MIST_Untitled

Military Institute of Science and Technology
Yusuf Reza Hasnat | Istiaque Ahmed Arik | Shihab Ahmed

Last modified: December 18, 2025

Contents

1 C++

1.1 template

```
/*
c++:
ios_base::sync_with_stdio(false);
cin.tie(nullptr), cout.tie(nullptr);

python:
import sys
input = sys.stdin.readline
sys.stdout.write("------")
*/
```

1.2 random

```
#define accuracy chrono::steady_clock::
    ↪ now().time_since_epoch().count()
mt19937 rng(accuracy);

ll rand(ll l, ll r) {
  uniform_int_distribution<ll> ludo(l, r
      ↪ );
  return ludo(rng);
}
```

1.3 gp_hash

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename p, typename q> using
    ↪ ht = gp_hash_table<p, q>;
```

1.4 pbds

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T>
using o_set = tree<T, null_type, less<T
    ↪ >, rb_tree_tag,
    ↪ tree_order_statistics_node_update
    ↪ >;
/*
find_by_order(k) - returns an iterator
    ↪ to the k-th largest element (0
    ↪ indexed);
order_of_key(k)- the number of elements
    ↪ in the set that are strictly
    ↪ smaller than k;
*/
```

1.5 debug

```
string to_string(const string &s) {
    ↪ return '"' + s + '"'; }
string to_string(const char *s) { return
    ↪  to_string(string(s)); }
string to_string(const char c) { return
    ↪ "'" + string(1, c) + "'"; }
string to_string(bool b) { return b ? "
    ↪ true" : "false"; }
template <typename A, typename B> string
    ↪  to_string(pair<A, B> p) {
  return "(" + to_string(p.first) + ", "
      ↪  + to_string(p.second) + ")";
}
template <typename A> string to_string(A
    ↪  v) {
  string res = "{";
  for (const auto &x : v) {
    res += to_string(x) + ", ";
  }
  res += "}";
  return res;
}
void debug_out() { cerr << endl; }
template <typename Head, typename...
    ↪ Tail> void debug_out(Head H, Tail
    ↪ ... T) {
  cerr << " " << to_string(H);
  debug_out(T...);
}
#define dbg(...)
    ↪
    ↪ \
  cerr << __LINE__ << ": [" << #
      ↪ __VA_ARGS__ << "] = ", debug_out
      ↪ (__VA_ARGS__)
```

1.6 stress

```
#!/usr/bin/env bash
wrong="solution"
correct="brute"
gen="gen"
g++ -g solution.cpp -DONPC -o "$wrong"
g++ -g brute.cpp -DONPC -o "$correct"
g++ -g gen.cpp -DONPC -o "$gen"

for ((testNum=0;testNum<$1;testNum++))
do
        ./$gen 2>/dev/null > stdinput
        ./$correct < stdinput 2>/dev/
            ↪ null > outSlow
        ./$wrong < stdinput 2>/dev/null
            ↪ > outWrong
        H1=`md5sum outWrong`
        H2=`md5sum outSlow`
        if !(cmp -s "outWrong" "outSlow"
            ↪ )
        then
        echo "Error found!"
        echo "Input:"
        cat stdinput
        echo "Wrong Output:"
        cat outWrong
        echo "Slow Output:"
        cat outSlow
        exit
        fi
done
echo Passed $1 tests
# Usage: ./contest.sh times
```

1.7 vscode

```
{
  "key" : "f5",
  "command" : "workbench.action.terminal
      ↪ .sendSequence",
  "args" : {
    "text" : "g++ ${
        ↪ fileBasenameNoExtension}.cpp -
        ↪ o  ${fileBasenameNoExtension}
        ↪ && ./ ${
        ↪ fileBasenameNoExtension} <in.
        ↪ txt> out.txt\n "
  }
}
```

2 Dsa

2.1 KMP

```
vector<ll> createLPS(string pattern) {
  ll n = pattern.length(), idx = 0;
  vector<ll> lps(n);
  for (ll i = 1; i < n;) {
    if (pattern[idx] == pattern[i]) {
      lps[i] = idx + 1;
      idx++, i++;
    } else {
      if (idx != 0)
        idx = lps[idx - 1];
      else
        lps[i] = idx, i++;
    }
  }
  return lps;
}
ll kmp(string text, string pattern) {
  ll cnt_of_match = 0, i = 0, j = 0;
  vector<ll> lps = createLPS(pattern);
  while (i < text.length()) {
```

```
    if (text[i] == pattern[j])
      i++, j++; // i = text, j = pattern
    else {
      if (j != 0)
        j = lps[j - 1];
      else
        i++;
    }
    if (j == pattern.length()) {
      cnt_of_match++;
      // the index where match found ->
          ↪ (i - pattern.length());
      j = lps[j - 1];
    }
  }
  return cnt_of_match;
}
```

2.2 Hashing

```
const ll N = 2e5 + 5;
const ll MOD1 = 127657753, MOD2 =
    ↪ 987654319;
const ll p1 = 137, p2 = 277;
ll ip1, ip2;
pair<ll, ll> pw[N], ipw[N];
void prec() {
  pw[0] = {1, 1};
  for (ll i = 1; i < N; i++) {
    pw[i].first = 1LL * pw[i - 1].first
        ↪ * p1 % MOD1;
    pw[i].second = 1LL * pw[i - 1].
        ↪ second * p2 % MOD2;
  }
  ip1 = binaryExp(p1, MOD1 - 2, MOD1);
  ip2 = binaryExp(p2, MOD2 - 2, MOD2);
  ipw[0] = {1, 1};
  for (ll i = 1; i < N; i++) {
    ipw[i].first = 1LL * ipw[i - 1].
        ↪ first * ip1 % MOD1;
    ipw[i].second = 1LL * ipw[i - 1].
        ↪ second * ip2 % MOD2;
  }
}
struct Hashing {
  ll n;
  string s;                    // 0 -
      ↪ indexed
  vector<pair<ll, ll>> hs; // 1 -
      ↪ indexed
  Hashing() {}
  Hashing(string _s) {
    n = _s.size();
    s = _s;
    hs.emplace_back(0, 0);
    for (ll i = 0; i < n; i++) {
      pair<ll, ll> p;
      p.first = (hs[i].first + 1LL * pw[
          ↪ i].first * s[i] % MOD1) %
          ↪ MOD1;
      p.second = (hs[i].second + 1LL *
          ↪ pw[i].second * s[i] % MOD2)
          ↪ % MOD2;
      hs.push_back(p);
    }
  }
  pair<ll, ll> get_hash(ll l, ll r) {
    // 1 - indexed
    assert(1 <= l && l <= r && r <= n);
    pair<ll, ll> ans;
    ans.first =
        (hs[r].first - hs[l - 1].first +
            ↪  MOD1) * 1LL * ipw[l - 1].
            ↪ first % MOD1;
    ans.second = (hs[r].second - hs[l -
        ↪ 1].second + MOD2) * 1LL *
                  ipw[l - 1].second %
                      ↪ MOD2;
    return ans;
  }
  pair<ll, ll> get_hash() { return
      ↪ get_hash(1, n); }
};
```

2.3 BigInteger

```
struct BigInteger {
  string str;
  // Constructor to initialize
  // BigInteger with a string
  BigInteger(string s) { str = s; }
  // Overload + operator to add
  // two BigInteger objects
  BigInteger operator+(const BigInteger
      ↪ &b) {
    string a = str, c = b.str;
    ll alen = a.length(), clen = c.
        ↪ length();
    ll n = max(alen, clen);
    if (alen > clen)
      c.insert(0, alen - clen, '0');
    else if (alen < clen)
      a.insert(0, clen - alen, '0');
    string res(n + 1, '0');
    ll carry = 0;
    for (ll i = n - 1; i >= 0; i--) {
            ll digit=(a[i -'0')+(c[i]-'0
                ↪ ')
    +carry;
            carry = digit / 10;
            res[i + 1] = digit % 10 + '0
                ↪ ';
    }
    if (carry == 1) {
            res[0] = '1';
            return BigInteger(res);
    } else
            return BigInteger(res.substr
                ↪ (1));
  }
  // Overload - operator to subtract
  // first check which number is greater
      ↪  and then subtract
  BigInteger operator-(const BigInteger
      ↪ &b) {
    string a = str;
    string c = b.str;
    ll alen = a.length(), clen = c.
        ↪ length();
    ll n = max(alen, clen);
    if (alen > clen)
            c.insert(0, alen - clen, '0'
                ↪ );
    else if (alen < clen)
            a.insert(0, clen - alen, '0'
                ↪ );
    if (a < c) {
            swap(a, c);
            swap(alen, clen);
    }
    string res(n, '0');
    ll carry = 0;
    for (ll i = n - 1; i >= 0; i--) {
            ll digit = (a[i] - '0') - (c
                ↪ [i] - '0') - carry;
            if (digit < 0)
              digit += 10, carry = 1;
            else
              carry = 0;
            res[i] = digit + '0';
    }
    // remove leading zeros
    ll i = 0;
    while (i < n && res[i] == '0')
            i++;
    if (i == n)
            return BigInteger("0");
    return BigInteger(res.substr(i));
  }
  // Overload * operator to multiply
  // two BigInteger objects
  BigInteger operator*(const BigInteger
      ↪ &b) {
    string a = str, c = b.str;
    ll alen = a.length(), clen = c.
```

```
        ↪ length();
    ll n = alen + clen;
    string res(n, '0');
    for (ll i = alen - 1; i >= 0; i--) {
            ll carry = 0;
            for (ll j = clen - 1; j >=
                ↪ 0; j--) {
              ll digit =
                  (a[i] - '0') * (c[j -
                      ↪ '0']) + (res[i +
                      ↪  j + 1] - '0') +
                      ↪  carry;
              carry = digit / 10;
              res[i + j + 1] = digit %
                  ↪ 10 + '0';
            }
            res[i] += carry;
    }
    ll i = 0;
    while (i < n && res[i] == '0')
            i++;
    if (i == n)
            return BigInteger("0");
    return BigInteger(res.substr(i));
  }
  // Overload << operator to output
  // BigInteger object
  friend ostream &operator<<(ostream &
      ↪ out, const BigInteger &b) {
    out << b.str;
    return out;
  }
};
```

2.4 Kadane

```
// return maximum subarray sum.
ll kadense(ll arr[], ll n) {
  ll mxsm = arr[0], curr_s = arr[0];
  for (ll i = 1; i < n; i++) {
    curr_s = max(arr[i], curr_s + arr[i
        ↪ ]);
    mxsm = max(mxsm, curr_s);
  }
  return mxsm;
}
```

2.5 Segement_tree

```
class SEGMENT_TREE {
public:
  vector<ll> v;
  vector<ll> seg;
  SEGMENT_TREE(ll n) {
    v.resize(n + 5);
    seg.resize(4 * n + 5);
  }
  //! initially: ti = 1, low = 1, high =
      ↪  n (number of elements in the
      ↪ array);
  void build(ll ti, ll low, ll high) {
    if (low == high) {
      seg[ti] = v[low];
      return;
    }
    ll mid = (low + high) / 2;
    build(2 * ti, low, mid);
    build(2 * ti + 1, mid + 1, high);
    seg[ti] = (seg[2 * ti] + seg[2 * ti
        ↪ + 1]);
  }
  //! initially: ti = 1, low = 1, high =
      ↪  n (number of elements in the
      ↪ array), (ql & qr)=user input in
      ↪ 1 based index;
  ll find(ll ti, ll tl, ll tr, ll ql, ll
      ↪  qr) {
    if (tl > qr || tr < ql) {
      return 0;
    }
    if (tl >= ql and tr <= qr)
      return seg[ti];
    ll mid = (tl + tr) / 2;
    ll l = find(2 * ti, tl, mid, ql, qr)
        ↪ ;
    ll r = find(2 * ti + 1, mid + 1, tr,
        ↪  ql, qr);
    return (l + r);
  }
  //! initially: ti = 1, tl = 1, tr = n
      ↪ (number of elements in the array
      ↪ ), id = user input in 1 based
      ↪ indexing, val = updated value;
  void update(ll ti, ll tl, ll tr, ll id
      ↪ , ll val) {
    if (id > tr or id < tl)
      return;
    if (id == tr and id == tl) {
      seg[til] = val;
      return;
    }
    ll mid = (tl + tr) / 2;
    update(2 * ti, tl, mid, id, val);
    update(2 * ti + 1, mid + 1, tr, id,
        ↪ val);
    seg[ti] = (seg[2 * ti] + seg[2 * ti
        ↪ + 1]);
  }
};
// use 1 based indexing;
```

2.6 Fenwick_tree

```
struct FenwickTree {
  vector<ll> bit; // binary indexed tree
  ll n;
  FenwickTree(ll n) {
    this->n = n;
    bit.assign(n, 0);
  }
  FenwickTree(vector<ll> a) :
      ↪ FenwickTree(a.size()) {
    for (size_t i = 0; i < a.size(); i
        ↪ ++)
      add(i, a[i]);
  }
  ll sum(ll r) {
    ll ret = 0;
    for (; r >= 0; r = (r & (r + 1)) -
        ↪ 1)
      ret += bit[r];
    return ret;
  }
  ll sum(ll l, ll r) { return sum(r) -
      ↪ sum(l - 1); }
  void add(ll idx, ll delta) {
    for (; idx < n; idx = idx | (idx +
        ↪ 1))
      bit[idx] += delta;
  }
};

// minimum
struct FenwickTreeMin {
  vector<ll> bit;
  ll n;
  const ll INF = (ll)1e9;
  FenwickTreeMin(ll n) {
    this->n = n;
    bit.assign(n, INF);
  }
  FenwickTreeMin(vector<ll> a) :
      ↪ FenwickTreeMin(a.size()) {
    for (size_t i = 0; i < a.size(); i
        ↪ ++)
      update(i, a[i]);
  }
  ll getmin(ll r) {
    ll ret = INF;
    for (; r >= 0; r = (r & (r + 1)) -
        ↪ 1)
      ret = min(ret, bit[r]);
    return ret;
  }
  void update(ll idx, ll val) {
    for (; idx < n; idx = idx | (idx +
        ↪ 1))
      bit[idx] = min(bit[idx], val);
```

```
  }
};
```

