Initial Setup and Definitions

Definitions

```cpp
typedef long long ll;
typedef vector< int > vi;
typedef vector< vi > vvi;
typedef pair< int, int > pii;
typedef vector< pii > vpii;
typedef vector< vpii > vvpii;
typedef pair< ll, ll > pll;
typedef vector< pll > vpll;
typedef vector< vpll > vvpll;
```

Fast Input

```cpp
ios_base::sync_with_stdio(false);
cin.tie(NULL);
cout.setf(ios::fixed);
cout.precision(4);
```

Mathematics

```cpp
#define gcd(a, b) __gcd(a, b)
#define lcm(a, b) gcd(a, b) ? ( (a)*(b) ) / gcd(a, b) ): 0
const double PI = 3.14159265358979323846264338327950;
const ll PRIME_BASE = (1 << 61) - 1;
```

Strings

Rolling Hashing

```cpp
struct RollingHashing {
  ll p, m, ns;
  vector< ll > pows, hash;
  RollingHashing(string s, ll p_=31, ll m_=1e9 + 7) {
    // if WA then other p and other m
    // if still WA then double hashing
    // if still WA maybe is not the answer RH
    p = p_; m = m_;
    ns = s.size();
    pows.resize(ns + 2);
    pows[0] = 1;
    for(int i = 1; i < ns + 2; i++)
      pows[i] = (pows[i - 1] * p) % m;
    hash.resize(ns + 1);
    hash[0] = 0;
    for(int i = 1; i <= ns; i++) {
      ll char_to_num = s[i - 1] - 'a' + 1;
      ll prev_hash = hash[i - 1];
      hash[i] = ((char_to_num * pows[i - 1]) % m + prev_hash) % m;
    }
  }

  ll compute_hashing(ll i, ll j) {
    return (hash[j] - hash[i - 1] + m) % m;
  }
}
```

Algorithms

Mo

```cpp
template< class T, class T2 >
// T -> elems, T2 -> Data Structure
struct MoAlgorithm {
   vector< T > ans;
   MoAlgorithm(vector< T > &v, vector< Query > &queries,
                    void (*add)(T2&, T), void (*remove)(T2&, T), T
     (*answer)(T2&, Query)) {
    // data structure needs constructor to initialize empty
      T2 ds(v.size());
      ans.assign(queries.size(), -1);
      sort(queries.begin(), queries.end());
      int l = 0; int r = -1;
      for(Query q : queries) {
         while (l > q.l) { l--; add(ds, v[l]); }
         while (r < q.r) { r++; add(ds, v[r]); }
         while (l < q.l) { remove(ds, v[l]); l++; }
         while (r > q.r) { remove(ds, v[r]); r--; }
         ans[q.i] = answer(ds, q);
      }
   }
};
```

Tortoise Hare

```cpp
template< T >
pll TortoiseHare(T x0, T (*f)(T, T)) {
  T t = f(x0); T h = f(f(x0));
  while(t != h) {
    t = f(t); h = f(f(h));
  }
  ll mu = 0;
  t = x0;
  while(t != h) {
    t = f(t); h = f(h);
    mu += 1;
  }
  ll lam = 1; h = f(t);
  while(t != h) {
    h = f(h); lam += 1;
  }
  // mu = start, lam = period
  return {mu, lam};
}
```

Data Structures

Min Queue

```cpp
// Todas las operaciones son O(1)
template <typename T>
struct MinQueue {
    MinStack<T> in, out;
    void push(T x) { in.push(x); }
    bool empty() { return in.empty() && out.empty(); }
    int size() { return in.size() + out.size(); }
    void pop() {
        if (out.empty()) {
            while (!in.empty()) {
                out.push(in.top());
                in.pop();
            }
        }
        out.pop();
    }
    T front() {
        if (!out.empty()) return out.top();
        while (!in.empty()) {
            out.push(in.top());
            in.pop();
        }
        return out.top();
    }
    T getMin() {
        if (in.empty()) return out.getMin();
        if (out.empty()) return in.getMin();
        return min(in.getMin(), out.getMin());
    }
};
```

Min Stack

```cpp
// Todas las operaciones son O(1)
template <typename T>
struct MinStack {
    stack<pair<T, T>> S;
    void push(T x) {
        T new_min = S.empty() ? x : min(x, S.top().second);
        S.push({x, new_min});
    }
    bool empty() { return S.empty(); }
    int size() { return S.size(); }
    void pop() { S.pop(); }
    T top() { return S.top().first; }
    T getMin() { return S.top().second; }
};
```

