Contents

1. Contest

1.1. template.h

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

#include <ext/pb_ds/assoc_container.hpp>
//#pragma GCC target("bmi,bmi2,lzcnt,popcnt")
//#pragma GCC optimize("O2,unroll-loops")
//#pragma GCC target("avx2")

//#pragma GCC optimize("O2")
//#pragma GCC optimize("Ofast")
//#pragma GCC target("avx,avx2,fma")
using namespace std;
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> o_set;
// order_of_key (val): returns the no. of values less than val
// find_by_order (k): returns the kth largest element.(0-based)
template <typename T>
using minHeap = priority_queue<T, vector<T>, greater<T>>;
template <typename T>
using maxHeap = priority_queue<T>;
#define int long long
#define all(s) s.begin(), s.end()
#define sz(s) (int)s.size()
using longer = __int128_t;
typedef vector<int> vi;
typedef pair<int, int> pii;
const int INF = numeric_limits<int>::max();
const int M = 1e9 + 7;
void solve() {}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(NULL);
  cout.tie(NULL);
  int tt;
  cin >> tt;
  while (tt--) solve();
}
```

1.2. mint.h

```
template <const i32 mod>
struct mint {
  constexpr mint(i32 x = 0) : val(x % mod + (x < 0) * mod) {}
  mint &operator+=(const mint &b) {
    val += b.val;
    val -= mod * (val >= mod);
```

```
  return *this;
 }
 mint &operator-=(const mint &b) {
  val -= b.val;
  val += mod * (val < 0);
  return *this;
 }
 mint &operator*=(const mint &b) {
  val = 1ll * val * b.val % mod;
  return *this;
 }
 mint &operator/=(const mint &b) { return *this *= b.inv(); }
 mint inv() const {
  i32 x = 1, y = 0, t;
  for (i32 a = val, b = mod; b; swap(a, b), swap(x, y))
   t = a / b, a -= t * b, x -= t * y;
  return mint(x);
 }
 mint pow(int b) const {
  mint a = *this, res(1);
  for (; b; a *= a, b /= 2)
   if (b & 1) res *= a;
  return res;
 }
 friend mint operator+(const mint &a, const mint &b) { return
mint(a) += b; }
 friend mint operator-(const mint &a, const mint &b) { return
mint(a) -= b; }
 friend mint operator*(const mint &a, const mint &b) { return
mint(a) *= b; }
 friend mint operator/(const mint &a, const mint &b) { return
mint(a) /= b; }
 friend bool operator==(const mint &a, const mint &b) { return a.val
== b.val; }
 friend bool operator!=(const mint &a, const mint &b) { return
a.val != b.val; }
 friend bool operator<(const mint &a, const mint &b) { return a.val
< b.val; }
 friend ostream &operator<<(ostream &os, const mint &a) { return
os << a.val; }
 i32 val;
};
```

2. Data Structures

2.1. SegTree.h

```
template <typename T, typename F>
struct SegTree {
 int n, off, ct;
 vector<T> t;
 const T id;
 F f;
 SegTree(const vector<T>& a, T _id, F _f)
    : n(sz(a)), off(1 << 32 - __builtin_clz(n)), ct(n ^ off >> 1), t(2 *
n), id(_id), f(_f) {
  for (int i = 0; i < 2 * ct; i++) t[off + i] = a[i];
  for (int i = 2 * ct; i < n; i++) t[i + off - n] = a[i];
```

```
  for (int i = n - 1; i >= 1; i--) t[i] = f(t[2 * i], t[2 * i + 1]);
 }
 int i2leaf(int i) { return i + off - (i < 2 * ct ? 0 : n); }
 int leaf2i(int l) { return l - off + (l < off ? n : 0); }
 T query(int l, int r) {
  l = (l < 2 * ct) ? (l + off) : 2 * (l + off - n);
  r = (r < 2 * ct) ? (r + off) : 2 * (r + off - n);
  r += (r >= 2 * n);
  T resl(id), resr(id);
  for (; l <= r; l >>= 1, r >>= 1) {
   if (l == r) {
    resl = f(resl, t[l]);
    break;
   }
   if (l & 1) resl = f(resl, t[l++]);
   if (!(r & 1)) resr = f(t[r--], resr);
  }
  return f(resl, resr);
 }
 void update(int v, T value) {
  for (t[v = i2leaf(v)] = value; v >>= 1;)
   t[v] = f(t[2 * v], t[2 * v + 1]);
 }
 int lower_bound(int k) {
  if (t[1] < k) return n;
  T rem = id;
  int v = 1;
  while (v < n) {
   T resl = f(rem, t[2 * v]);
   if (resl >= k) {
    v = 2 * v;
   } else {
    rem = resl;
    v = 2 * v + 1;
   }
  }
  return leaf2i(v);
 }
};
```

2.2. LazySegTree.h

```
template <typename T, typename U>
struct seg_tree_lazy {
 int S, H;
 T zero;
 vector<T> value;
 U noop;
 vector<bool> dirty;
 vector<U> prop;
 seg_tree_lazy(int _S, T _zero = T(), U _noop = U()) {
  zero = _zero, noop = _noop;
  for (S = 1, H = 1; S < _S;) S *= 2, H++;
  value.resize(2 * S, zero);
  dirty.resize(2 * S, false);
  prop.resize(2 * S, noop);
 }
```

