USM Handbook

Maintainers:
- Gabriel Carmona (MrYhatoh)
- Carlos Lagos (CharlesLakes)
- Sebastián Torrealba (storrealbac)

Contributors:
- Javier Oliva (JOliva)
- Marcelo Lemus (MarceloL)

General

Template

Description: Just the starting template code

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

using namespace std;

using ll = long long;
using ld = long double;

const ll mod = 1e9 + 7;
const ll inf = 1e12;
const ld pi = acos(-1);

int main() {
  ios::sync_with_stdio(0); cin.tie(0);
  cout << fixed << setprecision(9);

  return 0;
}
```

Troubleshoot

Pre-submit:

- Write a few simple test cases if sample is not enough.
- Are time limits close? If so, generate max cases.
- Is the memory usage fine?
- Could anything overflow?
- Make sure to submit the right file.

Wrong answer:

- Print your solution! Print debug output, as well.
- Are you clearing all data structures between test cases?
- Can your algorithm handle the whole range of input?
- Read the full problem statement again.
- Do you handle all corner cases correctly?
- Have you understood the problem correctly?
- Any uninitialized variables?
- Any overflows?
- Confusing N and M, i and j, etc.?
- Are you sure your algorithm works?
- What special cases have you not thought of?
- Are you sure the STL functions you use work as you think?
- Add some assertions, maybe resubmit.
- Create some testcases to run your algorithm on.
- Go through the algorithm for a simple case.
- Go through this list again.
- Explain your algorithm to a teammate.
- Ask the teammate to look at your code.
- Go for a small walk, e.g. to the toilet.
- Is your output format correct? (including whitespace)
- Are you using modular inverse?
- Rewrite your solution from the start or let a teammate do it.

Runtime error:

- Have you tested all corner cases locally?
- Any uninitialized variables?
- Are you reading or writing outside the range of any vector?
- Any assertions that might fail?
- Any possible division by 0? (mod 0 for example)
- Any possible infinite recursion?
- Invalidated pointers or iterators?
- Are you using too much memory?
- Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:

- Do you have any possible infinite loops?
- What is the complexity of your algorithm?
- Are you copying a lot of unnecessary data? (References)
- How big is the input and output? (consider scanf)
- Avoid vector, map. (use arrays/unordered_map)
- What do your teammates think about your algorithm?

Memory limit exceeded:

- What is the max amount of memory your algorithm should need?
- Are you clearing all data structures between test cases?

Strings

Rolling Hashing

You must provide a description for each template code!

```
template<class T>
struct rolling_hashing {
  int base, mod;
  vector<int> p, H;
  int n;
  rolling_hashing(const T &s, int b, int m): base(b), mod(m), n(s.size()) {
    p.assign(n+1, 1);
    H.assign(n+1, 0);
    for (int i = 0; i < n; ++i) {
      H[i+1] = (H[i] * base + s[i]) % mod;
      p[i+1] = (p[i] * base) % mod;
    }
  }
  int get(int l, int r) {
    int res = (H[r+1] - H[l]*p[r-l+1]) % mod;
    if (res < 0) res += mod;
    return res;
  }
};
```

K Rolling Hashing

You must provide a description for each template code!

```
template<class T>
struct rolling_hashing {
  vector<int> base, mod; int n, k;
  vector<vector<int>> p, H;
  rolling_hashing(T s, vector<int> b, vector<int> m): base(b), mod(m),
n(s.size()), k(b.size()) {
    p.resize(k); H.resize(k);
    for (int j = 0; j < k; j++) {
      p[j].assign(n + 1, 1);
      H[j].assign(n + 1, 0);
      for (int i = 0; i < n; i++) {
        H[j][i + 1] = (H[j][i] * b[j] + s[i]) % mod[j];
        p[j][i + 1] = (p[j][i] * b[j]) % mod[j];
      }
    }
  }
  vector<int> get(int l, int r) {
    vector<int> res(k);
    for (int j = 0; j < k; j++) {
      res[j] = H[j][r + 1] - H[j][l] * p[j][r - l + 1];
      res[j] %= mod[j];
      res[j] = (res[j] + mod[j]) % mod[j];
    }
    return res;
  }
};
```

KMP

Description: Find occurrences of a pattern within given text, runs in $O(n + m)$, where n and m are the lengths of the text and pattern respectively.
Status: Tested on CSES

```
template<class T> struct KMP {
  T pattern; vector<int> lps;
  KMP(T &pat): pattern(pat) {
    lps.resize(pat.size(), 0);
    int len = 0, i = 1;
    while (i < pattern.size()) {
      if (pattern[i] == pattern[len])
        lps[i++] = ++len;
      else {
        if (len != 0) len = lps[len - 1];
        else lps[i++] = 0;
      }
    }
  }
  vector<int> search(T &text) {
    vector<int> matches;
    int i = 0, j = 0;
    while (i < text.size()) {
      if (pattern[j] == text[i]) {
        i++, j++;
        if (j == pattern.size()) {
          matches.push_back(i - j);
          j = lps[j - 1];
        }
      } else {
        if (j != 0) j = lps[j - 1];
        else i++;
      }
    }
    return matches;
  }
};
```

Z

You must provide a description for each template code!

```
struct Z {
  int n, m;
  vector<int> z;
  Z(string s) {
    n = s.size();
    z.assign(n, 0);
    int l = 0, r = 0;
    for (int i = 1; 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;
    }
  }
  Z(string p, string t) {
    string s = p + "#" + t;
    n = p.size();
    m = t.size();
    z.assign(n + m + 1, 0);
    int l = 0, r = 0;
    for (int i = 1; i < n + m + 1; i++) {
      if (i <= r)
        z[i] = min(r - i + 1, z[i - l]);
      while (i + z[i] < n + m + 1 && s[z[i]] == s[i + z[i]])
        ++z[i];
      if (i + z[i] - 1 > r)
        l = i, r = i + z[i] - 1;
    }
  }
  void p_in_t(vector<int>& ans) {
    for (int i = n + 1; i < n + m + 1; i++) {
      if (z[i] == n)
        ans.push_back(i - n - 1);
    }
  }
};
```

Manacher

Description: Find palindromes centered at i in $O(n)$
Status: Tested on CSES

```
template<class T>
struct manacher {
  vector<int> odd, even;
  T s; int n;
  manacher(T &s): s(s), n(s.size()) {
    odd.resize(n);
    even.resize(n);
    for (int i = 0, l = 0, r = -1; i < n; i++) {
      int k = (i > r) ? 1 : min(odd[l+r-i], r-i+1);
      while (0 <= i-k and i+k < n and s[i-k] == s[i+k]) k++;
      odd[i] = k--;
      if (i+k > r) l = i-k, r = i+k;
    }
    for (int i = 0, l = 0, r = -1; i < n; i++) {
      int k = (i > r) ? 0 : min(even[l + r - i + 1], r - i + 1);
      while (0 <= i - k - 1 and i + k < n && s[i - k - 1] == s[i + k]) k++;
      even[i] = k--;
      if (i + k > r) l = i - k - 1, r = i + k;
    }
  }
  // Returns the longest palindrome centered at i
  pair<int, int> get(int i) {
    int o = 2 * odd[i] - 1;   // Normally centered (Is odd size)
    int e = 2 * even[i];      // Centered to the right
    if (o >= e)
      return {i - odd[i] + 1, i + odd[i] - 1};
    return {i - even[i], i + even[i] - 1};
  }
};
```

Trie

You must provide a description for each template code!

```
struct trie {
  vector <vector <ll>> tree;
  trie() {
    tree.push_back(vector < ll > (26, -1));
  };
  void insert(string &s, ll i = 0, ll u = 0) {
    if (s.size() == i) return;
    char c = s[i];
    if (tree[u][c - 'a'] != -1)
      insert(s, i + 1, tree[u][c - 'a']);
    else {
      ll pos = tree.size();
      tree.push_back(vector < ll > (26, -1));
      tree[u][c - 'a'] = pos;
      insert(s, i + 1, tree[u][c - 'a']);
    }
  }
};
```

Suffix Array

You must provide a description for each template code!

```
struct SA {
  int n;
  vector<int> C, R, R_, sa, sa_, lcp;
  inline int gr(int i) { return i < n ? R[i] : 0; }
  void csort(int maxv, int k) {
    C.assign(maxv + 1, 0);
    for (int i = 0; i < n; i++) C[gr(i + k)]++;
    for (int i = 1; i < maxv + 1; i++) C[i] += C[i - 1];
    for (int i = n - 1; i >= 0; i--) sa_[--C[gr(sa[i] + k)]] = sa[i];
    sa.swap(sa_);
  }
  void getSA(vector<int>& s) {
    R = R_ = sa = sa_ = vector<int>(n);
    for (ll i = 0; i < n; i++) sa[i] = i;
    sort(sa.begin(), sa.end(), [&s](int i, int j) { return s[i] < s[j]; });
    int r = R[sa[0]] = 1;
    for (ll i = 1; i < n; i++) R[sa[i]] = (s[sa[i]] != s[sa[i - 1]]) ? ++r :
r;
    for (int h = 1; h < n && r < n; h <<= 1) {
      csort(r, h);
      csort(r, 0);
      r = R_[sa[0]] = 1;
      for (int i = 1; i < n; i++) {
        if (R[sa[i]] != R[sa[i - 1]] || gr(sa[i] + h) != gr(sa[i - 1] + h))
r++;
        R_[sa[i]] = r;
      }
      R.swap(R_);
    }
  }
  void getLCP(vector<int> &s) {
    lcp.assign(n, 0);
    int k = 0;
    for (ll i = 0; i < n; i++) {
      int r = R[i] - 1;
      if (r == n - 1) {
        k = 0;
        continue;
      }
      int j = sa[r + 1];
      while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
      lcp[r] = k;
      if (k) k--;
    }
  }
  SA(vector<int> &s) {
    n = s.size();
    getSA(s);
    getLCP(s);
  }
};
```

Suffix Automaton

You must provide a description for each template code!

