updates: either switch to policy based BST, or use sqrt decomposition

struct MergeSortTree {
    struct Node {
        vector<int> vals;

        int l, r;

        //returns 1 + (# of nodes in left child's subtree)
        //https://cp-algorithms.com/data_structures/segment_tree.html#memory-efficient-implementation
        int rCh() const {
            return ((r - 1) & ~1) + 2;
        }
    };

    vector<Node> tree;

    //RTE's when 'arr' is empty
    MergeSortTree(const vector<int>& arr) : tree(2 * (int) arr.size() - 1) {
        build(arr, 0, 0, (int) arr.size() - 1);
    }
};
```

```

    }
    void build(const vector<int>& arr, int v, int tl, int tr) {
        if (tl == tr) {
            tree[v] = {
                {arr[tl]},
                tl,
                tr
            };
        } else {
            int tm = tl + (tr - tl) / 2;
            build(arr, v + 1, tl, tm);
            build(arr, v + 2 * (tm - tl + 1), tm + 1, tr);
            tree[v] = combine(tree[v + 1], tree[v + 2 * (tm - tl + 1)]);
        }
    }

    Node combine(const Node& L, const Node& R) {
        vector<int> par(L.vals.size() + R.vals.size());
        merge(L.vals.begin(), L.vals.end(), R.vals.begin(), R.vals.end(), par.begin());
        return {par, L.l, R.r};
    }

    //How many of arr[l], arr[l+1], ..., arr[r] are < x?
    //O(log^2(n))
    int query(int l, int r, int x) const {
        return query(0, l, r, x);
    }

    int query(int v, int l, int r, int x) const {
        if (tree[v].r < l || r < tree[v].l)
            return 0;

        if (l <= tree[v].l && tree[v].r <= r) {
            const vector<int>& vals = tree[v].vals;
            return lower_bound(vals.begin(), vals.end(), x) - vals.begin();
        }

        return query(v + 1, l, r, x) +
            query(v + tree[v].rCh(), l, r, x);
    }
};
```

Listing 24: STRINGS

Listing 25: Suffix Array

```
//cat suffix_array.h | ./hash.sh
//46840a
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/suffixarray,
    ↪ https://judge.yosupo.jp/problem/zalgorithm,
    ↪ https://judge.yosupo.jp/problem/number_of_substrings,
    ↪ https://judge.yosupo.jp/problem/enumerate_palindromes

//modified from here: https://judge.yosupo.jp/submission/37410
//
// SA-IS, linear-time suffix array construction
// Reference:
// G. Nong, S. Zhang, and W. H. Chan,
// Two Efficient Algorithms for Linear Time Suffix Array Construction
template<class T>
```

```
vector<int> sa_is(const T& s, int upper/*max element of 's'; for std::string, pass in
↳ 255*/) {
    int n = (int) s.size();
    if (n == 0) return {};
    if (n == 1) return {0};
    if (n == 2) {
        if (s[0] < s[1]) {
            return {0, 1};
        } else {
            return {1, 0};
        }
    }
    vector<int> sa(n);
    vector<bool> ls(n);
    for (int i = n - 2; i >= 0; i--)
        ls[i] = (s[i] == s[i + 1]) ? ls[i + 1] : (s[i] < s[i + 1]);
    vector<int> sum_l(upper + 1), sum_s(upper + 1);
    for (int i = 0; i < n; i++) {
        if (!ls[i])
            sum_s[s[i]]++;
        else
            sum_l[s[i] + 1]++;
    }
    for (int i = 0; i <= upper; i++) {
        sum_s[i] += sum_l[i];
        if (i < upper) sum_l[i + 1] += sum_s[i];
    }
    vector<int> buf(upper + 1);
    auto induce = [&](const vector<int>& lms) {
        fill(sa.begin(), sa.end(), -1);
        fill(buf.begin(), buf.end(), 0);
        copy(sum_s.begin(), sum_s.end(), buf.begin());
        for (auto d : lms) {
            if (d == n) continue;
            sa[buf[s[d]]++] = d;
        }
        copy(sum_l.begin(), sum_l.end(), buf.begin());
        sa[buf[s[n - 1]]++] = n - 1;
        for (int i = 0; i < n; i++) {
            int v = sa[i];
            if (v >= 1 && !ls[v - 1])
                sa[buf[s[v - 1]]++] = v - 1;
        }
        copy(sum_l.begin(), sum_l.end(), buf.begin());
        for (int i = n - 1; i >= 0; i--) {
            int v = sa[i];
            if (v >= 1 && ls[v - 1])
                sa[--buf[s[v - 1] + 1]] = v - 1;
        }
    };
    vector<int> lms_map(n + 1, -1);
    int m = 0;
    for (int i = 1; i < n; i++) {
        if (!ls[i - 1] && ls[i])
            lms_map[i] = m++;
    }
    vector<int> lms;
    lms.reserve(m);
    for (int i = 1; i < n; i++) {
        if (!ls[i - 1] && ls[i])
            lms.push_back(i);
    }
}
```

