.

.

Contents

1. Misc

1.1. Debug List

```
- Pre-submit:
  - Did you make a typo when copying a template?
  - Test more cases if unsure.
    - Write a naive solution and check small cases.
  - Submit the correct file.

- General Debugging:
  - Read the whole problem again.
  - Have a teammate read the problem.
  - Have a teammate read your code.
    - Explain you solution to them (or a rubber duck).
  - Print the code and its output / debug output.
  - Go to the toilet.

- Wrong Answer:
  - Any possible overflows?
    - > `__int128` ?
    - Try `-ftrapv` or `#pragma GCC optimize("trapv")`
  - Floating point errors?
    - > `long double` ?
    - turn off math optimizations
    - check for `==`, `>=`, `acos(1.000000001)`, etc.
  - Did you forget to sort or unique?
  - Generate large and worst "corner" cases.
  - Check your `m` / `n`, `i` / `j` and `x` / `y`.
  - Are everything initialized or reset properly?
  - Are you sure about the STL thing you are using?
    - Read cppreference (should be available).
  - Print everything and run it on pen and paper.

- Time Limit Exceeded:
  - Calculate your time complexity again.
  - Does the program actually end?
    - Check for `while(q.size())` etc.
  - Test the largest cases locally.
  - Did you do unnecessary stuff?
    - e.g. pass vectors by value
    - e.g. `memset` for every test case
  - Is your constant factor reasonable?

- Runtime Error:
  - Check memory usage.
    - Forget to clear or destroy stuff?
    - > `vector::shrink_to_fit()`
  - Stack overflow?
  - Bad pointer / array access?
    - Try `-fsanitize=address`
  - Division by zero? NaN's?
```

1.2. Algorithms

1.2.1. DP opt

Aliens

```cpp
// min dp[i] value and its i (smallest one)
pll get_dp(int cost);
ll aliens(int k, int l, int r) {
  while (l != r) {
    int m = (l + r) / 2;
    auto [f, s] = get_dp(m);
    if (s == k) return f - m * k;
    if (s < k) r = m;
    else l = m + 1;
  }
  return get_dp(l).first - l * k;
}
```

DnC DP

```cpp
int m, n;
vector<long long> dp_before, dp_cur;

long long C(int i, int j);

// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr) {
  if (l > r) return;

  int mid = (l + r) >> 1;
  pair<long long, int> best = {LLONG_MAX, -1};

  for (int k = optl; k <= min(mid, optr); k++) {
    best =
    min(best, {(k ? dp_before[k - 1] : 0) + C(k, mid), k});
  }

  dp_cur[mid] = best.first;
  int opt = best.second;

  compute(l, mid - 1, optl, opt);
  compute(mid + 1, r, opt, optr);
}

long long solve() {
  dp_before.assign(n, 0);
  dp_cur.assign(n, 0);

  for (int i = 0; i < n; i++) dp_before[i] = C(0, i);

  for (int i = 1; i < m; i++) {
    compute(0, n - 1, 0, n - 1);
    dp_before = dp_cur;
  }

  return dp_before[n - 1];
}
```

Knuth's Opt

```
int solve() {
  int N;
  int dp[N][N], opt[N][N];

  auto C = [&](int i, int j) {
    ... // Implement cost function C.
  };

  for (int i = 0; i < N; i++) {
    opt[i][i] = i;
    ... // Initialize dp[i][i] according to the problem
  }

  for (int i = N - 2; i >= 0; i--) {
    for (int j = i + 1; j < N; j++) {
      int mn = INT_MAX;
      int cost = C(i, j);
      for (int k = opt[i][j - 1];
           k <= min(j - 1, opt[i + 1][j]); k++) {
        if (mn >= dp[i][k] + dp[k + 1][j] + cost) {
          opt[i][j] = k;
          mn = dp[i][k] + dp[k + 1][j] + cost;
        }
      }
      dp[i][j] = mn;
    }
  }

  return dp[0][N - 1];
}
```

DP SOS

```
for (int i = 0; i < (1 << N); ++i) F[i] = A[i];
for (int i = 0; i < N; ++i)
  for (int mask = 0; mask < (1 << N); ++mask) {
    if (mask & (1 << i)) F[mask] += F[mask ^ (1 << i)];
  }
```

DP Open and Close

```
// how many ways to divide into groups such that sum of
// max-min of each group <= x sort a increasing dp[i][j][k]
// = first i elements, j open sets, sum = k v = dp[i][j][k],
// k1 = k + j(a[i + 1] - a[i]) put a[i] in own group:
// dp[i+1][j][k1] += v put a[i] in an open group:
// dp[i+1][j][k1] += j*v put a[i] in a new group (not close
// it): dp[i+1][j+1][k1] += v put a[i] in an open group and
// close it: dp[i+1][j-1][k1] += j*v cout dp[n][0][0->x]
```

1.2.2. Mo's Algorithm

```
void remove(
idx); // TODO: remove value at idx from data structure
void add(idx); // TODO: add value at idx from data structure
int get_answer(); // TODO: extract the current answer of the
                  // data structure

int block_size;

bool cmp(pair<int, int> p, pair<int, int> q) {
  if (p.first / BLOCK_SIZE != q.first / BLOCK_SIZE)
    return p < q;
  return (p.first / BLOCK_SIZE & 1) ? (p.second < q.second)
                                    : (p.second > q.second);
}

struct Query {
  int l, r, idx;
  bool operator<(Query other) const {
    return make_pair(l / block_size, r) <
           make_pair(other.l / block_size, other.r);
  }
};

vector<int> mo_s_algorithm(vector<Query> queries) {
  vector<int> answers(queries.size());
  sort(queries.begin(), queries.end());

  // TODO: initialize data structure

  int cur_l = 0;
  int cur_r = -1;
  // invariant: data structure will always reflect the range
  // [cur_l, cur_r]
  for (Query q : queries) {
    while (cur_l > q.l) {
      cur_l--;
      add(cur_l);
    }
    while (cur_r < q.r) {
      cur_r++;
      add(cur_r);
    }
    while (cur_l < q.l) {
      remove(cur_l);
      cur_l++;
    }
    while (cur_r > q.r) {
      remove(cur_r);
      cur_r--;
    }
    answers[q.idx] = get_answer();
  }
  return answers;
}
```

1.2.3. Mo's Algorithm on Tree

```
void MoAlgoOnTree() {
  Dfs(0, -1);
  vector<int> euler(tk);
  for (int i = 0; i < n; ++i) {
    euler[tin[i]] = i;
    euler[tout[i]] = i;
  }
  vector<int> l(q), r(q), qr(q), sp(q, -1);
  for (int i = 0; i < q; ++i) {
    if (tin[u[i]] > tin[v[i]]) swap(u[i], v[i]);
    int z = GetLCA(u[i], v[i]);
    sp[i] = z[i];
    if (z == u) l[i] = tin[u[i]], r[i] = tin[v[i]];
    else l[i] = tout[u[i]], r[i] = tin[v[i]];
    qr[i] = i;
  }
  sort(qr.begin(), qr.end(), [&](int i, int j) {
    if (l[i] / kB == l[j] / kB) return r[i] < r[j];
    return l[i] / kB < l[j] / kB;
  });
  vector<bool> used(n);
  // Add(v): add/remove v to/from the path based on used[v]
  for (int i = 0, tl = 0, tr = -1; i < q; ++i) {
    while (tl < l[qr[i]]) Add(euler[tl++]);
    while (tl > l[qr[i]]) Add(euler[--tl]);
    while (tr > r[qr[i]]) Add(euler[tr--]);
    while (tr < r[qr[i]]) Add(euler[++tr]);
    // add/remove LCA(u, v) if necessary
  }
}
```

1.2.4. Ternary search

```
template <class F> int ternSearch(int a, int b, F f) {
  assert(a <= b);
  while (b - a >= 5) {
    int mid = (a + b) / 2;
    if (f(mid) < f(mid + 1)) a = mid; // (A)
    else b = mid + 1;
  }
  rep(i, a + 1, b + 1) if (f(a) < f(i)) a = i; // (B)
  return a;
}
```