Segment Tree

```
template <class T>
struct SegmentTree {
  int N;
  vector<T> ST;
  T (*merge)(T, T);
  void build(int n, int l, int r, vector<T> &vs) {
    if(l == r) ST[n] = vs[l];
    else {
      build(n * 2, l, (r + l) / 2, vs);
      build(n * 2 + 1, (r + l) / 2 + 1, r, vs);
      ST[n] = merge(ST[n * 2], ST[n * 2 + 1]);
    }
  }
  SegmentTree(vector<T> &vs, T (*m)(T a, T b)) {
    merge = m; N = vs.size();
    ST.resize(4 * N + 3); build(1, 0, N - 1, vs);
  }
  T query(int i, int j) { return query(0, N - 1, 1, i, j); }
  T query(int l, int r, int n, int i, int j) {
    if(l >= i && r <= j) return ST[n];
    int mid = (r + l) / 2;
    if(mid < i) return query(mid + 1, r, n*2+1, i, j);
    if(mid >= j) return query(l, mid, n*2, i, j);
    return merge(query(l, mid, n * 2, i, j),
                 query(mid + 1, r, n * 2 + 1, i, j));
  }
  void update(int pos, T val) { update(0, N - 1, 1, pos, val); }
  void update(int l, int r, int n, int pos, T val) {
    if(r < pos || pos < l) return;
    if(l == r) ST[n] = val;
    else {
      int mid = (r + l) / 2;
      update(l, mid, n * 2, pos, val);
      update(mid + 1, r, n * 2 + 1, pos, val);
      ST[n] = merge(ST[n * 2], ST[n * 2 + 1]);
    }
  }
};
```

Segment Tree Lazy

```
template<
  class T1, // answer value stored on nodes
  class T2, // lazy update value stored on nodes
  T1 merge(T1, T1),
  void pushUpd(T2&, T2&, int, int, int, int), // push update value
     from a node to another. parent -> child
  void applyUpd(T2&, T1&, int, int)           // apply the update
     value of a node to its answer value. upd -> ans
>
struct SegmentTreeLazy{
  vector<T1> ST; vector<T2> lazy; vector<bool> upd;
  int n;
  void build(int i, int l, int r, vector<T1>&values){
    if (l == r){
        ST[i] = values[l];
        return;
    }
    build(i << 1, l, (l + r) >> 1, values);
    build(i << 1 | 1, (l + r) / 2 + 1, r, values);
    ST[i] = merge(ST[i << 1], ST[i << 1 | 1]);
  }
  SegmentTreeLazy(vector<T1>&values){
    n = values.size(); ST.resize(n << 2 | 3);
    lazy.resize(n << 2 | 3); upd.resize(n << 2 | 3, false);
    build(1, 0, n - 1, values);
  }
  void push(int i, int l, int r){
    if (upd[i]){
      applyUpd(lazy[i], ST[i], l, r);
      if (l != r){
        pushUpd(lazy[i], lazy[i << 1], l, r, l, (l + r) / 2);
        pushUpd(lazy[i], lazy[i << 1 | 1], l, r, (l + r) / 2 + 1, r
     );
        upd[i << 1] = 1;
        upd[i << 1 | 1] = 1;
      }
      upd[i] = false;
      lazy[i] = T2();
    }
  }
  void update(int i, int l, int r, int a, int b, T2 &u){
    if (l >= a and r <= b){
      pushUpd(u, lazy[i], a, b, l, r);
      upd[i] = true;
    }
    push(i, l, r);
    if (l > b or r < a) return;
    if (l >= a and r <= b) return;
    update(i << 1, l, (l + r) >> 1, a, b, u);
    update(i << 1 | 1, (l + r) / 2 + 1, r, a, b, u);
    ST[i] = merge(ST[i << 1], ST[i << 1 | 1]);
  }
  void update(int a, int b, T2 u){
    if (a > b){
      update(0, b, u);
      update(a, n - 1, u);
      return ;
    }
    update(1, 0, n - 1, a, b, u);
  }
  T1 query(int i, int l, int r, int a, int b){
    push(i, l, r);
    if (a <= l and r <= b)
      return ST[i];
    int mid = (l + r) >> 1;
    if (mid < a)
      return query(i << 1 | 1, mid + 1, r, a, b);
    if (mid >= b)
      return query(i << 1, l, mid, a, b);
    return merge(query(i << 1, l, mid, a, b), query(i << 1 | 1, mid
       + 1, r, a, b));
  }
  T1 query(int a, int b){
    if (a > b) return merge(query(a, n - 1), query(0, b));
    return query(1, 0, n - 1, a, b);
  }
};
ll merge(ll a, ll b){
  return a + b;
}
void pushUpd(ll &u1, ll &u2, int l1, int r1, int l2, int r2){
  u2 = u1;
}
void applyUpd(ll &u, ll &v, int l, int r){
  v = (r - l + 1) * u;
}
```