```
  void set_leaves(vector<T> &leaves) {
    copy(leaves.begin(), leaves.end(), value.begin() + S);
    for (int i = S - 1; i > 0; i--)
      value[i] = value[2 * i] + value[2 * i + 1];
  }
  void apply(int i, U &update) {
    value[i] = update(value[i]);
    if (i < S) {
      prop[i] = prop[i] + update;
      dirty[i] = true;
    }
  }
  void rebuild(int i) {
    for (int l = i / 2; l; l /= 2) {
      T combined = value[2 * l] + value[2 * l + 1];
      value[l] = prop[l](combined);
    }
  }
  void propagate(int i) {
    for (int h = H; h > 0; h--) {
      int l = i >> h;
      if (dirty[l]) {
        apply(2 * l, prop[l]);
        apply(2 * l + 1, prop[l]);

        prop[l] = noop;
        dirty[l] = false;
      }
    }
  }
  void upd(int i, int j, U update) {
    i += S, j += S;
    propagate(i), propagate(j);
    for (int l = i, r = j; l <= r; l /= 2, r /= 2) {
      if ((l & 1) == 1) apply(l++, update);
      if ((r & 1) == 0) apply(r--, update);
    }
    rebuild(i), rebuild(j);
  }
  T query(int i, int j) {
    i += S, j += S;
    propagate(i), propagate(j);
    T res_left = zero, res_right = zero;
    for (; i <= j; i /= 2, j /= 2) {
      if ((i & 1) == 1) res_left = res_left + value[i++];
      if ((j & 1) == 0) res_right = value[j--] + res_right;
    }
    return res_left + res_right;
  }
};
struct node {
  int sum, width;
  node operator+(const node &n) {
    // Change 1
    return {sum + n.sum, width + n.width};
  }
```

```
};
struct update {
  bool type;  // 0 for add, 1 for reset
  int value;
  node operator()(const node &n) {  // apply update on n
    // Change 2
    if (type)
      return {n.width * value, n.width};
    else
      return {n.sum + n.width * value, n.width};
  }
  update operator+(const update &u) {  // u is the recent update, *this
is the older update
    // Change 3
    if (u.type) return u;
    return {type, value + u.value};
  }
};
```

2.3. RMQ.h

```
template <class T>
struct RMQ {
  vector<vector<T>> jmp;
  RMQ(const vector<T>& V) : jmp(1, V) {
    for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *= 2, ++k) {
      jmp.emplace_back(sz(V) - pw * 2 + 1);
      for (int j = 0; j < sz(jmp[k]); j++)
        jmp[k][j] = min(jmp[k - 1][j], jmp[k - 1][j + pw]);
    }
  }
  T query(int a, int b) {
    assert(a <= b);  // tie(a, b) = minimax(a, b)
    int dep = 63 - __builtin_clzll(b - a + 1);
    return min(jmp[dep][a], jmp[dep][b - (1 << dep) + 1]);
  }
};
```

2.4. Fenwick.h

```
template <typename T>
struct Fenwick {
  vector<T> bit;
  vector<T>& original;
  Fenwick(vector<T>& _arr) : bit(_arr.size(), 0LL), original(_arr) {
    int n = sz(_arr);
    for (int i = 0; i < n; i++) {
      bit[i] = bit[i] + _arr[i];
      if ((i | (i + 1)) < n) bit[(i | (i + 1))] = bit[(i | (i + 1))] + bit[i];
    }
  }
  // returns smallest index i, st. sum[0..i] >= x, returns -1 if no such i
exists
  // returns n if x >= sum of array
  // ASSUMES NON NEGATIVE ENTRIES IN TREE
  int lower_bound(int x) {
    if (x < 0) return -1;
    if (x == 0) return 0;
```

```
  int pos = 0;
  for (int pw = 1LL << 20; pw; pw >>= 1)
   if (pw + pos <= sz(bit) and bit[pos + pw - 1] < x)
    pos += pw, x -= bit[pos - 1];
  return pos;
 }
 T query(int r) {
  assert(r < sz(bit));
  int ret = 0;
  for (r++; r > 0; r &= r - 1) ret += bit[r - 1];
  return ret;
 }
 T query(int l, int r) {
  T ret = query(r);
  if (l != 0) ret -= query(l - 1);
  return ret;
 }
 void update(int i, int x) {
  int n = bit.size();
  T diff = x - original[i];
  original[i] = x;
  for (; i < n; i = i | i + 1) bit[i] += diff;
 }
};
```

2.5. cht.h

```
struct Line {
 mutable i64 m, c, p;
 bool operator<(const Line& o) const { return m < o.m; }
 bool operator<(i64 x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
 // (for doubles, use inf = 1/.0, div(a,b) = a/b)
 static const i64 inf = LONG_LONG_MAX;
 i64 div(i64 a, i64 b) {  // floored division
  return a / b - ((a ^ b) < 0 && a % b);
 }
 bool isect(iterator x, iterator y) {
  if (y == end()) return x->p = inf, 0;
  if (x->m == y->m)
   x->p = x->c > y->c ? inf : -inf;
  else
   x->p = div(y->c - x->c, x->m - y->m);
  return x->p >= y->p;
 }
 void add(i64 m, i64 c) {
  auto z = insert({m, c, 0}), y = z++, x = y;
  while (isect(y, z)) z = erase(z);
  if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
  while ((y = x) != begin() && (--x)->p >= y->p)
   isect(x, erase(y));
 }
 i64 query(i64 x) {
  assert(!empty());
  auto l = *lower_bound(x);
```

