International Islamic University Chittagong

IIUC_MARK_US

Arman, Istiaque, Mizan

Graph & Tree

DSU

Description: Disjoint-set data structure with path compression and union by size. Supports `unite` and `findpar`.

Time: $\mathcal{O}(\alpha(N))$, where α is the inverse Ackermann function ($\approx$ constant).

```
{
public:
  vector<int> parent, size;
  int comp;
  DSU(int n) {
    parent.resize(n + 1, 0);
    size.resize(n + 1, 0);
    for (int i = 0; i <= n; i++) {
      parent[i] = i;
      size[i] = 1;
    }
    comp = n;
  }
  int findpar(int node) {
    if (node == parent[node])
      return node;

    return parent[node] = findpar(parent[node]);
  }
  void unite(int u, int v) {
    int ulpar_u = findpar(u);
    int ulpar_v = findpar(v);
    if (ulpar_u == ulpar_v)
      return;
    if (size[ulpar_u] < size[ulpar_v])
      swap(ulpar_u, ulpar_v);
    parent[ulpar_v] = ulpar_u;
    size[ulpar_u] += size[ulpar_v];
    comp--;
  }
};
```

SPFA (Shortest Path Faster Algo)

Description: Queue-optimized Bellman-Ford. Computes single-source shortest paths and detects negative cycles.

Time: Average $\mathcal{O}(E)$, Worst $\mathcal{O}(VE)$.

```
vector<int> dis(node + 1, inf);
queue<int> q;
vector<int> count(node + 1, 0);
vector<bool> inqueue(node + 1, false);
dis[1] = 0;
q.push(1);
while (!q.empty()) {
  int node = q.front();
  q.pop();
  inqueue[node] = false;
  for (auto it : graph[node]) {
    int newnode = it[0];
    int wt = it[1];
    if (dis[newnode] > dis[node] + wt) {
      dis[newnode] = dis[node] + wt;
      if (!inqueue[newnode]) {
        q.push(newnode);
        inqueue[newnode] = true;
        count[newnode]++;
        if (count[newnode] > node) {
          cout << "Negative Cycle Found" << endl;
          return;
        }
      }
    }
  }
}
```

Floyd Warshall

Description: All-pairs shortest path algorithm. Works with negative edges (no negative cycles).

Time: $\mathcal{O}(V^3)$.

```
for (int k = 1; k <= nodes; k++) {
    for (int i = 1; i <= nodes; i++) {
        for (int j = 1; j <= nodes; j++) {
            graph[i][j] = min(graph[i][j], graph[i][k] + graph[k][j]);
        }
    }
}
```

Dijkstra

Description: Single-source shortest path for non-negative edge weights using a priority queue.

Time: $\mathcal{O}(E \log V)$.

```
priority_queue<array<long long, 2>, vector<array<long long, 2>>, greater<array<long long, 2>>> pq;
int n = destination + 1;
vector<long long> dist(n, LONG_LONG_MAX);
vector<int> parent(n);
iota(parent.begin(), parent.end(), 0);

pq.push({0, source});
dist[source] = 0;

while (!pq.empty()) {
    int node = pq.top()[1];
    long long wt = pq.top()[0];
    pq.pop();
    if (wt > dist[node]) continue;
    for (auto it : graph[node]) {
        int newnode = it[0];
        long long newwt = it[1];
        if (dist[node] + newwt < dist[newnode]) {
            dist[newnode] = dist[node] + newwt;
            pq.push({dist[newnode], newnode});
            parent[newnode] = node;
        }
    }
}
```

SCC (Kosaraju)

Description: Finds strongly connected components using two DFS passes. Requires `rev[]` (transpose graph).

Time: $\mathcal{O}(V + E)$.

```
// Assume graph[] and rev[] (reverse graph) are built
{
  int u, v;
  cin >> u >> v;
  graph[u].pb(v);
  rev[v].pb(u);
}
vector<int> vis(n + 1, 0);
vector<int> order;
auto get = [&](auto &&self, int node) -> void {
  vis[node] = 1;
  for (auto it : graph[node]) {
    if (vis[it]) continue;
    self(self, it);
  }

  order.pb(node);
};

for (int i = 1; i <= n; i++) {
  if (vis[i])
    continue;
  get(get, i);
}
vis.assign(n + 1, 0);
reverse(all(order));
vector<int> cur;
vector<int> comp_id(n + 1, 1);
vector<vector<int>> component;
auto rec = [&](auto &&self, int node, int root, int cid) -> void {
  cur.pb(node);
  comp_id[node] = cid;
  vis[node] = 1;
  for (auto it : rev[node]) {
    if (vis[it]) continue;
    self(self, it, root, cid);
  }
};
component.pb({0});
for (auto it : order) {
  if (vis[it]) continue;
  int c = component.size();
  rec(rec, it, it, c);
  component.pb(cur);
  cur.clear();
}

int sz = component.size() - 1;
vector<vector<int>> scc(sz + 5);
for (int u = 1; u <= n; u++) {
  int compu = comp_id[u];
  for (auto v : graph[u]) {
    int compv = comp_id[v];
    if (compu != compv) {
      scc[compu].pb(compv);
    }
  }
}
for (int i = 1; i <= sz; i++) {
  sort(scc[i].begin(), scc[i].end());
  scc[i].erase(unique(scc[i].begin(), scc[i].end()), scc[i].end());
}
```

LCA (Binary Lifting)

Description: Lowest Common Ancestor using binary lifting. `kth` returns the k-th ancestor.

Time: Build $\mathcal{O}(N \log N)$, Query $\mathcal{O}(\log N)$.

```
int LOG = 1;
while ((1 << LOG) <= n) ++LOG;
vector<vector<int>> up(n + 1, vector<int>(LOG + 1, 0));
vector<vector<int>> mx(n + 1, vector<int>(LOG + 1, 0));
vector<vector<int>> mn(n + 1, vector<int>(LOG + 1, 1e9));

auto rec = [&](auto &&self, int node, int par, int cur) -> void {
  parent[node] = par;
  if (par != 0) depth[node] = depth[par] + 1;

  up[node][0] = par;
  mx[node][0] = cur;
  mn[node][0] = cur;
  for (int i = 1; i < LOG; i++) {
    int prev = up[node][i - 1];
    up[node][i] = up[prev][i - 1];
    mx[node][i] = max(mx[node][i - 1], mx[prev][i - 1]);
    mn[node][i] = min(mn[node][i - 1], mn[prev][i - 1]);
  }

  for (auto it : graph[node]) {
    if (it.ff == par)
      continue;
    self(self, it.ff, node, it.ss);
  }
};
rec(rec, 1, 0, 0);
auto kth = [&](int node, int k) -> array<int, 3> {
  int mxx = 0, mnn = 1e9;
  for (int i = 0; i < LOG; i++) {
    if ((1 << i) & k) {
      mxx = max(mxx, mx[node][i]);
      mnn = min(mnn, mn[node][i]);
      node = up[node][i];
    }
  }
  return {node, mnn, mxx};
};
auto lca = [&](int u, int v) -> pair<int, int> {
  int mxx = 0, mnn = 1e9;
  if (depth[u] > depth[v]) {
    auto it = kth(u, depth[u] - depth[v]);
    u = it[0];
    mnn = it[1];
    mxx = it[2];
  }
  else if (depth[v] > depth[u]) {
    auto it = kth(v, depth[v] - depth[u]);
    v = it[0];
```

```
    mnn = it[1];
    mxx = it[2];
  }

  if (u == v)
    return {mnn, mxx};

  for (int i = LOG - 1; i >= 0; i--) {
    if (up[u][i] != up[v][i]) {
      mxx = max({mxx, mx[u][i], mx[v][i]});
      mnn = min({mnn, mn[u][i], mn[v][i]});
      u = up[u][i];
      v = up[v][i];
    }
  }
  mxx = max({mxx, mx[u][0], mx[v][0]});
  mnn = min({mnn, mn[u][0], mn[v][0]});
  return {mnn, mxx};
};
```

Centroid Decomposition

Description: Decomposes a tree into a tree of centroids (depth $\mathcal{O}(\log N)$). update/qry example solves min distance to marked nodes.

Time: Construction $\mathcal{O}(N \log N)$, Queries $\mathcal{O}(\log N)$ or $\mathcal{O}(\log^2 N)$.

```
vector<int> used(n + 1), size(n + 1), parent(n + 1);
vector<int> ans(n + 1, 2e5);
function<int(int, int)> get_size = [&](int node, int par) {
  size[node] = 1;
  for (auto it : graph[node]) {
    if (it == par or used[it])
      continue;
    size[node] += get_size(it, node);
  }
  return size[node];
};
function<int(int, int, int)> get_cen = [&](int node, int par, int sz) {
  for (auto it : graph[node]) {
    if (it == par or used[it])
      continue;
    if (size[it] > sz / 2)
      return get_cen(it, node, sz);
  }
  return node;
};
function<void(int, int)> decompose = [&](int node, int par) {
  int sz = get_size(node, 0);
  int cen = get_cen(node, 0, sz);
  used[cen] = 1;
  if (par == 0)
    par = cen;
  parent[cen] = par;
  for (auto it : graph[cen]) {
    if (used[it])
      continue;
    decompose(it, cen);
  }
};
auto process=[&](int cent)->void {
    vector<int> nodes;
    cnt[0]=1;
    for(auto it : graph[cent]) {
        if(used[it]) continue;
        vector<int> sub_dis;
        auto get_dis=[&](auto &&self,int node,int par,int dis)->void {
            if(dis>k) return;
            sub_dis.pb(dis);
            for(auto it : graph[node]) {
                if(it==par or used[it]) continue;
                self(self,it,node,dis+1);
            }
        };
        get_dis(get_dis,it,cent,1);
        for(auto d : sub_dis) {
            ans+=cnt[k-d];
        }
        for(auto d : sub_dis) {
            if(d<=k) {
                cnt[d]++;
                nodes.pb(d);
            }
        }
    }
    for(auto d : nodes) {
        cnt[d]--;
    }
};

function<void(int)> update = [&](int cur) {
  int x = cur;
  ans[cur] = 0;
  while (1) {
    ans[x] = min(ans[x], getdis(x, cur));
    if (parent[x] == x)
      break;
    x = parent[x];
  }
};
function<int(int)> qry = [&](int cur) {
  int x = cur;
  int go = ans[x];
  while (1) {
    go = min(go, getdis(x, cur) + ans[x]);
    if (parent[x] == x)
      break;
    x = parent[x];
  }
  return go;
};
decompose(1, 0);
update(1);
while (q--) {
  int type;
  cin >> type;
  if (type == 1) {
    int u;
    cin >> u;
    update(u);
  }
  else {
    int u;
    cin >> u;
    cout << qry(u) << el;
  }
}
```

Block cut tree

```
const int N = 200005; // Max nodes (adjust as needed)
vector<int> adj[N];           // Original Graph
vector<int> tree_adj[2 * N]; // Block-Cut Tree (Size 2*N because of block nodes)
int tin[N], low[N];
int timer;
vector<int> stk;
int block_cnt; // Counts the number of blocks found
int n;          // Number of original nodes

// DFS to find Biconnected Components and build the tree
void dfs_bct(int u, int p = -1) {
    tin[u] = low[u] = ++timer;
    stk.push_back(u);

    for (int v : adj[u]) {
        if (v == p) continue;

        if (tin[v]) {
            // Back-edge
            low[u] = min(low[u], tin[v]);
        } else {
            // Tree-edge
            dfs_bct(v, u);
            low[u] = min(low[u], low[v]);

            // Check for Articulation Point / Block condition
            if (low[v] >= tin[u]) {
                block_cnt++;
                int block_node = n + block_cnt; // New node index for this block

                // Add edge between Articulation Point u and the Block Node
                tree_adj[u].push_back(block_node);
                tree_adj[block_node].push_back(u);

                // Pop all nodes in this component from stack
                while (true) {
                    int node = stk.back();
                    stk.pop_back();

                    // Link component node to block node
                    // (Avoid adding u again if it was already added above)
                    if (node != u) {
                        tree_adj[node].push_back(block_node);
                        tree_adj[block_node].push_back(node);
                    }
                    if (node == v) break;
                }
            }
        }
    }
}

void build_bct() {
    timer = 0;
    block_cnt = 0;
    stk.clear();

    // Clear previous tree adjacency if reusing
    for(int i = 1; i <= n * 2; i++) {
        tree_adj[i].clear();
        tin[i] = 0; // 0 means unvisited
    }

    for (int i = 1; i <= n; i++) {
        if (!tin[i]) {
            dfs_bct(i);
            // Handle isolated nodes or leftover stack if needed,
            // but the loop covers standard components.
            if (!stk.empty()) stk.pop_back();
        }
    }
}
```

Bridge and Articulation point

Description: Finds Bridges and Articulation Points in an undirected graph using DFS entry times (tin) and low-link values (low).

Time: $\mathcal{O}(V + E)$.

```
/*
  Finds articulation points and bridges in an undirected simple graph.
  - Nodes are 1..n
  - Input: adjacency list 'adj' where adj[u] contains neighbors of u
  - Output:
      vector<int> is_cut(n+1)  : is_cut[u] == 1 if u is an articulation point
      vector<pair<int,int>> bridges : list of bridges (u,v) with u < v
  Usage:
    build adj (size n+1), then call find_articulation_and_bridges(n, adj, is_cut, bridges)
  tin[v] = discovery time of v in DFS.
  low[v] = smallest discovery time reachable
```

```
      from the subtree of v via at most one back edge (i.e., possibly going up to an ancestor).
  If for a child to of v, low[to] > tin[v], then there's no back edge from to's subtree that reaches v or an ancestor of v. So removing the edge (v,to) disconnects the graph $\rightarrow$ a bridge.
If for a child to of non-root v, low[to] >= tin[v], then removing v disconnects to's subtree from the rest $\to$ v is an articulation point. The root is special: it is an articulation point only if it has $\ge 2$ children in the DFS tree.
*/
void dfs_art_bridge(int v, int p,
const vector<vector<int>> &adj
,vector<int> &tin, vector<int> &low
,vector<int> &is_cut, vector<pair<int, int>> &bridges, int &timer) {
  tin[v] = low[v] = ++timer;
  int children = 0;
  for (int to : adj[v]) {
    if (to == p)
      continue; // skip the edge back to parent (simple graph)
    if (tin[to]) { // back edge
      low[v] = min(low[v], tin[to]);
    }
    else { // tree edge
      ++children;
      dfs_art_bridge(to, v, adj, tin, low, is_cut, bridges, timer);
      low[v] = min(low[v], low[to]);

      // bridge condition (strict)
      if (low[to] > tin[v]) {
        int a = v, b = to;
        if (a > b)
          swap(a, b);
        bridges.emplace_back(a, b);
      }
      // articulation point (non-root)
      if (p != -1 && low[to] >= tin[v]) {
        is_cut[v] = 1;
      }
    }
  }
  // root articulation check
  if (p == -1 && children > 1)
    is_cut[v] = 1;
}
void find_articulation_and_bridges
    (int n, const vector<vector<int>> &adj,
          vector<int> &is_cut,
           vector<pair<int, int>> &bridges)
{
  is_cut.assign(n + 1, 0);
  bridges.clear();
  vector<int> tin(n + 1, 0), low(n + 1, 0);
  int timer = 0;
  for (int i = 1; i <= n; ++i) {
    if (!tin[i])
      dfs_art_bridge(i, -1, adj, tin, low, is_cut, bridges, timer);
  }
  sort(bridges.begin(), bridges.end()); // optional: sorted list of bridges
}
int main() {
  vector<int> is_cut;
  vector<pair<int, int>> bridges;
  find_articulation_and_bridges(n, adj, is_cut, bridges);

  // print articulation points
  vector<int> cuts;
  for (int i = 1; i <= n; ++i)
    if (is_cut[i])
      cuts.push_back(i);
  cout << "Articulation points (" << cuts.size() << "):";
  for (int x : cuts)
    cout << ' ' << x;
  cout << '\n';

  // print bridges
  cout << "Bridges (" << bridges.size() << "):\n";
  for (auto &e : bridges)
    cout << e.first << ' ' << e.second << '\n';
}
```

String

Hashing

Description: Double rolling hash using two sets of mods/bases to minimize collisions. Supports $\mathcal{O}(1)$ substring hash queries after $\mathcal{O}(N)$ precomputation. Uses 1-based indexing for queries.
Time: Construction $\mathcal{O}(N)$, Query $\mathcal{O}(1)$.

```
constexpr int mod1 = 1000012253;
constexpr int mod2 = 1000000009;
constexpr int base1=163;
constexpr int base2=271;
template<typename T>
class MultiHashing {
public:
    int n;
    string s;
    string rev;
    vector<pair<T, T>> prefix_hash;
    vector<pair<T, T>> suffix_hash;
    vector<pair<T, T>> power;
    vector<pair<T, T>> inv;
    T mul(T a, T b, T mod) {
        return ((1LL * a % mod) * (b % mod)) % mod;
    }
    T add(T a, T b, T mod) {
        return (1LL * a + b) % mod;
    }
    T sub(T a, T b, T mod) {
        return ((a % mod) - (b % mod) + 2LL * mod) % mod;
    }
    T bigmod(T base, T power, T mod) {
        T res = 1;
        while (power > 0) {
            if (power & 1) {
                res = mul(res, base, mod);
            }
            base = mul(base, base, mod);
            power >>= 1;
        }
        return res;
    }
    MultiHashing(const string& str) : s(str) {
        n = s.size();
        rev = s;
        reverse(rev.begin(), rev.end());
        prefix_hash.resize(n + 1, {0, 0});
        suffix_hash.resize(n + 1, {0, 0});
        power.resize(n + 1, {0, 0});
        inv.resize(n + 1, {0, 0});
        precom();
    }
    void precom() {
        power[0] = {1, 1};
        for (int i = 1; i <= n; i++) {
            power[i].first = mul(power[i - 1].first, base1, mod1);
            power[i].second = mul(power[i - 1].second, base2, mod2);
        }
        T inv_base1 = bigmod(base1, mod1 - 2, mod1);
        T inv_base2 = bigmod(base2, mod2 - 2, mod2);
        inv[0] = {1, 1};
        for (int i = 1; i <= n; i++) {
            inv[i].first = mul(inv[i - 1].first, inv_base1, mod1);
            inv[i].second = mul(inv[i - 1].second, inv_base2, mod2);
        }
        for (int i = 1; i <= n; i++) {
            int ch = s[i - 1] - 'a' + 1;
            prefix_hash[i].first = add(prefix_hash[i - 1].first, mul(ch, power[i - 1].first, mod1), mod1);
            prefix_hash[i].second = add(prefix_hash[i - 1].second, mul(ch, power[i - 1].second, mod2), mod2);
            ch = rev[i - 1] - 'a' + 1;
            suffix_hash[i].first = add(suffix_hash[i - 1].first, mul(ch, power[i - 1].first, mod1), mod1);
            suffix_hash[i].second = add(suffix_hash[i - 1].second, mul(ch, power[i - 1].second, mod2), mod2);

        }
    }
    pair<T, T> get_hash(int l, int r) {
        T val1 = sub(prefix_hash[r].first, prefix_hash[l - 1].first, mod1);
        val1 = mul(val1, inv[l].first, mod1);

        T val2 = sub(prefix_hash[r].second, prefix_hash[l - 1].second, mod2);
        val2 = mul(val2, inv[l].second, mod2);

        return {val1, val2};
    }
    pair<T, T> get_hash_rev(int l, int r) {
        T val1 = sub(suffix_hash[r].first, suffix_hash[l - 1].first, mod1);
        val1 = mul(val1, inv[l].first, mod1);
        T val2= sub(suffix_hash[r].second, suffix_hash[l - 1].second, mod2);
        val2 = mul(val2, inv[l].second, mod2);
        return {val1, val2};
    }
    pair<T, T> combine_hash(pair<T, T> h1, pair<T, T> h2, int l1) {
        T val1 = add(h1.first, mul(h2.first, power[l1].first, mod1), mod1);
        T val2 = add(h1.second, mul(h2.second, power[l1].second, mod2), mod2);
        return {val1, val2};
    }
};
```

Trie

Description: Prefix tree. insert adds string, count checks existence, erase lazily removes. pref counts words passing through node, end counts words ending at node.
Time: $\mathcal{O}(|S| \cdot \Sigma)$ per operation.

```
class Trie {
public:
    static const int N=26;
    struct Node {
        int next[N];
        int pref;
        int end;
        Node() {
            fill(next,next+N,-1);
            pref=0;
            end=0;
        }
    };
    vector<Node> tree;
    Trie(int sz=1) {
        tree.reserve(sz);
        tree.emplace_back();
    }
    void insert(string &s) {
        int cur=0;
        tree[cur].pref++;
        for(auto it : s) {
```

```
            int ch=it-'a';
            if(tree[cur].next[ch]==-1) {
                tree[cur].next[ch]=(int)tree.size();
                tree.emplace_back();
            }
            cur=tree[cur].next[ch];
            tree[cur].pref++;
        }
        tree[cur].end++;
    }
    int count(string &s) {
        int cur=0;
        for(auto it : s) {
            int ch=it-'a';
            if(tree[cur].next[ch]==-1) return 0;
            cur=tree[cur].next[ch];
        }
        return tree[cur].end;
    }
    int prefixnode(string &s) {
        int cur=0;
        for(auto it : s) {
            int ch=it-'a';
            if(tree[cur].next[ch]==-1) return -1;
            cur=tree[cur].next[ch];
        }
        return cur;
    }
    void erase(string &s) {
        if(count(s)==0) return;
        int cur=0;
        tree[cur].pref--;
        for(auto it : s) {
            int ch=it-'a';
            cur=tree[cur].next[ch];
            tree[cur].pref--;
        }
        tree[cur].end--;
    }
};
```

Z-Function

Description: `z[i]` is the length of the longest common prefix between string s and the suffix starting at i.

Time: $\mathcal{O}(N)$.

```
vector<int> z_function(const string& s) {
    int n = s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min(r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}
```

KMP

Description: `prefix_function` computes $\pi[i]$, the length of the longest proper prefix of $s[0\dots i]$ that is also a suffix of $s[0\dots i]$. `kmp_search` finds all occurrences of pattern.

Time: $\mathcal{O}(N)$.

```
vector<int> compute_pi(const string &p) {
    int m = p.size();
    vector<int> pi(m);
    for (int i = 1, j = 0; i < m; i++) {
        while (j > 0 && p[i] != p[j])
            j = pi[j - 1];
        if (p[i] == p[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
vector<int> kmp(const string &t, const string &p) {
    int n = t.size();
    int m = p.size();
    vector<int> matches;
    vector<int> pi = compute_pi(p);
    for (int i = 0, j = 0; i < n; i++) {
        while (j > 0 && t[i] != p[j])
            j = pi[j - 1];
        if (t[i] == p[j])
            j++;
        if (j == m) {
            matches.push_back(i - m + 1);
            j = pi[m - 1];
        }
    }
    return matches;
}
```

Suffix Array

Description: Constructs Suffix Array using Prefix Doubling & Radix Sort. Uses Kasai's Algorithm for LCP.

Variables:

- `sa[i]`: The starting index of the i-th lexicographically smallest suffix.
- `lcp[i]`: Longest Common Prefix between suffix `sa[i]` and `sa[i-1]`.

Time: Build $\mathcal{O}(N \log N)$, LCP $\mathcal{O}(N)$.

Note: Appends `char(0)` automatically. `sa[0]` is the sentinel. Valid indices $1\dots N$.

```
struct SuffixArray {
    string s;
    int n;
    vector<int> sa;
    vector<int> lcp;

    SuffixArray(string _s):s(_s),n(_s.length()){
        s+=char(0);
        n++;
        constructSA();
        constructLCP();
    }

    void constructSA() {
        const int ALPHABET = 256;
        int m = max(n, ALPHABET);
        sa.resize(n+5);
        vector<int> rank(n+5), new_rank(n+5);
        vector<int> cnt(m+5,0);

        for (int i = 0; i < n; i++){
            rank[i] = (unsigned char)s[i];
            cnt[rank[i]]++;
        }
        for (int i = 1; i < m; i++) cnt[i] += cnt[i - 1];
        for (int i = n - 1; i >= 0; i--) sa[--cnt[rank[i]]] = i;

        vector<int> p(n+5);
        for (int k = 1; k < n; k <<= 1) {
            int cur = 0;

            for (int i = n - k; i < n; i++) p[cur++] = i;
            for (int i = 0; i < n; i++) {
                if (sa[i] >= k) p[cur++] = sa[i] - k;
            }
            fill(cnt.begin(), cnt.begin() + m, 0);
            for (int i = 0; i < n; i++) cnt[rank[p[i]]]++;
            for (int i = 1; i < m; i++) cnt[i] += cnt[i - 1];
            for (int i = n - 1; i >= 0; i--) sa[--cnt[rank[p[i]]]] = p[i];

            new_rank[sa[0]] = 0;
            int classes = 1;
            for (int i = 1; i < n; i++) {
                bool first_half_same = rank[sa[i]] == rank[sa[i - 1]];
                bool second_half_same = true;

                if (sa[i] + k < n && sa[i - 1] + k < n) {
                    second_half_same = (rank[sa[i] + k] == rank[sa[i - 1] + k]);
                } else {
                    second_half_same = (sa[i] + k >= n && sa[i - 1] + k >= n);
                }

                if (!first_half_same || !second_half_same) classes++;
                new_rank[sa[i]] = classes - 1;
            }

            rank = new_rank;
            m = classes;
            if (m == n) break;
        }
    }
    void constructLCP() {
        lcp.assign(n+5, 0);
        vector<int> rank(n+5);
        for (int i = 0; i < n; i++) rank[sa[i]]=i;
        int k = 0;
        for (int i = 0; i < n; i++) {
            if (rank[i] == 0) {
                k = 0;
                continue;
            }
            int j = sa[rank[i] - 1];
            while (i + k < n && j + k < n && (unsigned char)s[i + k] == (unsigned char)s[j + k]) k++;
            lcp[rank[i]] = k;
            if (k > 0) k--;
        }
    }
};
```

Manacher

Description: Computes maximal palindrome lengths. `d1[i]`: max **odd** palindrome centered at i has radius `d1[i]-1`. `d2[i]`: max **even** palindrome centered at i has radius `d2[i]-1`.

Time: $\mathcal{O}(N)$.

```
vector<int> manacher(string s) {
    string t = "^#";
    for (char c : s) {
        t += c;
        t += "#";
    }
    t += "$";
    int n = t.size();
    vector<int> p(n,0);
    int l = 1, r = 1;

    for (int i = 1; i < n - 1; i++) {
        int i_mirror = l + (r - i);
        if (r > i) {
            p[i] = min(r - i, p[i_mirror]);
        }
        while (t[i + 1 + p[i]] == t[i - 1 - p[i]]) {
            p[i]++;
        }
        if (i + p[i] > r) {
            l = i - p[i];
            r = i + p[i];
        }
    }
    return p;
}
```

Data Structure

Sparse Table

Description: Static Range Minimum Query (RMQ). `query` is idempotent ($\mathcal{O}(1)$), `query1` is cascading for non-idempotent functions ($\mathcal{O}(\log N)$).