2.7 Segment_tree_lazy

```
class SEGMENT_TREE {
public:
  vector<ll> v;
  vector<ll> seg;
  vector<ll> lazy;
  SEGMENT_TREE(ll n) {
    v.resize(n + 5, 0);
    seg.resize(4 * n + 5, 0);
    lazy.resize(4 * n + 5, 0);
  }
  void pull(ll ti) { seg[ti] = (seg[2 *
      ↪ ti] & seg[2 * ti + 1]); }
  void push(ll ti, ll tl, ll tr) {
    if (lazy[ti] == 0)
      return;
    seg[ti] |= lazy[ti];
    if (tl != tr) {
      lazy[2 * ti] |= lazy[ti];
      lazy[2 * ti + 1] |= lazy[ti];
    }
    lazy[ti] = 0;
  }
  //! llially: ti = 1, low = 1, high = n
      ↪ (number of elements in the array
      ↪ );
  void build(ll ti, ll low, ll high) {
    lazy[ti] = 0;
    if (low == high) {
      seg[ti] = v[low];
      return;
    }
    ll mid = (low + high) / 2;
    build(2 * ti, low, mid);
    build(2 * ti + 1, mid + 1, high);
    pull(ti);
  }
  //! llially: ti = 1, low = 1, high = n
      ↪ (number of elements in the array
      ↪ ), (ql
  //! & qr) = user input in 1 based
      ↪ indexing;
  ll query(ll ti, ll tl, ll tr, ll ql,
      ↪ ll qr) {
    push(ti, tl, tr);
    if (tl > qr || tr < ql) {
      return (1LL << 32) - 1;
    }
    if (tl >= ql and tr <= qr)
      return seg[ti];
    ll mid = (tl + tr) / 2;
    ll l = query(2 * ti, tl, mid, ql, qr
        ↪ );
    ll r = query(2 * ti + 1, mid + 1, tr
        ↪ , ql, qr);
    return (l & r);
  }
  //! llially: ti = 1, tl = 1, tr = n(
      ↪ number of elements in the array)
      ↪ , id =
  //! user input in 1 based indexing,
      ↪ val = updated value;
  void update(ll ti, ll tl, ll tr, ll
      ↪ idL, ll idR, ll val) {
    push(ti, tl, tr);
    if (idR < tl or tr < idL)
      return;
    if (idL <= tl and tr <= idR) {
      lazy[ti] |= val;
      push(ti, tl, tr);
      return;
    }
    ll mid = (tl + tr) / 2;
    update(2 * ti, tl, mid, idL, idR,
        ↪ val);
    update(2 * ti + 1, mid + 1, tr, idL,
        ↪  idR, val);
    pull(ti);
  }
  // use 1 based indexing for input and
      ↪ queries and update;
};
```

2.8 Trie

```
const ll N = 26;
class Node {
public:
  ll EoW;
  Node *child[N];
  Node() {
    EoW = 0;
    for (ll i = 0; i < N; i++)
      child[i] = NULL;
  }
};

void insert(Node *node, string s) {
  for (size_t i = 0; i < s.size(); i++)
      ↪ {
    ll r = s[i] - 'A';
    if (node->child[r] == NULL)
      node->child[r] = new Node();
    node = node->child[r];
  }
  node->EoW += 1;
}
ll search(Node *node, string s) {
  for (size_t i = 0; i < s.size(); i++)
      ↪ {
    ll r = s[i] - 'A';
    if (node->child[r] == NULL)
      return 0;
  }
  return node->EoW;
}

void prll(Node *node, string s = "") {
  if (node->EoW)
    cout << s << "\n";
  for (ll i = 0; i < N; i++) {
    if (node->child[i] != NULL) {
      char c = i + 'A';
      prll(node->child[i], s + c);
    }
  }
}

bool isChild(Node *node) {
  for (ll i = 0; i < N; i++)
    if (node->child[i] != NULL)
      return true;
  return false;
}

bool isJunc(Node *node) {
  ll cnt = 0;
  for (ll i = 0; i < N; i++) {
    if (node->child[i] != NULL)
      cnt++;
  }
  if (cnt > 1)
    return true;
  return false;
}

ll trie_delete(Node *node, string s, ll
    ↪ k = 0) {
  if (node == NULL)
    return 0;
  if (k == (ll)s.size()) {
    if (node->EoW == 0)
      return 0;
    if (isChild(node)) {
      node->EoW = 0;
      return 0;
    }
    return 1;
  }
  ll r = s[k] - 'A';
  ll d = trie_delete(node->child[r], s,
      ↪ k + 1);
  ll j = isJunc(node);
  if (d)
    delete node->child[r];
  if (j)
```

```
    return 0;
  return d;
}
void delete_trie(Node *node) {
  for (ll i = 0; i < 15; i++) {
    if (node->child[i] != NULL)
      delete_trie(node->child[i]);
  }
  delete node;
}
```

2.9 DSU

```
class DisjollSet {
  vector<ll> par, sz, minElmt, maxElmt,
      ↪ cntElmt;
public:
  DisjollSet(ll n) {
    par.resize(n + 1);
    sz.resize(n + 1, 1);
    minElmt.resize(n + 1);
    maxElmt.resize(n + 1);
    cntElmt.resize(n + 1, 1);
    for (ll i = 1; i <= n; i++)
      par[i] = minElmt[i] = maxElmt[i] =
          ↪  i;
  }
  ll findUPar(ll u) {
    if (u == par[u])
      return u;
    return par[u] = findUPar(par[u]);
  }
  void unionBySize(ll u, ll v) {
    ll pU = findUPar(u);
    ll pV = findUPar(v);
    if (pU == pV)
      return;
    if (sz[pU] < sz[pV])
      swap(pU, pV);
    par[pV] = pU;
    sz[pU] += sz[pV];
    cntElmt[pU] += cntElmt[pV];
    minElmt[pU] = min(minElmt[pU],
        ↪ minElmt[pV]);
    maxElmt[pU] = max(maxElmt[pU],
        ↪ maxElmt[pV]);
  }
  ll getMinElementIntheSet(ll u) {
      ↪ return minElmt[findUPar(u)]; }
  ll getMaxElementIntheSet(ll u) {
      ↪ return maxElmt[findUPar(u)]; }
  ll getNumofElementIntheSet(ll u) {
      ↪ return cntElmt[findUPar(u)]; }
};
```

2.10 HLD

```
ll par[N], sub_tree_sz[N], heavy[N],
    ↪ wt_from_parent[N], depth[N], head[
    ↪ N],
    position[N];
vector<pair<ll, ll>> gd[N];

// HLD part start
ll dfs(ll node, ll p) {
  par[node] = p;
  sub_tree_sz[node] = 1;
  heavy[node] = -1;

  for (auto [v, w] : gd[node]) {
    if (v == p)
      continue;
    depth[v] = depth[node] + 1;
    wt_from_parent[v] = w;
    sub_tree_sz[node] += dfs(v, node);
    if (heavy[node] == -1 || sub_tree_sz
        ↪ [v] > sub_tree_sz[heavy[node
        ↪ ]]) {
      heavy[node] = v;
    }
  }
  return sub_tree_sz[node];
}
ll pos;
void decompose(ll node, ll hd) {
  head[node] = hd;
  position[node] = ++pos;
  if (heavy[node] != -1) {
    decompose(heavy[node], hd);
  }
  for (auto [v, w] : gd[node]) {
    if (v != par[node] && v != heavy[
        ↪ node]) {
      decompose(v, v);
    }
  }
}

// HLD part end

// in main function
ll n, m;
cin >> n;
SEGMENT_TREE seg(n); // Lazy if needed
vector<ll> edge_u(n), edge_v(n),
    ↪ edge_node(n);

for (int i = 1; i < n; i++) {
  ll u, v, wt = 1;
  cin >> u >> v >> wt;
  gd[u].push_back({v, wt});
  gd[v].push_back({u, wt});
  edge_u[i] = u;
  edge_v[i] = v;
}

dfs(1, -1);
pos = 0;
decompose(1, 1);

for (int i = 1; i <= n; i++) {
  // seg.v[position[i]] = val[i];  //
      ↪ for node value
  seg.v[position[i]] = wt_from_parent[i
      ↪ ]; // for edge value
}

// work on a specific edge
for (int i = 1; i < n; i++) {
  ll u = edge_u[i], v = edge_v[i];
  edge_node[i] = (depth[u] > depth[v]) ?
      ↪  u : v;
}

seg.build(1, 1, n);

auto updatePath = [&](ll u, ll v, ll x)
    ↪ {
  while (head[u] != head[v]) {
    if (depth[head[u]] < depth[head[v]])
      swap(u, v);
    seg.update(1, 1, n, position[head[u
        ↪ ]], position[u], x);
    u = par[head[u]];
  }
  if (depth[u] > depth[v])
    swap(u, v);
  // edge value
  if (u != v) {
    seg.update(1, 1, n, position[u] + 1,
        ↪  position[v], x);
  }
  // node value
  // seg.update(1, 1, n, position[u],
      ↪ position[v], x);
};

auto querypath = [&](ll u, ll v) {
  ll ans = -inf;
  while (head[u] != head[v]) {
    if (depth[head[u]] < depth[head[v]])
      swap(u, v);
    ans = max(ans, seg.query(1, 1, n,
        ↪ position[head[u]], position[u
        ↪ ]));
    u = par[head[u]];
  }
  if (depth[u] > depth[v])
    swap(u, v);
  // upward + downward
  if (u != v) {
    ans = max(ans, seg.query(1, 1, n,
```

```
        ↪ position[u] + 1, position[v]))
        ↪ ;
  }
  // only upward
//   ans = max(ans, seg.query(1, 1, n,
    ↪ position[u], position[v])); // for
    ↪  node value
  return ans;
};
seg.update(1, 1, n, position[edge_node[s
    ↪ ]], position[edge_node[s]], x); //
    ↪  single point update. if path
    ↪ update need call update path
cout << querypath(x, s) << '\n';
```

2.11 Manacher

```
struct Manacher {
  vector<ll> p[2];
  string s;
  // p[1][i] = (max odd length
     ↪ palindrome centered at i) / 2 [
     ↪ floor division]
  // p[0][i] = same for even, it
     ↪ considers the right center
  // e.g. for s = "abbabba", p[1][3] =
     ↪ 3, p[0][2] = 2
  Manacher(string s) {
    this->s = s;
    ll n = s.size();
    p[0].resize(n + 1);
    p[1].resize(n);
    for (ll z = 0; z < 2; z++) {
      for (ll i = 0, l = 0, r = 0; i < n
         ↪ ; i++) {
        ll t = r - i + !z;
        if (i < r)
          p[z][i] = min(t, p[z][l + t]);
        ll 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;
      }
    }
  }
  bool is_palindrome(ll l, ll r) {
    ll mid = (l + r + 1) / 2, len = r -
       ↪ l + 1;
    return 2 * p[len % 2][mid] + len % 2
       ↪  >= len;
  }
  string get_palin(ll i, bool odd = true
     ↪ ) {
    ll len = p[odd][i];
    return s.substr(i - len, 2 * len + 1
       ↪  - !odd);
  }
};
```

2.12 2D prefix Sum

```
pref[i][j] = a[i][j] + pref[i - 1][j] +
    ↪ pref[i][j - 1] - pref[i - 1][j -
    ↪ 1];
Sum of region = pref[row2 + 1][col2 + 1]
    ↪  - pref[row2 + 1][col1] - pref[
    ↪ row1][col2 + 1] + pref[row1][col1
    ↪ ];
```

2.13 CRT

```
class CRT {
  typedef long long vlong;
  typedef pair<vlong, vlong> pll;
  vector<pll> equations;

public:
  void clear() { equations.clear(); }
  vlong extended_euclid(vlong a, vlong b
     ↪ , vlong &x, vlong &y) {
    if (b == 0) {
      x = 1;
      y = 0;
      return a;
    }
    vlong x1, y1;
    vlong d = extended_euclid(b, a % b,
       ↪ x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
  }
  vlong inverse(vlong a, vlong m) {
    vlong x, y;
    vlong g = extended_euclid(a, m, x, y
       ↪ );
    if (g != 1)
      return -1;
    return (x % m + m) % m;
  }
  /** Add equation of the form x = r (
     ↪ mod m) */
  void addEquation(vlong r, vlong m) {
     ↪ equations.push_back({r, m}); }
  pll solve() {
    if (equations.size() == 0)
      return {-1, -1};

    vlong a1 = equations[0].first;
    vlong m1 = equations[0].second;
    a1 %= m1;
    for (int i = 1; i < equations.size()
       ↪ ; i++) {
      vlong a2 = equations[i].first;
      vlong m2 = equations[i].second;

      vlong g = __gcd(m1, m2);
      if (a1 % g != a2 % g)
        return {-1, -1};
      vlong p, q;
      extended_euclid(m1 / g, m2 / g, p,
         ↪  q);

      vlong mod = m1 / g * m2;
      vlong x = ((__int128)a1 * (m2 / g)
         ↪  % mod * q % mod +
                (__int128)a2 * (m1 / g)
                   ↪  % mod * p % mod)
                   ↪  %
               mod;
      a1 = x;
      if (a1 < 0)
        a1 += mod;
      m1 = mod;
    }
    return {a1, m1};
  }
};
```

2.14 Intersect two arithmetic progression

```
using T = __int128;
// ax + by = __gcd(a, b)
// returns __gcd(a, b)
T extended_euclid(T a, T b, T &x, T &y)
   ↪ {
  T xx = y = 0;
  T yy = x = 1;
  while (b) {
    T q = a / b;
    T t = b;
    b = a % b;
    a = t;
    t = xx;
    xx = x - q * xx;
    x = t;
    t = yy;
    yy = y - q * yy;
    y = t;
  }
  return a;
}
pair<T, T> CRT(T a1, T m1, T a2, T m2) {
  T p, q;
  T g = extended_euclid(m1, m2, p, q);
```

```
  if (a1 % g != a2 % g)
    return make_pair(0, -1);
  T m = m1 / g * m2;
  p = (p % m + m) % m;
  q = (q % m + m) % m;
  return make_pair((p * a2 % m * (m1 / g
      ↪ ) % m + q * a1 % m * (m2 / g) %
      ↪ m) % m,  m);
}
// intersecting AP of two APs: (a1 + d1x
    ↪ ) and (a2 + d2x)
pair<ll, ll> intersect(ll a1, ll d1, ll
    ↪ a2, ll d2) {
  auto x = CRT(a1 % d1, d1, a2 % d2, d2)
      ↪ ;
  ll a = x.first, d = x.second;
  if (d == -1)
    return {0, 0}; // empty
  ll st = max(a1, a2);
  a = a < st ? a + ((st - a + d - 1) / d
      ↪ ) : a; // while (a < st) a += d;
  return {a, d};
}
```