```
    }
    induce(lms);
    if (m) {
        vector<int> sorted_lms;
        sorted_lms.reserve(m);
        for (int v : sa) {
            if (lms_map[v] != -1) sorted_lms.push_back(v);
        }
        vector<int> rec_s(m);
        int rec_upper = 0;
        rec_s[lms_map[sorted_lms[0]]] = 0;
        for (int i = 1; i < m; i++) {
            int l = sorted_lms[i - 1], r = sorted_lms[i];
            int end_l = (lms_map[l] + 1 < m) ? lms[lms_map[l] + 1] : n;
            int end_r = (lms_map[r] + 1 < m) ? lms[lms_map[r] + 1] : n;
            bool same = true;
            if (end_l - l != end_r - r)
                same = false;
            else {
                while (l < end_l) {
                    if (s[l] != s[r])
                        break;
                    l++;
                    r++;
                }
                if (l == n || s[l] != s[r]) same = false;
            }
            if (!same) rec_upper++;
            rec_s[lms_map[sorted_lms[i]]] = rec_upper;
        }
        auto rec_sa =
            sa_is(rec_s, rec_upper);
        for (int i = 0; i < m; i++)
            sorted_lms[i] = lms[rec_sa[i]];
        induce(sorted_lms);
    }
    return sa;
}
```

Listing 26: Longest Common Prefix Array

```
//cat longest_common_prefix.h | ./hash.sh
//396173
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/zalgorithm,
↳ https://judge.yosupo.jp/problem/number_of_substrings,
↳ https://judge.yosupo.jp/problem/enumerate_palindromes

//modified from here: https://judge.yosupo.jp/submission/37410
//
// Reference:
// T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park,
// Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its
// Applications
template<class T>
vector<int> lcp_array(const T& s, const vector<int>& sa) {
    int n = s.size(), k = 0;
    vector<int> lcp(n, 0);
    vector<int> rank(n, 0);
    for (int i = 0; i < n; i++) rank[sa[i]] = i;
```

```
for (int i = 0; i < n; i++, k ? k-- : 0) {
    if (rank[i] == n - 1) {
        k = 0;
        continue;
    }
    int j = sa[rank[i] + 1];
    while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
    lcp[rank[i]] = k;
}
return lcp;
}
```

Listing 27: Prefix Function

```
//cat prefix_function.h | ./hash.sh
//aa0518
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/zalgorithm
//stress tests: tests/stress-tests/strings/kmp.cpp

//source: https://cp-algorithms.com/string/prefix-function.html#implementation
template <class T>
vector<int> prefix_function(const T& s) {
    int n = s.size();
    vector<int> pi(n, 0);
    for (int i = 1; i < n; i++) {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j]) j = pi[j - 1];
        pi[i] = j + (s[i] == s[j]);
    }
    return pi;
}
```

Listing 28: KMP

```
//cat kmp.h | ./hash.sh
//dd5c08
#pragma once
//stress tests: tests/stress-tests/strings/kmp.cpp

#include "prefix_function.h"

//usage:
// string needle;
// ...
// KMP_Match kmp(needle);
//or
// vector<int> needle;
// ...
// KMP_Match kmp(needle);

//kmp-doubling-trick: to check if 2 arrays are rotationally equivalent: run kmp
//with one array as the needle and the other array doubled as the haystack
//or just use kactl's min rotation code
template <class T>
struct KMP_Match {
    KMP_Match(const T& needle_) : pi(prefix_function(needle_)), needle(needle_) {}
```

```
// if haystack = "bananas"
// needle = "ana"
//
// then we find 2 matches:
// bananas
// _ana_
// __ana_
// 0123456 (indexes)
// and KMP_Match::find returns {1,3} - the indexes in haystack where
// each match starts.
//
// You can also pass in false for "all" and KMP_Match::find will only
// return the first match: {1}. Useful for checking if there exists
// some match:
//
// KMP_Match::find(<haystack>,false).size() > 0
vector<int> find(const T& haystack, bool all = true) const {
    vector<int> matches;
    for (int i = 0, j = 0; i < (int)haystack.size(); i++) {
        while (j > 0 && needle[j] != haystack[i]) j = pi[j - 1];
        if (needle[j] == haystack[i]) j++;
        if (j == (int)needle.size()) {
            matches.push_back(i - (int)needle.size() + 1);
            if (!all) return matches;
            j = pi[j - 1];
        }
    }
    return matches;
}
vector<int> pi; //prefix function
T needle;
```

Listing 29: Trie

