Contents

1 Funciones C++

#include <algorithm> #include <numeric>

Algo	**Params**	**Funcion**
sort, stable_sort	f, l	ordena el intervalo
nth_element	f, nth, l	*void* ordena el n-esimo, y particiona el resto
fill, fill_n	f, l / n, elem	*void* llena [f, l) o [f, f+n) con elem
lower_bound, upper_bound	f, l, elem	*it* al primer / ultimo donde se puede insertar elem para que quede ordenada
binary_search	f, l, elem	*bool* esta elem en [f, l)
copy	f, l, resul	hace resul+i=f+i $\forall i$
find, find_if, find_first_of	f, l, elem / pred / f2, l2	*it* encuentra i ∈[f,l) tq. i=elem, pred(i), i∈[f2,l2)
count, count_if	f, l, elem/pred	cuenta elem, pred(i)
search	f, l, f2, l2	busca [f2,l2) ∈ [f,l)
replace, replace_if	f, l, old / pred, new	cambia old / pred(i) por new
reverse	f, l	da vuelta
partition, stable_partition	f, l, pred	pred(i) ad, !pred(i) atras
min_element, max_element	f, l, [comp]	*it* min, max de [f,l]
lexicographical_compare	f1,l1,f2,l2	*bool* con [f1,l1]¡[f2,l2]
next/prev_permutation	f,l	deja en [f,l) la perm sig, ant
set_intersection, set_difference, set_union, set_symmetric_difference,	f1, l1, f2, l2, res	[res, ...) la op. de conj
push_heap, pop_heap, make_heap	f, l, e / e /	mete/saca e en heap [f,l), hace un heap de [f,l)
is_heap	f,l	*bool* es [f,l) un heap
accumulate	f,l,i,[op]	$T = \sum$/oper de [f,l)
inner_product	f1, l1, f2, i	T = i + [f1, l1) . [f2, ...)
partial_sum	f, l, r, [op]	r+i = $\sum$/oper de [f,f+i] $\forall i$ ∈[f,l)
__builtin_ffs	unsigned int	Pos. del primer 1 desde la derecha
__builtin_clz	unsigned int	Cant. de ceros desde la izquierda.
__builtin_ctz	unsigned int	Cant. de ceros desde la derecha.
__builtin_popcount	unsigned int	Cant. de 1's en x.
__builtin_parity	unsigned int	1 si x es par, 0 si es impar.
__builtin_XXXXXXll	unsigned ll	= pero para long long's.

2 Data Structures

2.1 SparseTable

```cpp
const int MAXN=100005, K=30;
int lg[MAXN+1];
int st[K + 1][MAXN];

int mini(int L, int R){
  int i = lg[R - L + 1];
  int minimum = min(st[i][L], st[i][R - (1 << i) + 1]);
  return minimum;
}

int main(){
  lg[1]=0;
  for (int i = 2; i <= MAXN; i++)
    lg[i] = lg[i/2] + 1;
  std::copy(a.begin(), a.end(), st[0]);

  for (int i = 1; i <= K; i++)
    for (int j = 0; j + (1 << i) <= n; j++)
        st[i][j] = min(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
}
```

2.2 Segment Tree

```cpp
struct SegmentTree {
  vector<ll> a;
  int n;

  SegmentTree(int _n) : a(2 * _n, 1e18), n(_n) {}

  void update(int pos, ll val) {
    for (a[pos += n] = val; pos > 1; pos >>= 1) {
      a[pos / 2] = min(a[pos], a[pos ^ 1]);
    }
  }

  ll get(int l, int r) {
    ll res = 1e18;
    for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
      if (l & 1) {
        res = min(res, a[l++]);
      }
      if (r & 1) {
        res = min(res, a[--r]);
      }
    }
    return res;
  }
};
```

2.3 Segment Tree Max Subarray Sum

```cpp
const ll inf=1e18;

struct Node {
  ll maxi, l_max, r_max, sum;

  Node(ll _maxi, ll _l_max, ll _r_max, ll _sum){
    maxi=_maxi;
    l_max=_l_max;
    r_max=_r_max;
    sum=_sum;
  }

  Node operator+(Node b) {
    return {max(max(maxi, b.maxi), r_max + b.l_max),
      max(l_max, sum + b.l_max), max(b.r_max, r_max + b.sum),
      sum + b.sum};
  }

};

struct SegmentTreeMaxSubSum{
  int n;
  vector<Node> t;

  SegmentTreeMaxSubSum(int _n) : n(_n), t(2 * _n, Node(-inf, -inf, -inf,
       -inf)) {}

  void update(int pos, ll val) {
    t[pos += n] = Node(val, val, val, val);
    for (pos>>=1; pos ; pos >>= 1) {
      t[pos] = t[2*pos]+t[2*pos+1];
```

```
    }
  }

  Node query(int l, int r) {
    Node node_l = Node(-inf, -inf, -inf, -inf);
    Node node_r = Node(-inf, -inf, -inf, -inf);
    for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
      if (l & 1) {
        node_l=node_l+t[l++];
      }
      if (r & 1) {
        node_r=t[--r]+node_r;
      }
    }
    return node_l+node_r;
  }
};
```

2.4 MergeSortTree

```
vector<int> t[200005];
int a[100005];
int n;

void build(){
  for(int i=0;i<n;i++){
    t[i+n].push_back(a[i]);
  }
  for(int i=n-1;i;i--){
    auto b=t[2*i], c=t[2*i+1];
    merge(b.begin(), b.end(), c.begin(), c.end(), back_inserter(t[i]));
  }
}


int q(int l, int r, int mid) {
  int res = 0;
  for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
    if (l&1){
        res+=upper_bound(all(t[l]), mid)-t[l].begin();
        l++;
    }
    if (r&1){
        r--;
        res+=upper_bound(all(t[r]), mid)-t[r].begin();
    }
  }
  return res;
}
```

2.5 LazySegmentTree

```
template <typename num_t>
struct segtree {
  int n, depth;
  vector<num_t> tree, lazy;

  void init(int s, long long* arr) {
    n = s;
    tree = vector<num_t>(4 * s, 0);
    lazy = vector<num_t>(4 * s, 0);
    init(0, 0, n - 1, arr);
  }

  num_t init(int i, int l, int r, long long* arr) {
    if (l == r) return tree[i] = arr[l];

    int mid = (l + r) / 2;
    num_t a = init(2 * i + 1, l, mid, arr),
          b = init(2 * i + 2, mid + 1, r, arr);
    return tree[i] = a.op(b);
  }

  void update(int l, int r, num_t v) {
  if (l > r) return;
    update(0, 0, n - 1, l, r, v);
  }

  num_t update(int i, int tl, int tr, int ql, int qr, num_t v) {
    eval_lazy(i, tl, tr);

  if (tr < ql || qr < tl) return tree[i];
    if (ql <= tl && tr <= qr) {
      lazy[i] = lazy[i].val + v.val;
      eval_lazy(i, tl, tr);
      return tree[i];
```

```
    }

    int mid = (tl + tr) / 2;
    num_t a = update(2 * i + 1, tl, mid, ql, qr, v),
          b = update(2 * i + 2, mid + 1, tr, ql, qr, v);
    return tree[i] = a.op(b);
  }

  num_t query(int l, int r) {
  if (l > r) return num_t::null_v;
    return query(0, 0, n-1, l, r);
  }

  num_t query(int i, int tl, int tr, int ql, int qr) {
    eval_lazy(i, tl, tr);

    if (ql <= tl && tr <= qr) return tree[i];
    if (tr < ql || qr < tl) return num_t::null_v;

    int mid = (tl + tr) / 2;
    num_t a = query(2 * i + 1, tl, mid, ql, qr),
          b = query(2 * i + 2, mid + 1, tr, ql, qr);
    return a.op(b);
  }

  void eval_lazy(int i, int l, int r) {
    tree[i] = tree[i].lazy_op(lazy[i], (r - l + 1));
    if (l != r) {
      lazy[i * 2 + 1] = lazy[i].val + lazy[i * 2 + 1].val;
      lazy[i * 2 + 2] = lazy[i].val + lazy[i * 2 + 2].val;
    }

    lazy[i] = num_t();
  }
};
```

```

int N, Q;
int a[maxN];

struct node {
  ll val;
  ll lzAdd;
  ll lzSet;
  node(){};
} tree[maxN << 2];

#define lc p << 1
#define rc (p << 1) + 1

inline void pushup(int p) {
  tree[p].val = tree[lc].val + tree[rc].val;
  return;
}

void pushdown(int p, int l, int mid, int r) {
  // lazy: range set
  if (tree[p].lzSet != 0) {
    tree[lc].lzSet = tree[rc].lzSet = tree[p].lzSet;
    tree[lc].val = (mid - l + 1) * tree[p].lzSet;
    tree[rc].val = (r - mid) * tree[p].lzSet;
    tree[lc].lzAdd = tree[rc].lzAdd = 0;
    tree[p].lzSet = 0;
  } else if (tree[p].lzAdd != 0) {  // lazy: range add
    if (tree[lc].lzSet == 0) tree[lc].lzAdd += tree[p].lzAdd;
    else {
      tree[lc].lzSet += tree[p].lzAdd;
      tree[lc].lzAdd = 0;
    }
    if (tree[rc].lzSet == 0) tree[rc].lzAdd += tree[p].lzAdd;
    else {
      tree[rc].lzSet += tree[p].lzAdd;
      tree[rc].lzAdd = 0;
    }
    tree[lc].val += (mid - l + 1) * tree[p].lzAdd;
    tree[rc].val += (r - mid) * tree[p].lzAdd;
    tree[p].lzAdd = 0;
  }
  return;
}

void build(int p, int l, int r) {
  tree[p].lzAdd = tree[p].lzSet = 0;
  if (l == r) {
    tree[p].val = a[l];
    return;
```

```
  }
  int mid = (l + r) >> 1;
  build(lc, l, mid);
  build(rc, mid + 1, r);
  pushup(p);
  return;
}

void add(int p, int l, int r, int a, int b, ll val) {
  if (a > r || b < l) return;
  if (a <= l && r <= b) {
    tree[p].val += (r - l + 1) * val;
    if (tree[p].lzSet == 0) tree[p].lzAdd += val;
    else tree[p].lzSet += val;
    return;
  }
  int mid = (l + r) >> 1;
  pushdown(p, l, mid, r);
  add(lc, l, mid, a, b, val);
  add(rc, mid + 1, r, a, b, val);
  pushup(p);
  return;
}

void set(int p, int l, int r, int a, int b, ll val) {
  if (a > r || b < l) return;
  if (a <= l && r <= b) {
    tree[p].val = (r - l + 1) * val;
    tree[p].lzAdd = 0;
    tree[p].lzSet = val;
    return;
  }
  int mid = (l + r) >> 1;
  pushdown(p, l, mid, r);
  set(lc, l, mid, a, b, val);
  set(rc, mid + 1, r, a, b, val);
  pushup(p);
  return;
}

ll query(int p, int l, int r, int a, int b) {
  if (a > r || b < l) return 0;
  if (a <= l && r <= b) return tree[p].val;
  int mid = (l + r) >> 1;
  pushdown(p, l, mid, r);
  return query(lc, l, mid, a, b) + query(rc, mid + 1, r, a, b);
}
```

2.6 PersistentSegmentTree

```
struct Node {
  ll val;
  Node *l, *r;

  Node(ll x) : val(x), l(nullptr), r(nullptr) {}
  Node(Node *_l, Node *_r) {
    l = _l, r = _r;
    val = 0;
    if (l) val += l->val;
    if (r) val += r->val;
  }
  Node(Node *cp) : val(cp->val), l(cp->l), r(cp->r) {}
};

int n, sz = 1;
ll a[200001];
Node *t[200001];

Node *build(int l = 1, int r = n) {
  if (l == r) return new Node(a[l]);
  int mid = (l + r) / 2;
  return new Node(build(l, mid), build(mid + 1, r));
}

Node *update(Node *node, int pos, int val, int l = 1, int r = n) {
  if (l == r) return new Node(val);
  int mid = (l + r) / 2;
  if (pos > mid)
    return new Node(node->l, update(node->r, pos, val, mid + 1, r));
  else return new Node(update(node->l, pos, val, l, mid), node->r);
}

ll query(Node *node, int a, int b, int l = 1, int r = n) {
  if (l > b || r < a) return 0;
  if (l >= a && r <= b) return node->val;
  int mid = (l + r) / 2;
```

```
  return query(node->l, a, b, l, mid) + query(node->r, a, b, mid + 1, r)
      ;
}

int main(){
  ios_base::sync_with_stdio(false); cin.tie(NULL);
  int q; cin >> n >> q;
  for(int i=1;i<=n;i++){
    cin >> a[i];
  }
  t[sz++]=build();
  while(q--){
    int ty; cin >> ty;
    if(ty==1){
      int k, pos, x; cin >> k >> pos >> x;
      t[k]=update(t[k], pos, x);
    }
    else if(ty==2){
      int k, l, r; cin >> k >> l >> r;
      cout << query(t[k], l, r) << endl;
    }
    else{
      int k; cin >> k;
      t[sz++]=new Node(t[k]);
    }
  }
}
```