```
struct SuffixAutomaton {
  struct state {
    int len, link;
    int next[26];
    state(int _len = 0, int _link = -1) : len(_len), link(_link) {
      memset(next, -1, sizeof(next));
    }
  };
  vector<state> st;
  int last;
  SuffixAutomaton() {}
  SuffixAutomaton(const string &s) { init(s); }
  inline int State(int len = 0, int link = -1) {
    st.emplace_back(len, link);
    return st.size() - 1;
  }
  void init(const string &s) {
    st.reserve(2 * s.size());
    last = State();
    for (char c : s)
      extend(c);
  }
  void extend(char _c) {
    int c = _c - 'a', cur = State(st[last].len + 1), P = last;
    while ((P != -1) && (st[P].next[c] == -1)) {
      st[P].next[c] = cur;
      P = st[P].link;
    }
    if (P == -1)
      st[cur].link = 0;
    else {
      int Q = st[P].next[c];
      if (st[P].len + 1 == st[Q].len)
        st[cur].link = Q;
      else {
        int C = State(st[P].len + 1, st[Q].link);
        copy(st[Q].next, st[Q].next + 26, st[C].next);
        while ((P != -1) && (st[P].next[c] == Q)) {
          st[P].next[c] = C;
          P = st[P].link;
        }
        st[Q].link = st[cur].link = C;
      }
    }
    last = cur;
  }
};
```

Algorithms

Tortoise Hare

You must provide a description for each template code!

```
template<typename T, T f(T)>
pair<ll, ll> tortoise_hare(T x0) {
  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++;
  ll lam = 1; h = f(t);
  while(t != h)h = f(h), lam += 1;
  // mu = start, lam = period
  return {mu, lam};
}
```

Data Structures

Segment Tree

You must provide a description for each template code!

```
template<class T, T m_(T, T)> struct segment_tree{
  int n; vector<T> ST;
  segment_tree(){}
  segment_tree(vector<T> &a){
    n = a.size(); ST.resize(n << 1);
    for (int i=n;i<(n<<1);i++)ST[i]=a[i-n];
    for (int i=n-1;i>0;i--)ST[i]=m_(ST[i<<1],ST[i<<1|1]);
  }
  void update(int pos, T val){ // replace with val
    ST[pos += n] = val;
    for (pos >>= 1; pos > 0; pos >>= 1)
      ST[pos] = m_(ST[pos<<1], ST[pos<<1|1]);
  }
  T query(int l, int r){ // [l, r]
    T ansL, ansR; bool hasL = 0, hasR = 0;
    for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
      if (l & 1)
        ansL=(hasL?m_(ansL,ST[l++]):ST[l++]),hasL=1;
      if (r & 1)
        ansR=(hasR?m_(ST[--r],ansR):ST[--r]),hasR=1;
    }
    if (!hasL) return ansR; if (!hasR) return ansL;
    return m_(ansL, ansR);
  }
};
```

Bit 2D

You must provide a description for each template code!

```
struct fenwick_tree_2d {
  int N, M;
  vector < vector < int >> BIT;
  fenwick_tree_2d(int N, int M): N(N), M(M) {
    BIT.assign(N + 1, vector < int > (M + 1, 0));
  }
  void update(int x, int y, int v) {
    for (int i = x; i <= N; i += (i & -i))
      for (int j = y; j <= M; j += (j & -j))
        BIT[i][j] += v;
  }
  int sum(int x, int y) {
    int s = 0;
    for (int i = x; i > 0; i -= (i & -i))
      for (int j = y; j > 0; j -= (j & -j))
        s += BIT[i][j];
    return s;
  }
  int query(int x1, int y1, int x2, int y2) {
    return sum(x2, y2) - sum(x2, y1 - 1) - sum(x1 - 1, y2) + sum(x1 - 1, y1
- 1);
  }
};
```

BIT

You must provide a description for each template code!

```
struct fenwick_tree {
  vector <int> bit; int n;
  fenwick_tree(int n): n(n) { bit.assign(n, 0); }
  fenwick_tree(vector <int> &a): fenwick_tree(a.size()) {
    for (size_t i = 0; i < a.size(); i++)
      add(i, a[i]);
  }
  int sum(int r) {
    int ret = 0;
    for (; r >= 0; r = (r & (r + 1)) - 1)
      ret += bit[r];
    return ret;
  }
  int sum(int l, int r) {
    return sum(r) - sum(l - 1);
  }
  void add(int idx, int delta) {
    for (; idx < n; idx = idx | (idx + 1))
      bit[idx] += delta;
  }
};
```

Implicit Treap

You must provide a description for each template code!

```
struct implicit_treap {
  static mt19937_64 MT;
  struct node { node *left, *right;
    int sz, priority, value, sum_value, lazy_sum, lazy_flip;
    node(ll v = 0) {
      left = right = NULL; priority = MT(); lazy_flip = false;
      sz = 1; lazy_sum = 0; sum_value = value = v; } };
  ll value(node* T) { return T ? T->value : 0; }
  ll sum_value(node* T) { return T ? T->sum_value : 0; }
  int sz(node* T) { return T ? T->sz : 0; }
  int key(node* T) { return sz(T->left); }
  void update(node* T) {
    T->sum_value =  T->value + sum_value(T->left) + sum_value(T->right);
    T->sz = 1 + sz(T->left) + sz(T->right);
  }
  void sum_push(node* T) {
    if(T->lazy_sum) {
      T->value += T->lazy_sum; T->sum_value += T->sz*T->lazy_sum;
      if(T->left) T->left->lazy_sum += T->lazy_sum;
      if(T->right) T->right->lazy_sum += T->lazy_sum;
    } T->lazy_sum = 0;
  }
  void flip_push(node* T) {
    if(T->lazy_flip) {
      swap(T->left, T->right);
      if(T->left) T->left->lazy_flip = !T->left->lazy_flip;
      if(T->right) T->right->lazy_flip = !T->right->lazy_flip;
    } T->lazy_flip = false;
  } node *root;
  void push(node* T) { sum_push(T);flip_push(T); }
  void merge(node* &T, node* T1, node* T2) {
    if(T1 == NULL) {T = T2; return;} if(T2 == NULL) { T = T1; return; }
    push(T1); push(T2);
    if(T1->priority > T2->priority) merge(T1->right, T1->right, T2), T = T1;
    else merge(T2->left, T1, T2->left), T = T2; return update(T);
  }
  void merge(node* &T,node* T1,node* T2,node* T3) {merge(T, T1, T2);
merge(T, T, T3);}
  void split(node* T, int k, node* &T1, node* &T2) {
    if(T == NULL) { T1 = T2 = NULL; return; } push(T);
    if(key(T) <= k) { split(T->right, k - (key(T)+1), T->right, T2); T1 = T;
    } else split(T->left, k, T1, T->left), T2 = T; return update(T);
  }
  void split(node* T, int i, int j, node* &T1, node* &T2, node* &T3) {
    split(T, i-1, T1, T2);  split(T2, j-i, T2, T3); }
  void set(node* T, int k, ll v) {
    push(T); if(key(T) == k) T->value = v;
    else if(k < key(T)) set(T->left, k, v);
    else set(T->right, k - (key(T)+1), v);
    return update(T); }
  node* find(node* T, int k) {
    push(T); if(key(T) == k) return T;
    if(k < key(T)) return find(T->left, k);
    return find(T->right, k - (key(T)+1)); }
  implicit_treap() { root = NULL; }
  implicit_treap(ll x) { root = new node(x); }
  int size() { return sz(root); }
  implicit_treap &merge(implicit_treap &O){ merge(root, root, O.root);
return *this;}
  implicit_treap split(int k) {
    implicit_treap ans; split(root, k, root, ans.root); return ans;
  }
  void erase(int i, int j){
    node *T1, *T2, *T3; split(root, i, j, T1, T2, T3); merge(root, T1, T3);
  }
  void erase(int k) { return erase(k, k); }
  void set(int k, ll v) { set(root, k, v); }
  ll operator[](int k) { return find(root, k)->value; }
  ll query(int i, int j) {
    node *T1, *T2, *T3; split(root, i, j, T1, T2, T3);
    ll ans = sum_value(T2); merge(root, T1, T2, T3);
    return ans;
  }
  void update(int i, int j, ll x) {
    node *T1, *T2, *T3; split(root, i, j, T1, T2, T3);
    T2->lazy_sum += x; merge(root, T1, T2, T3);
  }
  void flip(int i, int j) {
    node *T1, *T2, *T3; split(root, i, j, T1, T2, T3);
    T2->lazy_flip = !T2->lazy_flip; merge(root, T1, T2, T3);
  }
  void insert(int i, ll x) {
    node* T; split(root, i-1, root, T); merge(root, root, new node(x), T); }
  void push_back(ll x) { merge(root, root, new node(x)); }
  void push_front(ll x) { merge(root, new node(x), root); }
};
mt19937_64
implicit_treap::MT(chrono::system_clock::now().time_since_epoch().count());
```

LCT

You must provide a description for each template code!