```
//cat trie.h | ./hash.sh
//928e34
#pragma once

//status: not tested
//intended to be a base template and to be modified

const int K = 26; //character size

struct trie {

    const char minCh = 'a'; // 'A' for uppercase, '0' for digits

    struct node {
        bool leaf = 0;
        int next[K], id, p = -1;
        char pch;
        node(int _p = -1, char ch = '#') : p(_p), pch(ch) {
            fill(next, next + K, -1);
        }
    };

    vector<node> t;
```

```
trie() : t(1) {}

void add_string(const string& s, int id) {
    int c = 0;
    for (char ch : s) {
        int v = ch - minCh;
        if (t[c].next[v] == -1) {
            t[c].next[v] = t.size();
            t.emplace_back(c, ch);
        }
        c = t[c].next[v];
    }
    t[c].leaf = 1;
    t[c].id = id;
}

void remove_string(const string& s) {
    int c = 0;
    for (char ch : s) {
        int v = ch - minCh;
        if (t[c].next[v] == -1)
            return;
        c = t[c].next[v];
    }
    t[c].leaf = 0;
}

int find_string(const string& s) {
    int c = 0;
    for (char ch : s) {
        int v = ch - minCh;
        if (t[c].next[v] == -1)
            return -1;
        c = t[c].next[v];
    }
    if (!t[c].leaf) return -1;
    return t[c].id;
}
};
```

Listing 30: Longest Common Prefix Query

```
//cat lcp_queries.h | ./hash.sh
//839d2c
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/zalgorithm,
//↳ https://judge.yosupo.jp/problem/enumerate_palindromes

#include "suffix_array.h"
#include "longest_common_prefix.h"
#include "../range_data_structures/sparseTable.h"

//computes suffix array, lcp array, and then sparse table over lcp array
//O(n log n)

struct lcp_queries {
    lcp_queries(const string& s) : sa(sa_is(s, 255)), inv_sa(s.size()), lcp(lcp_array(s,
        ↳ sa)), st(lcp, [](int x, int y) {
        return min(x, y);
    }) {
```

```
        for (int i = 0; i < (int)s.size(); i++)
            inv_sa[sa[i]] = i;
    }

    //length of longest common prefix of suffixes s[idx1 ... n-1], s[idx2 ... n-1],
    //↳ 0-based indexing
    //You can check if two substrings s[L1..R1], s[L2..R2] are equal in O(1) by:
    //
    //R2-L2 == R1-L1 ⇔ longest_common_prefix(L1, L2) >= R2-L2+1
    int longest_common_prefix(int idx1, int idx2) const {
        if (idx1 == idx2) return (int) sa.size() - idx1;
        idx1 = inv_sa[idx1];
        idx2 = inv_sa[idx2];
        if (idx1 > idx2) swap(idx1, idx2);
        return st.query(idx1, idx2 - 1);
    }

    //returns true if suffix s[idx1 ... n-1] < s[idx2 ... n-1]
    //(so false if idx1 == idx2)
    bool less(int idx1, int idx2) const {
        return inv_sa[idx1] < inv_sa[idx2];
    }

    vector<int> sa, inv_sa, lcp;
    sparseTable<int> st;
};
```

Listing 31: MATH

Listing 32: BIN EXP MOD

```
//cat exp_mod.h | ./hash.sh
//deca76
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/system_of_linear_equations,
//↳ https://judge.yosupo.jp/problem/binomial_coefficient,
//↳ https://judge.yosupo.jp/problem/matrix_det,
//↳ https://judge.yosupo.jp/problem/inverse_matrix
//stress tests: tests/stress-tests/math/exp_mod.cpp

//returns (base^pw)%mod in O(log(pw)), but returns 1 for 0^0

//What if base doesn't fit in long long?
//Since (base^pw)%mod == ((base%mod)%pw)%mod we can calculate base under mod of 'mod'

//What if pw doesn't fit in long long?

//case 1: mod is prime
//(base^pw)%mod == (base^(pw%(mod-1)))%mod (from Fermat's little theorem)
//so calculate pw under mod of 'mod-1'

//case 2: non-prime mod
//let t = totient(mod)
//if pw >= log2(mod) then (base^pw)%mod == (base^(t+(pw%t)))%mod (proof
//↳ https://cp-algorithms.com/algebra/phi-function.html#generalization)
```



```
//so calculate pw under mod of 't'
//incidentally, totient(p) = p - 1 for every prime p, making this a more generalized
    ↪ version of case 1

int fastPow(long long base, long long pw, int mod) {
    assert(0 <= pw && 0 <= base && 1 <= mod);
    int res = 1;
    base %= mod;
    while (pw > 0) {
        if (pw & 1) res = res * base % mod;
        base = base * base % mod;
        pw >>= 1;
    }
    return res;
}
```

Listing 33: Fibonacci

```
//cat fib.h | ./hash.sh
//9ac293
#pragma once
//stress tests: tests/stress-tests/math/fib_matrix_exp.c

//https://codeforces.com/blog/entry/14516
unordered_map<long long, int> table;
int fib(long long n, int mod) {    /**O(log(n))**
    if (n < 2) return 1;
    if (table.find(n) != table.end()) return table[n];
    table[n] = (1LL * fib((n + 1) / 2, mod) * fib(n / 2, mod) + 1LL * fib((n - 1) / 2,
        ↪ mod) * fib((n - 2) / 2, mod)) % mod;
    return table[n];
}
```

Listing 34: Matrix Mult and Pow