Time: Build $\mathcal{O}(N \log N)$, Query $\mathcal{O}(1)$.

```cpp
template <typename T>
class SparseTable {
    public:
    vector<vector<T> > st;
    T op(T a,T b) {
        return max(a,b);
    }
    SparseTable(int n,vector<T> &vec) {
        st.resize(n+2,vector<T> (__lg(n)+2));
        for(int i=1;i<=n;i++) {
            st[i][0]=vec[i];
        }
        int k=__lg(n)+1;
        for(int j=1;j<=k;j++) {
            for(int i=1;i+(1<<j)<=n+1;i++) {
                st[i][j]=op(st[i][j-1],st[i+(1<<(j-1))][j-1]);
            }
        }
    }
    T query(int l,int r) {
        int j=__lg(r-l+1);
        return op(st[l][j],st[r-(1<<j)+1][j]);
    }
    T query1(int l,int r) {
        int ans=0;
        for(int j=__lg(r-l+1);j>=0;j--) {
            if((1<<j)<=(r-l+1)) {
                ans=op(ans,st[l][j]);
                l+=(1<<j);
            }
        }
        return ans;
    }
};
```

BIT 1D

Description: Point update, Prefix sum. `lower_bound` finds smallest index i such that sum$(1 \dots i) \geq$ val (requires non-negative values).
Time: $\mathcal{O}(\log N)$.

```cpp
struct BIT {
  int n;
  vector<long long> bit;
  BIT(int n=0){ init(n); }
  void init(int _n) {
      n = _n;
      bit.assign(n+1, 0);
  }
  // add value ‘delta‘ at index i (1-based)
  void add(int i, long long delta) {
      for (; i <= n; i += i & -i) bit[i] += delta;
  }
  // prefix sum [1..i] (1-based)
  long long sumPrefix(int i) {
      long long s = 0;
      for (; i > 0; i -= i & -i) s += bit[i];
      return s;
  }
  // range sum [l..r], 1-based
  long long sumRange(int l, int r) {
      if (r < l) return 0;
      return sumPrefix(r) - sumPrefix(l-1);
  }
  // find smallest index idx such that sumPrefix(idx) >= value (value >= 1)
  // returns n+1 if not found
  int lower_bound(long long value) {
      if (value <= 0) return 1;
      int idx = 0;
      int bitMask = 1;
      while (bitMask << 1 <= n) bitMask <<= 1;
      for (int k = bitMask; k; k >>= 1) {
          int next = idx + k;
          if (next <= n && bit[next] < value) {
              idx = next;
              value -= bit[next];
          }
      }
      return idx + 1;
  }
};
```

BIT 2D

Description: 2D Fenwick Tree for point updates and rectangle sums. 1-based indexing.
Time: $\mathcal{O}(\log N \log M)$.

```cpp
struct BIT2D {
  int n, m;
  vector<vector<long long>> bit;
  BIT2D(int _n=0, int _m=0){ init(_n,_m); }
  void init(int _n, int _m){
      n = _n; m = _m;
      bit.assign(n+1, vector<long long>(m+1, 0));
  }
  // point add at (x,y) (1-based)
  void add(int x, int y, long long delta){
      for (int i = x; i <= n; i += i & -i)
          for (int j = y; j <= m; j += j & -j)
              bit[i][j] += delta;
  }
  // prefix sum (1..x, 1..y)
  long long sumPrefix(int x, int y){
      long long res = 0;
      for (int i = x; i > 0; i -= i & -i)
          for (int j = y; j > 0; j -= j & -j)
              res += bit[i][j];
      return res;
  }
  // rectangle sum (x1,y1) .. (x2,y2), inclusive, 1-based
  long long range_sum(int x1, int y1, int x2, int y2){
      if (x2 < x1 || y2 < y1) return 0;
      return sumPrefix(x2, y2) - sumPrefix(x1-1, y2)
          - sumPrefix(x2, y1-1) + sumPrefix(x1-1, y1-1);
  }
};
```

MO's Algorithm (Hilbert)

Description: Offline range query processing using Hilbert Curve order to improve cache locality and reduce movement. Significantly faster than standard block sorting.
Time: $\mathcal{O}(N\sqrt{Q})$.

```cpp
class dat {
public:
  int l, r, id;
  dat() {};
  dat(int l, int r, int id) {
    this->l = l;
    this->r = r;
    this->id = id;
  }
};
void solve() {
  int n;
  cin >> n;
  vector<int> vec(n);
  for (int i = 0; i < n; i++) {
    cin >> vec[i];
  }
  int dis = 0;
  vector<int> freq(1e6 + 5, 0);
  int block_size = sqrt(n);
  int q;
  cin >> q;
  vector<dat> query(q);
  for (int i = 0; i < q; i++) {
    int l, r;
    cin >> l >> r;
    l--, r--;
    query[i] = dat(l, r, i);
  }
  auto hilbertorder = [&](int x, int y) -> long long {
    const int LOG = 21;
    long long d = 0;
    for (int s = 1 << (LOG - 1); s; s >>= 1) {
      bool rx = x & s, ry = y & s;
      d = (d << 2) | (rx * 3 ^ static_cast<int>(ry));
      if (!ry) {
        if (rx) {
          x = (1 << LOG) - 1 - x;
          y = (1 << LOG) - 1 - y;
        }
        swap(x, y);
      }
    }
    return d;
  };
  vector<pair<long long, int>> order(q);
  for (int i = 0; i < q; i++) {
    order[i] = {hilbertorder(query[i].l, query[i].r), i};
  }
  sort(order.begin(), order.end());
  vector<dat> sorted;
  sorted.reserve(q);
  for (auto [_, idx] : order)
    sorted.push_back(query[idx]);
  query.swap(sorted);
  vector<int> ans(q);
  auto add = [&](int ind) {
    freq[vec[ind]]++;
    if (freq[vec[ind]] == 1)
      dis++;
  };
  auto remove = [&](int ind) {
    freq[vec[ind]]--;
    if (freq[vec[ind]] == 0)
      dis--;
  };
  int L = 0, R = -1;
  for (int i = 0; i < q; i++) {
    int l = query[i].l;
    int r = query[i].r;
    int id = query[i].id;
    while (L > l) {
      add(--L);
    }
    while (R < r) {
      add(++R);
    }
    while (L < l) {
      remove(L++);
    }
    while (R > r) {
      remove(R--);
    }
    ans[id] = dis;
  }

  for (int i = 0; i < q; i++) {
    cout << ans[i] << el;
  }
}
```

Merge Sort Tree

```cpp
class node {
public:
    vector<int> v;
    vector<ll> pref;
    node(){};
    node(int x) {
        v.pb(x);
        pref.resize(1,0);
        pref[0]=x;
    }
};
template <typename Node=node>
class SegmentTree {
public:
    vector<Node> st;
    Node op(Node &a,Node &b) {
        node cur;
        int sz=a.v.size()+b.v.size();
        cur.v.resize(sz,0);
        cur.pref.resize(sz,0);
        merge(all(a.v),all(b.v),cur.v.begin());
        cur.pref[0]=cur.v[0];
        for(int i=1;i<sz;i++) {
            cur.pref[i]=cur.v[i]+cur.pref[i-1];
        }
```

```
        return cur;
    }
    SegmentTree(vector<int> &vec, int n) {
        st.resize(4*n,Node());
        function<void(int, int, int)> build = [&](int id, int start, int end) {
            if (start == end) {
                st[id]=Node(vec[start]);
                return;
            }
            int mid = (start + end) / 2;
            build(2 * id, start, mid);
            build(2 * id + 1, mid + 1, end);
            st[id] = op(st[2*id],st[2*id+1]);
        };
        build(1, 1, n);
    }
    ll query(int id, int start, int end,ll l, ll r,ll k) {
        if (start > r or end < l)
            return 0;
        if (start >= l and end <= r) {
            auto lo=upper_bound(all(st[id].v),k);
            int ind=lo-st[id].v.begin();
            if(ind==0) return 0;
            return st[id].pref[ind-1];
        };
        ll mid = start + (end - start) / 2;
        ll left = query(2 * id, start, mid, l, r,k);
        ll right = query(2 * id + 1, mid + 1, end, l, r,k);
        return (left+right);
    }
};
```

XOR Trie

Description: Binary Trie for integers. Supports finding pair with maximum XOR.
Time: $\mathcal{O}(\log(\max A))$.

```
class TrieNode {
public:
  TrieNode *left;
  TrieNode *right;
  int cnt = 0;
  TrieNode() {
    left = NULL;
    right = NULL;
    cnt = 0;
  }
};
class Trie {
  TrieNode *root;

public:
  Trie() {
    root = new TrieNode();
  }
  void insert(int n) {
    TrieNode *curr = root;
    for (int i = 31; i >= 0; i--) {
      int bit = (1 & (n >> i));
      if (bit == 0) {
        if (curr->left == NULL) {
          curr->left = new TrieNode();
        }
        curr = curr->left;
        curr->cnt++;
      }
      else {
        if (curr->right == NULL) {
          curr->right = new TrieNode();
        }
        curr = curr->right;
        curr->cnt++;
      }
    }
  }
  void remove(int n) {
    TrieNode *curr = root;
    for (int i = 31; i >= 0; i--) {
      if (curr == NULL)
        break;
      int bit = (n >> i) & 1;
      if (bit == 0) {
        curr = curr->left;
        curr->cnt--;
      }
      else {
        curr = curr->right;
        curr->cnt--;
      }
    }
  }
  int max_xor_pair(int n) {
    TrieNode *curr = root;
    int ans = 0;
    for (int i = 31; i >= 0; i--) {
      if (curr == NULL) {
        break;
      }
      int bit = (1 & (n >> i));
      if (bit == 0) {
        if (curr->right != NULL and curr->right->cnt > 0) {
          ans += (1 << i);
          curr = curr->right;
        }
        else
          curr = curr->left;
      }
      else {
        if (curr->left != NULL and curr->left->cnt > 0) {
          ans += (1 << i);
          curr = curr->left;
        }
        else
          curr = curr->right;
      }
    }
    return ans;
  }
};
```

LAZY SegTree

Description: Standard Lazy Propagation for range updates.
Time: $\mathcal{O}(\log N)$.

```
struct ST{
    int n;
    vector<int> t,lazy,arr;
    void init(int n) {
        this->n=n;
        t.assign(3*n+5,0);
        lazy.assign(3*n+5,0);
        arr.assign(n+5,0);
    }
    inline void push(int node,int l,int r){
        if(!lazy[node]) return;
        t[node]+=lazy[node]*(r-l+1); // check here
        if(l!=r){
            lazy[node*2]+=lazy[node];
            lazy[node*2+1]+=lazy[node];
        }
        lazy[node]=0;
    }
    inline void here(int node){
        t[node]=t[node*2]+t[node*2+1];  // check here
    }
    void build(int node,int l,int r){
        lazy[node]=0;
        if(l==r){
            t[node]=arr[l];
            return;
        }
        ll mid=(l+r)>>1;
        build(node*2,l,mid);
        build(node*2+1,mid+1,r);
        here(node);
    }
    void upd(int node,int l,int r,int i,int j,int value){
        push(node,l,r);
        if(l>j || r<i) return;
        if(i<=l && r<=j){
            lazy[node]+=value; // check here
            push(node,l,r);
            return;
        }
        ll mid=(l+r)>>1;
        upd(node*2,l,mid,i,j,value);
        upd(node*2+1,mid+1,r,i,j,value);
        here(node);
    }
    ll query(int node,int l,int r,int i,int j ){
        push(node,l,r);
        if(l>j || r<i) return 0;          /// check here
        if(i<=l && r<=j) return t[node];
        ll mid=(l+r)>>1;
        return query(node*2,l,mid,i,j)+query(node*2+1,mid+1,r,i,j); // check here
    }
}t;
```

PST (Persistent SegTree)

Description: Persistent segment tree. `add_copy` branches off a version.
Time: $\mathcal{O}(\log N)$ query/update. **Space:** $\mathcal{O}(Q \log N)$.

```
class PST{
    private:
        struct node{
            ll sum=0;
            int lc=0,rc=0; // left child right child
        };
    const int n;
    vector<node> tree;
    int timer=1;
    node join(int lc,int rc){
        return node{tree[lc].sum+tree[rc].sum,lc,rc}; // check here
    }
    int build_(int l,int r,const vector<int> &arr){
        int id=timer++;
        if(l==r){
            tree[id]={arr[l],0,0}; // check here
            return id;
        }
        int mid=(l+r)>>1;
        tree[id]=join(build_(l,mid,arr),build_(mid+1,r,arr));
        return id;
    }
    int upd_(int v,int l,int r,int pos,int val){
        int id=timer++;
        if(l==r){
            tree[id]={val,0,0}; // check here
            return id;
        }
        int mid=(l+r)>>1;
        if(pos<=mid) tree[id]=join(upd_(tree[v].lc,l,mid,pos,val),tree[v].rc);
        else tree[id]=join(tree[v].lc,upd_(tree[v].rc,mid+1,r,pos,val));
        return id;
    }
    ll query_(int v,int l,int r,int i,int j){
        if(l>j || r<i) return 0LL; /// check here
        if(i<=l && r<=j) return tree[v].sum;
        int mid=(l+r)>>1;
        return query_(tree[v].lc,l,mid,i,j)+
               query_(tree[v].rc,mid+1,r,i,j);
    }
    public:
    PST(int n,int mx_nodes) : n(n),tree(mx_nodes) {}
    int build(const vector<int> &arr) { return build_(1,n,arr); }
    int upd(int root,int pos,int val) { return upd_(root,1,n,pos,val); }
    ll query(int root,int l,int r) { return query_(root,1,n,l,r); }
    int add_copy(int root){
        tree[timer]=tree[root];
```

```cpp
        return timer++;
    }
};
int32_t main()
{
    const int mx_nodes=2*n+q*(2+__lg(n));
    PST t(n,mx_nodes);
    vector<int> roots = {t.build(a)};
    while(q--){
        int type,k; cin>>type>>k;
        k--;
        if(type==1){
            int pos,val; cin>>pos>>val;
            roots[k]=t.upd(roots[k],pos,val);
        }else if(type==2){
            int a,b; cin>>a>>b;
            cout<<t.query(roots[k],a,b)<<endl;
        }else{
            roots.PB(t.add_copy(roots[k]));
        }
    }
}
```

Dynamic SegTree

Description: Segment tree with sparse coordinates ($N \approx 10^9$). Nodes created on demand.

Time: $\mathcal{O}(\log(\text{Range}))$.

```cpp
class SparseSegTree {
private:
    struct node {
        ll freq=0;
        ll lazy=0;
        int left=0;
        int right=0;
        bool lazy_flag=false;
    };
    vector<node> tree;
    const ll n;
    int timer=1;
    // int comb(int a,int b) { return a+b; }
    void apply(int cur,ll l,ll r,ll val) { // check here
        tree[cur].lazy=val;
        tree[cur].lazy_flag=true;
        tree[cur].freq=(r-l+1)*val;
    }
    void push_down(int cur,ll l,ll r){
        if(!tree[cur].left){
            tree[cur].left= ++timer;
            tree.PB(node());
        }
        if(!tree[cur].right){
            tree[cur].right= ++timer;
            tree.PB(node());
        }
        if(!tree[cur].lazy_flag) return;
        ll mid=(l+r)>>1;
        apply(tree[cur].left,l,mid,tree[cur].lazy);
        apply(tree[cur].right,mid+1,r,tree[cur].lazy);
        tree[cur].lazy_flag=false;
        tree[cur].lazy=0;
    }
    void upd(int cur,ll l,ll r,ll ql,ll qr,ll val) {
        if(qr<l || ql>r) return;
        if(ql<=l && r<=qr) apply(cur,l,r,val);
        else {
            push_down(cur,l,r);
            ll mid=(l+r)>>1;
            upd(tree[cur].left,l,mid,ql,qr,val);
            upd(tree[cur].right,mid+1,r,ql,qr,val);
            tree[cur].freq=
                tree[tree[cur].left].freq + tree[tree[cur].right].freq; // check here
        }
    }
    ll query(int cur,ll l,ll r,ll ql,ll qr) {
        if(qr<l || ql>r || !cur) return 0;
        if(ql<=l && r<=qr) return tree[cur].freq;
        push_down(cur,l,r);
        ll mid=(l+r)>>1;
        return query(tree[cur].left,l,mid,ql,qr) +
               query(tree[cur].right,mid+1,r,ql,qr); // check here
    }
public:
    SparseSegTree(ll n,int q=0) : n(n) {
        if(q>0) { tree.reserve(2*q*__lg(n)); }
        tree.PB(node()); tree.PB(node());
    }
    void upd(ll ql,ll qr,ll val) { upd(1,1,n,ql,qr,val); }
    int query(ll ql,ll qr) {return query(1,1,n,ql,qr); }
};
int32_t main(){
    const int range_size=1e9;
    SparseSegTree st(range_size+1,q); // pass n+q if there is n given
}
```

Wavelet Tree

Description: Partitions array based on values. kth: k-th smallest in range. LTE: count values $\leq k$. count: range value freq.

Time: $\mathcal{O}(\log(\max A))$ per query.

```cpp
struct wavelet_tree
{
  int lo, hi;
  wavelet_tree *l, *r;
  vi b;
  wavelet_tree(int *from, int *to, int x, int y)
  {
    lo = x, hi = y;
    if (lo == hi or from >= to)
      return;
    int mid = (lo + hi) / 2;
    auto f = [mid](int x)
    {
      return x <= mid;
    };
    b.reserve(to - from + 1);
    b.pb(0);
    for (auto it = from; it != to; it++)
      b.pb(b.back() + f(*it));
    // see how lambda function is used here
    auto pivot = stable_partition(from, to, f);
    l = new wavelet_tree(from, pivot, lo, mid);
    r = new wavelet_tree(pivot, to, mid + 1, hi);
  }
  // kth smallest element in [l, r]
  int kth(int l, int r, int k)
  {
    if (l > r)
      return 0;
    if (lo == hi)
      return lo;
    int inLeft = b[r] - b[l - 1];
    int lb = b[l - 1]; // amt of nos in first (l-1) nos that go in left
    int rb = b[r];     // amt of nos in first (r) nos that go in left
    if (k <= inLeft)
      return this->l->kth(lb + 1, rb, k);
    return this->r->kth(l - lb, r - rb, k - inLeft);
  }
  // count of nos in [l, r] Less than or equal to k
  int LTE(int l, int r, int k)
  {
    if (l > r or k < lo)
      return 0;
    if (hi <= k)
      return r - l + 1;
    int lb = b[l - 1], rb = b[r];
    return this->l->LTE(lb + 1, rb, k) + this->r->LTE(l - lb, r - rb, k);
  }
  int count(int l, int r, int k)
  {
    if (l > r or k < lo or k > hi)
      return 0;
    if (lo == hi)
      return r - l + 1;
    int lb = b[l - 1], rb = b[r], mid = (lo + hi) / 2;
    if (k <= mid)
      return this->l->count(lb + 1, rb, k);
    return this->r->count(l - lb, r - rb, k);
  }
  ~wavelet_tree()
  {
    delete l;
    delete r;
  }
};
int main()
{
  wavelet_tree T(a + 1, a + n + 1, 1, MAX);
}
```

SEGTree Beats (main)

Description: "Jiry Match" Tree. Supports Range Chmin ($a_i = \min(a_i, x)$), Chmax, Add, Set, Mod, Divide, Negative. Handles history/break conditions.