```cpp
struct Node { // Splay tree. Root's pp contains tree's parent.
  Node *p = 0, *pp = 0, *c[2];
  bool flip = 0;
  Node() { c[0] = c[1] = 0; fix(); }
  void fix() {
    if (c[0]) c[0]->p = this;
    if (c[1]) c[1]->p = this;
    // (+ update sum of subtree elements etc. if wanted)
  }
  void pushFlip() {
    if (!flip) return;
    flip = 0; swap(c[0], c[1]);
    if (c[0]) c[0]->flip ^= 1;
    if (c[1]) c[1]->flip ^= 1;
  }
  int up() { return p ? p->c[1] == this : -1; }
  void rot(int i, int b) {
    int h = i ^ b;
    Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
    if ((y->p = p)) p->c[up()] = y;
    c[i] = z->c[i ^ 1];
    if (b < 2) {
      x->c[h] = y->c[h ^ 1];
      z->c[h ^ 1] = b ? x : this;
    }
    y->c[i ^ 1] = b ? this : x;
    fix(); x->fix(); y->fix();
    if (p) p->fix(); swap(pp, y->pp);
  }
  void splay() { /// Splay this up to the root. Always finishes without flip
set.
    for (pushFlip(); p;) {
      if (p->p) p->p->pushFlip();
      p->pushFlip(); pushFlip();
      int c1 = up(), c2 = p->up();
      if (c2 == -1) p->rot(c1, 2);
      else p->p->rot(c2, c1 != c2);
    }
  }
  Node *first() { /// Return the min element of the subtree rooted at this,
splayed to the top.
    pushFlip();
    return c[0] ? c[0]->first() : (splay(), this);
  }
};
struct link_cut {
  vector<Node> node;
  link_cut(int N) : node(N) {}
  void link(int u, int v) { // add an edge (u, v)
    makeRoot(&node[u]);
    node[u].pp = &node[v];
  }
  void cut(int u, int v) { // remove an edge (u, v)
    Node *x = &node[u], *top = &node[v];
    makeRoot(top);
    x->splay();
    assert(top == (x->pp ?: x->c[0]));
    if (x->pp) x->pp = 0;
    else {
      x->c[0] = top->p = 0;
      x->fix();
    }
  }
  bool connected(int u, int v) {
    Node *nu = access(&node[u])->first();
    return nu == access(&node[v])->first();
  }
  void makeRoot(Node *u) {
    access(u); u->splay();
    if (u->c[0]) {
      u->c[0]->p = 0; u->c[0]->flip ^= 1;
      u->c[0]->pp = u; u->c[0] = 0;
      u->fix();
    }
  }
  Node *access(Node *u) {
    u->splay();
    while (Node *pp = u->pp) {
      pp->splay(); u->pp = 0;
      if (pp->c[1]) {
        pp->c[1]->p = 0;
        pp->c[1]->pp = pp;
      }
      pp->c[1] = u;
      pp->fix(); u = pp;
    }
    return u;
  }
};
```

Min Queue

You must provide a description for each template code!

```cpp
// Todas las operaciones son O(1)
template <typename T>
struct min_queue {
    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

You must provide a description for each template code!

```cpp
// Todas las operaciones son O(1)
template <typename T>
struct min_stack {
  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; }
};
```

Ordered Set

You must provide a description for each template code!

```cpp
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

typedef tree<int, null_type, less<int>, rb_tree_tag,
             tree_order_statistics_node_update>
    ordered_set;

ordered_set p;

p.insert(5); p.insert(2); p.insert(6); p.insert(4); // O(log n)
// value at 3rd index in sorted array. O(log n). Output: 6
cout << "Value at 3rd index: " << *p.find_by_order(3) << endl;

// index of number 6. O(log n). Output: 3
cout << "Index of number 6: " << p.order_of_key(6) << endl;

// number 7 not in the set but it will show the index
// number if it was there in sorted array. Output: 4
cout << "Index of number 7:" << p.order_of_key(7) << endl;

// number of elements in the range [3, 10)
cout << p.order_of_key(10) - p.order_of_key(3) << endl;
```

Persistent Segment Tree

You must provide a description for each template code!

```
template<class T, T _m(T, T)>
struct persistent_segment_tree {
  vector<T> ST;
  vector<int> L, R;
  int n, rt;
  persistent_segment_tree(int n): ST(1, T()), L(1, 0), R(1, 0), n(n), rt(0)
{}
  int new_node(T v, int l = 0, int r = 0) {
    int ks = ST.size();
    ST.push_back(v); L.push_back(l); R.push_back(r);
    return ks;
  }
  int update(int k, int l, int r, int p, T v) {
    int ks = new_node(ST[k], L[k], R[k]);
    if (l == r) {
      ST[ks] = v; return ks;
    }
    int m = (l + r) / 2, ps;
    if (p <= m) {
      ps = update(L[ks], l, m, p, v);
      L[ks] = ps;
    } else {
      ps = update(R[ks], m + 1, r, p, v);
      R[ks] = ps;
    }
    ST[ks] = _m(ST[L[ks]], ST[R[ks]]);
    return ks;
  }
  T query(int k, int l, int r, int a, int b) {
    if (l >= a and r <= b)
      return ST[k];
    int m = (l + r) / 2;
    if (b <= m)
      return query(L[k], l, m, a, b);
    if (a > m)
      return query(R[k], m + 1, r, a, b);
    return _m(query(L[k], l, m, a, b), query(R[k], m + 1, r, a, b));
  }
  int update(int k, int p, T v) {
    return rt = update(k, 0, n - 1, p, v);
  }
  int update(int p, T v) {
    return update(rt, p, v);
  }
  T query(int k, int a, int b) {
    return query(k, 0, n - 1, a, b);
  }
};
```

Query Tree

You must provide a description for each template code!

```
struct query{
  int v,u;
  bool status;
  query(int _v,int _u) : v(_v),u(_u) {};
};

struct query_tree{
  vector<vector<query>> tree;
  int size;
  // rollback structure
  UnionFindRB uf;
  query_tree(int _size,int n) : size(_size) {uf = UnionFindRB(n);
tree.resize(4*_size + 4);}
  void addTree(int v,int l,int r,int ul,int ur, query& q){
    if(ul > ur) return;
    if(l == ul && ur == r){tree[v].push_back(q); return; }
    int mid = (l + r)/2;
    addTree(2*v,l,mid,ul,min(ur,mid),q);
    addTree(2*v + 1,mid + 1,r,max(ul,mid + 1),ur,q);
  }
  void add(query q,int l,int r){addTree(1,0,size - 1,l,r,q);}
  void dfs(int v,int l,int r,vector<int> &ans){
    // change in data structure
    for(query &q: tree[v]) q.status = uf.unionSet(q.v,q.u);
    if(l == r) ans[l] = uf.comps;
    else{
      int mid = (l + r)/2;
      dfs(2*v,l,mid,ans);
      dfs(2*v + 1,mid + 1,r,ans);
    }
    // rollback in data structure
    for(query q: tree[v]) if(q.status) uf.rb();
  }
  vector<int> getAns(){
    vector<int> ans(size);
    dfs(1,0,size - 1,ans);
    return ans;
  }
};
```

Segment Tree Lazy

You must provide a description for each template code!

```
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 segment_tree_lazy{
  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]);
  }
  segment_tree_lazy(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;
}
```

Segment Tree

You must provide a description for each template code!

```cpp
template<class T, T m_(T, T)> struct segment_tree{
  int n; vector<T> ST;
  segment_tree(){}
  segment_tree(vector<T> &a){
    n = a.size(); ST.resize(n << 1);
    for (int i=n;i<(n<<1);i++)ST[i]=a[i-n];
    for (int i=n-1;i>0;i--)ST[i]=m_(ST[i<<1],ST[i<<1|1]);
  }
  void update(int pos, T val){ // replace with val
    ST[pos += n] = val;
    for (pos >>= 1; pos > 0; pos >>= 1)
      ST[pos] = m_(ST[pos<<1], ST[pos<<1|1]);
  }
  T query(int l, int r){ // [l, r]
    T ansL, ansR; bool hasL = 0, hasR = 0;
    for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
      if (l & 1)
        ansL=(hasL?m_(ansL,ST[l++]):ST[l++]),hasL=1;
      if (r & 1)
        ansR=(hasR?m_(ST[--r],ansR):ST[--r]),hasR=1;
    }
    if (!hasL) return ansR; if (!hasR) return ansL;
    return m_(ansL, ansR);
  }
};
```

Sparse Table

You must provide a description for each template code!

```cpp
// Precomputacion en O(n logn), query en O(1)
template <typename T>
struct sparse_table {
  int n;
  vector<vector<T>> table;
  function<T(T, T)> merge;
  sparse_table(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; }
};
```