```
    return l.m * x + l.c;
  }
};
```

2.6. DSU.h

```
struct DSU {
  int n;
  vi parent;
  vi size;
  DSU(int _n) : n(_n), parent(n), size(n, 1) { iota(parent.begin(),
parent.end(), 0); }
  int find_set(int x) {
    if (parent[x] == x) return x;
    return parent[x] = find_set(parent[x]);
  }
  int getSize(int x) { return size[find_set(x)]; }  // returns size of
component of x
  void union_sets(int x, int y) {
    x = find_set(x);
    y = find_set(y);
    if (x == y) return;
    if (size[x] > size[y]) {
      parent[y] = x;
      size[x] += size[y];
    } else {
      parent[x] = y;
      size[y] += size[x];
    }
  }
};
```

2.7. Fenwick2D.h

```
const int mxn = 1000;
int grid[mxn + 1][mxn + 1];
int bit[mxn + 1][mxn + 1];
void update(int row, int col, int d) {
  grid[row][col] += d;
  for (int i = row; i <= mxn; i += (i & -i))
    for (int j = col; j <= mxn; j += (j & -j))
      bit[i][j] += d;
}
int sum(int row, int col) {
  // calculates sum from [1,1] till [row,col]
  int res = 0;
  for (int i = row; i > 0; i -= (i & -i))
    for (int j = col; j > 0; j -= (j & -j))
      res += bit[i][j];
  return res;
}
```

2.8. Mos.h

```
int BLOCK = DO_NOT_FORGET_TO_CHANGE_THIS;
struct Query {
  int l, r, id;
  Query(int _l, int _r, int _id) : l(_l), r(_r), id(_id) {}
  bool operator<(Query &o) {
    int mblock = l / BLOCK, oblock = o.l / BLOCK;
```

```
    return (mblock < oblock) or
           (mblock == oblock and mblock % 2 == 0 and r < o.r) or
           (mblock == oblock and mblock % 2 == 1 and r > o.r);
  };
};
void solve() {
  vector<Query> queries;
  queries.reserve(q);
  for (int i = 0; i < q; i++) {
    int l, r;
    cin >> l >> r;
    l--, r--;
    queries.emplace_back(l, r, i);
  }
  sort(all(queries));
  int ans = 0;
  auto add = [&](int v) {};
  auto rem = [&](int v) {};
  vector<int> out(q);  // Change out type if necessary
  int cur_l = 0, cur_r = -1;
  for (auto &[l, r, id] : queries) {
    while (cur_l > l) add(--cur_l);
    while (cur_l < l) rem(cur_l++);
    while (cur_r < r) add(++cur_r);
    while (cur_r > r) rem(cur_r--);
    out[id] = ans;
  }
}
```

2.9. Persistent.h

```
const int N = 5e5 + 10, LOGN = 18;
int L[N * LOGN], R[N * LOGN], ST[N * LOGN];
int nodeid = 0;
// usage newrootId = update(i, 0, n - 1, val, oldrootId)
// [update index i to val]
int update(int pos, int l, int r, int val, int id) {
  if (pos < l or pos > r) return id;
  int ID = ++nodeid, m = (l + r) / 2;
  if (l == r) return (ST[ID] = val, ID);
  L[ID] = update(pos, l, m, val, L[id]);
  R[ID] = update(pos, m + 1, r, val, R[id]);
  return (ST[ID] = ST[L[ID]] + ST[R[ID]], ID);
}
// usage query(l, r, 0, n - 1, rootId)
int query(int ql, int qr, int l, int r, int id) {
  if (ql > r or qr < l) return 0;
  if (ql <= l and r <= qr) return ST[id];
  int m = (l + r) / 2;
  return (query(ql, qr, l, m, L[id])) + query(ql, qr, m + 1, r, R[id]);
}
// searches for upper bound of x, call as descent(0, n - 1, x, rootId)
int descent(int l, int r, int x, int id) {
  if (l == r) return l;
  int m = (l + r) / 2;
  int leftCount = ST[L[id]];
  if (leftCount <= x) {
    // is in right half
```

```
    return descent(m + 1, r, x - leftCount, R[id]);
  } else {
    // is in left half
    return descent(l, m, x, L[id]);
  }
}
```

2.10. Treap.h

```
/*A short self-balancing tree. It acts as a
sequential container with log-time splits/joins, and
is easy to augment with additional data.
Time: $O(\log N)$*/

struct Node {
  Node *l = 0, *r = 0;
  int val, y, c = 1;
  Node(int val) : val(val), y(rand()) {}
  void recalc();
};

int cnt(Node* n) { return n ? n->c : 0; }
void Node::recalc() { c = cnt(l) + cnt(r) + 1; }

template <class F>
void each(Node* n, F f) {
  if (n) {
    each(n->l, f);
    f(n->val);
    each(n->r, f);
  }
}

pair<Node*, Node*> split(Node* n, int k) {
  if (!n) return {};
  if (cnt(n->l) >= k) {  // "n->val >= k" for lower_bound(k)
    auto pa = split(n->l, k);
    n->l = pa.second;
    n->recalc();
    return {pa.first, n};
  } else {
    auto pa = split(n->r, k - cnt(n->l) - 1);  // and just "k"
    n->r = pa.first;
    n->recalc();
    return {n, pa.second};
  }
}

Node* merge(Node* l, Node* r) {
  if (!l) return r;
  if (!r) return l;
  if (l->y > r->y) {
    l->r = merge(l->r, r);
    l->recalc();
    return l;
  } else {
    r->l = merge(l, r->l);
```

