.

Dividimos y No Conquistamos (D&!C)

Contents

1 Template

1.1 C++ Template

```cpp
#include <bits/stdc++.h>
using namespace std;
#define L(i, j, n) for (int i = (j); i < (int)n; i ++)
#define SZ(x) int((x).size())
#define ALL(x) begin(x),end(x)
#define vec vector
#define pb push_back
#define eb emplace_back
using ll = long long;
using ld = long double;
void solve(){}
int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    int TT = 1;
    //cin >> TT;
    while (TT--) {solve();}
}
// IF NEEDED FOR FILE READ
// freopen("in.txt", "r", stdin);
// freopen("out.txt", "w", stdout);
```

1.2 Bash CMD

```bash
co(){g++ $1.cpp -o $1 --std=c++20 -Wall -Wshadow -Wextra}
run(){for f in `ls *.txt`;do echo $f ;./$1 < $f; done}
#Build, template.cpp must exist!
for x in {A..Z}; do mkdir $x; cp template.cpp $x/$x.cpp;done
```

1.3 Python Template

```python
import os, sys, io
finput = io.BytesIO(os.read(0, os.fstat(0).st_size)).readline
fprint = sys.stdout.write
```

1.4 Java Template

```java
import java.io.*;
import java.util.*;
import java.math.BigInteger;

public class Main {
```

```java
    static BufferedReader br;
    static PrintWriter pw;
    static StringTokenizer st;

    public static void main(String[] args) throws IOException {
        br = new BufferedReader(new FileReader("datos.txt"));
        pw = new PrintWriter("salida.txt");
        solve();
        pw.close();
    }

    static void solve() throws IOException {
        // Your code here
        BigInteger a = nextBigInteger();
        BigInteger b = nextBigInteger();
        pw.println(a.add(b));
    }

    static String next() throws IOException {
        while (st == null || !st.hasMoreTokens())
            st = new StringTokenizer(br.readLine());
        return st.nextToken();
    }

    static BigInteger nextBigInteger() throws IOException {
        return new BigInteger(next());
    }

    static int nextInt() throws IOException {
        return Integer.parseInt(next());
    }

    static long nextLong() throws IOException {
        return Long.parseLong(next());
    }

    static double nextDouble() throws IOException {
        return Double.parseDouble(next());
    }

    static String nextLine() throws IOException {
        return br.readLine();
    }
}
```

2 Search

2.1 Ternary

```cpp
// Minimo de `f` en `(l,r)`.
template<class Fun>ll ternary(Fun f, ll l, ll r) {
  for (ll d = r-l; d > 2; d = r-l) {
    ll a = l + d/3, b = r - d/3;
    if (f(a) > f(b)) l = a; else r = b;
  }
  return l + 1;
}
// para error < EPS, usar iters=log((r-l)/EPS)/log(1.618)
template<class Fun>double golden(Fun f, double l, double r, int iters){
  double const ratio = (3-sqrt(5))/2;
  double x1=l+(r-l)*ratio, x2=r-(r-l)*ratio, f1=f(x1), f2=f(x2);
  while (iters--) {
    if (f1 > f2) l=x1, x1=x2, f1=f2, x2=r-(r-l)*ratio, f2=f(x2);
    else         r=x2, x2=x1, f2=f1, x1=l+(r-l)*ratio, f1=f(x1);
  }
  return (l+r)/2;
}
```