Treap

You must provide a description for each template code!

```cpp
struct treap {
  static mt19937_64 MT;
  struct node {
    node *left, *right; ll key, priority, value, max_value;
    node(ll k, ll v = 0) {
      left = right = NULL; key = k; priority = MT();
      max_value = value = v;
    }
  };
  ll value(node* T) { return T ? T->value : -INF; }
  ll max_value(node* T) { return T ? T->max_value : -INF; }
  void update(node* T) {
    T->max_value = max({T->value, max_value(T->left), max_value(T-
>right)});}
  node *root;
  void merge(node* &T, node* T1, node* T2) {
    if(T1 == NULL) { T = T2; return; }
    if(T2 == NULL) { T = T1; return; }
    if(T1->priority > T2->priority)
      merge(T1->right, T1->right, T2), T = T1;
    else merge(T2->left, T1, T2->left), T = T2;
    return update(T);
  }
  void merge(node* &T, node* T1, node* T2, node* T3) {
    merge(T, T1, T2); merge(T, T, T3);
  }
  void split(node* T, ll x, node* &T1, node* &T2) {
    if(T == NULL) { T1 = T2 = NULL; return; }
    if(T->key <= x) { split(T->right, x, T->right, T2); T1 = T; }
    else { split(T->left, x, T1, T->left); T2 = T; }
    return update(T);
  }
  void split(node* T, ll x, ll y, node* &T1, node* &T2, node* &T3) {
    split(T, x-1, T1, T2);  split(T2, y, T2, T3);
  }
  bool search(node* T, ll x) {
    if(T == NULL) return false; if(T->key == x) return true;
    if(x < T->key) return search(T->left, x);
    return search(T->right, x);
  }
  void insert(node* &T, node* n) {
    if(T == NULL) { T = n; return; }
    if(n->priority > T->priority) {
      split(T, n->key, n->left, n->right); T = n;
    } else if(n->key < T->key) insert(T->left, n);
    else insert(T->right, n);
    return update(T);
  }
  void erase(node* &T, ll x) {
    if(T == NULL) return;
    if(T->key == x) { merge(T, T->left, T->right); }
    else if(x < T->key) erase(T->left, x);
    else erase(T->right, x);
    return update(T);
  }
  bool set(node* T, ll k, ll v) {
    if(T == NULL) return false;
    bool found;
    if(T->key == k) T->value = k, found = true;
    else if(k < T->key) found = set(T->left, k, v);
    else found = set(T->right, k, v);
    if(found) update(T); return found;
  }
  node* find(node* T, ll k) {
    if(T == NULL) return NULL;
    if(T->key == k) return T;
    if(k < T->key) return find(T->left, k);
    return find(T->right, k);
  }
  treap() {root = NULL;}
  treap(ll x) {root = new node(x);}
  treap &merge(treap &O) {merge(root, root, O.root); return *this; }
  treap split(ll x) {treap ans; split(root, x, root, ans.root); return
ans; }
  bool search(ll x) {return search(root, x); }
  void insert(ll x) {if(search(root, x)) return; return insert(root, new
node(x));}
  void erase(ll x) {return erase(root, x); }
  void set(ll k, ll v) {if(set(root, k, v)) return; insert(root, new node(k,
v));}
  ll operator[](ll k) {
    node* n = find(root, k);
    if(n == NULL) n = new node(k), insert(root, n); return n->value;
  }
  ll query(ll a, ll b) {
    node *T1, *T2, *T3; split(root, a, b, T1, T2, T3);
    ll ans = max_value(T2); merge(root, T1, T2, T3);
    return ans;
  }
};
mt19937_64
treap::MT(chrono::system_clock::now().time_since_epoch().count());
```

Union Find

You must provide a description for each template code!

```
struct union_find {
  vector<int> e;
  union_find(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;
  }
};
```

Union Find Rollback

You must provide a description for each template code!

```
struct op{
  int v,u;
  int v_value,u_value;
  op(int _v,int _v_value,int _u,int _u_value):
v(_v),v_value(_v_value),u(_u),u_value(_u_value) {}
};
struct union_find_rb {
  vector<int> e;
  stack<op> ops;
  int comps;
  union_find_rb(){}
  union_find_rb(int n): comps(n) {e.assign(n, -1);}
  int findSet (int x) {
    return (e[x] < 0 ? 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);
    ops.push(op(x,e[x],y,e[y])); comps--;
    e[x] += e[y], e[y] = x;
    return 1;
  }
  void rb(){
    if(ops.empty()) return;
    op last = ops.top(); ops.pop();
    e[last.v] = last.v_value;
    e[last.u] = last.u_value;
    comps++;
  }
};
```

Wavelet Tree

You must provide a description for each template code!

```
struct WT {
  typedef vi::iterator iter;
  vvi r0; vi arrCopy; int n, s, q, w;
  void build(iter b, iter e, int l, int r, int u) {
    if (l == r) return;
    int m = (l + r) / 2;
    r0[u].reserve(e - b + 1); r0[u].pb(0);
    for (iter it = b; it != e; ++it)
      r0[u].pb(r0[u].back() + (*it <= m));
    iter p = stable_partition(b, e, [=](int i) { return i <= m; });
    build(b, p, l, m, u * 2); build(p, e, m + 1, r, u * 2 + 1);
  }
  int range(int a, int b, int l, int r, int u) {
    if (r < q or w < l) return 0;
    if (q <= l && r <= w) return b - a;
    int m = (l + r) / 2, za = r0[u][a], zb = r0[u][b];
    return range(za, zb, l, m, u * 2) + range(a - za, b - zb, m + 1, r, u *
2 + 1);
  }
  WT(vi arr, int sigma) { // arr[i] in [0,sigma)
      n = sz(arr); s = sigma; r0.resize(s * 2);
      arrCopy = arr;
      build(all(arr), 0, s - 1, 1);
  }
  // k in [1,n], [a,b) is 0-indexed, -1 if error
  int quantile(int k, int a, int b) {
    if (/*a < 0 or b > n or*/ k < 1 or k > b - a) return -1;
    int l = 0, r = s - 1, u = 1, m, za, zb;
    while (l != r) {
      m = (l + r) / 2;
      za = r0[u][a], zb = r0[u][b], u *= 2;
      if (k <= zb - za) a = za, b = zb, r = m;
      else k -= zb - za, a -= za, b -= zb, l = m + 1, ++u;
    }
    return r;
  }
  // counts numbers in [x,y] in positions [a,b)
  int range(int x, int y, int a, int b) {
    if (y < x or b <= a) return 0;
    q = x, w = y;
    return range(a, b, 0, s - 1, 1);
  }
  // count occurrences of x in positions [0,k)
  int rank(int x, int k) {
    int l = 0, r = s - 1, u = 1, m, z;
    while (l != r) {
      m = (l + r) / 2;
      z = r0[u][k], u *= 2;
      if (x <= m) k = z, r = m;
      else k -= z, l = m + 1, ++u;
    }
    return k;
  }
  void pb(int x) { // x in [0,sigma)
    int l = 0, r = s - 1, u = 1, m, p; ++n;
    while (l != r) {
      m = (l + r) / 2;
      p = (x <= m);
      r0[u].pb(r0[u].back() + p);
      u *= 2;
      if (p) r = m;
      else l = m + 1, ++u;
    }
  }
  void pop_back() { // doesn't check if empty
    int l = 0, r = s - 1, u = 1, m, p, k; --n;
    while (l != r) {
      m = (l + r) / 2;
      k = sz(r0[u]), p = r0[u][k - 1] - r0[u][k - 2];
      r0[u].pop_back(); u *= 2;
      if (p) r = m;
      else l = m + 1, ++u;
    }
  }
  void swap_adj(int i) { // swap arr[i] with arr[i+1], i in [0,n-1)
    int &x = arrCopy[i], &y = arrCopy[i + 1];
    int l = 0, r = s - 1, u = 1;
    while (l != r) {
      int m = (l + r) / 2, p = (x <= m), q = (y <= m);
      if (p != q) { r0[u][i + 1] ^= r0[u][i] ^ r0[u][i + 2]; break; }
      u *= 2; if (p) r = m;
      else l = m + 1, ++u;
    }
    swap(x, y);
  }
};
```

Dynamic Segment Tree

Description:
Status:

```
template <
    class T,              //Tipo de dato de los nodos
    class MAXi,           //Tipo de dato de los rangos (int, long long o
__int128)
    T merge(T, T),        //Merge
    T init(MAXi, MAXi)    //init(a, b) es el valor que tiene la query de a a
b si es que no hay
                          //updates en ese rango.
>
struct dynamic_segment_tree {
  vector<T> ST; vector<int>L, R;
  MAXi n; int n_count;
  dynamic_segment_tree (MAXi n, int r) :
    n(n),n_count(1),L(1),R(1),ST(1){
    ST.reserve(r);
    L.reserve(r);
    R.reserve(r);
    ST[0] = init(0, n - 1);
  }
  int addNode(MAXi l, MAXi r){
    L.push_back(0);
    R.push_back(0);
    ST.push_back(init(l, r));
    return n_count ++;
  }
  T query(int i, MAXi l, MAXi r, MAXi a, MAXi b) {
    if (a <= l and r <= b)
      return ST[i];
    MAXi mid = ((l + r) >> 1LL);
    if (b <= mid)
      return (L[i] != 0 ? query(L[i], l, mid, a, b) : init(l, mid));
    else if (a > mid)
      return (R[i] != 0 ? query(R[i], mid + 1, r, a, b) : init(mid + 1, r));
    if (L[i] == 0) L[i] = addNode(l, mid);
    if (R[i] == 0) R[i] = addNode(mid + 1, r);
    return merge(query(L[i], l, mid, a, b), query(R[i], mid + 1, r, a, b));
  }
  T query(MAXi a, MAXi b) {
    return query(0, 0, n - 1, a, b);
  }
  void update(int i, MAXi l, MAXi r, MAXi p, T v) {
    if (l == r){
      ST[i] = v; return;
    }
    MAXi mid = (l + r) / 2LL;
    if (p <= mid)
      update(L[i] != 0 ? L[i] : L[i] = addNode(l, mid), l, mid, p, v);
    else
      update(R[i] != 0 ? R[i] : R[i] = addNode(mid + 1, r), mid + 1, r, p,
v);
    ST[i] = merge(
        L[i] != 0 ? ST[L[i]] : init(l, mid),
        R[i] != 0 ? ST[R[i]] : init(mid + 1, r)
      );
  }
  void update(MAXi pos, T v) {
    update(0, 0, n - 1, pos, v);
  }
};
```