Time: Amortized $\mathcal{O}((N + Q) \log N)$.

```cpp
const ll INF=1e18;
const ll NINF=-1e18;
struct STBeats {
private:
    struct node{
        ll max1;                 // max value
        ll max2;                 // second max value
        int max_cnt;             // cnt of the largest value
        ll min1;                 // min value
        ll min2;                 // second min value
        int min_cnt;             // cnt of the smallest value
        ll sum;                  // sum of the range
        int len;                 // length of the range
        ll lazy_add;             // lazy teg
        ll lazy_set;
        bool lazy_neg;
        node() : max1(NINF),max2(NINF),max_cnt(0),
            min1(INF),min2(INF),min_cnt(0),sum(0),len(0),
            lazy_add(0),lazy_set(INF),lazy_neg(false) {}
    };

    int n;
    vector<node> tree;
    inline node merge(const node& left, const node& right) {       // O(1)
        node res;
        res.sum=left.sum+right.sum;
        res.len=left.len+right.len;
        res.lazy_add=0;
        res.lazy_set=INF;
        res.lazy_neg=false;
        if(left.max1>right.max1) { // merging max data for chmin
            res.max1=left.max1;
            res.max2=max(left.max2,right.max1);
            res.max_cnt=left.max_cnt;
        }else if(left.max1<right.max1) {
            res.max1=right.max1;
            res.max2=max(left.max1,right.max2);
            res.max_cnt=right.max_cnt;
        }else if(left.max1==right.max1) {
            res.max1=left.max1;
            res.max2=max(left.max2,right.max2
```

```
);
            res.max_cnt=left.max_cnt+right.max_cnt;
        }

        if(left.min1<right.min1) {  // margin min data for chmax
            res.min1=left.min1;
            res.min2=min(left.min2,right.min1);
            res.min_cnt=left.min_cnt;
        }else if(left.min1>right.min1) {
            res.min1=right.min1;
            res.min2=min(left.min1,right.min2);
            res.min_cnt=right.min_cnt;
        }else if(left.min1==right.min1) {
            res.min1=left.min1;
            res.min2=min(left.min2,right.min2);
            res.min_cnt=left.min_cnt+right.min_cnt;
        }
        return res;
    }
    inline void apply_negative(int v) {
        swap(tree[v].max1,tree[v].min1);
        swap(tree[v].max2,tree[v].min2);
        swap(tree[v].max_cnt,tree[v].min_cnt);
        tree[v].max1*=-1;
        if(tree[v].max2!=NINF) tree[v].max2*=-1;
        tree[v].min1*=-1;
        if(tree[v].min2!=INF) tree[v].min2*=-1;
        tree[v].sum*=-1;
        if(tree[v].lazy_set!=INF) tree[v].lazy_set*=-1;
        else tree[v].lazy_add*=-1;
        tree[v].lazy_neg^=1;
    }
    inline void apply_add(int v,ll x) {                // O(1)
        if(!x) return;
        tree[v].sum+=tree[v].len*x;
        tree[v].max1+=x;
        if(tree[v].max2!=NINF) tree[v].max2+=x;
        tree[v].min1+=x;
        if(tree[v].min2!=INF) tree[v].min2+=x;

        if(tree[v].lazy_set!=INF) tree[v].lazy_set+=x;
        else tree[v].lazy_add+=x;
    }

    inline void apply_set(int v,ll x) {
        tree[v].max1=x;
        tree[v].max2=NINF;
        tree[v].max_cnt=tree[v].len;
        tree[v].min1=x;
        tree[v].min2=INF;
        tree[v].min_cnt=tree[v].len;
        tree[v].sum=tree[v].len*x;
        tree[v].lazy_add=0;
        tree[v].lazy_set=x;
        tree[v].lazy_neg=false;
    }
    inline void apply_chmin(int v,ll x) {             // O(1)
        if(x>=tree[v].max1) return;
        tree[v].sum-=tree[v].max_cnt*(tree[v].max1-x);
        if(tree[v].min1==tree[v].max1) tree[v].min1=x;
        if(tree[v].min2==tree[v].max1) tree[v].min2=x;
        tree[v].max1=x;

        if(tree[v].lazy_set !=INF)
            tree[v].lazy_set=min(tree[v].lazy_set,x);
    }

    inline void apply_chmax(int v,ll x) {             // O(1)
        if(x<=tree[v].min1) return;
        tree[v].sum+=tree[v].min_cnt*(x-tree[v].min1);
        if(tree[v].max1==tree[v].min1) tree[v].max1=x;
        if(tree[v].max2==tree[v].min1) tree[v].max2=x;
        tree[v].min1=x;

        if(tree[v].lazy_set !=INF)
            tree[v].lazy_set=max(tree[v].lazy_set,x);
    }

    void push_lazy(int v,int tl,int tr) {                // O(1)
        if(tl==tr) return;
        if(tree[v].lazy_set!=INF) {
            apply_set(2*v,tree[v].lazy_set);
            apply_set(2*v+1,tree[v].lazy_set);
            tree[v].lazy_set=INF;
            return;
        }
        if(tree[v].lazy_neg) {
            apply_negative(2*v);
            apply_negative(2*v+1);
            tree[v].lazy_neg=false;
        }
        if(tree[v].lazy_add!=0) {        // for lazy add
            apply_add(2*v,tree[v].lazy_add);
            apply_add(2*v+1,tree[v].lazy_add);
            tree[v].lazy_add=0;
        }
    }
    void push_beats(int v,int tl,int tr) {
        if(tl==tr) return;
        apply_chmin(2*v,tree[v].max1);
        apply_chmin(2*v+1,tree[v].max1);
        apply_chmax(2*v,tree[v].min1);
        apply_chmax(2*v+1,tree[v].min1);
    }

    void build_(int v,int tl,int tr,const vector<ll>& a) {        // O(n)
        if(tl==tr){
            tree[v].len=1;
            tree[v].sum=a[tl];
            tree[v].max1=a[tl];
            tree[v].max_cnt=1;
            tree[v].max2=NINF;
            tree[v].min1=a[tl];
            tree[v].min_cnt=1;
            tree[v].min2=INF;
            tree[v].lazy_add=0;
            tree[v].lazy_set=INF;
            tree[v].lazy_neg=false;
        }else{
            int mid=(tl+tr)>>1;
            build_(2*v,tl,mid,a);
            build_(2*v+1,mid+1,tr,a);
            tree[v]=merge(tree[2*v],tree[2*v+1]);
        }
    }

    void upd_min_(int v,int tl,int tr,int ql,int qr,ll x) {   // O(log^2 n)
        push_lazy(v,tl,tr);
        if(tree[v].max1<=x || qr<tl || tr<ql) return;
        if(ql<=tl && tr<=qr && tree[v].max2<x) {
            apply_chmin(v,x);
            return;
        }
        push_beats(v,tl,tr);
        int mid=(tl+tr)>>1;
        upd_min_(2*v,tl,mid,ql,qr,x);
        upd_min_(2*v+1,mid+1,tr,ql,qr,x);
        tree[v]=merge(tree[2*v],tree[2*v+1]);
    }

    void upd_max_(int v,int tl,int tr,int ql,int qr,ll x) {   // O(log^2 n)
        push_lazy(v,tl,tr);
        if(tree[v].min1>=x || qr<tl || tr<ql) return;
        if(ql<=tl && tr<=qr && tree[v].min2>x) {
            apply_chmax(v,x);
            return;
        }
        push_beats(v,tl,tr);
        int mid=(tl+tr)>>1;
        upd_max_(2*v,tl,mid,ql,qr,x);
        upd_max_(2*v+1,mid+1,tr,ql,qr,x);
        tree[v]=merge(tree[2*v],tree[2*v+1]);
    }

    void upd_add_(int v,int tl,int tr,int ql,int qr,ll x) {   // O(log n)
        if(qr<tl || tr<ql) return;
        if(ql<=tl && tr<=qr) {
            apply_add(v,x);
            return;
        }
        push_lazy(v,tl,tr);
        push_beats(v,tl,tr);
        int mid=(tl+tr)>>1;
        upd_add_(2*v,tl,mid,ql,qr,x);
        upd_add_(2*v+1,mid+1,tr,ql,qr,x);
        tree[v]=merge(tree[2*v],tree[2*v+1]);
    }

    void upd_set_(int v,int tl,int tr,int ql,int qr,ll x) {   // O(log n) range set
        if(qr<tl || tr<ql) return;
        if(ql<=tl && tr<=qr) {
            apply_set(v,x);
            return;
        }
        push_lazy(v,tl,tr);
        push_beats(v,tl,tr);
        int mid=(tl+tr)>>1;
        upd_set_(2*v,tl,mid,ql,qr,x);
        upd_set_(2*v+1,mid+1,tr,ql,qr,x);
        tree[v]=merge(tree[2*v],tree[2*v+1]);
    }

    void upd_mod_(int v,int tl,int tr,int ql,int qr,ll x) {   // O(log^2 n)
        push_lazy(v,tl,tr);
        if(tree[v].max1<x || qr<tl || tr<ql) return;
        if(tl==tr) {
            apply_set(v,tree[v].sum%x);
            return;
        }
        push_beats(v,tl,tr);
        int mid=(tl+tr)>>1;
        upd_mod_(2*v,tl,mid,ql,qr,x);
        upd_mod_(2*v+1,mid+1,tr,ql,qr,x);
        tree[v]=merge(tree[2*v],tree[2*v+1]);
    }
    ll floor_div(ll a,ll b) {
        if(b<0) { a=-a,b=-b; }
        ll d=a/b;
        ll r=a%b;
        if(r<0) return d-1;
        return d;
    }
    void upd_negative_(int v,int tl,int tr,int ql,int qr) {
        if(qr<tl || tr<ql) return;
        if(ql<=tl && tr<=qr) {
            apply_negative(v);
            return;
        }
        push_lazy(v,tl,tr);
        push_beats(v,tl,tr);
        int mid=(tl+tr)>>1;
        upd_negative_(2*v,tl,mid,ql,qr);
        upd_negative_(2*v+1,mid+1,tr,ql,qr);
        tree[v]=merge(tree[2*v],tree[2*v+1]);
    }
    void upd_divide_(int v,int tl,int tr,int ql,int qr,ll x) {    // O(log^2 n)
```

```
        if(x==1) return;
        if(x==-1){
            upd_negative_(v,tl,tr,ql,qr);
            return;
        }
        if(qr<tl || tr<ql || !x) return;
        push_lazy(v,tl,tr);
        ll new_min=floor_div(tree[v].min1,x);
        ll new_max=floor_div(tree[v].max1,x);
        if(ql<=tl && tr<=qr && new_min==
            ↪ new_max){
            apply_set(v,new_min);
            return;
        }
        if(tl==tr) {
            ll val=floor_div(tree[v].sum,x);
            apply_set(v,val);
            return;
        }
        push_beats(v,tl,tr);
        int mid=(tl+tr)>>1;
        upd_divide_(2*v,tl,mid,ql,qr,x);
        upd_divide_(2*v+1,mid+1,tr,ql,qr,x);
        tree[v]=merge(tree[2*v],tree[2*v+1]);
    }

    ll query_sum_(int v,int tl,int tr,int ql,
        ↪ int qr) { // O(log n)
        if(qr<tl || tr<ql) return 0;
        if(ql<=tl && tr<=qr) return tree[v].
            ↪ sum;
        push_lazy(v,tl,tr);
        push_beats(v,tl,tr);
        int mid=(tl+tr)>>1;
        return query_sum_(2*v,tl,mid,ql,qr) +
            ↪  query_sum_(2*v+1,mid+1,tr,ql
            ↪ ,qr);
    }
    ll query_max_(int v,int tl,int tr,int ql,
        ↪ int qr) { // O(log n)
        if(qr<tl || tr<ql) return NINF;
        if(ql<=tl && tr<=qr) return tree[v].
            ↪ max1;
        push_lazy(v,tl,tr);
        push_beats(v,tl,tr);
        int mid=(tl+tr)>>1;
        return max(query_max_(2*v,tl,mid,ql,
            ↪ qr) , query_max_(2*v+1,mid+1,
            ↪ tr,ql,qr));
    }
    ll query_min_(int v,int tl,int tr,int ql,
        ↪ int qr) { // O(log n)
        if(qr<tl || tr<ql) return INF;
        if(ql<=tl && tr<=qr) return tree[v].
            ↪ min1;
        push_lazy(v,tl,tr);
        push_beats(v,tl,tr);
        int mid=(tl+tr)>>1;
        return min(query_min_(2*v,tl,mid,ql,
            ↪ qr) , query_min_(2*v+1,mid+1,
            ↪ tr,ql,qr));
    }

public:
    STBeats(int n_val) : n(n_val) { tree.
        ↪ resize(4*n+4); }
    void build(const vector<ll>& a) { build_
        ↪ (1,1,n,a); }
    void upd_min(int ql,int qr,ll x) {
        ↪ upd_min_(1,1,n,ql,qr,x); }
    void upd_max(int ql,int qr,ll x) {
        ↪ upd_max_(1,1,n,ql,qr,x); }
    void upd_add(int ql,int qr,ll x) {
        ↪ upd_add_(1,1,n,ql,qr,x); }
    void upd_set(int ql,int qr,ll x) {
        ↪ upd_set_(1,1,n,ql,qr,x); }
    void upd_mod(int ql,int qr,ll x) {
        ↪ upd_mod_(1,1,n,ql,qr,x); }
    void upd_divide(int ql,int qr,ll x) {
        ↪ upd_divide_(1,1,n,ql,qr,x); }
    ll query_sum(int ql,int qr) { return
        ↪ query_sum_(1,1,n,ql,qr); }
    ll query_max(int ql,int qr) { return
        ↪ query_max_(1,1,n,ql,qr); }
    ll query_min(int ql,int qr) { return
        ↪ query_min_(1,1,n,ql,qr); }
};
int32_t main(){
    ios_base :: sync_with_stdio(0); cin.tie
        ↪ (0);
    int t=1;
    // cin>>t;
    while(t--){
      int n; cin>>n;
      int q; cin>>q;
      STBeats t(n);
      vector<ll> v(n+1);
      for(int i=1;i<=n;i++) cin>>v[i];
      t.build(v);
      while(q--){
        int type,l,r; cin>>type>>l>>r;
        if(type==1) {
          ll val; cin>>val;
          t.upd_divide(l,r,val);
        }else if(type==2){
          ll val; cin>>val;
          t.upd_set(l,r,val);
        }else cout<<t.query_sum(l,r)<<endl;
      }
    }
}

/*
 @ the bellow code is dedicated for range and
    ↪  range divide it is more faster
    ↪ divide then main struct
 cause it is dedicated only for make divide
    ↪ very faster
*/

struct STBeats_Light {
private:
    struct node {
        ll sum;
        ll min1;
        ll max1;
        ll lazy_add;
        node() : sum(0), min1(INF), max1(NINF
            ↪ ), lazy_add(0) {}
    };
    int n;
    vector<node> tree;
    void pull(int v) {
        tree[v].sum = tree[2 * v].sum + tree
            ↪ [2 * v + 1].sum;
        tree[v].min1 = min(tree[2 * v].min1,
            ↪ tree[2 * v + 1].min1);
        tree[v].max1 = max(tree[2 * v].max1,
            ↪ tree[2 * v + 1].max1);
    }
    void apply_add(int v, int tl, int tr, ll
        ↪ x) {
        tree[v].sum += (tr - tl + 1) * x;
        tree[v].min1 += x;
        tree[v].max1 += x;
        tree[v].lazy_add += x;
    }
    void push(int v, int tl, int tr) {
        if (tree[v].lazy_add == 0) return;
        int mid = (tl + tr) >> 1;
        apply_add(2 * v, tl, mid, tree[v].
            ↪ lazy_add);
        apply_add(2 * v + 1, mid + 1, tr,
            ↪ tree[v].lazy_add);
        tree[v].lazy_add = 0;
    }
    void build_(int v, int tl, int tr, const
        ↪ vector<ll>& a) {
        // here write the build function from
            ↪  main STBeats
    }
    void upd_add_(int v, int tl, int tr, int
        ↪ ql, int qr, ll x) {
        if (qr < tl || tr < ql) return;
        if (ql <= tl && tr <= qr) {
            apply_add(v, tl, tr, x);
            return;
        }
        push(v, tl, tr);
        int mid = (tl + tr) >> 1;
        upd_add_(2 * v, tl, mid, ql, qr, x);
        upd_add_(2 * v + 1, mid + 1, tr, ql,
            ↪ qr, x);
        pull(v);
    }
    ll floor_div(ll a, ll b) {
        // here write the floor_div function
            ↪ from main STBeats
    }
    void upd_divide_(int v, int tl, int tr,
        ↪ int ql, int qr, ll x) {
        if (qr < tl || tr < ql) return;
        if (ql <= tl && tr <= qr) {
            ll new_min = floor_div(tree[v].
                ↪ min1, x);
            ll new_max = floor_div(tree[v].
                ↪ max1, x);
            ll delta_min = new_min - tree[v].
                ↪ min1;
            ll delta_max = new_max - tree[v].
                ↪ max1;
            if (delta_min == delta_max) {
                apply_add(v, tl, tr,
                    ↪ delta_min);
                return;
            }
            if (tl == tr) {
                ll new_val = floor_div(tree[v].
                    ↪ min1, x);
                tree[v].sum = tree[v].min1 = tree
                    ↪ [v].max1 = new_val;
                return;
            }
        }
        push(v, tl, tr);
        int mid = (tl + tr) >> 1;
        upd_divide_(2 * v, tl, mid, ql, qr, x
            ↪ );
        upd_divide_(2 * v + 1, mid + 1, tr,
            ↪ ql, qr, x);
        pull(v);
    }
    ll query_sum_(int v, int tl, int tr, int
        ↪ ql, int qr) {
        // here write the query_sum_ function
            ↪  from main STBeats
    }
    ll query_min_(int v, int tl, int tr, int
        ↪ ql, int qr) {
        // here write the query_min_ function
            ↪  from main STBeats
    }

public:
    STBeats_Light(int n_val) : n(n_val) {
        ↪ tree.resize(4 * n + 4); }
    void build(const vector<ll>& a) { build_
        ↪ (1, 1, n, a); }
    void upd_add(int ql, int qr, ll x) {
        ↪ upd_add_(1, 1, n, ql, qr, x); }
    void upd_divide(int ql, int qr, ll x) {
        ↪ upd_divide_(1, 1, n, ql, qr, x);
        ↪ }
    ll query_sum(int ql, int qr) { return
        ↪ query_sum_(1, 1, n, ql, qr); }
    ll query_min(int ql, int qr) { return
        ↪ query_min_(1, 1, n, ql, qr); }
};
```

SEGTree Beats Bit and Gcd

Description:
1. **Bitwise:** Range AND/OR/Set using `tree[v].all_or` and `tree[v].all_and` to detect if update affects range.
2. **GCD:** Range GCD + Min/Max/Add.

```
const ll INF=1e18;
const ll NINF=-1e18;

struct STBeats_Bit {
private:
    struct node {
        ll sum;
        int len;
        ll all_and;
        ll all_or;
        ll max_val;
        ll min_val;
        ll lazy_set;
```

```
        node() : sum(0),len(0),all_and(~0LL), all_or(0LL),
                max_val(NINF),min_val(INF), lazy_set(INF) {}
    };
    int n;
    vector<node> tree;
    node merge(const node& left,const node& right) {
        node res;
        res.sum=left.sum+right.sum;
        res.len=left.len+right.len;
        res.all_and=left.all_and & right.all_and;
        res.all_or=left.all_or | right.all_or;
        res.max_val=max(left.max_val,right.max_val);
        res.min_val=min(left.min_val,right.min_val);
        res.lazy_set=INF;
        return res;
    }
    void apply_set(int v,ll x) {
        tree[v].sum=tree[v].len*x;
        tree[v].all_and=x;
        tree[v].all_or=x;
        tree[v].max_val=x;
        tree[v].min_val=x;
        tree[v].lazy_set=x;
    }
    void push_down(int v,int tl,int tr) {
        if(tl==tr || tree[v].lazy_set==INF) return;
        apply_set(2*v,tree[v].lazy_set);
        apply_set(2*v+1,tree[v].lazy_set);
        tree[v].lazy_set=INF;
    }
    void build_(int v,int tl,int tr,const vector<ll>& a) {
        if(tl==tr) {
            tree[v].len=1;
            tree[v].sum=a[tl];
            tree[v].all_and=a[tl];
            tree[v].all_or=a[tl];
            tree[v].max_val=a[tl];
            tree[v].min_val=a[tl];
        }else {
            int mid=(tl+tr)>>1;
            build_(2*v,tl,mid,a);
            build_(2*v+1,mid+1,tr,a);
            tree[v]=merge(tree[2*v],tree[2*v+1]);
        }
    }
    void upd_or_(int v,int tl,int tr,int ql,int qr,ll x) {
        push_down(v,tl,tr);
        if(qr<tl || tr<ql) return;
        if((tree[v].all_and & x)==x) return;
        if(tl==tr) {
            apply_set(v,tree[v].sum | x);
            return;
        }
        int mid=(tl+tr)>>1;
        upd_or_(2*v,tl,mid,ql,qr,x);
        upd_or_(2*v+1,mid+1,tr,ql,qr,x);
        tree[v]=merge(tree[2*v],tree[2*v+1]);
    }
    void upd_and_(int v,int tl,int tr,int ql,int qr,ll x) {
        push_down(v,tl,tr);
        if(qr<tl || tr<ql) return;
        if((tree[v].all_or | x)==x) return;
        if(tl==tr) {
            apply_set(v,tree[v].sum & x);
            return;
        }
        int mid=(tl+tr)>>1;
        upd_and_(2*v,tl,mid,ql,qr,x);
        upd_and_(2*v+1,mid+1,tr,ql,qr,x);
        tree[v]=merge(tree[2*v],tree[2*v+1]);
    }
    void upd_set_(int v,int tl,int tr,int ql,int qr,ll x) {
        push_down(v,tl,tr);
        if(qr<tl || tr<ql) return;
        if(ql<=tl && tr<=qr) {
            apply_set(v,x);
            return;
        }
        int mid=(tl+tr)>>1;
        upd_set_(2*v,tl,mid,ql,qr,x);
        upd_set_(2*v+1,mid+1,tr,ql,qr,x);
        tree[v]=merge(tree[2*v],tree[2*v+1]);
    }
    ll query_sum_(int v,int tl,int tr,int ql,int qr) {
        if(qr<tl || tr<ql) return 0;
        push_down(v,tl,tr);
        if(ql<=tl && tr<=qr) return tree[v].sum;
        int mid=(tl+tr)>>1;
        return query_sum_(2*v,tl,mid,ql,qr) +
               query_sum_(2*v+1,mid+1,tr,ql,qr);
    }
    ll query_and_(int v,int tl,int tr,int ql,int qr) {
        if(qr<tl || tr<ql) return ~0LL;
        push_down(v,tl,tr);
        if(ql<=tl && tr<=qr) return tree[v].all_and;
        int mid=(tl+tr)>>1;
        return query_and_(2*v,tl,mid,ql,qr) &
               query_and_(2*v+1,mid+1,tr,ql,qr);
    }
    ll query_or_(int v,int tl,int tr,int ql,int qr) {
        if(qr<tl || tr<ql) return 0LL;
        push_down(v,tl,tr);
        if(ql<=tl && tr<=qr) return tree[v].all_or;
        int mid=(tl+tr)>>1;
        return query_or_(2*v,tl,mid,ql,qr) |
               query_or_(2*v+1,mid+1,tr,ql,qr);
    }
    ll query_max_(int v,int tl,int tr,int ql,int qr) {
        if(qr<tl || tr<ql) return NINF;
        push_down(v,tl,tr);
        if(ql<=tl && tr<=qr) return tree[v].max_val;
        int mid=(tl+tr)>>1;
        return max(query_max_(2*v,tl,mid,ql,qr) ,
                   query_max_(2*v+1,mid+1,tr,ql,qr));
    }
    ll query_min_(int v,int tl,int tr,int ql,int qr) {
        if(qr<tl || tr<ql) return INF;
        push_down(v,tl,tr);
        if(ql<=tl && tr<=qr) return tree[v].min_val;
        int mid=(tl+tr)>>1;
        return min(query_min_(2*v,tl,mid,ql,qr) ,
                   query_min_(2*v+1,mid+1,tr,ql,qr));
    }

public:
    STBeats_Bit(int n) : n(n) { tree.resize(4*n+4); }
    void build(const vector<ll>& a) { build_(1,1,n,a); }
    void upd_or(int ql,int qr,ll x) { upd_or_(1,1,n,ql,qr,x); }
    void upd_and(int ql,int qr,ll x) { upd_and_(1,1,n,ql,qr,x); }
    void upd_set(int ql,int qr,ll x) { upd_set_(1,1,n,ql,qr,x); }
    ll query_sum(int ql,int qr) { return query_sum_(1,1,n,ql,qr); }
    ll query_and(int ql,int qr) { return query_and_(1,1,n,ql,qr); }
    ll query_or(int ql,int qr) { return query_or_(1,1,n,ql,qr); }
    ll query_max(int ql,int qr) { return query_max_(1,1,n,ql,qr); }
    ll query_min(int ql,int qr) { return query_min_(1,1,n,ql,qr); }
};
int32_t main() {
    ios_base :: sync_with_stdio(0); cin.tie(0);

    int n,q; cin>>n>>q;
    vector<ll> a(n+1);
    for(int i=1;i<=n;i++) cin>>a[i];
    STBeats_Bit t(n);
    t.build(a);
}

/*
this code is for gcd and multiple update like  cmax cmin add set
*/

#include<bits/stdc++.h>

using namespace std;

const long long MX = 1e18;

struct node {
    long long max, max2, min, min2, sum, gcd, add = 0, set = 0, updmin = 0, updmax = 0;
    int cntmax, cntmin;
    node() {}
    node(long long x) {
        sum = max = min = x, cntmax = cntmin = 1;
        gcd = 0;
        max2 = -MX, min2 = MX;
    }
};

vector<node> t;
vector<long long> a;

void merge(node& res, node& a, node& b) {
    // max
    res.max = max(a.max, b.max);
    res.max2 = -MX;
    res.cntmax = 0;
    if (a.max == res.max) {
        res.cntmax += a.cntmax;
        res.max2 = max(res.max2, a.max2);
    } else {
        res.max2 = max(res.max2, a.max);
    }
    if (b.max == res.max) {
        res.cntmax += b.cntmax;
        res.max2 = max(res.max2, b.max2);
    } else {
        res.max2 = max(res.max2, b.max);
    }

    // min
    res.min = min(a.min, b.min);
    res.min2 = MX;
    res.cntmin = 0;
    if (a.min == res.min) {
        res.cntmin += a.cntmin;
        res.min2 = min(res.min2, a.min2);
    } else {
        res.min2 = min(res.min2, a.min);
    }
    if (b.min == res.min) {
        res.cntmin += b.cntmin;
        res.min2 = min(res.min2, b.min2);
    } else {
        res.min2 = min(res.min2, b.min);
    }

    //sum
    res.sum = a.sum + b.sum;

    //gcd
    res.gcd = __gcd(a.gcd, b.gcd);
    long long x = -1, y = -1;
    if (a.max2 != -MX && a.max2 != a.min) {
        x = a.max2;
    }
    if (b.max2 != -MX && b.max2 != b.min) {
```

```
        y = b.max2;
    }
    if (x != -1 && y != -1) {
        res.gcd = __gcd(res.gcd, abs(x - y));
    }
    for (long long z : {a.max, a.min, b.max,
        ↪ b.min}) {
        if (z == res.max) {
            continue;
        }
        if (z == res.min) {
            continue;
        }
        if (x != -1) {
            res.gcd = __gcd(res.gcd, abs(x -
                ↪ z));
        } else if (y != -1) {
            res.gcd = __gcd(res.gcd, abs(y -
                ↪ z));
        } else {
            x = z;
        }
    }
}

void push_add(int v, long long x) {
    if (t[v].set != 0) {
        t[v].set += x;
    } else {
        if (t[v].updmin != 0) {
            t[v].updmin += x;
        }
        if (t[v].updmax != 0) {
            t[v].updmax += x;
        }
        t[v].add += x;
    }
}

void push_max(int v, long long x) {
    if (t[v].set != 0) {
        t[v].set = min(t[v].set, x);
    } else if (t[v].updmin == 0 || x > t[v].
        ↪ updmin) {
        if (t[v].updmax == 0) {
            t[v].updmax = x;
        } else {
            t[v].updmax = min(t[v].updmax, x)
                ↪ ;
        }
    } else {
        t[v].set = x;
    }
}

void push_min(int v, long long x) {
    if (t[v].set != 0) {
        t[v].set = max(t[v].set, x);
    } else if (t[v].updmax == 0 || t[v].
        ↪ updmax > x) {
        if (t[v].updmin == 0) {
            t[v].updmin = x;
        } else {
            t[v].updmin = max(t[v].updmin, x)
                ↪ ;
        }
    } else {
        t[v].set = x;
    }
}

void push(int v, int l, int r) {
    if (t[v].set != 0) {
        if (l + 1 != r) {
            t[v * 2 + 1].set = t[v * 2 + 2].
                ↪ set = t[v].set;
        }
        t[v].max = t[v].min = t[v].set;
        t[v].cntmax = t[v].cntmin = r - l;
        t[v].sum = t[v].set * (long long) (r
            ↪ - l);
        t[v].add = t[v].set = t[v].gcd = t[v
            ↪ ].updmin = t[v].updmax = 0;
        t[v].max2 = -MX, t[v].min2 = MX;
    }
    if (t[v].add != 0) {
        if (l + 1 != r) {
            push_add(v * 2 + 1, t[v].add);
            push_add(v * 2 + 2, t[v].add);
        }
        t[v].max += t[v].add;
        t[v].min += t[v].add;
        if (t[v].max2 != -MX) {
            t[v].max2 += t[v].add;
        }
        if (t[v].min2 != MX) {
            t[v].min2 += t[v].add;
        }
        t[v].sum += t[v].add * (long long) (r
            ↪  - l);
        t[v].add = 0;
    }
    if (t[v].updmax != 0) {
        if (l + 1 != r) {
            push_max(v * 2 + 1, t[v].updmax);
            push_max(v * 2 + 2, t[v].updmax);
        }
        if (t[v].max == t[v].min) {
            if (t[v].updmax < t[v].max) {
                t[v].sum = t[v].updmax * (
                    ↪ long long) (r - l);
                t[v].max = t[v].min = t[v].
                    ↪ updmax;
            }
        } else {
            if (t[v].updmax < t[v].max) {
                t[v].sum -= (t[v].max - t[v].
                    ↪ updmax) * (long long)
                    ↪  t[v].cntmax;
                if (t[v].max == t[v].min2) {
                    t[v].min2 = t[v].updmax;
                }
                t[v].max = t[v].updmax;
            }
        }
        t[v].updmax = 0;
    }
    if (t[v].updmin != 0) {
        if (l + 1 != r) {
            push_min(v * 2 + 1, t[v].updmin);
            push_min(v * 2 + 2, t[v].updmin);
        }
        if (t[v].max == t[v].min) {
            if (t[v].updmin > t[v].min) {
                t[v].sum = t[v].updmin * (
                    ↪ long long) (r - l);
                t[v].max = t[v].min = t[v].
                    ↪ updmin;
            }
        } else {
            if (t[v].updmin > t[v].min) {
                t[v].sum += (t[v].updmin - t[
                    ↪ v].min) * (long long)
                    ↪  t[v].cntmin;
                if (t[v].min == t[v].max2) {
                    t[v].max2 = t[v].updmin;
                }
                t[v].min = t[v].updmin;
            }
        }
        t[v].updmin = 0;
    }
}

void build(int v, int l, int r) {
    if (l + 1 == r) {
        t[v] = node(a[l]);
        return;
    }
    int m = (l + r) / 2;
    build(v * 2 + 1, l, m);
    build(v * 2 + 2, m, r);
    merge(t[v], t[v * 2 + 1], t[v * 2 + 2]);
}

void updatemin(int v, int l, int r, int l1,
    ↪ int r1, long long x) {
    push(v, l, r);
    if (l1 >= r || l >= r1 || t[v].max <= x)
        ↪ return;
    if (l1 <= l && r <= r1 && t[v].max2 < x)
        ↪ {
        t[v].updmax = x;
        push(v, l, r);
        return;
    }
    int m = (l + r) / 2;
    updatemin(v * 2 + 1, l, m, l1, r1, x);
    updatemin(v * 2 + 2, m, r, l1, r1, x);
    merge(t[v], t[v * 2 + 1], t[v * 2 + 2]);
}

void updatemax(int v, int l, int r, int l1,
    ↪ int r1, long long x) {
    push(v, l, r);
    if (l1 >= r || l >= r1 || t[v].min >= x)
        ↪ return;
    if (l1 <= l && r <= r1 && t[v].min2 > x)
        ↪ {
        t[v].updmin = x;
        push(v, l, r);
        return;
    }
    int m = (l + r) / 2;
    updatemax(v * 2 + 1, l, m, l1, r1, x);
    updatemax(v * 2 + 2, m, r, l1, r1, x);
    merge(t[v], t[v * 2 + 1], t[v * 2 + 2]);
}

void updateset(int v, int l, int r, int l1,
    ↪ int r1, long long x) {
    push(v, l, r);
    if (l1 >= r || l >= r1) return;
    if (l1 <= l && r <= r1) {
        t[v].set = x;
        push(v, l, r);
        return;
    }
    int m = (l + r) / 2;
    updateset(v * 2 + 1, l, m, l1, r1, x);
    updateset(v * 2 + 2, m, r, l1, r1, x);
    merge(t[v], t[v * 2 + 1], t[v * 2 + 2]);
}

void updateadd(int v, int l, int r, int l1,
    ↪ int r1, long long x) {
    push(v, l, r);
    if (l1 >= r || l >= r1) return;
    if (l1 <= l && r <= r1) {
        t[v].add = x;
        push(v, l, r);
        return;
    }
    int m = (l + r) / 2;
    updateadd(v * 2 + 1, l, m, l1, r1, x);
    updateadd(v * 2 + 2, m, r, l1, r1, x);
    merge(t[v], t[v * 2 + 1], t[v * 2 + 2]);
}

long long getsum(int v, int l, int r, int l1,
    ↪  int r1) {
    push(v, l, r);
    if (l1 >= r || l >= r1) return 0ll;
    if (l1 <= l && r <= r1) return t[v].sum;
    int m = (l + r) / 2;
    return getsum(v * 2 + 1, l, m, l1, r1) +
        ↪ getsum(v * 2 + 2, m, r, l1, r1);
}

long long getmin(int v, int l, int r, int l1,
    ↪  int r1) {
    push(v, l, r);
    if (l1 >= r || l >= r1) return MX;
    if (l1 <= l && r <= r1) return t[v].min;
    int m = (l + r) / 2;
    return min(getmin(v * 2 + 1, l, m, l1, r1
        ↪ ), getmin(v * 2 + 2, m, r, l1, r1
        ↪ ));
}

long long getmax(int v, int l, int r, int l1,
    ↪  int r1) {
    push(v, l, r);
    if (l1 >= r || l >= r1) return -MX;
    if (l1 <= l && r <= r1) return t[v].max;
    int m = (l + r) / 2;
    return max(getmax(v * 2 + 1, l, m, l1, r1
        ↪ ), getmax(v * 2 + 2, m, r, l1, r1
        ↪ ));
}
```

```
long long getgcd(int v, int l, int r, int l1,
    ↪ int r1) {
    push(v, l, r);
    if (l1 >= r || l >= r1) return 0ll;
    if (l1 <= l && r <= r1) {
        long long res = __gcd(t[v].max, t[v].
            ↪ min);
        if (t[v].max2 != t[v].min && t[v].
            ↪ max2 != -MX) {
            res = __gcd(res, t[v].gcd);
            res = __gcd(res, t[v].max2);
        }
        return res;
    }
    int m = (l + r) / 2;
    return __gcd(getgcd(v * 2 + 1, l, m, l1,
        ↪ r1), getgcd(v * 2 + 2, m, r, l1,
        ↪ r1));
}
```