2.15 Find nth value in a recurrence relation in O(logn

```
[ 1, 1; 1, 0 ] ^ (n - 1) =
    [F(n), F(n - 1); F(n - 1), F(n - 2)]
    // Function to multiply two 2x2
        ↪ matrices
    void multiply(vector<vector<int>> &
        ↪ mat1, vector<vector<int>> &
        ↪ mat2) {
      // Perform matrix multiplication
      int x = mat1[0][0] * mat2[0][0] +
          ↪ mat1[0][1] * mat2[1][0];
      int y = mat1[0][0] * mat2[0][1] +
          ↪ mat1[0][1] * mat2[1][1];
      int z = mat1[1][0] * mat2[0][0] +
          ↪ mat1[1][1] * mat2[1][0];
      int w = mat1[1][0] * mat2[0][1] +
          ↪ mat1[1][1] * mat2[1][1];

      // Update matrix mat1 with the
          ↪ result
      mat1[0][0] = x;
      mat1[0][1] = y;
      mat1[1][0] = z;
      mat1[1][1] = w;
    }

// Function to perform matrix
    ↪ exponentiation
void matrixPower(vector<vector<int>> &
    ↪ mat1, int n) {
  // Base case for recursion
  if (n == 0 || n == 1)
    return;

  // Initialize a helper matrix
  vector<vector<int>> mat2 = {{1, 1},
      ↪ {1, 0}};

  // Recursively calculate mat1^(n/2)
  matrixPower(mat1, n / 2);

  // Square the matrix mat1
  multiply(mat1, mat1);

  // If n is odd, multiply by the helper
      ↪  matrix mat2
  if (n % 2 != 0) {
    multiply(mat1, mat2);
  }
}

// Function to calculate the nth
    ↪ Fibonacci number
// using matrix exponentiation
int nthFibonacci(int n) {
  if (n <= 1)
    return n;

  // Initialize the transformation
      ↪ matrix
  vector<vector<int>> mat1 = {{1, 1},
      ↪ {1, 0}};

  // Raise the matrix mat1 to the power
      ↪ of (n - 1)
  matrixPower(mat1, n - 1);

  // The result is in the top-left cell
      ↪ of the matrix
  return mat1[0][0];
}
```

2.16 All_solution_of_ax+by_equal_c

```
// a*x+b*y=c. returns valid x and y if
    ↪ possible.
// all solutions are of the form (x0 + k
    ↪  * b / g, y0 - k * b / g)
bool find_any_solution(ll a, ll b, ll c,
    ↪  ll &x0, ll &y0, ll &g) {
  if (a == 0 and b == 0) {
    if (c)
      return false;
    x0 = y0 = g = 0;
    return true;
  }
  g = extended_euclid(abs(a), abs(b), x0
      ↪ , y0);
  if (c % g != 0)
    return false;
  x0 *= c / g;
  y0 *= c / g;
  if (a < 0)
    x0 *= -1;
  if (b < 0)
    y0 *= -1;
  return true;
}
void shift_solution(ll &x, ll &y, ll a,
    ↪ ll b, ll cnt) {
  x += cnt * b;
  y -= cnt * a;
}
// returns the number of solutions where
    ↪  x is in the range[minx, maxx] and
    ↪  y is
// in the range[miny, maxy]
ll find_all_solutions(ll a, ll b, ll c,
    ↪ ll minx, ll maxx, ll miny, ll maxy
    ↪ ) {
  ll x, y, g;
  if (find_any_solution(a, b, c, x, y, g
      ↪ ) == 0)
    return 0;
  if (a == 0 and b == 0) {
    assert(c == 0);
    return 1LL * (maxx - minx + 1) * (
        ↪ maxy - miny + 1);
  }
  if (a == 0) {
    return (maxx - minx + 1) * (miny <=
        ↪ c / b and c / b <= maxy);
  }
  if (b == 0) {
    return (maxy - miny + 1) * (minx <=
        ↪ c / a and c / a <= maxx);
  }
  a /= g, b /= g;
  ll sign_a = a > 0 ? +1 : -1;
  ll sign_b = b > 0 ? +1 : -1;
  shift_solution(x, y, a, b, (minx - x)
      ↪ / b);
  if (x < minx)
    shift_solution(x, y, a, b, sign_b);
  if (x > maxx)
    return 0;
  ll lx1 = x;
  shift_solution(x, y, a, b, (maxx - x)
      ↪ / b);
  if (x > maxx)
    shift_solution(x, y, a, b, -sign_b);
  ll rx1 = x;
  shift_solution(x, y, a, b, -(miny - y)
      ↪  / a);
  if (y < miny)
    shift_solution(x, y, a, b, -sign_a);
```

```
  if (y > maxy)
    return 0;
  ll lx2 = x;
  shift_solution(x, y, a, b, -(maxy - y)
      ↪  / a);
  if (y > maxy)
    shift_solution(x, y, a, b, sign_a);
  ll rx2 = x;
  if (lx2 > rx2)
    swap(lx2, rx2);
  ll lx = max(lx1, lx2);
  ll rx = min(rx1, rx2);
  if (lx > rx)
    return 0;
  return (rx - lx) / abs(b) + 1;
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int t, cs = 0;
  cin >> t;
  while (t--) {
    ll a, b, c, x1, x2, y1, y2;
    cin >> a >> b >> c >> x1 >> x2 >> y1
        ↪  >> y2;
    cout << "Case " << ++cs << ": "
         << find_all_solutions(a, b, -c,
             ↪  x1, x2, y1, y2) << '\n';
  }
  return 0;
}
```

2.17 all soln of linear eq

```
struct Combi {
  int n;
  vector<ll> facts, finvs, invs;
  Combi(int _n) : n(_n), facts(_n),
      ↪ finvs(_n), invs(_n) {
    facts[0] = finvs[0] = 1;
    invs[1] = 1;
    for (int i = 2; i < n; i++)
      invs[i] = invs[mod % i] * (-mod /
          ↪ i);
    for (int i = 1; i < n; i++) {
      facts[i] = facts[i - 1] * i;
      finvs[i] = finvs[i - 1] * invs[i];
    }
  }
  inline ll fact(int n) { return facts[n
      ↪ ]; }
  inline ll finv(int n) { return finvs[n
      ↪ ]; }
  inline ll inv(int n) { return invs[n];
      ↪  }
  inline ll ncr(int n, int k) {
    return n < k ? 0 : facts[n] * finvs[
        ↪ k] * finvs[n - k];
  }
};
Combi C(N);
// returns the number of solutions to
    ↪ the equation
// x_1 + x_2 + ... + x_n = s and 0 <= l
    ↪ <= x_i <= r
ll yo(int n, int s, int l, int r) {
  if (s < l * n)
    return 0;
  s -= l * n;
  r -= l;
  ll ans = 0;
  for (int k = 0; k <= n; k++) {
    ll cur = C.ncr(s - k - k * r + n - 1
        ↪  + 1, n - 1 + 1) * C.ncr(n, k)
        ↪ ;
    if (k & 1)
      ans -= cur;
    else
      ans += cur;
  }
  return ans;
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  cout << yo(3, 3, 0, 1) << '\n';
  return 0;
}
```

2.18 Subset sum sqrt(n)

```
// Sum of elements <= N implies that
    ↪ every element is <= N
vector<int> freq(N + 1, 0);
for (int i = 0; i < N; i++) {
  int x;
  cin >> x;
  freq[x]++;
}
vector<pair<int, int>> compressed;
for (int i = 1; i <= N; i++) {
  if (freq[i] > 0)
    compressed.emplace_back(i, freq[i]);
}
vector<int> dp(N + 1, 0);
dp[0] = 1;
for (const auto &[w, k] : compressed) {
  vector<int> ndp = dp;
  for (int p = 0; p < w; p++) {
    int sum = 0;
    for (int multiple = p, count = 0;
        ↪ multiple <= N; multiple += w,
        ↪ count++) {
      if (count > k) {
        sum -= dp[multiple - w * count];
        count--;
      }
      if (sum > 0)
        ndp[multiple] = 1;
      sum += dp[multiple];
    }
  }
  swap(dp, ndp);
}
cout << "Possible subset sums are:\n";
for (int i = 0; i <= N; i++) {
  if (dp[i] > 0)
    cout << i << " ";
}
}
```

2.19 small giant (a^x = b mod m, find x, given other)

```
// Returns minimum x for which a ^ x % m
    ↪  = b % m, a and m are coprime.
int solve(int a, int b, int m) {
  a %= m, b %= m;
  int n = sqrt(m) + 1;
  int an = 1;
  for (int i = 0; i < n; ++i)
    an = (an * 1ll * a) % m;
  unordered_map<int, int> vals;
  for (int q = 0, cur = b; q <= n; ++q)
      ↪ {
    vals[cur] = q;
    cur = (cur * 1ll * a) % m;
  }
  for (int p = 1, cur = 1; p <= n; ++p)
      ↪ {
    cur = (cur * 1ll * an) % m;
    if (vals.count(cur)) {
      int ans = n * p - vals[cur];
      return ans;
    }
  }
  return -1;
}
// Returns minimum x for which a ^ x % m
    ↪  = b % m.
```

```
int solve(int a, int b, int m) {
  a %= m, b %= m;
  int k = 1, add = 0, g;
  while ((g = gcd(a, m)) > 1) {
    if (b == k)
      return add;
    if (b % g)
      return -1;
    b /= g, m /= g, ++add;
    k = (k * 1ll * a / g) % m;
  }
  int n = sqrt(m) + 1;
  int an = 1;
  for (int i = 0; i < n; ++i)
    an = (an * 1ll * a) % m;
  unordered_map<int, int> vals;
  for (int q = 0, cur = b; q <= n; ++q)
    ↪ {
    vals[cur] = q;
    cur = (cur * 1ll * a) % m;
  }
  for (int p = 1, cur = k; p <= n; ++p)
    ↪ {
    cur = (cur * 1ll * an) % m;
    if (vals.count(cur)) {
      int ans = n * p - vals[cur] + add;
      return ans;
    }
  }
  return -1;
}
```

2.20 Gaussian Elimination

```
class GaussianElimination {
public:
  GaussianElimination(vector<vector<
      ↪ double>> matrix, vector<double>
      ↪ results)
      : matrix(matrix), results(results)
          ↪ , n(matrix.size()) {}
  void solve() {
    fElim();
    bSub();
  }
  vector<vector<double>> matrix;
  vector<double> results, solution;
  ll n;
  void fElim() {
    for (ll i = 0; i < n; ++i) {
      ll maxRow = i;
      for (ll k = i + 1; k < n; ++k)
        if (abs(matrix[k][i]) > abs(
            ↪ matrix[maxRow][i]))
          maxRow = k;
      swap(matrix[i], matrix[maxRow]);
      swap(results[i], results[maxRow]);
      for (ll k = i + 1; k < n; ++k) {
        double factor = matrix[k][i] /
            ↪ matrix[i][i];
        for (ll j = i; j < n; ++j)
          matrix[k][j] -= factor *
              ↪ matrix[i][j];
        results[k] -= factor * results[i
            ↪ ];
      }
    }
  }
  void bSub() {
    solution.resize(n);
    for (ll i = n - 1; i >= 0; --i) {
      solution[i] = results[i];
      for (ll j = i + 1; j < n; ++j)
        solution[i] -= matrix[i][j] *
            ↪ solution[j];
      solution[i] /= matrix[i][i];
    }
  }
};
```

2.21 Grundy

```
ll calculateGrundy(ll n, vector<ll> &
    ↪ grundy, const vector<ll> &moves) {
  if (grundy[n] != -1)
    return grundy[n];
  unordered_set<ll> s;
  for (ll move : moves) {
    if (n >= move) {
      s.insert(calculateGrundy(n - move,
          ↪  grundy, moves));
    }
  }
  ll g = 0;
  while (s.count(g))
    g++;
  return grundy[n] = g;
}
vector<ll> computeGrundy(ll maxN, const
    ↪ vector<ll> &moves) {
  vector<ll> grundy(maxN + 1, -1);
  grundy[0] = 0;
  for (ll i = 1; i <= maxN; ++i) {
    calculateGrundy(i, grundy, moves);
  }
  return grundy;
}
/*
1. Nim = all xor
2. Misere Nim = Nim + corner case: if
    ↪ all piles are 1,reverse(nim)
3. Bogus Nim = Nim
4. Staircase Nim = Odd indexed pile Nim
    ↪ (Even indexed pile doesnt matter,
    ↪ as one player can give bogus moves
    ↪  to drop all even piles to ground)
5. Sprague Grundy = Every impartial game
    ↪  under the normal play convention
    ↪ is equivalent to a one-heap game
    ↪ of nim
*/
```

3 Dynamic Programming

3.1 LCS

```
/*
Fact about LCS:
1. Longest Increasing Substring
To solve this, we just care about when
    ↪ two char equals. Rest of the
    ↪ things should be neglected.
2. Longest Palindromic Subsequence(LPS)
To solve this, we just take a new string
    ↪  which is the reverse of the
    ↪ original string. Then just call
    ↪ the LCS function to find LPS.
3. Minimum insertions to make a string
    ↪ palindrome To solve this, we just
    ↪ basically do string length - LPS.
    ↪ Why this?
     Let's take an example: string s =
         ↪ aabca; Let's say aca is our
         ↪ LPS. Now we find how many char
         ↪  we need to insert to make the
         ↪  string palindrome while our
         ↪ LPS is fixed.
     a ab c a now to make the string
         ↪ palindrome we just need to
         ↪ insert the reverse of ab after
         ↪  c. So the new string looks
         ↪ like a ab c ba a
4. Minimum Number of Deletions and
    ↪ Insertions to make the string
    ↪ equals To solve this we just find
    ↪ the LCS of those string then just
    ↪ do: n + m - 2 * LCS.length() where
    ↪  n, m = strings length
*/
```

