Contents

2d prefix sum

```
// construction
for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= N; j++) {
        prefix[i][j] =
            arr[i][j]
            + prefix[i - 1][j]
            + prefix[i][j - 1]
            - prefix[i - 1][j - 1];
    }
}

// query
pfx[to_row][to_col]
- pfx[from_row - 1][to_col]
- pfx[to_row][from_col - 1]
+ pfx[from_row - 1][from_col - 1];

// partial: add value v to subrectangle [from_row, from_col] to [to_row,
to_col]
diff[from_row][from_col]     += v;
diff[from_row][to_col + 1] -= v;
diff[to_row + 1][from_col] -= v;
diff[to_row + 1][to_col + 1] += v;
// then construct diff, grid[i][j] += diff[i][j];
```

BIT, Fenwick

```
template<class T>
struct BIT { // 1-based
    int n;
    vector<T> tree;
    explicit BIT(int size) : n(size), tree(size + 1) { }

    void add(int i, T val) {
        for (; i <= n; i += i & -i)
            tree[i] += val;
    }

    T query_point(int i) {
        if (!i) return query(0);
        return query(i) - query(i - 1);
    }
    void set(int i, T val) {
        int c = query_point(i);
        add(i, val - c);
    }

    T query(int i) {
        T sum = 0;
        for (; i > 0; i -= i & -i)
```

```
            sum += tree[i];
        return sum;
    }

    T range(int l, int r) {
        return query(r) - query(l - 1);
    }

    int lower_bound(T target) {
        int i = 0;
        T curr = 0;
        for (int mask = 1 << __lg(n); mask > 0; mask >>= 1) {
            if (i + mask <= n && curr + tree[i + mask] < target) {
                curr += tree[i += mask];
            }
        }
        return i + 1;
    }
};
```

BIT range

```
template<typename T>
class BitR { // 0-based
    int n;
    vector<T> f, s;
    void add(vector<T> &a, int i, T val) {
        for(; i < n; i += i & -i)
            a[i] += val;
    }
public:
    BitR(int n) : n(n + 5), f(n + 6), s(n + 6) { }

    void add(int i, T val) {
        add(s, i + 1, -val);
    }

    T query_point(int i) {
        if (!i) return query(0);
        return query(i) - query(i - 1);
    }
    void set(int i, T val) {
        int c = query_point(i);
        add(i, val - c);
    }

    void add(int l, int r, T val) {
        l++, r++;
        add(f, l, val);
        add(f, r + 1, -val);
        add(s, l, val * (l - 1));
        add(s, r + 1, -val * r);
    }

    T query(int ii) {
        ii++;
        T sum = 0;
        int i = ii;
        for(; i > 0; i ^= i & -i)
            sum += f[i];
        sum *= ii;
        i = ii;
        for(; i > 0; i ^= i & -i)
            sum -= s[i];
        return sum;
    }

    T range(int l, int r) {
        return query(r) - query(l - 1);
    }
};
```

Segment tree iterative

```
struct info {
    ll sum;
    info(ll x) {
        sum = x;
    }
    info() { // default value
        sum = 0;
    }
    friend info operator+(const info &l, const info &r) {
        info ret;
        ret.sum = l.sum + r.sum;
        return ret;
    }
};
template<class info>
class segmentTreeIterative {
    int size;
    vector<info> tree;
    static info defaultVal;
public:
    explicit segmentTreeIterative(int n) : size(n), tree(size << 1, defaultVal)
{ }
```

```
    template<class U>
    explicit segmentTreeIterative(const U &arr) : size(arr.size()), tree(size
<< 1, defaultVal) {
        for(int i = 0; i < arr.size(); i++)
            tree[i + size] = arr[i];
        for(int i = size - 1; i > 0; i--)
            tree[i] = tree[i << 1] + tree[i << 1 | 1];
    }
    void set(int i, info v) {
        tree[i += size] = v;
        for(i >>= 1; i; i >>= 1)
            tree[i] = tree[i << 1] + tree[i << 1 | 1];
    }
    info get(int l, int r) {
        info resL = defaultVal, resR = defaultVal;
        l += size, r += size + 1;
        while(l < r) {
            if(l & 1) resL = resL + tree[l++];
            if(r & 1) resR = tree[--r] + resR;
            l >>= 1, r >>= 1;
        }
        return resL + resR;
    }
};
```

Segment Tree

```
struct info {
    ll sum;
    info(ll x) {
        sum = x;
    }
    info() { // default value
        sum = 0;
    }
    friend info operator+(const info &l, const info &r) {
        info ret;
        ret.sum = l.sum + r.sum;
        return ret;
    }
};
template<typename info>
class segmentTree {
    struct node {
        node *l, *r;
        info v;
        explicit node() : l(nullptr), r(nullptr), v() { }
        void create() {
            if(!l) l = new node(), r = new node();
        }
    } *root;
    int size;
    static info defaultVal;

    template<class U>
    void build(node *x, int lx, int rx, U &arr) {
        if(lx == rx) return void(x->v = arr[lx]);
        x->create();
        int m = (lx + rx) >> 1;
        build(x->l, lx, m, arr);
        build(x->r, m + 1, rx, arr);
        x->v = x->l->v + x->r->v;
    }

    info get(node *x, int lx, int rx, int l, int r) {
        if(lx != rx) x->create();
        if(lx > r || l > rx) return defaultVal;
        if(lx >= l && rx <= r) return x->v;
        int m = (lx + rx) >> 1;
        return get(x->l, lx, m, l, r) + get(x->r, m + 1, rx, l, r);
    }

    void set(node *x, int lx, int rx, int i, info val) {
        if(lx != rx) x->create();
        if (i < lx || i > rx) return;
        if(lx == rx) return void(x->v = val);
        int m = (lx + rx) >> 1;
        set(x->l, lx, m, i, val), set(x->r, m + 1, rx, i, val);
        x->v = x->l->v + x->r->v;
    }

    void del(node *x) {
        if(x){
            del(x->l), del(x->r);
            delete x;
        }
    }
public:
    explicit segmentTree(int n = 1'000'000'000) : size(n), root(new node()) { }

    template<class U>
    explicit segmentTree(U &arr) : size(int(arr.size()) - 1), root(new node()) {
        build(root, 0, size, arr);
    }
    ~segmentTree(){
        del(root);
    }
    void set(int i, info v) {
        set(root, 0, size, i, v);
    }
    info get(int l, int r) {
        return get(root, 0, size, l, r);
    }
};
template<> info segmentTree<info>::defaultVal = info();
```

Sparse table

```
template<typename T>
struct sparse{
    int Log, n;
    vector<vector<T>> table;
    function<T(T, T)> merge;
    template<class U>
    explicit sparse(vector<T> arr, U merge) :
      merge(merge),
      n((int)arr.size()),
      Log(__lg(arr.size()) + 1),
      table(Log, vector<T>(n))

    {
        table[0] = arr;
        for(int l = 1; l < Log; l++) {
            for(int i = 0; i + (1 << (l - 1)) < n; i++) {
                table[l][i] = merge(table[l - 1][i], table[l - 1][i + (1 << (l - 1))]);
            }
        }
    }
    T query(int l, int r) {
        if(l > r) return {};
        int len = __lg(r - l + 1);
        return merge(table[len][l], table[len][r - (1 << len) + 1]);
    }
};
```

2d BIT, 2d Fenwick

```
template<typename T>
class BIT2D {
public:
    vector<vector<T>> tree;
    int n, m;

    void init(int n, int m) {
        tree.assign(n + 2, vector<T>(m + 2, 0));
        this->n = n;
        this->m = m;
    }

    T merge(T &x, T &y) { return x + y; }

    void update(int x, int y, T val) {
        for (; x <= n; x += x & -x) {
            for (int z = y; z <= m; z += z & -z) {
                tree[x][z] = merge(tree[x][z], val);
            }
        }
    }

    T getPrefix(int x,int y){
        if(x <= 0)return 0;
        T ret = 0;
        for (; x ; x-=x&-x) {
            for (int z = y; z ; z-=z&-z) {
                ret = merge(ret,tree[x][z]);
            }
        }
        return ret;
    }

    T getSquare(int xl,int yl,int xr,int yr){
        return getPrefix(xr,yr) + getPrefix(xl - 1,yl - 1) -
                getPrefix(xr,yl - 1) - getPrefix(xl - 1,yr);
    }
};
```

2d sparse table

```
template<typename T = int>
struct sparse2d {
    int Log, n, m;
    vector<vector<vector<T>>> table;
    function<T(T, T)> merge;

    template<class U>
    explicit sparse2d(const vector<vector<T>>& arr, U merge)
      : merge(merge),
        n((int)arr.size()),
        m((int)arr[0].size()),
        Log(__lg(max(n, m)) + 1)
    {
        table.resize(Log+1);
        for (int k = 0; k <= Log; ++k) {
            int H = n - (1<<k) + 1;
            int W = m - (1<<k) + 1;
            if (H > 0 && W > 0)
                table[k].assign(H, vector<T>(W));
        }

        for (int i = 0; i < n; ++i)
            for (int j = 0; j < m; ++j)
                table[0][i][j] = arr[i][j];

        for (int k = 1; k <= Log; ++k) {
            int H = n - (1<<k) + 1;
            int W = m - (1<<k) + 1;
            for (int i = 0; i < H; i++) {
                for (int j = 0; j < W; j++) {
                    T a = table[k-1][i][j];
                    T b = table[k-1][i + (1<<(k-1))][j];
                    T c = table[k-1][i][j + (1<<(k-1))];
                    T d = table[k-1][i + (1<<(k-1))][j + (1<<(k-1))];
                    table[k][i][j] = merge( merge(a, b), merge(c, d) );
                }
            }
        }
    }

    T query(int x1, int y1, int x2, int y2) {
        int h = x2 - x1 + 1;
        int w = y2 - y1 + 1;
        int k = __lg(min(h, w));
        T a = table[k][x1][y1];
        T b = table[k][x2 - (1<<k) + 1][y1];
        T c = table[k][x1][y2 - (1<<k) + 1];
        T d = table[k][x2 - (1<<k) + 1][y2 - (1<<k) + 1];
        return merge( merge(a, b), merge(c, d) );
    }
```

```
};
```