2.2 Simulated Annealing

```cpp
using my_clock = chrono::steady_clock;
struct Random {
  mt19937_64 engine;
  Random(): engine(my_clock::now().time_since_epoch().count()) {}
  template<class Int>Int integer(Int n) {return integer<Int>(0, n);} // `[0,n)`
  template<class Int>Int integer(Int l, Int r)
    {return uniform_int_distribution{l, r-1}(engine);} // `[l,r)`
  double real() {return uniform_real_distribution{}(engine);} // `[0,1)`
} rng;
struct Timer {
  using time = my_clock::time_point;
  time start = my_clock::now();
  double elapsed() { // Segundos desde el inicio.
    time now = my_clock::now();
    return chrono::duration<double>(now - start).count();
  }
} timer;
```

```
template<class See,class Upd>struct Annealing {
  using energy = invoke_result_t<See>;
  energy curr, low;
  See see;
  Upd upd;
  Annealing(See _see, Upd _upd): see{_see}, upd{_upd}
    {curr = low = see(), upd();}
  void simulate(double s, double mult=1) { // Simula por ‘s‘ segundos.
    double t0 = timer.elapsed();
    for (double t = t0; t-t0 < s; t = timer.elapsed()) {
      energy near = see();
      auto delta = double(curr - near);
      if (delta >= 0) upd(), curr = near, low = min(low, curr);
      else {
        double temp = mult * (1 - (t-t0)/s);
        if (exp(delta/temp) > rng.real()) upd(), curr = near;
      }
    }
  }
};
auto see = [&] -> double {
    l = rng.integer(gsz);r = rng.integer(gsz);
    swap(groups[l], groups[r]);
    int ans = 0, rem =0;
    L(i,0,gsz){
        if (groups[i] > rem) {
            rem = x;
            ans ++;
        }
        rem -= groups[i];
    }
    swap(groups[l], groups[r]);
    return ans;
};
auto upd = [&] {swap(groups[l], groups[r]);};
```

3 Data structures

3.1 Fenwick

```
#define LSO(S) (S & -S) //LeastsignificantOne
struct FT { // 1-Index
    vec<int> ft; int n;
    FT(vec<int> &v): ft(SZ(v)+1), n(SZ(v)+1) { // O(n)
        L(i, 1, n){
            ft[i] += v[i-1];
            if (i + LSO(i) <= n) ft[i + LSO(i)]+=ft[i];
        }
    }
    void update(int pos, int x){ for (int it=pos;it<=n;it+=LSO(it))ft[it
        ]+=x; }
    int sum(int pos){
        int res = 0;
        for (int it=pos;it>0;it-=LSO(it))res+=ft[it];
        return res;
    }
    int getSum(int l, int r){return sum(r) - sum(l - 1);}
};
```

3.2 Fenwick - 2D

```
#define LSO(S) (S & -S)
struct BIT { // 1-Index
    vec<vec<int>> B;
    int n; // BUILD: N * N * log(N) * log(N)
    BIT(int n_ = 1): B(n_+1,vec<int>(n_+1)), sz(n_) {}
    void add(int i, int j, int delta){ // log(N) * log(N)
        for (int x = i; x <= n; x += LSO(x))
        for (int y = j; y <= n; y += LSO(y))
            B[x][y] += delta;
    }
    int sum(int i, int j){ // log(N) * log(N)
        int tot = 0;
        for (int x = i; x > 0; x -= LSO(x))
        for(int y = j; y > 0; y -= LSO(y))
            tot += B[x][y];
        return tot;
    }
    int getSum(int x1, int y1, int x2, int y2) {return sum(x2, y2) - sum
        (x2, y1) - sum(x1, y2) + sum(x1-1,y1-1);}
};
```

3.3 DSU

```
struct DSU {
    vec<int> par, sz; int n;
    DSU(int n = 1): par(n), sz(n, 1), n(n) { iota(ALL(par), 0); }
```

```
    int find(int a){return a == par[a] ? a : par[a] = find(par[a]);}
    void join(int a, int b){
        a=find(a);b=find(b);
        if (a == b) return;
        if (sz[b] > sz[a]) swap(a,b);
        par[b] = a; sz[a] += sz[b];
    }
};
```

3.4 Index Compession

```
template<class T>
struct Index{ // If only 1 use Don't need to copy T type
    vec<T> d; int sz;
    Index(const vec<T> &a): d(ALL(a)){
        sort(ALL(d)); // Sort
        d.erase(unique(ALL(d)), end(d)); // Erase continuous duplicates
        sz = SZ(d); }
    inline int of(T e) const{return lower_bound(ALL(d), e) - begin(d);}
        // get index
    inline T at(int i) const{return d[i];} // get value of index
};
```