2.7 Persistent Array

```
struct Node {
  int val;
  Node *l, *r;

  Node(ll x) : val(x), l(nullptr), r(nullptr) {}
  Node(Node *ll, Node *rr) : val(0), l(ll), r(rr) {}
};

int n, a[100001];     // The initial array and its size
Node *roots[100001];  // The persistent array's roots

Node *build(int l = 0, int r = n - 1) {
  if (l == r) return new Node(a[l]);
  int mid = (l + r) / 2;
  return new Node(build(l, mid), build(mid + 1, r));
}

Node *update(Node *node, int val, int pos, int l = 0, int r = n - 1) {
  if (l == r) return new Node(val);
  int mid = (l + r) / 2;
  if (pos > mid)
    return new Node(node->l, update(node->r, val, pos, mid + 1, r));
  else return new Node(update(node->l, val, pos, l, mid), node->r);
}

int query(Node *node, int pos, int l = 0, int r = n - 1) {
  if (l == r) return node->val;
  int mid = (l + r) / 2;
  if (pos > mid) return query(node->r, pos, mid + 1, r);
  return query(node->l, pos, l, mid);
}

int get_item(int index, int time) {
  // Gets the array item at a given index and time
  return query(roots[time], index);
}

void update_item(int index, int value, int prev_time, int curr_time) {
  // Updates the array item at a given index and time
  roots[curr_time] = update(roots[prev_time], index, value);
}

void init_arr(int nn, int *init) {
  // Initializes the persistent array, given an input array
  n = nn;
  for (int i = 0; i < n; i++) a[i] = init[i];
  roots[0] = build();
}
```

2.8 Fenwick

```
template <typename T>
struct Fenwick {
    int n;
    std::vector<T> a;

```

```
    Fenwick(int n_ = 0) {
        init(n_);
    }

    void init(int n_) {
        n = n_;
        a.assign(n, T{});
    }

    void add(int x, const T &v) {
        for (int i = x + 1; i <= n; i += i & -i) {
            a[i - 1] = a[i - 1] + v;
        }
    }

    T sum(int x) {
        T ans{};
        for (int i = x; i > 0; i -= i & -i) {
            ans = ans + a[i - 1];
        }
        return ans;
    }

    T rangeSum(int l, int r) {
        return sum(r) - sum(l);
    }

    int select(const T &k) {
        int x = 0;
        T cur{};
        for (int i = 1 << std::__lg(n); i; i /= 2) {
            if (x + i <= n && cur + a[x + i - 1] <= k) {
                x += i;
                cur = cur + a[x - 1];
            }
        }
        return x;
    }
};
```

2.9 Fenwick 2D

```
struct Fenwick2D{
  vector<vector<ll>> b;
  int n;

  Fenwick2D(int _n) : b(_n+5, vector<ll>(_n+5, 0)), n(_n) {}

  void update(int x, int y, int val){
    for(; x<=n; x+=(x&-x)){
      for(int j=y;j<=n;j+=(j&-j)){
        b[x][j]+=val;
      }
    }
  }

  ll get(int x, int y){
    ll ans=0;
    for(; x; x-=x&-x){
      for(int j=y; j ;j-=j&-j){
        ans+=b[x][j];
      }
    }
    return ans;
  }

  ll get1(int x1, int y1, int x2, int y2){
    return get(x2, y2)-get(x1-1, y2)-get(x2, y1-1)+ get(x1-1, y1-1);
  }

};
```

2.10 SegmentTree 2D

```
void build_y(int vx, int lx, int rx, int vy, int ly, int ry) {
    if (ly == ry) {
        if (lx == rx)
            t[vx][vy] = a[lx][ly];
        else
            t[vx][vy] = t[vx*2][vy] + t[vx*2+1][vy];
    } else {
        int my = (ly + ry) / 2;
        build_y(vx, lx, rx, vy*2, ly, my);
        build_y(vx, lx, rx, vy*2+1, my+1, ry);
        t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
    }
```

```
}

void build_x(int vx, int lx, int rx) {
    if (lx != rx) {
        int mx = (lx + rx) / 2;
        build_x(vx*2, lx, mx);
        build_x(vx*2+1, mx+1, rx);
    }
    build_y(vx, lx, rx, 1, 0, m-1);
}

int sum_y(int vx, int vy, int tly, int try_, int ly, int ry) {
    if (ly > ry)
        return 0;
    if (ly == tly && try_ == ry)
        return t[vx][vy];
    int tmy = (tly + try_) / 2;
    return sum_y(vx, vy*2, tly, tmy, ly, min(ry, tmy))
         + sum_y(vx, vy*2+1, tmy+1, try_, max(ly, tmy+1), ry);
}

int sum_x(int vx, int tlx, int trx, int lx, int rx, int ly, int ry) {
    if (lx > rx)
        return 0;
    if (lx == tlx && trx == rx)
        return sum_y(vx, 1, 0, m-1, ly, ry);
    int tmx = (tlx + trx) / 2;
    return sum_x(vx*2, tlx, tmx, lx, min(rx, tmx), ly, ry)
         + sum_x(vx*2+1, tmx+1, trx, max(lx, tmx+1), rx, ly, ry);
}


void update_y(int vx, int lx, int rx, int vy, int ly, int ry, int x, int
     y, int new_val) {
    if (ly == ry) {
        if (lx == rx)
            t[vx][vy] = new_val;
        else
            t[vx][vy] = t[vx*2][vy] + t[vx*2+1][vy];
    } else {
        int my = (ly + ry) / 2;
        if (y <= my)
            update_y(vx, lx, rx, vy*2, ly, my, x, y, new_val);
        else
            update_y(vx, lx, rx, vy*2+1, my+1, ry, x, y, new_val);
        t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
    }
}

void update_x(int vx, int lx, int rx, int x, int y, int new_val) {
    if (lx != rx) {
        int mx = (lx + rx) / 2;
        if (x <= mx)
            update_x(vx*2, lx, mx, x, y, new_val);
        else
            update_x(vx*2+1, mx+1, rx, x, y, new_val);
    }
    update_y(vx, lx, rx, 1, 0, m-1, x, y, new_val);
}
```

2.11 DynamicSegTree

```
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
            );
    }
};
```

2.12 OrderedSet

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

// to allow repetitions
#define ordered_set tree<int, null_type,less_equal<int>, rb_tree_tag,
    tree_order_statistics_node_update>

// to not allow repetitions
#define ordered_set tree<int, null_type,less<int>, rb_tree_tag,
    tree_order_statistics_node_update>

//order_of_key(x): number of items are strictly smaller than x

//find_by_order(k) iterator to the kth element
```

2.13 Disjoint Set Union

```
struct DSU {
  vector<int> e;
  vector<pair<int, int>> st;

  DSU(int N) : e(N, -1) {}

  int get(int x) { return e[x] < 0 ? x : e[x] = get(e[x]); }

  bool connected(int a, int b) { return get(a) == get(b); }

  int size(int x) { return -e[get(x)]; }

  bool unite(int x, int y) {
    x = get(x), y = get(y);
    if (x == y) { return false; }
    if (e[x] > e[y]) { swap(x, y); }
    st.push_back({x, e[x]});
    st.push_back({y, e[y]});
    e[x] += e[y];
    e[y] = x;
    return true;
  }

  //skip if no rollback
  int time() {return (int)st.size()};

  void rollback(int t) {
    for (int i = time(); i --> t;)
      e[st[i].first] = st[i].second;
    st.resize(t);
  }
};


//dsu for checking parity of path length (can be used for checking
    bipartiteness)
void make_set(int v) {
    parent[v] = make_pair(v, 0);
    rank[v] = 0;
    bipartite[v] = true;
}

pair<int, int> find_set(int v) {
    if (v != parent[v].first) {
        int parity = parent[v].second;
        parent[v] = find_set(parent[v].first);
        parent[v].second ^= parity;
    }
    return parent[v];
}
```

```cpp
void add_edge(int a, int b) {
    pair<int, int> pa = find_set(a);
    a = pa.first;
    int x = pa.second;

    pair<int, int> pb = find_set(b);
    b = pb.first;
    int y = pb.second;

    if (a == b) {
        if (x == y)
            bipartite[a] = false;
    } else {
        if (rank[a] < rank[b])
            swap (a, b);
        parent[b] = make_pair(a, x^y^1);
        bipartite[a] &= bipartite[b];
        if (rank[a] == rank[b])
            ++rank[a];
    }
}

bool is_bipartite(int v) {
    return bipartite[find_set(v).first];
}
```

2.14 BitTrie

```cpp
const int K = 2;
struct Vertex {
    int next[K];

    Vertex() {
        fill(begin(next), end(next), -1);
    }
};


//insert
for(int j=30;j>=0;j--) {
    int c = 1&(a[i]>>j);
    if (trie[v].next[c] == -1) {
        trie[v].next[c] = trie.size();
        trie.emplace_back();
        d.pb(-1);
    }
    v = trie[v].next[c];
}
```

2.15 Palindromic Tree

```cpp
const int N = 3e5 + 9;

/*
-> cnt contains the number of palindromic suffixes of the node
*/
struct PalindromicTree {
  struct node {
    int nxt[26], len, st, en, link, cnt, oc;
  };
  string s;
  vector<node> t;
  int sz, last;
  PalindromicTree() {}
  PalindromicTree(string _s) {
    s = _s;
    int n = s.size();
    t.clear();
    t.resize(n + 9);
    sz = 2, last = 2;
    t[1].len = -1, t[1].link = 1;
    t[2].len = 0, t[2].link = 1;
  }
  int extend(int pos) { // returns 1 if it creates a new palindrome
    int cur = last, curlen = 0;
    int ch = s[pos] - 'a';
    while (1) {
      curlen = t[cur].len;
      if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) break;
      cur = t[cur].link;
    }
    if (t[cur].nxt[ch]) {
      last = t[cur].nxt[ch];
      t[last].oc++;
      return 0;
    }
```

```
    sz++;
    last = sz;
    t[sz].oc = 1;
    t[sz].len = t[cur].len + 2;
    t[cur].nxt[ch] = sz;
    t[sz].en = pos;
    t[sz].st = pos - t[sz].len + 1;
    if (t[sz].len == 1) {
      t[sz].link = 2;
      t[sz].cnt = 1;
      return 1;
    }
    while (1) {
      cur = t[cur].link;
      curlen = t[cur].len;
      if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
        t[sz].link = t[cur].nxt[ch];
        break;
      }
    }
    t[sz].cnt = 1 + t[t[sz].link].cnt;
    return 1;
  }
  void calc_occurrences() {
    for (int i = sz; i >= 3; i--) t[t[i].link].oc += t[i].oc;
  }
} t;

int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  string s;
  cin >> s;
  PalindromicTree t(s);
  for (int i = 0; i < s.size(); i++) t.extend(i);
  t.calc_occurrences();
  long long ans = 0; // number of palindromes
  for (int i = 3; i <= t.sz; i++) ans += t.t[i].oc;
  cout << ans << '\n';
  return 0;
}
```

2.16 LineContainer

```
//Queries for maximum point x. To change this modify first comparator.
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;
  }
};
```

2.17 SqrtDecomposition

```

int n, numBlocks;
string s;

struct Block{
  int l, r;
  int sz(){
    return r-l;
  }
```

```
};

Block blocks[2*MAXI];
Block newBlocks[2*MAXI];

void rebuildDecomp(){
  string newS=s;
  int k=0;
  for(int i=0;i<numBlocks;i++){
    for(int j=blocks[i].l;j<blocks[i].r;j++){
      newS[k++]=s[j];
    }
  }
  numBlocks=1;
  blocks[0]={0, n};
  s=newS;
}

void cut(int a, int b){
  int pos=0, curBlock=0;
  for(int i=0;i<numBlocks;i++){
    Block B=blocks[i];
    bool containsA = pos < a && pos + B.sz() > a;
    bool containsB = pos < b && pos + B.sz() > b;
    int cutA = B.l + a - pos;
    int cutB = B.l + b - pos;
    if(containsA && containsB){
      newBlocks[curBlock++]={B.l, cutA};
      newBlocks[curBlock++]={cutA, cutB};
      newBlocks[curBlock++]={cutB, B.r};
    }
    else if(containsA){
      newBlocks[curBlock++]={B.l, cutA};
      newBlocks[curBlock++]={cutA, B.r};
    }
    else if(containsB){
      newBlocks[curBlock++]={B.l, cutB};
      newBlocks[curBlock++]={cutB, B.r};
    }
    else{
      newBlocks[curBlock++]=B;
    }
    pos += B.sz();
  }
  pos=0;
  numBlocks=0;
  for(int i=0;i<curBlock;i++){
    if(pos<a || pos>=b){
      blocks[numBlocks++]=newBlocks[i];
    }
    pos+=newBlocks[i].sz();
  }
  pos=0;
  for(int i=0;i<curBlock;i++){
    if(pos>=a && pos<b){
      blocks[numBlocks++]=newBlocks[i];
    }
    pos+=newBlocks[i].sz();
  }
}

// while doing operations
if(numBlocks>MAXI){
  rebuildDecomp();
}

// rebuild before final ans
rebuildDecomp();
cout << ans << endl;
```

2.18 Treap

```
typedef struct item * pitem;
struct item {
    int prior, value, cnt;
    bool rev;
    pitem l, r;
};

int cnt (pitem it) {
    return it ? it->cnt : 0;
}

void upd_cnt (pitem it) {
    if (it)
        it->cnt = cnt(it->l) + cnt(it->r) + 1;
```

