LNMIIT - Tushar Sukhwal

```
class DisjointSet {
 vector<int> rank, parent, size;

public:
 DisjointSet(int n) {
   rank.resize(n + 1, 0);
   parent.resize(n + 1);
   size.resize(n + 1);
   for (int i = 0; i <= n; i++) {
     parent[i] = i;
     size[i] = 1;
   }
 }
 int findUPar(int node) {
   if (node == parent[node]) return node;
   return parent[node] = findUPar(parent[node]);
 }
 void unionByRank(int u, int v) {
   int ulp_u = findUPar(u);
   int ulp_v = findUPar(v);
   if (ulp_u == ulp_v) return;
   if (rank[ulp_u] < rank[ulp_v]) {
     parent[ulp_u] = ulp_v;
   } else if (rank[ulp_u] > rank[ulp_v]) {
     parent[ulp_v] = ulp_u;
   } else {
     parent[ulp_v] = ulp_u;
     rank[ulp_u]++;
   }
 }
 void unionBySize(int u, int v) {
   int ulp_u = findUPar(u);
   int ulp_v = findUPar(v);
   if (ulp_u == ulp_v) return;
```

```
// LCA

const int N = 2e5 + 5, L = 20;
vector<int> g[N];
int up[N][L], tin[N], tout[N], d[N], T;

void dfs(int u, int p) {
 tin[u] = ++T;
 up[u][0] = p;
 for (int i = 1; i < L; i++) up[u][i] = up[up[u][i - 1]][i - 1];
 for (int v : g[u])
   if (v != p) d[v] = d[u] + 1, dfs(v, u);
 tout[u] = ++T;
}

bool anc(int u, int v) { return tin[u] <= tin[v] && tout[u] >= tout[v]; }

int lca(int u, int v) {
 if (anc(u, v)) return u;
 if (anc(v, u)) return v;
 for (int i = L - 1; i >= 0; i--)
   if (!anc(up[u][i], v)) u = up[u][i];
 return up[u][0];
}

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

int lift(int u, int k) {
 if (k > d[u]) return -1;
 for (int i = L - 1; i >= 0; i--)
   if (k >= (1 << i)) u = up[u][i], k -= (1 << i);
```

```
//SegTree
const int N = 2e5 + 5;
int tree[4*N], a[N], n;

void build(int node, int start, int end) {
   if(start == end) {
       tree[node] = a[start];
       return;
   }
   int mid = (start + end) >> 1;
   build(2*node, start, mid);
   build(2*node+1, mid+1, end);
   tree[node] = tree[2*node] ^ tree[2*node+1];  // Change operation here
}

void update(int node, int start, int end, int idx, int val) {
   if(start == end) {
       tree[node] = val;
       a[idx] = val;
       return;
   }
   int mid = (start + end) >> 1;
   if(idx <= mid) update(2*node, start, mid, idx, val);
   else update(2*node+1, mid+1, end, idx, val);
   tree[node] = tree[2*node] ^ tree[2*node+1];  // Change operation here
}

int query(int node, int start, int end, int l, int r) {
   if(r < start || end < l) return 0;  // Change
   if (size[ulp_u] < size[ulp_v]) {
     parent[ulp_u] = ulp_v;
     size[ulp_v] += size[ulp_u];
   } else {
     parent[ulp_v] = ulp_u;
     size[ulp_u] += size[ulp_v];
   }
 }
};
```

```
 return u;
}

// Call this in main before queries
void init(int root = 0) {
 T = 0;
 d[root] = 0;
 dfs(root, root);
}
```

```
identity element
   if(l <= start && end <= r) return tree[node];
   int mid = (start + end) >> 1;
   return query(2*node, start, mid, l, r) ^
          query(2*node+1, mid+1, end, l, r);  //
Change operation here
}

// Usage example:
// int n = array_size;
// for(int i = 0; i < n; i++) cin >> a[i];
// build(1, 0, n-1);
// update(1, 0, n-1, idx, val);
// int result = query(1, 0, n-1, left, right);