Maths

Fraction

Description: Just a fraction, every operation runs on $O(\log(\min(p, q)))$
Status: Not tested

```
#define lcm(a, b) ( gcd(a, b) ? ( (a)*(b) ) / gcd(a, b) ): 0
template <typename T>
struct Fraction {
  T p, q;
  Fraction() {}
  Fraction(T p, T q): p(p), q(q) {
    if (q < 0) this->p = -p, this->q = -q;
  }
  bool operator<(const Fraction o) {
    return p*o.q < o.p*q;
  }
  Fraction simplify(Fraction f){
    ll g = __gcd(f.p, f.q);
    return Fraction(f.p/g, f.q/g);
  }
  Fraction add(Fraction f){
    ll l = lcm(q, f.q);
    p *= (l/q);
    p += f.p * (l/f.q);
    return simplify(Fraction(p, l));
  }
};
```

Binary Pow

Description: Exponentiation by squares, computes $a^b \bmod m$ in $O(\log b)$
Status: Highly tested

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

Binary Pow 128 Bits

Description: Binary pow in 128 bits, computes $a^b \bmod m$ in $O(\log b)$
Status: Highly tested

```
using u64 = uint64_t;
using u128 = __uint128_t;
u64 binpow(u64 a, u64 b, u64 m) {
  a %= m;
  u64 res = 1;
  while (b > 0) {
    if (b & 1)
      res = (u128)res * a % m;
    a = (u128)a * a % m;
    b >>= 1;
  }
  return res;
}
```

Tetration

Description: Calculate ${}^{b}a \bmod m$
Status: Tested on josupo.jp

```
map<ll, ll> memophi;
ll tetration(ll a, ll b, ll m) {
  if (m == 1) return 0;
  if (a == 0) return (b+1) % 2 % m;
  if (a == 1 or b == 0) return 1;
  if (b == 1) return a % m;
  if (a == 2 and b == 2) return 4 % m;
  if (a == 2 and b == 3) return 16 % m;
  if (a == 3 and b == 2) return 27 % m;
  if (memophi.find(m) == memophi.end())
    memophi[m] = phi(m);
  ll tot = memophi[m];
  ll n = tetration(a, b-1, tot);
  return binpow(a, (n < tot ? n + tot : n), m);
}
```

Extended Gcd

Description: Extended euclidean algorithm
Status: Not stress-tested, but it gave AC in problems

```
struct GCD_type { ll x, y, d; };
GCD_type ex_GCD(ll a, ll b){
  if (b == 0) return {1, 0, a};
  GCD_type pom = ex_GCD(b, a % b);
  return {pom.y, pom.x - a / b * pom.y, pom.d};
}
```

CRT

Description: Chinese remainder theorem
Status: Not stress-tested, but it gave AC in problems

```
ll crt(vector<ll> a, vector<ll> m){
  int n = a.size();
  for (int i = 0; i < n; i ++){
    a[i] %= m[i];
    a[i] = a[i] < 0 ? a[i] + m[i] : a[i];
  }
  ll ans = a[0];
  ll M = m[0];
  for (int i = 1; i < n; i ++){
    auto pom = ex_GCD(M, m[i]);
    ll x1 = pom.x;
    ll d = pom.d;
    if ((a[i] - ans) % d != 0)
      return -1;
    ans = ans + x1 * (a[i] - ans) / d % (m[i] / d) * M;
    M = M * m[i] / d;
    ans %= M;
    ans = ans < 0 ? ans + M : ans;
    M = M / __gcd(M, m[i]) * m[i];
  }
  return ans;
}
```

Prime Factors

Description: Get all prime factors of n, runs on $O(\sqrt{n})$
Status: Highly tested

```
vector<ll> primeFactors(ll n) {
  vector<ll> factors;
  for (ll 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;
}
```

Counting Divisors

Description: Counting divisors in $O\left(n^{\frac{1}{3}}\right)$
Status: Tested on codeforces

```
const int MX_P = 1e6 + 1;
EratosthenesSieve sieve(MX_P);
int counting_divisors(int n) {
  int ret = 1;
  for (int p : sieve.primes) {
    if (p*p*p > n) break;
    int count = 1;
    while (n % p == 0)
      n /= p, count++;
    ret *= count;
  }
  int isqrt = sqrt(n);
  if (miller_rabin(n)) ret *= 2;
  else if (isqrt*isqrt == n and miller_rabin(isqrt)) ret *= 3;
  else if (n != 1) ret *= 4;
  return ret;
}
```

Divisors

Author: Sebastian Torrealba
Description: Get all divisors in $O(\sqrt{n})$, it includes 1 and n
Status: Highly tested

```
vector<ll> get_divisors(ll n) {
  vector<ll> left, right, ans;
  for (ll i = 1; i * i <= n; i++)
    if (n % i == 0) {
      if (i != n / i)
        right.push_back(n / i);
      left.push_back(i);
    }
  ans.resize(left.size() + right.size());
  reverse(begin(right), end(right));
  ll i = 0, j = 0;
  while (i < left.size() and j < right.size()) {
    if (left[i] < right[j])
      ans[i + j - 1] = left[i++];
    else ans[i + j - 1] = right[j++];
  }
  while(i < left.size()) ans[i + j - 1] = left[i++];
  while(j < right.size()) ans[i + j - 1] = right[j++];
  return ans;
}
```

Discrete Log

Author: Marcos Kolodny (MarcosK)
Description: Returns x such that $a^x = b (\bmod m)$ or -1 if inexistent
Status: Tested on josupo.jp

```
ll discrete_log(ll a,ll b,ll m) {
  a%=m, b%=m;
  if(b == 1) return 0;
  int cnt=0, tmp=1;
  for(int g=__gcd(a,m);g!=1;g=__gcd(a,m)) {
    if(b%g) return -1;
    m/=g, b/=g;
    tmp = tmp*a/g%m;
    ++cnt;
    if(b == tmp) return cnt;
  }
  map<ll,int> w;
  int s = ceil(sqrt(m)), base = b;
  for (int i = 0; i < s; i++)
    w[base] = i, base=base*a%m;
  base=binpow(a,s,m);
  ll key=tmp;
  for(int i = 1; i < s+2; i++) {
    key=base*key%m;
    if(w.count(key)) return i*s-w[key]+cnt;
  }
  return -1;
}
```

Erathostenes Sieve

Description: Get all prime numbers up to n, runs on $O(n \log(\log(n)))$
Status: Highly tested

```
// Corre en O(n log(log(n)) ))

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

Euler Phi

Description: Return $\varphi(n)$, runs on $O(\sqrt{n})$
Status: Not tested

```
ll phi(ll n) {
  ll result = n;
  for (ll 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;
}
```

Euler Phi Sieve

Description: Return $\varphi(n)$ for all positive n, runs on $O(n \log(\log(n)))$
Status: Not tested

```
struct euler_phi {
  vector<int> phi;
  euler_phi(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);
    }
  }
};
```

Factorial

Description: Calculate factorials from 1 to n and their factorial, runs on $O(n)$
Status: Highly tested

```
struct Factorial {
  vector<ll> f, finv, inv; ll mod;
  Factorial(ll n, ll mod): mod(mod) {
    f.assign(n+1, 1); inv.assign(n+1, 1); finv.assign(n+1, 1);
    for(ll i = 2; i <= n; ++i)
      inv[i] = mod - (mod/i) * inv[mod%i] % mod;
    for (ll i = 1; i <= n; ++i) {
      f[i] = (f[i-1] * i) % mod;
      finv[i] = (finv[i-1] * inv[i]) % mod;
    }
  }
}
```

FFT

Description: FFT(a) computes $\widehat{f(k)} = \sum_x a[x] \exp(2\pi i \cdot kx/N)$ for all k. N must be a power of 2, runs on $O(N \log N)$, where $N = |A| + |B|$
- For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by n, reverse(start+1, end), FFT back.
- Rounding is safe if $(\sum a_i^2 + \sum b_i^2) \log_2(N) < 9 \cdot 10^{14}$ (in practice 10^{16}; higher for random inputs).
Status: Highly tested (josupo.jp)