```
//cat matrixMultPow.h | ./hash.sh
//e2b9c4
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/matrix_product
//stress tests: tests/stress-tests/math/fib_matrix_exp.c

//empty matrix -> RTE

vector<vector<int>> mult(const vector<vector<int>>& a, const vector<vector<int>>& b, int
    ↪ mod) {
    assert(a[0].size() == b.size());
    int n = a.size(), m = b[0].size(), inner = b.size();
    vector<vector<int>> prod(n, vector<int>(m, 0));
    for (int i = 0; i < n; i++) {
        for (int k = 0; k < inner; k++) {
            for (int j = 0; j < m; j++)
                prod[i][j] = (prod[i][j] + 1LL * a[i][k] * b[k][j]) % mod;
        }
    }
    return prod;
}

vector<vector<int>> power(vector<vector<int>> matrix/*intentional pass by value*/, long
    ↪ long pw, int mod) {
```

```
int n = matrix.size();
vector<vector<int>> prod(n, vector<int>(n, 0));
for (int i = 0; i < n; i++)
    prod[i][i] = 1;
while (pw > 0) {
    if (pw % 2 == 1) prod = mult(prod, matrix, mod);
    matrix = mult(matrix, matrix, mod);
    pw /= 2;
}
return prod;
}
```

Listing 35: N Choose K MOD

```
//cat n_choose_k_mod.h | ./hash.sh
//db2a09
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/binomial_coefficient
//only the tests with prime mod

//for mod inverse
#include "exp_mod.h"

// usage:
// NchooseK nk(n, 1e9+7) to use 'choose', 'inv' with inputs < n
// or:
// NchooseK nk(mod, mod) to use 'chooseWithLucasTheorem'
struct NchooseK {
    // 'factSz' is the size of the factorial array, so only call 'choose', 'inv' with n
    ↪ < factSz
    NchooseK(int factSz, int currMod) : mod(currMod), fact(factSz, 1), invFact(factSz,
        ↪ 1) {
        //this implementation doesn't work if factSz > mod because n! % mod = 0 when n
        ↪ >= mod. So 'invFact' array will be all 0's
        assert(max(factSz, 2) <= mod);
        //assert mod is prime. mod is intended to fit inside an int so that
        //multiplications fit in a longlong before being modded down. So this
        //will take sqrt(2^31) time
        for (int i = 2; i * i <= mod; i++) assert(mod % i);
        for (int i = 2; i < factSz; i++)
            fact[i] = 1LL * fact[i - 1] * i % mod;
        invFact.back() = fastPow(fact.back(), mod - 2, mod);
        for (int i = factSz - 2; i >= 2; i--)
            invFact[i] = 1LL * invFact[i + 1] * (i + 1) % mod;
    }

    //classic n choose k
    //fails when n >= mod
    int choose(int n, int k) const {
        if (k < 0 || k > n) return 0;
        //now we know 0 <= k <= n so 0 <= n
        return 1LL * fact[n] * invFact[k] % mod * invFact[n - k] % mod;
    }

    //lucas theorem to calculate n choose k in O(log(k))
    //need to calculate all factorials in range [0,mod), so O(mod) time&space, so need
    ↪ smallish prime mod (< 1e6 maybe)
    //handles n >= mod correctly
    int chooseWithLucasTheorem(long long n, long long k) const {
        if (k < 0 || k > n) return 0;
```



```
        if (k == 0 || k == n) return 1;
        return 1LL * chooseWithLucasTheorem(n / mod, k / mod) * choose(n % mod, k % mod)
            ↪ % mod;
    }

    //returns inverse of n in O(1)
    int inv(int n) const {
        assert(1 <= n);    //don't divide by 0 :)
        return 1LL * fact[n - 1] * invFact[n] % mod;
    }

    int mod;
    vector<int> fact, invFact;
};
```

Listing 36: Partitions

```
//cat partitions.h | ./hash.sh
//3356f6
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/partition_function

//https://oeis.org/A000041
//O(n sqrt n) time, but small-ish constant factor (there does exist a O(n log n)
    ↪ solution too)
vector<int> partitions(int n/*size of dp array*/, int mod) {
    vector<int> dp(n, 1);
    for (int i = 1; i < n; i++) {
        long long sum = 0;
        for (int j = 1, pent = 1, sign = 1; pent <= i; j++, pent += 3 * j - 2, sign =
            ↪ -sign) {
            if (pent + j <= i) sum += dp[i - pent - j] * sign + mod;
            sum += dp[i - pent] * sign + mod;
        }
        dp[i] = sum % mod;
    }
    return dp;
}
```

Listing 37: Derangements

```
//cat derangements.h | ./hash.sh
//c221bb
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/montmort_number_mod