```
}

void push (pitem it) {
    if (it && it->rev) {
        it->rev = false;
        swap (it->l, it->r);
        if (it->l)  it->l->rev ^= true;
        if (it->r)  it->r->rev ^= true;
    }
}

void merge (pitem & t, pitem l, pitem r) {
    push (l);
    push (r);
    if (!l || !r)
        t = l ? l : r;
    else if (l->prior > r->prior)
        merge (l->r, l->r, r),  t = l;
    else
        merge (r->l, l, r->l),  t = r;
    upd_cnt (t);
}

void split (pitem t, pitem & l, pitem & r, int key, int add = 0) {
    if (!t)
        return void( l = r = 0 );
    push (t);
    int cur_key = add + cnt(t->l);
    if (key <= cur_key)
        split (t->l, l, t->l, key, add),  r = t;
    else
        split (t->r, t->r, r, key, add + 1 + cnt(t->l)),  l = t;
    upd_cnt (t);
}

void reverse (pitem t, int l, int r) {
    pitem t1, t2, t3;
    split (t, t1, t2, l);
    split (t2, t2, t3, r-l+1);
    t2->rev ^= true;
    merge (t, t1, t2);
    merge (t, t, t3);
}

void output (pitem t) {
    if (!t)  return;
    push (t);
    output (t->l);
    printf ("%d ", t->value);
    output (t->r);
}
```

2.19 Dynammic Connectivity

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

struct DSU {
  vector<int> e;
  vector<pair<int, int>> st;
  int cnt;

  DSU(){}

  DSU(int N) : e(N, -1), cnt(N) {}

  int get(int x) { return e[x] < 0 ? x : get(e[x]);}

  bool connected(int a, int b) { return get(a) == get(b); }

  int size(int x) { return -e[get(x)]; }

  bool unite(int x, int y) {
    x = get(x), y = get(y);
    if (x == y) { return false; }
    if (e[x] > e[y]) { swap(x, y); }
    st.push_back({x, e[x]});
    st.push_back({y, e[y]});
    e[x] += e[y];
    e[y] = x;
    cnt--;
    return true;
  }

```

```
  void rollback(){
    auto [x, y]=st.back();
    st.pop_back();
    e[x] = y;
    auto [a, b]=st.back();
    st.pop_back();
    e[a]=b;
    cnt++;
  }
};

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

struct QueryTree {
  vector<vector<query>> t;
  DSU dsu;
  int T;

  QueryTree(){}

  QueryTree(int _T, int n) : T(_T) {
    dsu = DSU(n);
    t.resize(4 * T + 4);
  }

  void add(int v, int l, int r, int ul, int ur, query& q) {
    if (ul > ur)
      return;
    if (l == ul && r == ur) {
      t[v].push_back(q);
      return;
    }
    int mid = (l + r) / 2;
    add(2 * v, l, mid, ul, min(ur, mid), q);
    add(2 * v + 1, mid + 1, r, max(ul, mid + 1), ur, q);
  }

  void add_query(query q, int l, int r) {
    add(1, 0, T - 1, l, r, q);
  }

  void dfs(int v, int l, int r, vector<int>& ans) {
    for (query& q : t[v]) {
      q.united = dsu.unite(q.v, q.u);
    }
    if (l == r)
      ans[l] = dsu.cnt;
    else {
      int mid = (l + r) / 2;
      dfs(2 * v, l, mid, ans);
      dfs(2 * v + 1, mid + 1, r, ans);
    }
    for (query q : t[v]) {
      if (q.united)
        dsu.rollback();
    }
  }
};


int main(){
  ios_base::sync_with_stdio(false); cin.tie(NULL);
  //freopen("connect.in", "r", stdin);
  //freopen("connect.out", "w", stdout);
  int n, k; cin >> n >> k;
  if(k==0) return 0;
  QueryTree st=QueryTree(k, n);
  map<pair<int, int>, int> mp;
  vector<int> ans(k), q;
  for(int i=0;i<k;i++){
    char c; cin >> c;
    if(c=='?'){
      q.push_back(i);
      continue;
    }
    int u, v; cin >> u >> v;
    u--; v--;
    if(u>v) swap(u, v);
    if(c=='+'){
      mp[{u, v}]=i;
    }
    else{
```

```
      st.add_query(query(u, v), mp[{u, v}], i);
      mp[{u, v}]=-1;
    }
  }
  for(auto [x, y]:mp){
    if(y!=-1){
      st.add_query(query(x.first, x.second), y, k-1);
    }
  }
  st.dfs(1, 0, k-1, ans);
  for(int x:q){
    cout << ans[x] << endl;
  }
}
```

2.20 Bitset

```
bitset<3001> b[3001];

//set() Set the bit value at the given index to 1.
//count() Count the number of set bits.
//any() Checks if any bit is set
//all() Check if all bit is set.
```

3 Math

3.1 Binary Exponentiation And Modular Division

```
ll binpow(ll a, ll b){
    ll r=1;
    while(b){
        if(b%2)
            r=(r*a)%MOD;
        a=(a*a)%MOD;
        b/=2;
    }
    return r;
}

ll divide(ll a, ll b){
    return ((a%MOD)*binpow(b, MOD-2))%MOD;
}
void inverses(long long p) {
  inv[MAXN] = exp(fac[MAXN], p - 2, p);
  for (int i = MAXN; i >= 1; i--) { inv[i - 1] = inv[i] * i % p; }
}
```

3.2 Euler's Totient

```
//counts coprimes to each number from 1 to n
vector<int> phi1(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; 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;
        }
    }
    return phi1;
}
```

3.3 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(ll n) {
    if (n < 2)
        return false;

    int r = 0;
    ll 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;
}
```

3.4 Diophantine

If one solution is (x_0, y_0) all solutions can be obtained by $x = x_0 + k * \frac{b}{gcd(a,b)}$ and $y = y_0 - k * \frac{a}{gcd(a,b)}$.

```
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;
}



//n variables
vector<ll> find_any_solution(vector<ll> a, ll c) {
  int n = a.size();
  vector<ll> x;
  bool all_zero = true;
  for (int i = 0; i < n; i++) {
    all_zero &= a[i] == 0;
  }
  if (all_zero) {
    if (c) return {};
    x.assign(n, 0);
    return x;
  }
  ll g = 0;
  for (int i = 0; i < n; i++) {
    g = __gcd(g, a[i]);
  }
  if (c % g != 0) return {};
  if (n == 1) {
    return {c / a[0]};
  }
  vector<ll> suf_gcd(n);
  suf_gcd[n - 1] = a[n - 1];
  for (int i = n - 2; i >= 0; i--) {
    suf_gcd[i] = __gcd(suf_gcd[i + 1], a[i]);
  }
  ll cur = c;
```

```
  for (int i = 0; i + 1 < n; i++) {
    ll x0, y0, g;
    // solve for a[i] * x + suf_gcd[i + 1] * (y / suf_gcd[i + 1]) = cur
    bool ok = find_any_solution(a[i], suf_gcd[i + 1], cur, x0, y0, g);
    assert(ok);
    {
      // trying to minimize x0 in case x0 becomes big
      // it is needed for this problem, not needed in general
      ll shift = abs(suf_gcd[i + 1] / g);
      x0 = (x0 % shift + shift) % shift;
    }
    x.push_back(x0);

    // now solve for the next suffix
    cur -= a[i] * x0;
  }
  x.push_back(a[n - 1] == 0 ? 0 : cur / a[n - 1]);
  return x;
}
```

3.5 Discrete Log

Finds discrete logarithm in $O(\sqrt{(m)})$.

```
// 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;
}
```

4 Polynomials

4.1 Fast Fourier Transform (FFT)

Fast Fourier Transform in $O(nlogn)$.

```
using cd = complex<double>;
const double PI = acos(-1);
//declare size of vectors used like this
const int MAXN=2<<19;

void fft(vector<cd> & a, bool invert) {
  int n = (int)a.size();

  for (int i = 1, j = 0; i < n; i++) {
    int bit = n >> 1;
    for (; j & bit; bit >>= 1)
      j ^= bit;
    j ^= bit;

    if (i < j)
      swap(a[i], a[j]);
  }

  for (int len = 2; len <= n; len <<= 1) {
    double ang = 2 * PI / len * (invert ? -1 : 1);
    cd wlen(cos(ang), sin(ang));
    for (int i = 0; i < n; i += len) {
      cd w(1);
      for (int j = 0; j < len / 2; j++) {
  cd u = a[i+j], v = a[i+j+len/2] * w;
  a[i+j] = u + v;
  a[i+j+len/2] = u - v;
  w *= wlen;
      }
    }
  }

  if (invert) {
    for (cd & x : a)
      x /= n;
  }
}

vector<int> multiply(vector<int> const& a, vector<int> const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}

//normalizing for when mult is between 2 big numbers and not polynomials
int carry = 0;
for (int i = 0; i < n; i++){
    result[i] += carry;
    carry = result[i] / 10;
    result[i] %= 10;
}
```

4.2 Number Theoretic Transform (NTT)

```
const int mod = 7340033;
const int root = 5;
const int root_1 = 4404020;
const int root_pw = 1 << 20;

//Coefficients modulo prime p

//declare size of vectors used like this
const int MAXN=2<<19;

void fft(vector<int> & a, bool invert) {
    int n = a.size();
```

```
13
14      for (int i = 1, j = 0; i < n; i++) {
15          int bit = n >> 1;
16          for (; j & bit; bit >>= 1)
17              j ^= bit;
18          j ^= bit;
19
20          if (i < j)
21              swap(a[i], a[j]);
22      }
23
24      for (int len = 2; len <= n; len <<= 1) {
25          int wlen = invert ? root_1 : root;
26          for (int i = len; i < root_pw; i <<= 1)
27              wlen = (int)(1LL * wlen * wlen % mod);
28
29          for (int i = 0; i < n; i += len) {
30              int w = 1;
31              for (int j = 0; j < len / 2; j++) {
32                  int u = a[i+j], v = (int)(1LL * a[i+j+len/2] * w % mod);
33                  a[i+j] = u + v < mod ? u + v : u + v - mod;
34                  a[i+j+len/2] = u - v >= 0 ? u - v : u - v + mod;
35                  w = (int)(1LL * w * wlen % mod);
36              }
37          }
38      }
39
40      if (invert) {
41          int n_1 = inverse(n, mod);
42          for (int & x : a)
43              x = (int)(1LL * x * n_1 % mod);
44      }
45  }
```

4.3 Berlekamp-Massey

```
1   template<typename T>
2   vector<T> berlekampMassey(const vector<T> &s) {
3       vector<T> c;    // the linear recurrence sequence we are building
4       vector<T> oldC; // the best previous version of c to use (the one
            with the rightmost left endpoint)
5       int f = -1;     // the index at which the best previous version of c
             failed on
6       for (int i=0; i<(int)s.size(); i++) {
7           // evaluate c(i)
8           // delta = s_i - \sum_{j=1}^n c_j s_{i-j}
9           // if delta == 0, c(i) is correct
10          T delta = s[i];
11          for (int j=1; j<=(int)c.size(); j++)
12              delta -= c[j-1] * s[i-j];   // c_j is one-indexed, so we
                    actually need index j - 1 in the code
13          if (delta == 0)
14              continue;   // c(i) is correct, keep going
15          // now at this point, delta != 0, so we need to adjust it
16          if (f == -1) {
17              // this is the first time we're updating c
18              // s_i was the first non-zero element we encountered
19              // we make c of length i + 1 so that s_i is part of the base
                     case
20              c.resize(i + 1);
21              mt19937 rng(chrono::steady_clock::now().time_since_epoch().
                    count());
22              for (T &x : c)
23                  x = rng();  // just to prove that the initial values don
                        't matter in the first step, I will set to random
                        values
24              f = i;
25          } else {
26              // we need to use a previous version of c to improve on this
                     one
27              // apply the 5 steps to build d
28              // 1. set d equal to our chosen sequence
29              vector<T> d = oldC;
30              // 2. multiply the sequence by -1
31              for (T &x : d)
32                  x = -x;
33              // 3. insert a 1 on the left
34              d.insert(d.begin(), 1);
35              // 4. multiply the sequence by delta / d(f + 1)
36              T df1 = 0;  // d(f + 1)
37              for (int j=1; j<=(int)d.size(); j++)
38                  df1 += d[j-1] * s[f+1-j];
39              assert(df1 != 0);
40              T coef = delta / df1;   // storing this in outer variable so
                     it's O(n^2) instead of O(n^2 log MOD)
41              for (T &x : d)
```

```
                x *= coef;
            // 5. insert i - f - 1 zeros on the left
            vector<T> zeros(i - f - 1);
            zeros.insert(zeros.end(), d.begin(), d.end());
            d = zeros;
            // now we have our new recurrence: c + d
            vector<T> temp = c; // save the last version of c because it
                 might have a better left endpoint
            c.resize(max(c.size(), d.size()));
            for (int j=0; j<(int)d.size(); j++)
                c[j] += d[j];
            // finally, let's consider updating oldC
            if (i - (int) temp.size() > f - (int) oldC.size()) {
                // better left endpoint, let's update!
                oldC = temp;
                f = i;
            }
        }
    }
    return c;
}
```

5 Linear Algebra

5.1 Simplex

```
/*
Parametric Self-Dual Simplex method
Solve a canonical LP:
  min or max. c x
  s.t. A x <= b
    x >= 0
*/
#include <bits/stdc++.h>
using namespace std;
const double eps = 1e-9, oo = numeric_limits<double>::infinity();

typedef vector<double> vec;
typedef vector<vec> mat;

pair<vec, double> simplexMethodPD(const mat &A, const vec &b, const vec
    &c, bool mini = true){
  int n = c.size(), m = b.size();
  mat T(m + 1, vec(n + m + 1));
  vector<int> base(n + m), row(m);

  for(int j = 0; j < m; ++j){
    for(int i = 0; i < n; ++i)
      T[j][i] = A[j][i];
    row[j] = n + j;
    T[j][n + j] = 1;
    base[n + j] = 1;
    T[j][n + m] = b[j];
  }

  for(int i = 0; i < n; ++i)
    T[m][i] = c[i] * (mini ? 1 : -1);

  while(true){
    int p = 0, q = 0;
    for(int i = 0; i < n + m; ++i)
      if(T[m][i] <= T[m][p])
        p = i;

    for(int j = 0; j < m; ++j)
      if(T[j][n + m] <= T[q][n + m])
        q = j;

    double t = min(T[m][p], T[q][n + m]);

    if(t >= -eps){
      vec x(n);
      for(int i = 0; i < m; ++i)
        if(row[i] < n) x[row[i]] = T[i][n + m];
      return {x, T[m][n + m] * (mini ? -1 : 1)}; // optimal
    }

    if(t < T[q][n + m]){
      // tight on c -> primal update
      for(int j = 0; j < m; ++j)
        if(T[j][p] >= eps)
          if(T[j][p] * (T[q][n + m] - t) >= T[q][p] * (T[j][n + m] - t))
            q = j;

      if(T[q][p] <= eps)
        return {vec(n), oo * (mini ? 1 : -1)}; // primal infeasible
```