2. Data Structures

2.1. Implicit Segment Tree

```
struct Vertex {
  int left, right;
  int sum = 0;
  Vertex *left_child = nullptr, *right_child = nullptr;

  Vertex(int lb, int rb) {
    left = lb;
    right = rb;
  }

  void extend() {
    if (!left_child && left + 1 < right) {
      int t = (left + right) / 2;
      left_child = new Vertex(left, t);
      right_child = new Vertex(t, right);
    }
  }

  void add(int k, int x) {
    extend();
    sum += x;
    if (left_child) {
```

```
      if (k < left_child->right) left_child->add(k, x);
      else right_child->add(k, x);
    }
  }

  int get_sum(int lq, int rq) {
    if (lq <= left && right <= rq) return sum;
    if (max(left, lq) >= min(right, rq)) return 0;
    extend();
    return left_child->get_sum(lq, rq) +
           right_child->get_sum(lq, rq);
  }
};
```

2.2. Persistent Segment Tree

```
struct Node {
  int id, lid, rid, sum;

  Node() : id(-1), lid(-1), rid(-1), sum(0) {}

  Node(int v) : id(-1), lid(-1), rid(-1), sum(v) {}

  Node(const Node &l, const Node &r)
      : id(-1), lid(l.id), rid(r.id), sum(0) {
    if (l.id != -1) sum += l.sum;
    if (r.id != -1) sum += r.sum;
  }
};
vector<Node> seg;
vector<int> roots;

int build(int l, int r) {
  if (l == r) {
    seg.push_back(Node(0));
    seg.back().id = isz(seg) - 1;
    return seg.back().id;
  }
  int mid = (l + r) >> 1;
  int lid = build(l, mid), rid = build(mid + 1, r);
  seg.push_back(Node(seg[lid], seg[rid]));
  seg.back().id = isz(seg) - 1;
  return seg.back().id;
}

int upd(int id, int i, int l, int r) {
  if (l == r) {
    seg.push_back(Node(seg[id].sum + 1));
    seg.back().id = isz(seg) - 1;
    return seg.back().id;
  }
  int mid = (l + r) >> 1;
  int lid, rid;
  if (i <= mid)
    lid = upd(seg[id].lid, i, l, mid), rid = seg[id].rid;
  else
    lid = seg[id].lid,
    rid = upd(seg[id].rid, i, mid + 1, r);
  seg.push_back(Node(seg[lid], seg[rid]));
  seg.back().id = isz(seg) - 1;
  return seg.back().id;
}

int get_kth(int idl, int idr, int k, int l, int r) {
  if (l == r) return l;
  int mid = (l + r) >> 1;
  int cnt = seg[seg[idr].lid].sum - seg[seg[idl].lid].sum;
  if (cnt >= k)
    return get_kth(seg[idl].lid, seg[idr].lid, k, l, mid);
  return get_kth(seg[idl].rid, seg[idr].rid, k - cnt,
                 mid + 1, r);
}

int cnt_k(int idl, int idr, int k, int l, int r) {
  if (r <= k) return 0;
  if (l > k) return seg[idr].sum - seg[idl].sum;
  int mid = (l + r) >> 1;
  return cnt_k(seg[idl].lid, seg[idr].lid, k, l, mid) +
         cnt_k(seg[idl].rid, seg[idr].rid, k, mid + 1, r);
}
```

2.3. Persistent Trie

```
using namespace std;

// find maximum value (x^a[j]) in the range (l,r) where
// l<=j<=r
const int N = 1e5 + 100;
const int K = 15;

struct node_t;
typedef node_t *pnode;

struct node_t {
  int time;
  pnode to[2];
  node_t() : time(0) { to[0] = to[1] = 0; }
  bool go(int l) const {
    if (!this) return false;
    return time >= l;
  }
  pnode clone() {
    pnode cur = new node_t();
    if (this) {
      cur->time = time;
      cur->to[0] = to[0];
      cur->to[1] = to[1];
    }
    return cur;
  }
};

pnode last;
pnode version[N];

void insert(int a, int time) {
  pnode v = version[time] = last = last->clone();
  for (int i = K - 1; i >= 0; --i) {
    int bit = (a >> i) & 1;
    pnode &child = v->to[bit];
    child = child->clone();
    v = child;
    v->time = time;
  }
}

int query(pnode v, int x, int l) {
  int ans = 0;
  for (int i = K - 1; i >= 0; --i) {
    int bit = (x >> i) & 1;
    if (v->to[bit]->go(l)) { // checking if this bit was
                             // inserted before the range
      ans |= 1 << i;
      v = v->to[bit];
    } else {
      v = v->to[bit ^ 1];
    }
  }
  return ans;
}

void solve() {
  int n, q;
  scanf("%d %d", &n, &q);
  last = 0;
  for (int i = 0; i < n; ++i) {
    int a;
    scanf("%d", &a);
    insert(a, i);
  }
  while (q--) {
    int x, l, r;
    scanf("%d %d %d", &x, &l, &r);
    --l, --r;
    printf("%d\n", query(version[r], ~x, l));
    // Trie version[r] contains the trie for [0...r]
    // elements
  }
}
// credit: mochow13
```

2.4. 2D Fenwick Tree

```
void fakeupdate(int x, int y) {
  for (int i =
       lower_bound(a.begin(), a.end(), x) - a.begin();
       i < (int)a.size(); i += i & (-i)) {
    pos[i].push_back(y);
  }
}
void fakeget(int x, int y) {
  for (int i =
       lower_bound(a.begin(), a.end(), x) - a.begin();
       i > 0; i -= i & (-i)) {
    pos[i].push_back(y);
  }
}
void update(int x, int y, int v) {
  for (int i =
       lower_bound(a.begin(), a.end(), x) - a.begin();
       i < (int)a.size(); i += i & (-i)) {
    for (int j =
         lower_bound(pos[i].begin(), pos[i].end(), y) -
         pos[i].begin();
         j < (int)pos[i].size(); j += j & (-j)) {
```

```
      fen[i][j] = max(fen[i][j], v);
    }
  }
}
int get(int x, int y) {
  int sum = 0;
  for (int i =
       lower_bound(a.begin(), a.end(), x) - a.begin();
       i > 0; i -= i & (-i)) {
    for (int j =
         lower_bound(pos[i].begin(), pos[i].end(), y) -
         pos[i].begin();
         j > 0; j -= j & (-j)) {
      // cout << i << " " << j << " " << a[i] << ' ' <<
      // pos[i][j] << " " << fen[i][j] << "\n";
      sum = max(sum, fen[i][j]);
    }
  }
  return sum;
}
```

2.5. Line Container

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

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll 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->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 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));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