```

```
vector<int> kahntopo(vector<bool>& vis,
vector<vector<int>>& gp) {
 int n = gp.size();
 vector<int> degree(n, 0);
 for (int i = 0; i < n; i++) {
   for (int j = 0; j < gp[i].size(); j++) {
     degree[gp[i][j]]++;
   }
 }
 queue<int> q;
 for (int i = 0; i < n; i++) {
   if (degree[i] == 0) {
     q.push(i);
   }
 }
 vector<int> topo;
 while (!q.empty()) {
   int node = q.front();
   topo.push_back(q.front());
   q.pop();
   for (auto child : gp[node]) {
```

```
const int MOD = 1e9 + 7;
template <class T>
class Math {
public:
 vector<T> fact, invfact;
 // Math<datatype> objname(n); use like this
 Math() {}
 Math(int n) {
   fact.resize(n);
   invfact.resize(n);
   fact[0] = invfact[0] = 1;
   for (int i = 1; i < n; i++) {
     fact[i] = modmul(i, fact[i - 1]);
     invfact[i] = modinv(fact[i]);
   }
 }
 T binpow(T a, T b, T m = MOD) {
   T res = 1;
   while (b > 0) {
     if (b & 1) res = modmul(res, a, m);
     a = modmul(a, a, m);
```

```
vector<int> NGE(vector<int> v) {  // O(2N) at worst
possible case
 int n = v.size();
 vector<int> nge(n);
 stack<int> st;
 for (int i = n - 1; i >= 0; i--) {
   while (!st.empty() && st.top() <= v[i]) {
     st.pop();
   }
   if (st.empty()) {
     nge[i] = -1;
   } else {
     nge[i] = st.top();
   }
   st.push(v[i]);
 }
}
------------------------------------------------------
-------
//Trie
struct Node {
     degree[child]--;
     if (degree[child] == 0) {
       q.push(child);
     }
   }
 }
 return topo;
}
---------------------------------------------------

vector<int> dijkstra(vector<vector<pair<int, int>>>
&adj, int src) {
 int v = adj.size();
 vector<int> dist(v, INT_MAX);
 dist[src] = 0;

 priority_queue<int, vector<int>, greater<int>> pq;

 pq.push(src);

 while (!pq.empty()) {
   auto curr = pq.top();
   pq.pop();

   for (auto it : adj[curr]) {
     int u = it.first;
     int wt = it.second;

     if (wt + dist[curr] < dist[u]) {
       dist[u] = wt + dist[curr];
       pq.push(u);
     }
   }
 }
 return dist;
```

```
    b >>= 1;
  }
  return res;
}
T modadd(T a, T b, T m = MOD) {
  a = a % m;
  b = b % m;
  return (((a + b) % m) + m) % m;
}
T modsub(T a, T b, T m = MOD) {
  a = a % m;
  b = b % m;
  return (((a - b) % m) + m) % m;
}
T modmul(T a, T b, T m = MOD) {
  a = a % m;
  b = b % m;
  return (((T)a * (T)b % m) + m) % m;
}
T modpow(T x, T y, T m = MOD) {
  T res = 1;
  x = x % m;
  while (y > 0) {
    if (y & 1) res = (res * x) % m;
    y = y >> 1;
    x = (x * x) % m;
  }
  return res;
}
T modinv(T x, T m = MOD) { return modpow(x, m - 2, m); }
T choose(T n, T k) {
  if (k < 0 || k > n) return 0;
  T ans = fact[n];
  ans = modmul(ans, invfact[k]);
  ans = modmul(ans, invfact[n - k]);
```

```
 Node* links[26];
 bool flag = false;
 bool containsref(char ch) { return links[ch - 'a'] !=
NULL; }
 void putref(char ch, Node* ref) { links[ch - 'a'] =
ref; }
 Node* getref(char ch) { return links[ch - 'a']; }
 void setend() { flag = true; }
 bool isend() { return flag; }
};
class Trie {
private:
 Node* root;
public:
 Trie() { root = new Node(); }
 void insert(string word) {  // O(word.size())