3.2 MCM

```
// TC: O(n ^ 3)
const ll N = 1005;
vector<ll> v;
```

```
ll dp[N][N], mark[N][N];
ll MCM(ll i, ll j) {
  if (i == j)
    return dp[i][j] = 0;
  if (dp[i][j] != -1)
    return dp[i][j];
  ll mn = INT_MAX;
  for (ll k = i; k < j; k++) {
    ll x = mn;
    mn = min(mn, MCM(i, k) + MCM(k + 1,
        ↪ j) + v[i - 1] * v[k] * v[j]);
    if (x != mn)
      mark[i][j] = k;
  }
  return dp[i][j] = mn;
}
void print_order(ll i, ll j) {
  if (i == j)
    cout << "X" << i;
  else {
    cout << "(";
    print_order(i, mark[i][j]);
    print_order(mark[i][j] + 1, j);
    cout << ")";
  }
}
// memset(dp, -1, sizeof dp);
// print_order(1, n);
```

3.3 LIS_length

```
vector<ll> v = {7, 3, 5, 3, 6, 2, 9, 8};
vector<ll> seq;
/*
here we basically check is the current
    ↪ element from v is greater than the
    ↪  last element of the sequence. if
    ↪ it is then push it to the seq
    ↪ array and if not then replace that
    ↪  index value. let's take an
    ↪ example:
v = 7 3 5 3 6 2 9 8
1st iteration seq = 7;
2nd iteration seq = 3;
3rd iteration seq = 3 5;
4th iteration seq = 3 3;
5th iteration seq = 3 3 6;
6th iteration seq = 2 3 6;
7th iteration seq = 2 3 6 9;
8th iteration seq = 2 3 6 8;
*/
for (auto i : v) {
  auto id = lower_bound(seq.begin(), seq
      ↪ .end(), i);
  if (id == seq.end())
    seq.push_back(i);
  else
    seq[id - seq.begin()] = i;
}
cout << seq.size() << endl;
```

3.4 LCIS

```
ll a[100] = {0}, b[100] = {0}, f[100] =
    ↪ {0};
ll n = 0, m = 0;
ll main(void) {
  cin >> n;
  for (ll i = 1; i <= n; i++)
    cin >> a[i];
  cin >> m;
  for (ll i = 1; i <= m; i++)
    cin >> b[i];
  for (ll i = 1; i <= n; i++) {
    ll k = 0;
    for (ll j = 1; j <= m; j++) {
      if (a[i] > b[j] && f[j] > k)
        k = f[j];
      else if (a[i] == b[j] && k + 1 > f
          ↪ [j])
        f[j] = k + 1;
    }
  }
  ll and = 0;
  for (ll i = 1; i <= m; i++)
    if (f[i] > ans)
      ans = f[i];
  cout << and << endl;
  return 0;
}
```

3.5 SOS DP

```
// sum over subsets
for (int i = 0; i < B; i++) {
  for (int mask = 0; mask < (1 << B);
      ↪ mask++) {
    if ((mask & (1 << i)) != 0) {
      f[mask] += f[mask ^ (1 << i)];
    }
  }
}

// sum over supersets
for (int i = 0; i < B; i++) {
  for (int mask = (1 << B) - 1; mask >=
      ↪ 0; mask--) {
    if ((mask & (1 << i)) == 0)
      g[mask] += g[mask ^ (1 << i)];
  }
}
// submask
for (int mask = 1; mask < (1 << 5); mask
    ↪ ++) {
  for (int submask = mask; submask > 0;
      ↪ submask = ((submask - 1) & mask)
      ↪ ) {
    int subset = mask ^ submask;
  }
}
/**
SOS DP (Sum Over Subsets Dynamic
    ↪ Programming) - 5 Key Points:
 1. CORE IDEA: Build DP table dp[i][mask
     ↪ ] = answer considering first i
     ↪ bits of mask Transition: dp[i][
     ↪ mask] = dp[i-1][mask] + dp[i-1][
     ↪ mask ^ (1<<(i-1))]
 2. SUBSET SUM: For x|y = x, iterate
     ↪ bits and add contributions from
     ↪ subsets If bit i is set in mask,
     ↪ add dp[i-1][mask without bit i]
 3. SUPERSET SUM: For x&y = x, iterate
     ↪ in reverse to handle supersets
     ↪ If bit i is unset in mask, add dp
     ↪ [i-1][mask with bit i set]
 **/
```

3.6 BS optimization

```
bitset<100005> bs = 1;
for (auto i : a) {
  bs |= (bs << i);
  // if previous 1 value pos is possible
      ↪  now ith bit or ith sm is also
      ↪ possible
}
cout << bs.count() - 1 << endl;
for (ll i = 1; i <= 100003; i++)
  if (bs[i])
    cout << i << " ";
cout << endl;
```

4 Graph

4.1 Dijkstra

```
// TC: O(V + ElogV)
typedef pair<ll, ll> pairi;
ll N = 20000 + 5;
vector<vector<pairi>> adj(N);
vector<ll> dis(N, inf), parent(N);

void dijkstra(ll src) {
```

```
  priority_queue<pairi, vector<pairi>,
     ↪ greater<pairi>> pq;
  dis[src] = 0;
  pq.push({0, src});
  while (pq.size()) {
    auto top = pq.top();
    pq.pop();
    for (auto i : adj[top.second]) {
      ll v = i.first;
      ll wt = i.second;
      if (dis[v] > dis[top.second] + wt)
         ↪  {
        dis[v] = dis[top.second] + wt;
        pq.push({dis[v], v});
        parent[v] = top.second
      }
    }
  }
}
ll node = n;
while (parent[node] != node) {
  path.push_back(node);
  node = parent[node];
}
path.push_back(1);
```

4.2 BellmanFord

```
// TC : O(V.E)
vector<ll> dist;
vector<ll> parent;
vector<vector<pair<ll, ll>>> adj;
// resize the vectors from main function

void bellmanFord(ll n, ll src) {
  dist[src] = 0;
  for (ll step = 0; step < n; step) {
    for (ll i = 1; i <= n; i++) {
      for (auto it : adj[i]) {
        ll u = i;
        ll v = it.first;
        ll wt = it.second;
        if (dist[u] != inf && ((dist[u]
           ↪ + wt) < dist[v])) {
          if (step == n - 1) {
            cout << "Negative cycle
               ↪ found\n ";
            return;
          }
          dist[v] = dist[u] + wt;
          parent[v] = u;
        }
      }
    }
  }
  for (ll i = 1; i <= n; i++)
    cout << dist[i] << " ";
  cout << endl;
}
```

4.3 FloydWarshall

```
// TC : O(n ^ 3)
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;
typedef vector<ll> VI;
typedef vector<VI> VVI;

bool FloydWarshall(VVT &w, VVI &prev) {
  ll n = w.size();
  prev = VVI(n, VI(n, -1));

  for (ll k = 0; k < n; k++) {
    for (ll i = 0; i < n; i++) {
      for (ll j = 0; j < n; j++) {
        if (w[i][j] > w[i][k] + w[k][j])
           ↪  {
          w[i][j] = w[i][k] + w[k][j];
          prev[i][j] = k;
        }
      }
    }
  }

  // check for negative weight cycles
  for (ll i = 0; i < n; i++)
    if (w[i][i] < 0)
      return false;
  return true;
}
```

4.4 Toposort

```
// TC : O(V + E)
map<ll, vector<ll>> adj;
map<ll, ll> degree;
set<ll> nodes;
vector<ll> ans;
// adj: graph input, degree: cnt
   ↪ indegree,
// node: unique nodes, ans: path
ll c = 0;
void topo_sort() {
  queue<ll> qu;
  // traverse all the nodes and check if
     ↪  its degree is 0 or not..
  for (ll i : nodes) {
    if (degree[i] == 0)
      qu.push(i);
  }
  while (!qu.empty()) {
    ll top = qu.front();
    qu.pop();
    ans.push_back(top);
    for (ll i : adj[top]) {
      degree[i]--;
      if (degree[i] == 0) {
        qu.push(i);
      }
    }
  }
}
```

4.5 Kruskal

```
// TC : O(ElogE)
typedef pair<ll, ll> edge;
class Graph {
  vector<pair<ll, edge>> G, T;
  vector<ll> parent;
  ll cost = 0;

public:
  Graph(ll n) {
    for (ll i = 0; i < n; i++)
      parent.push_back(i);
  }
  void add_edges(ll u, ll v, ll wt) { G.
     ↪ push_back({wt, {u, v}}); }
  ll find_set(ll n) {
    if (n == parent[n])
      return n;
    else
      return find_set(parent[n]);
  }

  void union_set(ll u, ll v) { parent[u]
     ↪  = parent[v]; }

  void kruskal() {
    sort(G.begin(), G.end());
    for (auto it : G) {
      ll uRep = find_set(it.second.first
         ↪ );
      ll vRep = find_set(it.second.
         ↪ second);
      if (uRep != vRep) {
        cost += it.first;
        T.push_back(it);
        union_set(uRep, vRep);
      }
    }
  }
  ll get_cost() { return cost; }
  void print() {
    for (auto it : T)
      cout << it.second.first << " " <<
         ↪ it.second.second << ": " <<
         ↪ it.first << endl;
  }
```

```
};
// g.add_edges(u, v, wt);
// g.kruskal();
```

4.6 Prims

```
// TC: O(ElogV)
typedef pair<ll, ll> pll;
class Prims {
  map<ll, vector<pll>> graph;
  map<ll, ll> visited;

public:
  void addEdge(ll u, ll v, ll w) {
    graph[u].push_back({v, w});
    graph[v].push_back({u, w});
  }

  vector<ll> path(pll start) {
    vector<ll> ans;
    priority_queue<pll, vector<pll>,
        ↪ greater<pll>> pq;
    // cost vs node
    pq.push({start.second, start.first})
        ↪ ;
    while (!pq.empty()) {
      pair<ll, ll> curr = pq.top();
      pq.pop();
      if (visited[curr.second])
        continue;
      visited[curr.second] = 1;
      ans.push_back(curr.second);
      for (auto i : graph[curr.second])
          ↪ {
        if (visited[i.first])
          continue;
        pq.push({i.second, i.first});
      }
    }
    return ans;
  }
};
```

4.7 LCA

```
// TC: preprocessing O(nlogn), each
    ↪ query O(logn)
ll n, l;
vector<vector<ll>> adj;
ll timer;
vector<ll> tin, tout;
vector<vector<ll>> up;

void dfs(ll v, ll p) {
  tin[v] = ++timer;
  up[v][0] = p;
  for (ll i = 1; i <= l; ++i)
    up[v][i] = up[up[v][i - 1]][i - 1];

  for (ll u : adj[v]) {
    if (u != p)
      dfs(u, v);
  }

  tout[v] = ++timer;
}

bool is_ancestor(ll u, ll v) { return
    ↪ tin[u] <= tin[v] && tout[u] >=
    ↪ tout[v]; }

ll lca(ll u, ll v) {
  if (is_ancestor(u, v))
    return u;
  if (is_ancestor(v, u))
    return v;
  for (ll i = l; i >= 0; --i) {
    if (!is_ancestor(up[u][i], v))
      u = up[u][i];
  }
  return up[u][0];
}

void preprocess(ll root) {
  tin.resize(n);
  tout.resize(n);
  timer = 0;
  l = ceil(log2(n));
  up.assign(n, vector<ll>(l + 1));
  dfs(root, root);
}
```

4.8 Rerooting

```
namespace reroot {
const auto exclusive = [](const auto &a,
    ↪  const auto &base,
                              const auto &
                                  ↪ merge_into
                                  ↪ , int
                                  ↪ vertex)
                                  ↪ {
  int n = (int)a.size();
  using Aggregate = decay_t<decltype(
      ↪ base)>;
  vector<Aggregate> b(n, base);
  for (int bit = (int)__lg(n); bit >= 0;
      ↪  --bit) {
    for (int i = n - 1; i >= 0; --i)
      b[i] = b[i >> 1];
    int sz = n - (n & !bit);
    for (int i = 0; i < sz; ++i) {
      int index = (i >> bit) ^ 1;
      b[index] = merge_into(b[index], a[
          ↪ i], vertex, i);
    }
  }
  return b;
};
// MergeInto : Aggregate * Value *
    ↪ Vertex(int) * EdgeIndex(int) ->
    ↪ Aggregate
// Base : Vertex(int) -> Aggregate
// FinalizeMerge : Aggregate * Vertex(
    ↪ int) * EdgeIndex(int) -> Value
const auto rerooter = [](const auto &g,
    ↪ const auto &base,
                           const auto &
                               ↪ merge_into
                               ↪ , const
                               ↪ auto &
                               ↪ finalize_merge
                               ↪ ) {
  int n = (int)g.size();
  using Aggregate = decay_t<decltype(
      ↪ base(0))>;
  using Value = decay_t<decltype(
      ↪ finalize_merge(base(0), 0, 0))>;
  vector<Value> root_dp(n), dp(n);
  vector<vector<Value>> edge_dp(n),
      ↪ redge_dp(n);

  vector<int> bfs, parent(n);
  bfs.reserve(n);
  bfs.push_back(0);
  for (int i = 0; i < n; ++i) {
    int u = bfs[i];
    for (auto v : g[u]) {
      if (parent[u] == v)
        continue;
      parent[v] = u;
      bfs.push_back(v);
    }
  }

  for (int i = n - 1; i >= 0; --i) {
    int u = bfs[i];
    int p_edge_index = -1;
    Aggregate aggregate = base(u);
    for (int edge_index = 0; edge_index
        ↪ < (int)g[u].size(); ++
        ↪ edge_index) {
      int v = g[u][edge_index];
      if (parent[u] == v) {
        p_edge_index = edge_index;
        continue;
      }
      aggregate = merge_into(aggregate,
          ↪ dp[v], u, edge_index);
    }
    dp[u] = finalize_merge(aggregate, u,
        ↪  p_edge_index);
```

```
  }
  for (auto u : bfs) {
    dp[parent[u]] = dp[u];
    edge_dp[u].reserve(g[u].size());
    for (auto v : g[u])
      edge_dp[u].push_back(dp[v]);
    auto dp_exclusive = exclusive(
        ↪ edge_dp[u], base(u),
        ↪ merge_into, u);
    redge_dp[u].reserve(g[u].size());
    for (int i = 0; i < (int)
        ↪ dp_exclusive.size(); ++i)
      redge_dp[u].push_back(
          ↪ finalize_merge(dp_exclusive[
          ↪ i], u, i));
    root_dp[u] = finalize_merge(
        n > 1 ? merge_into(dp_exclusive
            ↪ [0], edge_dp[u][0], u, 0)
            ↪ : base(u), u,
        -1);
    for (int i = 0; i < (int)g[u].size()
        ↪ ; ++i) {
      dp[g[u][i]] = redge_dp[u][i];
    }
  }
  return make_tuple(move(root_dp), move(
      ↪ edge_dp), move(redge_dp));
};
} // namespace reroot

int main() {
  ll n;
  cin >> n;
  vector<vector<ll>> g(n);
  // everything should be 0 based.

  using Aggregate = int;
  using Value = int;

  auto base = [](int vertex) ->
      ↪ Aggregate {
    // task here
  };
  auto merge_into = [](Aggregate
      ↪ vertex_dp, Value neighbor_dp,
      ↪ int vertex, int edge_index) ->
      ↪ Aggregate {
    // task here
  };
  auto finalize_merge = [](Aggregate
      ↪ vertex_dp, int vertex, int
      ↪ edge_index) -> Value {
    // task here
  };

  auto [reroot_result, edge_dp, redge_dp
      ↪ ] = reroot::rerooter(g, base,
      ↪ merge_into, finalize_merge);
}
```