```
struct FFT {
  const long double PI = acos(-1);
  typedef long double d; // to double if too slow
  void fft(vector<complex<d>> &a) {
    int n = a.size(), L = 31 - __builtin_clz(n);
    vector<complex<d>> R(2, 1), rt(2, 1);
    for (int k = 2; k < n; k *= 2) {
      R.resize(n); rt.resize(n);
      auto x = polar(1.0L, PI / k);
      for(int i = k; i < 2*k; ++i) rt[i] = R[i] = i & 1 ? R[i / 2] * x :
R[i / 2];
    }
    vector<int> rev(n);
    for(int i = 0; i < n; ++i) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    for(int i = 0; i < n; ++i) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k)
    for(int j = 0; j < k; ++j) {
      auto x = (d*)&rt[j + k], y = (d*)&a[i + j + k];
      complex<d> z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]);
      a[i + j + k] = a[i + j] - z, a[i + j] += z;
    }
  }
  vector<int> conv(vector<d> &a, vector<d> &b) {
    if (a.empty() || b.empty()) return {};
    vector<d> res(a.size() + b.size() - 1);
    int B = 32 - __builtin_clz(res.size()), n = 1 << B;
    vector<complex<d>> in(n), out(n);
    copy(a.begin(), a.end(), in.begin());
    for(int i = 0; i < b.size(); ++i) in[i].imag(b[i]);
    fft(in); for (auto &x : in) x *= x;
    for(int i = 0; i < n; ++i) out[i] = in[-i & (n - 1)] - conj(in[i]);
    fft(out); for(int i = 0; i < res.size(); ++i) res[i] = imag(out[i]) / (4
* n);
    vector<int> resint(n);
    for (int i = 0; i < n; i++) resint[i] = round(res[i]);
    return resint;
  }
  vector<int> convMod(vector<int> &a, vector<int> &b, int mod) {
    if (a.empty() || b.empty()) return {};
    vector<d> res(a.size() + b.size() - 1);
    int B = 32 - __builtin_clz(res.size()), n = 1 << B, cut =
int(sqrt(mod));
    vector<complex<d>> L(n), R(n), outs(n), outl(n);
    for (int i = 0; i < a.size(); i++) L[i] = complex<d>(a[i]/cut, a[i]
%cut);
    for (int i = 0; i < b.size(); i++) R[i] = complex<d>(b[i]/cut, b[i]
%cut);
    fft(L), fft(R);
    for (int i = 0; i < n; i++) {
      int j = -i & (n-1);
      outl[j] = (L[i] + conj(L[j])) * R[i] / ((d)2.0 * n);
      outs[j] = (L[i] - conj(L[j])) * R[i] / ((d)2.0 * n) / complex<d>(0,
1);
    }
    fft(outl), fft(outs);
    for (int i = 0; i < res.size(); i++) {
      int av = (int)(real(outl[i])+.5), cv = (int)(imag(outs[i])+.5);
      int bv = (int)(imag(outl[i])+.5) + (int)(real(outs[i])+.5);
      res[i] = ((av % mod * cut + bv) % mod * cut + cv) % mod;
    }
    vector<int> resint(n);
    for (int i = 0; i < n; i++) resint[i] = round(res[i]);
    return resint;
  }
};
```

NTT

Description: Same utility as FFT but with some magic primes, runs in $O(n \log n)$ with better constant
Posible primes and their roots:
- 998244353, 3
- 9223372036737335297, 3

Status: Tested only using 998244353 as mod

```
template<int mod, int root>
struct NTT {
  void ntt(int* x, int* temp, int* roots, int N, int skip) {
    if (N == 1) return;
    int n2 = N/2;
    ntt(x, temp, roots, n2, skip*2);
    ntt(x+skip, temp, roots, n2, skip*2);
    for (int i = 0; i < N; i++) temp[i] = x[i*skip];
    for (int i = 0; i < n2; i++) {
      int s = temp[2*i], t = temp[2*i+1] * roots[skip*i];
      x[skip*i] = (s + t) % mod;
      x[skip*(i+n2)] = (s - t) % mod;
    }
  }
  void ntt(vector<int>& x, bool inv = false) {
    int e = binpow(root, (mod-1)/(x.size()), mod);
    if (inv) e = binpow(e, mod-2, mod);
    vector<int> roots(x.size(), 1), temp = roots;
    for (int i = 1; i < x.size(); i++) roots[i] = roots[i-1] * e % mod;
    ntt(&x[0], &temp[0], &roots[0], x.size(), 1);
  }
  vector<int> conv(vector<int> a, vector<int> b) {
    int s = a.size()+b.size()-1;
    if (s <= 0) return {};
    int L = s > 1 ? 32 - __builtin_clz(s - 1) : 0, n = 1 << L;
    a.resize(n); ntt(a);
    b.resize(n); ntt(b);
    vector<int> c(n); int d = binpow(n, mod-2, mod);
    for (int i = 0; i < n; i++) c[i] = a[i] * b[i] % mod * d % mod;
    ntt(c, true); c.resize(s);
    for (int i = 0; i < n; i++) if(c[i] < 0) c[i] += mod;
    return c;
  }
};
```

Poly Shift

Description: Solves $f(x+c) = \sum_0^{n-1} b_i \cdot x^i$

```
vector<int> polyShift(vector<int> &a, int shift) {
  // change for any mod for ntt
  const int mod = 998244353;
  NTT<998244353, 3> ntt;
  int n = a.size() - 1;
  Factorial f(n, mod);
  vector<int> x(n+1), y(n+1);
  int cur = 1;
  for (int i = 0; i <= n; i++) {
    x[i] = cur * f.finv[i] % mod;
    cur = (cur * shift) % mod;
    y[i] = a[n - i] * f.f[n-i] % mod;
  }
  vector<int> tmp = ntt.conv(x, y), res(n+1);
  for (int i = 0; i <= n; i++)
    res[i] = tmp[n-i] * f.finv[i] % mod;
  return res;
}
```

Miller Rabin

Description: Detect if a number is prime or not in $O(\log^2(n))$, needs 128 bits binary pow
Status: Highly tested

```
bool miller_rabin(uint64_t n) {
  if (n <= 1) return false;
  auto check = [](uint64_t n, uint64_t a, uint64_t d, uint64_t s) {
    int x = binpow(a, d, n); // Usar binpow de 128bits
    if (x == 1 or x == n-1) return false;
    for (int r = 1; r < s; r++) {
      x = (__uint128_t)x*x % n;
      if (x == n-1) return false;
    }
    return true;
  };
  uint64_t r = 0, d = n - 1;
  while ((d & 1) == 0) d >>= 1, r++;
  for (int x : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
    if (x == n) return true;
    if (check(n, x, d, r)) return false;
  }
  return true;
}
```

Gauss

Description: Matrix elimination, runs on $O(n^3)$
Status: Not tested

```
const double EPS = 1e-18;
const int INF = 2; // it doesn't actually have to be infinity or a big
number
const int MOD = 1e9+7;
int gauss(vector<vector<ll>> a, vector<ll> & ans) {
  int n = (int) a.size();
  int m = (int) a[0].size() - 1;

  vector<int> where(m, -1);
  for (int col = 0, row = 0; col < m && row < n; ++col) {
    int sel = row;
    for (int i = row; i < n; ++i)
      if (abs(a[i][col]) > abs(a[sel][col]))
        sel = i;
    if (!a[sel][col])
      continue;
    for (int i = col; i <= m; ++i)
      swap(a[sel][i], a[row][i]);
    where[col] = row;

    for (int i = 0; i < n; ++i)
      if (i != row) {
        ll c = ((ll) a[i][col] * binpow(round(a[row][col]), MOD - 2, MOD)) %
MOD;
        for (int j = col; j <= m; ++j)
          a[i][j] = ((a[i][j] - a[row][j] * c) % MOD + MOD) % MOD;
      }
    ++row;
  }

  ans.assign(m, 0);
  for (int i = 0; i < m; ++i)
    if (where[i] != -1)
      ans[i] = ((ll) a[where[i]][m] * binpow(round(a[where[i]][i]), MOD - 2,
MOD)) % MOD;

  for (int i = 0; i < m; ++i)
    if (where[i] == -1)
      return INF;
  return 1;
}
```

Matrix

Author: Carlos Lagos
Description: Matrix and their operations
Status: ??

```
template<class T>
vector<vector<T>> multWithoutMOD(vector<vector<T>> &a, vector<vector<T>> &b)
{
    int n = a.size(),m = b[0].size(),l = a[0].size();
    vector<vector<T>> ans(n,vector<T>(m,0));
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            for(int k = 0; k < l; k++){
                ans[i][j] += a[i][k]*b[k][j];
            }
        }
    }
    return ans;
}

template<class T>
vector<vector<T>> mult(vector<vector<T>> a, vector<vector<T>> b,long long
mod){
    int n = a.size(),m = b[0].size(),l = a[0].size();
    vector<vector<T>> ans(n,vector<T>(m,0));
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            for(int k = 0; k < l; k++){
                T temp = (a[i][k]*b[k][j]) % mod;
                ans[i][j] = (ans[i][j] + temp) % mod;
            }
        }
    }
    /*
    for(auto &line: ans)
        for(T &a: line) a = (a % mod + mod) % mod;
    */
    return ans;
}

vector<vector<ll>> binpow(vector<vector<ll>> v,ll n,long long mod){
    ll dim = v.size(); vector<vector<ll>> ans(dim,vector<ll>(dim,0));
    for(ll i = 0; i < dim; i++) ans[i][i] = 1;
    while(n){
        if(n & 1) ans = mult(ans,v,mod);
        v = mult(v,v,mod);
        n = n >> 1;
    }
    return ans;
}
```

Graphs

DFS

Description: Just traverse a graph in dfs order
Status: Tested

```
void dfs(int s) {
  if (visited[s]) return;
  visited[s] = true;
  // process node s
  for (auto u: adj[s]) {
    dfs(u);
  }
}
```

BFS

Description: Just traverse a graph in bfs order
Status: Tested

```
visited[x] = true;
distance[x] = 0;
q.push(x);
while (!q.empty()) {
int s = q.front(); q.pop();
  // process node s
  for (auto u : adj[s]) {
    if (visited[u]) continue;
    visited[u] = true;
    distance[u] = distance[s]+1;
    q.push(u);
  }
}
```

Floyd Warshall

Description: All shortest path, runs on $O(V^3)$
Status: Tested

```
// Create distance matrix
for (int i = 1; i <= n; i++) {
  for (int j = 1; j <= n; j++) {
    if (i == j) distance[i][j] = 0;
    else if (adj[i][j]) distance[i][j] = adj[i][j];
    else distance[i][j] = INF;
  }
}