   Node* curr = root;
   for (int i = 0; i < word.size(); i++) {
     if (!curr->containsref(word[i])) {
       curr->putref(word[i], new Node());
     }
     curr = curr->getref(word[i]);
   }
   curr->setend();
 }
 bool search(string word) {  // O(word.size())
   Node* curr = root;
   for (int i = 0; i < word.size(); i++) {
     if (!curr->containsref(word[i])) {
       return false;
     }
     curr = curr->getref(word[i]);
   }
   if (curr->isend()) {
     return true;
```

```
}
----------------------------------------------------------
-------
vector<int> kmp(string s) {  // O(n)
 int n = s.size();
 vector<int> pi(n, 0);
 for (int i = 1; i < n; i++) {
   int j = pi[i - 1];  // how many characters
matched till here max
   while (j > 0 && s[i] != s[j]) {
     j = pi[j - 1];
   }  // keep going back until something matches but
i remains where it was
   if (s[i] == s[j]) {
     j++;
   }
   pi[i] = j;
 }
 return pi;
}

int string_matching(string s, string pat) {
 vector<int> pi = kmp(s);
 cout << endl;
 int i = 0, j = 0;
 int ans_pos = -1;
 while (i < s.size()) {
   if (s[i] == pat[j]) {
     i++, j++;
   } else {
     if (j > 0) {
       j = pi[j - 1];
     } else {
       i++;
     }
```

```
   return ans;
 }
};
----------------------------------------------------------
--------
template <typename T>
class SparseTable {
private:
 vector<vector<T>> table;
 vector<int> bin_log;
 int n;

public:
 SparseTable(vector<T>& arr) {
   n = arr.size();
   int log = 32 - __builtin_clz(n);
   table.resize(n, vector<T>(log));
   bin_log.resize(n + 1);
   bin_log[1] = 0;
   for (int i = 2; i <= n; i++) {
     bin_log[i] = bin_log[i / 2] + 1;
   }

   build(arr);
 }

 void build(vector<T>& arr) {  // Nlog(N)
   for (int i = 0; i < n; i++) {
     table[i][0] = arr[i];
   }

   for (int j = 1; (1 << j) <= n; j++) {
     for (int i = 0; i + (1 << j) - 1 < n; i++) {
       table[i][j] = min(table[i][j - 1], table[i + (1 <<
(j - 1))][j - 1]);
     }
```

```
   }
   return false;
 }
 bool startswith(string prefix) {  // O(prefix.size())
   Node* curr = root;
   for (int i = 0; i < prefix.size(); i++) {
     if (!curr->containsref(prefix[i])) {
       return false;
     }
     curr = curr->getref(prefix[i]);
   }
   return true;
 }
};
----------------------------------------------------------
-------
double f(double x) {
 return -x * x + 2 * x + 3;  // Change this function
}

// Returns x coordinate of maximum
double ternary_search(double l, double r) {
 const double eps = 1e-9;
 while (r - l > eps) {
   double m1 = l + (r - l) / 3;
   double m2 = r - (r - l) / 3;
   double f1 = f(m1);
   double f2 = f(m2);

   if (f1 < f2)
     l = m1;  // For maximum
   else
     r = m2;  // Change < to > for minimum
 }
 return l;
}
```

```
   }
   if (j == pat.size()) {
     ans_pos = i - pat.size();
     break;
   }
 }
 return ans_pos;
}
------------------------------------------------------------

// Basic Kadane's - returns max sum
long long kadane(vector<int>& arr) {
 long long maxSoFar = arr[0], maxEndingHere = arr[0];
 for (int i = 1; i < arr.size(); i++) {