4.9 Centroid_Tree

```
  const ll n = 1e5;
  vector<ll> sz(n + 5), dead(n + 5);
  function<void(ll, ll)> calculate_sz =
      ↪ [&](ll u, ll p) {
    sz[u] = 1;
    for (auto v : adj[u]) {
      if (v != p and !dead[v]) {
        calculate_sz(v, u);
        sz[u] += sz[v];
      }
    }
    return;
  };

  function<ll(ll, ll, ll)> find_centroid
      ↪  = [&](ll u, ll p, ll total) ->
      ↪ ll {
    for (auto v : adj[u]) {
      if (v != p and !dead[v] and 2 * sz
          ↪ [v] > total)
        return find_centroid(v, u, total
            ↪ );
    }
    return u;
  };

  function<void(ll)> decompose = [&] (ll
      ↪  u) -> void {
    // if needed change the parameter
    calculate_sz(u, -1);
    ll center = find_centroid(u, -1, sz[
        ↪ u]);
    // calculate the ans here
    dead[center] = 1;
    for(auto v : adj[center]) {
        if(!dead[v])
            decompose(v);
    }
  };
  // call decompose only
  decompose(1);
```

4.10 Euler_ckt

```
unordered_map<ll, ll> Start, End, Val;
unordered_map<ll, pair<ll, ll>> Range;
ll start = 0;
void dfs(ll node) {
  visited[node] = true;
  Start[node] = start++;
  for (auto child : adj[node]) {
    if (!visited[child])
      dfs(child);
  }
  End[node] = start - 1;
}
dfs(1);
vector<ll> FlatArray(start + 5);
for (auto i : Start) {
  FlatArray[i.second] = Val[i.first];
  Range[i.first] = {i.second, End[i.
      ↪ first]};
}
```

4.11 Min Cost Max Flow

```
#include <bits/stdc++.h>
using namespace std;
const int N = 3e5 + 9;

// Works for both directed, undirected
    ↪ and with negative cost too
// doesn't work for negative cycles
// for undirected edges just make the
    ↪ directed flag false
// Complexity: O(min(E^2 *V log V, E
    ↪ logV * flow))
using T = long long;
const T inf = 1LL << 61;
struct MCMF {
  struct edge {
    int u, v;
    T cap, cost;
    int id;
    edge(int _u, int _v, T _cap, T _cost
        ↪ , int _id) {
      u = _u;
      v = _v;
      cap = _cap;
      cost = _cost;
      id = _id;
    }
  };
  int n, s, t, mxid;
  T flow, cost;
  vector<vector<int>> g;
  vector<edge> e;
  vector<T> d, potential, flow_through;
  vector<int> par;
  bool neg;
  MCMF() {}
  MCMF(int _n) { // 0-based indexing
    n = _n + 10;
    g.assign(n, vector<int>());
    neg = false;
    mxid = 0;
  }
  void add_edge(int u, int v, T cap, T
```

```
    ↪ cost, int id = -1,
              bool directed = true) {
  if (cost < 0)
    neg = true;
  g[u].push_back(e.size());
  e.push_back(edge(u, v, cap, cost, id
    ↪ ));
  g[v].push_back(e.size());
  e.push_back(edge(v, u, 0, -cost, -1)
    ↪ );
  mxid = max(mxid, id);
  if (!directed)
    add_edge(v, u, cap, cost, -1, true
      ↪ );
}
bool dijkstra() {
  par.assign(n, -1);
  d.assign(n, inf);
  priority_queue<pair<T, T>, vector<
    ↪ pair<T, T>>, greater<pair<T, T
    ↪ >>> q;
  d[s] = 0;
  q.push(pair<T, T>(0, s));
  while (!q.empty()) {
    int u = q.top().second;
    T nw = q.top().first;
    q.pop();
    if (nw != d[u])
      continue;
    for (int i = 0; i < (int)g[u].size
      ↪ (); i++) {
      int id = g[u][i];
      int v = e[id].v;
      T cap = e[id].cap;
      T w = e[id].cost + potential[u]
        ↪ - potential[v];
      if (d[u] + w < d[v] && cap > 0)
        ↪ {
        d[v] = d[u] + w;
        par[v] = id;
        q.push(pair<T, T>(d[v], v));
      }
    }
  }
  for (int i = 0; i < n; i++) {
    if (d[i] < inf)
      d[i] += (potential[i] -
        ↪ potential[s]);
  }
  for (int i = 0; i < n; i++) {
    if (d[i] < inf)
      potential[i] = d[i];
  }
  return d[t] != inf; // for max flow
    ↪ min cost
  // return d[t] <= 0; // for min cost
    ↪  flow
}
T send_flow(int v, T cur) {
  if (par[v] == -1)
    return cur;
  int id = par[v];
  int u = e[id].u;
  T w = e[id].cost;
  T f = send_flow(u, min(cur, e[id].
    ↪ cap));
  cost += f * w;
  e[id].cap -= f;
  e[id ^ 1].cap += f;
  return f;
}
// returns {maxflow, mincost}
pair<T, T> solve(int _s, int _t, T
  ↪ goal = inf) {
  s = _s;
  t = _t;
  flow = 0, cost = 0;
  potential.assign(n, 0);
  if (neg) {
    // Run Bellman-Ford to find
      ↪ starting potential on the
      ↪ starting graph
    // If the starting graph (before
      ↪ pushing flow in the residual
      ↪  graph) is a
    // DAG, then this can be
      ↪ calculated in O(V + E) using
      ↪  DP: potential(v) =
    // min({potential[u] + cost[u][v
      ↪ ]}) for each u -> v and
      ↪ potential[s] = 0
    d.assign(n, inf);
    d[s] = 0;
    bool relax = true;
    for (int i = 0; i < n && relax; i
      ↪ ++) {
      relax = false;
      for (int u = 0; u < n; u++) {
        for (int k = 0; k < (int)g[u].
          ↪ size(); k++) {
          int id = g[u][k];
          int v = e[id].v;
          T cap = e[id].cap, w = e[id
            ↪ ].cost;
          if (d[v] > d[u] + w && cap >
            ↪  0) {
            d[v] = d[u] + w;
            relax = true;
          }
        }
      }
    }
    for (int i = 0; i < n; i++)
      if (d[i] < inf)
        potential[i] = d[i];
  }
  while (flow < goal && dijkstra())
    flow += send_flow(t, goal - flow);
  flow_through.assign(mxid + 10, 0);
  for (int u = 0; u < n; u++) {
    for (auto v : g[u]) {
      if (e[v].id >= 0)
        flow_through[e[v].id] = e[v ^
          ↪ 1].cap;
    }
  }
  return make_pair(flow, cost);
}
};
int main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n;
  cin >> n;
  assert(n <= 10);
  MCMF F(2 * n);
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      int k;
      cin >> k;
      F.add_edge(i, j + n, 1, k, i * 20
        ↪ + j);
    }
  }
  int s = 2 * n + 1, t = s + 1;
  for (int i = 0; i < n; i++) {
    F.add_edge(s, i, 1, 0);
    F.add_edge(i + n, t, 1, 0);
  }
  auto ans = F.solve(s, t).second;
  long long w = 0;
  set<int> se;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      int p = i * 20 + j;
      if (F.flow_through[p] > 0) {
        se.insert(j);
        w += F.flow_through[p];
      }
    }
  }
  assert(se.size() == n && w == n);
  cout << ans << '\n';
  return 0;
}
```

4.12 SCC

```
unordered_map<ll, vector<ll>> adj,
    ↪ InvAdj;
stack<ll> order;
unordered_map<ll, bool> visited;
unordered_map<ll, vector<ll>> all_scc;
unordered_map<ll, ll> compId;
void dfs_for_start(ll curr) {
  visited[curr] = 1;
  for (auto i : adj[curr])
    if (!visited[i])
      dfs_for_start(i);
  order.push(curr);
}
vector<ll> curr_comp;
void dfs_for_scc(ll curr) {
  visited[curr] = 1;
  for (auto i : InvAdj[curr])
    if (!visited[i])
      dfs_for_scc(i);
  curr_comp.push_back(curr);
}
inline void scc() {
  ll n, e, u, v;
  cin >> n >> e;
  for (ll i = 0; i < e; i++) {
    cin >> u >> v;
    adj[u].push_back(v);
    InvAdj[v].push_back(u);
  }
  for (ll i = 1; i <= n; i++)
    if (!visited[i])
      dfs_for_start(i);
  visited.clear();
  while (!order.empty()) {
    if (!visited[order.top()]) {
      curr_comp.clear();
      dfs_for_scc(order.top());
      ll sz = all_scc.size() + 1;
      all_scc[sz] = curr_comp;
      for (auto i : curr_comp)
        compId[i] = sz;
    }
    order.pop();
  }
}
// no.of ways and min cost of connecting
    ↪  the sccs
const ll MOD = 1e9 + 7, N = 1e5 + 2, INF
    ↪  = 1e18 + 2;
ll n, m, comp[N];
vector<ll> adj[N], rev[N];
bitset<N> vis;
void DFS1(ll u, stack<ll> &TS) {
  vis[u] = true;
  for (ll v : adj[u])
    if (!vis[v])
      DFS1(v, TS);
  TS.push(u);
}
void DFS2(ll u, const ll scc_no, ll &
    ↪ min_cost, ll &ways, vector<ll> &
    ↪ cost) {
  vis[u] = true;
  comp[u] = scc_no;
  for (ll v : rev[u])
    if (!vis[v]) {
      if (min_cost == cost[v])
        ++ways;
      else if (min_cost > cost[v]) {
        ways = 1;
        min_cost = cost[v];
      }
      DFS2(v, scc_no, min_cost, ways,
          ↪ cost);
    }
}
signed main() {
  FIO cin >> n;
  vector<ll> cost(n + 1);
  for (ll i = 1; i <= n; ++i)
    cin >> cost[i];
  cin >> m;
  while (m--) {
    ll u, v;
    cin >> u >> v;
    adj[u].push_back(v);
    rev[v].push_back(u);
  }
  ll tot = 0, ways = 1;
  stack<ll> TS;
  for (ll i = 1; i <= n; ++i)
    if (!vis[i])
      DFS1(i, TS);
  vis.reset();
  ll scc_no = 0;
  while (!TS.empty()) {
    ll u = TS.top();
    TS.pop();
    if (!vis[u]) {
      ll tmp_cst = cost[u], tmp_ways =
          ↪ 1;
      DFS2(u, ++scc_no, tmp_cst,
          ↪ tmp_ways, cost);
      tot += tmp_cst;
      ways = (ways * tmp_ways) % MOD;
    }
  }
  cout << tot << ' ' << ways;
} // TC: O(V+E)
```

4.13 0-1 BFS

```
vector<ll> d(n, INF);
d[s] = 0;
deque<ll> q;
q.push_front(s);
while (!q.empty()) {
  ll v = q.front();
  q.pop_front();
  for (auto edge : adj[v]) {
    ll u = edge.first;
    ll w = edge.second;
    if (d[v] + w < d[u]) {
      d[u] = d[v] + w;
      if (w == 1)
        q.push_back(u);
      else
        q.push_front(u);
    }
  }
}
```