2d segment tree

```
struct info {
    ll sum;
    info(ll x) {
        sum = x;
    }
    info() { // default value
        sum = 0;
    }
    friend info operator+(const info &l, const info &r) {
        info ret;
        ret.sum = l.sum + r.sum;
        return ret;
    }
};
template<class info>
class segmentTree2d {
    int nx, ny;
    vector<vector<info>> tree;
    static info defaultVal;
  public:
    explicit segmentTree2d(int n, int m)
      : nx(n), ny(m),
        tree((nx<<1), vector<info>(ny<<1, defaultVal))
    {}

    template<class U>
    explicit segmentTree2d(const vector<vector<U>> &a) {
        nx = int(a.size());
        ny = nx? int(a[0].size()) : 0;
        tree.assign((nx<<1), vector<info>(ny<<1, defaultVal));
        for(int i = 0; i < nx; i++){
            for(int j = 0; j < ny; j++)
                tree[i+nx][j+ny] = info(a[i][j]);
            for(int y = ny-1; y > 0; y--)
                tree[i+nx][y] = tree[i+nx][y<<1] + tree[i+nx][y<<1|1];

        }

        for(int x = nx-1; x > 0; x--)
            for(int y = 0; y < (ny<<1); y++)
                tree[x][y] = tree[x<<1][y] + tree[x<<1|1][y];
    }
    void set(int i, int j, info v) {
        if(i<0||i>=nx||j<0||j>=ny) return;
        int x = i + nx, y = j + ny;
        tree[x][y] = v;
        for(int yy = y>>1; yy > 0; yy >>= 1)
            tree[x][yy] = tree[x][yy<<1] + tree[x][yy<<1|1];
        for(int xx = x>>1; xx > 0; xx >>= 1) {
            tree[xx][y] = tree[xx<<1][y] + tree[xx<<1|1][y];
            for(int yy = y>>1; yy > 0; yy >>= 1)
                tree[xx][yy] = tree[xx][yy<<1] + tree[xx][yy<<1|1];
        }
    }

    info get(int x1, int y1, int x2, int y2) {
        if(nx==0||ny==0) return defaultVal;
        x1 = max(x1, 0); y1 = max(y1, 0);
        x2 = min(x2, nx-1); y2 = min(y2, ny-1);
        if(x1 > x2 || y1 > y2) return defaultVal;

        info resL = defaultVal, resR = defaultVal;
        int l = x1 + nx, r = x2 + nx + 1;
        for(; l < r; l >>= 1, r >>= 1) {
            if(l & 1) resL = resL + queryY(l++, y1, y2);
            if(r & 1) resR = queryY(--r, y1, y2) + resR;
        }
        return resL + resR;
    }

  private:
    info queryY(int nodeX, int y1, int y2) const {
        info resL = defaultVal, resR = defaultVal;
        int l = y1 + ny, r = y2 + ny + 1;
        for(; l < r; l >>= 1, r >>= 1) {
            if(l & 1) resL = resL + tree[nodeX][l++];
            if(r & 1) resR = tree[nodeX][--r] + resR;
        }
        return resL + resR;
    }
};

template<class info> info segmentTree2d<info>::defaultVal = info();
```

Monotonic stack / queue

```
template<class T>
```

```
struct Mono_stack{
    stack<pair<T,T>>st;
    void push(const T& val){
        if(st.empty())
            st.emplace(val,val);
        else st.emplace(val,std::max(val,st.top().second));
    }
    void pop(){
        st.pop();
    }
    bool empty(){
        return st.empty();
    }
    int size(){
        return st.size();
    }
    T top(){
        return st.top().first;
    }
    T max(){
        return st.top().second;
    }
};
template<class T>
struct Mono_queue{
    Mono_stack<T>pop_st,push_st;
    void push(const T& val){
        push_st.push(val);
    }
    void move(){
        if(pop_st.size())
            return;
        while(!push_st.empty())
            pop_st.push(push_st.top()),push_st.pop();
    }
    void pop(){
        move();
        pop_st.pop();
    }
    bool empty(){
        return pop_st.empty()&&push_st.empty();
    }
    int size(){
        return pop_st.size()+push_st.size();
    }
    T top(){
        move();
        return pop_st.top();
    }
    T max(){
        if(pop_st.empty())
            return push_st.max();
        if(push_st.empty())
            return pop_st.max();
        return std::max(push_st.max(),pop_st.max());
    }
};
```

DSU, unionfind

```
struct DSU {
    vector<int> p, sz;
    int n, comps;
    DSU(int _n = 0) { init(_n); }
    void init(int _n) {
        n = _n + 10; comps = _n;
        p.resize(n); sz.assign(n, 1);
        iota(p.begin(), p.end(), 0);
    }
    int find(int u) { return u == p[u] ? u : p[u] = find(p[u]); }
    bool unite(int u, int v) {
        u = find(u), v = find(v);
        if (u == v) return 0;
        if (sz[u] < sz[v]) swap(u, v);
        p[v] = u; sz[u] += sz[v]; comps--;
        return 1;
    }
    bool same(int u, int v) { return find(u) == find(v); }
    int size(int u) { return sz[find(u)]; }
    int size() { return comps; }
};
```

DSU, unionfind,with rollback

```
class DSU {
  private:
     vector<int> p, sz;
     vector<pair<int &, int>> history;
  public:
     DSU(int n) : p(n), sz(n+10, 1) { iota(p.begin(), p.end(), 0); }
     int get(int x) { return x == p[x] ? x : get(p[x]); }
     void unite(int a, int b) {
          a = get(a);
          b = get(b);
          if (a == b) { return; }
          if (sz[a] < sz[b]) { swap(a, b); }
          history.push_back({sz[a], sz[a]});
          history.push_back({p[b], p[b]});
          p[b] = a;
          sz[a] += sz[b];
     }
     int snapshot() { return history.size(); }
     void rollback(int until = 1) {
          while (snapshot() > until) {
               history.back().first = history.back().second;
               history.pop_back();
          }
     }
};
```

Ordered data structures

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T> using ordered_set = tree<T, null_type, less<T>,
    rb_tree_tag, tree_order_statistics_node_update>;
template <typename T, typename R> using ordered_map = tree<T, R, less<T>,
    rb_tree_tag, tree_order_statistics_node_update>;
```

Rolling Hash

```
namespace RollingHash {
    const int N = _+_+_+_+_, b1 = 31, b2 = 69, mod = 1e9 + 7,
        b1I = 129032259, b2I = 579710149;

    vector<int> Pb1(N + 1), Pb2(N + 1);
    char init = []() {
        Pb1[0] = Pb2[0] = 1;
        for (int i = 1; i <= N; i++) {
            Pb1[i] = int(1LL * Pb1[i - 1] * b1 % mod);
            Pb2[i] = int(1LL * Pb2[i - 1] * b2 % mod);
        }
        return char();
    }();

    struct Hash : array<int, 3> {
#define a (*this)
        Hash() : array() { }
        Hash(const array<int, 3> &x) : array(x) { }
        Hash(int x) : array({x % mod, x % mod, 1}) { }

        Hash(const string &x) : array() {
            for(char c : x) *this = *(this) + c;
        }

        void pop_front(int x) {
            a[0] = int((a[0] - 1LL * Pb1[--a[2]] * x % mod + mod) % mod);
            a[1] = int((a[1] - 1LL * Pb2[a[2]] * x % mod + mod) % mod);
        }

        void pop_back(int x) {
            a[0] = int((1LL * (a[0] - x + mod) * b1I) % mod);
            a[1] = int((1LL * (a[1] - x + mod) * b2I) % mod);
            a[2]--;
        }

        void clear() { a[0] = a[1] = a[2] = 0; }

        friend Hash operator+(const Hash &f, const Hash &o) {
            return array{int((1LL * f[0] * Pb1[o[2]] + o[0]) % mod),
                         int((1LL * f[1] * Pb2[o[2]] + o[1]) % mod)
                    , f[2] + o[2]};
        }
#undef a
    };

    struct HashRange {
        vector<Hash> p, s;
        HashRange(const string &t) : p(t.size()), s(t.size()) {
            p.front() = t.front();
            for(int i = 1; i < t.size(); i++) p[i] = p[i - 1] + t[i];
            s.back() = t.back();
            for(int i = int(t.size()) - 2; i >= 0; i--) s[i] = s[i + 1] + t[i];
        }
        Hash get(int l, int r) {
            if(l > r) return {};
            if(!l) return p[r];
            return array{int((p[r][0] - p[l - 1][0] * 1LL * Pb1[r - l + 1] %
mod + mod) % mod),
                         int((p[r][1] - p[l - 1][1] * 1LL * Pb2[r - l + 1] %
mod + mod) % mod)
                    , r - l + 1};
        }
        Hash inv(int l, int r) {
            if(l > r) return {};
            if(r + 1 == s.size()) return s[l];
            return array{int((s[l][0] - s[r + 1][0] * 1LL * Pb1[r - l + 1] %
mod + mod) % mod),
                         int((s[l][1] - s[r + 1][1] * 1LL * Pb2[r - l + 1] %
mod + mod) % mod)
                    , r - l + 1};
        }
    };
}
//using namespace RollingHash;
```

Fast rolling hash

```
po[0] = 1;
for(int i = 1; i <= N; i++)
    po[i] = int(po[i - 1] * 1LL * b1 % mod);

const int N = 4000 + 1;
vector<int> po(N + 1);
int b1 = 37, mod = 1e9 + 7;
struct Hash {
    vector<int> h;
    explicit Hash(const string &s) : h(s.size()) {
        int h1 = 0;
        for(int i = 0; i < s.size(); i++) {
            h1 = int(((h1 * 1LL * b1 % mod) + s[i]) % mod);
            h[i] = h1;
        }
    }
    int get(int l, int r) {
        if(l == 0) return h[r];
        int res = (h[r] - h[l - 1] * 1LL * po[r - l + 1]) % mod;
        if(res < 0) res += mod;
        return res;
    }
};
```

mex calculator

```
template<class T>
class mex_calculator {
    map<T, T> count;
    set<T> missing;
    public:
    mex_calculator(const vector<T>& arr, T upper_bound) { // n log n
```

```
        for (T x : arr)
            count[x]++;
        for (T i = 0; i <= upper_bound + 1; ++i)
            if (count[i] == 0)
                missing.insert(i);
    }
    void insert(T x) { // log
        count[x]++;
        if (count[x] == 1)
            missing.erase(x);
    }
    void remove(T x) { // log
        if (count[x] == 0) return;
        count[x]--;
        if (count[x] == 0)
            missing.insert(x);
    }
    T get_mex() { // 1
        return *missing.begin();
    }
};
```

xor trie

```
template<typename T>
class BinaryTrie {
private:
    vector<array<int, 2>> nodes;
    vector<int> sz;
    int numberOfBits;
    T one = 1;

    bool isNodeAvailable(int cur, bool nxt) {
        return nodes[cur][nxt] != 0 && sz[nodes[cur][nxt]] > 0;
    }

    int getNode(T x) {
        int cur = 0;
        for (int i = numberOfBits - 1; i >= 0; --i) {
            bool nxt = (x >> i) & 1;
            if (nodes[cur][nxt] == 0) {
                nodes[cur][nxt] = nodes.size();
                nodes.push_back({0, 0});
                sz.push_back(0);
            }
            cur = nodes[cur][nxt];
        }
        return cur;
    }

public:
    BinaryTrie() : nodes(1, {0, 0}), sz(1, 0), numberOfBits(sizeof(T) * 8) {}

    void insert(T x, int cnt = 1) {
        int cur = 0;
        sz[cur] += cnt;
        for (int i = numberOfBits - 1; i >= 0; --i) {
            bool nxt = (x >> i) & 1;
            if (nodes[cur][nxt] == 0) {
                nodes[cur][nxt] = nodes.size();
                nodes.push_back({0, 0});
                sz.push_back(0);
            }
            cur = nodes[cur][nxt];
            sz[cur] += cnt;
        }
    }

    void erase(T x, int cnt = 1) {
        insert(x, -cnt);
    }

    T getMinXor(T x) {
        if (sz[0] == 0) throw out_of_range("The trie is empty");
        int cur = 0;
        T ret = 0;
        for (int i = numberOfBits - 1; i >= 0; --i) {
            bool nxt = (x >> i) & 1;
            if (isNodeAvailable(cur, nxt)) {
                cur = nodes[cur][nxt];
            } else {
                ret |= (one << i);
                cur = nodes[cur][!nxt];
            }
        }
        return ret;
    }