   maxEndingHere = max(1LL * arr[i], maxEndingHere + arr[i]);
   maxSoFar = max(maxSoFar, maxEndingHere);
 }
 return maxSoFar;
}

// Kadane's with subarray indices
array<long long, 3> kadaneWithIndex(vector<int>& arr) {
 long long maxSoFar = arr[0], maxEndingHere = arr[0];
 int start = 0, end = 0, s = 0;
 for (int i = 1; i < arr.size(); i++) {
   if (maxEndingHere + arr[i] < arr[i]) {
     maxEndingHere = arr[i];
     s = i;
   } else {
     maxEndingHere = maxEndingHere + arr[i];
   }
```

```
   }
 }

 T query(int L, int R) {
   int length = R - L + 1;
   int k = bin_log[length];
   return min(table[L][k], table[R - (1 << k) + 1][k]);
 }
};
-------------------------------------------------------------------
struct DifferenceArray {
 vector<long long> diff, arr;
 int n;

 // Initialize with original array
 DifferenceArray(vector<int>& a) {
   n = a.size();
   diff.resize(n + 1, 0);
   arr = vector<long long>(a.begin(), a.end());
   build();
 }

 void build() {
   diff[0] = arr[0];
   for (int i = 1; i < n; i++) diff[i] = arr[i] - arr[i - 1];
 }

 // Add val to range [l,r]
 void update(int l, int r, long long val) {
   diff[l] += val;
   if (r + 1 < n) diff[r + 1] -= val;
 }

 // Get final array after updates
```

```

// For arrays/discrete values
int arr[100005];  // Global array
int discrete_ts(int l, int r) {
 while (r - l > 2) {
   int m1 = l + (r - l) / 3;
   int m2 = r - (r - l) / 3;

   if (arr[m1] < arr[m2])
     l = m1;  // For maximum
   else
     r = m2;  // Change < to > for minimum
 }

 int ans = arr[l], best = l;
 for (int i = l + 1; i <= r; i++) {
   if (arr[i] > ans) {
     ans = arr[i];
     best = i;
   }
 }
 return best;
}
-------------------------------------------------
// Basic Sieve - generates primes up to N
vector<bool> sieve(int N) {
 vector<bool> prime(N + 1, true);
 prime[0] = prime[1] = false;
 for (int i = 2; i * i <= N; i++)
   if (prime[i])
     for (int j = i * i; j <= N; j += i) prime[j] = false;
 return prime;
}
```

```
   if (maxEndingHere > maxSoFar) {
     maxSoFar = maxEndingHere;
     start = s;
     end = i;
   }
 }
 return {maxSoFar, start, end};
}

// Circular array maximum sum
long long circularKadane(vector<int>& arr) {
 long long normalSum = kadane(arr);
 if (normalSum < 0) return normalSum;

 long long totalSum = 0;
 for (int i = 0; i < arr.size(); i++) {
   totalSum += arr[i];
   arr[i] = -arr[i];
 }
 long long circularSum = totalSum + kadane(arr);
 return max(normalSum, circularSum);
}
----------------------------------------------------
-------
//cycle directed graph
// take care of non-connected graphs
bool dfs(int node, int par, vector<bool> &vis,
vector<bool> &pathvis,
        vector<vector<int>> &gp) {
 bool ans = false;
 vis[node] = true;
 pathvis[node] = true;
 for (auto &child : gp[node]) {
   if (pathvis[child] == true) {
     // cout << "HERE" << endl;
```

```
 vector<long long> getArray() {
   vector<long long> res(n);
   res[0] = diff[0];
   for (int i = 1; i < n; i++) res[i] = res[i - 1] +
diff[i];
   return res;
 }

 // Get value at index after updates
 long long getValue(int idx) {