4.14 Hull

```
// Convex Hull
#pragma GCC target("avx2")
#pragma GCC optimize("O3")
#pragma GCC optimize("unroll-loops")
#include <bits/stdc++.h>
using namespace std;

typedef long long int ll;
typedef long double ld;
typedef pair<ll, ll> pl;
typedef vector<ll> vl;
typedef complex<ll> pt;

#define G(x)
    ↪
    ↪ \
  ll x;
      ↪
      ↪ \
  cin >> x;
#define F(i, l, r) for (ll i = l; i < (r
    ↪ ); ++i)
#define A(a) (a).begin(), (a).end()
#define CRS(a, b) (conj(a) * (b)).Y
#define K first
#define V second
#define X real()
#define Y imag()
#define N 100010

namespace std {
bool operator<(pt a, pt b) { return a.X
    ↪ == b.X ? a.Y < b.Y : a.X < b.X; }
} // namespace std

bool in_hull(pt p, vector<pt> &hu,
```

```
    ↪ vector<pt> &hd) {
  if (p == *hu.begin() || p == *hd.begin
      ↪ ())
    return false; // change to true if
       ↪ border counts as inside
  if (p < *hu.begin() || *hd.begin() < p
      ↪ )
    return false;
  auto u = upper_bound(A(hu), p);
  auto d = lower_bound(hd.rbegin(), hd.
      ↪ rend(), p);
  return CRS(*u - p, *(u - 1) - p) > 0
      ↪ && CRS(*(d - 1) - p, *d - p) >
      ↪ 0;
  // change to >= if border counts as "
      ↪ inside"
}
void do_hull(vector<pt> &pts, vector<pt>
    ↪  &h) {
  for (pt p : pts) {
    while (h.size() > 1 && CRS(h.back()
        ↪ - p, h[h.size() - 2] - p) <=
        ↪ 0)
      // change to < 0 if border points
         ↪ included
      h.pop_back();
    h.push_back(p);
  }
}
pair<vector<pt>, vector<pt>> get_hull(
    ↪ vector<pt> &pts) {
  vector<pt> hu, hd;
  sort(A(pts)), do_hull(pts, hu);
  reverse(A(pts)), do_hull(pts, hd);
  return {hu, hd};
}
vector<pt> full_hull(vector<pt> &pts) {
  auto h = get_hull(pts);
  h.K.pop_back(), h.V.pop_back();
  for (pt p : h.V)
    h.K.push_back(p);
  return h.K;
}
int main() {
  G(n) vector<pt> v;
  F(i, 0, n) { G(x) G(y) v.push_back({x,
      ↪  y}); }
  vector<pt> h = full_hull(v);
}
```

4.15 Dynamic Hull

```
// Dynamic Convex Hull
#pragma GCC target("avx2")
#pragma GCC optimize("O3")
#pragma GCC optimize("unroll-loops")
#include <bits/stdc++.h>
using namespace std;

typedef long long int ll;
typedef long double ld;
typedef pair<ll, ll> pl;
typedef vector<ll> vl;
typedef complex<ll> pt;

#define G(x) ll x; cin >> x;
#define F(i, l, r) for (ll i = l; i < (r
    ↪ ); ++i)
#define A(a) (a).begin(), (a).end()
#define CRS(a, b) (conj(a) * (b)).Y
#define X real()
#define Y imag()
#define N 100010

namespace std {
bool operator<(pt a, pt b) { return a.X
    ↪ == b.X ? a.Y < b.Y : a.X < b.X; }
} // namespace std

// helper function for dyn_in_hull
bool in(pt p, set<pt> &h) {
  if (h.empty() || p < *h.begin() || *h.
      ↪ rbegin() < p)
    return false;
  auto i = h.upper_bound(p), j = i--;
  return CRS(*j - p, *i - p) > 0; //
      ↪ change to >= if border counts as
      ↪  "inside"
}

// returns true if p contained in
    ↪ dynamic hull hu / hd
bool in_hull(pt p, set<pt> &hu, set<pt>
    ↪ &hd) { return in(p, hu) && in(-p,
    ↪ hd); }

// helper function for dyn_add
void fix_bad(set<pt>::iterator i, set<pt
    ↪ > &h, bool l) {
  if (i == --h.begin() || i == h.end())
    return;
  pt p = *i;
  h.erase(p);
  if (!in(p, h))
    h.insert(p);
  else
    fix_bad(l ? --h.lower_bound(p) : h.
        ↪ upper_bound(p), h, l);
}

// helper function for dyn_add_to_hull
void add(pt p, set<pt> &h) {
  if (in(p, h))
    return;
  h.insert(p);
  fix_bad(--h.lower_bound(p), h, true);
  fix_bad(h.upper_bound(p), h, false);
}

// adds p to dynamic hull hu / hd
void add_to_hull(pt p, set<pt> &hu, set<
    ↪ pt> &hd) { add(p, hu), add(-p, hd)
    ↪ ; }

int main() {
  G(n) set<pt> hu, hd;
  F(i, 0, n) { G(x) G(y) add_to_hull({x,
      ↪  y}, hu, hd); }
}
```

4.16 Count SImple Cycle

```
void findNumberOfSimpleCycles(int N,
    ↪ vector<vector<int>> adj) {
  int ans = 0;
  int dp[(1 << N)][N];
  memset(dp, 0, sizeof dp);
  for (int mask = 0; mask < (1 << N);
      ↪ mask++) {
    int nodeSet = __builtin_popcountll(
        ↪ mask);
    int firstSetBit = __builtin_ffsl(
        ↪ mask);
    if (nodeSet == 1)
      dp[mask][firstSetBit] = 1;
    else {
      for (int j = firstSetBit + 1; j <
          ↪ N; j++) {
        if ((mask & (1 << j))) {
          int newNodeSet = mask ^ (1 <<
              ↪ j);
          for (int k = 0; k < N; k++) {
            if ((newNodeSet & (1 << k))
                ↪ && adj[k][j]) {
              dp[mask][j] += dp[
                  ↪ newNodeSet][k];
              if (adj[j][firstSetBit] &&
                  ↪  nodeSet > 2)
                ans += dp[mask][j];
            }
          }
        }
      }
    }
  }
  cout << ans << endl;
}
```

5 Misc

5.1 Max Pos and Next Greater

```
const ll MXX = 1e5 + 5;
ll mxtree[4 * MXX], arr[MXX];
void maxtree(ll idx, ll left, ll right)
    ↪ {
  if (left == right)
    mxtree[idx] = left;
  else {
    ll mid = (left + right) / 2;
    maxtree(idx * 2, left, mid);
    maxtree(idx * 2 + 1, mid + 1, right)
        ↪ ;
    ll left = mxtree[idx * 2];
    ll right = mxtree[idx * 2 + 1];
    if (arr[left] < arr[right])
      mxtree[idx] = right;
    else
      mxtree[idx] = left;
  }
}
ll mxPos(ll idx, ll tleft, ll tright, ll
    ↪  qleft, ll qright) {
  if (qleft > qright)
    return -1;
  if (qleft == tleft and qright ==
      ↪ tright)
    return mxtree[idx];
  ll tmid = (tleft + tright) / 2;
  ll left = mxPos(idx * 2, tleft, tmid,
      ↪ qleft, min(qright, tmid));
  ll right = mxPos(idx * 2 + 1, tmid +
      ↪ 1, tright, max(qleft, tmid + 1),
      ↪  qright);
  ll ans;
  if (left == -1)
    ans = right;
  else if (right == -1)
    ans = left;
  else if (arr[left] < arr[right])
    ans = right;
  else
    ans = left;
  return ans;
}
ll main() {
  ll t = 1, n, q, a, b;
  cin >> t;
  while (t--) {
    cin >> n >> q;
    for (ll i = 0; i < n; i++)
      cin >> arr[i];
    stack<ll> stk;
    ll nge[n];
    stk.push(0);
    for (ll i = 1; i < n; i++) {
      while (stk.size() and arr[stk.top
          ↪ ()] < arr[i]) {
        nge[stk.top()] = i;
        stk.pop();
      }
      stk.push(i);
    }
    while (stk.size()) {
      nge[stk.top()] = n;
      stk.pop();
    }
    ll ans[n];
    ans[n - 1] = 0;
    for (ll i = n - 2; i >= 0; i--) {
      ll tmp = nge[i];
      if (tmp == n)
        ans[i] = 0;
      else
        ans[i] = ans[tmp] + 1;
    }
    maxtree(1, 0, n - 1);
    for (ll i = 0; i < q; i++) {
      cin >> a >> b;
      if (a > b)
        swap(a, b);
      cout << ans[mxPos(1, 0, n - 1, a -
          ↪  1, b - 1)] << "\n";
    }
  }
}
```

5.2 Knight Move

```
ll X[8]={2,1,-1,-2,-2,-1,1,2};
ll Y[8]={1,2,2,1,-1,-2,-2,-1};
```

5.3 Matrix Exponentiation

```
using vvi = vector<vector<ll>>;

vvi martixMul(vvi &a, vvi &b) {
  ll r = a.size(), c = b[0].size();
  vvi ans(r, vector<ll>(c, 0));
  for (ll i = 0; i < r; i++) {
    for (ll j = 0; j < c; j++) {
      ans[i][j] = 0;
      for (ll k = 0; k < r; k++) {
        ans[i][j] += (a[i][k] * b[k][j])
            ↪  % mod;
        ans[i][j] %= mod;
      }
    }
  }
  return ans;
}

vvi martixExp(vvi base, ll power, ll MOD
    ↪  = mod) {
  vvi ans = base; // change the ans
      ↪ accordingly
  while (power) {
    if (power & 1)
      ans = martixMul(ans, base);
    base = martixMul(base, base);
    power /= 2;
  }
  return ans;
}
```

5.4 Ternary Search

```
double ternary_search(double l, double r
    ↪ ) {
  double eps = 1e-9; // error limit
  while (r - l > eps) {
    double m1 = l + (r - l) / 3, m2 = r
        ↪ - (r - l) / 3;
    double f1 = f(m1), f2 = f(m2); //
        ↪ evaluates the function at m1,
        ↪ m2
    if (f1 < f2)
      l = m1;
    else
      r = m2;
  }
  return f(l); // return the maximum of
      ↪ f(x) in [l, r]
}
```

6 Number Theory

6.1 Leap_year

```
bool isLeap(ll n) {
  if (n % 100 == 0)
    return (n % 400 == 0);
  else
    return (n % 4 == 0);
}
// leap year between l and r
ll calNum(ll y) { return (y / 4) - (y /
    ↪ 100) + (y / 400); }
ll leapNum(ll l, ll r) { return calNum(r
    ↪ ) - calNum(--l); }
```

6.2 Two Line Intersection

```
ll cross(ll x1, ll y1, ll x2, ll y2, ll
    ↪ x3, ll y3) {
  return (x2 - x1) * (y3 - y1) - (y2 -
      ↪ y1) * (x3 - x1);
```

```
}
bool intersect(ll x1, ll y1, ll x2, ll
    ↪ y2, ll x3, ll y3, ll x4, ll y4) {
  ll c1 = cross(x1, y1, x2, y2, x3, y3),
      ↪  c2 = cross(x1, y1, x2, y2, x4,
      ↪ y4),
      c3 = cross(x3, y3, x4, y4, x1, y1),
          ↪  c4 = cross(x3, y3, x4, y4,
          ↪ x2, y2);
  if ((!c1 && min(x1, x2) <= x3 && x3 <=
      ↪  max(x1, x2) && min(y1, y2) <=
      ↪ y3 &&
        y3 <= max(y1, y2)) |
       (!c2 && min(x1, x2) <= x4 && x4 <=
          ↪  max(x1, x2) && min(y1, y2)
          ↪ <= y4 &&
        y4 <= max(y1, y2)) |
       (!c3 && min(x3, x4) <= x1 && x1 <=
          ↪  max(x3, x4) && min(y3, y4)
          ↪ <= y1 &&
        y1 <= max(y3, y4)) |
       (!c4 && min(x3, x4) <= x2 && x2 <=
          ↪  max(x3, x4) && min(y3, y4)
          ↪ <= y2 &&
        y2 <= max(y3, y4)))
    return true;
  return (c1 > 0) != (c2 > 0) && (c3 >
      ↪ 0) != (c4 > 0);
}
```

6.3 Binary_exponentiation

```
ll binaryExp(ll base, ll power, ll MOD =
    ↪  mod) {
  ll res = 1;
  while (power) {
    if (power & 1)
      res = (res * base) % MOD;
    base = ((base % MOD) * (base % MOD))
        ↪  % MOD;
    power /= 2;
  }
  return res;
}
/*
task: a ^ b ^ c
binaryExp(a, binaryExp(b, c, mod - 1),
    ↪ mod)
*/
```

6.4 Count_divisor

```
ll maxVal = 1e6 + 1;
vector<ll> countDivisor(maxVal, 0);
void countingDivisor() {
  for (ll i = 1; i < maxVal; i++)
    for (ll j = i; j < maxVal; j += i)
      countDivisor[j]++;
}
// TC: nlog(n)
// count the number of divisors of all
    ↪ numbers in a range.
```

6.5 Check_prime

```
bool prime(ll n) {
  if (n < 2)
    return false;
  if (n <= 3)
    return true;
  if (!(n % 2) || !(n % 3))
    return false;
  for (ll i = 5; i * i <= n; i += 6) {
    if (!(n % i) || !(n % (i + 2)))
      return false;
  }
  return true;
}
// TC: sqrt(n) / 6;
```

6.6 SPF

```
// smallest prime factor using seive
const ll N = 1e7 + 5;
ll spf[N];
void smallestPrimeFactorUsingSeive() {
  for (ll i = 2; i < N; i++) {
    if (spf[i] == 0) {
      for (ll j = i; j < N; j += i) {
        if (spf[j] == 0)
          spf[j] = i;
      }
    }
  }
}
// smallest factor of a number
ll factor(ll n) {
  ll a;
  if (n % 2 == 0)
    return 2;
  for (a = 3; a * a <= n; a += 2) {
    if (n % a == 0)
      return a;
  }
  return n;
}
// complete factorization
ll r;
while (n > 1) {
  r = factor(n);
  cout << r << '\n';
  n /= r;
}
```

6.7 Seive

```
const ll N = 1e7 + 5;
ll prime[N];
void sieveOfEratosthenes() {
  for (ll i = 2; i < N; i++)
    prime[i] = 1;
  for (ll i = 4; i < N; i += 2)
    prime[i] = 0;
  for (ll i = 3; i * i < N; i++) {
    if (prime[i]) {
      for (ll j = i * i; j < N; j += i *
          ↪  2)
        prime[j] = 0;
    }
  }
}
```