```
    r->recalc();
    return r;
  }
}

Node* ins(Node* t, Node* n, int pos) {
  auto [l, r] = split(t, pos);
  return merge(merge(l, n), r);
}

// Example application: move the range [l, r) to index k
void move(Node*& t, int l, int r, int k) {
  Node *a, *b, *c;
  tie(a, b) = split(t, l);
  tie(b, c) = split(b, r - l);
  if (k <= l)
    t = merge(ins(a, b, k), c);
  else
    t = merge(a, ins(c, b, k - r));
}
```

3. Graph

3.1. SCC.h

```
struct SCC {
  int n;
  vi val, cc, z;
  vvi comps;
  SCC(vvi& adj) : n(sz(adj)), val(n), cc(n, -1) {
    int timer = 0;
    function<int(int)> dfs = [&](int x) {
      int low = val[x] = ++timer, b;
      z.push_back(x);
      for (auto y : adj[x])
        if (cc[y] < 0)
          low = min(low, val[y] ?: dfs(y));

      if (low == val[x]) {
        comps.push_back(vi());
        do {
          b = z.back();
          z.pop_back();
          comps.back().push_back(b);
          cc[b] = sz(comps) - 1;
        } while (x != b);
      }
      return val[x] = low;
    };
    for (int i = 0; i < n; i++)
      if (cc[i] < 0) dfs(i);
  }
  int operator[](int i) { return cc[i]; }
  int size(int i) { return sz(comps[cc[i]]); }
};
```

3.2. LCA.h

```
struct LCA {
  int T = 0;
  vi st, path, ret;
  vi en, d;
  RMQ<int> rmq;
  LCA(vector<vi>& C) : st(sz(C)), en(sz(C)), d(sz(C)), rmq((dfs(C, 0,
-1), ret)) {}
  void dfs(vvi& adj, int v, int par) {
    st[v] = T++;
    for (auto to : adj[v])
      if (to != par) {
        path.pb(v), ret.pb(st[v]);
        d[to] = d[v] + 1;
        dfs(adj, to, v);
      }
    en[v] = T - 1;
  }
  bool anc(int p, int c) { return st[p] <= st[c] and en[p] >= en[c]; }
  int lca(int a, int b) {
    if (a == b) return a;
    tie(a, b) = minmax(st[a], st[b]);
    return path[rmq.query(a, b - 1)];
  }
  int dist(int a, int b) { return d[a] + d[b] - 2 * d[lca(a, b)]; }
};
```

3.3. 2sat.h

```
/*
ts.either(x, y);
ts.either(~x, ~y); these two do x xor y

ts.setValue(x, x); assert x is true

use ~x to denote not x

call ts.solve() to run the solver, returns if a solution exists
if exists: ts.values[i] contains the assignments
*/

struct TwoSat {
  int N;
  vector<vi> gr;
  vi values;  // 0 = false, 1 = true

  TwoSat(int n = 0) : N(n), gr(2 * n) {}

  int addVar() {  // (optional)
    gr.emplace_back();
    gr.emplace_back();
    return N++;
  }

  void either(int f, int j) {
    f = max(2 * f, -1 - 2 * f);
    j = max(2 * j, -1 - 2 * j);
```

```
  gr[f].push_back(j ^ 1);
  gr[j].push_back(f ^ 1);
 }
 void setValue(int x) { either(x, x); }

 void atMostOne(const vi& li) {  // (optional)
  if (sz(li) <= 1) return;
  int cur = ~li[0];
  for (int i = 2; i < sz(li); i++) {
   int next = addVar();
   either(cur, ~li[i]);
   either(cur, next);
   either(~li[i], next);
   cur = ~next;
  }
  either(cur, ~li[1]);
 }

 vi val, comp, z;
 int time = 0;
 int dfs(int i) {
  int low = val[i] = ++time, x;
  z.push_back(i);
  for (int e : gr[i])
   if (!comp[e])
    low = min(low, val[e] ?: dfs(e));
  if (low == val[i]) do {
    x = z.back();
    z.pop_back();
    comp[x] = low;
    if (values[x >> 1] == -1)
     values[x >> 1] = x & 1;
   } while (x != i);
  return val[i] = low;
 }

 bool solve() {
  values.assign(N, -1);
  val.assign(2 * N, 0);
  comp = val;
  for (int i = 0; i < 2 * N; i++)
   if (!comp[i]) dfs(i);
  for (int i = 0; i < N; i++)
   if (comp[2 * i] == comp[2 * i + 1]) return 0;
  return 1;
 }
};
```

3.4. Dinic.h

```
// Flow algorithm with complexity $O(VE\log U)$ where $U = \max |\text{cap}|$.
// $O(\min(E^{1/2}, V^{2/3})E)$ if $U = 1$; $O(\sqrt{V}E)$ for bipartite matching.
using ll = long long;
#define rep(i, j, k) for (int i = j; i < k; i++)
struct Dinic {
```