   long long sum = 0;
   for (int i = 0; i <= idx; i++) sum += diff[i];
   return sum;
 }
};
----------------------------------------------------------
---------
// Returns {MST weight, MST edges}
pair<int, vector<pair<int, int>>> primMST(int V,
vector<vector<int>> adj[]) {
 priority_queue<pair<int, int>, vector<pair<int, int>>,
                greater<pair<int, int>>>
     pq;
 vector<int> vis(V, 0);
 vector<pair<int, int>> mst;  // stores edges in MST

 pq.push({0, 0});
 int sum = 0;
 vector<int> parent(V, -1);  // track parent for MST
construction

 while (!pq.empty()) {
   auto it = pq.top();
   pq.pop();
   int node = it.second;
   int wt = it.first;
```

```
// Modified Sieve - stores smallest prime factor
vector<int> spf(int N) {
 vector<int> spf(N + 1);
 for (int i = 2; i <= N; i++) spf[i] = i;
 for (int i = 2; i * i <= N; i++)
   if (spf[i] == i)
     for (int j = i * i; j <= N; j += i)
       if (spf[j] == j) spf[j] = i;
 return spf;
}

// Prime factorization using SPF - O(log n)
vector<int> factorize(int x, vector<int>& spf) {
 vector<int> factors;
 while (x != 1) {
   factors.push_back(spf[x]);
   x = x / spf[x];
 }
 return factors;
}

// Linear Sieve - O(n) time complexity
vector<int> linearSieve(int N) {
 vector<int> lp(N + 1), pr;
 for (int i = 2; i <= N; i++) {
   if (lp[i] == 0) {
     lp[i] = i;
     pr.push_back(i);
   }
   for (int j = 0; j < pr.size() && pr[j] <= lp[i] &&
i * pr[j] <= N; j++)
     lp[i * pr[j]] = pr[j];
 }
 return pr;
}
```

```
     return true;
   }
   if (!vis[child]) ans |= dfs(child, node, vis,
pathvis, gp);
   if (ans) {
     return true;
   }
 }
 pathvis[node] = false;
 return ans;
}
-----------------------------------------------------
-------// single source shortest path (negatives
edeges also)

// after one for relaxation if distance reduces then
negative cycle
vector<int> bellman_ford(int V, vector<vector<int>>&
edges, int S) {
 vector<int> dist(V, 1e8);
 dist[S] = 0;
 for (int i = 0; i < V - 1; i++) {
   for (auto it : edges) {
     int u = it[0];
     int v = it[1];
     int wt = it[2];
     if (dist[u] != 1e8 && dist[u] + wt < dist[v]) {
       dist[v] = dist[u] + wt;
     }
   }
 }
 // Nth relaxation to check negative cycle
 for (auto it : edges) {
   int u = it[0];
   int v = it[1];
```

```
   if (vis[node]) continue;

   vis[node] = 1;
   sum += wt;
   if (parent[node] != -1) mst.push_back({parent[node],
node});

   for (auto it : adj[node]) {
     int adjNode = it[0];
     int edW = it[1];
     if (!vis[adjNode]) {
       parent[adjNode] = node;
       pq.push({edW, adjNode});
     }
   }
 }
 return {sum, mst};
}

// Usage:
// auto [weight, tree] = primMST(V, adj);
// tree contains edges {u,v} in MST

int spanningTree(int V, vector<vector<int>> adj[]) {
 // 1 - 2 wt = 5
 /// 1 - > (2, 5)
 // 2 -> (1, 5)

 // 5, 1, 2
 // 5, 2, 1
 vector<pair<int, pair<int, int>>> edges;
 for (int i = 0; i < V; i++) {
   for (auto it : adj[i]) {
     int adjNode = it[0];
```

```
// Segmented Sieve for range [L,R]
vector<bool> segmentedSieve(long long L, long long R)
{
 // Generate primes up to sqrt(R)
 int limit = sqrt(R);