```
    }else{
      // tight on b -> dual update
      for(int i = 0; i < n + m + 1; ++i)
        T[q][i] = -T[q][i];

      for(int i = 0; i < n + m; ++i)
        if(T[q][i] >= eps)
          if(T[q][i] * (T[m][p] - t) >= T[q][p] * (T[m][i] - t))
            p = i;

      if(T[q][p] <= eps)
        return {vec(n), oo * (mini ? -1 : 1)}; // dual infeasible
    }

    for(int i = 0; i < m + n + 1; ++i)
      if(i != p) T[q][i] /= T[q][p];

    T[q][p] = 1; // pivot(q, p)
    base[p] = 1;
    base[row[q]] = 0;
    row[q] = p;

    for(int j = 0; j < m + 1; ++j){
      if(j != q){
        double alpha = T[j][p];
        for(int i = 0; i < n + m + 1; ++i)
          T[j][i] -= T[q][i] * alpha;
      }
    }
  }

  return {vec(n), oo};
}

int main(){
  int m, n;
  bool mini = true;
  cout << "Numero de restricciones: ";
  cin >> m;
  cout << "Numero de incognitas: ";
  cin >> n;
  mat A(m, vec(n));
  vec b(m), c(n);
  for(int i = 0; i < m; ++i){
    cout << "Restriccion #" << (i + 1) << ": ";
    for(int j = 0; j < n; ++j){
      cin >> A[i][j];
    }
    cin >> b[i];
  }
  cout << "[0]Max o [1]Min?: ";
  cin >> mini;
  cout << "Coeficientes de " << (mini ? "min" : "max") << " z: ";
  for(int i = 0; i < n; ++i){
    cin >> c[i];
  }
  cout.precision(6);
  auto ans = simplexMethodPD(A, b, c, mini);
  cout << (mini ? "Min" : "Max") << " z = " << ans.second << ", cuando: \n";
  for(int i = 0; i < ans.first.size(); ++i){
    cout << "x_" << (i + 1) << " = " << ans.first[i] << "\n";
  }
  return 0;
}
```

6 Graphs

6.1 Djikstra

```
priority_queue <ll> q;
dist[0]=0;
q.push({0, 0});
while((int)q.size()){
    auto [w, v]=q.top(); q.pop();
    w=-w;
    if (w>=dist[v]) continue;
    for (auto [u, p]:adj[v]){
        if(dist[v]+p<dist[u]){
            dist[u]=dist[v]+p;
            q.push({-dist[p], u});
        }
    }
}
```

6.2 FloydWarshall

```
vector<vector<ll>> d(n, vector<ll>(n, 1e18));

for (int k = 0; k < n; k++) {
  for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
      long long new_dist = d[i][k] + d[k][j];
      if (new_dist < d[i][j]) {
        d[i][j] = d[j][i] = new_dist;
      }
    }
  }
}
```

6.3 BellmanFord

```
void solve()
{
    vector<int> d(n, INF);
    d[v] = 0;
    vector<int> p(n, -1);
    int x;
    for (int i = 0; i < n; ++i) {
        x = -1;
        for (Edge e : edges)
            if (d[e.a] < INF)
                if (d[e.b] > d[e.a] + e.cost) {
                    d[e.b] = max(-INF, d[e.a] + e.cost);
                    p[e.b] = e.a;
                    x = e.b;
                }
    }

    if (x == -1)
        cout << "No negative cycle from " << v;
    else {
        int y = x;
        for (int i = 0; i < n; ++i)
            y = p[y];

        vector<int> path;
        for (int cur = y;; cur = p[cur]) {
            path.push_back(cur);
            if (cur == y && path.size() > 1)
                break;
        }
        reverse(path.begin(), path.end());

        cout << "Negative cycle: ";
        for (int u : path)
            cout << u << ' ';
    }
}
```

6.4 BellmanFordFast(SPFA)

```
const int INF = 1000000000;
vector<vector<pair<int, int>>> adj;

bool spfa(int s, vector<int>& d) {
    int n = adj.size();
    d.assign(n, INF);
    vector<int> cnt(n, 0);
    vector<bool> inqueue(n, false);
    queue<int> q;

    d[s] = 0;
    q.push(s);
    inqueue[s] = true;
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        inqueue[v] = false;

        for (auto edge : adj[v]) {
            int to = edge.first;
            int len = edge.second;

            if (d[v] + len < d[to]) {
                d[to] = d[v] + len;
                if (!inqueue[to]) {
                    q.push(to);
                    inqueue[to] = true;
                    cnt[to]++;
                    if (cnt[to] > n)
                        return false;  // negative cycle
                }
```

```
            }
        }
    }
    return true;
}
```

6.5 EulerianPath

An Eulerian Path is a path that passes through every edge once. For an undirected graph an eulerian path exists if the degree of every node is even or the degree of exactly two nodes is odd. In the first case, the eulerian path is also an eulerian circuit or cycle. In a directed graph, an eulerian path exists if at most one noda has $out_i - in_i = 1$ and at most one node has $in_i - out_i = 1$. A cycle exists if $in_i - out_i = 0$ for all i.

```
// check if all edges are visite
// directed
void dfs(int node) {
  while (!g[node].empty()) {
    auto [son, idx] = g[node].back();
    g[node].pop_back();
    if (seen[idx]) { continue; }
    seen[idx] = true;
    dfs(son);
  }
  path.push_back(node);
}
//undirected
void dfs(int node) {
  while (!g[node].empty()) {
    int son = g[node].back();
    g[node].pop_back();
    dfs(son);
  }
  path.push_back(node);
}
```

6.6 Kruskal

```
template <class T> T kruskal(int N, vector<pair<T, pair<int, int>>>
    edges) {
  sort(edges.begin(), edges.end());
  T ans = 0;
  DSU D(N + 1);  // edges that unite are in MST
  for (pair<T, pair<int, int>> &e : edges) {
    if (D.unite(e.second.first, e.second.second)) { ans += e.first; }
  }
  // -1 if the graph is not connected, otherwise the sum of the edge
      lengths
  return (D.size(1) == N ? ans : -1);
}
```

6.7 Strongly Connected Components

```
const int MAXN=100005;
vector<int> adj[MAXN];
vector<int> adj1[MAXN];
int comp[MAXN];
bool vis[MAXN];
vector<int> order;
int component=0;

void dfs(int cur){
  vis[cur]=1;
  for (int x: adj[cur]){
    if(!vis[x]){
      dfs(x);
    }
  }
  order.push_back(cur);
}

void dfs1(int cur){
  vis[cur]=1;
  comp[cur]=component;
  for (int x: adj1[cur]){
    if(!vis[x]){
      dfs1(x);
    }
  }
}
```

6.8 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==1 means not a
        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);
    }
};
```

6.9 Finding Bridges

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
        }
    }
```

```
}

void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}
```

6.10 Finding Bridges Online

```
vector<int> par, dsu_2ecc, dsu_cc, dsu_cc_size;
int bridges;
int lca_iteration;
vector<int> last_visit;

void init(int n) {
    par.resize(n);
    dsu_2ecc.resize(n);
    dsu_cc.resize(n);
    dsu_cc_size.resize(n);
    lca_iteration = 0;
    last_visit.assign(n, 0);
    for (int i=0; i<n; ++i) {
        dsu_2ecc[i] = i;
        dsu_cc[i] = i;
        dsu_cc_size[i] = 1;
        par[i] = -1;
    }
    bridges = 0;
}

int find_2ecc(int v) {
    if (v == -1)
        return -1;
    return dsu_2ecc[v] == v ? v : dsu_2ecc[v] = find_2ecc(dsu_2ecc[v]);
}

int find_cc(int v) {
    v = find_2ecc(v);
    return dsu_cc[v] == v ? v : dsu_cc[v] = find_cc(dsu_cc[v]);
}

void make_root(int v) {
    int root = v;
    int child = -1;
    while (v != -1) {
        int p = find_2ecc(par[v]);
        par[v] = child;
        dsu_cc[v] = root;
        child = v;
        v = p;
    }
    dsu_cc_size[root] = dsu_cc_size[child];
}

void merge_path (int a, int b) {
    ++lca_iteration;
    vector<int> path_a, path_b;
    int lca = -1;
    while (lca == -1) {
        if (a != -1) {
            a = find_2ecc(a);
            path_a.push_back(a);
            if (last_visit[a] == lca_iteration){
                lca = a;
                break;
                }
            last_visit[a] = lca_iteration;
            a = par[a];
        }
        if (b != -1) {
            b = find_2ecc(b);
            path_b.push_back(b);
            if (last_visit[b] == lca_iteration){
                lca = b;
                break;
                }
            last_visit[b] = lca_iteration;
            b = par[b];
        }

```

```
    }

    for (int v : path_a) {
        dsu_2ecc[v] = lca;
        if (v == lca)
            break;
        --bridges;
    }
    for (int v : path_b) {
        dsu_2ecc[v] = lca;
        if (v == lca)
            break;
        --bridges;
    }
}

void add_edge(int a, int b) {
    a = find_2ecc(a);
    b = find_2ecc(b);
    if (a == b)
        return;

    int ca = find_cc(a);
    int cb = find_cc(b);

    if (ca != cb) {
        ++bridges;
        if (dsu_cc_size[ca] > dsu_cc_size[cb]) {
            swap(a, b);
            swap(ca, cb);
        }
        make_root(a);
        par[a] = dsu_cc[a] = b;
        dsu_cc_size[cb] += dsu_cc_size[a];
    } else {
        merge_path(a, b);
    }
}
```

6.11 Articulation Points

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v);
}

void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs (i);
    }
}
```

6.12 Biconnected Components (Block Cut Tree)

Builds the Block Cut Tree of graph in $O(n)$.

```
vector<vector<int>> biconnected_components(vector<vector<int>> &g,

                                           vector<bool> &is_cutpoint,
                                           vector<int> &id) {
```

```
  int n = (int)g.size();
  vector<vector<int>> comps;
  vector<int> stk;
  vector<int> num(n);
  vector<int> low(n);
  is_cutpoint.resize(n);
  // Finds the biconnected components
  function<void(int, int, int &)> dfs = [&](int node, int parent, int &
      timer) {
    num[node] = low[node] = ++timer;
    stk.push_back(node);
    for (int son : g[node]) {
      if (son == parent) { continue; }
      if (num[son]) {
        low[node] = min(low[node], num[son]);
      } else {
        dfs(son, node, timer);
        low[node] = min(low[node], low[son]);
        if (low[son] >= num[node]) {
          is_cutpoint[node] = (num[node] > 1 || num[son] > 2);
          comps.push_back({node});
          while (comps.back().back() != son) {
            comps.back().push_back(stk.back());
            stk.pop_back();
          }
        }
      }
    }
  };

  int timer = 0;
  dfs(0, -1, timer);
  id.resize(n);
  // Build the block-cut tree

  function<vector<vector<int>>()> build_tree = [&]() {
    vector<vector<int>> t(1);
    int node_id = 0;
    for (int node = 0; node < n; node++) {
      if (is_cutpoint[node]) {
        id[node] = node_id++;
        t.push_back({});
      }
    }
    for (auto &comp : comps) {
      int node = node_id++;
      t.push_back({});
      for (int u : comp)
        if (!is_cutpoint[u]) {
          id[u] = node;
        } else {
          t[node].push_back(id[u]);
          t[id[u]].push_back(node);
        }
    }
    return t;
  };
  return build_tree();

}
```