// Floyd-Warshall
for (int k = 1; k <= n; k++) {
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
      distance[i][j] = min(distance[i][j], distance[i][k]+distance[k][j]);
    }
  }
}
```

Dijkstra

Author: Antti Laaksonen
Description: Computes shortest path in $O(V\log(V+E))$, does not allow negative weights

```
for (int i = 1; i <= n; i++) distance[i] = INF;
distance[x] = 0;
q.push({0,x});
while (!q.empty()) {
  int a = q.top().second; q.pop();
  if (processed[a]) continue;
  processed[a] = true;
  for (auto u : adj[a]) {
    int b = u.first, w = u.second;
    if (distance[a]+w < distance[b]) {
      distance[b] = distance[a]+w;
      q.push({-distance[b],b});
    }
  }
}
```

Kruskal

Description: Minimum spanning tree in $O(E\log E)$
Status: Tested, but needs to be re-written

```
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();
  UnionFind 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;
}
```

Dinic

Author: Pablo Messina
Source: https://github.com/PabloMessina/Competitive-Programming-Material
Description: Flow algorithm with complexity $O(|E|\cdot|V|^2)$
Status: Not tested

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

Mfed

Description: Max flow but with lowerbound of flow for each edge. To check is a feasible solution, all edges of super-sink (N) node should be saturated
Status: Tested

```
struct max_flow_edge_demands {
  Dinic mf;
  vector<ll> in, out;
  ll N;
  max_flow_edge_demands(ll N): N(N), mf(N+2), in(N), out(N) {}
  void add_edge(ll u, ll v, ll cap, ll dem = 0) {
    mf.add_edge(u, v, cap - dem);
    out[u] += dem, in[v] += dem;
  }
  ll max_flow(ll s, ll t) {
    mf.add_edge(t, s, inf);
    for (ll i = 0; i < N; i++) {
      mf.add_edge(N, i, in[i]);
      mf.add_edge(i, N+1, out[i]);
    }
    return mf.max_flow(N, N+1);
  }
};
```

Articulation Points

Description: Finds articulation points in a graph using DFS. Runs on $O(V+E)$
Status: Tested on CSES

```
struct ArticulationPoints {
  int n, timer;
  vector <int> tin, low, vis;
  set <int> points;
  void dfs(int v, int p, vector<vector<int>> &adj) {
    vis[v] = true; tin[v] = low[v] = timer++;
    int child = 0;
    for (int to: adj[v]) {
      if (to == p) continue;
      if (vis[to]) low[v] = min(low[v], tin[to]);
      else {
        dfs(to, v, adj);
        low[v] = min(low[v], low[to]);
        if ((low[to] >= tin[v]) && (p != -1))
          points.insert(v);
        child++;
      }
    }
    if ((p == -1) && (child > 1)) points.insert(v);
  }
  ArticulationPoints(vector < vector < int >> & adj, int root = 0) {
    n = adj.size();
    vis.resize(n, false);
    tin.resize(n);
    low.resize(n);
    dfs(root, -1, adj);
  }
};
```

HLD

Author: Javier Oliva
Description: Answer queries in $O(\log(V) \cdot A \cdot B)$, where A is the complexity of merging two chains and B is the complexity of the data structure query.
Status: Partially tested

```
template <class DS, class T, T merge(T, T), int IN_EDGES>
struct heavy_light {
  vector <int> parent, depth, heavy, head, pos_down;
  int n, cur_pos_down; DS ds_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 <T> & a_down, vector <T> & values) {
    head[v] = h, pos_down[v] = cur_pos_down++;
    a_down[pos_down[v]] = values[v];
    if (heavy[v] != -1)
      decompose(heavy[v], h, adj, a_down, values);
    for (int c: adj[v]) {
      if (c != parent[v] && c != heavy[v])
        decompose(c, c, adj, a_down, values);
    }
  }
  heavy_light(vector <vector<int>> &adj, vector <T> & values) {
    n = adj.size();
    parent.resize(n);
    depth.resize(n);
    heavy.resize(n, -1);
    head.resize(n);
    pos_down.resize(n);
    vector < T > a_down(n);
    cur_pos_down = 0;
    dfs(0, adj);
    decompose(0, 0, adj, a_down, values);
    ds_down = DS(a_down);
  }
  void update(int a, int b, T x) {
    while (head[a] != head[b]) {
      if (depth[head[a]] < depth[head[b]])
        swap(a, b);
      ds_down.update(pos_down[head[a]], pos_down[a], x);
      a = parent[head[a]];
    }
    if (depth[a] < depth[b])
      swap(a, b);
    if (pos_down[b] + IN_EDGES > pos_down[a])
      return;
    ds_down.update(pos_down[b] + IN_EDGES, pos_down[a], x);
  }
  void update(int a, T x) { ds_down.update(pos_down[a], x); }
  T query(int a, int b) {
    T ans; bool has = 0;
    while (head[a] != head[b]) {
      if (depth[head[a]] < depth[head[b]])
        swap(a, b);
      ans = has ? merge(ans, ds_down.query(pos_down[head[a]],
pos_down[a])) : ds_down.query(pos_down[head[a]], pos_down[a]);
      has = 1;
      a = parent[head[a]];
    }
    if (depth[a] < depth[b])
      swap(a, b);
    if (pos_down[b] + IN_EDGES > pos_down[a])
      return ans;
    return has ? merge(ans, ds_down.query(pos_down[b] + IN_EDGES,
pos_down[a])) : ds_down.query(pos_down[b] + IN_EDGES, pos_down[a]);
  }
};
```

Ahld

Author: Javier Oliva
Description: Same complexity of HLD, but with worst constant. This implementation only uses asociativity
Status: Partially tested

```
template < class DS, class T, T merge(T, T), int IN_EDGES >
  struct associative_heavy_light {
    vector <int> parent, depth, heavy, head, pos_up, pos_down;
    int n, cur_pos_up, cur_pos_down;
    DS ds_up, ds_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 < T > & a_up, vector < T > & a_down, vector < T >
& 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);
      }
    }
    associative_heavy_light(vector < vector < int > >
      const & adj, vector < T > & 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 <T> 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);
      ds_up = DS(a_up);
      ds_down = DS(a_down);
    }
    void update(int a, int b, T x) {
      while (head[a] != head[b]) {
        if (depth[head[a]] < depth[head[b]])
          swap(a, b);
        ds_up.update(pos_up[a], pos_up[head[a]], x);
        ds_down.update(pos_down[head[a]], pos_down[a], x);
        a = parent[head[a]];
      }
      if (depth[a] < depth[b])
        swap(a, b);
      if (pos_up[a] > pos_up[b] - IN_EDGES)
        return;
      ds_up.update(pos_up[a], pos_up[b] - IN_EDGES, x);
      ds_down.update(pos_down[b] + IN_EDGES, pos_down[a], x);
    }
    void update(int a, T x) {
      ds_up.update(pos_up[a], x);
      ds_down.update(pos_down[a], x);
    }
    T query(int a, int b) {
      T ansL, ansR;
      bool hasL = 0, hasR = 0;
      while (head[a] != head[b]) {
        if (depth[head[a]] > depth[head[b]]) {
          hasL ? ansL = merge(ansL, ds_up.query(pos_up[a],
pos_up[head[a]])) : ansL = ds_up.query(pos_up[a], pos_up[head[a]]), hasL =
1;
          a = parent[head[a]];
        } else {
          hasR ? ansR = merge(ds_down.query(pos_down[head[b]], pos_down[b]),
ansR) : ansR = ds_down.query(pos_down[head[b]], pos_down[b]), hasR = 1;
          b = parent[head[b]];
        }
      }
      if (depth[a] > depth[b] && pos_up[a] <= pos_up[b] - IN_EDGES)
        hasL ? ansL = merge(ansL, ds_up.query(pos_up[a], pos_up[b] -
IN_EDGES)) : ansL = ds_up.query(pos_up[a], pos_up[b] - IN_EDGES), hasL = 1;
      else if (depth[a] <= depth[b] && pos_down[a] + IN_EDGES <=
pos_down[b])
        hasR ? ansR = merge(ds_down.query(pos_down[a] + IN_EDGES,
pos_down[b]), ansR) : ansR = ds_down.query(pos_down[a] + IN_EDGES,
pos_down[b]), hasR = 1;

      return (!hasL) ? ansR : (!hasR ? ansL : merge(ansL, ansR));
    }
  };
```

Eppstein

Description: Solve k-shortest path problem
Status: Tested on josupo.jp and CSES