 vector<int> primes = simpleSieve(limit);

 // Mark primes in [L,R]
 vector<bool> isPrime(R - L + 1, true);
 if (L == 1) isPrime[0] = false;

 // Mark composites in range
 for (int p : primes) {
   // Find first multiple of p >= L
   long long firstMultiple = (L / p) * p;
   if (firstMultiple < L) firstMultiple += p;
   if (firstMultiple == p) firstMultiple += p;

   // Mark multiples of p in range
   for (long long j = firstMultiple; j <= R; j += p)
isPrime[j - L] = false;
 }
 return isPrime;
}

// Usage example:
// vector<bool> primes = segmentedSieve(1000000000,
1000001000);
// First index (0) corresponds to number L
// Time: O(√R + (R-L+1)log(log√R))
// Space: O(√R + (R-L+1))
------------------------------------------------------
--------
void dfs(int node, vector<int> &vis, vector<int>
   int wt = it[2];
   if (dist[u] != 1e8 && dist[u] + wt < dist[v]) {
     return {-1};
   }
 }

 return dist;
}
---------------------------------------------
//Bridges
int timer = 1;
void dfs(int node, int parent, vector<int> &vis,
vector<int> adj[], int tin[],
        int low[], vector<pair<int, int>> &bridges)
{
 vis[node] = 1;
 tin[node] = low[node] = timer++;
 for (auto it : adj[node]) {
   if (it == parent) continue;
   if (!vis[it]) {
     dfs(it, node, vis, adj, tin, low, bridges);
     low[node] = min(low[it], low[node]);
     if (low[it] > tin[node]) bridges.push_back({it,
node});
   } else
     low[node] = min(low[node], low[it]);
 }
}

vector<pair<int, int>> findBridges(int n,
vector<pair<int, int>> &edges) {
 vector<int> adj[n];
 for (auto [u, v] : edges) {
   adj[u].push_back(v);
   adj[v].push_back(u);
```

```
     int wt = it[1];
     int node = i;

     edges.push_back({wt, {node, adjNode}});
   }
 }
 DisjointSet ds(V);
 sort(edges.begin(), edges.end());
 int mstWt = 0;
 for (auto it : edges) {
   int wt = it.first;
   int u = it.second.first;
   int v = it.second.second;

   if (ds.findUPar(u) != ds.findUPar(v)) {
     mstWt += wt;
     ds.unionBySize(u, v);
   }
 }

 return mstWt;
}
----------------------------------------------------------
---------
//Articulation Point
int timer = 1;
void dfs(int node, int parent, vector<int> &vis, int
tin[], int low[],
        vector<int> &ap, vector<int> adj[]) {
 vis[node] = 1;
 tin[node] = low[node] = timer++;
 int child = 0;
 for (auto it : adj[node]) {
   if (it == parent) continue;
   if (!vis[it]) {
     dfs(it, node, vis, tin, low, ap, adj);
```

```
adj[], stack<int> &st) {
 vis[node] = 1;
 for (auto it : adj[node]) {
   if (!vis[it]) dfs(it, vis, adj, st);
 }
 st.push(node);
}

void dfs3(int node, vector<int> &vis, vector<int>
adjT[], vector<int> &comp) {
 vis[node] = 1;
 comp.push_back(node);
 for (auto it : adjT[node]) {
   if (!vis[it]) dfs3(it, vis, adjT, comp);
 }
}

// Returns {number of SCCs, components}
pair<int, vector<vector<int>>> kosaraju(int V,
vector<int> adj[]) {
 vector<int> vis(V, 0);
 stack<int> st;
 for (int i = 0; i < V; i++)
   if (!vis[i]) dfs(i, vis, adj, st);

 vector<int> adjT[V];
 for (int i = 0; i < V; i++) {
   vis[i] = 0;
   for (auto it : adj[i]) adjT[it].push_back(i);