```
struct Edge {
  int to, rev;
  ll c, oc;
  ll flow() { return max(oc - c, 0LL); }  // if you need flows
};
vi lvl, ptr, q;
vector<vector<Edge>> adj;
Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
void addEdge(int a, int b, ll c, ll rcap = 0) {
  adj[a].push_back({b, sz(adj[b]), c, c});
  adj[b].push_back({a, sz(adj[a]) - 1, rcap, rcap});
}
ll dfs(int v, int t, ll f) {
  if (v == t || !f) return f;
  for (int& i = ptr[v]; i < sz(adj[v]); i++) {
    Edge& e = adj[v][i];
    if (lvl[e.to] == lvl[v] + 1)
      if (ll p = dfs(e.to, t, min(f, e.c))) {
        e.c -= p, adj[e.to][e.rev].c += p;
        return p;
      }
  }
  return 0;
}
ll calc(int s, int t) {
  ll flow = 0;
  q[0] = s;
  rep(L, 0, 31) do {  // 'int L=30' maybe faster for random data
    lvl = ptr = vi(sz(q));
    int qi = 0, qe = lvl[s] = 1;
    while (qi < qe && !lvl[t]) {
      int v = q[qi++];
      for (Edge e : adj[v])
        if (!lvl[e.to] && e.c >> (30 - L))
          q[qe++] = e.to, lvl[e.to] = lvl[v] + 1;
    }
    while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
  }
  while (lvl[t])
    ;
  return flow;
}
bool leftOfMinCut(int a) { return lvl[a] != 0; }
};
```

3.5. HLD.h

```
struct HLD {
  int n, timer = 0;
  vi top, tin, p, sub;
  HLD(vvi &adj) : n(sz(adj)), top(n), tin(n), p(n, -1), sub(n, 1) {
    vi ord(n + 1);
    for (int i = 0, t = 0, v = ord[i]; i < n; v = ord[++i])
      for (auto &to : adj[v])
        if (to != p[v]) p[to] = v, ord[++t] = to;
    for (int i = n - 1, v = ord[i]; i > 0; v = ord[--i]) sub[p[v]] += sub[v];
    for (int v = 0; v < n; v++)
```

```
    if (sz(adj[v])) iter_swap(begin(adj[v]), max_element(all(adj[v]),
[&](int a, int b) { return make_pair(a != p[v], sub[a]) < make_pair(b !
= p[v], sub[b]); }));
   function<void(int)> dfs = [&](int v) {
    tin[v] = timer++;
    for (auto &to : adj[v])
     if (to != p[v]) {
      top[to] = (to == adj[v][0] ? top[v] : to);
      dfs(to);
     }
   };
   dfs(0);
  }
  int lca(int u, int v) {
   return process(u, v, [](...) {});
  }
  template <class B>
  int process(int a, int b, B op, bool ignore_lca = false) {
   for (int v;; op(tin[v], tin[b]), b = p[v]) {
    if (tin[a] > tin[b]) swap(a, b);
    if ((v = top[b]) == top[a]) break;
   }
   if (int l = tin[a] + ignore_lca, r = tin[b]; l <= r) op(l, r);
   return a;
  }
  template <class B>
  void subtree(int v, B op, bool ignore_lca = false) {
   if (sub[v] > 1 or !ignore_lca) op(tin[v] + ignore_lca, tin[v] +
sub[v] - 1);
  }
};
```

3.6. KthAnc.h

```
struct LCA {
 int n;
 vvi& adjLists;
 int lg;
 vvi up;
 vi depth;
 LCA(vvi& _adjLists, int root = 0) : n(sz(_adjLists)),
adjLists(_adjLists) {
  lg = 1;
  int pw = 1;
  while (pw <= n) pw <<= 1, lg++;
  // lg = 20
  up = vvi(n, vi(lg));
  depth.assign(n, -1);
  function<void(int, int)> parentDFS = [&](int from, int parent) {
   depth[from] = depth[parent] + 1;
   up[from][0] = parent;
   for (auto to : adjLists[from]) {
    if (to == parent) continue;
    parentDFS(to, from);
   }
  };
  parentDFS(root, root);
```

```
  for (int j = 1; j < lg; j++) {
   for (int i = 0; i < n; i++) {
    up[i][j] = up[up[i][j - 1]][j - 1];
   }
  }
 }
 int kthAnc(int v, int k) {
  int ret = v;
  int pw = 0;
  while (k) {
   if (k & 1) ret = up[ret][pw];
   k >>= 1;
   pw++;
  }
  return ret;
 }
 int lca(int u, int v) {
  if (depth[u] > depth[v]) swap(u, v);
  v = kthAnc(v, depth[v] - depth[u]);
  if (u == v) return v;
  while (up[u][0] != up[v][0]) {
   int i = 0;
   for (; i < lg - 1; i++) {
    if (up[u][i + 1] == up[v][i + 1]) break;
   }
   u = up[u][i], v = up[v][i];
  }
  return up[u][0];
 };
 int dist(int u, int v) {
  return depth[u] + depth[v] - 2 * depth[lca(u, v)];
 }
};
```

3.7. MinCostMaxFlow.h

```
template <const int MAX_N, typename flow_t,
          typename cost_t, flow_t FLOW_INF,
          cost_t COST_INF, const int SCALE = 16>
struct CostScalingMCMF {
#define sz(a) a.size()
#define zero_stl(v, sz) fill(v.begin(), v.begin() + (sz), 0)
 struct Edge {
  int v;
  flow_t c;
  cost_t d;
  int r;
  Edge() = default;
  Edge(int v, flow_t c, cost_t d, int r) : v(v), c(c), d(d), r(r) {}
 };
 vector<Edge> g[MAX_N];
 cost_t negativeSelfLoop;
 array<cost_t, MAX_N> pi, excess;
 array<int, MAX_N> level, ptr;
 CostScalingMCMF() { negativeSelfLoop = 0; }
 void clear() {
  negativeSelfLoop = 0;
```