2.6. Wavelet Tree

```
struct Node {
  int mn, mx, lid, rid;
  vector<int> val, cnt;

  Node() : mn(inf32), mx(-inf32), lid(-1), rid(-1) {
    val.push_back(0);
    cnt.push_back(0);
  }
};
vector<Node> wavelet;

void build(int id = 0) {
  for (int i = 1; i < isz(wavelet[id].val); ++i) {
    wavelet[id].mn =
    min(wavelet[id].mn, wavelet[id].val[i]);
    wavelet[id].mx =
    max(wavelet[id].mx, wavelet[id].val[i]);
  }
  int mid = (wavelet[id].mn + wavelet[id].mx) >> 1;
  Node l, r;
  for (int i = 1; i < isz(wavelet[id].val); ++i) {
    wavelet[id].cnt.push_back(wavelet[id].cnt.back());
    if (wavelet[id].val[i] <= mid) {
      ++wavelet[id].cnt.back();
      l.val.push_back(wavelet[id].val[i]);
    } else r.val.push_back(wavelet[id].val[i]);
  }
  if (isz(wavelet[id].val) <= 2 ||
      wavelet[id].mn == wavelet[id].mx)
    return;
  if (isz(l.val) > 1) {
    wavelet[id].lid = isz(wavelet);
    wavelet.push_back(l);
    build(wavelet[id].lid);
  }
  if (isz(r.val) > 1) {
    wavelet[id].rid = isz(wavelet);
    wavelet.push_back(r);
    build(wavelet[id].rid);
  }
}

int get_kth(int id, int k, int l, int r) {
  if (id == -1 || l > r) return 0;
  if (wavelet[id].mn == wavelet[id].mx)
    return wavelet[id].mn;
  int cnt = wavelet[id].cnt[r] - wavelet[id].cnt[l - 1];
  if (cnt >= k)
    return get_kth(wavelet[id].lid, k,
                   wavelet[id].cnt[l - 1] + 1,
                   wavelet[id].cnt[r]);
  return get_kth(wavelet[id].rid, k - cnt,
                 l - wavelet[id].cnt[l - 1],
                 r - wavelet[id].cnt[r]);
}

int cnt_k(int id, int k, int l, int r) {
  if (id == -1 || l > r || wavelet[id].mx <= k) return 0;
  if (wavelet[id].mn > k) return r - l + 1;
  return cnt_k(wavelet[id].lid, k,
               wavelet[id].cnt[l - 1] + 1,
               wavelet[id].cnt[r]) +
         cnt_k(wavelet[id].rid, k,
               l - wavelet[id].cnt[l - 1],
               r - wavelet[id].cnt[r]);
}
```

2.7. Segtree Beats

```
struct Node {
  ll sum;  // Sum tag
  ll max1; // Max value
  ll max2; // Second Max value
  ll maxc; // Max value count
  ll min1; // Min value
  ll min2; // Second Min value
  ll minc; // Min value count
  ll lazy; // Lazy tag
} T[MAXN * 4];

void merge(int t) {
  // sum
  T[t].sum = T[t << 1].sum + T[t << 1 | 1].sum;

  // max
  if (T[t << 1].max1 == T[t << 1 | 1].max1) {
    T[t].max1 = T[t << 1].max1;
    T[t].max2 = max(T[t << 1].max2, T[t << 1 | 1].max2);
    T[t].maxc = T[t << 1].maxc + T[t << 1 | 1].maxc;
  } else {
    if (T[t << 1].max1 > T[t << 1 | 1].max1) {
      T[t].max1 = T[t << 1].max1;
      T[t].max2 = max(T[t << 1].max2, T[t << 1 | 1].max1);
      T[t].maxc = T[t << 1].maxc;
    } else {
      T[t].max1 = T[t << 1 | 1].max1;
      T[t].max2 = max(T[t << 1].max1, T[t << 1 | 1].max2);
      T[t].maxc = T[t << 1 | 1].maxc;
    }
  }

  // min
  if (T[t << 1].min1 == T[t << 1 | 1].min1) {
    T[t].min1 = T[t << 1].min1;
    T[t].min2 = min(T[t << 1].min2, T[t << 1 | 1].min2);
    T[t].minc = T[t << 1].minc + T[t << 1 | 1].minc;
  } else {
    if (T[t << 1].min1 < T[t << 1 | 1].min1) {
      T[t].min1 = T[t << 1].min1;
      T[t].min2 = min(T[t << 1].min2, T[t << 1 | 1].min1);
      T[t].minc = T[t << 1].minc;
    } else {
      T[t].min1 = T[t << 1 | 1].min1;
      T[t].min2 = min(T[t << 1].min1, T[t << 1 | 1].min2);
      T[t].minc = T[t << 1 | 1].minc;
    }
  }
}

void push_add(int t, int tl, int tr, ll v) {
  if (v == 0) { return; }
  T[t].sum += (tr - tl + 1) * v;
  T[t].max1 += v;
  if (T[t].max2 != -llINF) { T[t].max2 += v; }
  T[t].min1 += v;
  if (T[t].min2 != llINF) { T[t].min2 += v; }
  T[t].lazy += v;
```

```
}

// corresponds to a chmin update
void push_max(int t, ll v, bool l) {
  if (v >= T[t].max1) { return; }
  T[t].sum -= T[t].max1 * T[t].maxc;
  T[t].max1 = v;
  T[t].sum += T[t].max1 * T[t].maxc;
  if (l) {
    T[t].min1 = T[t].max1;
  } else {
    if (v <= T[t].min1) {
      T[t].min1 = v;
    } else if (v < T[t].min2) {
      T[t].min2 = v;
    }
  }
}

// corresponds to a chmax update
void push_min(int t, ll v, bool l) {
  if (v <= T[t].min1) { return; }
  T[t].sum -= T[t].min1 * T[t].minc;
  T[t].min1 = v;
  T[t].sum += T[t].min1 * T[t].minc;
  if (l) {
    T[t].max1 = T[t].min1;
  } else {
    if (v >= T[t].max1) {
      T[t].max1 = v;
    } else if (v > T[t].max2) {
      T[t].max2 = v;
    }
  }
}

void pushdown(int t, int tl, int tr) {
  if (tl == tr) return;
  // sum
  int tm = (tl + tr) >> 1;
  push_add(t << 1, tl, tm, T[t].lazy);
  push_add(t << 1 | 1, tm + 1, tr, T[t].lazy);
  T[t].lazy = 0;

  // max
  push_max(t << 1, T[t].max1, tl == tm);
  push_max(t << 1 | 1, T[t].max1, tm + 1 == tr);

  // min
  push_min(t << 1, T[t].min1, tl == tm);
  push_min(t << 1 | 1, T[t].min1, tm + 1 == tr);
}

void build(int t = 1, int tl = 0, int tr = N - 1) {
  T[t].lazy = 0;
  if (tl == tr) {
    T[t].sum = T[t].max1 = T[t].min1 = A[tl];
    T[t].maxc = T[t].minc = 1;
    T[t].max2 = -llINF;
    T[t].min2 = llINF;
    return;
  }

  int tm = (tl + tr) >> 1;
  build(t << 1, tl, tm);
  build(t << 1 | 1, tm + 1, tr);
  merge(t);
}

void update_add(int l, int r, ll v, int t = 1, int tl = 0,
                int tr = N - 1) {
  if (r < tl || tr < l) { return; }
  if (l <= tl && tr <= r) {
    push_add(t, tl, tr, v);
    return;
  }
  pushdown(t, tl, tr);

  int tm = (tl + tr) >> 1;
  update_add(l, r, v, t << 1, tl, tm);
  update_add(l, r, v, t << 1 | 1, tm + 1, tr);
  merge(t);
}

void update_chmin(int l, int r, ll v, int t = 1, int tl = 0,
                  int tr = N - 1) {
  if (r < tl || tr < l || v >= T[t].max1) { return; }
  if (l <= tl && tr <= r && v > T[t].max2) {
    push_max(t, v, tl == tr);
    return;
  }
  pushdown(t, tl, tr);

  int tm = (tl + tr) >> 1;
  update_chmin(l, r, v, t << 1, tl, tm);
  update_chmin(l, r, v, t << 1 | 1, tm + 1, tr);
  merge(t);
}

void update_chmax(int l, int r, ll v, int t = 1, int tl = 0,
                  int tr = N - 1) {
  if (r < tl || tr < l || v <= T[t].min1) { return; }
  if (l <= tl && tr <= r && v < T[t].min2) {
    push_min(t, v, tl == tr);
    return;
  }
  pushdown(t, tl, tr);

  int tm = (tl + tr) >> 1;
  update_chmax(l, r, v, t << 1, tl, tm);
  update_chmax(l, r, v, t << 1 | 1, tm + 1, tr);
  merge(t);
}

ll query_sum(int l, int r, int t = 1, int tl = 0,
             int tr = N - 1) {
  if (r < tl || tr < l) { return 0; }
  if (l <= tl && tr <= r) { return T[t].sum; }
  pushdown(t, tl, tr);

  int tm = (tl + tr) >> 1;
  return query_sum(l, r, t << 1, tl, tm) +
         query_sum(l, r, t << 1 | 1, tm + 1, tr);
}
```