Sparse Table

```
// Precomputacin en O(n logn), query en O(1)
template <typename T>
struct SparseTable {
  int n;
  vector<vector<T>> table;
  function<T(T, T)> merge;
  SparseTable(const vector<T> &arr, function<T(T, T)> m) : merge(m)
      {
    n = arr.size();
    int k = log2_floor(n) + 1;
    table.assign(n, vector<T>(k));
    for (int i = 0; i < n; i++)
      table[i][0] = arr[i];
    for (int j = 1; j < k; j++)
      for (int i = 0; i + (1 << j) <= n; i++)
        table[i][j] = merge(table[i][j - 1], table[i + (1 << (j -
     1))][j - 1]);
  }
  T query(int l, int r) {
    int k = log2_floor(r - l + 1);
    return merge(table[l][k], table[r - (1 << k) + 1][k]);
  }
  int log2_floor(int n) { return n ? __builtin_clzll(1) -
      __builtin_clzll(n) : -1; }
};
```

Union Find

```
struct UnionFind {
  vector<int> e;
  UnionFind(int n) { e.assign(n, -1); }
  int findSet (int x) {
    return (e[x] < 0 ? x : e[x] = findSet(e[x]));
  }
  bool sameSet (int x, int y) { return findSet(x) == findSet(y); }
  int size (int x) { return -e[findSet(x)]; }
  bool unionSet (int x, int y) {
    x = findSet(x), y = findSet(y);
    if (x == y) return 0;
    if (e[x] > e[y]) swap(x, y);
    e[x] += e[y], e[y] = x;
    return 1;
  }
};
```

Maths

Binary Pow

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

Chinese Remainder Theorem

```
ll GCRT(vector<ll> &A, vector<ll> &N) {
  int k = A.size();
  ll a = A[0], n = N[0];
  for(int i = 1; i < k; ++i) {
    vector<ll> v = egcd(n,N[i]);
    ll g = v[0], m1 = v[1], m2 = v[2];
    if((a - A[i])%g != 0) return -1;
    ll nn = N[i]/g*n; a = (a*m2%nn*(N[i]/g) + A[i]*m1%nn*(n/g))%nn;
    n = nn; if(a < 0) a += n;
  }
  return a;
}
```

Eratosthenes Sieve

```
// Corre en O(n log(log(n)) ))
struct EratosthenesSieve {
  vector<ll> primes;
  vector<bool> isPrime;
  EratosthenesSieve(ll n) {
    isPrime.resize(n + 1, true);
    isPrime[0] = isPrime[1] = false;
    for (ll i = 2; i*i <= n; i++) {
      if (isPrime[i]) {
        primes.push_back(i);
        for (int j = i*i; j <= n; j += i)
          isPrime[j] = false;
      }
    }
  }
};
```

Eulers Totient Function

```
// Corre en O(n): Recomendado para obtener solo un numero
int phi(int n) {
  int result = n;
  for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) {
      while (n % i == 0) n /= i;
        result -= result / i;
    }
  }
  if (n > 1)
    result -= result / n;
  return result;
}
// Funcin Phi de 1 a n en O(n log(log n))
struct EulerPhi {
  vector<int> phi;
  EulerPhi(int n) {
    phi.resize(n + 1);
    for (int i = 1; i <= n; i++)
      phi[i] = i;
    for (int i = 2; i <= n; i++) {
      if (phi[i] == i)
        for (int j = i; j <= n; j += i)
          phi[j] = phi[j] / i * (i - 1);
    }
  }
};
```

Extended Euclidian Algorithm

```
vector<ll> egcd(ll n, ll m) {
  ll r0 = n, r1 = m;
  ll s0 = 1, s1 = 0;
  ll t0 = 0, t1 = 1;
  while(r1 != 0) {
    ll q = r0/r1;
    ll r = r0 - q*r1; r0 = r1; r1 = r;
    ll s = s0 - q*s1; s0 = s1; s1 = s;
    ll t = t0 - q*t1; t0 = t1; t1 = t;
  }
  return {r0,s0,t0};
}
```