//https://oeis.org/A000166
//
//for a permutation of size i:
//there are (i-1) places to move 0 to not be at index 0. Let's say we moved 0 to index j
    ↪ (j>0).
//If we move value j to index 0 (forming a cycle of length 2), then there are dp[i-2]
    ↪ derangements of the remaining i-2 elements
//else there are dp[i-1] derangements of the remaining i-1 elements (including j)
vector<int> derangements(int n/*size of dp array*/, int mod) {
    vector<int> dp(n, 0);
    dp[0] = 1;
    for (int i = 2; i < n; i++)
        dp[i] = 1LL * (i - 1) * (dp[i - 1] + dp[i - 2]) % mod;
```

```
        return dp;
    }
```

Listing 38: Prime Sieve Mobius

```
//cat primeSieveMobius.h | ./hash.sh
//1f8513
#pragma once

//status: not tested

//mobius[i] = 0 iff there exists a prime p s.t. i%(p^2)=0
//mobius[i] = -1 iff i has an odd number of distinct prime factors
//mobius[i] = 1 iff i has an even number of distinct prime factors
const int N = 2e6 + 10;
int mobius[N];
void calcMobius() {
    mobius[1] = 1;
    for (int i = 1; i < N; i++) {
        for (int j = i + i; j < N; j += i)
            mobius[j] -= mobius[i];
    }
}

int minPrime[N];
void calcSeive() {
    fill(minPrime, minPrime + N, N);
    for (int i = N - 1; i >= 2; i--) {
        for (int j = i; j < N; j += i)
            minPrime[j] = i;
    }
}
```

Listing 39: Row Reduce

```
//cat row_reduce.h | ./hash.sh
//ad11ab
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/system_of_linear_equations,
    ↪ https://judge.yosupo.jp/problem/matrix_det,
    ↪ https://judge.yosupo.jp/problem/inverse_matrix

//for mod inverse
#include "exp_mod.h"

//First 'cols' columns of A represents a matrix to be left in reduced row echelon form
//Row operations will be performed to all later columns
//
//example usage:
// row_reduce(A, A[0].size(), mod) //row reduce matrix with no extra columns
pair<int/*rank*/, int/*determinant*/> row_reduce(vector<vector<int>>& A, const int cols,
    ↪ const int mod) {
    int n = A.size(), m = A[0].size(), rank = 0, det = 1;
    assert(cols <= m);
    for (int col = 0; col < cols && rank < n; col++) {
        //find arbitrary pivot and swap pivot to current row
        for (int i = rank; i < n; i++)
            if (A[i][col] != 0) {
                if (rank != i) det = det == 0 ? 0 : mod - det;
```

```
        swap(A[i], A[rank]);
        break;
    }
    if (A[rank][col] == 0) {
        det = 0;
        continue;
    }
    det = (1LL * det * A[rank][col]) % mod;
    //make pivot 1 by dividing row by inverse of pivot
    const int aInv = fastPow(A[rank][col], mod - 2, mod);
    for (int j = 0; j < m; j++)
        A[rank][j] = (1LL * A[rank][j] * aInv) % mod;
    //zero-out all numbers above & below pivot
    for (int i = 0; i < n; i++)
        if (i != rank && A[i][col] != 0) {
            const int val = A[i][col];
            for (int j = 0; j < m; j++) {
                A[i][j] -= 1LL * A[rank][j] * val % mod;
                if (A[i][j] < 0) A[i][j] += mod;
            }
        }
    rank++;
}
assert(rank <= min(n, cols));
return {rank, det};
}
```

Listing 40: Solve Linear Equations MOD

```
//cat solve_linear_mod.h | ./hash.sh
//e458de
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/system_of_linear_equations

#include "row_reduce.h"

struct matrixInfo {
    int rank, det;
    vector<int> x;
};

//Solves A * x = b under prime mod.
//A is a n (rows) by m (cols) matrix, b is a length n column vector, x is a length m
    ↳ column vector.
//assumes n,m >= 1, else RTE
//Returns rank of A, determinant of A, and x (solution vector to A * x = b). x is empty
    ↳ if no solution. If multiple solutions, an arbitrary one is returned.
//Leaves A in reduced row echelon form (unlike kactl) with b appended.
//O(n * m * min(n,m))
matrixInfo solve_linear_mod(vector<vector<int>>& A, const vector<int>& b, const int mod)
    ↳ {
    assert(A.size() == b.size());
    int n = A.size(), m = A[0].size();
    for (int i = 0; i < n; i++)
        A[i].push_back(b[i]);
    auto [rank, det] = row_reduce(A, m, mod); //row reduce not including the last column
    //check if solution exists
    for (int i = rank; i < n; i++) {
        if (A[i].back() != 0) return {rank, det, {} }; //no solution exists
    }
}
```

```
//initialize solution vector ('x') from row-reduced matrix
vector<int> x(m, 0);
for (int i = 0, j = 0; i < rank; i++) {
    while (A[i][j] == 0) j++; //find pivot column
    x[j] = A[i].back();
}
return {rank, det, x};
}
```

Listing 41: Matrix Inverse