2.8. Dynamic Connectivity

```
class DSU {
private:
  vector<ll> p, sz, sum;
  // stores all history info related to merges
  vector<pair<ll &, ll>> history;

public:
  DSU(int n) : p(n), sz(n, 1), sum(n) {
    iota(p.begin(), p.end(), 0);
  }

  void init_sum(const vector<ll> a) {
    for (int i = 0; i < (int)a.size(); i++) {
      sum[i] = a[i];
    }
  }

  int get(int x) { return (p[x] == x) ? x : get(p[x]); }

  ll get_sum(int x) { return sum[get(x)]; }

  void unite(int a, int b) {
    a = get(a);
    b = get(b);
    if (a == b) { return; }
    if (sz[a] < sz[b]) { swap(a, b); }

    // add to history
    history.push_back({p[b], p[b]});
    history.push_back({sz[a], sz[a]});
    history.push_back({sum[a], sum[a]});

    p[b] = a;
    sz[a] += sz[b];
    sum[a] += sum[b];
  }

  void add(int x, ll v) {
    x = get(x);
    history.push_back({sum[x], sum[x]});
    sum[x] += v;
  }

  int snapshot() { return history.size(); }

  void rollback(int until) {
    while (snapshot() > until) {
      history.back().first = history.back().second;
      history.pop_back();
    }
  }
};

const int MAXN = 3e5;

DSU dsu(MAXN);

struct Query {
  int t, u, v, x;
```

```
};

vector<Query> tree[MAXN * 4];

void update(Query &q, int v, int query_l, int query_r,
            int tree_l, int tree_r) {
  if (query_l > tree_r || query_r < tree_l) { return; }
  if (query_l <= tree_l && query_r >= tree_r) {
    tree[v].push_back(q);
    return;
  }
  int m = (tree_l + tree_r) / 2;
  update(q, v * 2, query_l, query_r, tree_l, m);
  update(q, v * 2 + 1, query_l, query_r, m + 1, tree_r);
}

void dfs(int v, int l, int r, vector<ll> &ans) {
  int snapshot = dsu.snapshot();
  // perform all available operations upon entering
  for (Query &q : tree[v]) {
    if (q.t == 1) { dsu.unite(q.u, q.v); }
    if (q.t == 2) { dsu.add(q.v, q.x); }
  }
  if (l == r) {
    // answer type 3 query if we have one
    for (Query &q : tree[v]) {
      if (q.t == 3) { ans[l] = dsu.get_sum(q.v); }
    }
  } else {
    // go deeper into the tree
    int m = (l + r) / 2;
    dfs(2 * v, l, m, ans);
    dfs(2 * v + 1, m + 1, r, ans);
  }
  // undo operations upon exiting
  dsu.rollback(snapshot);
}
```

3. Graph

3.1. SCC

3.1.1. Kosaraju

```
void dfs1(int u) {
  vis[u] = 1;
  for (int v : g[u])
    if (!vis[v]) dfs1(v);
  topo.push_back(u);
}

void dfs2(int u, int r) {
  root[u] = r;
  for (int v : g_rev[u])
    if (!root[v]) dfs2(v, r);
}

int main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);

  cin >> n >> m;
  int u, v;
  while (m--) {
    cin >> u >> v;
    g[u].push_back(v);
    g_rev[v].push_back(u);
  }
  for (int u = 1; u <= n; ++u)
    if (!vis[u]) dfs1(u);
  reverse(all(topo));
  int cnt = 0;
  for (int u : topo) {
    if (!root[u]) {
      dfs2(u, u);
      roots.push_back(
      u); // roots is also in topological order
      ++cnt;
    }
  }
  cout << cnt;
  for (int u = 1; u <= n; ++u) {
    for (int v : g[u])
      if (root[u] != root[v])
        g_scc[root[u]].push_back(root[v]);
  }

  return 0;
}
```

3.1.2. 2SAT

```
struct TwoSatSolver {
  int n_vars;
  int n_vertices;
  vector<vector<int>> adj, adj_t;
  vector<bool> used;
  vector<int> order, comp;
  vector<bool> assignment;

  TwoSatSolver(int _n_vars)
      : n_vars(_n_vars), n_vertices(2 * n_vars),
        adj(n_vertices), adj_t(n_vertices),
        used(n_vertices), order(), comp(n_vertices, -1),
        assignment(n_vars) {
    order.reserve(n_vertices);
  }
  void dfs1(int v) {
    used[v] = true;
    for (int u : adj[v]) {
      if (!used[u]) dfs1(u);
    }
    order.push_back(v);
  }

  void dfs2(int v, int cl) {
    comp[v] = cl;
    for (int u : adj_t[v]) {
      if (comp[u] == -1) dfs2(u, cl);
    }
  }

  bool solve_2SAT() {
    order.clear();
    used.assign(n_vertices, false);
    for (int i = 0; i < n_vertices; ++i) {
      if (!used[i]) dfs1(i);
    }

    comp.assign(n_vertices, -1);
    for (int i = 0, j = 0; i < n_vertices; ++i) {
      int v = order[n_vertices - i - 1];
      if (comp[v] == -1) dfs2(v, j++);
    }

    assignment.assign(n_vars, false);
    for (int i = 0; i < n_vertices; i += 2) {
      if (comp[i] == comp[i + 1]) return false;
      assignment[i / 2] = comp[i] > comp[i + 1];
    }
    return true;
  }

  void add_disjunction(int a, bool na, int b, bool nb) {
    // na and nb signify whether a and b are to be negated
    a = 2 * a ^ na;
    b = 2 * b ^ nb;
    int neg_a = a ^ 1;
    int neg_b = b ^ 1;
    adj[neg_a].push_back(b);
    adj[neg_b].push_back(a);
    adj_t[b].push_back(neg_a);
    adj_t[a].push_back(neg_b);
  }

  static void example_usage() {
    TwoSatSolver solver(3); // a, b, c
    solver.add_disjunction(0, false, 1,
                           true); //     a  v  not b
    solver.add_disjunction(0, true, 1,
                           true); // not a  v  not b
    solver.add_disjunction(1, false, 2,
                           false); //     b  v      c
    solver.add_disjunction(0, false, 0,
                           false); //     a  v      a
    assert(solver.solve_2SAT() == true);
    auto expected = vector<bool>(True, False, True);
    assert(solver.assignment == expected);
  }
};
```

3.2. Matching/Flows

3.2.1. Edmonds-Karp

```
int bfs(int s, int t, vector<int> &parent) {
  fill(parent.begin(), parent.end(), -1);
  parent[s] = -2;
  queue<pair<int, int>> q;
  q.push({s, INF});

  while (!q.empty()) {
    int cur = q.front().first;
```

```
    int flow = q.front().second;
    q.pop();

    for (int next : adj[cur]) {
      if (parent[next] == -1 && capacity[cur][next]) {
        parent[next] = cur;
        int new_flow = min(flow, capacity[cur][next]);
        if (next == t) return new_flow;
        q.push({next, new_flow});
      }
    }
  }

  return 0;
}

int maxflow(int s, int t) {
  int flow = 0;
  vector<int> parent(n);
  int new_flow;

  while (new_flow = bfs(s, t, parent)) {
    flow += new_flow;
    int cur = t;
    while (cur != s) {
      int prev = parent[cur];
      capacity[prev][cur] -= new_flow;
      capacity[cur][prev] += new_flow;
      cur = prev;
    }
  }

  return flow;
}
```