Fraction

```
struct Fraction {
  ll numerator, denominator;
  Fraction(ll a, ll b){
    numerator = a, denominator = b;
  }
  Fraction simplify(Fraction f){
    ll g = gcd(f.numerator, f.denominator);
    return Fraction(f.numerator/g, f.denominator/g);
  }
  Fraction add(Fraction f){
    ll l = lcm(denominator, f.denominator);
    numerator *= (l/denominator);
    numerator += f.numerator * (l/f.denominator);
    return simplify(Fraction(numerator, l));
  }
};
```

Prime Factor

```
// Corre en O(n)
vector<int> primeFactors(int n) {
  vector<int> factors;
  for (int i = 2; (i*i) <= n; i++) {
    while (n % i == 0) {
      factors.push_back(i);
      n /= i;
    }
  }
  if (n > 1) factors.push_back(n);
  return factors;
}
```

Graphs

Bellman Ford

```
void BellmanFord(int s) {
  // remember to assign INF in vector D
  D[s] = 0;
  bool flag = false;
  for(int i = 0; i < n; i++) {
    for(int a = 0; a < n; a++) {
      for(pi e : G[a]) {
        b = e.first;
        w = e.second;
        // this is to check negative cycle
        if(i == n - 1) flag = (D[b] > D[a] + w ? true : false);
        else D[b] = min(D[b], D[a] + w);
      }
    }
  }
}
```

BFS

```
void BFS(int a) {
  queue<int> Q;
  D[a] = 0;
  Q.push(a);
  while(!Q.empty()) {
    int u = Q.front();
    Q.pop();
    for(int v : G[u]) {
      if(D[v] > D[u] + 1) {
        D[v] = D[u] + 1;
        Q.push(v);
      }
    }
  }
}
```

DFS

```
void DFS(int u) {
  visited[u] = 1;
  for(int v : G[u]) {
    if(!visited[v]) {
      DFS(v);
    }
  }
}
```

Dijsktra

```
void Dijsktra(int a) {
  D[a] = 0;
  priority_queue< pii, vpii, greater<pi>> PQ;
  PQ.push(pi(0, a));
  while(!PQ.empty()) {
    int u = PQ.top().second;
    int d = PQ.top().first;
    PQ.pop();
    if(d > D[u]) continue;
    // only in case that final node exists
    if(u == f) continue
    for(pi next : G[u]) {
      int v = next.first;
      int w = next.second;
      if(D[v] > D[u] + w) {
        D[v] = D[u] + w;
        PQ.push(pi(D[v], v));
      }
    }
  }
}
```

Dinic

```
//https://github.com/PabloMessina/Competitive-Programming-Material/
      blob/master/Graphs/Dinic.cpp
struct Dinic {
  struct Edge { int to, rev; int f, c; };
  int n, t_; vector<vector<Edge>> G;
  vector<int> D;
  vector<int> q, W;
  bool bfs(int s, int t) {
    W.assign(n, 0); D.assign(n, -1); D[s] = 0;
    int f = 0, l = 0; q[l++] = s;
    while (f < l) {
      int u = q[f++];
      for (const Edge &e : G[u]) if (D[e.to] == -1 && e.f < e.c)
        D[e.to] = D[u] + 1, q[l++] = e.to;
    }
    return D[t] != -1;
  }
  int dfs(int u, int f) {
    if (u == t_) return f;
    for (int &i = W[u]; i < (int)G[u].size(); ++i) {
      Edge &e = G[u][i]; int v = e.to;
      if (e.c <= e.f || D[v] != D[u] + 1) continue;
      int df = dfs(v, min(f, e.c - e.f));
      if (df > 0) { e.f += df, G[v][e.rev].f -= df; return df; }
    }
    return 0;
  }
  Dinic(int N) : n(N), G(N), D(N), q(N) {}
  void add_edge(int u, int v, int cap) {
    G[u].push_back({v, (int)G[v].size(), 0, cap});
    G[v].push_back({u, (int)G[u].size() - 1, 0, 0}); // Use cap
      instead of 0 if bidirectional
  }
  int max_flow(int s, int t) {
    t_ = t; int ans = 0;
    while (bfs(s, t)) while (int dl = dfs(s, LLONG_MAX)) ans += dl;
    return ans;
  }
};
```