 }

 vector<vector<int>> sccs;
 int scc_count = 0;
 while (!st.empty()) {
   int node = st.top();
 }
 vector<int> vis(n);
 int tin[n], low[n];
 vector<pair<int, int>> bridges;
 dfs(0, -1, vis, adj, tin, low, bridges);
 return bridges;
}
```

```
     low[node] = min(low[node], low[it]);
     if (low[it] >= tin[node] && parent != -1) ap[node] = 1;
     child++;
   } else
     low[node] = min(low[node], tin[it]);
 }
 if (child > 1 && parent == -1) ap[node] = 1;
}

vector<int> findArticulationPoints(int n, vector<pair<int, int>> &edges) {
 vector<int> adj[n];
 for (auto [u, v] : edges) {
   adj[u].push_back(v);
   adj[v].push_back(u);
 }
 vector<int> vis(n), ap(n);
 int tin[n], low[n];

 for (int i = 0; i < n; i++)
   if (!vis[i]) dfs(i, -1, vis, tin, low, ap, adj);

 vector<int> ans;
 for (int i = 0; i < n; i++)
   if (ap[i]) ans.push_back(i);

 return ans.empty() ? vector<int>{-1} : ans;
}
```

```
   st.pop();
   if (!vis[node]) {
     vector<int> comp;
     dfs3(node, vis, adjT, comp);
     sccs.push_back(comp);
     scc_count++;
   }
 }
 return {scc_count, sccs};
}

// Usage:
// auto [count, components] = kosaraju(V, adj);
// count = number of SCCs
// components[i] = nodes in ith SCC
```

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

```
long long binpow(long long a, long long b) {
 long long res = 1;
 while (b > 0) {
   if (b & 1) res = res * a;
   a = a * a;
   b >>= 1;
 }
 return res;
}

int gcd(int a, int b) {if (b > a) {return gcd(b, a);} if (b == 0) {return a;} return gcd(b, a % b);}
int expo(int a, int b, int mod) {int res = 1; while (b > 0) {if (b & 1)res = (res * a) % mod; a = (a * a) % mod; b = b >> 1;} return res;}
void extendgcd(int a, int b, int*v) {if (b == 0) {v[0] = 1; v[1] = 0; v[2] = a; return ;} extendgcd(b, a % b, v); int x = v[1]; v[1] = v[0] - v[1] * (a / b); v[0] = x; return;}
int mminv(int a, int b) {int arr[3]; extendgcd(a, b, arr); return arr[0];}
int mminvprime(int a, int b) {return expo(a, b - 2, b);}
bool revsort(int a, int b) {return a > b;}
int combination(int n, int r, int m, int *fact, int *ifact) {int val1 = fact[n]; int val2 = ifact[n - r]; int val3 = ifact[r]; return (((val1 * val2) % m) * val3) % m;}
void google(int t) {cout << "Case #" << t << ": ";}
vector<int> sieve(int n) {int*arr = new int[n + 1](); vector<int> vect; for (int i = 2; i <= n; i++)if (arr[i] == 0) {vect.push_back(i); for (int j = 2 * i; j <= n; j += i)arr[j] = 1;} delete[] arr; return vect;}
int mod_add(int a, int b, int m) {a = a % m; b = b % m; return (((a + b) % m) + m) % m;}
int mod_mul(int a, int b, int m) {a = a % m; b = b % m; return (((a * b) % m) + m) % m;}
int mod_sub(int a, int b, int m) {a = a % m; b = b % m; return (((a - b) % m) + m) % m;}
int mod_div(int a, int b, int m) {a = a % m; b = b % m; return (mod_mul(a, mminvprime(b, m), m) + m) % m;}
int phin(int n) {int number = n; if (n % 2 == 0) {number /= 2; while (n % 2 == 0) n /= 2;} for (int i = 3; i <= sqrt(n); i += 2) {if (n % i == 0) {while (n % i == 0)n /= i; number = (number / i * (i - 1));}} if (n > 1)number = (number / n * (n - 1)); return number;}
```