3.2.2. Dinic

```
void add_edge(int v, int u, long long cap) {
  edges.emplace_back(v, u, cap);
  edges.emplace_back(u, v, 0);
  adj[v].push_back(m);
  adj[u].push_back(m + 1);
  m += 2;
}

bool bfs() {
  while (!q.empty()) {
    int v = q.front();
    q.pop();
    for (int id : adj[v]) {
      if (edges[id].cap == edges[id].flow) continue;
      if (level[edges[id].u] != -1) continue;
      level[edges[id].u] = level[v] + 1;
      q.push(edges[id].u);
    }
  }
  return level[t] != -1;
}

long long dfs(int v, long long pushed) {
  if (pushed == 0) return 0;
  if (v == t) return pushed;
  for (int &cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
    int id = adj[v][cid];
    int u = edges[id].u;
    if (level[v] + 1 != level[u]) continue;
    long long tr =
    dfs(u, min(pushed, edges[id].cap - edges[id].flow));
    if (tr == 0) continue;
    edges[id].flow += tr;
    edges[id ^ 1].flow -= tr;
    return tr;
  }
  return 0;
}

long long flow() {
  long long f = 0;
  while (true) {
    fill(level.begin(), level.end(), -1);
    level[s] = 0;
    q.push(s);
    if (!bfs()) break;
    fill(ptr.begin(), ptr.end(), 0);
    while (long long pushed = dfs(s, flow_inf)) {
      f += pushed;
    }
  }
  return f;
}
```

3.2.3. Min-cost Flow

```
struct MCF {
  struct edge {
    ll to, from, cap, flow, cost, rev;
  } *fromE[MAXN];
  vector<edge> v[MAXN];
  ll n, s, t, flows[MAXN], dis[MAXN], pi[MAXN], flowlim;
  void make_edge(int s, int t, ll cap, ll cost) {
    if (!cap) return;
    v[s].pb(edge{t, s, cap, 0LL, cost, v[t].size()});
    v[t].pb(edge{s, t, 0LL, 0LL, -cost, v[s].size() - 1});
  }
  bitset<MAXN> vis;
  void dijkstra() {
    vis.reset();
    __gnu_pbds::priority_queue<pair<ll, int>> q;
    vector<decltype(q)::point_iterator> its(n);
    q.push({0LL, s});
    while (!q.empty()) {
      int now = q.top().second;
      q.pop();
      if (vis[now]) continue;
      vis[now] = 1;
      ll ndis = dis[now] + pi[now];
      for (edge &e : v[now]) {
        if (e.flow == e.cap || vis[e.to]) continue;
        if (dis[e.to] > ndis + e.cost - pi[e.to]) {
          dis[e.to] = ndis + e.cost - pi[e.to];
          flows[e.to] = min(flows[now], e.cap - e.flow);
          fromE[e.to] = &e;
          if (its[e.to] == q.end())
            its[e.to] = q.push({-dis[e.to], e.to});
          else q.modify(its[e.to], {-dis[e.to], e.to});
        }
      }
    }
  }
  bool AP(ll &flow) {
    fill_n(dis, n, INF);
    fromE[s] = 0;
    dis[s] = 0;
    flows[s] = flowlim - flow;
    dijkstra();
    if (dis[t] == INF) return false;
    flow += flows[t];
    for (edge *e = fromE[t]; e; e = fromE[e->from]) {
      e->flow += flows[t];
      v[e->to][e->rev].flow -= flows[t];
    }
    for (int i = 0; i < n; i++)
      pi[i] = min(pi[i] + dis[i], INF);
    return true;
  }
  pll solve(int _s, int _t, ll _flowlim = INF) {
    s = _s, t = _t, flowlim = _flowlim;
    pll re;
    while (re.F != flowlim && AP(re.F));
    for (int i = 0; i < n; i++)
      for (edge &e : v[i])
        if (e.flow != 0) re.S += e.flow * e.cost;
    re.S /= 2;
    return re;
  }
  void init(int _n) {
    n = _n;
    fill_n(pi, n, 0);
    for (int i = 0; i < n; i++) v[i].clear();
  }
  void setpi(int s) {
    fill_n(pi, n, INF);
    pi[s] = 0;
    for (ll it = 0, flag = 1, tdis; flag && it < n; it++) {
      flag = 0;
      for (int i = 0; i < n; i++)
        if (pi[i] != INF)
          for (edge &e : v[i])
            if (e.cap && (tdis = pi[i] + e.cost) < pi[e.to])
              pi[e.to] = tdis, flag = 1;
    }
  }
};
```

3.2.4. Kuhn

```
bool try_kuhn(int v) {
  if (used[v]) return false;
  used[v] = true;
  for (int to : g[v]) {
    if (mt[to] == -1 || try_kuhn(mt[to])) {
      mt[to] = v;
      return true;
    }
  }
  return false;
}
```

```
int main() {
  mt.assign(k, -1);
  for (int v = 0; v < n; ++v) {
    used.assign(n, false);
    try_kuhn(v);
  }
  for (int i = 0; i < k; ++i)
    if (mt[i] != -1) printf("%d %d\n", mt[i] + 1, i + 1);

  // heuristic faster
  mt.assign(k, -1);
  vector<bool> used1(n, false);
  for (int v = 0; v < n; ++v) {
    for (int to : g[v]) {
      if (mt[to] == -1) {
        mt[to] = v;
        used1[v] = true;
        break;
      }
    }
  }
  for (int v = 0; v < n; ++v) {
    if (used1[v]) continue;
    used.assign(n, false);
    try_kuhn(v);
  }

  for (int i = 0; i < k; ++i)
    if (mt[i] != -1) printf("%d %d\n", mt[i] + 1, i + 1);
}
```

3.3. Eulerian cycle

```
void dfs(int u) {
  while (!g[u].empty()) {
    int v = g[u].back().v;
    int eid = g[u].back().eid;
    g[u].pop_back();
    if (vis[eid])
      continue; // if directed graph then don't need vis
    vis[eid] = 1;
    dfs(v);
  }
  tour.push_back(u);
}

int main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);

  cin >> n >> m;
  int u, v;
  for (int i = 1; i <= m; ++i) {
    cin >> u >> v;
    g[u].push_back({v, i});
    g[v].push_back({u, i});
    ++deg[v];
    ++deg[u];
  }
  for (int u = 1; u <= n; ++u) {
    if (deg[u] % 2 !=
        0) { // if directed graph then degin[u] != degout[u]
      cout << "Not an eulerian graph";
      return 0;
    }
  }
  dfs(1);
  if (tour.size() != m + 1) cout << "Not an eulerian graph";
  else {
    cout << "Is an eulerian graph" << endl;
    for (int u : tour)
      cout
      << u
      << ' '; // if directed graph then need to reverse tour
  }

  return 0;
}
```

3.4. Bridge Tree

```
void dfs(int u) {
  tin[u] = low[u] = ++timer;
  comp.push_back(u);
  for (int i : g[u]) {
    if (vis[i]) continue;
    vis[i] = 1;
    int v = u ^ e[i].u ^ e[i].v;
    if (!tin[v]) {
      dfs(v);
      low[u] = min(low[u], low[v]);
      if (low[v] == tin[v]) {
        ++bridge;
        cnt[id[v] = ++cur] = 1;
        while (comp.back() != v) {
          ++cnt[id[comp.back()] = cur];
          comp.pop_back();
        }
        comp.pop_back();
      }
    } else low[u] = min(low[u], tin[v]);
  }
  if (u == 1) {
    cnt[id[u] = ++cur] = 1;
    while (comp.back() != u) {
      ++cnt[id[comp.back()] = cur];
      comp.pop_back();
    }
    comp.pop_back();
  }
}

void build_bt() {
  for (int u = 1; u <= n; ++u) {
    for (int i : g[u]) {
      int v = u ^ e[i].u ^ e[i].v;
      if (id[u] != id[v]) bt[id[u]].push_back(id[v]);
    }
  }
}
```