    T getMaxXor(T x) {
        if (sz[0] == 0) throw out_of_range("The trie is empty");
        int cur = 0;
        T ret = 0;
        for (int i = numberOfBits - 1; i >= 0; --i) {
            bool nxt = !((x >> i) & 1);
            if (isNodeAvailable(cur, nxt)) {
                ret |= (one << i);
                cur = nodes[cur][nxt];
            } else {
                cur = nodes[cur][!nxt];
            }
        }
        return ret;
    }

    size_t size() {
        return sz[0];
    }

    bool contains(T x) {
        return sz[0] && getMinXor(x) == 0;
    }

    int count(T x) {
        return sz[getNode(x)];
    }
};
```

xor basis

```
const int bits = __lg(100000) + 1;
struct basis {
    int sz = 0;
    array<int, bits> a{};
    void add(int x) {
        if(sz == bits) return;
        for(int i = __lg(x); x; x ^= a[i], i = __lg(x)) {
            if(!a[i]) return sz++, void(a[i] = x);
        }
    }
    bool find(int x) {
        if(sz == bits) return true;
        for(int i = __lg(x); x; i = __lg(x)) {
            if(a[i]) x ^= a[i];
            else return false;
        }
        return true;
    }
    void clear() {
        if(sz) a.fill(0), sz = 0;
    }
    int getMax() {
        int r = 0;
        for(int i = bits - 1; i >= 0; i--) r = max(r ^ a[i], r);
        return r;
    }
    int find_k(int k) { // index-0
        assert(k >= 0 && k < 1 << sz);
        int curr = 0;
        for(int i = bits - 1, b = sz - 1; i >= 0; i--) {
            if(a[i]) {
                if((k >> b & 1) ^ (curr >> i & 1)) curr ^= a[i];
                b--;
            }
        }
        return curr;
    }
    basis& operator+=(const basis &o) {
        if(sz == bits) return *this;
        if(o.sz == bits) return *this = o;
        for(int i = 0; i < bits; i++) if(o.a[i])
                add(o.a[i]);
        return *this;
    }
};
```

Binary lifting, LCA

```
int Log = __lg(n) + 1;
vector<vector<int>> lift(n+1, vector<int>(Log, -1));
function<void(int, int)> dfs2 = [&](int u, int p) {
    for (auto v : tree[u]) {
        if (v == p) continue;
        lift[v][0] = u;
        for (int l = 1; l < Log; l++) {
            if (lift[v][l-1] == -1)
                break;
            lift[v][l] = lift[ lift[v][l-1] ][l-1];
        }
        dfs2(v, u);
    }
};
dfs2(root, -1);

if (depth[u] < depth[v])
```

```
    swap(u, v);
int k = depth[u] - depth[v];

for (int l = Log-1; ~l && ~u; l--) {
    if (k >> l & 1)
        u = lift[u][l];
}
    if (u == v)
    return u;
for (int l = Log-1; l > -1; l--) {
    if (lift[u][l] != lift[v][l])
        u = lift[u][l], v = lift[v][l];
}
return lift[u][0];
```

Tree diameter

```
pair<int, int> dfs(int u, int p) {
    pair<int, int> ret = {0, u};
    for (int v : tree[u]) {
        if (v == p) continue;
        print(p, u, v, ret);
        auto x = dfs(v, u);
        chmax(ret, {x.first + 1, x.second});
    }
    return ret;
}
```

Count below

```
// counts how many pairs that sum <= limit
template <typename T>
ll count_below(vector<T>& v, int sz, ll Limit){
    // unique pairs such that v[i]+v[j] <= limit
    assert(is_sorted(v.begin(), v.end()));
    ll total = 0;
    for (int l = 0, r = sz-1; l < r; l++){
        while(r > l && v[l] + v[r] > Limit)
            r--;
        total += max(0, r-l);
    }
    return total;
}
```

Kadane

```
template<class T>
array<ll, 3> kadane(const vector<T>& arr) {
    ll mx = LLONG_MIN, csum = 0;
    ll s = 0, e = 0, ts = 0, sz = arr.size();
    for (ll i = 0; i < sz; i++) {
        if (csum + arr[i] > arr[i]) {
            csum += arr[i];
        } else {
            csum = arr[i];
            ts = i;
        }
        if (csum > mx) {
            mx = csum;
            s = ts;
            e = i;
        }
    }
    return {mx, s, e};
}
```

Max Xor Subset In Range

```
// Given queries L,R find max XOR subset in range
const int N = 5e5 + 5;
vector<pair<int,int>>qqq[N];
int ans[N],basis[22],arr[N],last[22];

void add(int ind,int x){
    for (int i = 21; i >= 0; --i) {
        if((x >> i & 1) == 0)continue;
        if(ind > last[i]){
            swap(x,basis[i]);
            swap(ind,last[i]);
        }
        x ^= basis[i];
    }
}
int query(int ind){
    int ret = 0;
    for (int i = 21; i >= 0; --i) {
        if(last[i] >= ind){
            ret = max(ret,ret ^ basis[i]);
        }
    }
    return ret;
}

void solve() {
    int n;cin >> n;
    memset(last,-1,sizeof last);
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }
    int q;cin >> q;
    for (int i = 0; i < q; ++i) {
        int l,r;cin >> l >> r;
        l--,r--;
        qqq[r].emplace_back(l,i);
    }
    for (int i = 0; i < n; ++i) {
        add(i,arr[i]);
        for(auto &x:qqq[i]){
            int l = x.fi;
            int ind = x.se;
            ans[ind] = query(l);
        }
    }
    for (int i = 0; i < q; ++i) {
        cout << ans[i] << endl;
    }
}
```

Meet in the middle

```
vector<int> a;
auto get_subset_sums = [&](int l, int r) -> vector<ll> {
    int len = r - l + 1;
    vector<ll> res;
    for (int i = 0; i < (1 << len); i++) {
        ll sum = 0;
        for (int j = 0; j < len; j++)
            if (i & (1 << j))
                sum += a[l + j];

        res.push_back(sum);
    }
    return res;
};
vector<ll> left = get_subset_sums(0, n / 2 - 1);
vector<ll> right = get_subset_sums(n / 2, n - 1);
```

```
sort(left.begin(), left.end());
sort(right.begin(), right.end());
ll ans = 0;
for (ll i : left) {
    auto low_iterator = lower_bound(right.begin(), right.end(), x - i);
    auto high_iterator = upper_bound(right.begin(), right.end(), x - i);
    ans += high_iterator - low_iterator;
}
```

Z-Algo

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

longest common substring in O(n log(k)) [automaton]

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

const int MAXLEN = 100010;
state st[MAXLEN * 2];
int sz, last;

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

void sa_extend(char c) {
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    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;
            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;
}

string lcs (string S, string T) {
    sa_init();
    for (int i = 0; i < S.size(); i++)
        sa_extend(S[i]);

    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);
}
```

Largest lexicographical substring

```
string largestLexSubstring(const string &s) {
    int n = int(s.size());
    int i = 0, j = 1, k = 0;

    while (j + k < n) {
        if (s[i + k] == s[j + k]) k++;
        else if (s[i + k] < s[j + k])
            i = max(i + k + 1, j), j = i + 1, k = 0; /* change it to > if you
want lowest */
        else j = j + k + 1, k = 0;
    }

    return s.substr(i);
}
```

count distinct palindromes [Even, Odd] in O(N)

```
struct PalindromicTree {
    struct Node {
        array<int,26> next;
        int len, link;
        Node(int l=0): len(l), link(0) { next.fill(0); }
    };
    vector<Node> tree;
    string s;
    int last;

    PalindromicTree(int n = 0) {
        tree.reserve(n + 3);
        // two roots: len = -1 and len = 0
        tree.emplace_back(-1);
        tree.emplace_back(0);
        tree[0].link = 0;
        tree[1].link = 0;
        last = 1;
    }

    void add(char c) {
        s.push_back(c);
        int pos = s.size() - 1;
        int cur = last;
        while (true) {
            int curlen = tree[cur].len;
            if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == c)
                break;
            cur = tree[cur].link;
        }
        int idx = c - 'a';
        if (!tree[cur].next[idx]) {
            tree.emplace_back(tree[cur].len + 2);
            int newNode = tree.size() - 1;
            int linkCandidate = tree[cur].link;
            while (true) {
                int candlen = tree[linkCandidate].len;
                if (pos - 1 - candlen >= 0 && s[pos - 1 - candlen] == c)
                    break;
                linkCandidate = tree[linkCandidate].link;
            }
            tree[newNode].link = tree[linkCandidate].next[idx]
                                 ? tree[linkCandidate].next[idx]
                                 : 1;
            tree[cur].next[idx] = newNode;
        }
        last = tree[cur].next[idx];
    }

    // total distinct palindromes (excluding the two roots)
    int countDistinct() const {
        return tree.size() - 2;
    }

    // count even-length and odd-length palindromes separately
    pair<int,int> countEvenOdd() const {
        int even = 0, odd = 0;
        for (int i = 2; i < (int)tree.size(); ++i) {
            if (tree[i].len % 2 == 0) ++even;
            else ++odd;
        }
        return {even, odd};
    }
};
```

palindromes [Odd, Even, Longest] in O(N)

```
struct Manacher {
    vector<int> d1, d2;
    int odd_palindromes = 0, even_palindromes = 0;
    int max_len = 0;

    Manacher(const string& s) {
        int n = s.size();
        d1.assign(n, 0);
        d2.assign(n, 0);

        // odd-length centers
        for (int i = 0, l = 0, r = -1; i < n; i++) {
            int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
            while (i - k >= 0 && i + k < n && s[i - k] == s[i + k]) k++;
            d1[i] = k--;
            odd_palindromes += d1[i];

            // update max odd-pal length = 2*d1[i] - 1
            max_len = max(max_len, 2*d1[i] - 1);

            if (i + k > r) l = i - k, r = i + k;
        }

        // even-length centers
        for (int i = 0, l = 0, r = -1; i < n; i++) {
            int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
            while (i - k - 1 >= 0 && i + k < n && s[i - k - 1] == s[i + k])
k++;
            d2[i] = k--;
            even_palindromes += d2[i];

            // update max even-pal length = 2*d2[i]
            max_len = max(max_len, 2*d2[i]);

            if (i + k > r) l = i - k - 1, r = i + k;
        }
    }
};

auto manacher(const string &t) { // 0-based
    string s = "%#";
    s.reserve(t.size() * 2 + 3);
    for(char c : t) s += c + "#"s;
    s += '$';
    // t = aabaacaabaa -> s = %#a#a#b#a#a#c#a#a#b#a#a#$

    vector<int> res(s.size());
    for(int i = 1, l = 1, r = 1; i < s.size(); i++) {
        res[i] = max(0, min(r - i, res[l + r - i]));
        while(s[i + res[i]] == s[i - res[i]]) res[i]++;
        if(i + res[i] > r) {
            l = i - res[i];
            r = i + res[i];
        }
    }
    for(auto &i : res) i--;
    return vector(res.begin() + 2, res.end() - 2); // a#a#b#a#a#c#a#a#b#a#a
    // get max odd len = res[2 * i]; aba -> i = b
    // get max even len = res[2 * i + 1]; abba -> i = first b
}
```