3.5 Sparse Table

```
struct SPT {
    vec<vec<int>> st;
    SPT(vec<int> &a) {
        int n = SZ(a), K = 0; while((1<<K)<=n) K ++;
        st = vec<vec<int>>(K, vec<int>(n));
        L(i,0,n) st[0][i] = a[i];
        L(i,1,K) for (int j = 0; j + (1 << i) <= n; j ++)
            st[i][j] = min(st[i-1][j], st[i - 1][j + (1 << (i - 1))]);
                // change op
    }
    int get(int l, int r) {
        int bit = log2(r - l + 1);
        return min(st[bit][l], st[bit][r - (1<<bit) + 1]); // change op
    }
};
```

3.6 Segment tree

```
#define LC(v) (v<<1)
#define RC(v) ((v<<1)|1)
#define MD(L, R) (L+((R-L)>>1))
struct node { ll mx;ll cant; };
struct ST {
    vec<node> st; vec<ll> lz; int n;
    ST(int n = 1): st(4 * n + 10, {oo, oo}), lz(4 * n + 10, 0), n(n) {
        build(1, 0, n - 1);}
    node merge(node a, node b){
        if (a.mx == oo) return b; if (b.mx == oo) return a;
        if (a.mx == b.mx) return {a.mx, a.cant + b.cant};
        return {max(a.mx, b.mx), a.mx > b.mx ? a.cant : b.cant};
    }
    void build(int v, int L, int R){
        if (L == R){ st[v] = {0, 1}; return ;}
        int m = MD(L, R);
        build(LC(v), L, m); build(RC(v), m + 1, R);
        st[v] = merge(st[LC(v)], st[RC(v)]);
    }
    void push(int v, int L, int R){
        if (lz[v]){
            if (L != R){
                st[LC(v)].mx += lz[v]; // Apply to left
                st[RC(v)].mx += lz[v]; // And right
                lz[LC(v)] += lz[v];
                lz[RC(v)] += lz[v];
            }
            lz[v] = 0;
        }
    }
    void update(int v, int L, int R, int ql, int qr, ll w){
        if (ql > R || qr < L) return;
        push(v, L, R);
        if (ql == L && qr == R){
            st[v].mx += w; // Update acutal node
            lz[v] += w; // Add lazy
            push(v, L, R); // Initial spread
            return;
        }
        int m = MD(L, R);
        update(LC(v), L, m, ql, min(qr, m), w);
        update(RC(v), m + 1, R, max(m + 1, ql), qr, w);
        st[v] = merge(st[LC(v)], st[RC(v)]);
    }
```

```
    node query(int v, int L, int R, int ql, int qr){
        if (ql > R || qr < L) return {oo, oo};
        push(v, L, R);
        if (ql == L && qr == R) return st[v];
        int m = MD(L, R);
        return merge(query(LC(v), L, m, ql, min(m, qr)), query(RC(v), m
            + 1, R, max(m + 1, ql), qr));
    }
    node query(int l, int r){return query(1, 0, n - 1, l, r);}
    void update(int l, int r, ll w){update(1, 0, n - 1, l, r, w);}
};
```

3.7 Segment Tree Iterativo

```
struct STI {
    vec<ll> st; int n, K;
    STI(vec<ll> &a): n(SZ(a)), K(1) {
        while(K<=n) K<<=1;
        st.assign(2*K, 0); // 0 default
        L(i,0,n) st[K+i] = a[i];
        for (int i = K - 1; i > 0; i --) st[i] = st[i*2] + st[i*2+1];}
    void upd(int pos, ll w) {
        pos += K; st[pos] += w;
        while((pos>>=1) > 0) st[pos] = st[pos * 2] + st[pos * 2 + 1];}
    ll query(int l, int r) { // [l, r]
        ll res = 0;
        for (l += K, r += K + 1; l < r; l>>=1, r>>=1){
            if (l & 1) res += st[l++];
            if (r & 1) res += st[--r];
        }
        return res;
    }
};
```