Floyd Warshall

```
void FloydWarshall() {
  for(int k = 0; k < n; k++) {
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < n; j++) {
        D[i][j] = min(D[i][j], D[i][k] + D[k][j]);
      }
    }
  }
}
```

Heavy Light Decomposition

```
template<class EST, class NODE, NODE merge(NODE, NODE)>
struct HeavyLight{
  vector<int> parent, depth, heavy, head, pos_up, pos_down;
  int n, cur_pos_up, cur_pos_down;
  EST est_up, est_down;
  int dfs(int v, vector<vector<int>> const& adj) {
    int size = 1;
    int max_c_size = 0;
    for (int c : adj[v]) if (c != parent[v]){
      parent[c] = v, depth[c] = depth[v] + 1;
      int c_size = dfs(c, adj);
      size += c_size;
      if (c_size > max_c_size) max_c_size = c_size, heavy[v] = c;
    }
    return size;
  }
  void decompose(int v, int h, vector<vector<int>> const& adj,
     vector<NODE>& a_up, vector<NODE>& a_down, vector<NODE>& values
     ) {
    head[v] = h, pos_up[v] = cur_pos_up--, pos_down[v] =
     cur_pos_down++;
    a_up[pos_up[v]] = values[v];
    a_down[pos_down[v]] = values[v];
    if (heavy[v] != -1)
      decompose(heavy[v], h, adj, a_up, a_down, values);
    for (int c : adj[v]) {
      if (c != parent[v] && c != heavy[v])
        decompose(c, c, adj, a_up, a_down, values);
    }
  }
  HeavyLight(vector<vector<int> > const& adj, vector<NODE>& values)
     {
    n = adj.size(); parent.resize(n);
    depth.resize(n); heavy.resize(n, -1);
    head.resize(n); pos_up.resize(n);
    pos_down.resize(n);
    vector<NODE> a_up(n), a_down(n);
    cur_pos_up = n - 1; cur_pos_down = 0;
    dfs(0, adj);
    decompose(0, 0, adj, a_up, a_down, values);
    est_up = EST(a_up); est_down = EST(a_down);
  }
  void update(int a, int b, NODE x){
    while(head[a] != head[b]) {
      if (depth[head[a]] > depth[head[b]]) {
        est_up.update(pos_up[a], pos_up[head[a]], x);
        est_down.update(pos_down[head[a]], pos_down[a], x);
        a = parent[head[a]];
      } else {
        est_down.update(pos_down[head[b]], pos_down[b], x);
        est_up.update(pos_up[b], pos_up[head[b]], x);
        b = parent[head[b]];
      }
    }
    if (depth[a] > depth[b]){
      est_up.update(pos_up[a], pos_up[b], x);
      est_down.update(pos_down[b], pos_down[a], x);
    } else {
      est_down.update(pos_down[a], pos_down[b], x);
      est_up.update(pos_up[b], pos_up[a], x);
    }
  }
  void update(int a, NODE x){
    est_up.update(pos_up[a], x);
    est_down.update(pos_down[a], x);
  }
  NODE query(int a, int b) {
    NODE ansL, ansR; bool hasL = 0, hasR = 0;
    while (head[a] != head[b]) {
      if (depth[head[a]] > depth[head[b]]){
        hasL ? ansL = merge(ansL, est_up.query(pos_up[a], pos_up[
     head[a]])) : ansL = est_up.query(pos_up[a], pos_up[head[a]]),
     hasL = 1;
        a = parent[head[a]];
      } else {
        hasR ? ansR = merge(est_down.query(pos_down[head[b]],
     pos_down[b]), ansR) : ansR = est_down.query(pos_down[head[b]],
      pos_down[b]), hasR = 1;
        b = parent[head[b]];
      }
    }
    if (depth[a] > depth[b])
      hasL ? ansL = merge(ansL, est_up.query(pos_up[a], pos_up[b]))
      : ansL = est_up.query(pos_up[a], pos_up[b]), hasL = 1;
    else
      hasR ? ansR = merge(est_down.query(pos_down[a], pos_down[b]),
      ansR) : ansR = est_down.query(pos_down[a], pos_down[b]), hasR
      = 1;
    return (!hasL) ? ansR : (!hasR ? ansL : merge(ansL, ansR));
  }
};
// example
HeavyLight<SegmentTreeLazy<int, int, merge, pushUpd, applyUpd>, int
     , merge> hld(G, arr);
```