6.13 Blossom

Finds maximum matching in general graph in $O(n^3)$.

```
const int N = 2e3 + 9;

mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());
struct Blossom {
  int vis[N], par[N], orig[N], match[N], aux[N], t;
  int n;
  bool ad[N];
  vector<int> g[N];
  queue<int> Q;
  Blossom() {}
  Blossom(int _n) {
    n = _n;
    t = 0;
    for (int i = 0; i <= _n; ++i) {
      g[i].clear();
      match[i] = aux[i] = par[i] = vis[i] = aux[i] = ad[i] = orig[i] =
          0;
    }
  }
```

```
  void add_edge(int u, int v) {
    g[u].push_back(v);
    g[v].push_back(u);
  }
  void augment(int u, int v) {
    int pv = v, nv;
    do {
      pv = par[v];
      nv = match[pv];
      match[v] = pv;
      match[pv] = v;
      v = nv;
    } while (u != pv);
  }
  int lca(int v, int w) {
    ++t;
    while (true) {
      if (v) {
        if (aux[v] == t) return v;
        aux[v] = t;
        v = orig[par[match[v]]];
      }
      swap(v, w);
    }
  }
  void blossom(int v, int w, int a) {
    while (orig[v] != a) {
      par[v] = w;
      w = match[v];
      ad[v] = true;
      if (vis[w] == 1) Q.push(w), vis[w] = 0;
      orig[v] = orig[w] = a;
      v = par[w];
    }
  }
  //it finds an augmented path starting from u
  bool bfs(int u) {
    fill(vis + 1, vis + n + 1, -1);
    iota(orig + 1, orig + n + 1, 1);
    Q = queue<int> ();
    Q.push(u);
    vis[u] = 0;
    while (!Q.empty()) {
      int v = Q.front();
      Q.pop();
      ad[v] = true;
      for (int x : g[v]) {
        if (vis[x] == -1) {
          par[x] = v;
          vis[x] = 1;
          if (!match[x]) return augment(u, x), true;
          Q.push(match[x]);
          vis[match[x]] = 0;
        } else if (vis[x] == 0 && orig[v] != orig[x]) {
          int a = lca(orig[v], orig[x]);
          blossom(x, v, a);
          blossom(v, x, a);
        }
      }
    }
    return false;
  }
  int maximum_matching() {
    int ans = 0;
    vector<int> p(n - 1);
    iota(p.begin(), p.end(), 1);
    shuffle(p.begin(), p.end(), rnd);
    for (int i = 1; i <= n; i++) shuffle(g[i].begin(), g[i].end(), rnd);
    for (auto u : p) {
      if (!match[u]) {
        for(auto v : g[u]) {
          if (!match[v]) {
            match[u] = v, match[v] = u;
            ++ans;
            break;
          }
        }
      }
    }
    for(int i = 1; i <= n; ++i) if (!match[i] && bfs(i)) ++ans;
    return ans;
  }
} M;
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
```

```
  int t;
  cin >> t;
  while (t--) {
    int n, m;
    cin >> n >> m;
    M = Blossom (n);
    while (m--) {
      int u, v;
      cin >> u >> v;
      M.add_edge(u, v);
    }
    int ans = M.maximum_matching();
    if (ans * 2 == n) cout << 0 << '\n';
    else {
      memset(M.ad, 0, sizeof M.ad);
      for (int i = 1; i <= n; i++) if (M.match[i] == 0) M.bfs(i);
      int ans = 0;
      for (int i = 1; i <= n; i++) ans += M.ad[i]; //nodes which are
          unmatched in some maximum matching
      cout << ans << '\n';
    }
  }
  return 0;
}
```

6.14 Marriage

```
// MALE OPTIMAL STABLE MARRIAGE PROBLEM O(N^2)
// gv[i][j] jth most prefered female for ith male
// om[i][j] jth most prefered male for ith female
#define MAXN 1000
int gv[MAXN][MAXN],om[MAXN][MAXN];
int pv[MAXN],pm[MAXN];              // ans
int pun[MAXN];                      // Auxiliary

void stableMarriage(int n) {
  fill_n(pv,n,-1); fill_n(pm,n,-1); fill_n(pun,n,0);
  int s = n, i = n-1;
  #define engage pm[j] = i; pv[i] = j;
  while (s) {
    while (pv[i] == -1) {
      int j = gv[i][pun[i]++];
      if (pm[j] == -1) {
        s--;
        engage;
      }
      else if (om[j][i] < om[j][pm[j]]) {
        int loser = pm[j];
        pv[loser] = -1;
        engage;
        i = loser;
      }
    }
    i--;
    if (i < 0) i = n-1;
  }
}
```

7 Trees

7.1 Tree Diameter

```
pair<int, int> dfs(const vector<vector<int>> &tree, int node = 1,
  int previous = 0, int length = 0) {
  pair<int, int> max_path = {node, length};
  for (const int &i : tree[node]) {
    if (i == previous) { continue; }
    pair<int, int> other = dfs(tree, i, node, length + 1);
    if (other.second > max_path.second) { max_path = other; }
  }
  return max_path;
}
```

7.2 Heavy Light Decomposition

```
//call dfs1 first
struct SegmentTree {
  vector<ll> a;
  int n;

  SegmentTree(int _n) : a(2 * _n, 0), n(_n) {}

  void update(int pos, ll val) {
    for (a[pos += n] = val; pos > 1; pos >>= 1) {
      a[pos / 2] = (a[pos]^a[pos ^ 1]);
```

```
    }
  }

  ll get(int l, int r) {
    ll res = 0;
    for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
      if (l & 1) {
        res^=a[l++];
      }
      if (r & 1) {
        res^=a[--r];
      }
    }
    return res;
  }
};


const int MAXN=500005;
vector<int> adj[MAXN];
SegmentTree st(MAXN);
int a[MAXN], sz[MAXN], to[MAXN], dpth[MAXN], s[MAXN], par[MAXN];
int cnt=0;

void dfs1(int cur, int p){
    sz[cur]=1;
    for(int x:adj[cur]){
        if(x==p) continue;
        dpth[x]=dpth[cur]+1;
        par[x]=cur;
        dfs1(x, cur);
        sz[cur]+=sz[x];
    }
}

void dfs(int cur, int p, int l){
    st.update(cnt, a[cur]);
    s[cur]=cnt++;
    to[cur]=l;
    int g=-1;
    for(int x:adj[cur]){
        if(x==p) continue;
        if(g==-1 || sz[g]<sz[x]){
            g=x;
        }
    }
    if(g==-1) return;
    dfs(g, cur, l);
    for(int x:adj[cur]){
        if(x==p || x==g) continue;
        dfs(x, cur, x);
    }
}

int query(int u, int v){
    int res=0;
    while(to[u]!=to[v]){
        if(dpth[to[u]]<dpth[to[v]]) swap(u, v);
        res^=st.get(s[to[u]], s[u]+1);
        u=par[to[u]];
    }
    if(dpth[u]>dpth[v]) swap(u, v);
    res^=st.get(s[u], s[v]+1);
    return res;
}



//alternate implementation
vector<int> parent, depth, heavy, head, pos;
int cur_pos;

int dfs(int v, vector<vector<int>> const& adj) {
    int size = 1;
    int max_c_size = 0;
    for (int c : adj[v]) {
        if (c != parent[v]) {
            parent[c] = v, depth[c] = depth[v] + 1;
            int c_size = dfs(c, adj);
            size += c_size;
            if (c_size > max_c_size)
                max_c_size = c_size, heavy[v] = c;
        }
    }
    return size;
```

```
}

void decompose(int v, int h, vector<vector<int>> const& adj) {
    head[v] = h, pos[v] = cur_pos++;
    if (heavy[v] != -1)
        decompose(heavy[v], h, adj);
    for (int c : adj[v]) {
        if (c != parent[v] && c != heavy[v])
            decompose(c, c, adj);
    }
}

void init(vector<vector<int>> const& adj) {
    int n = adj.size();
    parent = vector<int>(n);
    depth = vector<int>(n);
    heavy = vector<int>(n, -1);
    head = vector<int>(n);
    pos = vector<int>(n);
    cur_pos = 0;

    dfs(0, adj);
    decompose(0, 0, adj);
}

int query(int a, int b) {
    int res = 0;
    for (; head[a] != head[b]; b = parent[head[b]]) {
        if (depth[head[a]] > depth[head[b]])
            swap(a, b);
        int cur_heavy_path_max = segment_tree_query(pos[head[b]], pos[b
            ]);
        res = max(res, cur_heavy_path_max);
    }
    if (depth[a] > depth[b])
        swap(a, b);
    int last_heavy_path_max = segment_tree_query(pos[a], pos[b]);
    res = max(res, last_heavy_path_max);
    return res;
}
```

7.3 Lowest Common Ancestor (LCA)

```
const int N=200005;
vector<int> adj[N];
vector<int> start(N), end1(N), depth(N);
vector<vector<int>> t(N, vi(32));
int timer=0;
int n, l;
// l=(int)ceil(log2(n))
// call dfs(1, 1, 0)
// 1 indexed, dont use 0 indexing


void dfs(int cur, int p, int cnt){
    depth[cur]=cnt;
    t[cur][0]=p;
    start[cur]=timer++;
    for(int i=1;i<=l;i++){
        t[cur][i]=t[t[cur][i-1]][i-1];
    }
    for(int x:adj[cur]){
        if(x==p) continue;
        dfs(x, cur, cnt+1);
    }
    end1[cur]=++timer;
}

bool ancestor(int u, int v){
    return start[u]<=start[v] && end1[u]>=end1[v];
}

int lca(int u, int v){
    if(ancestor(u, v))
        return u;
    if (ancestor(v, u)){
        return v;
    }
    for(int i=l;i>=0;i--){
        if(!ancestor(t[u][i], v)){
            u=t[u][i];
        }
    }
    return t[u][0];
}
```

7.4 Centroid Decomposition

```cpp
// code for xenia and tree
const int MAXN=200005;

vector<int> adj[MAXN];
vector<bool> is_removed(MAXN, false);
vector<int> subtree_size(MAXN, 0);
vector<int> dis(MAXN, 1e9);
vector<vector<pair<int, int>>> ancestor(MAXN);

int get_subtree_size(int node, int parent = -1) {
  subtree_size[node] = 1;
  for (int child : adj[node]) {
    if (child == parent || is_removed[child]) { continue; }
    subtree_size[node] += get_subtree_size(child, node);
  }
  return subtree_size[node];
}

int get_centroid(int node, int tree_size, int parent = -1) {
  for (int child : adj[node]) {
    if (child == parent || is_removed[child]) { continue; }
    if (subtree_size[child] * 2 > tree_size) {
      return get_centroid(child, tree_size, node);
    }
  }
  return node;
}

void getDist(int cur, int centroid, int p=-1, int dist=1){
    for (int child:adj[cur]){
        if(child==p ||is_removed[child])
            continue;
        dist++;
        getDist(child, centroid, cur, dist);
        dist--;
    }
    ancestor[cur].push_back(make_pair(centroid, dist));
}

void update(int cur){
    for (int i=0;i<ancestor[cur].size();i++){
        dis[ancestor[cur][i].first]=min(dis[ancestor[cur][i].first],
            ancestor[cur][i].second);
    }
    dis[cur]=0;
}

int query(int cur){
    int mini=dis[cur];
    for (int i=0;i<ancestor[cur].size();i++){
       mini=min(mini, ancestor[cur][i].second+dis[ancestor[cur][i].first
           ]);
    }
    return mini;
}

void build_centroid_decomp(int node = 1) {
  int centroid = get_centroid(node, get_subtree_size(node));

  for (int child : adj[centroid]) {
    if (is_removed[child]) { continue; }
    getDist(child, centroid, centroid);
  }

  is_removed[centroid] = true;

  for (int child : adj[centroid]) {
    if (is_removed[child]) { continue; }
    build_centroid_decomp(child);
  }
}
```

8 Flows

8.1 Dinic

```cpp
struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
```

```
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    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 < 1)
                    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] || edges[id].cap - edges[id].
                flow < 1)
                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;
    }
};
```

8.2 Hungarian

```
#define forn(i,n) for(int i=0;i<int(n);++i)
#define forsn(i,s,n) for(int i=s;i<int(n);++i)
#define forall(i,c) for(typeof(c.begin()) i=c.begin();i!=c.end();++i)
#define DBG(X) cerr << #X << " = " << X << endl;
typedef vector<int> vint;
typedef vector<vint> vvint;

void showmt();
```

```

/* begin notebook */

#define MAXN 256
#define INFTO 0x7f7f7f7f
int n;
int mt[MAXN][MAXN]; // Matriz de costos (X * Y)
int xy[MAXN], yx[MAXN]; // Matching resultante (X->Y, Y->X)

int lx[MAXN], ly[MAXN], slk[MAXN], slkx[MAXN], prv[MAXN];
char S[MAXN], T[MAXN];

void updtree(int x) {
  forn(y, n) if (lx[x] + ly[y] - mt[x][y] < slk[y]) {
    slk[y] = lx[x] + ly[y] - mt[x][y];
    slkx[y] = x;
  }
}
int hungar() {
  forn(i, n) {
    ly[i] = 0;
    lx[i] = *max_element(mt[i], mt[i]+n);
  }
  memset(xy, -1, sizeof(xy));
  memset(yx, -1, sizeof(yx));

  forn(m, n) {
    memset(S, 0, sizeof(S));
    memset(T, 0, sizeof(T));
    memset(prv, -1, sizeof(prv));
    memset(slk, 0x7f, sizeof(slk));
    queue<int> q;
    #define bpone(e, p) { q.push(e); prv[e] = p; S[e] = 1; updtree(e); }
    forn(i, n) if (xy[i] == -1) { bpone(i, -2); break; }

    int x=0, y=-1;
    while (y==-1) {
      while (!q.empty() && y==-1) {
        x = q.front(); q.pop();
        forn(j, n) if (mt[x][j] == lx[x] + ly[j] && !T[j]) {
          if (yx[j] == -1) { y = j; break; }
          T[j] = 1;
          bpone(yx[j], x);
        }
      }
      if (y!=-1) break;
      int dlt = INFTO;
      forn(j, n) if (!T[j]) dlt = min(dlt, slk[j]);
      forn(k, n) {
        if (S[k]) lx[k] -= dlt;
        if (T[k]) ly[k] += dlt;
        if (!T[k]) slk[k] -= dlt;
      }
//      q = queue<int>();
      forn(j, n) if (!T[j] && !slk[j]) {
        if (yx[j] == -1) {
          x = slkx[j]; y = j; break;
        } else {
          T[j] = 1;
          if (!S[yx[j]]) bpone(yx[j], slkx[j]);
        }
      }
    }
    if (y!=-1) {
      for(int p = x; p != -2; p = prv[p]) {
        yx[y] = p;
        int ty = xy[p]; xy[p] = y; y = ty;
      }
    } else break;
  }
  int res = 0;
  forn(i, n) res += mt[i][xy[i]];
  return res;
}
```