```
//cat matrix_inverse.h | ./hash.sh
//7f9c8c
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/inverse_matrix

#include "row_reduce.h"

//returns inverse of square matrix A, empty if no inverse
vector<vector<int>> matrix_inverse(vector<vector<int>> A/*intentional pass by value*/,
    ↳ const int mod) {
    int n = A.size();
    assert(n == (int)A[0].size());
    //append identity matrix
    for (int i = 0; i < n; i++) {
        A[i].resize(2 * n, 0);
        A[i][i + n] = 1;
    }
    auto [rank, det] = row_reduce(A, n, mod); //row reduce first n columns, leaving
    ↳ inverse in last n columns
    if (rank < n) return {}; //no inverse
    for (int i = 0; i < n; i++)
        A[i].erase(A[i].begin(), A[i].begin() + n);
    return A;
}
```

Listing 42: Euler’s Totient Phi Function

```
//cat totient.h | ./hash.sh
//36bd41
#pragma once
//stress tests: tests/stress-tests/math/totient.cpp

// Euler’s totient function counts the positive integers
// up to a given integer n that are relatively prime to n.

//To improve, use Pollard-rho to find prime factors

int totient(int n) {
    int res = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0) n /= i;
            res -= res / i;
        }
    }
    if (n > 1) res -= res / n;
    return res;
}
```

Listing 43: MAX FLOW

Listing 44: Dinic

```
//cat dinic.h | ./hash.sh
//44e407
#pragma once

//status: not tested

struct maxflow {
    typedef long long ll;
    ll n, s, t;
    maxflow(int _n, int _s, int _t) : n(_n), s(_s), t(_t), d(n), ptr(n), q(n), g(n) {}
    void addedge(ll a, ll b, ll cap) {
        edgeMap[a * n + b] = e.size();
        edge e1 = { a, b, cap, 0 };
        edge e2 = { b, a, 0, 0 };
        g[a].push_back((ll) e.size());
        e.push_back(e1);
        g[b].push_back((ll) e.size());
        e.push_back(e2);
    }
    ll getflow() {
        ll flow = 0;
        for (;;) {
            if (!bfs()) break;
            ptr.assign(ptr.size(), 0);
            while (ll pushed = dfs(s, inf))
                flow += pushed;
        }
        return flow;
    }
    ll getFlowForEdge(ll a, ll b) {
        return e[edgeMap[a * n + b]].flow;
    }

    const ll inf = 1e18;
    struct edge {
        ll a, b, cap, flow;
    };
    unordered_map<int, ll> edgeMap;
    vector<ll> d, ptr, q;
    vector<edge> e;
    vector<vector<ll>> g;
    bool bfs() {
        ll qh = 0, qt = 0;
        q[qt++] = s;
        d.assign(d.size(), -1);
        d[s] = 0;
        while (qh < qt && d[t] == -1) {
            ll v = q[qh++];
            for (size_t i = 0; i < g[v].size(); i++) {
                ll id = g[v][i],
                    to = e[id].b;
                if (d[to] == -1 && e[id].flow < e[id].cap) {
                    q[qt++] = to;
                    d[to] = d[v] + 1;
                }
            }
        }
    }
};
```

```
    }
    return d[t] != -1;
}
ll dfs(ll v, ll flow) {
    if (!flow) return 0;
    if (v == t) return flow;
    for (; ptr[v] < (ll) g[v].size(); ptr[v]++) {
        ll id = g[v][ptr[v]];
        ll to = e[id].b;
        if (d[to] != d[v] + 1) continue;
        ll pushed = dfs(to, min(flow, e[id].cap - e[id].flow));
        if (pushed) {
            e[id].flow -= pushed;
            e[id ^ 1].flow += pushed;
            return pushed;
        }
    }
    return 0;
}
};
```

Listing 45: Hungarian

```
//cat hungarian.h | ./hash.sh
//979cb1
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/assignment

const long long inf = 1e18;

//source: https://e-maxx.ru/algo/assignment_hungary

// this is one-indexed
// jobs X workers cost matrix
// cost[i][j] is cost of job i done by worker j
// #jobs must be <= #workers
// Default finds min cost; to find max cost set all costs[i][j] to -costs[i][j], set all
// ↪ unused to positive inf

struct match {
    long long cost;
    vector<int> matching;
};

match HungarianMatch(const vector<vector<long long>>& cost) {
    long long n = cost.size() - 1;
    long long m = cost[0].size() - 1;
    vector<int> p(m + 1), way(m + 1);
    vector<long long> u(n + 1), v(m + 1);
    for (int i = 1; i <= n; i++) {
        p[0] = i;
        int j0 = 0;
        vector<long long> minv(m + 1, inf);
        vector<char> used(m + 1, false);
        do {
            used[j0] = true;
            int i0 = p[j0], j1 = 0;
            long long delta = inf;
            for (int j = 1; j <= m; j++)
                if (!used[j]) {

```