Kosaraju

```
// Kosaraju, en O(V + E)
template<typename T>
struct SCC {
  vector<vector<int>> GT, G, SCC_G, SCC_GT, comp_nodes;
  vector<T> data, cdata;
  stack<int> order;
  vector<int> comp, dp;
  vector<bool> visited;
  T (*cfunc)(T, T);
  int comp_count = 0;
  void topsort(int u) {
    visited[u] = true;
    for (int v : G[u])
      if (!visited[v])
        topsort(v);
      order.push(u);
  }
  void build_component(int u) {
    visited[u] = true;
    for (int v : GT[u])
      if (!visited[v])
        build_component(v);
    comp[u] = comp_count;
    comp_nodes[comp_count].push_back(u);
  }
  void compress_graph() {
    for (int u = 0; u < G.size(); u++)
      cdata[comp[u]] = cfunc(cdata[comp[u]], data[u]);
      for (int u = 0; u < G.size(); u++)
        for (int v : G[u])
          if (comp[u] != comp[v]) {
            SCC_G[comp[u]].push_back(comp[v]);
            SCC_GT[comp[v]].push_back(comp[u]);
          }
  }
  T process(int cmp, T (*func)(T a, T b), T (*merge)(T a, T b)) {
    if (dp[cmp]) return dp[cmp];
    dp[cmp] = cdata[cmp];
    for (int u : SCC_G[cmp])
      dp[cmp] = merge(dp[cmp], func(process(u, func, merge), cdata[
     cmp]));
    return dp[cmp];
  }
  SCC(vector<vector<int>> &G, vector<T> &data, T (*cfunc)(T a, T b)
     , T comp_identity, T dp_identity): cfunc(cfunc), G(G), data(
     data) {
    GT.resize(G.size()); comp_nodes.resize(G.size());
    visited.assign(G.size(), 0);
    cdata.assign(G.size(), comp_identity);
    comp.assign(G.size(), 0);
    SCC_G.resize(G.size()); SCC_GT.resize(G.size());
    dp.assign(G.size(), dp_identity);
    for (int u = 0; u < G.size(); u++)
    for (int v : G[u])
      GT[v].push_back(u);
      for (int u = 0; u < G.size(); u++)
        if (!visited[u])
          topsort(u);
    visited.assign(G.size(), 0);
    while (!order.empty()) {
      int u = order.top();
      order.pop();
      if (visited[u]) continue;
      build_component(u);
      comp_count++;
    }
    compress_graph();
  }
};
```

Kruskal

```
struct Edge {
  int a; int b; int w;
  Edge(int a_, int b_, int w_) : a(a_), b(b_), w(w_) {}
}
bool c_edge(Edge &a, Edge &b) {
  return a.w < b.w;
}
int Kruskal() {
  int n = G.size();
  DSU sets(n);
  vector< Edge > edges;
  for(int i = 0; i < n; i++) {
    for(pi eg : G[i]) {
      // node i to node eg.first with cost eg.second
      Edge e(i, eg.first, eg.second);
      edges.push_back(e);
    }
  }
  sort(edges.begin(), edges.end(), c_edge);
  int min_cost = 0;
  for(Edge e : Edges) {
    if(sets.find(e.a, e.b) != true) {
      tree.push_back(Edge(e.a, e.b, e.w));
      min_cost += e.w;
      sets.union(e.a, e.b);
    }
  }
  return min_cost;
}
```

LCA

```
struct LCA {
  vector<vector<int>> T, parent;
  vector<int> depth;
  int LOGN, V;
  // Si da WA, probablemente el logn es muy chico
  LCA(vector<vector<int>> &T, int logn = 20) {
    this->LOGN = logn;
    this->T = T;
    T.assign(T.size()+1, vector<int>());
    parent.assign(T.size()+1, vector<int>(LOGN, 0));
    depth.assign(T.size()+1, 0);
    dfs();
  }
  void dfs(int u = 0, int p = -1) {
    for (int v : T[u]) {
      if (p != v) {
        depth[v] = depth[u] + 1;
        parent[v][0] = u;
        for (int j = 1; j < LOGN; j++)
          parent[v][j] = parent[parent[v][j-1]][j-1];
          dfs(v, u);
      }
    }
  }
  int query(int u, int v) {
    if (depth[u] < depth[v]) swap(u, v);
    int k = depth[u]-depth[v];
    for (int j = LOGN - 1; j >= 0; j--)
      if (k & (1 << j))
        u = parent[u][j];
    if (u == v)
      return u;
    for (int j = LOGN - 1; j >= 0; j--) {
      if (parent[u][j] != parent[v][j]) {
        u = parent[u][j];
        v = parent[v][j];
      }
    }
    return parent[u][0];
  }
};
```