```
struct Eppstein {
  #define x first
  #define y second
  using T = int; const T INF = 1e18;
  using Edge = pair<int, T>;
  struct Node { int E[2] = {}, s{0};  Edge x;  };
  T shortest;
  priority_queue<pair<T, int>> Q;
  vector<Node> P{1}; vector<int> h;
  Eppstein(vector<vector<Edge>>& G, int s, int t) {
    int n = G.size();
    vector<vector<Edge>> H(n);
    for(int i = 0; i < n; i++)
      for (Edge &e : G[i])
        H[e.x].push_back({i, e.y});
    vector<int> ord, par(n, -1);
    vector<T> d(n, -INF);
    Q.push({d[t] = 0, t});
    while (!Q.empty()) {
      auto v = Q.top(); Q.pop();
      if (d[v.y] == v.x) {
        ord.push_back(v.y);
        for (Edge &e : H[v.y])
          if (v.x-e.y > d[e.x]) {
            Q.push({d[e.x] = v.x-e.y, e.x});
            par[e.x] = v.y;
          }
      }
    }
    if ((shortest = -d[s]) >= INF) return;
    h.resize(n);
    for (int v : ord) {
      int p = par[v];
      if (p+1) h[v] = h[p];
      for (Edge &e : G[v])
        if (d[e.x] > -INF) {
          T k = e.y - d[e.x] + d[v];
          if (k or e.x != p) h[v] = push(h[v], {e.x, k});
          else p = -1;
        }
    }
    P[0].x.x = s;
    Q.push({0, 0});
  }
  int push(int t, Edge x) {
    P.push_back(P[t]);
    if (!P[t = int(P.size())-1].s or P[t].x.y >= x.y)
      swap(x, P[t].x);
    if (P[t].s) {
      int i = P[t].E[0], j = P[t].E[1];
      int d = P[i].s > P[j].s;
      int k = push(d ? j : i, x);
      P[t].E[d] = k;
    }
    P[t].s++;
    return t;
  }
  int nextPath() {
    if (Q.empty()) return -1;
    auto v = Q.top(); Q.pop();
    for (int i : P[v.y].E) if (i)
      Q.push({ v.x-P[i].x.y+P[v.y].x.y, i });
    int t = h[P[v.y].x.x];
    if (t) Q.push({ v.x - P[t].x.y, t });
    return shortest - v.x;
  }
};
```

Hungarian

Description: Solves assignament problem in $O(n^3)$. If the matrix is rectangular in $O(n^2m)$, where n is the longest side
Status: Tested

```
void Hungarian(vector<vector<int>> &A, vector<pair<int, int>> &result, int
&C, const int INF = 1e6 + 1) {
  int n = A.size() - 1, m = A[0].size() - 1;
  vector<int> minv(m + 1), u(n + 1), v(m + 1), p(m + 1), way(m + 1);
  vector<bool> used(m + 1);
  for (int i = 1; i <= n; ++i) {
    p[0] = i; int j0 = 0;
    for (int j = 0; j <= m; ++j)
      minv[j] = INF;
    for (int j = 0; j <= m; ++j)
      used[j] = false;
    do {
      used[j0] = true;
      int i0 = p[j0], delta = INF, j1;
      for (int j = 1; j <= m; ++j)
        if (!used[j]) {
          int cur = A[i0][j] - u[i0] - v[j];
          if (cur < minv[j]) minv[j] = cur, way[j] = j0;
          if (minv[j] < delta) delta = minv[j], j1 = j;
        }
      for (int j = 0; j <= m; ++j) {
        if (used[j]) u[p[j]] += delta, v[j] -= delta;
        else minv[j] -= delta;
      }
      j0 = j1;
    } while (p[j0] != 0);
    do {
      int j1 = way[j0];
      p[j0] = p[j1];
      j0 = j1;
    } while(j0);
  }
  for (int i = 1; i <= m; ++i)
    result.push_back(make_pair(p[i], i));
  C = -v[0];
}
```

Kosaraju

Description: SCC in $O(V + E)$
Status: Tested, but needs to be re-written, is too large

```
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();
  }
};
```

LCA

Description: Computes lowest common ancestor, precomputed in $O(V \log V)$, $O(\log V)$ per query, uses binary lifting and works for directed and undirected graphs.

```
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];
  }
};
```

Bellman Ford

Description: Calculates shortest paths from s in a graph that might have negative edge weights.
Status: Tested on CSES

```
struct BellmanFord {
  struct Edge { int from, to, weight; };
  int n, last_updated = -1; const int INF = 1e18;
  vector<int> p, dist;
  BellmanFord(vector<Edge> &G, int s) {
    n = G.size(); dist.assign(n+2, INF);
    p.assign(n+2, -1); dist[s] = 0;
    for (int i = 1; i <= n; i++) {
      last_updated = -1;
      for (Edge &e : G)
        if (dist[e.from] + e.weight < dist[e.to]) {
          dist[e.to] = dist[e.from] + e.weight;
          p[e.to] = e.from; last_updated = e.to;
        }
    }
  }
  bool getCycle(vector<int> &cycle) {
    if (last_updated == -1) return false;
    for (int i = 0; i < n-1; i++)
      last_updated = p[last_updated];
    for (int x = last_updated ;; x=p[x]) {
      cycle.push_back(x);
      if (x == last_updated and cycle.size() > 1) break;
    }
    reverse(cycle.begin(), cycle.end());
    return true;
  }
};
```

Geometry

Convex Hull

You must provide a description for each template code!

```
template<typename T>
vector<Point2D<T>> convexHull(vector<Point2D<T>> cloud, bool ac = 0) {
  int n = cloud.size(), k = 0;
  sort(cloud.begin(), cloud.end(), [](Point2D<T> &a, Point2D<T> &b) {
    return a.x < b.x or (a.x == b.x and a.y < b.y);
  });
  if (n <= 2 or (ac and n <= 3)) return cloud;
  bool allCollinear = true;
  for (int i = 2; i < n; ++i) {
    if (((cloud[1] - cloud[0]) ^ (cloud[i] - cloud[0])) != 0) {
      allCollinear = false; break;
    }
  }
  if (allCollinear) return ac ? cloud : vector<Point2D<T>>{cloud[0],
cloud.back()};
  vector<Point2D<T>> ch(2 * n);
  auto process = [&](int st, int end, int stp, int t, auto cmp) {
    for (int i = st; i != end; i += stp) {
      while (k >= t and cmp(ch[k - 1], ch[k - 2], cloud[i])) k--;
      ch[k++] = cloud[i];
    }
  };
  process(0, n, 1, 2, [&](auto a, auto b, auto c) {
    return ((a - b) ^ (c - b)) < (ac ? 0 : 1);
  });
  process(n - 2, -1, -1, k + 1, [&](auto a, auto b, auto c) {
    return ((a - b) ^ (c - b)) < (ac ? 0 : 1);
  });
  ch.resize(k - 1);
  return ch;
}
```

Dynamic Programming

LIS

You must provide a description for each template code!

```
template <class I> vector<int> LIS(const vector<I> &S) {
  if (S.empty()) return {};
  vector<int> prev(S.size());
  vector<pair<I, int>> res;
  for (int i = 0; i < S.size(); i++) {
    auto it = lower_bound(res.begin(), res.end(), pair<I, int>{S[i], i});
    if (it == res.end()) res.emplace_back(), it = res.end() - 1;
    *it = {S[i], i};
    prev[i] = (it == res.begin() ? 0 : (it - 1)->second);
  }
  int L = res.size(), cur = res.back().second;
  vector<int> ans(L);
  while (L--) ans[L] = cur, cur = prev[cur];
  /* Get the sequence
  for (int i = 0; i+1 < ans.size(); i++)
    ans[i] = S[ans[i]];
  */
  return ans;
}
```

Knuth

Description: ?
Status: ? (we need to change this xd)

```
int N;
vector<int> A;
vector<vector<int>> DP, OPT;
int main() {
  DP.assign(N + 1, vi(N + 1));
  OPT.assign(N + 1, vi(N + 1));
  rep(i, N) {
    DP[i][i + 1] = A[i + 1] - A[i];
    OPT[i][i + 1] = i;
  }
  repx(d, 2, N + 1)
    rep(l, N + 1 - d) {
      int r = l + d, l_ = OPT[l][r - 1], r_ = OPT[l + 1][r];
      DP[l][r] = 1e9;
      repx(i, l_, r_ + 1) {
        int aux = DP[l][i] + DP[i][r] + A[r] - A[l];
        if (aux < DP[l][r]) DP[l][r] = aux, OPT[l][r] = i;
      }
    }
}
```

Egg Drop

You must provide a description for each template code!

```
vector<vector<ll>> egg_drop(ll h,ll k){
  vector<vector<ll>> dp(h + 1,vector<ll>(k + 1));
  for(int i = 0; i < k + 1; i++) dp[0][i] = 0;
  for(int i = 1; i < h + 1; i++) dp[i][0] = INT_MAX;
  for(int j = 1; j < k + 1; j++) {
    for(int i = 1; i < h + 1; i++) {
      ll ans=INT_MAX,x=1,y=i;
      while(x <= y){
        ll mid = (x + y)/2;
        ll bottom = dp[mid - 1][j - 1];
        ll top = dp[i - mid][j];
        ll temp = max(bottom,top);
        if(bottom < top)
          x = mid + 1;
        else y = mid - 1;
        ans = min(ans,temp);
      }
      dp[i][j] = 1 + ans;

    }
  }
  return dp;
}
```

D&C

Description: ?
Status: ? (we need to change this xd)

```
// dp(i, j) = min dp(i-1,k-1) + C(k,j) for all k in [0, j]
// C(a,c) + C(b, d) <= C(a,d) + C(b,c) for all a <= b <= c <= d
vp c;
vl acum1, acum2;
ll cost(ll i, ll j) {
  return c[j].first * (acum1[j+1] - acum1[i]) - (acum2[j+1] - acum2[i]);
}
vector<ll> last, now;
void compute(int l, int r, int optl, int optr) {
    if (l > r) return;
    int mid = (l + r) / 2;
    pair<ll, int> best = {cost(0, mid), -1};
    for(int k = max(1, optl); k < min(mid, optr) + 1; k++)
        best = min(best, {last[k - 1] + cost(k, mid), k});
    now[mid] = best.first;
    compute(l, mid - 1, optl, best.second);
    compute(mid + 1, r, best.second, optr);
}
```