3.8 Segment Tree Persistente

```
struct Vertex{Vertex * l, *r;int sum;};
const int MVertex = 6000000; // ~= N * logN * 2
Vertex pool[MVertex]; // the idea is to keep versions on vec<Vertex*>
    roots; roots.pb(build(ST_L, ST_R));
int p_num = 0;          //
Vertex * init_leaf(int x) {
    pool[p_num].sum = x;
    pool[p_num].l = pool[p_num].r = NULL;
    return &pool[p_num++];
}
Vertex * init_node(Vertex * l, Vertex * r) {
    int sum = 0;
    if (l) sum += l->sum;
    if (r) sum += r->sum;
    pool[p_num].sum = sum; pool[p_num].l = l; pool[p_num].r = r;
    return &pool[p_num++];}
Vertex * build(int L, int R){
    if (L == R){return init_leaf(0);}
    int m = MD(L, R); return init_node(build(L, m), build(m + 1, R));}
Vertex * update(Vertex * v, int L, int R, int pos, int w){
    if (L == R)return init_leaf(v->sum + w);
    int m = MD(L, R);
    if (pos <= m) return init_node(update(v->l, L, m, pos, w), v->r);
    return init_node(v->l, update(v->r, m + 1, R, pos, w));}
int query(Vertex * vl, Vertex * vr, int L, int R, int ql, int qr) {
    if (!vl || !vr) return 0;
    if (ql > R || qr < L) return 0;
    if (ql == L && qr == R) {return vr->sum - vl->sum;}
    int m = MD(L, R);
    return query(vl->l, vr->l, L, m, ql, min(m, qr)) +
        query(vl->r, vr->r, m + 1, R, max(m + 1, ql), qr);}
```

3.9 Policy Based

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
template<typename Key, typename Val=null_type>
using indexed_set = tree<Key, Val, less<Key>, rb_tree_tag,
                         tree_order_statistics_node_update>;
// indexed_set<char> s;
// char val = *s.find_by_order(0); // acceso por indice
// int idx = s.order_of_key('a'); // busca indice del valor
template<class Key,class Val=null_type>using htable=gp_hash_table<Key,
    Val>;
// como unordered_map (o unordered_set si Val es vacio), pero sin metodo
     count
```

3.10 SQRT Decomposition

```
struct SQRTDecomp {
    vec<int> B, Bs, Bid; int n;// DEFINE BLOCK_SIZE ~= sqrt N
```

```
    SQRTDecomp(int n_): B(n_), Bid(n_), n(n_), Bs((n_ + BLOCK_SIZE - 1)/
        BLOCK_SIZE) {
        L(i,1,n) Bid[i] = Bid[i - 1] + (i % BLOCK_SIZE == 0);
    } // useful if many updates not many queries, may be better than st
    void upd(int ix, int w) { B[ix] += w; Bs[Bid[ix]] += w;} // O(1)
    int query(int l, int r){ // O(BLOCK_SIZE)
        int ans = 0;
        for (int i = l; i <= r;) { // [l, r]
            if (i + BLOCK_SIZE > r || (i % BLOCK_SIZE) != 0) ans += B[i
                ++];
            else { ans += Bs[Bid[i]]; i += BLOCK_SIZE;}
        }
        return ans;
    }
};
```

3.11 Chull Trick

```
typedef ll tc;
const tc is_query=-(1LL<<62); // special value for query
struct Line {
  tc m,b;
  mutable multiset<Line>::iterator it,end;
  const Line* succ(multiset<Line>::iterator it) const {
    return (++it==end? NULL : &*it);}
  bool operator<(const Line& rhs) const {
    if(rhs.b!=is_query)return m<rhs.m;
    const Line *s=succ(it);
    if(!s)return 0;
    return b-s->b<(s->m-m)*rhs.m;
  }
};
struct HullDynamic : public multiset<Line> { // for maximum
  bool bad(iterator y){
    iterator z=next(y);
    if(y==begin()){
      if(z==end())return false;
      return y->m==z->m&&y->b<=z->b;
    }
    iterator x=prev(y);
    if(z==end())return y->m==x->m&&y->b<=x->b;
    return 1.0*(x->b-y->b)*(z->m-y->m)>=1.0*(y->b-z->b)*(y->m-x->m);
  }//Take care of overflow!
  iterator next(iterator y){return ++y;}
  iterator prev(iterator y){return --y;}
  void add(tc m, tc b){
    iterator y=insert((Line){m,b});
    y->it=y;y->end=end();
    if(bad(y)){erase(y);return;}
    while(next(y)!=end()&&bad(next(y)))erase(next(y));
    while(y!=begin()&&bad(prev(y)))erase(prev(y));
  }
  tc eval(tc x){
    Line l=*lower_bound((Line){x,is_query});
    return l.m*x+l.b;
  }
};
```

4 Graph

4.1 Bellman Ford