8.3 Min-cost Max-Flow

```
/**
 *  If costs can be negative, call setpi before maxflow, but note that
     negative cost cycles are not supported.
 *  To obtain the actual flow, look at positive values only
 * Time: $O(F E \log(V))$ where F is max flow. $O(VE)$ for setpi.
 */
#include <bits/stdc++.h>
using namespace std;

```

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

#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

#pragma once

// #include <bits/extc++.h> /// include-line, keep-include

const ll INF = numeric_limits<ll>::max() / 4;

struct MCMF {
  struct edge {
    int from, to, rev;
    ll cap, cost, flow;
  };
  int N;
  vector<vector<edge>> ed;
  vi seen;
  vector<ll> dist, pi;
  vector<edge*> par;

  MCMF(int N) : N(N), ed(N), seen(N), dist(N), pi(N), par(N) {}

  void addEdge(int from, int to, ll cap, ll cost) {
    if (from == to) return;
    ed[from].push_back(edge{ from,to,sz(ed[to]),cap,cost,0 });
    ed[to].push_back(edge{ to,from,sz(ed[from])-1,0,-cost,0 });
  }

  void path(int s) {
    fill(all(seen), 0);
    fill(all(dist), INF);
    dist[s] = 0; ll di;

    __gnu_pbds::priority_queue<pair<ll, int>> q;
    vector<decltype(q)::point_iterator> its(N);
    q.push({ 0, s });

    while (!q.empty()) {
      s = q.top().second; q.pop();
      seen[s] = 1; di = dist[s] + pi[s];
      for (edge& e : ed[s]) if (!seen[e.to]) {
        ll val = di - pi[e.to] + e.cost;
        if (e.cap - e.flow > 0 && val < dist[e.to]) {
          dist[e.to] = val;
          par[e.to] = &e;
          if (its[e.to] == q.end())
            its[e.to] = q.push({ -dist[e.to], e.to });
          else
            q.modify(its[e.to], { -dist[e.to], e.to });
        }
      }
    }
    rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
  }

  pair<ll, ll> maxflow(int s, int t) {
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) {
      ll fl = INF;
      for (edge* x = par[t]; x; x = par[x->from])
        fl = min(fl, x->cap - x->flow);

      totflow += fl;
      for (edge* x = par[t]; x; x = par[x->from]) {
        x->flow += fl;
        ed[x->to][x->rev].flow -= fl;
      }
    }
    rep(i,0,N) for(edge& e : ed[i]) totcost += e.cost * e.flow;
    return {totflow, totcost/2};
  }

  // If some costs can be negative, call this before maxflow:
  void setpi(int s) { // (otherwise, leave this out)
    fill(all(pi), INF); pi[s] = 0;
    int it = N, ch = 1; ll v;
    while (ch-- && it--)
      rep(i,0,N) if (pi[i] != INF)
        for (edge& e : ed[i]) if (e.cap)
```

```
          if ((v = pi[i] + e.cost) < pi[e.to])
            pi[e.to] = v, ch = 1;
    assert(it >= 0); // negative cost cycle
  }
};
```

9 Strings

9.1 KMP-Prefix Function

```
// Maximum length of substring that ends at position i and is prefix of
    string
vector<int> KMP(string s){
  int n=s.length();
  vector<int> prefixFunction(n, 0);
  for(int i=1;i<n;i++){
    int j=prefixFunction[i-1];
    while(j>0 && s[i]!=s[j]){
      j=prefixFunction[j-1];
    }
    if(s[i]==s[j]){
      prefixFunction[i]=j+1;
    }
  }
  return prefixFunction;
}
```

9.2 Z-Function

```
// Maximum length of substring that begins at position i and is prefix
    of stringS
vector<int> z_function(string s) {
  int n = s.size();
  vector<int> z(n);
  int l = 0, r = 0;
  for(int i = 1; i < n; i++) {
    if(i < r) {
      z[i] = min(r - i, z[i - l]);
    }
    while(i + z[i] < n && s[z[i]] == s[i + z[i]]) {
      z[i]++;
    }
    if(i + z[i] > r) {
      l = i;
      r = i + z[i];
    }
  }
  return z;
}
```

9.3 Manacher

```
// finds length of palindromes centered at i
vector<int> manacher_odd(string s) {
    int n = s.size();
    s = "$" + s + "^";
    vector<int> p(n + 2);
    int l = 1, r = 1;
    for(int i = 1; i <= n; i++) {
        p[i] = max(0, 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);
}
```

9.4 String Hashing

```
const ll N = 2e5+1;
const ll MOD = 212345678987654321LL;
const ll base = 33;

// double hash or big mod values

class HashedString {
  private:
  // change M and B if you want
  static const long long M = 1e9 + 9;
  static const long long B = 9973;

  // pow[i] contains B^i % M
  static vector<long long> pow;

```

```
  // p_hash[i] is the hash of the first i characters of the given string
  vector<long long> p_hash;

  public:
  HashedString(const string &s) : p_hash(s.size() + 1) {
    while (pow.size() < s.size()) { pow.push_back((pow.back() * B) % M);
         }

    p_hash[0] = 0;
    for (int i = 0; i < s.size(); i++) {
      p_hash[i + 1] = ((p_hash[i] * B) % M + s[i]) % M;
    }
  }

  long long get_hash(int start, int end) {
    long long raw_val =
        (p_hash[end + 1] - (p_hash[start] * pow[end - start + 1]));
    return (raw_val % M + M) % M;
  }
};
vector<long long> HashedString::pow = {1};
```

9.5 MinCyclicRotation

```
string least_rotation(string s)
{
  s += s;
  vector<int> f(s.size(), -1);
  int k = 0;
  for(int j = 1; j < s.size(); j++)
  {
    char sj = s[j];
    int i = f[j - k - 1];
    while(i != -1 && sj != s[k + i + 1])
    {
      if(sj < s[k + i + 1]){
        k = j - i - 1;
      }
      i = f[i];
    }
    if(sj != s[k + i + 1])
    {
      if(sj < s[k]){
        k = j;
      }
      f[j - k] = -1;
    }
    else
      f[j - k] = i + 1;
  }
  return s.substr(k, s.size() / 2);
}
```

9.6 Suffix Array

```
//beware of first element of p being $
vector<int> suffixArray(string s){
  int n, k=0;
  s+="$";
  n=s.size();
  vector <int> p(n), c(n);
  vector <pair<char, int>> a(n);
  for (int i=0;i<n;i++)a[i]= {s[i], i};
  sort(a.begin(), a.end());
  for (int i=0;i<n;i++) p[i]=a[i].second;
  c[p[0]]=0;
  for (int i=1;i<n;i++){
    if(a[i].first!=a[i-1].first)  c[p[i]]=c[p[i-1]]+1;
    else c[p[i]]=c[p[i-1]];
  }
  while((1<<k)<n){
    for(int i=0;i<n;i++){
      p[i]=(p[i]-(1<<k)+n)%n;
    }
    vector<int> cnt(n);
    for(auto x:c){
      cnt[x]++;
    }
    vector<int> p1(n), pos(n);
    pos[0]=0;
    for(int i=1;i<n;i++){
      pos[i]=pos[i-1]+cnt[i-1];
    }
    for(auto x:p){
      int i=c[x];
      p1[pos[i]]=x;
```

```
      pos[i]++;
    }
    p=p1;
    vector<int> c1(n);
    c1[p[0]]=0;
    for(int i=1;i<n;i++){
      pair<int, int> aa={c[p[i]], c[(p[i]+(1<<k))%n]};
      pair<int, int> bb={c[p[i-1]], c[(p[i-1]+(1<<k))%n]};
      if(bb==aa){
        c1[p[i]]=c1[p[i-1]];
      }
      else{
        c1[p[i]]=c1[p[i-1]]+1;
      }
    }
    c=c1;
    k++;
  }
  return p;
}

vector<int> lcp_construction(string const& s, vector<int> const& p) {
    int n = s.size();
    vector<int> rank(n, 0);
    for (int i = 0; i < n; i++)
        rank[p[i]] = i;

    int k = 0;
    vector<int> lcp(n-1, 0);
    for (int i = 0; i < n; i++) {
        if (rank[i] == n - 1) {
            k = 0;
            continue;
        }
        int j = p[rank[i] + 1];
        while (i + k < n && j + k < n && s[i+k] == s[j+k])
            k++;
        lcp[rank[i]] = k;
        if (k)
            k--;
    }
    return lcp;
}
```

9.7 Trie AhoCorasick

```
const int K = 26;

struct Vertex {
    int next[K];
    bool output = false;
    int p = -1;
    char pch;
    int link = -1;
    int go[K];

    Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};

vector<Vertex> t(1);

void add_string(string const& s) {
    int v = 0;
    for (char ch : s) {
        int c = ch - 'a';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
    }
    t[v].output = true;
}

int go(int v, char ch);

int get_link(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0)
            t[v].link = 0;
        else
            t[v].link = go(get_link(t[v].p), t[v].pch);
    }
    return t[v].link;
```

```
}

int go(int v, char ch) {
    int c = ch - 'a';
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1)
            t[v].go[c] = t[v].next[c];
        else
            t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
    }
    return t[v].go[c];
}
```

9.8 SuffixAutomaton

```
struct state {
  int len, link, firstposition;
  map<char, int> next;
};

const int MAXN = 100000;
state st[MAXN * 2];
ll cnt[MAXN*2], cntPaths[MAXN*2], cntSum[MAXN*2];
int sz, last;

void initSuffixAutomaton() {
  st[0].len = 0;
  st[0].link = -1;
  sz++;
  last = 0;
}

void insertChar(char c) {
  int cur = sz++;
  st[cur].len = st[last].len + 1;
  st[cur].firstposition=st[last].len;
  int p = last;
  while (p != -1 && !st[p].next.count(c)) {
      st[p].next[c] = cur;
      p = st[p].link;
  }
  if (p == -1) {
      st[cur].link = 0;
  } else {
      int q = st[p].next[c];
      if (st[p].len + 1 == st[q].len) {
          st[cur].link = q;
      } else {
          int clone = sz++;
          st[clone].len = st[p].len + 1;
          st[clone].next = st[q].next;
          st[clone].link = st[q].link;
          st[clone].firstposition=st[q].firstposition;
          while (p != -1 && st[p].next[c] == q) {
              st[p].next[c] = clone;
              p = st[p].link;
          }
          st[q].link = st[cur].link = clone;
      }
  }
  last = cur;
  cnt[last]=1;
}

int search(string s){
  int cur=0, i=0, n=(int)s.length();
  while(i<n){
    if(!st[cur].next.count(s[i])) return -1;
    cur=st[cur].next[s[i]];
    i++;
  }
  //sumar 2 si se quiere 1 indexado
  return st[cur].firstposition-n+1;
}

void dfs(int cur){
  cntPaths[cur]=1;
  for(auto [x, y]:st[cur].next){
    if(cntPaths[y]==0) dfs(y);
    cntPaths[cur]+=cntPaths[y];
  }
}

void countPaths(){
  dfs(0);
}
```

```

string kthSmallestDistinct(ll k){
  string s="";
  int cur=0;
  while(k>0){
    for(auto [c, y]:st[cur].next){
      if(k>cntPaths[y]) k-=cntPaths[y];
      else{
        k--;
        s+=c;
        cur=y;
        break;
      }
    }
  }
  return s;
}

void countOcurrences(){
  vector<pair<int, int>> a;
  for(int i=sz-1;i>0;i--){
    a.push_back({st[i].len, i});
  }
  sort(a.begin(), a.end());
  for(int i=sz-2;i>=0;i--){
    cnt[st[a[i].second].link]+=cnt[a[i].second];
  }
}

void dfs1(int cur){
  for(auto [x, y]:st[cur].next){
    if(cntSum[y]==cnt[y]) dfs1(y);
    cntSum[cur]+=cntSum[y];
  }
}

void countSumOcurrences(){
  for(int i=0;i<sz;i++){
    cntSum[i]=cnt[i];
  }
  dfs1(0);
}

string kthSmallest(ll k){
  string s="";
  int cur=0;
  while(k>0){
    for(auto [c, y]:st[cur].next){
      if(k>cntSum[y]) k-=cntSum[y];
      else{
        k-=cnt[y];
        s+=c;
        cur=y;
        break;
      }
    }
  }
  return s;
}

//longest common substring
//build automaton for s first
string lcs (string S, string T) {
    int v = 0, l = 0, best = 0, bestpos = 0;
    for (int i = 0; i < T.size(); i++) {
        while (v && !st[v].next.count(T[i])) {
            v = st[v].link ;
            l = st[v].len;
        }
        if (st[v].next.count(T[i])) {
            v = st [v].next[T[i]];
            l++;
        }
        if (l > best) {
            best = l;
            bestpos = i;
        }
    }
    return T.substr(bestpos - best + 1, best);
}

int main(){
  ios_base::sync_with_stdio(false); cin.tie(NULL);
  string s; cin >> s;
  initSuffixAutomaton();
  for(char c:s){
```