Coef Line

```
template< T >
struct CoefLine {
  T A; T B; T C;
  double EPS;
  CoefLine(double eps) : EPS(eps) {}
  // Line of Segment Integer
  // here we asume that P and Q are only points
  void LSI(Point2D< T > P, Point2D< T > Q){
    // Ax + By + C
    A = P.y - Q.y; B = Q.x - P.x;
    C = -1 * A * P.x - B * P.y;
    T gcdABC = gcd(A, gcd(B, C));
    A /= gcdABC; B /= gcdABC; C /= gcdABC;
    if(A < 0 || (A == 0 && B < 0)) {
      A *= -1; B *= -1; C *= -1;
    }
    return L;
  }
  T det(T a, T b, T c, T d) { return a * d - b * c; }
  // Line of Segment Real
  void LSR(Point2D< T > P, Point2D< T > Q, T eps){
    // Ax + By + C
    A = P.y - Q.y;
    B = Q.x - Px;
    C = -1 * A * P.x - B * P.y;
    T z = sqrt(L.A * L.A + L.B * L.B);
    A /= z; B /= z; C /= z;
    if(A < -1 * eps || (abs(A) < eps && B < -1 * eps)) {
      A *= -1; B *= -1; C *= -1;
    }
    return L;
  }
  bool intersect(CoefLine l, Point2D &res) {
    double z = det(a, b, l.a, l.b);
    if(abs(z) < EPS) { return false; }
    res.x = -det(c, b, l.c, l.b) / z;
    res.y = -det(a, c, l.a, l.c) / z;
    return true;
  }
  bool parallel(CoefLine l) { return abs(det(a, b, l.a, l.b)) < EPS
     ; }
  bool equivalent(CoefLine l) {
    return abs(det(a, b, l.a, l.b)) < EPS &&
           abs(det(a, c, l.a, l.c)) < EPS &&
           abs(det(b, c, l.b, l.c)) < EPS;
  }
};
```

Point2D

```
template< T >
struct Point2D {
  T x, y;
  Point2D() {};
  Point2D(T x_, T y_) : x(x_), y(y_) {}
  Point2D< T >& operator=(const Point2D< T > &t) {
    x = t.x; y = t.y;
    return *this;
  }
  Point2D< T >& operator+= (const Point2D< T > &t) {
    x += t.x; y += t.y;
    return *this;
  }
  Point2D< T >& operator-= (const Point2D< T > &t) {
    x -= t.x; y -= t.y;
    return *this;
  }
  Point2D< T >& operator*= (const Point2D< T > &t) {
    x *= t; y *= t;
    return *this;
  }
  Point2D< T >& operator/= (const Point2D< T > &t) {
    x /= t; y /= t;
    return *this;
  }
  Point2D< T > operator+(const Point2D< T > &t) const {
    return Point2D(*this) += t;
  }
  Point2D< T > operator-(const Point2D< T > &t) const {
    return Point2D(*this) -= t;
  }
  Point2D< T > operator*(T t) const {
    return Point2D(*this) *= t;
  }
  Point2D< T > operator/(T t) const {
    return Point2D(*this) /= t;
  }
  T dot(Point2D< T >& b) { return x * b.x + a.y * b.y; }
  T cross(Point2D< T >& b) { return x * b.y - a.y * a.x; }
  T norm() { return dot(*this); }
  double abs() { return sqrt(norm()); }
  double proj(Point2D< T >& b) { return dot(b) / b.abs(); }
  double angle(Point2D< T >& b) { return acos(dot(b) / abs() / b.
      abs()); }
};
template< T >
Point2D< T > operator*(T a, Point2D< T > b) { return b * a; }
```