6.8 Optimize_seive

```
vector<ll> sieve(const ll N, const ll Q
    ↪ = 17, const ll L = 1 << 15) {
  static const ll rs[] = {1, 7, 11, 13,
      ↪ 17, 19, 23, 29};
  struct P {
    P(ll p) : p(p) {}
    ll p;
    ll pos[8];
  };
  auto approx_prime_count = [](const ll
      ↪ N) -> ll {
    return N > 60184 ? N / (log(N) -
        ↪ 1.1) : max(1., N / (log(N) -
        ↪ 1.11)) + 1;
  };
  const ll v = sqrt(N), vv = sqrt(v);
  vector<bool> isp(v + 1, true);
  for (ll i = 2; i <= vv; ++i)
    if (isp[i]) {
      for (ll j = i * i; j <= v; j += i)
        isp[j] = false;
    }
  const ll rsize = approx_prime_count(N
      ↪ + 30);
  vector<ll> primes = {2, 3, 5};
  ll psize = 3;
  primes.resize(rsize);

  vector<P> sprimes;
  size_t pbeg = 0;
  ll prod = 1;
```

```
  for (ll p = 7; p <= v; ++p) {
    if (!isp[p])
      continue;
    if (p <= Q)
      prod *= p, ++pbeg, primes[psize++]
         ↪  = p;
    auto pp = P(p);
    for (ll t = 0; t < 8; ++t) {
      ll j = (p <= Q) ? p : p * p;
      while (j % 30 != rs[t])
        j += p << 1;
      pp.pos[t] = j / 30;
    }
    sprimes.push_back(pp);
  }
  vector<unsigned char> pre(prod, 0xFF);
  for (size_t pi = 0; pi < pbeg; ++pi) {
    auto pp = sprimes[pi];
    const ll p = pp.p;
    for (ll t = 0; t < 8; ++t) {
      const unsigned char m = ~(1 << t);
      for (ll i = pp.pos[t]; i < prod; i
         ↪  += p)
        pre[i] &= m;
    }
  }
  const ll block_size = (L + prod - 1) /
     ↪  prod * prod;
  vector<unsigned char> block(block_size
     ↪ );
  unsigned char *pblock = block.data();
  const ll M = (N + 29) / 30;

  for (ll beg = 0; beg < M; beg +=
     ↪ block_size, pblock -= block_size
     ↪ ) {
    ll end = min(M, beg + block_size);
    for (ll i = beg; i < end; i += prod)
       ↪  {
      copy(pre.begin(), pre.end(),
         ↪ pblock + i);
    }
    if (beg == 0)
      pblock[0] &= 0xFE;
    for (size_t pi = pbeg; pi < sprimes.
       ↪ size(); ++pi) {
      auto &pp = sprimes[pi];
      const ll p = pp.p;
      for (ll t = 0; t < 8; ++t) {
        ll i = pp.pos[t];
        const unsigned char m = ~(1 << t
           ↪ );
        for (; i < end; i += p)
          pblock[i] &= m;
        pp.pos[t] = i;
      }
    }
    for (ll i = beg; i < end; ++i) {
      for (ll m = pblock[i]; m > 0; m &=
         ↪  m - 1) {
        primes[psize++] = i * 30 + rs[
           ↪ __builtin_ctz(m)];
      }
    }
  }
  assert(psize <= rsize);
  while (psize > 0 && primes[psize - 1]
     ↪ > N)
    --psize;
  primes.resize(psize);
  return primes;
}
// it takes 500ms for generating prime
   ↪ upto 1e9
```

6.9 nth_prime_number

```
vector<ll> nth_prime;
const ll MX = 86200005;
bitset<MX> visited;
void optimized_prime() {
  nth_prime.push_back(2);
  for (ll i = 3; i < MX; i += 2) {
    if (visited[i])
      continue;
    nth_prime.push_back(i);
    if (1ll * i * i > MX)
      continue;
    for (ll j = i * i; j < MX; j += i +
       ↪ i)
      visited[j] = true;
  }
}
```

6.10 nCr

```
// 1:
// more space, less time
const ll MAX = 1e7 + 5;
vector<ll> fact(MAX), ifact(MAX), inv(
   ↪ MAX);
void factorial() {
  inv[1] = fact[0] = ifact[0] = 1;
  for (ll i = 2; i < MAX; i++)
    inv[i] = inv[mod % i] * (mod - mod /
       ↪  i) % mod;
  for (ll i = 1; i < MAX; i++)
    fact[i] = (fact[i - 1] * i) % mod;
  for (ll i = 1; i < MAX; i++)
    ifact[i] = ifact[i - 1] * inv[i] %
       ↪ mod;
}
ll nCr(ll n, ll r) {
  if (r < 0 || r > n)
    return 0;
  return (ll)fact[n] * ifact[r] % mod *
     ↪ ifact[n - r] % mod;
}
// 2:
// less space, more time
const ll MAX = 1e7 + 10;
vector<ll> fact(MAX), inv(MAX);
void factorial() {
  fact[0] = 1;
  for (ll i = 1; i < MAX; i++)
    fact[i] = (i * fact[i - 1]) % mod;
}
ll binaryExp(ll a, ll n, ll M = mod){};
   ↪ // needs to implement
void inverse() {
  for (ll i = 0; i < MAX; ++i)
    inv[i] = binaryExp(fact[i], mod - 2)
       ↪ ;
}
ll nCr(ll a, ll b) {
  if (a < b or a < 0 or b < 0)
    return 0;
  ll de = (inv[b] * inv[a - b]) % mod;
  return (fact[a] * de) % mod;
}
// 3:
// nCr mod m where m is not prime
ll C_mod_p(ll n, ll k, ll p) {
  if (k > n)
    return 0;
  vector<ll> fac(p);
  fac[0] = 1;
  for (int i = 1; i < p; i++)
    fac[i] = fac[i - 1] * i % p;

  ll res = 1;
  while (n || k) {
    ll ni = n % p, ki = k % p;
    if (ki > ni)
      return 0;
    res = res * fac[ni] % p * modInv(fac
       ↪ [ki], p) % p * modInv(fac[ni -
       ↪  ki], p) %
          p;
    n /= p;
    k /= p;
  }
  return res;
}
// compute nCr mod composite m (non-
   ↪ prime)
ll nCr_mod_m(ll n, ll k, ll m) {
  // Step 1: factorize m
```

```
  vector<int> primes;
  int tmp = m;
  for (int i = 2; i * i <= tmp; i++) {
    if (tmp % i == 0) {
      primes.push_back(i);
      while (tmp % i == 0)
        tmp /= i;
    }
  }
  if (tmp > 1)
    primes.push_back(tmp);

  // Step 2: compute result mod each
     ↪ prime
  vector<ll> rem, mod;
  for (int p : primes) {
    rem.push_back(C_mod_p(n, k, p));
    mod.push_back(p);
  }
  // Step 3: Chinese Remainder Theorem (
     ↪ combine)
  ll res = 0;
  for (int i = 0; i < (int)mod.size(); i
     ↪ ++) {
    ll Mi = m / mod[i];
    ll invMi = binaryExp(Mi, mod[i] - 2,
       ↪  mod[i]); // modular inverse
    res = (res + rem[i] * Mi % m * invMi
       ↪  % m) % m;
  }

  return res;
}
```

6.11 Factorial_mod

```
// n! mod p : Here P is mod value
// For binaryExp we call 1.6 function
ll factmod(ll n, ll p) {
  ll res = 1;
  while (n > 1) {
    res = (res * binaryExp(p - 1, n / p,
       ↪  p)) % p;
    for (ll i = 2; i <= n % p; ++i)
      res = (res * i) % p;
    n /= p;
  }
  return (res % p);
}
```

6.12 PHI

```
// the positive integers less than or
   ↪ equal to n that are relatively
   ↪ prime to n.
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;
}
// PHI of 1 to N
const int N = 1e6 + 9;
int phi[N];
int phiS[N];
void totient() {
  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;
    }
  }
  phiS[0] = phi[0];
  for (int i = 1; i < N; i++)
    phiS[i] = phiS[i - 1] + phi[i];
}
```

6.13 Catalan

```
void catalan(ll n) {
  ll res = 1;
  cout << res << " ";
  for (ll i = 1; i < n; i++) {
    res = (res * (4 * i - 2)) / (i + 1);
    cout << res << " ";
  }
}
```

6.14 Extended_GCD

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

6.15 Large Mod

```
ll mod(string &num, ll a) {
  ll res = 0;
  for (ll i = 0; i < num.length(); i++)
    res = (res * 10 + num[i] - '0') % a;
  return res;
}
```

6.16 Factorial_Divisor

```
ll factorialDivisors(ll n) {
  ll result = 1;
  for (ll i = 0; i < allPrimes.size(); i
     ↪ ++) {
    ll p = allPrimes[i];
    ll exp = 0;
    while (p <= n) {
      exp = exp + (n / p);
      p = p * allPrimes[i];
    }
    result = result * (exp + 1);
  }
  return result;
}
```

6.17 Number_conversion

```
// 10 - ary to m - ary
char a[16] = {'0', '1', '2', '3', '4', '
   ↪ 5', '6', '7', '8', '9', 'A', 'B',
   ↪ 'C', 'D', 'E', 'F'};
string tenToM(ll n, ll m) {
  ll temp = n;
  string result = "";
  while (temp != 0) {
    result = a[temp % m] + result;
    temp /= m;
  }
  return result;
}
// m - ary to 10 - ary
string num = "0123456789ABCDE";
ll mToTen(string n, ll m) {
```

```
  ll multi = 1;
  ll result = 0;
  for (ll i = n.size() - 1; i >= 0; i--)
    ↪ {
    result += num.find(n[i]) * multi;
    multi *= m;
  }
  return result;
}
```

6.18 Number_of_1_in_bit_till_N

```
ll cntOnes(ll n) {
  ll cnt = 0;
  for (ll i = 1; i <= n; i <<= 1) {
    ll x = (n + 1) / (i << 1);
    cnt += x * i;
    if ((n + 1) % i && n & i)
      cnt += (n + 1) % i;
  }
  return cnt;
}
```

6.19 Disarrangement

```
ll disarrange(ll n) {
  if (n == 1)
    return 0;
  if (n == 2)
    return 1;
  return (n - 1) * (disarrange(n - 1) +
    ↪ disarrange(n - 2));
}
// D(n) = (n!)/e
```

6.20 Millar_Rabin

```
bool check_composite(ll n, ll a, ll d,
  ↪ ll s) {
  ll x = binaryExp(a, d, n);
  if (x == 1 || x == n - 1)
    return false;
  for (ll r = 1; r < s; r++) {
    x = (u128)x * x % n;
    if (x == n - 1)
      return false;
  }
  return true;
};
bool MillerRabin(ll n, ll iter = 5) {
  // returns true if n is probably prime
    ↪ , else returns false.
  if (n < 4)
    return n == 2 || n == 3;
  ll s = 0;
  ll d = n - 1;
  while ((d & 1) == 0) {
    d >>= 1;
    s++;
  }
  for (ll i = 0; i < iter; i++) {
    ll a = 2 + rand() % (n - 3);
    if (check_composite(n, a, d, s))
      return false;
  }
  return true;
}
```

6.21 Modular_operation

```
// Addition :
ll mod_add(ll a, ll b, ll MOD = mod) {
  a = a % MOD, b = b % MOD;
  return (((a + b) % MOD) + MOD) % MOD;
}
// Subtraction :
ll mod_sub(ll a, ll b, ll MOD = mod) {
  a = a % MOD, b = b % MOD;
  return (((a - b) % MOD) + MOD) % MOD;
}
// Multiplication :
ll mod_mul(ll a, ll b, ll MOD = mod) {
  a = a % MOD, b = b % MOD;
  return (((a * b) % MOD) + MOD) % MOD;
}
// Division :
ll mminvprime(ll a, ll b) { return
  ↪ binaryExp(a, b - 2, b); }
ll mod_div(ll a, ll b, ll MOD = mod) {
  a = a % MOD, b = b % MOD;
  return (mod_mul(a, mminvprime(b, MOD),
    ↪ MOD) + MOD) % MOD;
}
```

6.22 MSLCM

```
// For a given number N, maximum sum LCM
  ↪ indicates the set of numbers
  ↪ whose LCM
// is N and summation is maximum. Let,
  ↪ MSLCM(N) denote this maximum sum
  ↪ of
// numbers. Given the value of N you
  ↪ will have to find the value:
  ↪ sumation of
// MSLCM(i) from i to 2n
ll MSLCM(ll n) {
  ll l = 1, r, val, ret = 0;
  while (l <= n) {
    val = n / l, r = n / val;
    ret += val * ((l + r) * (r - l + 1)
      ↪ / 2);
    l = r + 1;
  }
  return ret - 1;
}
```

6.23 Find numbers in between [L, R] which are divisible by all Array elements

```
void solve(ll *arr, ll N, ll L, ll R) {
  ll LCM = arr[0];
  for (ll i = 1; i < N; i++) {
    LCM = (LCM * arr[i]) / (__gcd(LCM,
      ↪ arr[i]));
  }
  if ((LCM < L && LCM * 2 > R) || LCM >
    ↪ R) {
    return;
  }
  ll k = (L / LCM) * LCM;
  //
  if (k < L)
    k = k + LCM;
  for (ll i = k; i <= R; i = i + LCM)
    cout << i << ' ';
}
```