Max Flow (Dinik)

```
class Dinic { // O(n^2 * m), 0-based
   private:
    struct E {
        int to, rev;
        ll c, oc;
        ll f() { return max(oc - c, 0LL); }
    };
    vector<int> lvl, ptr, q;
    vector<vector<E>> g;
    ll dfs(int v, int t, ll f) {
        if (v == t || !f) return f;
        for (int &i = ptr[v]; i < g[v].size(); ++i) {
            E &e = g[v][i];
            if (lvl[e.to] == lvl[v] + 1 && e.c)
                if (ll p = dfs(e.to, t, min(f, e.c))) {
                    e.c -= p;
                    g[e.to][e.rev].c += p;
                    return p;
                }
        }
        return 0;
    }

   public:
    Dinic(int n) : lvl(n), ptr(n), q(n), g(n) {}
    void add_edge(int u, int v, ll c) { // directed
        g[u].push_back({v, (int)g[v].size(), c, c});
        g[v].push_back({u, (int)g[u].size() - 1, 0, 0});
    }
    ll calc(int s, int t) { // source to destination
        ll flow = 0;
        while (true) {
            fill(lvl.begin(), lvl.end(), 0);
            int qs = 0, qe = 0;
            lvl[q[qe++] = s] = 1;
            while (qs < qe && !lvl[t]) {
                int v = q[qs++];
                for (auto &e : g[v])
                    if (!lvl[e.to] && e.c) lvl[q[qe++] = e.to] = lvl[v] + 1;
            }
            if (!lvl[t]) break;
            fill(ptr.begin(), ptr.end(), 0);
            while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
        }
        return flow;
    }
    void reset() { // before calling calc again
        for (auto &adj : g)
            for (auto &e : adj) e.c = e.oc;
    }
    bool leftOfMinCut(int a) { return lvl[a] != 0; }
};
```

Dynamic Connectivity

```
struct Query {
    char t;
    int u, v;
};

struct Elem {
    int u, v, szU, cnt;
};

struct DSURollback {
    // offline : [+ a b] add edge between a, b
    //           [- a b] remove edge between a, b
    //           [?] number of connected components
    int cnt;
    stack<Elem> st;
    vector<bool> ans;
    vector<int> sz, par;
    map<int, vector<pair<int, int>>> g;

    DSURollback(int n) {
        cnt = n;
        par.resize(n + 1);
        sz.resize(n + 1, 1);
        iota(par.begin(), par.end(), 0);
    }

    void rollback(int x) {
        while (st.size() > x) {
            auto e = st.top();
            st.pop();
            cnt = e.cnt;
            sz[e.u] = e.szU;
            par[e.v] = e.v;
        }
    }

    int findSet(int u) {
        return par[u] == u ? u : findSet(par[u]);
    }

    void update(int u, int v) {
        st.push({u, v, sz[u], cnt});
        cnt--;
        par[v] = u;
        sz[u] += sz[v];
    }

    void unionSet(int u, int v) {
        u = findSet(u);
        v = findSet(v);
        if (u != v) {
            if (sz[u] < sz[v])
                swap(u, v);
            update(u, v);
        }
    }

    void solve(int x, int l, int r) {
        int cur = st.size();

        for (auto i: g[x])
            unionSet(i.first, i.second);

        if (l == r) {
            if (ans[l])
                cout << cnt << endl;
            rollback(cur);
            return;
        }
        int m = (l + r) >> 1;
        solve(x * 2, l, m);
        solve(x * 2 + 1, m + 1, r);
        rollback(cur);
    }

    void traverse(int x, int lX, int rX, int l, int r, int u, int v) {
        if (rX < l || lX > r)
            return;
        if (lX >= l && rX <= r) {
            g[x].emplace_back(u, v);
            return;
        }
        int m = (lX + rX) >> 1;
        traverse(x * 2, lX, m, l, r, u, v);
        traverse(x * 2 + 1, m + 1, rX, l, r, u, v);
    }

    void build(vector<Query> &queries) {
        int q = queries.size();
        ans.resize(q);
        map<pair<int, int>, vector<pair<int, int>>> mp;
        for (int i = 0; i < queries.size(); i++) {
            auto cur = queries[i];
            if (cur.u > cur.v)
                swap(cur.u, cur.v);
            if (cur.t == '?')
                ans[i] = 1;
            else if (cur.t == '+')
                mp[{cur.u, cur.v}].emplace_back(i, queries.size());
            else {
                mp[{cur.u, cur.v}].back().second = i - 1;
                traverse(1, 0, q - 1, mp[{cur.u, cur.v}].back().first,
mp[{cur.u, cur.v}].back().second, cur.u, cur.v);
            }
        }

        for (auto i: mp) {
            for (auto j: i.second) {
                if (j.second == q)
                    traverse(1, 0, q - 1, j.first, q - 1, i.first.first,
i.first.second);
            }
        }
    }
};
```

```
void testCase() {
    int n, q;
    cin >> n >> q;
    if (!q)
        return;
    DSURollback dsu(n);
    vector<Query> queries(q);
    char t;
    int x, y;
    for (int i = 0; i < q; i++) {
        cin >> t;
        if (t == '?')
            queries[i] = {t, 0, 0};
        else {
            cin >> x >> y;
            if (y < x)
                swap(x, y);
            queries[i] = {t, x, y};
        }
    }

    dsu.build(queries);
    dsu.solve(1, 0, q - 1);
}
```

Max bipartite matching (Karp) [with building]

```
/// fast, matching, edges in form [i, j], call with (nl, nr, edges)
// edges: (i, l[i]) || i == r[l[i]] || l[i] != -1
struct matching2 {
    vector<int> g, l, r; int ans;
    matching2(int n, int m, const vector<pair<int,int>> &e)
    : g(e.size()), l(n, -1), r(m, -1), ans(0) {
        vector<int> deg(n + 1), a, p, q(n);
        for (auto &[x, y] : e) deg[x]++;
        for (int i = 1; i <= n; i++) deg[i] += deg[i - 1];
        for (auto &[x, y] : e) g[--deg[x]] = y;
        for (bool match=true; match;) {
            a.assign(n,-1), p.assign(n,-1); int t=0; match=false;
            for (int i = 0; i < n; i++)
                if (l[i] == -1) q[t++] = a[i] = p[i] = i;
            for (int i = 0; i < t; i++) {
                int x = q[i];
                if (~l[a[x]]) continue;
                for (int j = deg[x]; j < deg[x + 1]; j++) {
                    int y = g[j];
                    if (r[y] == -1) {
                        while (~y) r[y] = x, swap(l[x], y), x = p[x];
                        match = true; ans++; break;
                    }
                    if(p[r[y]]==-1)q[t++]=y=r[y],p[y]=x,a[y]=a[x];
                }
            }
        }
    }
};

struct matching {
    int nl, nr;
    vector<vector<int>> g;
    vector<int> dis, ml, mr;
    explicit matching(int nl, int nr) : nl(nl), nr(nr), g(nl), dis(nl), ml(nl, -1), mr(nr, -1) { }

    void add(int l, int r) { // [i, j]
        g[l].push_back(r);
    }

    void bfs() {
        queue<int> q;
        for(int u = 0; u < nl; u++) {
            if(ml[u] == -1) q.push(u), dis[u] = 0;
            else dis[u] = -1;
        }
        while(!q.empty()) {
            int l = q.front(); q.pop();
            for(int r : g[l]) {
                if(mr[r] != -1 && dis[mr[r]] == -1)
                    q.push(mr[r]), dis[mr[r]] = dis[l] + 1;
            }
        }
    }

    bool canMatch(int l) {
        for(int r : g[l]) if(mr[r] == -1)
            return mr[r] = l, ml[l] = r, true;
        for(int r : g[l]) if(dis[l] + 1 == dis[mr[r]] && canMatch(mr[r]))
            return mr[r] = l, ml[l] = r, true;
        return false;
    }

    int maxMatch() {
        int ans = 0, turn = 1;
        while(turn) {
            bfs(), turn = 0;
            for(int l = 0; l < nl; l++) if(ml[l] == -1)
                turn += canMatch(l);
            ans += turn;
        }
        return ans;
    }

    pair<vector<int>, vector<int>> minCover() {
        vector<int> L, R;
        for (int u = 0; u < nl; ++u) {
            if(dis[u] == -1) L.push_back(u);
            else if(ml[u] != -1) R.push_back(ml[u]);
        }
        return {L, R};
    }
};
```

Dijkstra

```
vector<bool> vis(101);
vector<int> dist(101, 1e18);
dist[a] = 0;
priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>>
pq;
pq.push({0, a}); // distance, source
while (pq.size()) {
    auto [d, top] = pq.top();
    pq.pop();
    if (vis[top]) continue;
    vis[top] = 1;

    // children connected to top
    if (top + 1 <= b && dist[top] + x < dist[top+1]) {
        dist[top+1] = dist[top] + x;
        pq.push({dist[top+1], top + 1});
    }
    if ((top ^ 1) <= b && dist[top] + y < dist[top^1]) {
        dist[top^1] = dist[top] + y;
        pq.push({dist[top^1], top^1});
    }
}
cout << (dist[b] == (int)1e18 ? -1 : dist[b]) << '\n';
```

Euler Tour

```
in[u] = timer;
eul[timer++] = val[u];
for (int v : tree[u])
    if (v != p)
        euler(v, u);
out[u] = timer;
// out[u] - in[u] => 'u' subtree size
// query on subtree 'u': [ in[u], out[u]-1 ]
```

Topological Sort

```
queue<int> queue;
for (int i = 0; i < n; i++)
     if (in_degree[i] == 0) { queue.push(i); }

vector<int> top_sort;
while (!queue.empty()) {
     int curr = queue.front();
     queue.pop();
     top_sort.push_back(curr);
     for (int next : graph[curr]) {
          if (--in_degree[next] == 0) { queue.push(next); }
     }
}
```