```
  for (int i = 0; i < MAX_N; i++) g[i].clear();
 }
 void addEdge(int s, int e, flow_t cap, cost_t cost) {
  if (s == e) {
   if (cost < 0) negativeSelfLoop += cap * cost;
   return;
  }
  g[s].push_back(Edge(e, cap, cost, sz(g[e])));
  g[e].push_back(Edge(s, 0, -cost, sz(g[s]) - 1));
 }
 flow_t getMaxFlow(int V, int S, int T) {
  auto BFS = [&]() {
   zero_stl(level, V);
   queue<int> q;
   q.push(S);
   level[S] = 1;
   for (q.push(S); !q.empty(); q.pop()) {
    int v = q.front();
    for (const auto &e : g[v])
     if (!level[e.v] && e.c) q.push(e.v), level[e.v] = level[v] + 1;
   }
   return level[T];
  };
  function<flow_t(int, flow_t)> DFS = [&](int v, flow_t fl) {
   if (v == T || fl == 0) return fl;
   for (int &i = ptr[v]; i < (int)g[v].size(); i++) {
    Edge &e = g[v][i];
    if (level[e.v] != level[v] + 1 || !e.c) continue;
    flow_t delta = DFS(e.v, min(fl, e.c));
    if (delta) {
     e.c -= delta;
     g[e.v][e.r].c += delta;
     return delta;
    }
   }
   return flow_t(0);
  };
  flow_t maxFlow = 0, tmp = 0;
  while (BFS()) {
   zero_stl(ptr, V);
   while ((tmp = DFS(S, FLOW_INF))) maxFlow += tmp;
  }
  return maxFlow;
 }
 pair<flow_t, cost_t> maxflow(int N, int S, int T) {
  flow_t maxFlow = 0;
  cost_t eps = 0, minCost = 0;
  stack<int, vector<int>> stk;
  auto c_pi = [&](int v, const Edge &edge) { return edge.d + pi[v] -
pi[edge.v]; };
  auto push = [&](int v, Edge &edge, flow_t delta, bool flag) {
   delta = min(delta, edge.c);
   edge.c -= delta;
   g[edge.v][edge.r].c += delta;
   excess[v] -= delta;
   excess[edge.v] += delta;
```

```
    if (flag && 0 < excess[edge.v] && excess[edge.v] <= delta)
stk.push(edge.v);
   };
   auto relabel = [&](int v, cost_t delta) { pi[v] -= delta + eps; };
   auto lookAhead = [&](int v) {
    if (excess[v]) return false;
    cost_t delta = COST_INF;
    for (auto &e : g[v]) {
     if (e.c <= 0) continue;
     cost_t cp = c_pi(v, e);
     if (cp < 0)
      return false;
     else
      delta = min(delta, cp);
    }
    relabel(v, delta);
    return true;
   };
   auto discharge = [&](int v) {
    cost_t delta = COST_INF;
    for (int i = 0; i < sz(g[v]); i++) {
     Edge &e = g[v][i];
     if (e.c <= 0) continue;
     cost_t cp = c_pi(v, e);
     if (cp < 0) {
      if (lookAhead(e.v)) {
       i--;
       continue;
      }
      push(v, e, excess[v], true);
      if (excess[v] == 0) return;
     } else
      delta = min(delta, cp);
    }
    relabel(v, delta);
    stk.push(v);
   };
   zero_stl(pi, N);
   zero_stl(excess, N);
   for (int i = 0; i < N; i++)
    for (auto &e : g[i]) minCost += e.c * e.d, e.d *= MAX_N + 1, eps
= max(eps, e.d);
   maxFlow = getMaxFlow(N, S, T);
   while (eps > 1) {
    eps /= SCALE;
    if (eps < 1) eps = 1;
    stk = stack<int, vector<int>>();
    for (int v = 0; v < N; v++)
     for (auto &e : g[v])
      if (c_pi(v, e) < 0 && e.c > 0) push(v, e, e.c, false);
    for (int v = 0; v < N; v++)
     if (excess[v] > 0) stk.push(v);
    while (stk.size()) {
     int top = stk.top();
     stk.pop();
     discharge(top);
```

```
    }
   }
   for (int v = 0; v < N; v++)
    for (auto &e : g[v]) e.d /= MAX_N + 1, minCost -= e.c * e.d;
   minCost = minCost / 2 + negativeSelfLoop;
   return {maxFlow, minCost};
  }
};

void solve() {
  CostScalingMCMF<102, int, int, 100, 100> flow;
  int n, m;
  cin >> n >> m;
  int start = 0;
  for (int i = 0; i < n; i++) {
   for (int j = 0; j < m; j++) {
    int inp;
    cin >> inp;
    if (inp) {
     flow.addEdge(i + 1, n + 1 + j, 1, 0);
     start++;
    } else
     flow.addEdge(i + 1, n + 1 + j, 1, 1);
   }
  }
  int counta = 0, countb = 0;
  for (int i = 0; i < n; i++) {
   int inp;
   cin >> inp;
   counta += inp;
   flow.addEdge(0, i + 1, inp, 0);
  }
  for (int i = 0; i < m; i++) {
   int inp;
   cin >> inp;
   countb += inp;
   flow.addEdge(n + i + 1, n + m + 1, inp, 0);
  }
  if (counta != countb) {
   cout << -1 << endl;
   return;
  }
  pii t = flow.maxflow(102, 0, n + m + 1);
  if (t.first != counta) {
   cout << -1 << endl;
   return;
  }
  cout << t.second + start + t.second - counta << endl;
}
```

4. Number Theory

4.1. MillerRabin.h