SEGTree with Hashing

```
const ll p=137; const ll N=2e5+10; // check
    ↪ range
const pair<ll,ll> mod={127657753,987654319};
ll powerr(ll a,ll b,ll mod){
  ll r=1;
  while(b){
    if(b%2) r=((r%mod) *(a%mod))%mod;
    a=((a%mod)*(a%mod))%mod;
    b/=2;
  }
  return r;
}
ll add(ll a,ll b,ll mod){return ((a%mod)+(b%
    ↪ mod)+mod)%mod;}
ll substract(ll a,ll b,ll mod){return ((a%mod
    ↪ )-(b%mod)+mod)%mod;}
ll mult(ll a,ll b,ll mod) {return ((a%mod)*(b
    ↪ %mod))%mod;}
ll fn(char ch){if(islower(ch)) return ch-'a'
    ↪ +1;if(isupper(ch)) return ch-'A'+1;
    ↪ return ch-'0'+1;}
// ll fn(ll a[i]) return a[i]; //for integer
    ↪ hash

pair<ll,ll> pw[N+10],inv[N+10],inv_p_minus1;
void precal(){
  pw[0].F=pw[0].S=1;
  for(int i=1;i<N;i++){
    pw[i].F=mult(pw[i-1].F,p,mod.F);
    pw[i].S=mult(pw[i-1].S,p,mod.S);
  }
  ll pw_inv1=powerr(p,mod.F-2,mod.F);
  ll pw_inv2=powerr(p,mod.S-2,mod.S);
  inv[0].F=inv[0].S=1;
  for(int i=1;i<N;i++){
    inv[i].F=mult(inv[i-1].F,pw_inv1,mod.F);
    inv[i].S=mult(inv[i-1].S,pw_inv2,mod.S);
  }
    inv_p_minus1 = {
        powerr(p-1, mod.F-2, mod.F),
        powerr(p-1, mod.S-2, mod.S)
    };
}
struct hashing {
  vector<pair<ll,ll>> t;
  vector<char>lazy; // lazy of integer for
      ↪ integer hash
  string s; // integer hash make vector<ll> a
  hashing(){}
  hashing(string _s){
    s=_s;
    ll n=s.size();
    t.resize(n*4);
    lazy.resize(n*4,'?');
  }
  inline void push(int node,int l,int r){
    if(lazy[node]=='?') return;
    ll len=(r-l+1);
    ll sum1 = mult(mult(substract(pw[len].F,
        ↪ 1, mod.F), inv_p_minus1.F, mod.F)
        ↪ , pw[l].F, mod.F);
    ll sum2 = mult(mult(substract(pw[len].S,
        ↪ 1, mod.S), inv_p_minus1.S, mod.S)
        ↪ , pw[l].S, mod.S);

    t[node].F = mult(sum1, fn(lazy[node]),
        ↪ mod.F);
    t[node].S = mult(sum2, fn(lazy[node]),
        ↪ mod.S);
    if(l!=r){
        lazy[node*2]=lazy[node*2+1]=lazy[node
          ↪ ];
    }
    lazy[node]='?';
  }
  inline void here(int node){
      t[node].F=add(t[node*2].F,t[node*2+1].F
          ↪ ,mod.F);
      t[node].S=add(t[node*2].S,t[node*2+1].S
          ↪ ,mod.S);
  }
  void build(int node,int l,int r){
    if(l==r){
        t[node].F=mult(pw[l].F,fn(s[l]),mod.F
          ↪ );
        t[node].S=mult(pw[l].S,fn(s[l]),mod.S
          ↪ );
        return;
    }
    ll mid=(l+r)>>1;
    build(node*2,l,mid);
    build(node*2+1,mid+1,r);
    here(node);
  }
  void upd(int node,int l,int r,int i,int j,
      ↪ char value){
    push(node,l,r);
    if(l>j || r<i) return;
    if(i<=l && r<=j){
        lazy[node]=value;
        push(node,l,r);
        return;
    }
    ll mid=(l+r)>>1;
    upd(node*2,l,mid,i,j,value);
    upd(node*2+1,mid+1,r,i,j,value);
    here(node);
  }
  pair<ll,ll> query(int node,int l,int r,int
      ↪ i,int j){
    push(node,l,r);
    if(l>j || r<i) return {0,0};          //
        ↪ / check here
    if(i<=l && r<=j) return t[node];
    ll mid=(l+r)>>1;
    pair<ll,ll> x=query(node*2,l,mid,i,j);
    pair<ll,ll> y=query(node*2+1,mid+1,r,i,j)
        ↪ ;
    return {add(x.F,y.F,mod.F),add(x.S,y.S,
        ↪ mod.S)};
  }
  pair<ll,ll> get_hash(int l,int r,int n){
    pair<ll,ll> ck=query(1,0,n-1,l,r);
    ck.F=mult(ck.F,inv[l].F,mod.F);
    ck.S=mult(ck.S,inv[l].S,mod.S);
    return ck;
  }
}a;
int main(){
  precal();
  ll n,m,x; cin>>n>>m>>x;
  ll q=m+x;
  string s; cin>>s;
  a = hashing(s);
  a.build(1,0,n-1);
  while(q--){
    ll i; cin>>i;
    if(i==1){
      ll l,r; char c; cin>>l>>r>>c; l--,r--;
      a.upd(1,0,n-1,l,r,c);
    }else{
      ll l,r,d; cin>>l>>r>>d;
      --l,--r;
      if(d==(r-l+1) || a.get_hash(l,r-d,n)==a
          ↪ .get_hash(l+d,r,n))
      cout<<"YES"<<endl;
      else cout<<"NO"<<endl;
    }
  }
}
```

Fast Fourier Transform (FFT)

Description: Iterative Cooley-Tukey FFT. Computes convolution of two polynomials A and B.

Optimization: Packs A into real part and B into imaginary part ($P(x) = A(x) + iB(x)$) to compute DFT of both using **a single forward FFT** call. Total ops: 1 Forward FFT + 1 Inverse FFT.

Time: $\mathcal{O}(N \log N)$, where N is the smallest power of $2 \geq |A| + |B| - 1$.

Note: Uses `complex<double>`. Precision errors may occur for result values $> 10^{14}$.

```
class FFT {
    using cd = complex<double>;
    static constexpr double PI = acos(-1.0);
    void fft(vector<cd>& a, bool invert)
        ↪ const {
        int n = (int)a.size();
        for (int i = 1, j = 0; i < n; ++i) {
            int bit = n >> 1;
            for (; j & bit; bit >>= 1) j ^=
                ↪ bit;
            j ^= bit;
            if (i < j) swap(a[i], a[j]);
        }

        for (int len = 2; len <= n; len <<=
            ↪ 1) {
            double ang = 2 * PI / len * (
                ↪ invert ? -1 : 1);
            cd wlen(cos(ang), sin(ang));
            for (int i = 0; i < n; i += len)
                ↪ {
                cd w(1);
                for (int k = 0; k < len/2; ++
                    ↪ k) {
                    cd u = a[i + k];
                    cd v = a[i + k + len/2] *
                        ↪ w;
                    a[i + k] = u + v;
                    a[i + k + len/2] = u - v;
                    w *= wlen;
                }
            }
        }
        if (invert) for (cd & x : a) x /= n;
    }

    static int next_pow2(int x) {
        int n = 1;
        while (n < x) n <<= 1;
        return n;
    }

public:

    vector<long long> multiply(const vector<
        ↪ ll>& A, const vector<ll>& B )
        ↪ const {
        if (A.empty() || B.empty()) return
            ↪ {};
        int n = (int)A.size(), m = (int)B.
            ↪ size();
        int need = n + m - 1;
        int sz = next_pow2(need);

        vector<cd> fa(sz);
        for (int i = 0; i < n; ++i) fa[i].
            ↪ real((double)A[i]);
        for (int i = 0; i < m; ++i) fa[i].
            ↪ imag((double)B[i]);

        fft(fa, false);

        vector<cd> fb(sz);
        for (int i = 0; i < sz; ++i) {
            int j = (i == 0 ? 0 : sz - i);
            cd a1 = (fa[i] + conj(fa[j])) *
                ↪ cd(0.5, 0.0);
            cd b1 = (fa[i] - conj(fa[j])) *
                ↪ cd(0.0, -0.5);
            fb[i] = a1 * b1;
        }
```

```
        fft(fb, true);

        vector<long long> res(need);
        for (int i = 0; i < need; ++i) res[i]
            =llround(fb[i].real());
        return res;
    }
};
```

Number Theory

nCr & nPr

```
const ll mod=1e9+7;
ll fact[69];
ll poW(ll x, ll n){
    ll result = 1;
    while (n > 0){
        if (n & 1LL == 1){
            result = (result * x)%mod;
        }
        x = (x * x)%mod;
        n = n >> 1LL;
    }
    return result%mod;
}
ll nCr(ll n,ll r){
    return (fact[n] * poW((fact[r]*fact[n-r])
        %mod,mod-2)) % mod;
}
ll nPr(ll n,ll r){
    return (fact[n] * poW(fact[n-r]%mod,mod
        -2)) % mod;
}
int32_t main(){
    fact[0]=1;
    for(int i=1;i<=60;i++){
        fact[i]=(fact[i-1]*i*1LL)%mod;
    }
}
```

Sieve & Primes

Description: Linear Sieve (spf), Segmented Sieve, Segmented Factorization, $\phi(n)$ (Euler Totient), Factorization.
Time: Sieve $\mathcal{O}(N)$, Factorize $\mathcal{O}(\log N)$ (with spf) or $\mathcal{O}(\sqrt{N})$.

```
struct NumberTheory {
  static ll power(ll x, ll n) {
    ll res = 1;
    while (n > 0) {
      if (n & 1)
        res *= x;
      x *= x;
      n >>= 1;
    }
    return res;
  }
  vector<ll> primes;
  vector<int> spf;
  void sieve(ll n) { // O(n)
    spf.assign(n + 1, 0);
    for (int i = 2; i <= n; ++i) {
      if (!spf[i]) {
        spf[i] = i;
        primes.PB(i);
      }
      for (auto j : primes) {
        ll prime = j;
        ll composite_num = 1LL * i * prime;
        if (composite_num > n)
          break;
        spf[composite_num] = prime;
        if (prime == spf[i])
          break;
      }
    }
  }
  vector<ll> segmentedSieve(ll L, ll R) {
    vector<bool> mark(R - L + 1, true);
    if (L == 1)
      mark[0] = false;
    for (auto p : primes) {
      if (1LL * p * p > R)
        break;
      ll base = max(p * p, ((L + p - 1) / p)
          * p);
      for (ll j = base; j <= R; j += p)
        mark[j - L] = false;
    }
    vector<ll> seg;
    for (ll i = 0; i <= R - L; i++)
      if (mark[i])
        seg.push_back(L + i);
    return seg;
  }
  vector<vector<ll>> segment_factor;
  void segment_fact(ll L, ll R) {
    segment_factor.assign(R - L + 1, vector<
        ll>());
    vector<ll> range_primes(R - L + 1);
    for (ll i = 0; i <= R - L; i++)
      range_primes[i] = L + i;
    for (auto p : primes) {
      if (1LL * p * p > R)
        break;
      ll base = p * ((L + p - 1) / p);

      for (ll j = base; j <= R; j += p) {
        ll index = j - L;
        while (!(range_primes[index] % p)) {
          segment_factor[index].PB(p);
          range_primes[index] /= p;
        }
      }
    }
    for (ll i = 0; i <= R - L; i++) {
      if (range_primes[i] <= 1)
        continue;

      segment_factor[i].PB(range_primes[i]);
    }
  }
  vector<ll> factorize(ll n) {
    vector<ll> f;
    for (auto p : primes) {
      if (1LL * p * p > n)
        break;
      while (n % p == 0) {
        f.push_back(p);
        n /= p;
      }
    }
    if (n > 1)
      f.push_back(n);
    return f;
  }
  ll phi(ll n) {
    ll res = n;
    for (auto p : primes) {
      if (1LL * p * p > n)
        break;
      if (n % p == 0) {
        while (n % p == 0)
          n /= p;
        res -= res / p;
      }
    }
    if (n > 1)
      res -= res / n;
    return res;
  }
  ll phi2(ll n) {
    vector<ll> v = factorize(n);
    map<ll, ll> mp;
    ll res = 1;
    for (int i = 0; i < v.size(); ++i) {
      ll p = v[i], exp = 0;
      while (i < v.size() && v[i] == p) {
        exp++;
        i++;
      }
      i--;
      res *= power(p, exp - 1) * (p - 1);
    }
    return res;
  }
  static ll xorUpto(ll n) {
    ll x = n % 4;
    if (x == 0)
      return n;
    if (x == 1)
      return 1;
    if (x == 2)
      return n + 1;
    return 0;
  }
  static ll nCr(ll n, ll r) {
    if (r > n)
      return 0;
    r = min(r, n - r);
    ll res = 1;
    for (ll i = 1; i <= r; i++) {
      res = res * (n - i + 1) / i;
    }
    return res;
  }
} P;
```

Pollard Rho & Miller Rabin

Description: Deterministic Miller-Rabin primality test (up to 10^{18}) and Pollard's Rho factorization. Requires __int128 for modular multiplication to avoid overflow.
Time: Primality $\mathcal{O}(k \log^3 N)$, Factorization $\mathcal{O}(N^{1/4})$.

```
// this is the topic to find prime fact of a
    big number
using ll = unsigned long long;
mt19937_64 rng(chrono::steady_clock::now().
    time_since_epoch().count());
ll rand(ll n) { return rng() % (n - 2) + 1; }
ll modMul(ll a,ll b,ll mod) {
    return (__int128)a*b%mod;
}
ll modPower(ll base,ll exp,ll mod) {
    ll res=1;
    base%=mod;
    while(exp>0) {
        if(exp%2==1) res=modMul(res,base,mod)
            ;
        base=modMul(base,base,mod);
        exp/=2;
    }
    return res;
}
ll gcd(ll a,ll b) {
    while(b) {
        a%=b;
        swap(a,b);
    }
    return a;
}
const int MAX_SIEVE=1000001;
vector<int> spf(MAX_SIEVE);
void init_sieve() {
    vector<int> primes;
    for(int i=2;i<MAX_SIEVE;++i) {
        if(!spf[i]) {
            spf[i]=i;
            primes.PB(i);
        }
        for(int p:primes) {
            if(i*(ll)p>=MAX_SIEVE) break;
            spf[i*p]=p;
            if(!(i%p)) break;
        }
    }
}
bool MillerRabin(ll n,ll a,ll d,int s) {
    ll x=modPower(a,d,n);
    if(x==1 || x==n-1) return true;
    for(int r=1;r<s;r++) {
        x=modMul(x,x,n);
        if(x==1) return false;
        if(x==n-1) return true;
    }
    return false;
}
bool isPrime(ll n) {
    if(n<=1) return false;
    if(n<MAX_SIEVE) return spf[n]==n;
    if(n==2 || n==3) return true;
    if(!(n%2)) return false;
    ll d=n-1;
    int s=0;
    while(!(d%2)) {
        d/=2;
        s++;
    }
```

```
    vector<ll> witnesses
        ={2,3,5,7,11,13,17,19,23,29,31,37};
    for(ll a:witnesses) {
        if(n==a) return true;
        if(!(MillerRabin(n,a,d,s))) return
            false;
    }
    return true;
}
ll pollard_rho(ll n) {
    auto f =[&](ll x,ll c) {
        return (modMul(x,x,n)+c)%n;
    };
    ll c=rand(n);
    ll tortoise=2,hare=2,d=1;
    ll product=1;
    const int BATCH_SIZE=128;
    int count=0;
    while(1) {
        tortoise=f(tortoise,c);
        hare=f(f(hare,c),c);
        if(tortoise==hare) {
            c=rand(n);
            tortoise=2; hare=2; product=1;
                count=0;
            continue;
        }
        ll prev_product=product,diff;
        if(tortoise>hare) diff=tortoise-hare;
        else diff=hare-tortoise;
        product=modMul(product,diff,n);
        if(!product) {
            d=gcd(prev_product,n);
            if(d==1) d=gcd(diff,n);
            break;
        }
        count++;
        if(count==BATCH_SIZE) {
            d=gcd(product,n);
            if(d>1) break;
            count=0;
            product=1;
        }
    }
    if(d==n || d==1) return pollard_rho(n);
    return d;
}
void factorize(ll n,vector<ll>& primeFactors)
    {
    if(n<=1) return;
    while(!(n%2)) {
        primeFactors.PB(2);
        n/=2;
    }
    if(n==1) return;
    while(n>1 && n<MAX_SIEVE) {
        primeFactors.PB(spf[n]);
        n/=spf[n];
    }
    if(n==1) return;
    if(isPrime(n)) {
        primeFactors.PB(n);
        return;
    }
    ll d=pollard_rho(n);
    factorize(d,primeFactors);
    factorize(n/d,primeFactors);
}
int32_t main() {
    init_sieve(); // run it before testcase
    ll n; cin>>n;
    vector<ll> ans;
    factorize(n,ans);
}
```

Mobius Function

Description: Linear Sieve to compute $\mu(i)$ and $\phi(i)$. $h[i]$ stores helper values for LCM sums.

Time: $\mathcal{O}(N)$.

```
const int MX=1000001;
vector<int> mu(MX);
vector<int> phi(MX);
vector<int> spf(MX);
vector<ll> h(MX,0); // for LCM
vector<int> primes;
void mobius_sieve(){
  mu[1]=1; h[1]=1;
  for(int i=2;i<MX;i++){
    if(!spf[i]){
        spf[i]=i;
        mu[i]=-1;
        phi[i]=i-1;
        h[i]=(1-i+MOD);
        primes.PB(i);
    }
    for(int p:primes){
        if(1LL*i*p>=MX) break;
        spf[i*p]=p;
        if(!(i%p)){
          h[i*p]=h[i];
          phi[i*p]=phi[i]*p;
          mu[i*p]=0;
          break;
        }else {
          mu[i*p]=-mu[i];
          phi[i*p]=phi[i]*(p-1);
          h[i*p]=(h[i]*h[p])%MOD;
        }
    }
  }
}
```

Mobius Inversion Formulas

Description:

1. count(n,k): Pairs with $\gcd(i,j) = k$. Uses $\sum_{d=1}^{\lfloor n/k \rfloor} \mu(d) \lfloor \frac{n}{kd} \rfloor^2$.
2. count(n): Sum of $\gcd(i,j)$ for $1 \le i,j \le n$.
3. solve_lcm: Count subsequences with LCM $= k$.
4. primitive: Count primitive strings.

```
// count gcd(i,j)==1 hard
// but count of gcd(i,j)%k==0 is easy cause i
    %k==0 and j%k==0
// that is N/k this much value can be divide
    by k and pairs are (N/k)*(N/k)
ll count(int n, int k)
{ // count gcd(i,j)==k i,j<=n
  n /= k;
  if (!n)
    return 0;
  ll ans = 0;
  for (int i = 1; i <= n; i++)
  { // this will find in O(n)
    ll g_i = (n / i) * (n / i);
    ans += 1LL * mu[i] * g_i;
  }
  return ans;
}
ll count_faster(int n, int k)
{ // this will find in O(sqrt n)
  n /= k;
  if (!n)
    return 0;
  ll ans = 0;
  for (int l = 1; l <= n;)
  {
    int val = n / l;
    int r = n / val;
    ll g_val = 1LL * val * val;
    ll mu_sum = mu_pre[r] - mu_pre[l - 1];
    ans += mu_sum * g_val;
    l = r + 1;
  }
  return ans;
}
ll count(ll n)
{
  ll ans = 0;
  for (int i = 1; i <= n;)
  {
    ll val = n / i;
    if (!val)
      break;
    ll r = n / val;
    ll g_val = (val * (val - 1)) / 2;
    ans += g_val * (pre_phi[r] - pre_phi[i -
        1]);
    i = r + 1;
  }
  return ans;
}
void solve_lcm()
{ // ans for subsequence LCM=k;
  mobius_sieve();
  pow2[0] = 1;
  for (int i = 1; i < mx; i++)
    pow2[i] = pow2[i - 1] * 2;
  int n;
  cin >> n;
  map<int, int> freq;
  for (int i = 1; i <= n; i++)
  {
    int x;
    cin >> x;
    freq[x]++;
  }
  // now calculating the easy g[k] that is c[
      k]= count of numbers in A that
      divides k
  vector<int> c(mx, 0);
  for (auto const &[val, count] : freq)
  {
    for (int k = val; k < mx; k += val)
      c[k] += count;
  }
  vector<mi> g(mx);
  for (int k = 1; k < mx; k++)
  {
    g[k] = pow2[c[k]] - 1;
  }
  // f[n] = sum( g[d] * mu[n/d] )
  vector<mi> f(mx, 0);
  for (int d = 1; d < mx; d++)
  {
    // if(!g[d]) continue;
    for (int n = d; n < mx; n += d)
      f[n] += g[d] * mu[n / d];
  }
  // f[k] is the ans for subsequence LCM=k;
  int k;
  cin >> k;
  cout << f[k] << endl;
}
/*
*****Problem Statement: "Given N, and an
    alphabet of K letters,
find the number of primitive strings of
    length n for all n from 1 to N."
(A string is primitive if it's not a
    repetition of a smaller block,
e.g., "abcab" is primitive, but "ababab" is
    not).
*/
void solve_primitive_strings()
{
  int n = 100000, k = 26;
  mobius_sieve();
  vector<mi> g(n + 1);
  g[0] = 1;
  for (int i = 1; i <= n; i++)
    g[i] = g[i - 1] * k;
  vector<mi> f(mx, 0);
  for (int d = 1; d < mx; d++)
  {
    // if(!g[d]) continue;
    for (int n = d; n < mx; n += d)
      f[n] += g[d] * mu[n / d];
  }
  // cout << "Primitive strings of length 4 (
      K=26): " << f[4] << endl;
}
```

Mobius LCM Array

Description: Computes sum of LCM of all pairs in an array. Uses precomputed $h[i]$ from sieve.

```
  int n; cin>>n;
  int mx=0;
  vector<int> v(n+1);
  mi sum=0;
  for(int i=1;i<=n;i++) {
    cin>>v[i];
    mx=max(mx,v[i]);
    sum+=v[i];
  }
  vector<int>fre(mx+1,0);
  for(int i=1;i<=n;i++) fre[v[i]]++;
  vector<ll>mp(mx+1,0);
```

```cpp
  for(int i=1;i<=mx;i++) {
    for(int j=i;j<=mx;j+=i) {
      ll k=j/i;
      mp[i]+=1LL*k*fre[j];
    }
  }
  mi ans=0;
  for(int i=1;i<=mx;i++) {
    mi term=mi(i)*mi(h[i])*mi(mp[i]);
    ans+=term;
  }
  cout<<ans<<endl; // all pair lcm sum
  cout<<mi(ans-sum)<<endl; // exclude i=j
  mi inv=mi((MOD+1)/2);
  cout<<mi(mi(ans-sum)*inv)<<endl; // all pair lcm i<j
```

Fast Prime Count

Description: Counts $\pi(n)$ (number of primes $\leq n$) in sub-linear time.

Time: $\mathcal{O}(N^{2/3})$.

```cpp
const int N=3e5+9;
namespace pcf {
    #define MAXN 20000010
    #define MAX_PRIMES 2000010
    #define PHI_N 100000
    #define PHI_K 100
    int len=0; // number of prime gen by sieve
    int primes[MAX_PRIMES];
    int pref[MAXN]; // number of primes <=i
    int dp[PHI_N][PHI_K];
    bitset<MAXN> f;
    void sieve(int n) {
        f[1]=true;
        for(int i=4;i<=n;i+=2) f[i]=true;
        for(int i=3;i*i<=n;i+=2) {
            if(!f[i]) {
                for(int j=i*i;j<=n;j+=i<<1) f[j]=true;
            }
        }
        for(int i=1;i<=n;i++) {
            if(!f[i]) primes[len++]=i;
            pref[i]=len;
        }
    }
    void init() {
        sieve(MAXN-1);
        for(int n=0;n<PHI_N;n++) dp[n][0]=n;
        for(int k=1;k<PHI_K;k++) {
            for(int n=0;n<PHI_N;n++) {
                dp[n][k]=dp[n][k-1]-dp[n/primes[k-1]][k-1];
            }
        }
    }
    ll bro(ll n,int k) { // number of int <=n not div by first k primes
        if(n<PHI_N && k<PHI_K) return dp[n][k];
        if(k==1) return ((++n)>>1);
        if(primes[k-1]>=n) return 1;
        return bro(n,k-1)-bro(n/primes[k-1],k-1);
    }
    ll lehmer(ll n) { // runs under 0.2s for n=1e12
        if(n<MAXN) return pref[n];
        ll w,res=0;
        int b=sqrt(n),c=lehmer(cbrt(n)),a=lehmer(sqrt(b));b=lehmer(b);
        res=bro(n,a)+((1LL*(b+a-2)*(b-a+1))>>1);
        for(int i=a;i<b;i++) {
            w=n/primes[i];
            int lim=lehmer(sqrt(w)); res-=lehmer(w);
            if(i<=c) {
                for(int j=i;j<lim;j++) {
                    res+=j;
                    res-=lehmer(w/primes[j]);
                }
            }
        }
        return res;
    }
}
int32_t main() {
    pcf::init();
    ll n; cin>>n;
    cout<<pcf::lehmer(n)<<endl;
}
```

Modular Combinatorics

Description: Solves Combinations when standard Fermat's Little Theorem fails.

1. lucas: Use when n, r are huge (10^{18}) but p is small (10^5).

2. nCr_pk: Use when modulus is a prime power (e.g., 27, 25) so inverses don't normally exist (removes factor p).