```
struct Edge {int a, b, cost;};
vector<Edge> edges;
int solve(int s) // Source
{
    vector<int> d(n, INF);
    d[s] = 0;
    for (int i = 0; i < n - 1; ++i)
        for (Edge e : edges)
            if (d[e.a] < INF)
                d[e.b] = min(d[e.b], d[e.a] + e.cost);
}
```

4.2 SCC

```
vec<int> dfs_num(N, -1), dfs_low(N, -1), in_stack(N);
int dfs_count = 0;
int numSCC = 0;
stack<int> st;
void dfs(int u){
  dfs_low[u]=dfs_num[u]=dfs_count++;
  st.push(u);
  in_stack[u] = 1;
  for(int v: G[u]) {
    if (dfs_num[v] == -1) dfs(v);
```

```
    if (in_stack[v]) dfs_low[u] = min(dfs_low[u], dfs_low[v]);
  }
  if (dfs_num[u] == dfs_low[u]){
    numSCC ++;
    while(1){
      int v = st.top(); st.pop();
      in_stack[v] = 0;
      if (u == v) break;
    }
  }
}
```

4.3 Bipartite Matching Hopcroft-Karp - With Konig

```
mt19937 rng((int) chrono::steady_clock::now().time_since_epoch().count()
    );
struct hopcroft_karp {
  int n, m; // n is Left Partition Size, m is Right Partition Size
  vec<vec<int>> g;
  vec<int> dist, nxt, ma, mb;
  hopcroft_karp(int n_, int m_) : n(n_), m(m_), g(n),
    dist(n), nxt(n), ma(n, -1), mb(m, -1) {}
  void add(int a, int b) { g[a].pb(b); }
  bool dfs(int i) {
    for (int &id = nxt[i]; id < g[i].size(); id++) {
      int j = g[i][id];
      if (mb[j] == -1 or (dist[mb[j]] == dist[i]+1 and dfs(mb[j]))) {
        ma[i] = j, mb[j] = i;
        return true;
      }
    }
    return false;
  }
  bool bfs() {
    for (int i = 0; i < n; i++) dist[i] = n;
    queue<int> q;
    for (int i = 0; i < n; i++) if (ma[i] == -1) {
      dist[i] = 0;
      q.push(i);
    }
    bool rep = 0;
    while (q.size()) {
      int i = q.front(); q.pop();
      for (int j : g[i]) {
        if (mb[j] == -1) rep = 1;
        else if (dist[mb[j]] > dist[i] + 1) {
          dist[mb[j]] = dist[i] + 1;
          q.push(mb[j]);
        }
      }
    }
    return rep;
  }
  int matching() {
    int ret = 0;
    for (auto& i : g) shuffle(ALL(i), rng);
    while (bfs()) {
      for (int i = 0; i < n; i++) nxt[i] = 0;
      for (int i = 0; i < n; i++)
        if (ma[i] == -1 and dfs(i)) ret++;
    }
    return ret;
  }
  vec<int> cover[2]; // if cover[i][j] = 1 -> node i, j is part of cover
  int konig() {
    cover[0].assign(n,1); // n left size
    cover[1].assign(m,0); // m right size
    auto go = [&](auto&& me, int u) -> void {
      cover[0][u] = false;
      for (auto v : g[u]) if (!cover[1][v]) {
        cover[1][v] = true;
        me(me,mb[v]);
      }
    };
    L(u,0,n) if (ma[u] < 0) go(go,u);
    return size;
  }
};
```

4.4 Hungarian