```
u64 mult(u64 a, u64 b, u64 m = M) {
  i64 ret = a * b - m * (u64)(1.L / m * a * b);
  return ret + m * (ret < 0) - m * (ret >= (i64)m);
}
```

```
u64 pw(u64 b, u64 e, u64 m = M) {
  u64 ret = 1;
  for (; e; b = mult(b, b, m), e >>= 1)
    if (e & 1) ret = mult(ret, b, m);
  return ret;
}
bool isPrime(u64 n) {  // determistic upto 7e^18
  if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
  u64 A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
      s = __builtin_ctzll(n - 1), d = n >> s;
  for (u64 a : A) {
    u64 p = pw(a % n, d, n), i = s;
    while (p != 1 && p != n - 1 && a % n && i--)
      p = mult(p, p, n);
    if (p != n - 1 && i != s) return 0;
  }
  return 1;
}
```

4.2. gcdextended.h

```
int euclid(int a, int b, int &x, int &y) {
  if (!b) return x = 1, y = 0, a;
  int d = euclid(b, a % b, y, x);
  return y -= a / b * x, d;
}
```

4.3. spf.h

```
#define SIEVE_TILL (int)1e6
vector<int> primes;
vector<int> spf;
void sieve() {
  spf = vector<int>(SIEVE_TILL + 1, 0);
  for (int i = 2; i <= SIEVE_TILL; i++) {
    if (spf[i] == 0) primes.push_back(i), spf[i] = i;
    for (int j = 0; j < sz(primes) and i * primes[j] <= SIEVE_TILL; j+
+) {
      spf[i * primes[j]] = primes[j];
      if (spf[i] == primes[j]) break;
    }
  }
}
bool isPrime(int n) {
  if (n <= 1) return false;
  return spf[n] == n;
}
```

5. Strings

5.1. Manacher.h

```
/* Description: p[0][i] = half length of longest even palindrome
behind pos i,
p[1][i] = longest odd with center at pos i(half rounded down). */
array<vi, 2> manacher(const string& s) {
  int n = sz(s);
  array<vi, 2> p = {vi(n + 1), vi(n)};
  for (int z = 0; z < 2; z++)
    for (int i = 0, l = 0, r = 0; i < n; i++) {
      int t = r - i + !z;
```

```
    if (i < r) p[z][i] = min(t, p[z][l + t]);
    int L = i - p[z][i], R = i + p[z][i] - !z;
    while (L >= 1 && R + 1 < n && s[L - 1] == s[R + 1])
      p[z][i]++, L--, R++;
    if (R > r) l = L, r = R;
   }
  return p;
}
```

5.2. Trie.h

```
struct trieobject {
 trieobject() {
  children[0] = NULL;
  children[1] = NULL;
  numelems = 0;
 };

 struct trieobject* children[2];
 int numelems;
};
struct trie {
 trieobject base;
 trie() {
  trieobject base;
 }
 void add(int x) {
  int pow2 = (1ll << 31ll);
  trieobject* temp = &base;
  while (pow2 > 0) {
   if (temp->children[1 && (x & pow2)] == NULL) {
    temp->children[1 && (x & pow2)] = new trieobject;
   }
   temp->children[1 && (x & pow2)]->numelems++;
   temp = temp->children[1 && (x & pow2)];
   pow2 /= 2;
  }
 }
 // ADD FUNCTION BELOW
};
```

5.3. SuffixArray.h

```
/*Builds suffix array for a string.
\texttt{sa[i]} is the starting index of the suffix which
is $i$'th in the sorted suffix array.
The returned vector is of size $n+1$, and \texttt{sa[0] = n}.
The \texttt{lcp} array contains longest common prefixes for
neighbouring strings in the suffix array:
\texttt{lcp[i] = lcp(sa[i], sa[i-1])}, \texttt{lcp[0] = 0}.
The input string must not contain any zero bytes.
Time: O(n \log n)*/
#define rep(i, j, k) for (int i = j; i < k; i++)
struct SuffixArray {
 vi sa, lcp;
 SuffixArray(string& s, int lim = 256) {  // or basic_string<int>
  int n = sz(s) + 1, k = 0, a, b;
  vi x(all(s)), y(n), ws(max(n, lim));
```

```
  x.push_back(0), sa = lcp = y, iota(all(sa), 0);
  for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
   p = j, iota(all(y), n - j);
   rep(i, 0, n) if (sa[i] >= j) y[p++] = sa[i] - j;
   fill(all(ws), 0);
   rep(i, 0, n) ws[x[i]]++;
   rep(i, 1, lim) ws[i] += ws[i - 1];
   for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
   swap(x, y), p = 1, x[sa[0]] = 0;
   rep(i, 1, n) a = sa[i - 1], b = sa[i], x[b] = (y[a] == y[b] && y[a + j]
== y[b + j]) ? p - 1 : p++;
  }
  for (int i = 0, j; i < n - 1; lcp[x[i++]] = k)
   for (k &&k--, j = sa[x[i] - 1];
       s[i + k] == s[j + k]; k++)
    ;
 }
};
```

6. Numerical

6.1. NTT.h