```
158     insertChar(c);
159   }
160 }
```

10 DP

10.1 Digit DP

```
1  vector<int> num;
2  ll DP[20][20][2][2];
3
4  ll g(int pos, int last, int f, int z){
5
6      if(pos == num.size()){
7          return 1;
8      }
9
10     if(DP[pos][last][f][z] != -1) return DP[pos][last][f][z];
11     ll res = 0;
12
13     int l=(f ? 9 :num[pos]);
14
15     for(int dgt = 0; dgt<=l; dgt++){
16         if(dgt==last && !(dgt==0 && z==1)) continue;
17         int nf = f;
18         if(f == 0 && dgt < l) nf = 1;
19         if(z && !dgt) res+=g(pos+1, dgt, nf, 1);
20         else res += g(pos+1, dgt, nf, 0);
21     }
22     DP[pos][last][f][z]=res;
23     return res;
24 }
25
26 ll solve(ll x){
27     num.clear();
28     if(x==-1) return 0;
29     memset(DP, -1, sizeof(DP));
30     while(x>0){
31         num.pb(x%10);
32         x/=10;
33     }
34     reverse(all(num));
35     return g(0, 0, 0, 1);
36 }
```

10.2 Convex Hull Trick Deque

```
1  // needs fixing
2
3  struct line {
4    ll a, b;
5    line(ll A, ll B) : a(A), b(B) {}
6    double intersect(const line &line1) const {
7        return 1.0 * (line1.b - b) / (a - line1.a);
8    }
9    ll eval(ll x) {
10       return a * x + b;
11   }
12 };
13
14 // this finds the minimum and slope in increasing
15 deque<line> l[p+1];
16 l[0].push_front({-1, -s[1]});
17 for(int i=1;i<=m;i++){
18   for(int j=p;j>0;j--){
19     if(j>i) continue;
20     while((int)l[j-1].size()>=2 && l[j-1].back().eval(a[i])>=l[j-1][(int
           )l[j-1].size()-2].eval(a[i])){
21 l[j-1].pop_back();
22     }
23     dp[i][j]=l[j-1].back().eval(a[i])+(a[i]*(i))+s[i];
24     line cur(-i-1, dp[i][j]-s[i+1]);
25     while((int)l[j].size()>=2 && cur.intersect(l[j][1])<=l[j][0].
           intersect(l[j][1])){
26 l[j].pop_front();
27     }
28     l[j].push_front(cur);
29   }
30 }
```

10.3 Longest Common Subsequence(LCS)

```
1  string s, t; cin >> s >> t;
2  int n=s.length(), m=t.length();
3  int dp[n+1][m+1];
4  memset(dp, 0, sizeof(dp));
5  for(int i=1;i<=n;i++){
```

```
    for(int j=1;j<=m;j++){
        dp[i][j]=max(dp[i-1][j], dp[i][j-1]);
        if(s[i-1]==t[j-1]){
            dp[i][j]=dp[i-1][j-1]+1;
        }
    }
}
int i=n, j=m;
string ans="";
while(i && j){
    if(s[i-1]==t[j-1]){
        ans+=s[i-1];
        i--; j--;
    }
    else if(dp[i][j-1]>=dp[i-1][j]){
        j--;
    }
    else{
        i--;
    }
}
reverse(all(ans));
cout << ans << endl;
```

10.4 Edit Distance

```
string s, t; cin >> s>> t;
    int n=s.length(), m=t.length();
    for (int i=0;i<=n;i++){
        fill(dp[i], dp[i]+m+1, 1e9);
    }
    dp[0][0]=0;
    for (int i=0;i<=n;i++){
        for (int j=0;j<=m;j++){
            if(j){
                dp[i][j]=min(dp[i][j], dp[i][j-1]+1);
            }
            if(i){
                dp[i][j]=min(dp[i][j], dp[i-1][j]+1);
            }
            if(i && j){
                int a=(s[i-1]!=t[j-1] ? 1:0);
                dp[i][j]=min(dp[i][j], dp[i-1][j-1]+a);
            }
        }
    }
```

10.5 Longest Increasing Subsequence(LIS)

```
vector <int> dp;
for (int i=0;i<n;i++){
    auto it=lower_bound(dp.begin(), dp.end(), v[i]);
    if(it==dp.end()){
        dp.push_back(v[i]);
    }
    else{
        int pos=it-dp.begin();
        dp[pos]=v[i];
    }
}
cout << dp.size() << endl;
```

11 Geometry

11.1 Geometry Primitives

```
struct point2d {
    ftype x, y;
    point2d() {}
    point2d(ftype x, ftype y): x(x), y(y) {}
    point2d& operator+=(const point2d &t) {
        x += t.x;
        y += t.y;
        return *this;
    }
    point2d& operator-=(const point2d &t) {
        x -= t.x;
        y -= t.y;
        return *this;
    }
    point2d& operator*=(ftype t) {
        x *= t;
        y *= t;
        return *this;
    }
    point2d& operator/=(ftype t) {
```

```
        x /= t;
        y /= t;
        return *this;
    }
    point2d operator+(const point2d &t) const {return point2d(*this) += t;}
    point2d operator-(const point2d &t) const {return point2d(*this) -= t;}
    point2d operator*(ftype t) const {return point2d(*this) *= t;}
    point2d operator/(ftype t) const {return point2d(*this) /= t;}
    ftype dot(point2d a, point2d b) {return a.x * b.x + a.y * b.y;}
    ftype norm(point2d a) {return dot(a, a);}
    double abs(point2d a) {return sqrt(norm(a));}
    double proj(point2d a, point2d b) {return dot(a, b) / abs(b)}
    double angle(point2d a, point2d b) {return acos(dot(a, b) / abs(a) / abs(b));}

    //intersect between two lines
    point2d intersect(point2d a1, point2d d1, point2d a2, point2d d2) {
        return a1 + cross(a2 - a1, d2) / cross(d1, d2) * d1;
    }
};
point2d operator*(ftype a, point2d b) { return b * a;}

struct point3d {
    ftype x, y, z;
    point3d() {}
    point3d(ftype x, ftype y, ftype z): x(x), y(y), z(z) {}
    point3d& operator+=(const point3d &t) {
        x += t.x;
        y += t.y;
        z += t.z;
        return *this;
    }
    point3d& operator-=(const point3d &t) {
        x -= t.x;
        y -= t.y;
        z -= t.z;
        return *this;
    }
    point3d& operator*=(ftype t) {
        x *= t;
        y *= t;
        z *= t;
        return *this;
    }
    point3d& operator/=(ftype t) {
        x /= t;
        y /= t;
        z /= t;
        return *this;
    }
    point3d operator+(const point3d &t) const { return point3d(*this) += t;}
    point3d operator-(const point3d &t) const {return point3d(*this) -= t;}
    point3d operator*(ftype t) const {return point3d(*this) *= t;}
    point3d operator/(ftype t) const {return point3d(*this) /= t;}
    ftype dot(point3d a, point3d b) {return a.x * b.x + a.y * b.y + a.z * b.z;}

    ftype norm(point3d a) { return dot(a, a);}
    double abs(point3d a) {return sqrt(norm(a));}
    double proj(point3d a, point3d b) {return dot(a, b) / abs(b);}
    double angle(point3d a, point3d b) {return acos(dot(a, b) / abs(a) / abs(b));}

    //cross product
    point3d cross(point3d a, point3d b) {
        return point3d(a.y * b.z - a.z * b.y,
                       a.z * b.x - a.x * b.z,
                       a.x * b.y - a.y * b.x);
    }
    //intersect between three planes
    point3d intersect(point3d a1, point3d n1, point3d a2, point3d n2,
        point3d a3, point3d n3) {
        point3d x(n1.x, n2.x, n3.x);
        point3d y(n1.y, n2.y, n3.y);
        point3d z(n1.z, n2.z, n3.z);
        point3d d(dot(a1, n1), dot(a2, n2), dot(a3, n3));
        return point3d(triple(d, y, z),
                       triple(x, d, z),
                       triple(x, y, d)) / triple(n1, n2, n3);
    }

};
```

```
point3d operator*(ftype a, point3d b) {return b * a; }
```

11.2 Area of simple Polygon

```
// sum ((px-qx)*(py+qy))/2
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;
}
```

11.3 Point Inside Convex Polygon

```
void normalize(vector<pto> &pt){//delete collinear points first!
  //this makes it clockwise:
    if(pt[2].left(pt[0], pt[1])) reverse(pt.begin(), pt.end());
  int n=sz(pt), pi=0;
  forn(i, n)
    if(pt[i].x<pt[pi].x || (pt[i].x==pt[pi].x && pt[i].y<pt[pi].y))
      pi=i;
  vector<pto> shift(n);//puts pi as first point
    forn(i, n) shift[i]=pt[(pi+i)%n];
    pt.swap(shift);
}
bool inPolygon(pto p, const vector<pto> &pt){
  //call normalize first!
  if(p.left(pt[0], pt[1]) || p.left(pt[sz(pt)-1], pt[0])) return false;
  int a=1, b=sz(pt)-1;
  while(b-a>1){
    int c=(a+b)/2;
    if(!p.left(pt[0], pt[c])) a=c;
    else b=c;
  }
  return !p.left(pt[a], pt[a+1]);
}
```

11.4 Nearest Pair Of Points

```
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];
        }
    }
}

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

11.5 ConvexHull

```
struct pt {
    double x, y;
    bool operator == (pt const& t) const {
        return x == t.x && y == t.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 collinear(pt a, pt b, pt c) { return orientation(a, b, c) == 0; }

void convex_hull(vector<pt>& a, bool include_collinear = false) {
    pt p0 = *min_element(a.begin(), a.end(), [](pt a, pt b) {
        return make_pair(a.y, a.x) < make_pair(b.y, b.x);
    });
    sort(a.begin(), a.end(), [&p0](const pt& a, const pt& b) {
        int o = orientation(p0, a, b);
        if (o == 0)
            return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
                < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
        return o < 0;
    });
    if (include_collinear) {
        int i = (int)a.size()-1;
        while (i >= 0 && collinear(p0, a[i], a.back())) i--;
        reverse(a.begin()+i+1, a.end());
    }

    vector<pt> st;
    for (int i = 0; i < (int)a.size(); i++) {
        while (st.size() > 1 && !cw(st[st.size()-2], st.back(), a[i],
            include_collinear))
            st.pop_back();
        st.push_back(a[i]);
    }

    if (include_collinear == false && st.size() == 2 && st[0] == st[1])
        st.pop_back();

    a = st;
}
```

11.6 Rotating Calipers

```
#define ll long long
#define pii array<int,2>
#define nx(i) (i+1)%n
#define pv(i) (i-1+n)%n

struct Point {
    ll x,y;
```

```
    Point operator+(const Point &p) {
        return {x + p.x, y + p.y};
    }

    Point operator-(const Point &p) {
        return {x - p.x, y - p.y};
    }
};

ll cross(Point p1, Point p2) {
    return p1.x * p2.y - p1.y * p2.x;
}

int sign(ll num) {
    if (num < 0) return -1;
    else if (num == 0) return 0;
    else return 1;
}


vector<pii> all_anti_podal(int n, vector<Point> &p) {
    int p1 = 0, p2 = 0; // two "pointers"
    vector<pii> result;

    // parallel edges should't be visited twice
    vector<bool> vis(n, false);

    for (;p1<n;p1++) {
        // the edge that we are going to consider in this iteration
        // the datatype is Point, but it acts as a vector
        Point base = p[nx(p1)] - p[p1];

        // the last condition makes sure that the cross products don't
            have the same sign
        while (p2 == p1 || p2 == nx(p1) || sign(cross(base, p[nx(p2)] -
            p[p2])) == sign(cross(base, p[p2] - p[pv(p2)]))) {
            p2 = nx(p2);
        }

        if (vis[p1]) continue;
        vis[p1] = true;

        result.push_back({p1, p2});
        result.push_back({nx(p1), p2});

        // if both edges from p1 and p2 are parallel to each other
        if (cross(base, p[nx(p2)] - p[p2]) == 0) {
            result.push_back({p1, nx(p2)});
            result.push_back({nx(p1), nx(p2)});
            vis[p2] = true;
        }
    }

    return result;
}
```

12 Miscellaneous

12.1 Custom Comparators

```
bool cmp(const Edge &x, const Edge &y) { return x.w < y.w; }

sort(a.begin(), a.end(), cmp);

set<int, greater<int>> a;
map<int, string, greater<int>> b;
priority_queue<int, vector<int>, greater<int>> c;
```

12.2 Random Number Generator

```
//to avoid hacks
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
//you can also just write seed_value if hacks are not an issue

// rng() for generating random numbers between 0 and 2<<31-1

// for generating numbers with uniform probability in range
uniform_int_distribution<int>(0, n)(rng)
std::normal_distribution<> normal_dist(mean, 2)
exponential_distribution


// for shuffling array
shuffle(permutation.begin(), permutation.end(), rng);
```

12.3 Int Ternary Search

```
int lo = -1, hi = n;
while (hi - lo > 1){
    int mid = (hi + lo)>>1;
    if (f(mid) > f(mid + 1))
         hi = mid;
    else
         lo = mid;
}
//lo + 1 is the answer
```

12.4 Ternary Search

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

12.5 Parallel Binary Search

```
int lo[maxn], hi[maxn];
vector<int> tocheck[maxn];

bool c=true;
while(c){
    c=false;
    //initialize changes of structure to 0

    for(int i=0;i<k;i++){
        if(low[i]!=high[i]){
            check[(low[i]+high[i])/2].pb(i);
        }
    }

    for(int i=0;i<m;i++){
        // apply change for ith query

        while(check[i].size()){
            c=true;
            int x=check[i].back();
            check[i].pop_back();

            if(operationToCheck){
                high[x]=i;
            }
            else{
                low[x]=i+1;
            }
        }
    }
}
```