Point3D

```
template< T >
struct Point3D {
  T x, y, z;
  Point3D() {};
  Point3D(T x_, T y_, T z_) : x(x_), y(y_), z(z_) {}
  Point3D< T >& operator=(const Point3D< T > &t) {
    x = t.x; y = t.y; z = z.y;
    return *this;
  }
  Point3D< T >& operator+= (const Point3D< T > &t) {
    x += t.x; y += t.y; z += t.z;
    return *this;
  }
  Point3D< T >& operator-= (const Point3D< T > &t) {
    x -= t.x; y -= t.y; z -= t.z;
    return *this;
  }
  Point3D< T >& operator*= (const Point3D< T > &t) {
    x *= t; y *= t; z *= t;
    return *this;
  }
  Point3D< T >& operator/= (const Point3D< T > &t) {
    x /= t; y /= t; z /= t;
    return *this;
  }
  Point3D< T > operator+(const Point3D< T > &t) const {
    return Point3D(*this) += t;
  }
  Point3D< T > operator-(const Point3D< T > &t) const {
    return Point3D(*this) -= t;
  }
  Point3D< T > operator*(T t) const {
    return Point3D(*this) *= t;
  }
  Point3D< T > operator/(T t) const {
    return Point3D(*this) /= t;
  }
  T dot(Point3D< T >& b) { return x * b.x + y * b.y + z * b.z; }
  Point3D< T > cross(Point3D< T > & b) {
    return Point3D(y * b.z - z * b.y,
                   z * b.x - x * b.z,
                   x * b.y - y * b.x);
  }
  T norm() { return dot(*this); }
  double abs() { return sqrt(norm()); }
  double proj(Point3D< T >& b) { return dot(b) / b.abs(); }
  double angle(Point3D< T >& b) { return acos(dot(b) / abs() / b.
      abs()); }
};
template< T >
Point3D< T > operator*(T a, Point3D< T > b) { return b * a; }
template< T >
T triple(Point3D< T > a, Point3D< T > b, Point3D< T > c) {
  return dot(a, cross(b, c));
}
```

Polygon Area

```
// Recuerda que si quieres sumar varias areas factoriza 1/2
// Para numeros enteros, solo hay que cambiar el tipo de dato
double area(const vector<Point>& fig) {
    double res = 0;
    for (unsigned i = 0; i < fig.size(); i++) {
        Point p = i ? fig[i - 1] : fig.back();
        Point q = fig[i];
        res += (p.x - q.x) * (p.y + q.y);
    }
    return fabs(res) / 2;
}
```

Segment

```
template< T >
struct Segment {
  // Segment (P, Q)
  Point2D< T > P;
  Point2D< T > Q;
  Segment(Point2D< T > P_, Point2D< T > Q_) {
    P = P_;
    Q = Q_;
  }
  bool check(T a, T b, T c, T d) {
    if(a > b) swap(a, b);
    if(c > d) swap(c, d);
    return max(a, c) <= min(b, d);
  }
  int sign(T x) { return x > 0 ? 1 : (x < 0 : -1 : 0); }
  bool intersect(Segment< T > S) {
    if((P - S.P).cross(S.Q - S.P) == 0 && (Q - S.P).cross(S.Q - S.P
     ) == 0) {
      return check(P.x, Q.x, S.P.x, S.Q.x) && check(P.y, Q.y, S.P.y
     , S.Q.y);
    }
    return sign((Q - P).cross(S.P - P)) != sign((Q - P).cross(S.Q -
      P)) &&
    sign((S.Q - S.P).cross(P - S.P)) != sign((S.Q - S.P).cross(Q -
     S.P));
  }
};
```

Vec Line

```
template< T >
struct Line {
  Point2D< T > a;
  Point2D< T > d;
  Line() {}
  Line(Point2D< T > a_, Point2D< T > d_) {
    a = a_;
    d = d_;
  }
  Line(Point2D< T > p1, Point2D< T > p2) {
    // TO DO
  }
  Point2D< T > intersect(Line< T > l) {
    Point2D a2a1 = l.a - a;
    return a + a2a1.cross(l.d) / d.cross(l.d) * d;
  }
};
```

Vec Plane

```
template< T >
struct Plane {
  Point3D< T > a;
  Point3D< T > n;
  Plane() {}
  Plane(Point3D< T > a_, Point3D< T > d_) : a(a_), d(d_) {}
  Point3D< T > insersect(Plane< T > p1, Plane< T > p2) {
    Point3D< T > x(n.x, p1.n.x, p2.n.x);
    Point3D< T > x(n.y, p1.n.y, p2.n.y);
    Point3D< T > x(n.z, p1.n.z, p2.n.z);
    Point3D< T > d(a.dot(n), p1.a.dot(p1.n), p2.a.dot(p2.n));
    return Point3D(triple(d, y, z),
                   triple(x, d, z),
                   triple(x, y, d)) / triple(n, p1.n, p2.n);
  }
};
```