Tree Hash (not rooted)

```
string hashGraph(vector<vector<int>> &g) {
    int n = int(g.size()), rem = n;
    vector<int> deg(n);
    queue<int> q;

    for(int i = 0; i < n; i++) {
        if(g[i].size() <= 1) q.push(i);
        else deg[i] = int(g[i].size());
    }
    vector<vector<string>> hash(n);
    auto calc = [&](int i) {
        sort(hash[i].begin(), hash[i].end());
        string res = "(";
        for(string &s : hash[i]) res += s;
        res += ')';
        return res;
    };
    while(rem > 2) {
        int sz = int(q.size()); rem -= sz;
        while(sz--) {
            int u = q.front(); q.pop();
            string curr = calc(u);
            for(int nxt : g[u]) {
                hash[nxt].push_back(curr);
                if(--deg[nxt] == 1) q.push(nxt);
            }
        }
    }
    int h1 = q.front();
    string s1 = calc(q.front()); q.pop();
    if(q.empty()) return s1;
    int h2 = q.front();
    string s2 = calc(q.front()); q.pop();
    hash[h1].push_back(s2);
    hash[h2].push_back(s1);
    return min(calc(h1), calc(h2));
}
```

bellman ford

```
// n and m tends to(1e13) => time complexity(sqrt(n))
const int INF = 1000000000;
vector<vector<pair<int, int>>> adj;
// return false if a negative cycle reachable from s exist
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;
}
```

Dynamic max subarray sum

```
struct info {
    int sum, pref, suff, ans, mnelement = -1e4-10;;
    info(int x) {
        sum = pref = suff = x;
//        ans = max<int>(x, 0);
        ans = x; // if empty subarray is not allowed
    }
    info() { // default value
        sum = pref = suff = ans = mnelement;
    }
    friend info operator+(const info &l, const info &r) {
        info ret;
        ret.sum = l.sum + r.sum;
        ret.pref = max(l.pref, l.sum + r.pref);
        ret.suff = max(r.suff, r.sum + l.suff);
        ret.ans = max({l.ans, r.ans, l.suff + r.pref});
        return ret;
    }
};
```

nth fib number

```
pair<ll, ll> fib(ll n, ll MOD) { // return f_n, f_n+1
    if (n == 0)
        return {0, 1};
    auto p = fib(n >> 1, MOD);
    ll c = (p.first * (2 * p.second % MOD - p.first + MOD)) % MOD;
    ll d = (p.first * p.first % MOD + p.second * p.second % MOD) % MOD;
    if (n & 1)
```

```
        return {d, (c + d) % MOD};
    else
        return {c, d};
}
```

Number of divisors up to 1e18

```
namespace countdivisors {
    // for one second: 5000 1e18, 100000 1e9, 200000 1e6
    using ull = unsigned long long;
    using u128 = __uint128_t;
    static mt19937_64
rnd(chrono::steady_clock::now().time_since_epoch().count());
    u128 mul128(ull a, ull b, ull m) { return (u128)a * b % m; }
    ull mul_mod(ull a, ull b, ull m) { return (ull)mul128(a, b, m); }
    ull pow_mod(ull a, ull d, ull m) {
        ull r = 1;
        while (d) {
            if (d & 1) r = mul_mod(r, a, m);
            a = mul_mod(a, a, m);
            d >>= 1;
        }
        return r;
    }
    bool isPrime(ull n) {
        if (n < 2) return false;
        for (ull p : vector<ull>({2ULL,3,5,7,11,13,17,19,23,29,31,37}))
            if (n % p == 0) return n == p;
        ull d = n - 1, s = 0;
        while (!(d & 1)) d >>= 1, ++s;
        for (ull a :
vector<ull>({2ULL,325,9375,28178,450775,9780504,1795265022})) {
            if (a % n == 0) continue;
            ull x = pow_mod(a, d, n);
            if (x == 1 || x == n - 1) continue;
            bool comp = true;
            for (ull r = 1; r < s; ++r) {
                x = mul_mod(x, x, n);
                if (x == n - 1) { comp = false; break; }
            }
            if (comp) return false;
        }
        return true;
    }
    ull pollards_rho(ull n) {
        if (n % 2 == 0) return 2;
        while (true) {
            ull c = uniform_int_distribution<ull>(1, n - 1)(rnd);
            auto f = [&](ull x){ return (mul_mod(x, x, n) + c) % n; };
            ull x = rnd() % n, y = x, d = 1;
            while (d == 1) {
                x = f(x); y = f(f(y));
                d = gcd<ull>(x > y ? x - y : y - x, n);
            }
            if (d < n) return d;
        }
    }
    void factor(ull n, map<ull,int>& cnt) {
        if (n < 2) return;
        if (isPrime(n)) { cnt[n]++; return; }
        ull d = pollards_rho(n);
        factor(d, cnt);
        factor(n/d, cnt);
    }
    ull ans(ull n) {
        map<ull,int> cnt;
        factor(n, cnt);
        ull res = 1;
        for (auto [p, e] : cnt) res *= (e + 1);
        return res;
    }
}
```

sum of: n mod 1 + n mod 2 + n mod 3 + .. + n mod m

```
// n and m tends to(1e13) => time complexity(sqrt(n))
void solve(int idx){
    int k=n/idx;
    int st=n%idx%mod;
    int l=n/(k+1);
    int len=(idx-l)%mod;
    ans=(ans + st*len%mod + k%mod*(len*(len-1)/2%mod)%mod)%mod;
    if(l) solve(l);
}

signed main() {
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin>>n>>m;
    solve(m);
    cout<<ans;
}
```

max subarray xor

```
pref = 0;
trie.insert(0); // Insert base case: prefix XOR = 0
for (int i = 0; i < n; i++)
    cin >> x;
    pref ^= x;
    ans = max(ans, trie.max(pref));
    trie.insert(pref);
}
```

Combinatorics

```
namespace comb {
    const int mod = 1e9 + 7;
    int MXS_ = 1;
    vector<int> fac_(1, 1), inv_(1, 1);

    int fp(int b, int p = mod - 2, int MOD = mod) {
        int ans = 1;
        while(p) {
            if(p & 1) ans = int(ans * 1LL * b % MOD);
            b = int(b * 1LL * b % MOD);
            p >>= 1;
        }
        return ans;
    }

    void up_(int nw) {
        if (MXS_ > nw) return;
        nw = max(MXS_ << 1, 1 << (__lg(nw) + 1));
        fac_.resize(nw), inv_.resize(nw);
        for(int i = MXS_; i < fac_.size(); i++)
            fac_[i] = int(fac_[i - 1] * 1LL * i % mod);

        inv_.back() = fp(fac_.back(), mod - 2);
        for(int i = int(inv_.size()) - 2; i >= MXS_; i--)
            inv_[i] = int(inv_[i + 1] * 1LL * (i + 1) % mod);
        MXS_ = nw;
    }

    inline int nCr(int n, int r) {
        if(r < 0 || r > n) return 0;
        up_(n);
        return int(fac_[n] * 1LL * inv_[r] % mod * inv_[n - r] % mod);
    }
    inline int nCr1(int n, int r) {
        if(r < 0 || r > n) return 0;
        r = min(r, n - r);
        up_(r);
        int ans = inv_[r];
        for(int i = n - r + 1; i <= n; i++) {
            ans = int(ans * 1LL * i % mod);
        }
        return ans;
    }
    inline int nPr(int n, int r) {
        if(r < 0 || r > n) return 0;
        up_(n);
        return int(fac_[n] * 1LL * inv_[n - r] % mod);
    }

    inline int add(int x, int y) {
        x = y < 0? x + y + mod: x + y;
        return x >= mod? x - mod: x;
    }
    inline int mul(int x, int y) {
        return int(x * 1LL * y % mod);
    }

    inline int sub(int x, int y) {
        return ((x - y) % mod + mod) % mod;
    }

    inline int Inv(int x) {
        return fp(x, mod - 2);
    }

    inline int divide(int a, int b) {
        return mul(a, Inv(b));
    }

    inline int catalan(int n) {
        return mul(nCr(2 * n, n), Inv(n + 1));
    }

    inline int StarsAndPars(int n, int k) {
        return nCr(n + k - 1, k - 1);
    }
}
```

Sieve, Factors, Divisors, Phi, 2d gcd

```
const int NS = 1e7;
const int NP = (NS/log(NS)) * (1 + 1.28 / log(NS));
int prSz;
int spf[NS], prm[NP];

auto pre_Sieve = []() {
    for (int i = 2; i < NS; i++){
        if(!spf[i]) spf[i] = prm[prSz++] = i;
        for(int j = 0; i * prm[j] < NS; j++) {
            spf[i * prm[j]] = prm[j];
            if(spf[i] == prm[j]) break;
        }
    }
    return 0;
}();

auto factors(int n) {
    vector<array<int, 2>> res;
    if(n < 2) return res;
    int p = spf[n];
    while(p > 1) {
        res.push_back({p, 0});
        while(n % p == 0) n /= p, res.back()[1]++;
        p = spf[n];
    }
    return res;
}

auto getDivisors(int _n) {
    auto _fac = factors(_n);
    int cnt = 1;
    for(auto [pr, pw] : _fac) cnt *= pw + 1;
    vector<int> res(1, 1); res.reserve(cnt);

    for(auto [pr, pw] : _fac)
        for(int i = int(res.size()) - 1; i >= 0; i--)
            for(int b = pr, j = 0; j < pw; j++, b *= pr)
                res.push_back(res[i] * b);
    sort(res.begin(), res.end());
    return res;
}

bool isPrime(ll n) {
    if(n < 4) return n > 1;
    if(n % 2 == 0 || n % 3 == 0) return false;
    for (ll i = 5; i * i <= n; i += 6)
        if (n % i == 0 || n % (i + 2) == 0)
            return false;
    return true;
}

void phi_1_to_n(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;
        }
    }
}

// 2d gcd in n^2
vector<vector<int>> GCD(N, vector<int>(N, 1));
for(int d = 2; d < N; ++d)
    for(int i = d; i < N; i += d)
        for(int j = d; j < N; j += d)
            GC[i][j] = d;
```

Egcd, linear diaphontine, Mod Inv

```
array<ll, 3> eGcd(ll a, ll b) {
    if (b == 0) return {a, 1, 0};
    auto [g, x1, y1] = eGcd(b, a % b);
    return {g, y1, x1 - (a / b) * y1};
}

ax0 + by0 = g ==> ax + by = c
x0 *= c/g, y0 *= c/g

all solutions:
           x = x0 + k * (b/g),
            y = y0 - k * (a/g);

    int ceildiv(int a, int b) {
        if ((a ^ b) >= 0) return (a + b - 1) / b;
        else return a / b;
    }
    int flordiv(int a, int b) {
        if ((a ^ b) >= 0) return a / b;
        else return (a - b + 1) / b;
    }

non-negative solution:
        k >= ceildiv(-x*g,b)
        k <= flordiv(x*g, a)

bool havenonnegsol(int a, int b, int c, int& x, int& y) {
    int g = egcd(a, b, x, y);
    x *= c/g;
    y *= c/g;
    int l1 = ceildiv(-x*g, b);
    int l2 = flordiv(y*g, a);
    return l1 <= l2;
}

// MOD INV
ll modInv(ll a, ll m) {
    ll x = 1, x1 = 0, q, t, b = m;
    while(b) {
        q = a / b;
        a -= q * b, t = a, a = b, b = t;
        x -= q * x1, t = x, x = x1, x1 = t;
    }
    assert(a == 1);
    return (x + m) % m;
}
```

Math

Stirling numbers of the first kind : the number of permutations of n elements with k disjoint cycles
S(n,k) = n * S(n-1,k) + S(n-1,k-1)
Sum k = 0 -> n of S(n,k) = n!