Time: Lucas $O(p + \log_p n)$. Prime Power $O(p^k \log n)$.

```cpp
ll power(ll base,ll exp,ll mod) {
  ll res=1;
  base%=mod;
  while(exp>0) {
    if(exp%2==1) res=(res*base)%mod;
    base=(base*base)%mod;
    exp/=2;
  }
  return res;
}
// Use this for simple primes like 3,5,7..
ll nCr(ll n,ll r,ll p) {
  if(r<0 || r>n) return 0;
  if(!r || r==n) return 1;
  if(r>n/2) r=n-r;
  ll num=1;
  ll den=1;
  for(ll i=1;i<=r;i++) {
    num=(num*(n-i+1))%p;
    den=(den*i)%p;
  }
  return (num*power(den,p-2,p))%p;
}
// Use this to calculate nCr % p when n & r is huge p is small
ll lucas(ll n,ll r,ll p) {
  if(!r) return 1;
  return (lucas(n/p, r/p, p) * nCr(n%p, r%p, p)) %p;
}
//---> billow part use to calcuate nCr if mod is p^k
ll extended_euclid(ll a,ll b,ll &x,ll &y) { // ax+by=gcd(a,b)
  if(!b) {
    x=1,y=0;
    return a;
  }
  ll x1,y1;
  ll d=extended_euclid(b,a%b,x1,y1);
  x=y1;
  y=x1-y1*(a/b);

  return d;
}
ll inverse_pk(ll n,ll mod) {
  ll x,y;
  extended_euclid(n,mod,x,y);
  return (x%mod+mod)%mod;
}
ll fact_no_p(ll n,ll p,ll pk) {
  if(!n) return 1;
  ll ans=1;
  for(ll i=1;i<=pk;i++) {
    if(i%p) ans=(ans*i)%pk;
  }
  ans=power(ans,n/pk,pk);
  for(ll i=1;i<=n%pk;i++) {
    if(i%p) ans=(ans*i)%pk;
  }
  return (ans*fact_no_p(n/p,p,pk))%pk;
}
ll count_p(ll n,ll p) {
  ll ans=0;
  while(n) {
    ans+=n/p;
    n/=p;
  }
  return ans;
}
// nCr % p^k
// Use this for cases like 27 (3^3), 25 (5^2)
// pk=p^k
ll nCr_pk(ll n,ll r,ll p,ll pk) {
  if(r<0 || r>n) return 0;
  ll num=fact_no_p(n,p,pk);
  ll den1=fact_no_p(r,p,pk);
  ll den2=fact_no_p(n-r,p,pk);
  ll ans=(num*inverse_pk(den1,pk))%pk;
  ans=(ans*inverse_pk(den2,pk))%pk;
  ll pow_p=count_p(n,p)-count_p(r,p)-count_p(n-r,p);
  ans=(ans*power(p,pow_p,pk))%pk;

  return ans;
}
```

Kth FIB K$\leq 10^{18}$

```cpp
/*
 * Note : If MOD is constant then use (const int mod=Given mod)
 * and remove mod from function variable declare it will make it
 * 5-10X faster if no need __int128 then remove it from mul more fast
 * For prefix sum f(n+2)-(a+b)
 * sum of f(0)^2+f(1)^2+..+f(n)^2 = f(n)*f(n+1)
 * gcd(f(n),f(m)) = f(gcd(n,m))
 * odd index sum = f(2n)
 * evne index sum f(2n+1)-1
 * THEOREM: Every positive integer N can be uniquely represented as the sum
   of non-consecutive Fibonacci numbers. (i.e., If you use F[i], you cannot use F[i-1] or F[i+1]).

 2. SEQUENCE: Uses Fib starting 1, 2, 3, 5, 8... (Index: F[0]=1, F[1]=2...)
 3. EXAMPLE: N = 100
    - Largest Fib <= 100 is 89. (Rem = 11)
    - Largest Fib <= 11 is 8.   (Rem = 3)
    - Largest Fib <= 3 is 3.    (Rem = 0)
    -> 100 = 89 + 8 + 3
*/
const int mod=1e8+7;
inline ll mul(ll a,ll b,ll mod) {
  return (__int128)a*b%mod;
}
// Works for any modulo m
pair<ll,ll> FIB(ll n,ll mod) {
  if(!n) return {0,1};
  ll a=0,b=1;
  for(int i=63-__builtin_clzll(n);i>=0;i--) {
    ll c=mul(a,(2*b%mod-a+mod)%mod,mod);  // F(2k)
    ll d=(mul(a,a,mod)+mul(b,b,mod))%mod; // F(2k+1)
    if((n>>i)&1) {
      a=d;     // F(2k+1)
      b=(c+d)%mod; // F(2k+2)
    }else {
      a=c;     // F(2k)
      b=d;     // F(2k+1)
    }
  }
  return {a,b};
}
ll kth(ll a,ll b,ll n,ll mod) {
  if(mod==1) return 0;
  if(!n) return a%mod;
  if(n==1) return b%mod;
  pair<ll,ll> fibs=FIB(n-1,mod); //
  return (mul(a,fibs.F,mod) + mul(b,fibs.S,mod))%mod;
}

void GLITCH_() {
  ll n; cin>>n;
  cout<<kth(0,1,n,mod)<<endl;
}
```

Kth FIB n is large

```
inline ll mul(ll a,ll b,ll mod) {
  return (__int128)a*b%mod;
}
struct Mat {
  ll m[2][2];
  Mat() {m[0][0]=m[0][1]=m[1][0]=m[1][1]=0;}
};
Mat matMul(Mat A,Mat B,ll mod) {
  Mat C;
  for(int i=0;i<2;i++) {
    for(int j=0;j<2;j++) {
      for(int k=0;k<2;k++) {
        C.m[i][j]=(C.m[i][j]+mul(A.m[i][k],B.m[k][j],mod))%mod;
      }
    }
  }
  return C;
}
Mat matPow(Mat A,ll p,ll mod) {
  Mat res; res.m[0][0]=res.m[1][1]=1;
  while(p) {
    if(p&1) res=matMul(res,A,mod);
    A=matMul(A,A,mod);
    p>>=1;
  }
  return res;
}
ll kth_string(ll a,ll b,string n,ll mod) {
  if(mod==1) return 0;
  Mat T;
  T.m[0][0]=1; T.m[0][1]=1;
  T.m[1][0]=1; T.m[1][1]=0;
  Mat res;
  res.m[0][0]=1; res.m[1][1]=1;

  for(char c:n) {
    int digit=c-'0';
    res=matPow(res,10,mod);
    res=matMul(res,matPow(T,digit,mod),mod);
  }
  return (mul(a,res.m[1][1],mod)+mul(b,res.m[1][0],mod))%mod;
}
ll kth(ll a,ll b,ll n,ll mod) {
    if(mod==1) return 0;
    if(n==0) return a%mod;
    if(n==1) return b%mod;
    Mat T;
    T.m[0][0]=1; T.m[0][1]=1;
    T.m[1][0]=1; T.m[1][1]=0;

    // Use binary exponentiation directly
    Mat res=matPow(T,n,mod);

    return (mul(a,res.m[1][1],mod) + mul(b, res.m[1][0],mod))%mod;
}
// this can cal nth fib for n is large or <=1e18
void GLITCH_() {
  string n; cin>>n;
  cout<<kth_string(0,1,n,1e8+7)<<endl;
}
```

Extended EGCD

Description: Solves $ax + by = \gcd(a, b)$. Essential for finding Modular Inverse when M is **not prime** (unlike Fermat's Little Theorem) and solving Linear Diophantine Equations.
Time: $O(\log(\min(a, b)))$.

```
ll extended_euclid(ll a,ll b,ll &x,ll &y) { // ax+by=gcd(a,b)
  if(!b) {
    x=1,y=0;
    return a;
  }
  ll x1,y1;
  ll d=extended_euclid(b,a%b,x1,y1);
  x=y1;
  y=x1-y1*(a/b);

  return d;
}
ll inverse(ll a,ll m) {
  ll x,y;
  ll g=extended_euclid(a,m,x,y);
  if(g!=1) return -1;
  return (x%m+m)%m;
}
int main() {
  ll a,b; cin>>a>>b;
  ll x,y, gc=extended_euclid(a,b,x,y);
}
```

CRT

Description: Solves the system of congruences $x \equiv a_i \pmod{m_i}$. Works even if moduli are **not coprime**. Returns $\{x, L\}$ where x is the unique solution modulo $L = \operatorname{lcm}(m_i)$. Returns $\{-1, -1\}$ if no solution exists.
Time: $O(N \log(\operatorname{lcm}(M)))$.

```
ll extended_euclid(ll a,ll b,ll &x,ll &y) { // ax+by=gcd(a,b)
  if(!b) {
    x=1,y=0; return a;
  }
  ll x1,y1;
  ll d=extended_euclid(b,a%b,x1,y1);
  x=y1;
  y=x1-y1*(a/b);
  return d;
}
/** Works for non-coprime moduli.
 Returns {-1,-1} if solution does not exist or input is invalid.
 Otherwise, returns {x,L}, where x is the solution unique to mod L
*/
pair<ll,ll> CRT(vector<ll>A, vector<ll>M) {
  if(A.size()!=M.size()) return {-1,-1};
  int n=A.size();
  ll a1=A[0];
  ll m1=M[0];
  for(int i=1;i<n;i++) {
    ll a2=A[i];
    ll m2=M[i];
    ll g=__gcd(m1,m2);
    if(a1%g != a2%g) return {-1,-1};
    // Marge two equation
    ll p,q, d=extended_euclid(m1/g,m2/g,p,q);
    ll mod=m1/g*m2; // LCM of m1,m2
    ll x = ((__int128)a1 * (m2 / g) * q + (__int128)a2 * (m1 / g) * p) % mod ;

    a1=(x+mod)%mod;
    m1=mod;
  }
  return {a1,m1};
}
int32_t main() {
  vector<ll> A,M;
  pair<ll,ll> ans=CRT(A,M);
}
```

Catalan Number

Description: $C_n = \frac{1}{n+1}\binom{2n}{n}$. Counts valid parenthesis sequences, binary trees, polygon triangulations, etc.
Time: $\mathcal{O}(N)$.

```
ll dp[M], fac[2 * M];
void fact() {
  fac[0] = 1;
  for (int i = 1; i < 2 * M; i++)
    fac[i] = (fac[i - 1] * i) % mod;
}
void cal() {                          /// O(n*logn)
  dp[0] = dp[1] = 1; /// x = (2*x)!/((x+1)!*x!)
  for (int i = 2; i < M; i++)
    dp[i] =
    (fac[2*i]*bigmod((fac[i+1]*fac[i])%mod, mod-2,mod))%mod;
}
```

Custom Bitset (Dynamic)

```
// Compact, fast bitset wrapper using uint64_t blocks.
// - b : number of bits the bitset represents (logical length).
// - n : number of uint64_t words used = ceil(b / 64).
// - bits : underlying storage; bits[0] stores bits [0..63], bits[1] -> [64..127], etc.
//
// Notes:
// - Indexing and public methods use 0-based bit indices in range [0, b).
// - _clean() masks off unused high bits in the last word so count()/find_first() behave correctly.
// - left_shift/right_shift implement block+intra-block shifts using OR to accumulate results
//   (your implementation performs |= shifts; if you want pure shift (assignment) semantics,
//   you would need to zero the target before ORing).
struct Cool_Bitset {
  vector<uint64_t> bits;   // storage
  int64_t b, n;            // b = number of bits, n = number of 64-bit words
  // ctor: optional initial bit length
  Cool_Bitset(int64_t _b = 0) {
      init(_b);
  }
  // initialize to hold _b bits (all cleared)
  void init(int64_t _b) {
    b = _b;
    n = (b + 63) / 64;            // number of 64-bit words required
    bits.assign(n, 0);            // zero-initialize
  }
  // completely free storage
  void clear() {
    b = n = 0;
    bits.clear();
  }
  // reset contents to zero but keep size
  void reset() {
    bits.assign(n, 0);
  }
  // mask out unused high bits in the last word (if b is not a multiple of 64).
  // This ensures operations like count() and find_first() don't see garbage bits past 'b'.
  void _clean() {
    if (b != 64 * n) {
      // compute number of valid bits in last word and mask others off
      bits.back() &= (1ULL << (b - 64 * (n - 1))) - 1;
    }
  }
  // read bit at index (0-based). Returns 0/1.
  bool get(int64_t index) const {
    // no bounds check here for speed; caller should ensure 0 <= index < b
    return (bits[index / 64] >> (index % 64)) & 1ULL;
  }
  // write bit at index to 'value' (true => 1, false => 0)
  void set(int64_t index, bool value) {
    assert(0 <= index && index < b);          // debug-only check
    int64_t word = index / 64;
    int shift = index % 64;
    // clear the target bit then set accordingly
    bits[word] &= ~(1ULL << shift);
    bits[word] |= (uint64_t(value) << shift);
  }
  // LEFT shift by 'shift' bits (logical shift). Implementation uses |= so it accumulates bits.
  // Complexity: O(n)
  void left_shift(int64_t shift) {
    int64_t div = shift / 64;    // whole-word shift
```

```
    int64_t mod = shift % 64;    // intra-word shift
    if (mod == 0) {
      // shift by whole words: move words upward
      for (int64_t i = n - 1; i >= div; i--)
        bits[i] |= bits[i - div];
      // note: words [0..div-1] are unchanged (ORed with 0)
      return;
    }
    // shift with both whole-word and bit offset
    for (int64_t i = n - 1; i >= div + 1; i--) {
      // combine higher-part and lower-part of source words
      bits[i] |= (bits[i - (div + 1)] >> (64 - mod)) | (bits[i - div] << mod);
    }
    // handle the boundary word (if any)
    if (div < n)
      bits[div] |= bits[0] << mod;
    _clean(); // ensure we didn't set bits past 'b'
  }
  // RIGHT shift by 'shift' bits (logical). Implementation uses |= so it accumulates bits.
  // Complexity: O(n)
  void right_shift(int64_t shift) {
    int64_t div = shift / 64;
    int64_t mod = shift % 64;
    if (mod == 0) {
      for (int64_t i = div; i < n; i++)
        bits[i - div] |= bits[i];
      return;
    }
    for (int64_t i = 0; i < n - (div + 1); i++)
      bits[i] |= (bits[i + (div + 1)] << (64 - mod)) | (bits[i + div] >> mod);
    if (div < n)
      bits[n - div - 1] |= bits[n - 1] >> mod;
    _clean();
  }
  // population count (number of set bits). Uses builtin popcountll on each word.
  int64_t count() const {
    int64_t res = 0;
    for (int64_t i = 0; i < n; i++)
      res += __builtin_popcountll(bits[i]);
    return res;
  }
  // find index of first set bit (lowest index). Returns -1 if none.
  // Complexity: O(n) in worst case, but fast because it scans word-by-word and uses ctz.
  int64_t find_first() const {
      for (int64_t i = 0; i < n; i++)
        if (bits[i] != 0)
          return 64 * i + __builtin_ctzll(bits[i]); // ctz: count trailing zeros
      return -1;
  }
  // find next set bit strictly after x (i.e., search from x+1).
  // Safety: original loop could read past 'b', so we added a guard that stops at 'b'.
  // Returns -1 if none.
  int64_t find_next(int64_t x) const {
    // first scan in the same word (from x+1 up to end of that word)
    int64_t start = x + 1;
    if (start < b) {
      int64_t end_same_word = min<int64_t>( (x / 64) * 64 + 64, b ); // exclusive bound
      for (int64_t i = start; i < end_same_word; ++i) {
          if (get(i)) return i;
      }
    }
    // then scan entire following words
    for (int64_t i = x / 64 + 1; i < n; i++)
      if (bits[i] != 0)
        return 64 * i + __builtin_ctzll(bits[i]);

    return -1;
  }
  // in-place AND with another bitset (must be same size)
  Cool_Bitset& operator&=(const Cool_Bitset & other) {
    assert(b == other.b);
    for (int64_t i = 0; i < n; i++)
        bits[i] &= other.bits[i];
    return *this;
  }
  // return new bitset = this & other
  Cool_Bitset operator&(const Cool_Bitset & other) const {
    assert(b == other.b);
    Cool_Bitset res(b);
    for (int64_t i = 0; i < n; i++) res.bits[i] = bits[i] & other.bits[i];
    return res;
  }
};
```

XOR Basis

```
const int MX=301;
struct bigxorBasis {
  bitset<MX> basis[MX];
  bool has_basis[MX];
  int sz;
  bigxorBasis() {
    for(int i=0;i<MX;i++) has_basis[i]=false;
    sz=0;
  }
  void insert(bitset<MX> mask) {
    for(int i=MX-1;i>=0;i--) {
      if(!mask.test(i)) continue;
      if(!has_basis[i]) {
        basis[i]=mask;
        has_basis[i]=true;
        sz++;
        return;
      }
      mask^=basis[i];
    }
  }
  ll zeros(ll n) {
    return (n-sz);
  }
};

const int LOG_K=64;
struct xorBasis {
  ll basis[LOG_K];
  int sz;
  bool dirty;
  xorBasis() {
    fill(basis,basis+LOG_K,0);
    sz=0;
  }
  bool insert(ll x) {
    for(int i=LOG_K-1;i>=0;i--) {
      if(!(x&(1LL<<i))) continue;
      if(!basis[i]) {
        basis[i]=x;
        sz++;
        dirty=true;
        return true;
      }
      x^=basis[i];
    }
    return false; // it means x got 0 and it is makeable by others
  }
  void RREF() { // Reduced row echekon form
    for(int i=LOG_K-1;i>=0;i--) {
      if(basis[i]) {
        for(int j=i-1;j>=0;j--) {
          if(basis[j] && basis[i]&(1LL<<j))
            basis[i]^=basis[j];
        }
      }
    }
  }
  ll unique(ll n) {
    return (1LL<<sz);
  }
  ll how_many_can_make(ll n) {
    return (1LL<<(n-sz));
    //return n-sz;
  }
  ll can_make_x(ll x) {
    for(int i=LOG_K-1;i>=0;i--) {
      if(x&(1LL<<i)) x^=basis[i];
    }
    if(!x) return 1;
    else return 0;
  }
  ll kth(ll k) {
    if(dirty) RREF();
    vector<ll> v;
    for(int i=0;i<LOG_K;i++) if(basis[i]) v.PB(basis[i]);
    if((1LL<<sz)<k) return -1;
    k--;
    ll ans=0;
    for(int i=0;i<LOG_K;i++) {
      if(k&(1LL<<i)) ans^=v[i];
    }
    return ans;
  }
  ll max() {
    RREF();
    ll ans=0;
    for(int i=0;i<LOG_K;i++) ans^=basis[i];
    return ans;
  }
};
```

XOR Basis range query

```
const int LOG_K=60;
struct xorBasis {
  ll basis[LOG_K];
  ll pos[LOG_K];
  int sz;
  bool dirty;
  xorBasis() {
    fill(basis,basis+LOG_K,0);
    fill(pos,pos+LOG_K,0);
    sz=0;
  }
  bool insert(ll x,ll ind) {
    for(int i=LOG_K-1;i>=0;i--) {
      if(!(x&(1LL<<i))) continue;
      if(!basis[i]) {
        basis[i]=x;
        sz++;
        pos[i]=ind;
        //dirty=true;
        return true;
      }
      if(pos[i]<ind) {
        swap(basis[i],x);
        swap(pos[i],ind);
      }
      x^=basis[i];
    }
    return false; // it means x got 0 and it is makeable by others
  }
  //int MAX(int L) {
    //int ans=0;
    //for (int i = LOG_K - 1; i >= 0; i--) {
      //if(pos[i]>=L) {
        //ans=max(ans,basis[i]^ans);
      //}
    //}
    //return ans;
  //}
  ll can_make_x(ll x,ll L) {
    for(int i=LOG_K-1;i>=0;i--) {
      if(pos[i]>=L)
        if(x&(1LL<<i)) x^=basis[i];
      }
```

```
    return (x==0);
  }
};
vector<xorBasis> prefix_basis;
void GLITCH_() {
  int n; cin>>n;
  prefix_basis.resize(n+1);
  for(int i=1;i<=n;i++) {
    ll val; cin>>val;
    prefix_basis[i]=prefix_basis[i-1];
    prefix_basis[i].insert(val,i);
  }
  int q; cin>>q;
  for(int i=1;i<=q;i++) {
    int l,r; cin>>l>>r;
    ll x; cin>>x;
    if(prefix_basis[r].can_make_x(x,l)) ha();
    else na();
    //cout<<prefix_basis[r].can_make_x(x,l)<<endl;
  }
}
```

XOR Basis subset print

```
const int LOG_K=60;

struct Filter {
  ll basis[LOG_K];
  Filter() {
    fill(basis,basis+LOG_K,0);
  }
  bool insert(ll val) {
    for(int i=LOG_K-1;i>=0;i--) {
      if(!(val & (1LL<<i))) continue;
      if(!basis[i]) {
        basis[i]=val;
        return true;
      }
      val^=basis[i];
    }
    return false;
  }
};

struct construct {
  ll basis[LOG_K];
  ll mask[LOG_K];
  construct() {
    fill(basis,basis+LOG_K,0);
    fill(mask,mask+LOG_K,0);
  }
  void insert(ll val,int pivort) {
    ll current_mask=(1LL<<pivort);
    for(int i=LOG_K-1;i>=0;i--) {
      if(!(val & (1LL<<i))) continue;
      if(!basis[i]) {
        basis[i]=val;
        mask[i]=current_mask;
        return;
      }
      val^=basis[i];
      current_mask^=mask[i];
    }
  }
  ll get_mask(ll terget) {
    ll ans_mask=0;
    for(int i=LOG_K-1;i>=0;i--) {
      if((terget>>i) & 1) {
        if(!basis[i]) return -1; //
        terget^=basis[i];
        ans_mask^=mask[i];
      }
    }
    return ans_mask;
  }
};
void GLITCH_() {
  Filter filter;
  construct solver;
  int n; cin>>n;
  vector<int> ind;
  int cnt=0;
  for(int i=1;i<=n;i++) {
    ll val;
    cin>>val;
    if(filter.insert(val)) {
      solver.insert(val,cnt);
      cnt++;
      ind.PB(i);
    }
  }
  int q; cin>>q;
  while(q--) {
    ll x; cin>>x;
    ll used_mask=solver.get_mask(x);
    ll ans[n+1] {};
    for(int i=0;i<ind.size();i++) {
      if((used_mask>>i) & 1) {
        ans[ind[i]]=1;
      }
    }
    for(int i=1;i<=n;i++) cout<<ans[i]; cout<<endl;
  }
}
```

Matrix Multiplication & Matrix Exponentiation

```
const int MOD = 1e9 + 7; const int SZ = 2;
struct Matrix {
    long long mat[SZ][SZ];
    Matrix() { memset(mat, 0, sizeof(mat)); }
    static Matrix identity() {
        Matrix res;
        for (int i = 0; i < SZ; i++)
            res.mat[i][i] = 1;
        return res;
    }
    Matrix operator*(const Matrix& other) const { // Matrix Mul: A * B
        Matrix res;
        for (int i = 0; i < SZ; i++) {
            for (int k = 0; k < SZ; k++) {
                if (mat[i][k] == 0) continue;
                for (int j = 0; j < SZ; j++) {
                    res.mat[i][j] = (res.mat[i][j] + mat[i][k] * other.mat[k][j]) % MOD;
                }
            }
        }
        return res;
    }
};
Matrix power(Matrix a, long long p) {
    Matrix res = Matrix::identity();
    while (p > 0) {
        if (p & 1) res = res * a;
        a = a * a; p >>= 1;
    }
    return res;
}
int main() {
    long long n; cin >> n;
    if (n == 0) {
        cout << 0 << endl; return 0;
    }
    Matrix T;
    T.mat[0][0] = 1; T.mat[0][1] = 1;
    T.mat[1][0] = 1; T.mat[1][1] = 0;
    T = power(T, n - 1);
// The answer is T[0][0] * F(1) + T[0][1]*F(0)
//Since F(1)=1 and F(0)=0, answer is just T[0][0]
    cout << T.mat[0][0] << endl;
}
```