12.6 Next and Prev Smaller/Greater

```
vector<int> nextSmaller(vector<int> a, int n){
  stack<int> s;
  vector<int> res(n, n);
  for(int i=0;i<n;i++){
    while(s.size() && a[s.top()]>a[i]){
      res[s.top()]=i;
      s.pop();
    }
    s.push(i);
  }
  return res;
}

vector<int> prevSmaller(vector<int> a, int n){
  stack<int> s;
  vector<int> res(n, -1);
  for(int i=n-1;i>=0;i--){
    while(s.size() && a[s.top()]>a[i]){
      res[s.top()]=i;
      s.pop();
    }
```

```
    s.push(i);
  }
  return res;
}
```

12.7 2D Prefix Sum

```
int b[MAXN][MAXN];
int a[MAXN][MAXN];

for (int i = 1; i <= N; i++) {
  for (int j = 1; j <= N; j++) {
    b[i][j] = a[i][j] + b[i - 1][j] +
                        b[i][j - 1] - b[i - 1][j - 1];
  }
}

for (int q = 0; q < Q; q++) {
  int from_row, to_row, from_col, to_col;
  cin >> from_row >> from_col >> to_row >> to_col;
  cout << b[to_row][to_col] - b[from_row - 1][to_col] -
              b[to_row][from_col - 1] +
              b[from_row - 1][from_col - 1]
       << '\n';
}
```

12.8 Day of Week

```
int dayOfWeek(int d, int m, lli y){
  if(m == 1 || m == 2){
    m += 12;
    --y;
  }
  int k = y % 100;
  lli j = y / 100;
  return (d + 13*(m+1)/5 + k + k/4 + j/4 + 5*j) % 7;
}
```

12.9 Iterating over all subsets of mask

```
for (int mk = 0; mk < (1 << k); mk++) {
    Ap[mk] = 0;
    for (int s = mk;; s = (s - 1) & mk) {
        Ap[mk] += A[s];
        if (!s)
            break;
    }
}
```

12.10 Int 128

```
//cout for __int128
ostream &operator<<(ostream &os, const __int128 & value){
  char buffer[64];
  char *pos = end(buffer) - 1;
  *pos = '\0';
  __int128 tmp = value < 0 ? -value : value;
  do{
    --pos;
    *pos = tmp % 10 + '0';
    tmp /= 10;
  }while(tmp != 0);
  if(value < 0){
    --pos;
    *pos = '-';
  }
  return os << pos;
}

//cin for __int128
istream &operator>>(istream &is, __int128 & value){
  char buffer[64];
  is >> buffer;
  char *pos = begin(buffer);
  int sgn = 1;
  value = 0;
  if(*pos == '-'){
    sgn = -1;
    ++pos;
  }else if(*pos == '+'){
    ++pos;
  }
  while(*pos != '\0'){
    value = (value << 3) + (value << 1) + (*pos - '0');
    ++pos;
  }
  value *= sgn;
```

```
  return is;
}


ll mult(__int128 a, __int128 b){ return ((a*1LL*b)%MOD + MOD)%MOD; }
```

12.11 XOR Basis

```

int basis[d]; // basis[i] keeps the mask of the vector whose f value is
    i

int sz; // Current size of the basis

void insertVector(int mask) {
  //turn for around if u want max xor
  for (int i = 0; i < d; i++) {
    if ((mask & 1 << i) == 0) continue; // continue if i != f(mask)

    if (!basis[i]) { // If there is no basis vector with the i'th bit
        set, then insert this vector into the basis
      basis[i] = mask;
      ++sz;

      return;
    }

    mask ^= basis[i]; // Otherwise subtract the basis vector from this
        vector
  }
}
```

12.12 XOR Convolution

```
void FWHT (int A[], int k, int inv) {
    for (int j = 0; j < k; j++)
        for (int i = 0; i < (1 << k); i++)
            if (~i & (1 << j)) {
                int p0 = A[i];
                int p1 = A[i | (1 << j)];

                A[i] = p0 + p1;
                A[i | (1 << j)] = p0 - p1;

                if (inv) {
                    A[i] /= 2;
                    A[i | (1 << j)] /= 2;
                }
            }
}

void XOR_conv (int A[], int B[], int C[], int k) {
    FWHT(A, k, false);
    FWHT(B, k, false);

    for (int i = 0; i < (1 << k); i++)
        C[i] = A[i] * B[i];

    FWHT(A, k, true);
    FWHT(B, k, true);
    FWHT(C, k, true);
}
```

12.13 GCD Convolution

```
vector<int> PrimeEnumerate(int n) {
  vector<int> P; vector<bool> B(n + 1, 1);
  for (int i = 2; i <= n; i++) {
    if (B[i]) P.push_back(i);
    for (int j : P) { if (i * j > n) break; B[i * j] = 0; if (i % j ==
        0) break; }
  }
  return P;
}


template<typename T>
void MultipleZetaTransform(vector<T>& v) {
  const int n = (int)v.size() - 1;
  for (int p : PrimeEnumerate(n)) {
    for (int i = n / p; i; i--)
      v[i] += v[i * p];
  }
}

template<typename T>
void MultipleMobiusTransform(vector<T>& v) {
```

```
  const int n = (int)v.size() - 1;
  for (int p : PrimeEnumerate(n)) {
    for (int i = 1; i * p <= n; i++)
      v[i] -= v[i * p];
  }
}

template<typename T>
vector<T> GCDConvolution(vector<T> A, vector<T> B) {
    MultipleZetaTransform(A);
    MultipleZetaTransform(B);
    for (int i = 0; i < A.size(); i++) A[i] *= B[i];
    MultipleMobiusTransform(A);
    return A;
}
```

12.14 LCM Convolution

```
/* Linear Sieve, O(n) */
vector<int> PrimeEnumerate(int n) {
  vector<int> P; vector<bool> B(n + 1, 1);
  for (int i = 2; i <= n; i++) {
    if (B[i]) P.push_back(i);
    for (int j : P) { if (i * j > n) break; B[i * j] = 0; if (i % j ==
        0) break; }
  }
  return P;
}

template<typename T>
void DivisorZetaTransform(vector<T>& v) {
  const int n = (int)v.size() - 1;
  for (int p : PrimeEnumerate(n)) {
    for (int i = 1; i * p <= n; i++)
      v[i * p] += v[i];
  }
}

template<typename T>
void DivisorMobiusTransform(vector<T>& v) {
  const int n = (int)v.size() - 1;
  for (int p : PrimeEnumerate(n)) {
    for (int i = n / p; i; i--)
      v[i * p] -= v[i];
  }
}


template<typename T>
vector<T> LCMConvolution(vector<T> A, vector<T> B) {
    DivisorZetaTransform(A);
    DivisorZetaTransform(B);
    for (int i = 0; i < A.size(); i++) A[i] *= B[i];
    DivisorMobiusTransform(A);
    return A;
}
```

12.15 OR, AND Convolution

For OR convolution run subset SOS DP for arrays, multiply then inverse SOS DP. For AND convolution same idea but superset.

12.16 Mo's Algorithm

```
ll n, q;
ll cur=0;
ll cnt[1000005];
ll answers[200500];
ll BLOCK_SIZE;
ll arr[200500];

pair< pair<ll, ll>, ll> queries[200500];

inline bool cmp(const pair< pair<ll, ll>, ll> &x, const pair< pair<ll,
    ll>, ll> &y) {
    ll block_x = x.first.first / BLOCK_SIZE;
    ll block_y = y.first.first / BLOCK_SIZE;
    if(block_x != block_y)
        return block_x < block_y;
    return x.first.second < y.first.second;
}

int main(){
    cin >> n >> q;
    BLOCK_SIZE =(ll)(sqrt(n));
    for(int i = 0; i < n; i++)
```

```
        cin >> arr[i];

    for(int i = 0; i < q; i++) {
        cin >> queries[i].first.first >> queries[i].first.second;
        queries[i].second = i;
    }

    sort(queries, queries + q, cmp);

    ll l = 0, r = -1;

    for(int i = 0; i < q; i++) {
        ll left = queries[i].first.first;
        left--;
        ll right = queries[i].first.second;
        right--;

        while(r < right) {
            //operations
            r++;
        }
        while(r > right) {
            //operations
            r--;
        }

        while(l < left) {
            //operations
            l++;
        }
        while(l > left) {
            //operations
            l--;
        }
        answers[queries[i].second] = cur;
    }
}
```

12.17 Matrix Exponentiation

```
vector<vector<ll>> mul(vector<vector<ll>> a, vector<vector<ll>> b, int n
    ) {
  vector<vector<ll>> res(n, vector<ll>(n, 0));
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      for (int k = 0; k < n; k++) {
        res[i][j] += a[i][k] * b[k][j];
        res[i][j] %= MOD;
      }
    }
  }

  return res;
}

//for fibonacci
vector<vll> a(2, vll(2));
vector<vll> iden(2, vll(2));
a[0][0]=1;
a[0][1]=1;
a[1][0]=1;
a[1][1]=0;
iden[0][0]=1;
iden[1][1]=1;
iden[0][1]=0;
iden[1][0]=0;
for (;k>0;k/=2){
    if(k&1) iden=mul(iden, a);
    a=mul(a, a);
}
```

12.18 Sprague-Grundy Theorem

1) Get all transitions from state.
2) Calculate Grundy value for each independent game and xor them.
3) Grundy value of a state is mex of the Grundy values of transitions.
4) Zero means losing, anything else means winning state.

13 Stress Testing Scripts

13.1 build.sh

This file should be called before stress.sh or validate.sh. build.sh name.cpp

```
g++ -static -DLOCAL -lm -s -x c++ -Wall -Wextra -O2 -std=c++17 -o $1 $1.
    cpp
```

13.2 stress.sh

Format is stress.sh Awrong Aslow Agen Numtests

```
#!/usr/bin/env bash

for ((testNum=0;testNum<$4;testNum++))
do
    ./$3 > input
    ./$2 < input > outSlow
    ./$1 < input > outWrong
    H1=`md5sum outWrong`
    H2=`md5sum outSlow`
    if !(cmp -s "outWrong" "outSlow")
    then
        echo "Error found!"
        echo "Input:"
        cat input
        echo "Wrong Output:"
        cat outWrong
        echo "Slow Output:"
        cat outSlow
        exit
    fi
done
echo Passed $4 tests
```

13.3 validate.sh

Format is validate.sh Awrong Avalidator Agen NumTests

```
#!/usr/bin/env bash

for ((testNum=0;testNum<$4;testNum++))
do
    ./$3 > input
    ./$1 < input > out
    cat input out > data
    ./$2 < data > res
    result=$(cat res)
    if [ "${result:0:2}" != "OK" ];
    then
        echo "Error found!"
        echo "Input:"
        cat input
        echo "Output:"
        cat out
        echo "Validator Result:"
        cat res
        exit
    fi
done
echo Passed $4 tests
```

14 Useful things

14.1 Sums

$$c^a + c^{a+1} + \cdots + c^b = \frac{c^{b+1} - c^a}{c-1}, c \neq 1$$

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$
$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n+1)(n+1)}{6}$$
$$1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$
$$1^4 + 2^4 + 3^4 + \cdots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

14.2 Catalan numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1,\ C_{n+1} = \frac{2(2n+1)}{n+2}C_n,\ C_{n+1} = \sum C_i C_{n-i}$$

$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \ldots$

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with n pairs of parenthesis, correctly nested. If prefix is given, number of ways is $\binom{n}{remaining_closed}$-$\binom{n}{remaining_closed+1}$.
- binary trees with with $n+1$ leaves (0 or 2 children).
- ordered trees with $n+1$ vertices.

- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

14.3 Cayley's formula

Number of labeled trees of n vertices is n^{n-2}. Number of rooted forest of n vertices is $(n+1)^{n-1}$.

14.4 Geometric series

Infinite
$a + ar + ar^2 + ar^3 + ... + \sum_{k=0}^{\infty} ar^k$
Sum=$\frac{a}{1-r}$
Finite
$a + ar + ar^2 + ar^3 + ... + \sum_{k=0}^{n} ar^k$
Sum=$\frac{a(1-r^{n+1})}{1-r}$

14.5 Estimates For Divisors

$\sum_{d|n} d = O(n \log \log n)$.
The number of divisors of n is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 200 000 for $n < 1e19$.

14.6 Sum of divisors

$\sum d|n = \frac{{p_1}^{e_1+1}-1}{p_1-1} + \frac{{p_2}^{e_2+1}-1}{p_2-1} + ... + \frac{{p_n}^{e_n+1}-1}{p_n-1}$

14.7 Pythagorean Triplets

The Pythagorean triples are uniquely generated by

$$a = k \cdot (m^2 - n^2),\ \ b = k \cdot (2mn),\ \ c = k \cdot (m^2 + n^2),$$

with $m > n > 0$, $k > 0$, $m \perp n$, and either m or n even.

14.8 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1) + D(n-2)) = nD(n-1) + (-1)^n = \left\lfloor \frac{n!}{e} \right\rceil$$

15 C++ things

- **Compiling**
  g++ -std=c++20 name.cpp -o name
- **Bits library**
  #include <bits/stdc++.h>
- **Fast IO**
  ios_base::sync_with_stdio(false); cin.tie(NULL);
- **Compiler Optimizations**
  #pragma GCC target("popcnt"): makes bit operations faster
  #pragma GCC optimize(" Ofast"): auto-vectorize for loops and optimizes floating points better (assumes associativity and turns off denormals).
  #pragma GCC target(" avx2"): can double performance of vectorized code, but causes crashes on old machines.
  #pragma GCC optimize("O3,unroll-loops")
  #pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
- **Decimal Printing**
  cout < < fixed < < setprecision(n) < < a < < endl;
- **Bit Tricks**
  x & -x is least bit in x.
  c = x & -x, r = x+c; (((r^x) > > 2)/c) | r, next number bigger than x same number of bits set.