Stirling numbers of the second kind : the number of ways to partition a set of n elements into k nonempty subsets
S(n,k) = k * S(n-1,k) + S(n-1,k-1)
Sum k = 0 -> n of S(n,k) = Bn

Bell numbers : the possible partitions of a set into nonempty subsets
Sum k = 0 -> n-1 of n-1Ck * Bk = Bn

Sum of first n even numbers : n*(n+1)

```
Sum of first n odd numbers : n*n

Sum of squares of first n numbers:
n*(n+1)*(2*n+1)/6
Sum of squares of first n even numbers:
2*n*(n+1)*(2*n+1)/3
Sum of squares of first n odd numbers:
n*(2*n+1)*(2*n-1)/3

Number of ways to pick equal number of elements from two sets : (n+m)C(m)

Sum of phi(d) for all d | n is equal to n.
Number of pairs (x,y) that satisfy x+y=n and gcd(x,y)=1 is phi(n).

Sum(nCk) for k [0,n] = 2^n
Sum(mCk) for all m [0,n] = (n+1)C(k+1)
Sum((n+k)Ck) for all k [0,m] = (n+m+1)C(m)
Sum((nCk)^2) for all k [0,n] = (2*n)Cn
Sum(i*nCi) for all i [1,n] = n*2^(n-1)
Sum((n-i)Ci) for all i [0,n] = F(n+1)
Number of arrays with size n and sum m = (n-1+m)C(m) = (n-1+m)C(n-1)

P(A|B) is the probability of event A given that event B happened.
P(A&B) is the probability of events A and B happening.

P(A|B) = (P(B|A) * P(A) ) / P(B)
P(A|B) = P(A&B) / P(B)

Divisibility:
2 if the rightmost digit is divisible by 2
3 if the sum of the digits is divisible by 3
4 if the number formed by the last two digits is divisible by 4
5 if the rightmost digit is 0 or 5
6 if it's divisible by 2 and 3
7 if The number formed by all digits except the right-most digit - (2 * right-
most digit) is divisible by 7
8 if the number formed by the last 3 digits is divisible by 8
9 if the sum of the digits is divisible by 9

-Getting half of the binomial expansion (Only odd indices or only even indices)
    by using (a+b)^n and (a-b)^n and adding both of them

To solve quadratic equation a*x^2 + b*x + c = 0
x = -b (+-)sqrt(b^2-4*a*c) / (2 * a)

-Sum of geometric series ar^i (i from 0 to n) = a(1 - r^n) / (1 - r)
-Sum of geometric series ar^i (i from 0 to infinity) = a / (1 - r)
```

```

- XOR from 1 to n = n%4 [n, 1, n+1, 0]

- n can be sum of 2 squares if n has no p^k [p = 3 % 4, k is odd]
    in its prime factorization
- n can be sum of 3 squares if n is not in form [4^a (8b + 7)]
- n always can be sum of >= 4 squares (remaining are zeros)

=============== FADY THE GOOOOOAAT ===============
 sum of divisors
 prime^power * prime2^power2 * ...

 ((prime^(power + 1) - 1) / (prime - 1)) * ((prime2^(power2 + 1) - 1) / (prime2 - 1)) * ...
 ================================================================
 a % m == b
 a and m not coprime
 g = gcd(a, m)
 (a / g) % (m / g) = b / g

 a^x % m == b
 a and m not coprime
 g = gcd(a, m)
 (a^(x-1) * (a / g)) % (m / g) = b / g
 ================================================================
 a^(power%phi(m)) % m;
 ================================================================
 count balanced brackets
 r=n/2  ||  or r = number of opened brackets
 nCr(n, r) - nCr(n, r-1)
 ================================================================
 // different n*n grids whose each square have m colors
 // if possible to rotate one of them so that they look the same then they same
 t = n * n;
 total = (fp(m, t)
     + fp(m, (t + 1) / 2)
     + 2 * fp(m, (t / 4) + (n % 2))) % mod;
 total = mul(total, fp(4, mod - 2));
 ================================================================
 biggest divisors
 735134400 1344 => 2^6 3^3 5^2 7 11 13 17
 73513440 768
 ================================================================
 for (int x = mask; x > 0; x = (x - 1) & mask)
 get all x such that mask = mask | x
 ================================================================
 sum from 1 to n: i * nCr(n, i) = n * (1LL << (n - 1))
 sum of odd between [1, n]  = ((n+1) / 2)^2
 sum of even between [1, n] = (n/2) * (n/2 + 1)

ll sum_total(ll l, ll r) { // sum [l, r]
    return (r - l + 1) * (l + r) / 2;
}

ll sum_even(ll l, ll r) { // sum even [l, r]
    if (l % 2 != 0) ++l;
    if (r % 2 != 0) --r;
    if (l > r) return 0;
    ll n = (r - l) / 2 + 1;
    return n * (l + r) / 2;
}

ll sum_odd(ll l, ll r) { // sum odd [l, r]
    if (l % 2 == 0) ++l;
    if (r % 2 == 0) --r;
    if (l > r) return 0;
    ll n = (r - l) / 2 + 1;
    return n * (l + r) / 2;
}

ll arithm1(ll l, ll r, ll a, ll d) { // [l, r] starting a and diff d
    if (d == 0) return (a >= l && a <= r) ? a : 0;
    ll n1 = ((l - a + d - (d > 0 ? 1 : -1)) / d);
    ll first = a + n1 * d;
    if ((d > 0 && first > r)
        || (d < 0 && first < r))
            return 0;
    ll n2 = ((r - a) / d);
    ll last = a + n2 * d;
    ll n = (n2 - n1 + 1);
    return n * (first + last) / 2;
}

ll arithm2(ll a, ll d, ll n) { // starting a, diff d, 'n' terms
    return n * (2 * a + (n - 1) * d) / 2;
}
 */

using ll = int64_t;
const int mod = 1'000'000'007, N = 1e5 + 1;

ll phi(ll x) { // sqrt(x)
    ll ans = x;
    for(ll i = 2; i * i <= x; i++) {
        if(x % i == 0) {
            while(x % i == 0) x /= i;
            ans -= ans / i;
        }
    }
    if(x > 1) ans -= ans / x;
    return ans;
}

array<ll, 2> CRT(ll a1, ll m1, ll a2, ll m2) {
    // x = a1 % m1, x = a2 % m2
    a1 %= m1, a2 %= m2;
    auto [g, q1, q2] = eGcd(m1, -m2);
    if ((a2 - a1) % g) return {-1, -1};
    ll lcm = m1 / g * m2;
    ll m = m2 / g;
    q1 = (a2 - a1) / g % m * q1 % m;
    ll res = (a1 + m1 * q1) % lcm;
    if (res < 0) res += lcm;
    return {res, lcm};
}

ll BSGS(ll a, ll b, ll p) { // a^x = b (mod p)
    a %= p, b %= p;
    if(b == 1) return 0;
    if(a == 0) return b == 0? 1: -1;
    int add = 0;
    ll g, tmp = 1;
    while ((g = gcd(a, p)) > 1) {
        if(b % g) return -1;
        p /= g, b /= g, tmp = tmp * (a / g) % p, ++add;
        if(tmp == b) return add;
    }
    b = b * modInv(tmp, p) % p;
    int n = (int)sqrtl(p) + 1;
    unordered_map<ll, int> mp;
    for (ll q = 0, cur = 1; q <= n; ++q)
        mp.emplace(cur, q), cur = cur * a % p;
    ll an = 1;
    for (ll i = 0; i < n; ++i) an = an * a % p;
    an = modInv(an, p);
    for (ll i = 0, cur = b; i <= n; ++i) {
        auto it = mp.find(cur);
        if(it != mp.end()) return i * n + it->second + add;
        cur = cur * an % p;
    }
    return -1;
```

```
}

int fp(int b, int p) {
    int res = 1;
    while(p) {
        if(p & 1) res = int(res * 1LL * b % mod);
        b = int(b * 1LL * b % mod), p >>= 1;
    }
    return res;
}

int sumNPowerM(int n, int m) { // 1^m + 2^m ... n^m
    int k = m + 3;
    vector<int> res(k);
    for(int i = 1; i < k; i++) res[i] = (res[i - 1] + fp(i, m)) % mod;
    if(n < k) return res[n];
    int facK = k;
    vector<int> p(k); p[0] = 1;
    for(int i = 1; i < k; i++) {
        p[i] = int(p[i - 1] * 1LL * (n - i) % mod);
        facK = int(facK * 1LL * i % mod);
    }
    vector<int> inv(k + 1), s(k + 1);
    inv[k] = fp(facK, mod - 2), s[k] = 1;
    for(int i = k - 1; i >= 0; i--) {
        s[i] = int(s[i + 1] * 1LL * (n - i) % mod);
        inv[i] = int(inv[i + 1] * 1LL * (i + 1) % mod);
    }
    int ans = 0;
    for(int i = 1; i < k; i++) {
//        int cur = res[i];
//        for(int j = 1; j < k; j++) {
//            if(i == j) continue;
//            cur = int(cur * 1LL * (n - j) % mod);
//            cur = int(cur * 1LL * fp(abs(i - j), mod - 2) % mod);
//        }
//        if((k - i + 1) & 1) cur = (mod - cur) % mod;
        int cur = int(res[i] * 1LL * p[i - 1] % mod * s[i + 1] % mod * inv[i -
1] % mod * inv[k - i - 1] % mod);
        if((k - i + 1) & 1) cur = (mod - cur) % mod;
        ans = (ans + cur) % mod;
    }
    return ans;
}
```

MillerRabin

```
ll binpower(ll base, ll e, ll mod) {
    ll result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (__uint128_t)result * base % mod;
        base = (__uint128_t)base * base % mod;
        e >>= 1;
    }
    return result;
}

bool check_composite(ll n, ll a, ll d, int s) {
    ll x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (__uint128_t)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
};

bool MillerRabin(long long n) {
    if (n < 2)
        return false;

    vector<int> primes = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
    for (auto a : primes)
        if (n % a == 0)
            return false;

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

    for (int a : primes) {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true;
}
```

Binary gcd

```
int bgcd(int a, int b) {
    if (!a || !b)
        return a | b;
    unsigned shift = __builtin_ctz(a | b);
    a >>= __builtin_ctz(a);
    do {
        b >>= __builtin_ctz(b);
        if (a > b)
            swap(a, b);
        b -= a;
    } while (b);
    return a << shift;
}
```

nCr, nPr without precomputation

```
ll nCr(ll n, ll r) {
    if (r < 0 || r > n) return 0;
    r = min(r, n-r);
    ll ret = 1;
    for (int i = 0; i < r; i++)
        ret = ret * (n-i) / (i+1);
    return ret;
}
ll nPr(int n, int r) {
    if (r < 0 || r > n) return 0;
    ll ret = 1;
    for (int i = 0; i < r; ++i) {
        ret *= (n - i);
    }
    return ret;
}
```

Geometry Notes

```
Generate 2 points on a line
// ax + by + c = 0
if(a == 0){
        p1 ={0,-1.0 * c/b};
        p2 = {1,-1.0 * c/b};
```

```
    }
    else{
        p1 = {-1.0* c/a,0};
        p2 = {-1.0 * (c + b)/a,1};
    }
```

You're given 4 integers which are the
coefficients A B and C of the normal equation of the
straight line and a distance value R.

```
void solve() {
    double a,b,c,r;cin >> a >> b >> c >> r;
    double base = sqrt(a * a + b * b);
    a /= base;
    b /= base;
    c /= base;
    cout << setprecision(15) << a << ' ' << b << ' ' << c + r << endl;
    cout << setprecision(15) << a << ' ' << b << ' ' << c - r << endl;
}
```

Area of the sector without an angle = (l * r) / 2
The length of the arc l = (theta / 360) * 2 * pi * r

The area of the parallelogram = the cross-product of 2 adjacent sides = 2 * area of the triangle made by 3 points.