Geo Template

```
const int N = 3e5 + 9;

const double inf = 1e100;
const double eps = 1e-9;
const double PI = acos((double)-1.0);
int sign(double x) { return (x > eps) - (x < -eps); }
struct PT {
    double x, y;
    PT() { x = 0, y = 0; }
    PT(double x, double y) : x(x), y(y) {}
    PT(const PT &p) : x(p.x), y(p.y)    {}
    PT operator + (const PT &a) const { return PT(x + a.x, y + a.y); }
    PT operator - (const PT &a) const { return PT(x - a.x, y - a.y); }
    PT operator * (const double a) const { return PT(x * a, y * a); }
    friend PT operator * (const double &a, const PT &b) { return PT(a * b.x, a * b.y); }
    PT operator / (const double a) const { return PT(x / a, y / a); }
    bool operator == (PT a) const { return sign(a.x - x) == 0 && sign(a.y - y) == 0; }
    bool operator != (PT a) const { return !(*this == a); }
    bool operator < (PT a) const { return sign(a.x - x) == 0 ? y < a.y : x < a.x; }
    bool operator > (PT a) const { return sign(a.x - x) == 0 ? y > a.y : x > a.x; }
    double norm() { return sqrt(x * x + y * y); }
    double norm2() { return x * x + y * y; }
    PT perp() { return PT(-y, x); }
    double arg() { return atan2(y, x); }
    PT truncate(double r) { // returns a vector with norm r and having same direction
        double k = norm();
        if (!sign(k)) return *this;
        r /= k;
        return PT(x * r, y * r);
    }
};
istream &operator >> (istream &in, PT &p) { return in >> p.x >> p.y; }
ostream &operator << (ostream &out, PT &p) { return out << "(" << p.x << "," << p.y << ")"; }
inline double dot(PT a, PT b) { return a.x * b.x + a.y * b.y; }
inline double dist2(PT a, PT b) { return dot(a - b, a - b); }
inline double dist(PT a, PT b) { return sqrt(dot(a - b, a - b)); }
inline double cross(PT a, PT b) { return a.x * b.y - a.y * b.x; }
inline double cross2(PT a, PT b, PT c) { return cross(b - a, c - a); }
inline int orientation(PT a, PT b, PT c) { return sign(cross(b - a, c - a)); }
PT perp(PT a) { return PT(-a.y, a.x); }
PT rotateccw90(PT a) { return PT(-a.y, a.x); }
PT rotatecw90(PT a) { return PT(a.y, -a.x); }
PT rotateccw(PT a, double t) { return PT(a.x * cos(t) - a.y * sin(t), a.x * sin(t) + a.y * cos(t)); }
PT rotatecw(PT a, double t) { return PT(a.x * cos(t) + a.y * sin(t), -a.x * sin(t) + a.y * cos(t)); }
double SQ(double x) { return x * x; }
double rad_to_deg(double r) { return (r * 180.0 / PI); }
double deg_to_rad(double d) { return (d * PI / 180.0); }
double get_angle(PT a, PT b) {
    double costheta = dot(a, b) / a.norm() / b.norm();
    return acos(max((double)-1.0, min((double)1.0, costheta)));
}
bool is_point_in_angle(PT b, PT a, PT c, PT p) { // does point p lie in angle <bac
    assert(orientation(a, b, c) != 0);
    if (orientation(a, c, b) < 0) swap(b, c);
    return orientation(a, c, p) >= 0 && orientation(a, b, p) <= 0;
}
bool half(PT p) {
    return p.y > 0.0 || (p.y == 0.0 && p.x < 0.0);
```

```
}
void polar_sort(vector<PT> &v) { // sort points in counterclockwise
    sort(v.begin(), v.end(), [](PT a,PT b) {
        return make_tuple(half(a), 0.0, a.norm2()) < make_tuple(half(b), cross(a, b), b.norm2());
    });
}
void polar_sort(vector<PT> &v, PT o) { // sort points in counterclockwise with respect to point o
    sort(v.begin(), v.end(), [&](PT a,PT b) {
        return make_tuple(half(a - o), 0.0, (a - o).norm2()) < make_tuple(half(b - o), cross(a - o, b - o), (b - o).norm2());
    });
}
struct line {
    PT a, b; // goes through points a and b
    PT v; double c;  //line form: direction vec [cross] (x, y) = c
    line() {}
    //direction vector v and offset c
  line(PT v, double c) : v(v), c(c) {
        auto p = get_points();
        a = p.first; b = p.second;
  }
  // equation ax + by + c = 0
  line(double _a, double _b, double _c) : v({_b, -_a}), c(-_c) {
    auto p = get_points();
        a = p.first; b = p.second;
  }
  // goes through points p and q
  line(PT p, PT q) : v(q - p), c(cross(v, p)), a(p), b(q) {}
      pair<PT, PT> get_points() { //extract any two points from this line
    PT p, q; double a = -v.y, b = v.x; // ax + by = c
    if (sign(a) == 0) {
        p = PT(0, c / b);
        q = PT(1, c / b);
    }
    else if (sign(b) == 0) {
        p = PT(c / a, 0);
        q = PT(c / a, 1);
    }
    else {
        p = PT(0, c / b);
        q = PT(1, (c - a) / b);
    }
    return {p, q};
      }
    // ax + by + c = 0
    array<double, 3> get_abc() {
        double a = -v.y, b = v.x;
        return {a, b, -c};
    }
    // 1 if on the left, -1 if on the right, 0 if on the line
    int side(PT p) { return sign(cross(v, p) - c); }
    // line that is perpendicular to this and goes through point p
    line perpendicular_through(PT p) { return {p, p + perp(v)}; }
    // translate the line by vector t i.e. shifting it by vector t
    line translate(PT t) { return {v, c + cross(v, t)}; }
    // compare two points by their orthogonal projection on this line
    // a projection point comes before another if it comes first according to vector v
    bool cmp_by_projection(PT p, PT q) { return dot(v, p) < dot(v, q); }
  line shift_left(double d) {
    PT z = v.perp().truncate(d);
    return line(a + z, b + z);
  }
};
// find a point from a through b with distance d
PT point_along_line(PT a, PT b, double d) {
    assert(a != b);
    return a + (((b - a) / (b - a).norm()) * d);
}
// projection point c onto line through a and b assuming a != b
PT project_from_point_to_line(PT a, PT b, PT c) {
    return a + (b - a) * dot(c - a, b - a) / (b - a).norm2();
}
// reflection point c onto line through a and b assuming a != b
PT reflection_from_point_to_line(PT a, PT b, PT c) {
    PT p = project_from_point_to_line(a,b,c);
    return p + p - c;
}
// minimum distance from point c to line through a and b
double dist_from_point_to_line(PT a, PT b, PT c) {
    return fabs(cross(b - a, c - a) / (b - a).norm());
}
// returns true if  point p is on line segment ab
bool is_point_on_seg(PT a, PT b, PT p) {
    if (fabs(cross(p - b, a - b)) < eps) {
        if (p.x < min(a.x, b.x) - eps || p.x > max(a.x, b.x) + eps) return false;
        if (p.y < min(a.y, b.y) - eps || p.y > max(a.y, b.y) + eps) return false;
        return true;
    }
    return false;
}
// minimum distance point from point c to segment ab that lies on segment ab
PT project_from_point_to_seg(PT a, PT b, PT c) {
    double r = dist2(a, b);
    if (sign(r) == 0) return a;
    r = dot(c - a, b - a) / r;
    if (r < 0) return a;
    if (r > 1) return b;
    return a + (b - a) * r;
}
// minimum distance from point c to segment ab
double dist_from_point_to_seg(PT a, PT b, PT c) {
    return dist(c, project_from_point_to_seg(a, b, c));
}
// 0 if not parallel, 1 if parallel, 2 if collinear
int is_parallel(PT a, PT b, PT c, PT d) {
    double k = fabs(cross(b - a, d - c));
    if (k < eps){
        if (fabs(cross(a - b, a - c)) < eps && fabs(cross(c - d, c - a)) < eps) return 2;
        else return 1;
    }
    else return 0;
}
// check if two lines are same
bool are_lines_same(PT a, PT b, PT c, PT d) {
    if (fabs(cross(a - c, c - d)) < eps && fabs(cross(b - c, c - d)) < eps) return true;
    return false;
}
// bisector vector of <abc
PT angle_bisector(PT &a, PT &b, PT &c){
    PT p = a - b, q = c - b;
    return p + q * sqrt(dot(p, p) / dot(q, q));
}
// 1 if point is ccw to the line, 2 if point is cw to the line, 3 if point is on the line
int point_line_relation(PT a, PT b, PT p) {
    int c = sign(cross(p - a, b - a));
    if (c < 0) return 1;
    if (c > 0) return 2;
    return 3;
}
// intersection point between ab and cd assuming unique intersection exists
bool line_line_intersection(PT a, PT b, PT c, PT d, PT &ans) {
    double a1 = a.y - b.y, b1 = b.x - a.x, c1 = cross(a, b);
    double a2 = c.y - d.y, b2 = d.x - c.x, c2 = cross(c, d);
    double det = a1 * b2 - a2 * b1;
    if (det == 0) return 0;
    ans = PT((b1 * c2 - b2 * c1) / det, (c1 * a2 - a1 * c2) / det);
    return 1;
}
// intersection point between segment ab and segment cd assuming unique intersection exists
bool seg_seg_intersection(PT a, PT b, PT c, PT d, PT &ans) {
    double oa = cross2(c, d, a), ob = cross2(c, d, b);
    double oc = cross2(a, b, c), od = cross2(a, b, d);
    if (oa * ob < 0 && oc * od < 0){
        ans = (a * ob - b * oa) / (ob - oa);
        return 1;
    }
    else return 0;
}
// intersection point between segment ab and segment cd assuming unique intersection may not exists
// se.size()==0 means no intersection
// se.size()==1 means one intersection
// se.size()==2 means range intersection
set<PT> seg_seg_intersection_inside(PT a,  PT b,  PT c,  PT d) {
    PT ans;
    if (seg_seg_intersection(a, b, c, d, ans)) return {ans};
    set<PT> se;
    if (is_point_on_seg(c, d, a)) se.insert(a);
    if (is_point_on_seg(c, d, b)) se.insert(b);
    if (is_point_on_seg(a, b, c)) se.insert(c);
    if (is_point_on_seg(a, b, d)) se.insert(d);
    return se;
}
// intersection  between segment ab and line cd
// 0 if do not intersect, 1 if proper intersect, 2 if segment intersect
int seg_line_relation(PT a, PT b, PT c, PT d) {
    double p = cross2(c, d, a);
    double q = cross2(c, d, b);
    if (sign(p) == 0 && sign(q) == 0) return 2;
    else if (p * q < 0) return 1;
    else return 0;
}
// intersection between segament ab and line cd assuming unique intersection exists
bool seg_line_intersection(PT a, PT b, PT c, PT d, PT &ans) {
    bool k = seg_line_relation(a, b, c, d);
    assert(k != 2);
    if (k) line_line_intersection(a, b, c, d, ans);
    return k;
}
// minimum distance from segment ab to segment cd
double dist_from_seg_to_seg(PT a, PT b, PT c, PT d) {
```

```cpp
    PT dummy;
    if (seg_seg_intersection(a, b, c, d, dummy)) return 0.0;
    else return min({dist_from_point_to_seg(a, b, c), dist_from_point_to_seg(a, b, d),
        dist_from_point_to_seg(c, d, a), dist_from_point_to_seg(c, d, b)});
}
// minimum distance from point c to ray (starting point a and direction vector b)
double dist_from_point_to_ray(PT a, PT b, PT c) {
    b = a + b;
    double r = dot(c - a, b - a);
    if (r < 0.0) return dist(c, a);
    return dist_from_point_to_line(a, b, c);
}
// starting point as and direction vector ad
bool ray_ray_intersection(PT as, PT ad, PT bs, PT bd) {
    double dx = bs.x - as.x, dy = bs.y - as.y;
    double det = bd.x * ad.y - bd.y * ad.x;
    if (fabs(det) < eps) return 0;
    double u = (dy * bd.x - dx * bd.y) / det;
    double v = (dy * ad.x - dx * ad.y) / det;
    if (sign(u) >= 0 && sign(v) >= 0) return 1;
    else return 0;
}
double ray_ray_distance(PT as, PT ad, PT bs, PT bd) {
    if (ray_ray_intersection(as, ad, bs, bd)) return 0.0;
    double ans = dist_from_point_to_ray(as, ad, bs);
    ans = min(ans, dist_from_point_to_ray(bs, bd, as));
    return ans;
}
struct circle {
    PT p; double r;
    circle() {}
    circle(PT _p, double _r): p(_p), r(_r) {};
    // center (x, y) and radius r
    circle(double x, double y, double _r): p(PT(x, y)), r(_r) {};
    // circumcircle of a triangle
    // the three points must be unique
    circle(PT a, PT b, PT c) {
        b = (a + b) * 0.5;
        c = (a + c) * 0.5;
        line_line_intersection(b, b + rotatecw90(a - b), c, c + rotatecw90(a - c), p);
        r = dist(a, p);
    }
    // inscribed circle of a triangle
    // pass a bool just to differentiate from circumcircle
    circle(PT a, PT b, PT c, bool t) {
        line u, v;
        double m = atan2(b.y - a.y, b.x - a.x), n = atan2(c.y - a.y, c.x - a.x);
        u.a = a;
        u.b = u.a + (PT(cos((n + m)/2.0), sin((n + m)/2.0)));
        v.a = b;
        m = atan2(a.y - b.y, a.x - b.x), n = atan2(c.y - b.y, c.x - b.x);
        v.b = v.a + (PT(cos((n + m)/2.0), sin((n + m)/2.0)));
        line_line_intersection(u.a, u.b, v.a, v.b, p);
        r = dist_from_point_to_seg(a, b, p);
    }
    bool operator == (circle v) { return p == v.p && sign(r - v.r) == 0; }
    double area() { return PI * r * r; }
    double circumference() { return 2.0 * PI * r; }
};
//0 if outside, 1 if on circumference, 2 if inside circle
int circle_point_relation(PT p, double r, PT b) {
    double d = dist(p, b);
    if (sign(d - r) < 0) return 2;
    if (sign(d - r) == 0) return 1;
    return 0;
}
// 0 if outside, 1 if on circumference, 2 if inside circle
int circle_line_relation(PT p, double r, PT a, PT b) {
    double d = dist_from_point_to_line(a, b, p);
    if (sign(d - r) < 0) return 2;
    if (sign(d - r) == 0) return 1;
    return 0;
}
//compute intersection of line through points a and b with
//circle centered at c with radius r > 0
vector<PT> circle_line_intersection(PT c, double r, PT a, PT b) {
    vector<PT> ret;
    b = b - a; a = a - c;
    double A = dot(b, b), B = dot(a, b);
    double C = dot(a, a) - r * r, D = B * B - A * C;
    if (D < -eps) return ret;
    ret.push_back(c + a + b * (-B + sqrt(D + eps)) / A);
    if (D > eps) ret.push_back(c + a + b * (-B - sqrt(D)) / A);
    return ret;
}
//5 - outside and do not intersect
//4 - intersect outside in one point
//3 - intersect in 2 points
//2 - intersect inside in one point
//1 - inside and do not intersect
int circle_circle_relation(PT a, double r, PT b, double R) {
    double d = dist(a, b);
    if (sign(d - r - R) > 0)  return 5;
    if (sign(d - r - R) == 0) return 4;
    double l = fabs(r - R);
    if (sign(d - r - R) < 0 && sign(d - l) > 0) return 3;
    if (sign(d - l) == 0) return 2;
    if (sign(d - l) < 0) return 1;
    assert(0); return -1;
}
// returns area of intersection between two circles
double circle_circle_area(PT a, double r1, PT b, double r2) {
    double d = (a - b).norm();
    if(r1 + r2 < d + eps) return 0;
    if(r1 + d < r2 + eps) return PI * r1 * r1;
    if(r2 + d < r1 + eps) return PI * r2 * r2;
    double theta_1 = acos((r1 * r1 + d * d - r2 * r2) / (2 * r1 * d)),
      theta_2 = acos((r2 * r2 + d * d - r1 * r1)/(2 * r2 * d));
    return r1 * r1 * (theta_1 - sin(2 * theta_1)/2.) + r2 * r2 * (theta_2 - sin(2 * theta_2)/2.);
}
vector<PT> convex_hull(vector<PT> &p) {
  if (p.size() <= 1) return p;
  vector<PT> v = p;
    sort(v.begin(), v.end());
    vector<PT> up, dn;
    for (auto& p : v) {
        while (up.size() > 1 && orientation(up[up.size() - 2], up.back(), p) >= 0) {
            up.pop_back();
        }
        while (dn.size() > 1 && orientation(dn[dn.size() - 2], dn.back(), p) <= 0) {
            dn.pop_back();
        }
        up.push_back(p);
        dn.push_back(p);
    }
    v = dn;
    if (v.size() > 1) v.pop_back();
    reverse(up.begin(), up.end());
    up.pop_back();
    for (auto& p : up) {
        v.push_back(p);
    }
    if (v.size() == 2 && v[0] == v[1]) v.pop_back();
    return v;
}
 //checks if convex or not
bool is_convex(vector<PT> &p) {
    bool s[3]; s[0] = s[1] = s[2] = 0;
    int n = p.size();
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        int k = (j + 1) % n;
        s[sign(cross(p[j] - p[i], p[k] - p[i])) + 1] = 1;
        if (s[0] && s[2]) return 0;
    }
    return 1;
}
// -1 if strictly inside, 0 if on the polygon, 1 if strictly outside
// it must be strictly convex, otherwise make it strictly convex first
int is_point_in_convex(vector<PT> &p, const PT& x) { // O(log n)
    int n = p.size(); assert(n >= 3);
    int a = orientation(p[0], p[1], x), b = orientation(p[0], p[n - 1], x);
    if (a < 0 || b > 0) return 1;
    int l = 1, r = n - 1;
    while (l + 1 < r) {
        int mid = l + r >> 1;
        if (orientation(p[0], p[mid], x) >= 0) l = mid;
        else r = mid;
    }
    int k = orientation(p[l], p[r], x);
    if (k <= 0) return -k;
    if (l == 1 && a == 0) return 0;
    if (r == n - 1 && b == 0) return 0;
    return -1;
}
```

Closest Pair of Points

Description: Finds the minimum distance between any two points in a set.

- **Algorithm:** Divide & Conquer.
- **Logic:** 1. Sort points by X-coordinate. 2. Divide into left-/right halves. Recurse to find $d = \min(d_L, d_R)$. 3. **Merge Step:** The closest pair might span the dividing line. Gather points within distance d of the middle X-line into a "strip". 4. Sort strip by Y-coordinate. For each point, check neighbors in the strip. (Geometry guarantees we only need to check the next ≈ 7 points).
- **Time:** $\mathcal{O}(N \log N)$ (if we merge-sort by Y during recursion) or $\mathcal{O}(N \log^2 N)$ (if we sort strip explicitly). The code below uses `inplace_merge` for $\mathcal{O}(N \log N)$.

```cpp
// Auxiliary function for recursion
ld closestPairRec(vector<P>& pts, int l, int r, vector<P>& aux) {
    if (r - l <= 3) {
        ld best = numeric_limits<ld>::infinity();
        for (int i = l; i < r; ++i)
            for (int j = i+1; j < r; ++j)
                best = min(best, dist(pts[i], pts[j]));
        // Sort by Y for the merge step
        sort(pts.begin()+l, pts.begin()+r, [](const P& a, const P& b){ return a.y < b.y; });
        return best;
    }
    int m = (l + r) >> 1;
    ld midx = pts[m].x;
    ld d = min(closestPairRec(pts, l, m, aux), closestPairRec(pts, m, r, aux));
```

```
    // Merge both sorted halves by Y-
        ↪ coordinate
    inplace_merge(pts.begin()+l, pts.begin()+
        ↪ m, pts.begin()+r,
                 [](const P& a, const P& b){
                    ↪ return a.y < b.y;
                    ↪ });

    // Create strip: only keep points within
        ↪ 'd' horizontal distance from midx
    int sz = 0;
    for (int i = l; i < r; ++i) {
        if (fabsl(pts[i].x - midx) < d + EPS)
            ↪ aux[sz++] = pts[i];
    }
    // Check points in strip against their
        ↪ neighbors (within vertical
        ↪ distance d)
    for (int i = 0; i < sz; ++i) {
        for (int j = i+1; j < sz && (aux[j].y
            ↪ - aux[i].y) < d + EPS; ++j)
            ↪ {
            d = min(d, dist(aux[i], aux[j]));
        }
    }
    return d;
}
inline ld closestPair(vector<P> pts) {
    sort(pts.begin(), pts.end(), point_cmp);
        ↪ // Sort by X initially
    vector<P> aux(pts.size());
    return closestPairRec(pts, 0, pts.size(),
        ↪ aux);
}
```

DP

Binary Optimization

Description: Solves Bounded Knapsack (limited count of items) by decomposing counts into powers of 2 $(1, 2, 4, \ldots, rem)$. Turns $\mathcal{O}(W \cdot \text{count})$ into $\mathcal{O}(W \cdot \log(\text{count}))$.
Time: $\mathcal{O}(W \cdot \sum \log(\text{count}))$.

```
map<int, int> mp;
for (auto it : vec)
  mp[it]++;
vector<int> dp(n + 1, 1e9);
dp[0] = 0;
for (auto [w, cnt] : mp) {
  int cur = 1;
  while (cnt > 0) {
    int use = min(cnt, cur);
    for (int i = n; i >= w * use; i--) {
      dp[i] = min(dp[i], dp[i - w * use] +
          ↪ use);
    }
    cnt -= use;
    cur *= 2;
  }
}
```

Mathematics

Equations

The extremum of a quadratic is given by $x = -b/2a$.
Cramer's Rule: Given an equation $Ax = b$, the solution to a variable x_i is given by

$$x_i = \frac{\det A_i'}{\det A} \qquad \text{[where } A_i' \text{ is } A \text{ with the } i\text{'th column replaced by } b.\text{]}$$

Example (3x3):

$$2x + 3y - 5z = 1$$
$$x + y - z = 2$$
$$2y + z = 8$$

$D = \begin{vmatrix} 2 & 3 & -5 \\ 1 & 1 & -1 \\ 0 & 2 & 1 \end{vmatrix} = -7$ $D_x = \begin{vmatrix} 1 & 3 & -5 \\ 2 & 1 & -1 \\ 8 & 2 & 1 \end{vmatrix} = -7$ $D_y = \begin{vmatrix} 2 & 1 & -5 \\ 1 & 2 & -1 \\ 0 & 8 & 1 \end{vmatrix} = -21$ $D_z = \begin{vmatrix} 2 & 3 & 1 \\ 1 & 1 & 2 \\ 0 & 2 & 8 \end{vmatrix} = 14$ $x = \frac{D_x}{D} = 1$, $y = \frac{D_y}{D} = 3$, $z = \frac{D_z}{D} = -2$

Vieta's Formulas: Let $P(x) = a_n x^n + \ldots + a_0$, be a polynomial with complex coefficients and degree n, having complex roots $r_n, \ldots, r_1$. Then for any integer $0 \le k \le n$,

$$\sum_{1 \le i_1 < i_2 < \ldots < i_k \le n} r_{i_1} r_{i_2} \ldots r_{i_k} = (-1)^k \frac{a_{n-k}}{a_n}$$

Rational Root Theorem: If $\frac{p}{q}$ is a reduced rational root of a polynomial with **integer coeffs**, then $p \mid a_0$ and $q \mid a_n$.

Number Theory

Sum of Divisors (S.O.D): If $N = a^p \cdot b^q \cdot c^r \ldots$

$$\text{S.O.D} = \frac{a^{p+1} - 1}{a - 1} \cdot \frac{b^{q+1} - 1}{b - 1} \cdot \frac{c^{r+1} - 1}{c - 1} \ldots$$

Number of Divisors (N.O.D): If $N = a^p \cdot b^q \cdot c^r \ldots$

$$\text{N.O.D} = (p + 1)(q + 1)(r + 1) \ldots$$

Product of Divisors (P.O.D): If N has $D = \text{N.O.D}(N)$ divisors:

$$\text{P.O.D}(N) = N^{D/2} = (\sqrt{N})^D$$

Euclidean Algorithm Property:

$$\gcd(a, b) = \gcd(a, a - b) \quad [a > b]$$

Fibonacci GCD:

$$\gcd(F(a), F(b)) = F(\gcd(a, b))$$

Euler's Totient Theorem:

$$a^{\phi(n)} \equiv 1 \pmod n$$

where $\phi(n)$ is Euler's Totient Function.
Modular Exponentiation:

$$a^b \pmod m \equiv a^{b \pmod{\phi(m)}} \pmod m$$

(if a and m are coprime)
Primitive roots modulo n exists iff $n = 1, 2, 4$ or, $n = p^k, 2p^k$ where p is an odd prime. Furthermore, the number of roots are $\phi(\phi(n))$.
To Find Generator g of M, factor $M - 1$ and get the distinct primes p_i. If $g^{(M-1)/p_i} \neq 1(MODM)$ for each p_i then g is a valid root. Try all g until a hit is found (usually found very quick).

- **Euclidean Step:** For $i > j$, $\gcd(i, j) = \gcd(i - j, j) \le (i - j)$
- **Lattice Points:** Points on segment (x_1, y_1) to (x_2, y_2) is $\gcd(|x_1 - x_2|, |y_1 - y_2|) + 1$
- **Power Divisibility:** Count $x \le n$ such that $d|x^k$:

$$\sum_{x=1}^{n} [d|x^k] = \left\lfloor \frac{n}{\prod p_i^{\lceil e_i/k \rceil}} \right\rfloor \quad \text{where } d = \prod p_i^{e_i}$$

- **Odd Divisor Count:** $d(n)$ is odd $\iff$ n is a perfect square.
- **Odd Divisor Sum:** $\sigma(n)$ is odd $\iff$ $n = 2^r k^2$ (n is square or twice a square).
- **Triple Divisor Sum:** Sum of $d(ijk)$ for $i \le A, j \le B, k \le C$:

$$\sum_{i,j,k} d(ijk) = \sum_{\gcd(i,j)=\gcd(j,k)=\gcd(k,i)=1} \left\lfloor \frac{A}{i} \right\rfloor \left\lfloor \frac{B}{j} \right\rfloor \left\lfloor \frac{C}{k} \right\rfloor$$

- **Factorial Modulo n:** $(n-1)! \pmod n = \begin{cases} n-1 & n \text{ is prime} \\ 2 & n = 4 \\ 0 & \text{otherwise} \end{cases}$
- **Generalized Wilson:** Product of integers coprime to m modulo m:

$$\prod_{\substack{1 \le k < m \\ \gcd(k,m)=1}} k \equiv \begin{cases} -1 & m = 4, p^\alpha, 2p^\alpha \\ 1 & \text{otherwise} \end{cases} \pmod m$$

(-1 iff primitive root exists)
- **Linear Representations:** Number of solutions to $n = ax + by$ $(x, y \ge 0, \gcd(a, b) = 1)$:

$$\frac{n}{ab} - \left\{ \frac{b'n}{a} \right\} - \left\{ \frac{a'n}{b} \right\} + 1$$

($\{x\}$ = fractional part. a', b' are inverses: $aa' \equiv 1 \pmod b$, $bb' \equiv 1 \pmod a$)

Description: Properties of $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$. Key for tiling, GCD, and modular periodicity.

- **Binet's Formula:** $F_n = \frac{1}{\sqrt5}\left((\frac{1+\sqrt5}{2})^n - (\frac{1-\sqrt5}{2})^n \right)$ *(Closed Form)*
- **Combinatorial Sum:** $F_n = \sum_{k=0}^{\lfloor \frac{n-1}{2} \rfloor} \binom{n-k-1}{k}$ *(Sum of shallow diagonals)*
- **Sum of Odd Indices:** $\sum_{i=0}^{n-1} F_{2i+1} = F_{2n}$
- **Addition Identity:** $F_{m+n} = F_{m-1}F_n + F_m F_{n+1}$
- **Shifted Addition:** $F_{m+n-1} = F_m F_n + F_{m-1} F_{n-1}$
- **Doubling Identity:** $F_{2n} = F_n(F_{n+1} + F_{n-1}) = F_{n+1}^2 - F_{n-1}^2$ *(Fast doubling)*
- **General Subtraction:** $F_m F_{n+1} - F_{m-1} F_n = (-1)^n F_{m-n}$
- **Square Check:** n is Fib $\iff$ $5n^2 + 4$ or $5n^2 - 4$ is a perfect square.
- **Strong Divisibility:** $F_k | F_n \iff k | n$. *(Every k^{th} Fib is a multiple of F_k)*
- **Coprimality:** $\gcd(F_n, F_{n+1}) = 1$. Any 3 consecutive are pairwise coprime.
- **Pisano Period:** Sequence modulo n is periodic with period $\pi(n) \le 6n$.

Mobius Function

$$\mu(n) = \begin{cases} 0 & n \text{ is not square free} \\ 1 & n \text{ has even number of prime factors} \\ -1 & n \text{ has odd number of prime factors} \end{cases}$$

Mobius Inversion:

$$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d) g(n/d)$$

Other useful formulas/forms:
$\sum_{d|n} \mu(d) = [n = 1]$, $\phi(n) = \sum_{d|n} \mu(d) \frac{n}{d}$
$g(n) = \sum_{n|d} f(d) \Leftrightarrow f(n) = \sum_{n|d} \mu(\frac{d}{n}) g(d)$
$g(n) = \sum_{1 \le m \le n} f(\lfloor \frac{n}{m} \rfloor) \Leftrightarrow f(n) = \sum_{1 \le m \le n} \mu(m) g(\lfloor \frac{n}{m} \rfloor)$
If f multiplicative, $\sum_{d|n} \mu(d) f(d) = \prod_{\text{prime } p|n} (1 - f(p))$ and $\sum_{d|n} \mu^2(d) f(d) = \prod_{\text{prime } p|n} (1 + f(p))$.