3.5. Block cut Tree

```
void dfs(int u, int p) {
  tin[u] = low[u] = ++timer;
  comp.push_back(u);
  for (int v : g[u]) {
    if (v == p) continue;
    if (!tin[v]) {
      dfs(v, u);
      low[u] = min(low[u], low[v]);
      if (low[v] >= tin[u]) {
        is_joint[u] = (tin[u] > 1 || tin[v] > 2);
        bccs.push_back({u});
        while (bccs.back().back() != v) {
          bccs.back().push_back(comp.back());
          comp.pop_back();
        }
      }
    } else low[u] = min(low[u], tin[v]);
  }
}

void build_bct() {
  int cur = 0;
  for (int u = 1; u <= n; ++u)
    if (is_joint[u]) id[u] = ++cur;
  for (const vector<int> &bcc : bccs) {
    ++cur;
    for (int u : bcc) {
      if (is_joint[u]) {
        bct[id[u]].push_back(cur);
        bct[cur].push_back(id[u]);
      } else id[u] = cur;
    }
  }
}
```

3.6. Bellman-Ford

```
void bellman_ford(int s) {
  fill(dist + 1, dist + n + 1, inf64);
  dist[s] = 0;
  for (int i = 1; i < n; ++i) {
    for (auto [u, v, w] : g) {
      if (dist[u] != inf64 && dist[v] > dist[u] + w) {
        dist[v] = dist[u] + w;
        trace[v] = u;
      }
    }
  }
}

bool find_negative_cycle(vector<int> &neg_cycle) {
  // afer Bellman-Ford, this only check negative cycle in
  // the component containing the source node
  int neg_start = -1;
  for (int i = 0; i < n; ++i) {
    for (auto &[u, v, w] : g) {
      if (dist[u] != inf64 && dist[v] > dist[u] + w) {
        dist[v] = -inf64;
        trace[v] = u;
        neg_start = v;
```

```
      }
    }
  }
  if (neg_start == -1) return 0;
  int u = neg_start;
  for (int i = 0; i < n; ++i) u = trace[u];
  neg_cycle = vector<int>(1, u);
  for (int v = trace[u]; v != u; v = trace[v])
    neg_cycle.push_back(v);
  neg_cycle.push_back(u);
  reverse(all(neg_cycle));
  return 1;
}

vector<int> trace_path(int s, int u) {
  vector<int> path;
  if (dist[u] == inf64) return path;
  for (int v = u; v != 0; v = trace[v]) path.push_back(v);
  reverse(all(path));
  return path;
}
```

3.7. HLD

```
int dfs(int u) {
  int sz = 1, mx_csz = 0;
  for (int v : g[u]) {
    if (v == par[u]) continue;
    par[v] = u;
    dep[v] = dep[u] + 1;
    int csz = dfs(v);
    sz += csz;
    if (csz > mx_csz) {
      mx_csz = csz;
      heavy[u] = v;
    }
  }
  return sz;
}

void hld(int u, int h, int &cur_pos) {
  head[u] = h;
  pos[u] = ++cur_pos;
  upd(1, pos[u], val[u], 1, n);
  if (heavy[u] != 0) hld(heavy[u], h, cur_pos);
  for (int v : g[u])
    if (v != par[u] && v != heavy[u]) hld(v, v, cur_pos);
}

long long get(int u, int v) {
  long long res = 0;
  while (head[u] != head[v]) {
    if (dep[head[u]] < dep[head[v]]) swap(u, v);
    res = max(res, get(1, pos[head[u]], pos[u], 1, n));
    u = par[head[u]];
  }
  if (dep[u] > dep[v]) swap(u, v);
  res = max(res, get(1, pos[u], pos[v], 1, n));
  return res;
}
```

4. Math

4.1. Chinese Remainder Theorem

```
struct Congruence {
  long long a, m;
};

long long chinese_remainder_theorem(
vector<Congruence> const &congruences) {
  long long M = 1;
  for (auto const &congruence : congruences) {
    M *= congruence.m;
  }

  long long solution = 0;
  for (auto const &congruence : congruences) {
    long long a_i = congruence.a;
    long long M_i = M / congruence.m;
    long long N_i = mod_inv(M_i, congruence.m);
    solution = (solution + a_i * M_i % M * N_i) % M;
  }
  return solution;
}
```

4.2. Diophantine Equation

```
int gcd(int a, int b, int &x, int &y) {
  if (b == 0) {
    x = 1;
    y = 0;
    return a;
  }
  int x1, y1;
  int d = gcd(b, a % b, x1, y1);
  x = y1;
  y = x1 - y1 * (a / b);
  return d;
}

bool find_any_solution(int a, int b, int c, int &x0,
                       int &y0, int &g) {
  g = gcd(abs(a), abs(b), x0, y0);
  if (c % g) { return false; }

  x0 *= c / g;
  y0 *= c / g;
  if (a < 0) x0 = -x0;
  if (b < 0) y0 = -y0;
  return true;
}
```

4.3. Miller-Rabin

```
using u64 = uint64_t;
using u128 = __uint128_t;

u64 binpower(u64 base, u64 e, u64 mod) {
  u64 result = 1;
  base %= mod;
  while (e) {
    if (e & 1) result = (u128)result * base % mod;
    base = (u128)base * base % mod;
    e >>= 1;
  }
  return result;
}

bool check_composite(u64 n, u64 a, u64 d, int s) {
  u64 x = binpower(a, d, n);
  if (x == 1 || x == n - 1) return false;
  for (int r = 1; r < s; r++) {
    x = (u128)x * x % n;
    if (x == n - 1) return false;
  }
  return true;
};

bool MillerRabin(
u64 n) { // returns true if n is prime, else returns false.
  if (n < 2) return false;

  int r = 0;
  u64 d = n - 1;
  while ((d & 1) == 0) {
    d >>= 1;
    r++;
  }

  for (int a :
       {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
    if (n == a) return true;
    if (check_composite(n, a, d, r)) return false;
  }
  return true;
}
```

4.4. Pollard's Rho

```
ll f(ll x, ll mod) { return (x * x + 1) % mod; }
// n should be composite
ll pollard_rho(ll n) {
  if (!(n & 1)) return 2;
  while (1) {
    ll y = 2, x = RNG() % (n - 1) + 1, res = 1;
    for (int sz = 2; res == 1; sz *= 2) {
      for (int i = 0; i < sz && res <= 1; i++) {
        x = f(x, n);
        res = __gcd(abs(x - y), n);
      }
      y = x;
    }
    if (res != 0 && res != n) return res;
  }
}
```

4.5. Gaussian Elimination

```
const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be
                   // infinity or a big number
```

```cpp
int gauss(vector<vector<double>> a, vector<double> &ans) {
  int n = (int)a.size();
  int m = (int)a[0].size() - 1;

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

    for (int i = 0; i < n; ++i)
      if (i != row) {
        double c = a[i][col] / a[row][col];
        for (int j = col; j <= m; ++j)
          a[i][j] -= a[row][j] * c;
      }
    ++row;
  }

  ans.assign(m, 0);
  for (int i = 0; i < m; ++i)
    if (where[i] != -1)
      ans[i] = a[where[i]][m] / a[where[i]][i];
  for (int i = 0; i < n; ++i) {
    double sum = 0;
    for (int j = 0; j < m; ++j) sum += ans[j] * a[i][j];
    if (abs(sum - a[i][m]) > EPS) return 0;
  }

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

4.6. FFT

```cpp
template <typename T>
void fft_(int n, vector<T> &a, vector<T> &rt, bool inv) {
  vector<int> br(n);
  for (int i = 1; i < n; i++) {
    br[i] = (i & 1) ? br[i - 1] + n / 2 : br[i / 2] / 2;
    if (br[i] > i) swap(a[i], a[br[i]]);
  }
  for (int len = 2; len <= n; len *= 2)
    for (int i = 0; i < n; i += len)
      for (int j = 0; j < len / 2; j++) {
        int pos = n / len * (inv ? len - j : j);
        T u = a[i + j], v = a[i + j + len / 2] * rt[pos];
        a[i + j] = u + v, a[i + j + len / 2] = u - v;
      }
  if (T minv = T(1) / T(n); inv)
    for (T &x : a) x *= minv;
}
```

4.7. General Lucas' Theorem