```
        long long cur = cost[i0][j] - u[i0] - v[j];
        if (cur < minv[j])
            minv[j] = cur, way[j] = j0;
        if (minv[j] < delta)
            delta = minv[j], j1 = j;
    }
    for (int j = 0; j <= m; j++)
        if (used[j])
            u[p[j]] += delta, v[j] -= delta;
        else
            minv[j] -= delta;
    j0 = j1;
} while (p[j0] != 0);
do {
    int j1 = way[j0];
    p[j0] = p[j1];
    j0 = j1;
} while (j0);
}
// For each N, it contains the M it selected
vector<int> ans(n + 1);
for (int j = 1; j <= m ; j++)
    ans[p[j]] = j;
return {-v[0], ans};
}
```

Listing 46: Min Cost Max Flow

```
//cat minCostMaxFlow.h | ./hash.sh
//7c6851
#pragma once

//status: not tested

const long long inf = 1e18;

struct mincostmaxflow {
    typedef long long ll;

    struct edge {
        ll a, b, cap, cost, flow;
        size_t back;
    };

    vector<edge> e;
    vector<vector<ll>> g;
    ll n, s, t;
    ll k = inf; // The maximum amount of flow allowed

    mincostmaxflow(int _n, int _s, int _t) : n(_n), s(_s), t(_t) {
        g.resize(n);
    }

    void addedge(ll a, ll b, ll cap, ll cost) {
        edge e1 = {a, b, cap, cost, 0, g[b].size() };
        edge e2 = {b, a, 0, -cost, 0, g[a].size() };
        g[a].push_back((ll) e.size());
        e.push_back(e1);
        g[b].push_back((ll) e.size());
        e.push_back(e2);
    }
};
```

```
    }

    // Returns {flow, cost}
    pair<ll, ll> getflow() {
        ll flow = 0, cost = 0;
        while (flow < k) {
            vector<ll> id(n, 0);
            vector<ll> d(n, inf);
            vector<ll> q(n);
            vector<ll> p(n);
            vector<size_t> p_edge(n);
            ll qh = 0, qt = 0;
            q[qt++] = s;
            d[s] = 0;
            while (qh != qt) {
                ll v = q[qh++];
                id[v] = 2;
                if (qh == n) qh = 0;
                for (size_t i = 0; i < g[v].size(); i++) {
                    edge& r = e[g[v][i]];
                    if (r.flow < r.cap && d[v] + r.cost < d[r.b]) {
                        d[r.b] = d[v] + r.cost;
                        if (id[r.b] == 0) {
                            q[qt++] = r.b;
                            if (qt == n) qt = 0;
                        } else if (id[r.b] == 2) {
                            if (--qh == -1) qh = n - 1;
                            q[qh] = r.b;
                        }
                        id[r.b] = 1;
                        p[r.b] = v;
                        p_edge[r.b] = i;
                    }
                }
            }
            if (d[t] == inf) break;
            ll addflow = k - flow;
            for (ll v = t; v != s; v = p[v]) {
                ll pv = p[v];
                size_t pr = p_edge[v];
                addflow = min(addflow, e[g[pv][pr]].cap - e[g[pv][pr]].flow);
            }
            for (ll v = t; v != s; v = p[v]) {
                ll pv = p[v];
                size_t pr = p_edge[v], r = e[g[pv][pr]].back;
                e[g[pv][pr]].flow += addflow;
                e[g[v][r]].flow -= addflow;
                cost += e[g[pv][pr]].cost * addflow;
            }
            flow += addflow;
        }
        return {flow, cost};
    }
};
```

Listing 47: MISC

Listing 48: Disjoint Set

```
//cat disjointSet.h | ./hash.sh
//8369d6
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/unionfind
//stress tests: tests/stress-tests/graphs/disjointSet.cpp

struct disjointSet {
    int numSets;
    vector<int> par;
    disjointSet(int n) : numSets(n), par(n, -1) {}
    disjointSet(const disjointSet& rhs) : numSets(rhs.numSets), par(rhs.par) {}
    int find(int x) {
        return par[x] < 0 ? x : par[x] = find(par[x]);
    }
    int sizeOfSet(int x) {
        return -par[find(x)];
    }
    bool merge(int x, int y) {
        if ((x = find(x)) == (y = find(y))) return false;
        if (par[y] < par[x]) swap(x, y);
        par[x] += par[y];
        par[y] = x;
        numSets--;
        return true;
    }
};
```

Listing 49: PBDS

```
//cat policy_based_data_structures.h | ./hash.sh
//807de9
#pragma once

//status: not tested

//place these includes *before* the ‘#define int long long’ else compile error
//not using <bits/extc++.h> as it compile errors on codeforces c++20 compiler
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