If $s_f(n) = \sum_{i=1}^n f(i)$ is a prefix sum of mulitplicative f then $s_{f*g}(n) = \sum_{1 \le xy \le n} f(x)g(y)$. Then $s_f(n) = \{s_{f*g}(n) - \sum_{d=2}^n s_f(\lfloor n/d \rfloor) g(d)\}/g(1)$ where $f * g(n) = \sum_{d|n} f(d) g(n/d)$ (Dirichlet). Precompute (linear sieve) $O(n^{2/3})$ first values of s_f for complexity $O(n^{2/3})$.
Useful sums and convolutions: $\epsilon = \mu * \mathbf{1}$, $\text{id} = \phi * \mathbf{1}$, $\text{id} = g * \text{id}_2$, where $\epsilon(n) = [n = 1]$, $\mathbf{1}(n) = 1$, $\text{id}(n) = n$, $\text{id}_k(n) = n^k$, $g(n) = \sum_{d|n} \mu(d) nd$.
coprime pairs in $[1, n]$ is $\sum_{d=1}^n \mu(d) \lfloor n/d \rfloor^2$. Sum of GCD pairs in $[1, n]$ is $\sum_{d=1}^n \phi(d) \lfloor n/d \rfloor^2$. Sum of LCM pairs in $[1, n]$ is $\sum_{d=1}^n (\frac{\lfloor n/d \rfloor (1 + \lfloor n/d \rfloor)}{2})^2 g(d)$, where g is defined above with $g(p^k) = p^k - p^{k+1}$.

GCD and LCM

Description: Identities for simplifying GCD/LCM sums, counting coprime pairs, and optimizing range queries.

- **Euclidean & Base:** $\gcd(a, b) = \gcd(b, a \pmod b)$, $\gcd(a, 0) = a$
- **Product Relation:** $\gcd(a, b) \cdot \text{lcm}(a, b) = |a \cdot b|$
- **Linear Combination:** $\gcd(a + m \cdot b, b) = \gcd(a, b)$
- **Distributivity:** $\gcd(a, \text{lcm}(b, c)) = \text{lcm}(\gcd(a, b), \gcd(a, c))$
- **GCD of Exponents:** $\gcd(n^a - 1, n^b - 1) = n^{\gcd(a,b)} - 1$
- **Difference Trick:** $\gcd(A_L, \ldots, A_R) = \gcd(A_L, A_{L+1} - A_L, \ldots, A_R - A_{R-1})$
- **Gauss' Identity:** $\sum_{d|n} \phi(d) = n$ and $\gcd(a, b) = \sum_{k|a, k|b} \phi(k)$
- **Sum of LCM(1...n, n):** $\sum_{i=1}^n \text{lcm}(i, n) = \frac{n}{2}\left(1 + \sum_{d|n} d \cdot \phi(d)\right)$
- **Count GCD=k:** $\sum_{i=1}^n [\gcd(i, n) = k] = \phi(n/k)$
- **Sum of GCD:** $\sum_{k=1}^n \gcd(k, n) = \sum_{d|n} d \cdot \phi(n/d)$
- **Power of GCD:** $\sum_{k=1}^n x^{\gcd(k,n)} = \sum_{d|n} x^d \cdot \phi(n/d)$
- **Inverse GCD Sum:** $\sum_{k=1}^n \frac{1}{\gcd(k,n)} = \frac{1}{n} \sum_{d|n} d \cdot \phi(d)$
- **Weighted Inverse:** $\sum_{k=1}^n \frac{k}{\gcd(k,n)} = \frac{1}{2} \sum_{d|n} d \cdot \phi(d)$
- **Relation:** $\sum_{k=1}^n \frac{n}{\gcd(k,n)} = 2 \sum_{k=1}^n \frac{k}{\gcd(k,n)} - 1 \quad (n > 1)$
- **Coprime Pairs:** $\sum_{i=1}^n \sum_{j=1}^n [\gcd(i, j) = 1] = \sum_{d=1}^n \mu(d) \lfloor \frac{n}{d} \rfloor^2$
- **Sum of GCD(i, j):** $\sum_{i=1}^n \sum_{j=1}^n \gcd(i, j) = \sum_{d=1}^n \phi(d) \lfloor \frac{n}{d} \rfloor^2$
- **Weighted Coprime:** $\sum_{i=1}^n \sum_{j=1}^n i \cdot j [\gcd(i, j) = 1] = \sum_{i=1}^n \phi(i) i^2$
- **Sum of LCM(i, j):** $\sum_{i=1}^n \sum_{j=1}^n \text{lcm}(i, j) = \sum_{l=1}^n \left(\frac{\lfloor n/l \rfloor (\lfloor n/l \rfloor + 1)}{2} \right)^2 \sum_{d|l} \mu(d) l d$

Euler's Totient Function $\phi(n)$

Description: $\phi(n)$ counts positive integers $\le n$ that are relatively prime to n. Key for modular arithmetic and GCD counting.

- **Definition:** $\phi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right)$
- **Prime Power:** $\phi(p^k) = p^k - p^{k-1} = p^k(1 - \frac{1}{p})$
- **Multiplicative:** $\phi(mn) = \phi(m)\phi(n)$ *(If $\gcd(m, n) = 1$)*
- **General Product:** $\phi(mn) = \frac{\phi(m)\phi(n)d}{\phi(d)}$ *($d = \gcd(m, n)$)*
- **LCM Relation:** $\phi(\text{lcm}(m, n)) \cdot \phi(\gcd(m, n)) = \phi(m) \cdot \phi(n)$
- **Radical Identity:** $\frac{\phi(n)}{n} = \frac{\phi(\text{rad}(n))}{\text{rad}(n)}$ *($\text{rad}(n) = \prod_{p|n} p$)*
- **Gauss' Identity:** $\sum_{d|n} \phi(d) = n$
- **Möbius Inversion:** $\phi(n) = \sum_{d|n} \mu(d) \frac{n}{d} = \sum_{d|n} d \cdot \mu(\frac{n}{d})$

- **Sum $\phi\times$ Floor:** $\sum_{i=1}^{n}\phi(i)\lfloor\frac{n}{i}\rfloor=\frac{n(n+1)}{2}$
- **Sum Odd Indices:** $\sum_{i \text{ odd}}^{n}\phi(i)\lfloor\frac{n}{i}\rfloor=\sum_{k\ge1}[\frac{n}{2^k}]^2$ *([·] is round)*
- **Inverse Phi Sum:** $\sum_{d|n}\frac{\mu^2(d)}{\phi(d)}=\frac{n}{\phi(n)}$
- **Sum of Coprimes:** $\sum_{k=1,\gcd(k,n)=1}^{n}k=\frac{1}{2}n\phi(n)$ *(Avg = n/2)*
- **Weighted Double Sum:** $\sum_{i=1}^{n}\sum_{j=1}^{n}ij[\gcd(i,j)=1]=\sum_{i=1}^{n}\phi(i)i^2$
- **Shifted GCD Sum:** $\sum_{\gcd(k,n)=1}\gcd(k-1,n)=\phi(n)d(n)$
- **Count GCD=d:** There are exactly $\phi(n/d)$ integers $i\le n$ such that $\gcd(i,n)=d$.
- **Divisibility I:** $a|b \implies \phi(a)|\phi(b)$
- **Divisibility II:** $n|\phi(a^n-1)$ *(For $a,n>1$)*
- **Evenness:** $\phi(n)$ is even ($n\ge3$). If n has r odd primes, $2^r|\phi(n)$.
- **Power Tower:** $a^x\equiv a^{\phi(m)+(x \pmod{\phi(m)})} \pmod m$ *(If $x\ge\log_2 m$)*
- **Lower Bound:** $\phi(n)\ge\sqrt{n/2}$ *(Roughly; $\phi(n)\ge\log_2 n$)*
- **Jordan Function $J_k(n)$:** Counts k-tuples $\le n$ forming coprime $(k+1)$-tuple with n.
- **Jordan Formula:** $J_k(n)=n^k\prod_{p|n}(1-p^{-k})$ *($J_1(n)=\phi(n)$)*
- **Jordan Sum:** $\sum_{d|n}J_k(d)=n^k$

Partition Function: $p(n)$

Pattern: Form a sum n where the **order does not matter**.

- "How many ways to write n as a sum of positive integers?"
- "How many ways to put n *identical* balls into *identical* boxes?"

Definition: Number of ways of writing n as a sum of positive integers, disregarding order. **Sequence** $p(n)$ **for** $n=0,1,2,\dots$:

$$1,1,2,3,5,7,11,15,22,30,42,56,77,\dots$$

Recurrence (Pentagonal Number Theorem):

$$p(n)=\sum_{k\in\mathbb{Z}\setminus\{0\}}(-1)^{k-1}p(n-k(3k-1)/2)$$
$$=p(n-1)+p(n-2)-p(n-5)-p(n-7)+\dots$$

Ceils and Floors

For $x,y\in\mathbb{R}$, $m,n\in\mathbb{Z}$:

- $\lfloor x\rfloor\le x<\lfloor x\rfloor+1$; $\lceil x\rceil-1<x\le\lceil x\rceil$
- $-\lfloor x\rfloor=\lceil -x\rceil$; $-\lceil x\rceil=\lfloor -x\rfloor$
- $\lfloor x+n\rfloor=\lfloor x\rfloor+n$, $\lceil x+n\rceil=\lceil x\rceil+n$
- $\lfloor x\rfloor=m \Leftrightarrow x-1<m\le x<m+1$
- $\lceil x\rceil=n \Leftrightarrow n-1<x\le n<x+1$
- If $n>0$, $\lfloor\frac{\lfloor x\rfloor+m}{n}\rfloor=\lfloor\frac{x+m}{n}\rfloor$
- If $n>0$, $\lceil\frac{\lceil x\rceil+m}{n}\rceil=\lceil\frac{x+m}{n}\rceil$
- If $n>0$, $\lfloor\frac{\lfloor\frac{x}{m}\rfloor}{n}\rfloor=\lfloor\frac{x}{mn}\rfloor$
- If $n>0$, $\lceil\frac{\lceil\frac{x}{m}\rceil}{n}\rceil=\lceil\frac{x}{mn}\rceil$
- For $m,n>0$, $\sum_{k=1}^{n-1}\lfloor\frac{km}{n}\rfloor=\frac{(m-1)(n-1)+\gcd(m,n)-1}{2}$
- $\lfloor n/j\rfloor=x$ for $j\in[\lfloor n/(x+1)\rfloor+1,\lfloor n/x\rfloor]$
- Modulo definition: $a \pmod m=a-m\lfloor a/m\rfloor$

Recurrences

If $a_n=c_1a_{n-1}+\dots+c_ka_{n-k}$, and $r_1,\dots,r_k$ are distinct roots of $x^k-c_1x^{k-1}-\dots-c_k$, there are $d_1,\dots,d_k$ s.t.

$$a_n=d_1r_1^n+\dots+d_kr_k^n.$$

Non-distinct roots r become polynomial factors, e.g. $a_n=(d_1n+d_2)r^n$.

Trigonometry

$$\sin(v+w)=\sin v\cos w+\cos v\sin w$$
$$\cos(v+w)=\cos v\cos w-\sin v\sin w$$
$$\tan(v+w)=\frac{\tan v+\tan w}{1-\tan v\tan w}$$
$$\sin v+\sin w=2\sin\frac{v+w}{2}\cos\frac{v-w}{2}$$
$$\cos v+\cos w=2\cos\frac{v+w}{2}\cos\frac{v-w}{2}$$

$$(V+W)\tan(\frac{v-w}{2})=(V-W)\tan(\frac{v+w}{2})$$

V,W are sides opposite to angles v,w. $a\cos x+b\sin x=r\cos(x-\phi)$

$a\sin x+b\cos x=r\sin(x+\phi)$

where $r=\sqrt{a^2+b^2}, \phi=\operatorname{atan2}(b,a)$.

Geometry

Rectangles and Squares

- Area of a rectangle: $A=l\cdot w$
- Perimeter of a rectangle: $P=2l+2w$
- Diagonal of a rectangle: $d=\sqrt{l^2+w^2}$
- Area of a square: $A=\text{side}^2$
- Perimeter of a square: $P=4\cdot\text{side}$
- Diagonal of a square: $d=\sqrt{2}\cdot\text{side}$

Triangles

Side lengths: a,b,c; Semiperimeter: $p=\dfrac{a+b+c}{2}$

- Area: $A=\frac{1}{2}\cdot b\cdot h$
- Perimeter: $P=a+b+c$
- Heron's Area: $A=\sqrt{p(p-a)(p-b)(p-c)}$
- Circumradius: $R=\dfrac{abc}{4A}$
- Inradius: $r=\dfrac{A}{p}$
- Length of median: $m_a=\frac{1}{2}\sqrt{2b^2+2c^2-a^2}$
- Length of bisector: $s_a=\sqrt{bc\left[1-(a/(b+c))^2\right]}$
- Law of Sines: $\dfrac{\sin\alpha}{a}=\dfrac{\sin\beta}{b}=\dfrac{\sin\gamma}{c}=\dfrac{1}{2R}$
- Law of Cosines: $a^2=b^2+c^2-2bc\cos\alpha$
- Law of Tangents: $\dfrac{a+b}{a-b}=\dfrac{\tan((\alpha+\beta)/2)}{\tan((\alpha-\beta)/2)}$

Circles

- Area: $A=\pi\cdot r^2$
- Circumference: $C=2\pi\cdot r$
- Sector Area: $A_{\text{sector}}=\frac{\theta}{360^\circ}\cdot\pi\cdot r^2$ (in degrees)
- Arc Length: $l=\frac{\theta}{360^\circ}\cdot 2\pi\cdot r$ (in degrees)

Polygons (n-sided)

- Sum of interior angles: $(n-2)\times180^\circ$
- A single angle (regular): $\frac{(n-2)\times180^\circ}{n}$
- Amount of diagonals: $\frac{n(n-3)}{2}$
- Sum of exterior angles: 360°
- Area (regular): $\frac{1}{4}ns^2\cot(\frac{\pi}{n})$
- Area (with apothem): $\frac{1}{2}\cdot n\cdot s\cdot a$

3D Shapes

- **Cube:** Volume $V=s^3$, Surface Area $SA=6s^2$
- **Sphere:** Volume $V=\frac{4}{3}\pi r^3$, Surface Area $SA=4\pi r^2$
- **Cylinder:** Volume $V=\pi r^2h$, Surface Area $SA=2\pi r^2+2\pi rh$
- **Cone:** Volume $V=\frac{1}{3}\pi r^2h$, Surface Area $SA=\pi rs+\pi r^2$, where $s=\sqrt{h^2+r^2}$
- **Cuboid:** Volume $V=lwh$, Surface Area $SA=2(wh+lw+lh)$

Quadrilaterals

With side lengths a,b,c,d, diagonals e,f, diagonals angle θ, area A and magic flux $F=b^2+d^2-a^2-c^2$:

$$4A=2ef\cdot\sin\theta=F\tan\theta=\sqrt{4e^2f^2-F^2}$$

For cyclic quadrilaterals the sum of opposite angles is 180°, $ef=ac+bd$, and $A=\sqrt{(p-a)(p-b)(p-c)(p-d)}$

Pick's Theorem

For a polygon on a grid:

$$A=I+\frac{B}{2}-1$$

A = Area, I = Interior points, B = Boundary points.

Spherical coordinates

$$x=r\sin\theta\cos\phi \qquad r=\sqrt{x^2+y^2+z^2}$$
$$y=r\sin\theta\sin\phi \qquad \theta=\operatorname{acos}(z/\sqrt{x^2+y^2+z^2})$$
$$z=r\cos\theta \qquad \phi=\operatorname{atan2}(y,x)$$

Geometry

Description: Essential formulas for 2D/3D shapes, triangle properties, and lattice points.

- **Ellipse Area:** $A=\pi ab$ *(a,b are semi-axes)*
- **Regular Polygon Area:** $A=\frac{1}{2}nRr$ *(R = circumradius, r = apothem)*
- **Sector Area:** $A=\frac{\theta}{2}r^2$ *(θ in radians)*
- **Chord Length:** $d=2r\sin(\frac{\theta}{2})=2\sqrt{r^2-x^2}$ *(x = dist from center)*
- **Pick's Theorem:** $A=I+\frac{B}{2}-1$ *(I = interior, B = boundary points)*
- **Sphere Volume:** $V=\frac{4}{3}\pi r^3$
- **Cone Volume:** $V=\frac{1}{3}\pi r^2h$
- **Pyramid Volume:** $V=\frac{1}{3}Bh$ *(B = base area)*
- **Rectangular Prism:** $V=lwh$
- **Law of Sines:** $\frac{a}{\sin A}=\frac{b}{\sin B}=\frac{c}{\sin C}=2R$
- **Law of Cosines:** $c^2=a^2+b^2-2ab\cos C$
- **Altitude (h_a):** $h_a=\frac{2A}{a}=c\sin B=b\sin C$
- **Median (m_a):** $m_a=\frac{1}{2}\sqrt{2b^2+2c^2-a^2}$
- **Angle Bisector Theorem:** $\frac{BD}{DC}=\frac{AB}{AC}$ *(Divides opposite side)*
- **Circumradius (R):** $R=\frac{abc}{4A}=\frac{a}{2\sin A}$
- **Inradius (r):** $r=\frac{A}{s}=\frac{\sqrt{(s-a)(s-b)(s-c)}}{s}$ *(s = semi-perimeter)*

Coordinate Geometry

- **Distance (2 points):** $(x_1,y_1),(x_2,y_2)$ $D=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$
- **Midpoint:** $M=\left(\frac{x_1+x_2}{2},\frac{y_1+y_2}{2}\right)$
- **Slope (2 points):** $m=\frac{y_2-y_1}{x_2-x_1}$
- **Line (point-slope):** $y-y_1=m(x-x_1)$
- **Line (slope-intercept):** $y=mx+b$
- **Line (two-point):** $y-y_1=\frac{y_2-y_1}{x_2-x_1}(x-x_1)$
- **Line (general):** $Ax+By+C=0$
- **Slope (from general):** $m=-A/B$
- **Parallel lines:** have the same slope ($m_1=m_2$)
- **Perpendicular lines:** $m_1=-1/m_2$
- **Distance (point to line):** Point (x_0,y_0) to line $Ax+By+C=0$. $D=\frac{|Ax_0+By_0+C|}{\sqrt{A^2+B^2}}$
- **Area of Triangle (vertices):** $(x_1,y_1),(x_2,y_2),(x_3,y_3)$ $A=\frac{1}{2}|x_1(y_2-y_3)+x_2(y_3-y_1)+x_3(y_1-y_2)|$
- **Circle:** Center (h,k), radius r. $(x-h)^2+(y-k)^2=r^2$
- **Distance (2 circle centers):** $D=\sqrt{(h_2-h_1)^2+(k_2-k_1)^2}$
- **Tangent slope on circle:** At point (x_0,y_0) on circle $x^2+y^2=r^2$. $m=-x_0/y_0$
- **Area of Parallelogram (vertices):** $(x_1,y_1),\dots,(x_4,y_4)$ $A=|x_1y_2+x_2y_3+x_3y_4+x_4y_1-x_2y_1-x_3y_2-x_4y_3-x_1y_4|$
- **Ellipse:** $\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$
- **Hyperbola:** $\frac{x^2}{a^2}-\frac{y^2}{b^2}=1$
- **Parabola:** Vertex (h,k), focus $(h+p,k)$. $(x-h)=4p(y-k)$

Derivatives/Integrals

$$\frac{d}{dx}\arcsin x=\frac{1}{\sqrt{1-x^2}} \qquad \frac{d}{dx}\arccos x=-\frac{1}{\sqrt{1-x^2}}$$
$$\frac{d}{dx}\tan x=1+\tan^2 x \qquad \frac{d}{dx}\arctan x=\frac{1}{1+x^2}$$
$$\int\tan ax=-\frac{\ln|\cos ax|}{a} \qquad \int xe^{ax}dx=\frac{e^{ax}}{a^2}(ax-1)$$
$$\int e^{-x^2}=\frac{\sqrt{\pi}}{2}\operatorname{erf}(x) \qquad \int x\sin ax=\frac{\sin ax-ax\cos ax}{a^2}$$

$$\int_a^b f(x)g(x)dx=[F(x)g(x)]_a^b-\int_a^b F(x)g'(x)dx$$

Sums

Basic Sums

- $\sum_{i=1}^{n}1=n$
- $\sum_{i=1}^{n}i=\frac{n(n+1)}{2}$
- $\sum_{i=1}^{n}i^2=\frac{n(n+1)(2n+1)}{6}$
- $\sum_{i=1}^{n}i^3=\left(\frac{n(n+1)}{2}\right)^2=\frac{n^2(n+1)^2}{4}$
- $\sum_{i=1}^{n}i^4=\frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$
- Sum of first n odd: $\sum_{i=1}^{n}(2i-1)=n^2$
- Sum of first n even: $\sum_{i=1}^{n}2i=n(n+1)$

Arithmetic Progression (AP)

$a_n=a_1+(n-1)d$ $S_n=\frac{n}{2}(2a_1+(n-1)d)=\frac{n}{2}(a_1+a_n)$

$a_n=a_m+(n-m)d$

Geometric Progression (GP)

$a_n=a_1r^{(n-1)}$ $S_n=\frac{a_1(r^n-1)}{r-1}$ (finite) $S_\infty=\frac{a_1}{1-r}$ (for $|r|<1$) $P_n=a_1^nr^{n(n-1)/2}$ $c^a+c^{a+1}+\dots+c^b=\frac{c^{b+1}-c^a}{c-1}, c\ne1$

Bernoulli Numbers & Sum of Powers

Pattern: Compute $\sum_{i=1}^{n} i^k$ where n **is large** but k **is small**.
- "Find $(1^5 + 2^5 + \cdots + n^5) \pmod{10^9+7}$ for $n = 10^{18}$."

Sequence B_k for $k = 0, 1, 2, \ldots$:

$$1, \frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, 0, -\frac{1}{30}, \ldots$$

(Note: Using $B_1 = +1/2$. The $B_1 = -1/2$ convention also exists.)
EGF for B_k (using $B_1 = -1/2$):

$$\frac{x}{e^x - 1} = \sum_{k=0}^{\infty} B_k \frac{x^k}{k!}$$

Faulhaber's Formula (Sum of Powers):

$$\sum_{i=0}^{n-1} i^m = \frac{1}{m+1} \sum_{k=0}^{m} \binom{m+1}{k} B_k n^{m+1-k}$$

Combinatorics

Binomial Theorem

Description: Used for expanding powers of binomials $(a+b)^p$. The coefficients $\binom{p}{k}$ give the number of ways to choose k items from p.
Formula:

$$(a+b)^p = \sum_{k=0}^{p} \binom{p}{k} a^k b^{p-k}$$

Stars and Bars

Description: Used to find the number of ways to distribute **identical (unlabeled)** objects (n) into **distinct** bins (k).
Formulas:
- **Empty bins NOT valid (Positive Integer Solutions):** $\binom{n-1}{k-1}$
- **Empty bins VALID (Non-Negative Integer Solutions):** $\binom{n+k-1}{k-1}$
- **Bounded Constraints (via Inclusion-Exclusion):**

  Formula: $\sum_{j=0}^{k} (-1)^j \binom{k}{j} \binom{\text{Top}}{k-1}$

  (Stop summation when $Top < k-1$)
  - $0 \le x_i \le v$: Top $= n - j(v+1) + k - 1$
  - $0 \le x_i < v$: Top $= n - j(v) + k - 1$
  - $0 < x_i \le v$: Top $= n - j(v) - 1$
  - $0 < x_i < v$: Top $= n - j(v-1) - 1$

Binomial Coefficients $\binom{n}{k}$

Description: $\binom{n}{k}$ is the number of ways to choose k elements from n distinct elements. Essential for DP, probability, and modular arithmetic.
- **Definition:** $\binom{n}{k} = \frac{n!}{k!(n-k)!}$
- **Symmetry:** $\binom{n}{k} = \binom{n}{n-k}$
- **Multiplicative ($\mathcal{O}(k)$):** $\binom{n}{k} = \prod_{i=1}^{k} \frac{n-i+1}{i}$
- **Base Cases:** $\binom{n}{0} = 1, \quad \binom{n}{n} = 1$
- **Pascal's Identity ($\mathcal{O}(1)$ DP):** $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$
- **Absorption Identity:** $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$
- **Shifted Recurrence I:** $\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1}$
- **Shifted Recurrence II:** $\binom{n+1}{k} = \frac{n+1}{n-k+1}\binom{n}{k}$
- **Vandermonde's Identity:** $\binom{m+n}{r} = \sum_{k=0}^{r} \binom{m}{k}\binom{n}{r-k}$
- **Sum of Row (Total Subsets):** $\sum_{k=0}^{n} \binom{n}{k} = 2^n$
- **Sum of K (Weighted Sum):** $\sum_{k=1}^{n} k\binom{n}{k} = n2^{n-1}$
- **Sum of K^2 (Weighted Sum II):** $\sum_{k=1}^{n} k^2\binom{n}{k} = n(n+1)2^{n-2}$
- **Extraction Identity:** $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$ *(Isolate k)*
- **Even/Odd Index Sum:** $\sum_{i\ge 0}\binom{n}{2i} = \sum_{i \ge 0}\binom{n}{2i+1} = 2^{n-1}$
- **Alternating Partial Sum:** $\sum_{i=0}^{k}(-1)^i\binom{n}{i} = (-1)^k\binom{n-1}{k}$
- **Partial Row Sum:** $\sum_{i=0}^{n}\binom{2n}{i} = 2^{2n-1} + \frac{1}{2}\binom{2n}{n}$
- **Binomial Expansion:** $\sum_{i=0}^{n} k^i\binom{n}{i} = (k+1)^n$
- **Hockey-Stick Identity:** $\sum_{i=r}^{n}\binom{i}{r} = \binom{n+1}{r+1}$ *(Col sum: fix r, vary n)*
- **Parallel Summation:** $\sum_{i=0}^{k}\binom{n+i}{i} = \binom{n+k+1}{k}$ *(Diag sum: vary both)*
- **Fibonacci Sum:** $\sum_{k=0}^{n}\binom{n-k}{k} = Fib_{n+1}$ *(Shallow diagonals)*
- **Sum of Squares:** $\sum_{i=0}^{k}\binom{k}{i}^2 = \binom{2k}{k}$ *(Case $m = n = r = k$)*
- **Convolution Product:** $\sum_{k=0}^{n}\binom{n}{k}\binom{n}{n-k} = \binom{2n}{n}$
- **Fixed Element Convolution:** $\sum_{i=1}^{n}\binom{n}{i}\binom{n-1}{i-1} = \binom{2n-1}{n-1}$
- **Subset of a Subset:** $\sum_{k=q}^{n}\binom{n}{k}\binom{k}{q} = 2^{n-q}\binom{n}{q}$
- **Partial Sum of Squares:** $\sum_{i=0}^{n}\binom{2n}{i}^2 = \frac{1}{2}\left[\binom{4n}{2n} + \binom{2n}{n}^2\right]$

Stirling Numbers of the First Kind: $c(n,k)$

Pattern: Count permutations in terms of their **cycle structure**.
- "Arrange n people around k identical round tables."
- "Count permutations of n elements with exactly k cycles."
- Lets $[n,k]$ be the stirling number of the first kind, then

$$\left[{n \atop n-k}\right] = \sum_{0 \le i_1 < i_2 < i_k < n} i_1 i_2 \ldots . i_k.$$

Definition: Number of permutations of n items with k cycles.

$$c(n,k) = (n-1)c(n-1,k) + c(n-1,k-1)$$
$$c(n,0) = 0 \quad (n > 0), \quad c(0,0) = 1$$
$$\sum_{k=0}^{n} c(n,k)x^k = x(x+1)\ldots(x+n-1)$$

Sequence $c(n,2)$ for $n = 0, 1, 2, \ldots$:

$$0, 0, 1, 3, 11, 50, 274, 1764, 13068, \ldots$$

Stirling Numbers of the Second Kind: $S(n,k)$ or $\left\{{n \atop k}\right\}$

Pattern: Partition n **distinct items** into k **identical, non-empty boxes**.
- "How many ways to put n *labeled* balls into k *unlabeled* boxes?"
- "Count ways to partition a set of n elements into k non-empty subsets."