Given 1 side of the Pythagorean Triangle ... Get the missing 2 sides :

```
ll n;
cin >> n;
if (n == 1 || n==2)
    cout << -1;
else if (n & 1)
    cout << (n * n + 1) / 2 << " " << (n * n - 1) / 2;
else
    cout << n * n / 4 + 1 << " " << n * n / 4 - 1;
```

In triangle abc angle bac cos(theta) = (b^2 + c^2 - a^2) / (2 * b * c)

n = number of sides of a regular polygon
S = side length of the polygon
ap = apothem the distance from the center of the polygon to the middle of any side
r = radius of the polygon which is the distance from the center of the polygon to any corner.
p = perimeter of the polygon

```
p = S * n
ap = S / (2 * tan(180/n)) = r * cos(180/n)
r = S / (2 * sin(180/n)) = ap / cos(180/n)
Area = (p * ap)/2 , (S^2 * n) / (4 * tan(180/n)) = ap^2 * n * tan(180/n) = (r^2
* n * sin(360/n))/2

sin(2*theta) = 2 * sin(theta) * cos(theta)
cos(2*theta) = cos(theta)^2 - sin(theta)^2 = 2 * cos(theta)^2 - 1 = 1 - 2 *
sin(2*theta)^2
sin(theta)^2 = (1 - cos(2 * theta))/2
cos(theta)^2 = (1 + cos(2 * theta))/2
tan(2*theta) = (2 * tan(theta)) / (1 - tan(theta)^2)

Circle intersection r1,r2,d where r1 >= r2
If d = r1 + r2 they touch from outside
If d = r1 - r2 they touch from inside
If r1 - r2 < d < r1 + r2 they intersect in two points

Plane equation ax + by + cz + d = 0
AB = (Bx-Ax,By-Ay,Bz-Az)
AC = (Cx-Ax,Cy-Ay,Cz-Az)
AB x AC = (a,b,c)
a = (By-Ay)*(Cz-Az)-(Cy-Ay)*(Bz-Az)
b = (Bz-Az)*(Cx-Ax)-(Cz-Az)*(Bx-Ax)
c = (Bx-Ax)*(Cy-Ay)-(Cx-Ax)*(By-Ay)
d = -(a*Ax+b*Ay+c*Az)

// Checks if four points lie in the same plane or not
bool samePlane(point a,point b,point c){
    // a * (b x c) = volume = 0
    return (a.dot(b.cross(c)) == 0);
}

void solve() {
    vector<point>v(4);
    for (int i = 0; i < 4; ++i) {
        cin >> v[i].x >> v[i].y >> v[i].z;
    }
    for (int i = 0; i < 4; ++i) {
        v[i].x -= v[3].x;
        v[i].y -= v[3].y;
        v[i].z -= v[3].z;
    }
    cout << (samePlane(v[0],v[1],v[2]) ? "YES":"NO") << endl;
}
```

Geometry

```
/*
 conj(a) -> a.imag() *= -1
 abs(point) distance between (0,0) to this point
 norm(point) squared magnitude -> real² + imag²
 hypot(x, y) -> sqrt(x² + y²)
 arg(vector) angle between this vector and x-axis
 clamp(a, l, r) == min(r, max(l, a))
 polar(rho, theta) -> make vector with length rho and angle theta
 internal angle = (n - 2) * 180 / n
 number of diagonals n * (n - 3) / 2
 Area(p) = internal_points_cnt + (boundary_points/2) - 1
 boundary_point in vector = gcd(|x2-x1|, |y2-y1|) + 1
 line have infinity point, segment have to end points
 vector(x, y) perpendicular to vector(-y, x) and (y, -x)
*/

using ll = int64_t;
using ld = double;
using point = complex<ld>;

const ll inf = 7e18;
const ld EPS = 1e-9;
const ld pi = acos(-1);

#define X real()
#define Y imag()

#define dot(a, b) (conj(a) * (b)).X
#define cross(a, b) (conj(a) * (b)).Y

struct compX{
    bool operator()(point a, point b) const {
        return a.X != b.X ? a.X < b.X : a.Y < b.Y;
    }
};
struct compY{
    bool operator()(point a, point b) const {
        return a.Y != b.Y ? a.Y < b.Y : a.X < b.X;
    }
};

int sign(ld x) {
    return (x > EPS) - (x < -EPS);
}

```

```
// =============== line, segment ==========================

// projection of point p onto line ab
point project(point a, point b, point p) {
    point ab = b - a;
    return a + ab * dot(p - a, ab) / norm(ab);
}

// works for any orientation
bool onSegment(point a, point b, point p) {
    return sign(cross(b - a, p - a)) == 0 &&
           sign(dot(p - a, p - b)) <= 0;
}

// ccw: >0 left, <0 right, =0 collinear
int ccw(point a, point b, point c) {
    return sign(cross(b - a, c - a));
}

// works for any points
ld distanceToLine(point a, point b, point p) {
    return fabs(cross(b - a, p - a)) / abs(b - a);
}

// works for any points
ld distanceToSegment(point a, point b, point p) {
    if (dot(b - a, p - a) < 0) return abs(p - a);
    if (dot(a - b, p - b) < 0) return abs(p - b);
    return distanceToLine(a, b, p);
}

// works for intersecting lines (not parallel)
point lineIntersect(point a, point b, point c, point d) {
    point ab = b - a, cd = d - c;
    return a + ab * (cross(c - a, cd) / cross(ab, cd));
}

// works for all segments (returns intersection point if exists)
bool segmentsIntersect(point a, point b, point c, point d, point &inter) {
    int d1 = ccw(a, b, c), d2 = ccw(a, b, d);
    int d3 = ccw(c, d, a), d4 = ccw(c, d, b);

    if(d1 * d2 < 0 && d3 * d4 < 0)
        return inter = lineIntersect(a, b, c, d), true;

    if(d1 == 0 && onSegment(a, b, c)) return inter = c, true;
    if(d2 == 0 && onSegment(a, b, d)) return inter = d, true;
    if(d3 == 0 && onSegment(c, d, a)) return inter = a, true;
    if(d4 == 0 && onSegment(c, d, b)) return inter = b, true;

    return false;
}

// works for any triangle
ld triangleArea(point a, point b, point c) {
    return 0.5 * fabs(cross(b - a, c - a));
}

bool pointInTriangle(point a, point b, point c, point p) {
    ld s1 = cross(b - a, p - a);
    ld s2 = cross(c - b, p - b);
    ld s3 = cross(a - c, p - c);
    return (sign(s1) >= 0 && sign(s2) >= 0 && sign(s3) >= 0) ||
           (sign(s1) <= 0 && sign(s2) <= 0 && sign(s3) <= 0);
}

// angle abc in radians
ld angle_abc(point a, point b, point c) {
    return acos(clamp<ld>(dot(a - b, c - b) / (abs(a - b) * abs(c - b)), -1,
1));
}

// ============================ Circles ================================

pair<ld, point> findCircle(point a, point b, point c) {
    point m1 = (a + b) / 2.0, m2 = (b + c) / 2.0;
    point ab = b - a, bc = c - b;
    point center = lineIntersect(m1, m1 + point(-ab.Y, ab.X),
                                 m2, m2 + point(-bc.Y, bc.X));
    return {abs(center - a), center};
}

vector<point> lineCircleIntersect(point a, point b, point center, ld r) {
    point ab = b - a, ao = center - a;
    point proj = a + ab * dot(ao, ab) / norm(ab);
    ld d = abs(proj - center);
    if (d > r + EPS) return {};
    if (abs(d - r) < EPS) return {proj};
    ld h = (ld)sqrtl(r*r - d*d);
    point dir = ab / abs(ab);
    return {proj + dir * h, proj - dir * h};
}

// in 0, 1, 2 points
```

```
vector<point> circleCircleIntersect(point c1, ld r1, point c2, ld r2) {
    ld d = abs(c2 - c1);
    if(d > r1 + r2 + EPS || d < abs(r1 - r2) - EPS) return {};
    if(abs(d) < EPS && abs(r1 - r2) < EPS) return vector(3, c1); // infinity intersection

    ld a = (r1*r1 - r2*r2 + d*d) / (2 * d), h2 = r1*r1 - a*a;
    if (h2 < -EPS) return {};

    point dir = (c2 - c1) / d, p = c1 + dir * a;
    if (abs(h2) < EPS) return {p};
    ld h = sqrt(h2);
    point offset = dir * point(0, 1) * h;
    return {p + offset, p - offset};
}

pair<ld, point> minimumEnclosingCircle(vector<point> p) {
    using circle = pair<ld, point>;
    shuffle(p.begin(), p.end(), mt19937(random_device{}()));
    auto contains = [](circle c, const vector<point>& pts) {
        return all_of(pts.begin(), pts.end(),
                      [&](auto p) {return abs(p - c.second) <= c.first + EPS;});
    };
    auto circleFrom2 = [](point a, point b) {
        point c = (a + b) / 2.0;
        return circle{abs(a - c), c};
    };
    auto circleFrom3 = [](point a, point b, point c) {
        point ab = (a + b) / 2.0, ac = (a + c) / 2.0;
        point ab_perp = (b - a) * point(0, 1), ac_perp = (c - a) * point(0, 1);
        point o = lineIntersect(ab, ab + ab_perp, ac, ac + ac_perp);
        return circle{abs(o - a), o};
    };
    vector<point> R;
    function<circle(int)> welzl = [&](int n) -> circle {
        if (n == 0 || R.size() == 3) {
            if (R.empty()) return {};
            if (R.size() == 1) return {0, R[0]};
            if (R.size() == 2) return circleFrom2(R[0], R[1]);
            return circleFrom3(R[0], R[1], R[2]);
        }
        point q = p[n - 1];
        circle D = welzl(n - 1);
        if (contains(D, {q})) return D;
        R.push_back(q);
        auto res = welzl(n - 1);
        R.pop_back();
        return res;
    };
    return welzl((int)p.size());
}

// ===================== polygon =============================

// works for any polygon (returns +1 for ccw, -1 for cw)
ld polygonSign(vector<point>& p) {
    ld area = 0;
    int n = (int)p.size();
    p.push_back(p[0]);
    for(int i = 0; i < n; ++i) area += cross(p[i], p[i + 1]);
    p.pop_back();
    return sign(0.5 * area);
}

// works for any polygon (removes dups, enforces ccw order)
void normPolygon(vector<point>& p) {
    vector<point> res;
    for(auto i : p) if(res.empty() || abs(i - res.back()) > EPS)
            res.push_back(i);

    if(res.size() > 1 && abs(res.front() - res.back()) < EPS)
        res.pop_back();

    if(polygonSign(res) < 0) reverse(res.begin(), res.end());

    p = res;
}

// works for simple polygons with integer coordinates
ll internalPointsCount(vector<point>& p) {
    ll A2 = 0, B = 0;
    int n = (int)p.size();
    p.push_back(p[0]);
    for (int i = 0; i < n; ++i) {
        point a = p[i], b = p[i + 1];
        A2 += ll(a.X * b.Y - a.Y * b.X);
        B += __gcd((ll)abs(b.X - a.X), (ll)abs(b.Y - a.Y));
    }
    p.pop_back();
    return (abs(A2) - B + 2) / 2;
}

// works for any polygon (cw or ccw, convex or not)
```

```
ld polygonArea(vector<point> &p) {
    int n = (int)p.size();
    p.push_back(p[0]);
    ld area = 0;
    for (int i = 0; i < n; ++i) area += cross(p[i], p[i + 1]);
    p.pop_back();
    return fabs(area) / 2.0;
}

// works for any polygon (cw or ccw, convex or not)
bool pointInPolygon(vector<point> &p, point o) {
    int in = 0, n = (int)p.size();
    p.push_back(p[0]);
    for (int i = 0; i < n; ++i) {
        point a = p[i], b = p[i + 1];
        if (onSegment(a, b, o)) return p.pop_back(), true;
        if ((a.Y > o.Y) != (b.Y > o.Y)) {
            ld x = a.X + (b.X - a.X) *
                          (o.Y - a.Y) / (b.Y - a.Y);
            if(x > o.X) in ^= 1;
        }
    }
    p.pop_back();
    return in;
}

// work for simple convex polygon
bool pointInConvex(vector<point> &poly, point p) {
    int n = int(poly.size());
    if(n == 1) return sign(abs(poly[0] - p)) == 0;
    if(n == 2) return onSegment(poly[0], poly[1], p);

    point f = poly[0];

    if(sign(cross(poly[1] - f, p - f)) < 0 || sign(cross(poly[n - 1] - f, p -
f)) > 0) return false;

    int l = 1, r = n - 1;
    while(r > l + 1) {
        int mid = (l + r) >> 1;
        if(sign(cross(poly[mid] - f, p - f)) > 0) l = mid;
        else r = mid;
    }
    return pointInTriangle(f, poly[l], poly[r], p);
}

// works for any simple polygon (cw or ccw)
```

```
point polygonCentroid(vector<point>& p) {
    ld A = 0, c;
    point C(0, 0);
    int n = (int)p.size();
    p.push_back(p[0]);
    for (int i = 0; i < n; ++i) {
        point cur = p[i], nxt = p[i + 1];
        c = cross(cur, nxt);
        A += c;
        C += (cur + nxt) * c;
    }
    p.pop_back();
    A *= 0.5;
    if (abs(A) < EPS) return C;
    return C / (6.0 * A);
}
```