```cpp
ll crt(vector<ll> &x, vector<ll> &mod) {
  int n = x.size();
  ll M = 1;
  for (ll m : mod) M *= m;
  ll res = 0;
  for (int i = 0; i < n; i++) {
    ll out = M / mod[i];
    res += x[i] * inv(out, mod[i]) * out;
  }
  return res;
}
ll f(ll n, ll k, ll p, ll q) {
  auto fac = [](ll n, ll p, ll q) {
    ll x = 1, y = powi(p, q);
    for (int i = 2; i <= n; i++)
      if (i % p != 0) x = x * i % y;
    return x % y;
  };
  ll r = n - k, x = powi(p, q);
  ll e0 = 0, eq = 0;
  ll mul = (p == 2 && q >= 3) ? 1 : -1;
  ll cr = r, cm = k, car = 0, cnt = 0;
  while (cr || cm || car) {
    ll rr = cr % p, rm = cm % p;
    cnt++, car += rr + rm;
    if (car >= p) {
      e0++;
      if (cnt >= q) eq++;
    }
    car /= p, cr /= p, cm /= p;
  }
  mul = powi(p, e0) * powi(mul, eq);
  ll ret = (mul % x + x) % x;
  ll tmp = 1;
  for (;; tmp *= p) {
    ret = ret * fac(n / tmp % x, p, q) % x;
    ret = ret * inv(fac(n / tmp % x, p, q), x) % x;
    ret = ret * inv(fac(n / tmp % x, p, q), x) % x;
    if (tmp > n / p && tmp > k / p && tmp > r / p) break;
  }
  return (ret % x + x) % x;
}
int comb(ll n, ll k, int m) {
  int _m = m; // can use better factorization
  vector<ll> x, mod;
  for (int p = 2; p * p <= _m; p += 1 + (p & 1)) {
    if (_m % p == 0) {
      int q = 0;
      for (; _m % p == 0; _m /= p) q++;
      x.push_back(f(n, k, p, q));
      mod.push_back(powi(p, q));
    }
  }
  if (_m > 1)
    x.push_back(f(n, k, _m, 1)), mod.push_back(_m);
  return crt(x, mod) % m;
}
```

5. Geometry

5.1. Pick's theorem

i: number of integer points inside the polygon
b: number of integer points on the boundary

$$\text{Area} = i + \frac{b}{2} - 1$$

5.2. Chebyshev Distance

```cpp
pair<int, int> mantoche(int x, int y) {
  return {x + y, y - x};
}
```

5.3. Convex Hull

```cpp
struct pt {
  double x, y;
};

int orientation(pt a, pt b, pt c) {
  double v =
  a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x * (a.y - b.y);
  if (v < 0) return -1; // clockwise
  if (v > 0) return +1; // counter-clockwise
  return 0;
}

bool cw(pt a, pt b, pt c, bool include_collinear) {
  int o = orientation(a, b, c);
  return o < 0 || (include_collinear && o == 0);
}
bool ccw(pt a, pt b, pt c, bool include_collinear) {
  int o = orientation(a, b, c);
  return o > 0 || (include_collinear && o == 0);
}

void convex_hull(vector<pt> &a,
                 bool include_collinear = false) {
  if (a.size() == 1) return;

  sort(a.begin(), a.end(), [](pt a, pt b) {
    return make_pair(a.x, a.y) < make_pair(b.x, b.y);
  });
  pt p1 = a[0], p2 = a.back();
  vector<pt> up, down;
  up.push_back(p1);
  down.push_back(p1);
  for (int i = 1; i < (int)a.size(); i++) {
    if (i == a.size() - 1 ||
        cw(p1, a[i], p2, include_collinear)) {
      while (up.size() >= 2 &&
             !cw(up[up.size() - 2], up[up.size() - 1], a[i],
                 include_collinear))
        up.pop_back();
      up.push_back(a[i]);
    }
    if (i == a.size() - 1 ||
        ccw(p1, a[i], p2, include_collinear)) {
      while (down.size() >= 2 &&
             !ccw(down[down.size() - 2],
                  down[down.size() - 1], a[i],
                  include_collinear))
        down.pop_back();
```

```
      down.push_back(a[i]);
    }
  }

  if (include_collinear && up.size() == a.size()) {
    reverse(a.begin(), a.end());
    return;
  }
  a.clear();
  for (int i = 0; i < (int)up.size(); i++)
    a.push_back(up[i]);
  for (int i = down.size() - 2; i > 0; i--)
    a.push_back(down[i]);
}
```

5.4. Polygon Area

```
double area(const vector<point> &fig) {
  double res = 0;
  for (unsigned i = 0; i < fig.size(); i++) {
    point p = i ? fig[i - 1] : fig.back();
    point q = fig[i];
    res += (p.x - q.x) * (p.y + q.y);
  }
  return fabs(res) / 2;
}
```

5.5. Minkowski Sum

```
struct pt {
  long long x, y;
  pt operator+(const pt &p) const {
    return pt{x + p.x, y + p.y};
  }
  pt operator-(const pt &p) const {
    return pt{x - p.x, y - p.y};
  }
  long long cross(const pt &p) const {
    return x * p.y - y * p.x;
  }
};

void reorder_polygon(vector<pt> &P) {
  size_t pos = 0;
  for (size_t i = 1; i < P.size(); i++) {
    if (P[i].y < P[pos].y ||
        (P[i].y == P[pos].y && P[i].x < P[pos].x))
      pos = i;
  }
  rotate(P.begin(), P.begin() + pos, P.end());
}

vector<pt> minkowski(vector<pt> P, vector<pt> Q) {
  // the first vertex must be the lowest
  reorder_polygon(P);
  reorder_polygon(Q);
  // we must ensure cyclic indexing
  P.push_back(P[0]);
  P.push_back(P[1]);
  Q.push_back(Q[0]);
  Q.push_back(Q[1]);
  // main part
  vector<pt> result;
  size_t i = 0, j = 0;
  while (i < P.size() - 2 || j < Q.size() - 2) {
    result.push_back(P[i] + Q[j]);
    auto cross = (P[i + 1] - P[i]).cross(Q[j + 1] - Q[j]);
    if (cross >= 0 && i < P.size() - 2) ++i;
    if (cross <= 0 && j < Q.size() - 2) ++j;
  }
  return result;
}
```

5.6. Point In Polygon

```
struct pt {
  long long x, y;
  pt() {}
  pt(long long _x, long long _y) : x(_x), y(_y) {}
  pt operator+(const pt &p) const {
    return pt(x + p.x, y + p.y);
  }
  pt operator-(const pt &p) const {
    return pt(x - p.x, y - p.y);
  }
  long long cross(const pt &p) const {
    return x * p.y - y * p.x;
  }
  long long dot(const pt &p) const {
    return x * p.x + y * p.y;
  }
  long long cross(const pt &a, const pt &b) const {
    return (a - *this).cross(b - *this);
  }
  long long dot(const pt &a, const pt &b) const {
    return (a - *this).dot(b - *this);
  }
  long long sqrLen() const { return this->dot(*this); }
};

bool lexComp(const pt &l, const pt &r) {
  return l.x < r.x || (l.x == r.x && l.y < r.y);
}

int sgn(long long val) {
  return val > 0 ? 1 : (val == 0 ? 0 : -1);
}

vector<pt> seq;
pt translation;
int n;

bool pointInTriangle(pt a, pt b, pt c, pt point) {
  long long s1 = abs(a.cross(b, c));
  long long s2 = abs(point.cross(a, b)) +
                 abs(point.cross(b, c)) +
                 abs(point.cross(c, a));
  return s1 == s2;
}

void prepare(vector<pt> &points) {
  n = points.size();
  int pos = 0;
  for (int i = 1; i < n; i++) {
    if (lexComp(points[i], points[pos])) pos = i;
  }
  rotate(points.begin(), points.begin() + pos,
         points.end());

  n--;
  seq.resize(n);
  for (int i = 0; i < n; i++)
    seq[i] = points[i + 1] - points[0];
  translation = points[0];
}

bool pointInConvexPolygon(pt point) {
  point = point - translation;
  if (seq[0].cross(point) != 0 &&
      sgn(seq[0].cross(point)) !=
      sgn(seq[0].cross(seq[n - 1])))
    return false;
  if (seq[n - 1].cross(point) != 0 &&
      sgn(seq[n - 1].cross(point)) !=
      sgn(seq[n - 1].cross(seq[0])))
    return false;

  if (seq[0].cross(point) == 0)
    return seq[0].sqrLen() >= point.sqrLen();

  int l = 0, r = n - 1;
  while (r - l > 1) {
    int mid = (l + r) / 2;
    int pos = mid;
    if (seq[pos].cross(point) >= 0) l = mid;
    else r = mid;
  }
  int pos = l;
  return pointInTriangle(seq[pos], seq[pos + 1], pt(0, 0),
                         point);
}
```

5.7. Closest Pair