Definition: Number of partitions of n distinct elements into exactly k non-empty subsets.

$$S(n,k) = S(n-1,k-1) + k \cdot S(n-1,k)$$
$$S(n,1) = 1, \quad S(n,n) = 1$$
$$S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

- $S(n,2) = 2^{n-1} - 1$
- $S(n,k) \cdot k!$ = number of ways to color n nodes using colors from 1 to k such that each color is used at least once.
- An r-associated Stirling number of the second kind is the number of ways to partition a set of n objects into k subsets, with each subset containing at least r elements. It is denoted by $S_r(n,k)$ and obeys the recurrence relation.

  $$S_r(n+1,k) = kS_r(n,k) + \binom{n}{r-1}S_r(n-r+1,k-1)$$

- Denote the n objects to partition by the integers $1, 2, \ldots, n$. Define the reduced Stirling numbers of the second kind, denoted $S^d(n,k)$, to be the number of ways to partition the integers $1, 2, \ldots, n$ into k nonempty subsets such that all elements in each subset have pairwise distance at least d. That is, for any integers i and j in a given subset, it is required that $|i-j| \ge d$. It has been shown that these numbers satisfy, $S^d(n,k) = S(n-d+1, k-d+1), n \ge k \ge d$

Bell Numbers: $B(n)$

Pattern: Total ways to partition n **distinct items** (number of boxes doesn't matter).
- "Find the total number of equivalence relations on a set of n elements."
- "How many ways to put n *labeled* balls into *unlabeled* boxes?"
- Counts the number of partitions of a set.

Definition: Total number of partitions of n distinct elements.

$$B(n) = \sum_{k=0}^{n} S(n,k)$$

Sequence $B(n)$ for $n = 0, 1, 2, \ldots$:

$$1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \ldots$$

Recurrence (P-set construction):

$$B(n+1) = \sum_{k=0}^{n}\binom{n}{k}B(k)$$

Catalan Numbers: C_n

Pattern: One of the most famous sequences. Look for:
- **Balanced sequences:** "Valid (balanced) parenthesis strings of length $2n$."
- **Recursive splitting:** "Number of full binary trees with n nodes."
- **Non-crossing paths:** "Paths from (0, 0) to (n, n) on a grid that do not go above $y = x$."
- **Polygon triangulation:** "Ways to triangulate a convex polygon with $n+2$ sides."

Sequence C_n for $n = 0, 1, 2, \ldots$:

$$1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, \ldots$$

Closed Form:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1}$$

Recurrence Relations:

$$C_0 = 1, \quad C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$$
$$C_0 = 1, \quad C_{n+1} = \frac{2(2n+1)}{n+2}C_n$$

- **Balanced Parentheses count with prefix:** The count of balanced parentheses sequences consisting of $n+k$ pairs of parentheses where the first k symbols are open brackets. Let the number be $C_n^{(k)}$, then

  $$C_n^{(k)} = \frac{k+1}{n+k+1}\binom{2n+k}{n}$$

Eulerian Numbers: $E(n,k)$

Pattern: Count permutations based on their **"runs"** or **"ascents/descents"**.
- "Count permutations of $\{1, \ldots, n\}$ with exactly k ascents $(p_i < p_{i+1})$."

Definition: Number of n-permutations with exactly k rises (positions i with $p_i > p_{i-1}$).

$$E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$$
$$E(n,0) = E(n,n-1) = 1$$
$$E(n,k) = \sum_{j=0}^{k}(-1)^j\binom{n+1}{j}(k-j+1)^n$$

Derangements: $D(n)$ or $!n$

Pattern: The "mixed-up hats" or "secret santa" problem.
- "Count permutations of n elements where **no element is in its original position.**"
- "Find the number of permutations with **no fixed points** ($p_i \ne i$ for all i)."

Definition: Permutations of a set such that no element appears in its original position. **Sequence $D(n)$ for $n = 0, 1, 2, \ldots$:**

$$1, 0, 1, 2, 9, 44, 265, 1854, 14833, \ldots$$

Recurrence Relations:

$$D(n) = (n-1)(D(n-1) + D(n-2))$$
$$D(n) = n \cdot D(n-1) + (-1)^n$$
$$D(n) = \left\lfloor \frac{n!}{e} + \frac{1}{2} \right\rfloor = \left\lceil \frac{n!}{e} \right\rfloor \quad (n \ge 1)$$

Burnside's Lemma

Pattern: Count "distinct" objects under **symmetry** (rotations, reflections).
- "Count distinct ways to color a necklace/bracelet/cube under rotation."
- The key is "up to symmetry," "distinct under rotation," etc.

Definition: Given a group G of symmetries acting on a set X. The number of distinct elements of X up to symmetry (number of orbits) is:

$$|X/G| = \frac{1}{|G|}\sum_{g \in G}|X^g|$$

where $X^g = \{x \in X \mid g \cdot x = x\}$ are the elements fixed by g.
Special Case (Necklaces): For k colors and n beads, with $G = \mathbb{Z}_n$ (rotations):

$$\text{Count} = \frac{1}{n}\sum_{d|n}\phi(d) \cdot k^{n/d}$$

Permutation Cycles (EGF)

Pattern: Count permutations where **cycle lengths are restricted** to a set S.
- "Count permutations of n elements that consist *only* of cycles of length 2 (involutions)."

Definition: Let $g_S(n)$ be the number of n-permutations whose cycle lengths all belong to S. The Exponential Generating Function (EGF) is:

$$\sum_{n\ge 0} g_S(n)\frac{x^n}{n!} = \exp\left(\sum_{n \in S}\frac{x^n}{n}\right)$$

,

Lucas's Theorem

Pattern: Compute $\binom{n}{k} \pmod p$ where n, k **are large** but p **is a small prime.**
- "Calculate $\binom{10^{18}}{10^9} \pmod 7$."

Definition: Let n, m be non-negative integers and p a prime. Write n and m in base p:

$$n = n_k p^k + \cdots + n_1 p + n_0$$

$$m = m_k p^k + \cdots + m_1 p + m_0$$

Then:

$$\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod p$$

(Note: $\binom{a}{b} = 0$ if $a < b$)

Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots, (-\infty < x < \infty)$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots, (-1 < x \le 1)$$

$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \ldots, (-1 \le x \le 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, (-\infty < x < \infty)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots, (-\infty < x < \infty)$$

$$(1-x)^{-r} = \sum_{i=0}^{\infty} \binom{r+i-1}{i} x^i, (r \in \mathbb{R})$$

Bitwise Formulas

$$a|b = a \oplus b + a\&b$$

$$a \oplus (a\&b) = (a|b) \oplus b \qquad a \oplus b = (a\&b) \oplus (a|b)$$

$$a + b = a|b + a\&b \qquad a + b = a \oplus b + 2(a\&b)$$

$a - b = (a \oplus (a\&b)) - ((a|b) \oplus a) = ((a|b) \oplus b) - ((a|b) \oplus a) = (a \oplus (a\&b)) - (b \oplus (a\&b)) = ((a|b) \oplus b) - (b \oplus (a\&b))$

- k_{th} bit is set in x iff $x \mod 2^{k-1} - x \mod 2^k \ne 0$ ($= 2^k$ to be exact). It comes handy when you need to look at the bits of the numbers which are pair sums or subset sums etc.
- $n \mod 2^i = n\&(2^i - 1)$
- $1 \oplus 2 \oplus 3 \oplus \cdots \oplus (4k-1) = 0$ for any $k \ge 0$

Algorithms

Rotation of a n*m matrix: $(i,j) \to (j, n-i-1) \to (n-i-1, m-j-1) \to (m-j-1, i)$

Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x. It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x x p_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$ where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.
Expectation is linear:

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent X and Y,

$$V(aX + bY) = a^2 V(X) + b^2 V(Y).$$

Discrete distributions

Binomial distribution: The number of successes in n independent yes/no experiments, each which yields success with probability p is $\mathrm{Bin}(n,p)$, $n = 1, 2, \ldots$, $0 \le p \le 1$.

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$\mu = np, \; \sigma^2 = np(1-p)$$

$\mathrm{Bin}(n,p)$ is approximately $\mathrm{Po}(np)$ for small p.
First success distribution: The number of trials needed to get the first success in independent yes/no experiments, each which yields success with probability p is $\mathrm{Fs}(p)$, $0 \le p \le 1$.

$$p(k) = p(1-p)^{k-1}, \; k = 1, 2, \ldots$$

$$\mu = \frac{1}{p}, \; \sigma^2 = \frac{1-p}{p^2}$$

Poisson distribution: The number of events occurring in a fixed period of time t if these events occur with a known average rate κ and independently of the time since the last event is $\mathrm{Po}(\lambda)$, $\lambda = t\kappa$.

$$p(k) = e^{-\lambda} \frac{\lambda^k}{k!}, \; k = 0, 1, 2, \ldots$$

$$\mu = \lambda, \; \sigma^2 = \lambda$$

Continuous distributions

Uniform distribution: If the probability density function is constant between a and b and 0 elsewhere it is $\mathrm{U}(a,b)$, $a < b$.

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a+b}{2}, \; \sigma^2 = \frac{(b-a)^2}{12}$$

Exponential distribution: The time between events in a Poisson process is $\mathrm{Exp}(\lambda)$, $\lambda > 0$.

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \ge 0 \\ 0 & x < 0 \end{cases}$$

$$\mu = \frac{1}{\lambda}, \; \sigma^2 = \frac{1}{\lambda^2}$$

Normal distribution: Most real random values with mean μ and variance σ^2 are well described by $\mathcal{N}(\mu, \sigma^2)$, $\sigma > 0$.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

If $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$ then

$$aX_1 + bX_2 + c \sim \mathcal{N}(\mu_1 + \mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2)$$

Graph Theory

Cayley's Formula

Pattern: Count **spanning trees** on n **labeled** vertices in a **complete graph** K_n.
- "How many trees can be formed using n labeled nodes?"

Definition: The number of spanning trees on n labeled vertices (in K_n) is n^{n-2}. **Sequence n^{n-2} for $n = 1, 2, 3, \ldots$:**

$$1, 1, 3, 16, 125, 1296, 16807, \ldots$$

Generalizations:
- # with degrees d_i: $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\ldots(d_n-1)!}$ (Prufer Sequence)

Kirchhoff's Matrix Tree Theorem

Pattern: Count **spanning trees** in a **general graph** G (not complete).
- "Given a grid graph, find the number of spanning trees."

Definition: Counts spanning trees in a graph G.
1. Create the **Laplacian Matrix** $L = D - A$:
   - D = Degree Matrix (diagonal, $D_{ii} = \deg(i)$)
   - A = Adjacency Matrix

Or, $L_{ij} = \begin{cases} \deg(i) & \text{if } i = j \\ -1 & \text{if } i \ne j \text{ and } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$

2. Remove **any** row i and **any** column j to get $L_{i,j}$.
3. The number of spanning trees is $\det(L_{i,j})$.

Erdős–Gallai Theorem

Pattern: Given a sequence of numbers, can it be the **degree sequence** of a **simple graph**?
- "Is the sequence $d_1, \ldots, d_n$ a valid graphic sequence?"

Definition: A simple graph with node degrees $d_1 \ge \cdots \ge d_n$ exists iff:
1. $\sum_{i=1}^{n} d_i$ is even.
2. For every $k \in [1, n]$:

$$\sum_{i=1}^{k} d_i \le k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$$

Game Theory

Sprague–Grundy Theorem

Pattern: An **impartial game** (moves depend on position, not player).
- **Classic Nim:** "A game with multiple piles of stones."
- **Sum of games:** Game breaks into independent sub-games.

Definition: For impartial games.
- **Grundy Value (G-value) / Nim-sum:**

$$G(v) = \mathrm{mex}(\{G(v_i) \mid v \to v_i \text{ is a valid move}\})$$

where $\mathrm{mex}(S)$ is the Minimum Excluded value.
- **Losing Position:** $G(v) = 0$.
- **Winning Position:** $G(v) > 0$.
- **Sum of Games:** If a game is a sum of independent games $g_1, \ldots, g_k$:

$$G_{\text{total}} = G(g_1) \oplus G(g_2) \oplus \cdots \oplus G(g_k)$$

where $\oplus$ is the bitwise XOR operator.

Trivia

Pythagorean triples: The Pythagorean triples are uniquely generated by $a = k \cdot (m^2 - n^2)$, $b = k \cdot (2mn)$, $c = k \cdot (m^2 + n^2)$ with $m > n > 0$, $k > 0$, $\gcd(m,n) = 1$, both m, n not odd.
Primes: $p = 962592769$ is such that $2^{21} \mid p - 1$, which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.
Estimates: $\sum_{d|n} d = O(n \log\log n)$.
Prime Gaps: For primes $> 10^{12}$, the max gap is not definitively known, but a gap of 1600 is a safe upper bound for practical purposes. (The largest known gap is 1550).
Prime count: 5133 upto 5e4. 9592 upto 1e5. 17984 upto 2e5. 78498 upto 1e6. 5761455 upto 1e8.
max NOD $\le n$: 100 for n = 5e4. 500 for n = 1e7. 2000 for n = 1e10. 200 000 for n = 1e19.
max Unique Prime Factors: 6 upto 5e5. 7 upto 9e6. 8 upto 2e8. 9 upto 6e9. 11 upto 7e12. 15 upto 3e19.
Quadratic Residue: $(\frac{a}{p})$ is 0 if $p|a$, 1 if a is a quadratic residue, -1 otherwise. Euler: $(\frac{a}{p}) = a^{(p-1)/2} (\mod p)$ (prime). Jacobi: if $n = p_1^{e1} \cdots p_k^{ek}$ then $(\frac{a}{n}) = \prod (\frac{a}{p_i})^{e_i}$.
Chicken McNugget: If a, b coprime, there are $\frac{1}{2}(a-1)(b-1)$ numbers not of form $ax + by$ $(x, y \ge 0)$, the largest being $ab - a - b$.

Extra Formulas

Math Identities & Algebra

Description: Fundamental algebraic identities, inequalities, and optimization theorems.
- **Factorial Sum:** $\sum_{i=0}^{n} i \cdot i! = (n+1)! - 1$
- **Difference of Powers:** $a^n - b^n = (a-b)(a^{n-1} + a^{n-2}b + \cdots + b^{n-1})$
- **Weighted Geo. Sum:** $\sum_{i=1}^{n} ia^i = \frac{a(na^{n+1} - (n+1)a^n + 1)}{(a-1)^2}$
- **Lagrange's Identity:** $(\sum a_k^2)(\sum b_k^2) - (\sum a_k b_k)^2 = \sum_{i<j} (a_i b_j - a_j b_i)^2$
- **Subset Product Sum:** Sum of products of all subsets of A is $\prod_{i=1}^{n} (a_i + 1)$.
- **Min/Max Identity:** $\min(a+b, c) = a + \min(b, c-a)$
- **Abs Diff Identity:** $|a-b| + |b-c| + |c-a| = 2(\max(a,b,c) - \min(a,b,c))$
- **Floor Inequality:** $ab \le c \iff a \le \lfloor c/b \rfloor$ *(Also valid for* $\ge, >, <$*)*
- **Nested Floor:** $\lfloor \frac{\lfloor x/m \rfloor}{n} \rfloor = \lfloor \frac{x}{mn} \rfloor$ *(Same for* $\lceil \cdot \rceil$*)*
- **Vieta's Formulas:** $\sum_{1 \le i_1 < \cdots < i_k \le n} \left(\prod_{j=1}^{k} r_{i_j}\right) = (-1)^k \frac{a_{n-k}}{a_n}$
- **Min Absolute Error:** $\min_x \sum |a_i - x| \implies x = \mathrm{median}(a)$
- **Min Squared Error:** $\min_x \sum (a_i - x)^2 \implies x = \mathrm{mean}(a)$

Pythagorean Triples & Sum of Squares

Description: Generating $a^2 + b^2 = c^2$ and counting ways to write integers as sums of squares.
- **Euclid's Formula:** $a = k(m^2 - n^2), b = k(2mn), c = k(m^2 + n^2)$
  ($m > n$, $\gcd(m,n) = 1$, distinct parity generates primitive triples)
- **Count Hypotenuse** n: $\frac{1}{2}\left(\prod_{p|n, p \equiv 1 \pmod 4} (2\alpha_p + 1) - 1\right)$ $(n = \prod p^{\alpha_p})$
- **2 Squares** $(r_2(n))$: $4(d_1(n) - d_3(n))$ *(d_1, d_3 count divisors* $\equiv 1, 3 \pmod 4$*)*
- **4 Squares** $(r_4(n))$: $8 \sum_{d|n, 4 \nmid d} d$
- **8 Squares** $(r_8(n))$: $16 \sum_{d|n} (-1)^{n+d} d^3$

Divisor Functions $\sigma_x(n)$

Description: Properties of $\sigma_x(n) = \sum_{d|n} d^x$. σ_0 is count, σ_1 is sum.
- **Computation:** $\sigma_x(p^a) = \frac{p^{(a+1)x} - 1}{p^x - 1}$. Multiplicative: $\sigma_x(ab) = \sigma_x(a)\sigma_x(b)$.
- **Divisor Product:** $\prod_{d|n} d = n^{\sigma_0(n)/2}$
- **Summatory** σ_0: $\sum_{i=1}^{x} \sigma_0(i) = 2\sum_{k=1}^{\sqrt{x}} \lfloor \frac{x}{k} \rfloor - \lfloor \sqrt{x} \rfloor^2$ *(Hyperbola Method)*
- **Summatory** σ_1: $\sum_{i=1}^{x} \sigma_1(i) = \sum_{k=1}^{x} k \lfloor \frac{x}{k} \rfloor$

Modular Arithmetic Properties

Description: Essential identities for modular operations.
- **Cancellation Law:** $ac \equiv bc \pmod m \implies a \equiv b \pmod{m/\gcd(c,m)}$
- **Freshman's Dream:** $(x+y)^p \equiv x^p + y^p \pmod p$ *(p is prime)*
- **Modulo Distributivity:** $ab \equiv a(b \bmod c) \pmod{ac}$

Narayana Numbers $N(n,k)$

Description: Counts Dyck paths of length $2n$ with k peaks, or valid parentheses with k distinct nestings '()'.
- **Formula:** $N(n,k) = \frac{1}{n}\binom{n}{k}\binom{n}{k-1}$
- **Usage:** Number of expressions with n pairs of parentheses

containing exactly k immediate ‘()‘ sub-patterns.

- **Example:** $N(4,2) = 6$ (6 valid sequences of 4 pairs have exactly two ‘()‘ nestings).
- **Relation:** $\sum_{k=1}^{n} N(n,k) = C_n$ *(Sums to n-th Catalan number)*

Combinatorics

Description: Counting sequences, Pascal properties, inversions, and permutation restrictions.

- **Power of 2 in $\binom{2n}{n}$:** Highest power is 2^x, where x is the number of 1s in binary n.
- **Pascal Parity:** Odd terms in row n is 2^x (x = count of 1s in binary n). Row $2^n - 1$ is all odd.
- **Pascal Prime Row:** For prime p, all $\binom{p}{k}$ $(1 \le k < p)$ are divisible by p.
- **Primality Test:** $n \ge 2$ is prime $\iff$ $n \mid \binom{n}{k}$ for all $1 \le k < n$.
- **Kummer's Theorem:** Largest power of p dividing $\binom{n}{m}$ is the number of carries adding m to $n-m$ in base p.
- **No Adjacent 0s:** Binary sequences of length n with no adjacent 0s $= Fib_{n+1}$.
- **Comb. with Repetition:** Choose k from n elements with replacement $= \binom{n+k-1}{k}$.
- **Equal Group Division:** Ways to divide n into n/k groups of size k: $\frac{n!}{(k!)^{n/k}(n/k)!}$.
- **Integer Solutions:** Non-negative solutions to $x_1 + \cdots + x_k = n$ is $\binom{n+k-1}{n}$.
- **Separated Selection:** Choose n ids from b with dist $\ge k$: $\binom{b-(n-1)(k-1)}{n}$.
- **Alternating Sum:** $\sum_{i \text{ odd}}^{n} \binom{n}{i} a^{n-i} b^i = \frac{1}{2}((a+b)^n - (a-b)^n)$.
- **Quotient Sum:** $\sum_{i=0}^{n} \frac{\binom{k}{i}}{\binom{n}{i}} = \frac{\binom{n+1}{n-k+1}}{\binom{n}{k}}$.
- **Finite Difference:** If x_{i+1} is sum of prev row n times, n-th row first col is $p(n) = \sum_{k=0}^{n} \binom{n}{k} x(k)$.
- **Binomial Inversion I:** $P(n) = \sum_{k=0}^{n} \binom{n}{k} Q(k) \iff Q(n) = \sum_{k=0}^{n} (-1)^{n-k} \binom{n}{k} P(k)$.
- **Binomial Inversion II:** $P(n) = \sum_{k=0}^{n} (-1)^k \binom{n}{k} Q(k) \iff Q(n) = \sum_{k=0}^{n} (-1)^k \binom{n}{k} P(k)$.
- **Derangements:** $d(n) = (n-1)(d(n-1) + d(n-2))$ with $d(0) = 1, d(1) = 0$.
- **Involutions:** Permutations where $p^2 = id$. $a_n = a_{n-1} + (n-1)a_{n-2}$.
- **Restricted Cycles** $T(n,k)$: Permutations size n with all cycles $\le k$:

$$T(n,k) = \begin{cases} n! & n \le k \\ nT(n-1,k) - \frac{n!}{(n-k)!} T(n-k-1,k) & n > k \end{cases}$$

Template & Utils

PBDS (Ordered Set & Hash Map)

```cpp
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using orderS = tree<ll,null_type,less<ll>, rb_tree_tag, tree_order_statistics_node_update>;

struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
template <typename K, typename V>
using hash_map = gp_hash_table<K, V, custom_hash>;
```

Pragmas & Optimization

Description: Aggressive GCC optimizations. `Ofast` ignores strict IEEE floating point standards (be careful with geometry precision).

```cpp
#pragma GCC optimize("O3")
#pragma GCC optimize("Ofast,unroll-loops")
#pragma GCC optimize("tree-vectorize")
#pragma GCC target("avx2,sse4.2,popcnt")
```

Random Number Generator

Description: Mersenne Twister (`mt19937`) seeded with high-resolution clock. Much better than `rand()`.

```cpp
mt19937 rng(chrono::high_resolution_clock::now().time_since_epoch().count());
inline ll getrandom(ll a,ll b) { return uniform_int_distribution<ll>(a,b)(rng); }
```

Basic Math Utils

Description: 1. `bigmod`: Modular Exponentiation $\mathcal{O}(\log P)$. 2. `inversemod`: Modular Inverse using Fermat's Little Theorem (Requires Prime Mod). 3. `sqrtt`: Integer Square Root (avoids precision errors of `sqrt`).

```cpp
ll bigmod(ll base, ll power) {
    ll res = 1; ll p = base % mod;
    while (power > 0) {
        if (power % 2 == 1) res = ((res % mod) * (p % mod)) % mod;
        power /= 2;
        p = ((p % mod) * (p % mod)) % mod;
    }
    return res;
}
ll inversemod(ll base) { return bigmod(base, mod - 2); }

int gcd(ll a, ll b) {
    while (b) { a %= b; swap(a, b); }
    return a;
}
ll sqrtt(ll a) {
    long long x = sqrt(a) + 2;
    while (x * x > a) x--;
    return x;
}
```

Grid Moves (2D)

```cpp
int dx[]={-1, 1 , 0 , 0 , -1 ,-1, 1, 1};
int dy[]={ 0, 0 ,-1 , 1 , -1 , 1,-1, 1};
constexpr ld PI = 3.14159265358979323846264338327950288L;
```

CP Environment Setup

C++ Library Header (stdc.h)

Description: Run `g++ stdc.h -o stdc.h.gch` once to enable fast precompilation.

Base Solution File (template.cpp)

Description: The starting file for every problem. Includes the necessary macros and I/O setup.

```cpp
// IIUC_MARK_US
#include "bits/stdc++.h"
using namespace std;

#define sz(x) (int) (x).size()
#define all(x) begin(x), end(x)
#define rep(i, a, b) for (int i = a; i < (b); ++i)
using ll = long long; using pii = pair<int, int>;
using pll = pair<ll, ll>; using vi = vector<int>;
template<class T> using V = vector<T>;

inline void file() {
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif
}
int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
clock_t start= clock();
cerr << "Time: " <<((double)(clock() - start) / CLOCKS_PER_SEC)<<el;
}
```

Fast Compile & Run (cf)

Description: Saves as `cf`. Setup: `chmod +x cf`.

Usage: Run `./cf A` (no need for .cpp). Compiles with **-O2** and **C++17**, runs against `input.txt`, and shows execution time.

```bash
#!/bin/bash
g++ -o sol -Wall -Wextra -std=c++17 -O2 $1.cpp
if [ $? -eq 0 ]; then
    time ./sol < input.txt
fi
```

Runtime Error Check (rte)

Description: Saves as `rte`. Setup: `chmod +x rte`.

Usage: Run `./rte A` (no need for .cpp). Compiles with **AddressSanitizer** and **UBSan** to catch out-of-bounds, overflows, and memory leaks.

```bash
#!/bin/bash
g++ -o sol -std=c++17 -O2 -fsanitize=address,undefined $1.cpp
if [ $? -eq 0 ]; then
    ./sol < input.txt
fi
```

Stress Test Script (run.sh)

Description: Bash script to compare your solution against a brute force solution using a generator. Stops on the first mismatch.

Usage: Save as `run.sh`, give permission (`chmod +x run.sh`), and run (`./run.sh`).

```bash
set -e
g++ code.cpp -o code
g++ gen.cpp -o gen
g++ brute.cpp -o brute

for((i=1;; ++i)); do
    ./gen $i > input_file
    ./code < input_file > myAnswer
    ./brute < input_file > correctAnswer

    # -Z ignores trailing whitespace
    diff -Z myAnswer correctAnswer > /dev/null || break
    echo "Passed test: " $i
done

echo "WA on the following test:"
cat input_file
echo "Your answer is:"
cat myAnswer
echo "Correct answer is:"
cat correctAnswer
```

Generator (gen.cpp)

```cpp
#include <bits/stdc++.h>
using namespace std;
mt19937 rng;
long long rand(long long l, long long r) {
    uniform_int_distribution<long long> dist(l, r);
    return dist(rng);
}
int main(int argc, char* argv[]) {
    // Seed rng with test case number
    rng.seed(atoi(argv[1]));
    int n = rand(1, 10);
    cout << n << endl;
    for(int i = 0; i < n; i++) {
        cout << rand(1, 100) << (i == n-1 ? "" : " ");
    }
    cout << endl;
}
```