__int128_t

```
istream& operator>>(istream& is, __int128_t& v) {
    string s; is >> s; v = 0;
    for (char c : s) if (isdigit(c)) v = v * 10 + c - '0';
    if (s[0] == '-') v = -v;
    return is;
}
ostream& operator<<(ostream& os, __int128_t v) {
    if (!v) return os << "0";
    if (v < 0) os << '-', v = -v;
    string s;
    while (v) s += '0' + v % 10, v /= 10;
    reverse(s.begin(), s.end());
    return os << s;
}
```

Floyd Tricks

```
void TransitiveClosure(int n, vector<vector<int>>& adj) {
     // 0 = disconnected, 1 = connected
     for (int k = 0; k < n; ++k)
          for (int i = 0; i < n; ++i)
               for (int j = 0; j < n; ++j)
                    adj[i][j] |= (adj[i][k] & adj[k][j]);
}

void minimax(int n, vector<vector<int>>& adj) {
     // Path such that max value on road is minimized
     for (int k = 0; k < n; ++k)
          for (int i = 0; i < n; ++i)
               for (int j = 0; j < n; ++j)
                    adj[i][j] = min(adj[i][j], max(adj[i][k], adj[k][j]));
}

void maximin(int n, vector<vector<int>>& adj) {
     // Path such that min value on road is maximized
     for (int k = 0; k < n; ++k)
          for (int i = 0; i < n; ++i)
               for (int j = 0; j < n; ++j)
                    adj[i][j] = max(adj[i][j], min(adj[i][k], adj[k][j]));
}

void longestPathDAG(int n, vector<vector<int>>& adj) {
     // Only works for DAGs (no positive cycles)
     for (int k = 0; k < n; ++k)
          for (int i = 0; i < n; ++i)
               for (int j = 0; j < n; ++j)
                    adj[i][j] = max(adj[i][j], max(adj[i][k], adj[k][j]));
}

void countPaths(int n, vector<vector<int>>& adj) {
     // Floyd-Warshall for counting number of paths
     for (int k = 0; k < n; ++k)
          for (int i = 0; i < n; ++i)
               for (int j = 0; j < n; ++j)
                    adj[i][j] += adj[i][k] * adj[k][j];
}

bool isNegativeCycle(int n, const vector<vector<int>>& adj) {
     // run warshal first
     for (int i = 0; i < n; ++i)
          if (adj[i][i] < 0)
               return true;
     return false;
}

bool anyEffectiveCycle(int n, const vector<vector<int>>& adj, int src, int
dest, int OO) {
     // run warshal first
     for (int i = 0; i < n; ++i)
          if (adj[i][i] < 0 && adj[src][i] < OO && adj[i][dest] < OO)
               return true;
     return false;
}
```

Direction arrays

```
int dx[8] = { 2, 1, -1, -2, -2, -1, 1, 2 };
int dy[8] = { 1, 2, 2, 1, -1, -2, -2, -1 }; // knight

int dx[8] = {-1,0,1,-1,1,-1,0,1};
int dy[8] = {-1,-1,-1,0,0,1,1,1}; // king

int dx[4] = {1, -1, 0, 0};
int dy[4] = {0, 0, -1, 1};
```

Random

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
ll rnd(ll l, ll r) {
    static mt19937_64
gen(chrono::steady_clock::now().time_since_epoch().count());
    return uniform_int_distribution<ll>(l, r)(gen);
}
```

Custom comparator

```
struct cmp {
    bool operator() (int a, int b) const {
        return ...;
    }
};
std::set<int, cmp> s;

template <typename T> using ordered_set = tree<T, null_type, cmp, rb_tree_tag,
tree_order_statistics_node_update>;

template <typename T, typename R> using ordered_map = tree<T, R, cmp,
rb_tree_tag, tree_order_statistics_node_update>;
```

Custom hash, pair hash

```
struct hash_pair {
    template <class T1, class T2>
    size_t operator()(const pair<T1, T2>& p) const {
        static const size_t RANDOM_SEED = rng();

        size_t h1 = std::hash<T1>{}(p.first);
        size_t h2 = std::hash<T2>{}(p.second);

        const size_t FNV_PRIME = 1099511628211ULL;
        size_t hash = RANDOM_SEED;
        hash = (hash ^ h1) * FNV_PRIME;
        hash = (hash ^ h2) * FNV_PRIME;

        return hash;
    }
};

struct hash_pair {
    template <class T1, class T2>
    size_t operator()(const pair<T1, T2>& p) const {
        static const size_t RANDOM_SEED = rng();

        size_t h1 = std::hash<T1>{}(p.first);
        size_t h2 = std::hash<T2>{}(p.second);

        const size_t FNV_PRIME = 1099511628211ULL;
        size_t hash = RANDOM_SEED;
        hash = (hash ^ h1) * FNV_PRIME;
        hash = (hash ^ h2) * FNV_PRIME;

        return hash;
    }
};
```

K-th permutation, permutation index

```
ll fac[21];
void preFac() {
    fac[0] = 1;
    for(int i = 1; i <= 20; ++i)
        fac[i] = fac[i - 1] * i;
}

vector<int> kth_permutation(int n, ll k) { // n^2
    vector<int> a(n), ans;
    iota(a.begin(), a.end(), 1);
    for (int i = n; i >= 1; i--) {
        ll f = fac[i - 1];
        ans.push_back(a[k / f]);
        a.erase(a.begin() + k / f);
        k %= f;
    }
    return ans;
}

```

```
ll permutation_index(vector<int>& p) { // n^2
    int n = int(p.size());
    vector<int> a(n);
    iota(a.begin(), a.end(), 1);

    ll k = 0;
    for (int i = 0; i < n; ++i) {
        int j = int(find(a.begin(), a.end(), p[i]) - a.begin());
        k += j * fac[n - 1 - i];
        a.erase(a.begin() + j);
    }
    return k;
}
```

shortest distance between a point m and a line segment

```
using ld = long double;
using point = complex<ld>;
pair<ld, point> pointSegDis(point m, point a, point b) {
    point ab_vec = b - a;
    point am_vec = m - a;
    ld t = real(conj(ab_vec) * am_vec) / norm(ab_vec);
    t = max(0.0L, min(1.0L, t));
    point closest = a + t * ab_vec;
    return make_pair(abs(m - closest), closest);
}
```

FloorValues

```
// code to get all different values of floor(n/i)
for (ll l = 1, r = 1; (n/l); l = r + 1) { // O(sqrt)
    r = (n/(n/l));
    // q = (n/l), process the range [l, r]
}
```

Debug.h

```
template<typename A, typename B> string to_string(pair<A,B> p);
template<typename Container> auto to_string(const Container& c)
    -> decltype(c.begin(), c.end(), string());
template<typename T, size_t N>
    string to_string(const array<T, N>& a);

string to_string(char c) { return string(1, c); }
string to_string(bool b) { return b ? "T" : "F"; }
string to_string(string s) { return "\""+s+"\""; }
string to_string(const char* s) { return string(s); }

template<typename Container>
auto to_string(const Container& c)
    -> decltype(c.begin(), c.end(), string()) {
    string s="{";
    bool first = true;
    for(const auto& item : c) {
        if(!first) s += ", ";
        s += to_string(item);
        first = false;
    }
    return s+"}";
}

template<typename T, size_t N>
string to_string(const array<T, N>& a) {
    string s="{";
    for(size_t i=0;i<N;i++) s+=(i?", ":"")+to_string(a[i]);
    return s+"}";
}

template<typename A, typename B> string to_string(pair<A,B> p) {
    return "("+to_string(p.first)+", "+to_string(p.second)+")";
}

void debug_out() { cout << "\n"; }
template<typename H, typename... T>
void debug_out(H&& h, T&&... t) {
    cout << " " << to_string(std::forward<H>(h));
    debug_out(std::forward<T>(t)...);
}
#define print(...) cout<<"["<<#__VA_ARGS__<<"]:",debug_out(__VA_ARGS__)
```