```
struct pt {
  int x, y, id;
};

struct cmp_x {
  bool operator()(const pt &a, const pt &b) const {
    return a.x < b.x || (a.x == b.x && a.y < b.y);
  }
};

struct cmp_y {
  bool operator()(const pt &a, const pt &b) const {
    return a.y < b.y;
  }
};

int n;
vector<pt> a;

double mindist;
pair<int, int> best_pair;
```

```
void upd_ans(const pt &a, const pt &b) {
  double dist = sqrt((a.x - b.x) * (a.x - b.x) +
                     (a.y - b.y) * (a.y - b.y));
  if (dist < mindist) {
    mindist = dist;
    best_pair = {a.id, b.id};
  }
}

vector<pt> t;

void rec(int l, int r) {
  if (r - l <= 3) {
    for (int i = l; i < r; ++i) {
      for (int j = i + 1; j < r; ++j) {
        upd_ans(a[i], a[j]);
      }
    }
    sort(a.begin() + l, a.begin() + r, cmp_y());
    return;
  }

  int m = (l + r) >> 1;
  int midx = a[m].x;
  rec(l, m);
  rec(m, r);

  merge(a.begin() + l, a.begin() + m, a.begin() + m,
        a.begin() + r, t.begin(), cmp_y());
  copy(t.begin(), t.begin() + r - l, a.begin() + l);

  int tsz = 0;
  for (int i = l; i < r; ++i) {
    if (abs(a[i].x - midx) < mindist) {
      for (int j = tsz - 1;
           j >= 0 && a[i].y - t[j].y < mindist; --j)
        upd_ans(a[i], t[j]);
      t[tsz++] = a[i];
    }
  }
}

main() {
  t.resize(n);
  sort(a.begin(), a.end(), cmp_x());
  mindist = 1E20;
  rec(0, n);
}
```

6. Strings

6.1. Aho-Corasick Automaton

```
struct Node {
  Node *child[26], *go[26], *link;
  int sum, topo;
  vector<int> query;

  Node() {
    for (int i = 0; i < 26; i++) { child[i] = go[i] = 0; }
    link = 0;
    sum = topo = 0;
  }
};

struct aho_corasick {
  Node *root;
  vector<Node *> v;

  aho_corasick() {
    root = new Node();
    root->link = root;
    for (int i = 0; i < 26; i++) {
      root->child[i] = new Node();
    }
  }

  void add_string(string &s, int id) {
    Node *p = root;
    for (int i = 0; i < (int)s.size(); i++) {
      int c = s[i] - 'a';
      if (p->child[c] == 0) { p->child[c] = new Node(); }
      p = p->child[c];
    }
    p->query.pb(id);
  }

  void bfs() {
    queue<Node *> q;
    q.push(root);
    while (q.size()) {
      Node *u = q.front();
      v.push_back(u);
      q.pop();
      for (int c = 0; c < 26; c++) {
        if (u->child[c]) {
          u->child[c]->link = u->link->go[c];
          if (u == root) { u->child[c]->link = root; }
          u->child[c]->link->topo++;
          u->go[c] = u->child[c];
          q.push(u->child[c]);
        } else {
          u->go[c] = u->link->go[c];
        }
      }
    }
  }

  void compute() {
    queue<Node *> q;
    for (auto x : v) {
      if (x->topo == 0) { q.push(x); }
    }
    while (q.size()) {
      Node *u = q.front();
      for (int x : u->query) { ans[x] = u->sum; }
      q.pop();
      u->link->topo--;
      u->link->sum += u->sum;
      if (u->link->topo == 0) { q.push(u->link); }
    }
  }
};
aho_corasick aho;
```

6.2. Knuth-Morris-Pratt Algorithm

```
vector<int> prefix_func(const string &s) {
  int n = isz(s);
  vector<int> pi(n);
  for (int i = 1; i < n; ++i) {
    int j = pi[i - 1];
    while (j > 0 && s[i] != s[j]) j = pi[j - 1];
    if (s[i] == s[j]) ++j;
    pi[i] = j;
  }
  return pi;
}
```

6.3. Suffix Array

```
void build_suf_arr() {
  for (int i = 0; i < n; ++i) ++cnt[a[i]];
  for (int i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
  for (int i = 0; i < n; ++i) p[--cnt[a[i]]] = i;
  c[p[0]] = 0;
  int cls = 1;
  for (int i = 1; i < n; ++i) {
    if (a[p[i]] != a[p[i - 1]]) ++cls;
    c[p[i]] = cls - 1;
  }

  for (int k = 0; MASK32(k) < n; ++k) {
    for (int i = 0; i < n; ++i)
      pn[i] = (p[i] - MASK32(k) + n) % n;
    fill(cnt, cnt + cls, 0);
    for (int i = 0; i < n; ++i) ++cnt[c[pn[i]]];
    for (int i = 1; i < cls; ++i) cnt[i] += cnt[i - 1];
    for (int i = n - 1; i >= 0; --i)
      p[--cnt[c[pn[i]]]] = pn[i];
    cn[p[0]] = 0;
    cls = 1;
    for (int i = 1; i < n; ++i) {
      pair<int, int> cur = {c[p[i]],
                            c[(p[i] + MASK32(k)) % n]};
      pair<int, int> prev = {c[p[i - 1]],
                             c[(p[i - 1] + MASK32(k)) % n]};
      if (cur != prev) ++cls;
      cn[p[i]] = cls - 1;
    }
    for (int i = 0; i < n; ++i) c[i] = cn[i];
  }

  for (int i = 0; i < n; ++i) rnk[p[i]] = i;
  for (int i = 0, k = 0; i < n; ++i) {
    if (rnk[i] == n - 1) {
      k = 0;
      continue;
    }
    int j = p[rnk[i] + 1];
    while (i + k < n && j + k < n && a[i + k] == a[j + k])
      ++k;
    lcp[rnk[i]] = k;
```

```
    if (k > 0) --k;
  }
}
```

6.4. Z Algorithm

```
vector<int> z_func(const string &s) {
  int n = isz(s);
  vector<int> z(n);
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
    if (i <= r) z[i] = min(z[i - l], r - i + 1);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
    if (i + z[i] - 1 > r) {
      l = i;
      r = i + z[i] - 1;
    }
  }
  return z;
}
```

6.5. Manacher's Algorithm

```
vector<int> manacher_odd(string s) {
  int n = s.size();
  s = "$" + s + "^";
  vector<int> p(n + 2);
  int l = 0, r = 1;
  for (int i = 1; i <= n; i++) {
    p[i] = min(r - i, p[l + (r - i)]);
    while (s[i - p[i]] == s[i + p[i]]) { p[i]++; }
    if (i + p[i] > r) { l = i - p[i], r = i + p[i]; }
  }
  return vector<int>(begin(p) + 1, end(p) - 1);
}

vector<int> manacher(string s) {
  string t;
  for (auto c : s) { t += string("#") + c; }
  auto res = manacher_odd(t + "#");
  return vector<int>(begin(res) + 1, end(res) - 1);
}
```