```
/* Description: Can be used for convolutions modulo specific nice
primes of the form 2^a b+1, where the convolution result has size at
most 2^a
 * (125000001 << 3) + 1 = 1e9 + 7, therefore do not use this for M =
1e9 + 7.
 * For $p < 2^30$ there is also e.g. (5 << 25, 3), (7 << 26, 3),
 * For other primes/integers, use two different primes and combine
with CRT. (479 << 21, 3) and (483 << 21, 5). The last two are > 10^9
 * Inputs must be in [0, mod).
 */
// Requires mod func
const int M = 998244353;
const int root = 3;
// (119 << 23) + 1, root = 3;  // for M = 998244353
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) % M;
  x[skip * (i + n2)] = (s - t) % M;
 }
}
void ntt(vi& x, bool inv = false) {
 int e = pw(root, (M - 1) / sz(x));
 if (inv) e = pw(e, M - 2);
 vi roots(sz(x), 1), temp = roots;
 for (int i = 1; i < sz(x); i++) roots[i] = roots[i - 1] * e % M;
 ntt(&x[0], &temp[0], &roots[0], sz(x), 1);
}
// Usage: just pass the two coefficients list to get a*b (modulo M)
vi conv(vi a, vi b) {
```

```
 int s = sz(a) + sz(b) - 1;
 if (s <= 0) return {};
 int L = s > 1 ? 32 - __builtin_clzll(s - 1) : 0, n = 1 << L;
 if (s <= 200) {  // (factor 10 optimization for |a|,|b| = 10)
  vi c(s);
  for (int i = 0; i < sz(a); i++)
   for (int j = 0; j < sz(b); j++)
    c[i + j] = (c[i + j] + a[i] * b[j]) % M;
  return c;
 }
 a.resize(n);
 ntt(a);
 b.resize(n);
 ntt(b);
 vi c(n);
 int d = pw(n, M - 2);
 for (int i = 0; i < n; i++) c[i] = a[i] * b[i] % M * d % M;
 ntt(c, true);
 c.resize(s);
 return c;
}
```

6.2. FastFourierTransform.h

```
typedef complex<double> C;
typedef vector<double> vd;

void fft(vector<C>& a) {
 int n = sz(a), L = 31 - __builtin_clz(n);
 static vector<complex<long double>> R(2, 1);
 static vector<C> rt(2, 1);  // (^ 10% faster if double)
 for (static int k = 2; k < n; k *= 2) {
  R.resize(n);
  rt.resize(n);
  auto x = polar(1.0L, acos(-1.0L) / k);
  rep(i, k, 2 * k) rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
 }
 vi rev(n);
 rep(i, 0, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
 rep(i, 0, n) 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) rep(j, 0, k) {
   // C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled)  ///
include-line
   auto x = (double*)&rt[j + k], y = (double*)&a[i + j + k];  ///
exclude-line
   C z(x[0] * y[0] - x[1] * y[1], x[0] * y[1] + x[1] * y[0]);  ///
exclude-line
   a[i + j + k] = a[i + j] - z;
   a[i + j] += z;
  }
}
vd conv(const vd& a, const vd& b) {
 if (a.empty() || b.empty()) return {};
 vd res(sz(a) + sz(b) - 1);
 int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
 vector<C> in(n), out(n);
```

```
  copy(all(a), begin(in));
  rep(i, 0, sz(b)) in[i].imag(b[i]);
  fft(in);
  for (C& x : in) x *= x;
  rep(i, 0, n) out[i] = in[-i & (n - 1)] - conj(in[i]);
  fft(out);
  rep(i, 0, sz(res)) res[i] = imag(out[i]) / (4 * n);
  return res;
}
```

7. Geometry

7.1. ConvexHull.h

```
// Needs point
typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
  if (sz(pts) <= 1) return pts;
  sort(all(pts));
  vector<P> h(sz(pts) + 1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(all(pts)))
    for (P p : pts) {
      while (t >= s + 2 && h[t - 2].cross(h[t - 1], p) <= 0) t--;
      h[t++] = p;
    }
  return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
```

7.2. Point.h

```
template <class T>
int sgn(T x) { return (x > 0) - (x < 0); }
template <class T>
struct Point {
  typedef Point P;
  T x, y;
  explicit Point(T x = 0, T y = 0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x, y) < tie(p.x, p.y); }
  bool operator==(P p) const { return tie(x, y) == tie(p.x, p.y); }
  P operator+(P p) const { return P(x + p.x, y + p.y); }
  P operator-(P p) const { return P(x - p.x, y - p.y); }
  P operator*(T d) const { return P(x * d, y * d); }
  P operator/(T d) const { return P(x / d, y / d); }
  T dot(P p) const { return x * p.x + y * p.y; }
  T cross(P p) const { return x * p.y - y * p.x; }
  T cross(P a, P b) const { return (a - *this).cross(b - *this); }
  T dist2() const { return x * x + y * y; }
  double dist() const { return sqrt((double)dist2()); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this / dist(); }  // makes dist()=1
  P perp() const { return P(-y, x); }        // rotates +90 degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around the origin
  P rotate(double a) const {
    return P(x * cos(a) - y * sin(a), x * sin(a) + y * cos(a));
  }
  friend ostream& operator<<(ostream& os, P p) {
```

```
    return os << "(" << p.x << "," << p.y << ")";
  }
};
```

7.3. ClosestPair.h

```
// Requires point
typedef Point<int> P;
pair<P, P> closest(vector<P> v) {
  assert(sz(v) > 1);
  set<P> S;
  sort(all(v), [](P a, P b) { return a.y < b.y; });
  pair<int, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
  int j = 0;
  for (P p : v) {
    P d{1 + (int)sqrtl(ret.first), 0};
    while (v[j].y <= p.y - d.x) S.erase(v[j++]);
    auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
    for (; lo != hi; ++lo)
      ret = min(ret, {(*lo - p).dist2(), {*lo, p}});
    S.insert(p);
  }
  return ret.second;
}
```