//BST with extra functions https://codeforces.com/blog/entry/11080
//order_of_key - # of elements *strictly* less than given element
//find_by_order - find kth largest element, k is 0 based so find_by_order(0) returns min
//<math>\hookrightarrow</math> element
template<class T>
using indexed_set = tree<T, null_type, less<T>, rb_tree_tag,
//<math>\hookrightarrow</math> tree_order_statistics_node_update>;
//example initialization:
indexed_set<pair<long long, int>> is;

//hash table (apparently faster than unordered_map):
//<math>\hookrightarrow</math> https://codeforces.com/blog/entry/60737
//example initialization:
gp_hash_table<string, long long> ht;
```

Listing 50: Count Rectangles

```
//cat cntRectangles.h | ./hash.sh
//16dcb8
#pragma once
//stress tests: tests/stress-tests/misc/cntRectangles.cpp

//given a 2D boolean matrix, calculate cnt[i][j]
//cnt[i][j] = the number of times an (i * j) rectangle appears in the matrix
//such that all cells in the rectangle are false
//Note cnt[0][j] and cnt[i][0] will contain garbage values
//O(R*C)
vector<vector<int>>> getNumRectangles(const vector<vector<bool>>& grid) {
    const int n = grid.size(), m = grid[0].size();
    vector<vector<int>>> cnt(n + 1, vector<int>(m + 1, 0)), arr(n + 2, vector<int>(m + 1,
//<math>\hookrightarrow</math> 0));
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            arr[i][j] = 1 + arr[i][j - 1];
            if (grid[i - 1][j - 1]) arr[i][j] = 0;
        }
    }
    for (int j = 1; j <= m; j++) {
        arr[n + 1][j] = 0;
        stack<pair<int, int>> st;
        st.push({0, 0});
        for (int i = 1; i <= n + 1; i++) {
            pair<int, int> curr = {i, arr[i][j]};
            while (arr[i][j] < st.top().second) {
                curr = st.top();
                st.pop();
                cnt[i - curr.first][curr.second]++;
                cnt[i - curr.first][max(arr[i][j], st.top().second)]--;
            }
            st.push({curr.first, arr[i][j]});
        }
    }
    for (int j = 1; j <= m; j++) {
        for (int k = 0; k < 2; k++) {
            for (int i = n - 1; i >= 1; i--)
                cnt[i][j] += cnt[i + 1][j];
        }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = m - 1; j >= 1; j--)
            cnt[i][j] += cnt[i][j + 1];
    }
    return cnt;
}
```

Listing 51: Longest Increasing Subsequence

```
//cat longest_increasing_subsequence.h | ./hash.sh
//d47c52
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/static_range_lis_query

//returns array of indexes representing the longest *strictly* increasing subsequence
//for non-decreasing: pass in a vector<pair<T, int>> where second is 0, 1, ..., n-1
//alternatively, there’s this https://codeforces.com/blog/entry/13225
template<class T>
vector<int> lis(const vector<T>& arr) {
```

```
if (arr.empty()) return {};
vector<int> dp{0}/*array of indexes into 'arr'*/, prev(arr.size(), -1);
for (int i = 1; i < (int)arr.size(); i++) {
    auto it = lower_bound(dp.begin(), dp.end(), i, [&](int x, int y) -> bool {
        return arr[x] < arr[y];
    });
    if (it == dp.end()) {
        prev[i] = dp.back();
        dp.push_back(i);
    } else {
        prev[i] = it == dp.begin() ? -1 : *(it - 1);
        *it = i;
    }
    //here, dp.size() = length of LIS of prefix of arr ending at index i
}
vector<int> res(dp.size());
for (int i = dp.back(), j = dp.size(); i != -1; i = prev[i])
    res[--j] = i;
return res;
}
```

```

}

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);
}
};

//usage:
unordered_map<long long, int, custom_hash> safe_map;

#include "policy_based_data_structures.h" //not needed when using 'unordered_map'
gp_hash_table<long long, int, custom_hash> safe_hash_table;
```

Listing 52: Monotonic Stack

```
//cat monotonic_stack.h | ./hash.sh
//f4c28f
#pragma once
//library checker tests: https://judge.yosupo.jp/problem/cartesian_tree

//leftLower[i] is the closest smaller index where arr is less.
//Formally: for every index j with leftLower[i] < j < i: arr[j] >= arr[i]
//and
//arr[leftLower[i]] < arr[i] if leftLower[i] != -1
vector<int> monotonic_stack(const vector<int>& arr) {
    int n = arr.size();
    stack<int> st;
    vector<int> leftLower(n, -1);
    for (int i = 0; i < n; i++) {
        while (!st.empty() && arr[st.top()] >= arr[i]) st.pop();
        if (!st.empty()) leftLower[i] = st.top();
        st.push(i);
    }
    return leftLower;
}
```

Listing 53: Safe Hash

```
//cat safehash.h | ./hash.sh
//d9ea53
#pragma once
//status: not tested

//source: https://codeforces.com/blog/entry/62393
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
};
```