6.24 PollarRho

```
namespace PollardRho {
  mt19937 rnd(chrono::steady_clock::now
    ↪ ().time_since_epoch().count());
  const int P = 1e6 + 9;
  ll seq[P];
  int primes[P], spf[P];
  inline ll add_mod(ll x, ll y, ll m) {
    return (x += y) < m ? x : x - m;
  }
  inline ll mul_mod(ll x, ll y, ll m) {
    ll res = __int128(x) * y % m;
    return res;
    // ll res = x * y - (ll)((long
      ↪ double)x * y / m + 0.5) * m;
    // return res < 0 ? res + m : res;
  }
  inline ll pow_mod(ll x, ll n, ll m) {
    ll res = 1 % m;
    for (; n; n >>= 1) {
      if (n & 1) res = mul_mod(res, x, m
        ↪ );
      x = mul_mod(x, x, m);
    }
    return res;
  }
```

```
  // O(it * (logn)^3), it = number of
     ↪ rounds performed
  inline bool miller_rabin(ll n) {
    if (n <= 2 || (n & 1 ^ 1)) return (n
       ↪  == 2);
    if (n < P) return spf[n] == n;
    ll c, d, s = 0, r = n - 1;
    for (; !(r & 1); r >>= 1, s++) {}
    // each iteration is a round
    for (int i = 0; primes[i] < n &&
       ↪ primes[i] < 32; i++) {
      c = pow_mod(primes[i], r, n);
      for (int j = 0; j < s; j++) {
        d = mul_mod(c, c, n);
        if (d == 1 && c != 1 && c != (n
           ↪ - 1)) return false;
        c = d;
      }
      if (c != 1) return false;
    }
    return true;
  }
  void init() {
    int cnt = 0;
    for (int i = 2; i < P; i++) {
      if (!spf[i]) primes[cnt++] = spf[i
         ↪ ] = i;
      for (int j = 0, k; (k = i * primes
         ↪ [j]) < P; j++) {
        spf[k] = primes[j];
        if (spf[i] == spf[k]) break;
      }
    }
  }
  // returns O(n^(1/4))
  ll pollard_rho(ll n) {
    while (1) {
      ll x = rnd() % n, y = x, c = rnd()
         ↪  % n, u = 1, v, t = 0;
      ll *px = seq, *py = seq;
      while (1) {
        *py++ = y = add_mod(mul_mod(y, y
           ↪ , n), c, n);
        *py++ = y = add_mod(mul_mod(y, y
           ↪ , n), c, n);
        if ((x = *px++) == y) break;
        v = u;
        u = mul_mod(u, abs(y - x), n);
        if (!u) return __gcd(v, n);
        if (++t == 32) {
          t = 0;
          if ((u = __gcd(u, n)) > 1 && u
             ↪  < n) return u;
        }
      }
      if (t && (u = __gcd(u, n)) > 1 &&
         ↪ u < n) return u;
    }
  }
  vector<ll> factorize(ll n) {
    if (n == 1) return vector <ll>();
    if (miller_rabin(n)) return vector<
       ↪ ll> {n};
    vector <ll> v, w;
    while (n > 1 && n < P) {
      v.push_back(spf[n]);
      n /= spf[n];
    }
    if (n >= P) {
      ll x = pollard_rho(n);
      v = factorize(x);
      w = factorize(n / x);
      v.insert(v.end(), w.begin(), w.end
         ↪ ());
    }
    return v;
  }
}
// auto f = PollardRho::factorize(n);
// sort(f.begin(), f.end());
// cout << f.size() << ' ';
// for (auto x: f) cout << x << ' ';
   ↪ cout << '\n';
```

6.25 2D Seg Tree

```
template <typename T> class BIT2D {
private:
  const int n, m;
  vector<vector<T>> bit;
public:
  BIT2D(int n, int m) : n(n), m(m), bit(
     ↪ n + 1, vector<T>(m + 1)) {}
  // adds val to the point (r, c) */
  void add(int r, int c, T val) {
    r++, c++;
    for (; r <= n; r += r & -r) {
      for (int i = c; i <= m; i += i & -
         ↪ i) {
        bit[r][i] += val;
      }
    }
  }
  // sum of points with row in [0, r]
     ↪ and column in [0, c] */
  T rect_sum(int r, int c) {
    r++, c++;
    T sum = 0;
    for (; r > 0; r -= r & -r) {
      for (int i = c; i > 0; i -= i & -i
         ↪ ) {
        sum += bit[r][i];
      }
    }
    return sum;
  }
  // sum of points with row in [r1, r2]
     ↪ and column in [c1, c2] */
  T rect_sum(int r1, int c1, int r2, int
     ↪  c2) {
    return rect_sum(r2, c2) - rect_sum(
       ↪ r2, c1 - 1) - rect_sum(r1 - 1,
       ↪  c2) +
           rect_sum(r1 - 1, c1 - 1);
  }
};
```

6.26 Next Prev Smaller

```
vector<ll> next_smallest(vector<ll> v) {
  ll n = v.size();
  vector<ll> nextS(n, n);
  stack<ll> st;
  for (int i = 0; i < n; i++) {
    while (!st.empty() and v[i] < v[st.
       ↪ top()]) {
      nextS[st.top()] = i;
      st.pop();
    }
    st.push(i);
  }
  return nextS;
}
vector<ll> prev_smallest(vector<ll> v) {
  ll n = v.size();
  vector<ll> prevS(n, -1);
  stack<ll> st;
  for (int i = 0; i < n; ++i) {
    while (!st.empty() && v[i] < v[st.
       ↪ top()])
      st.pop();
    if (!st.empty())
      prevS[i] = st.top();
    st.push(i);
  }
  return prevS;
}
```

7 Information

7.1 Numbers with Most Divisors

Max Value (N)	Number with Most Divisors (n)	Number of Divisors ($\tau(n)$)
10^3	83,160	128
10^6	720,720	240
10^7	$9,609,600$	640
10^8	$98,280,000$	672
10^9	$735,134,400$	1,344
10^{10}	$7,242,460,800$	2,688
10^{11}	$73,346,256,000$	5,376
10^{12}	$936,966,912,400$	10,752

7.2 Totient Function inside:

- For a prime p: $\phi(p) = p - 1$
- For a prime power p^e: $\phi(p^e) = (p-1)p^{e-1}$
- Divisor sum identity: $\sum_{d|n} \phi(d) = n$
- Sum of integers coprime to n: $f(\mathrm{n}) = \sum_{\substack{1 \le i \le n \\ \gcd(i,n)=1}} i = \dfrac{\phi(n) \cdot n}{2}$
- If GCD $(n, i) = x$ then, $\phi\left(\dfrac{n}{x}\right) = 1$
- Sum of GCD over all pairs (i, j) with $1 \le i, j \le n$: $\sum_{i=1}^{n} \sum_{d|i} d \cdot \phi\left(\dfrac{i}{d}\right)$
- Sum of LCM over all pairs (i, j) with $1 \le i, j \le n$: $\sum_{i=1}^{n} \sum_{d|i} i \cdot f\left(\dfrac{i}{d}\right)$

7.3 Combinatorics Information

Distinct-element counts over all subarrays: To count the number of distinct elements across all subarrays, consider every ending index i. By tracking the last occurrence of each value, we can determine how far left we can extend while keeping all elements distinct. This gives the maximum valid window $[L_i, i]$. The number of subarrays ending at i that have all distinct elements is simply the window size $(i - L_i + 1)$. Summing these values for all i yields the total distinct-element count over all subarrays.

Bitwise OR over all subarrays: For subarrays ending at index i, the bitwise OR value can only increase as the subarray grows, since once a bit becomes 1, it never returns to 0. Let S_{i-1} be the set of OR values of all subarrays ending at $i - 1$. To compute the OR values for subarrays ending at i, take each value in S_{i-1}, OR it with a_i, and also include the single element OR value a_i. After merging duplicates, the resulting set contains all OR values achieved by subarrays ending at i. Summing all these values over every i yields the total bitwise OR over all subarrays.

Bitwise XOR over all subarrays: Let pref[i] be the prefix XOR up to index i. The XOR of a subarray $[l, r]$ is pref[r]$\oplus$pref[$l-1$]. Consider a single bit position. This bit is 1 in the subarray XOR exactly when the two prefixes differ at that bit. Count how many prefixes have bit 0 and how many have bit 1. The number of contributing pairs is their product. Summing this contribution over all bits gives the total XOR value of all subarrays.

XOR of all pairs: For any bit position, the XOR of two values is 1 at that bit if and only if one value has the bit set and the other does not. If c_1 numbers have the bit 1 and c_0 have the bit 0, then the number of pairs contributing a 1 at that bit is $c_0 \cdot c_1$. Summing these contributions over all bit positions yields the XOR of all unordered pairs.

XOR of OR of every subarray: A bit in the OR of a subarray is 1 if the subarray contains at least one element with that bit set. If a bit appears in k positions, count how many subarrays include at least one such position. If this number is odd, the bit remains in the final XOR; otherwise it cancels out.

Tree: total length of all simple paths: Removing an edge splits the tree into parts of sizes s and $n - s$. Any path connecting one node from each part must use this edge, giving $s(n-s)$ such paths. Multiplying by the edge weight and summing over all edges yields the total path length.

Tree: distinct elements over all simple paths: Using DSU-on-tree, maintain the largest child's data structure for each subtree and merge smaller children into it. This efficiently tracks all distinct values appearing along all paths.

Minimum Hamming distance over all cyclic shifts: The Hamming distance for each shift equals the number of mismatched positions. Map equal characters to $+1$ and unequal characters to -1, then use convolution to compute match counts for all shifts simultaneously. More matches imply a smaller Hamming distance.

Total Hamming distance over all pairs: For each bit, if c values have that bit set and $n - c$ do not, then $c(n - c)$ pairs differ at that bit. Summing over all bits yields the total Hamming distance.

Sum of products over all subsequences: Expanding the product $(1+a_1)(1+a_2)\cdots(1+a_n)$ selects either 1 or a_i from each term, corresponding exactly to skipping or including a_i. Every subsequence product appears once; subtracting 1 removes the empty subsequence.

Widths over all subsequences: After sorting, element a_i is maximum in 2^i subsequences and minimum in 2^{n-i-1} subsequences. Summing the contributions of all elements gives the total width.

Optimal weighted sum with range increments: A difference array counts how many range operations affect each position. Sorting both the array values and their frequencies and pairing largest with largest maximizes the total sum.

Sum of divisors for 1 to N: Each integer i is a divisor of all multiples $i, 2i, 3i, \ldots$. Adding i to the divisor sum of each of these values for all i from 1 to N computes all divisor sums.

Sum of absolute differences over all pairs: After sorting, each a_i contributes to the absolute difference

with all smaller elements on its left and all larger elements on its right. Prefix sums compute these contributions efficiently.

Sum of inversion counts over all permutations: Each pair (i, j) with $i < j$ has a $1/2$ probability of appearing as an inversion in a random permutation. With $\frac{n(n-1)}{2}$ such pairs, the expected number of inversions is $\frac{n(n-1)}{4}$. Multiplying by $n!$ gives the total inversion sum.

Inversion sum over binary strings with x zeros and y ones: There are xy zero-one pairs, and each such pair contributes an inversion in exactly half the binary strings of length $x + y$ containing x zeros and y ones. Multiplying by the total count $\binom{x+y}{x}$ yields the inversion sum.

8 Mathematics

8.1 Area Formulas

Rectangle	length × width
Square	side^2
Triangle	$\frac{1}{2}$ × base × height
Parallelogram	base × height
Pyramid (no base)	$\frac{1}{2}$ × (perimeter of base) × (slant height)
Polygon	$\frac{1}{2}\left\lvert\sum_{i=1}^{n}(x_i y_{i+1} - x_{i+1} y_i)\right\rvert$ $a + \frac{b}{2} - 1$ (for lattice coordinates)

a = interior lattice pts, b = boundary pts.

8.2 Volume Formulas

Cube	side^3
Rectangular Prism	length × width × height
Cylinder	π × radius^2 × height
Sphere	$\frac{4}{3}\pi$ × radius^3
Pyramid	$\frac{1}{3}$ × (base area) × (height)

8.3 Surface Area Formulas

Cube	$6 \times \text{side}^2$
Rectangular Prism	$2(lw + lh + wh)$ (l = length, w = width, h = height)
Cylinder	$2\pi r(r + h)$
Sphere	$4\pi r^2$
Pyramid	base area + $\frac{1}{2}$ × (perimeter) × (slant height)

8.4 Triangles

Semiperimeter	$s = \frac{a+b+c}{2}$
Area	$A = \sqrt{s(s-a)(s-b)(s-c)}$
Circumradius	$R = \frac{abc}{4A}$
Inradius	$r = \frac{A}{s}$
Median	$m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$
Angle bisector	$s_a = \sqrt{\frac{bc}{1-(a/(b+c))^2}}$

Side lengths: a, b, c.

8.5 Sum Equations

$$\sum_{i=k}^{n} c^i = \frac{c^{n+1} - c^k}{c-1} \qquad \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \qquad \sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2$$

$$\sum_{i=1}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} \qquad \sum_{i=1}^{n}(2i-1) = n^2$$

8.6 Logarithmic Basic

$\log_b 1 = 0$	$\log_b b = 0$
$b^{\log_b a} = a$	$x^{\log_b y} = y^{\log_b x}$
$\log_a b = \frac{1}{\log_b a}$	$\log_a x = \frac{\log_b x}{\log_b a}$
$\log_b(AB) = \log_b A + \log_b B$	
$\log_b\left(\frac{A}{B}\right) = \log_b A - \log_b B$	
$\log_a c = \log_a b \times \log_b c$	
$\log_b(A^x) = x \log_b A$	

8.7 Series

Catalan: $C_n = \frac{1}{n+1}\binom{2n}{n}$, $C_n = \sum_{k=0}^{n} C_k C_{n-k}$
Arithmetic: $a_n = a + (n-1)d$, $s_n = \frac{n}{2}(2a + (n-1)d)$
Geometric: $a_n = ar^{n-1}$, $s_n = \frac{a(1-r^n)}{1-r}$
Derangements: $D_n = n! \sum_{k=0}^{n} \frac{(-1)^k}{k!}$, $D_n = \left\lfloor \frac{n!}{e} + \frac{1}{2} \right\rfloor$
Fibonacci: $f_n = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}}$, $\phi = \frac{1+\sqrt{5}}{2}$

8.8 Pick's Theorem

$A = I - \frac{1}{2}B + 1$ (I = interior points, B = boundary points)

8.9 Stars and Bars

Number of solutions of $x_1 + \cdots + x_k = n$:
$\binom{n-1}{k-1}$ when $x_i > 0$; $\binom{n+k-1}{k-1}$ when $x_i \geq 0$.

8.10 Facts

$\left\lceil \frac{a}{b} \right\rceil = \left\lfloor \frac{a-1}{b} \right\rfloor + 1$
Sum l to r: $\frac{l+r}{2}(r - l + 1)$
$\left\lfloor \frac{\lfloor n/a \rfloor}{b} \right\rfloor = \left\lfloor \frac{n}{ab} \right\rfloor$

8.11 LCM

$\text{lcm}(a, n) + \text{lcm}(n - a, n) = \frac{n^2}{\gcd(a,n)}$
$\text{SUM} = \frac{n}{2}\left(\sum_{d|n} \varphi(d)d + 1\right)$