```
using vi = vec<int>;
using vd = vec<ld>;
const ld INF = 1e100;    // Para max asignacion, INF = 0, y negar costos
bool zero(ld x) {return fabs(x) < 1e-9;}  // Para int/ll: return x==0;
vec<pii> ans; // Guarda las aristas usadas en el matching: [0..n)x[0..m)
```

```
struct Hungarian{
  int n; vec<vd> cs; vi vL, vR;
  Hungarian(int N, int M) : n(max(N,M)), cs(n,vd(n)), vL(n), vR(n){
    L(x, 0, N) L(y, 0, M) cs[x][y] = INF;
  }
  void set(int x, int y, ld c) { cs[x][y] = c; }
  ld assign(){
    int mat = 0; vd ds(n), u(n), v(n); vi dad(n), sn(n);
    L(i, 0, n) u[i] = *min_element(ALL(cs[i]));
    L(j, 0, n){
      v[j] = cs[0][j]-u[0];
      L(i, 1, n) v[j] = min(v[j], cs[i][j] - u[i]);
    }
    vL = vR = vi(n, -1);
    L(i,0, n) L(j, 0, n) if(vR[j] == -1 and zero(cs[i][j] - u[i] - v[j])
        ){
      vL[i] = j; vR[j] = i; mat++; break;
    }
    for(; mat < n; mat ++){
      int s = 0, j = 0, i;
      while(vL[s] != -1) s++;
      fill(ALL(dad), -1); fill(ALL(sn), 0);
      L(k, 0, n) ds[k] = cs[s][k]-u[s]-v[k];
      while(true){
        j = -1;
        L(k, 0, n) if(!sn[k] and (j == -1 or ds[k] < ds[j])) j = k;
        sn[j] = 1; i = vR[j];
        if(i == -1) break;
        L(k, 0, n) if(!sn[k]){
          auto new_ds = ds[j] + cs[i][k] - u[i]-v[k];
          if(ds[k] > new_ds) ds[k]=new_ds, dad[k]=j;
        }
      }
      L(k, 0, n) if(k!=j and sn[k]){
        auto w = ds[k]-ds[j]; v[k] += w, u[vR[k]] -= w;
      }
      u[s] += ds[j];
      while(dad[j] >= 0){ int d = dad[j]; vR[j] = vR[d]; vL[vR[j]] = j;
          j = d; }
      vR[j] = s; vL[s] = j;
    }
    ld value = 0; L(i, 0, n) value += cs[i][vL[i]], ans.pb({i, vL[i]});
    return value;
  }
};
```

4.5 Flow - Dinics

```
const int oo = (int)1e9;
struct Dinic {
  bool scaling = false; // com scaling -> O(nm log(MAXCAP)),
  int lim;              // com constante alta
  struct edge {
    int to, cap, rev, flow;
    bool res;
    edge(int to_, int cap_, int rev_, bool res_)
      : to(to_), cap(cap_), rev(rev_), flow(0), res(res_) {}
  };
  vec<vec<edge>> g;
  vec<int> lev, beg;
  ll F;
  Dinic(int n) : g(n), F(0) {}
  void add(int a, int b, int c) {
    g[a].emplace_back(b, c, g[b].size(), false);
    g[b].emplace_back(a, 0, g[a].size()-1, true);
  }
  bool bfs(int s, int t) {
    lev = vector<int>(g.size(), -1); lev[s] = 0;
    beg = vector<int>(g.size(), 0);
    queue<int> q; q.push(s);
    while (q.size()) {
      int u = q.front(); q.pop();
      for (auto& i : g[u]) {
        if (lev[i.to] != -1 or (i.flow == i.cap)) continue;
        if (scaling and i.cap - i.flow < lim) continue;
        lev[i.to] = lev[u] + 1;
        q.push(i.to);
      }
    }
    return lev[t] != -1;
  }
  int dfs(int v, int s, int f = oo) {
    if (!f or v == s) return f;
    for (int& i = beg[v]; i < g[v].size(); i++) {
      auto& e = g[v][i];
      if (lev[e.to] != lev[v] + 1) continue;
```

```
      int foi = dfs(e.to, s, min(f, e.cap - e.flow));
      if (!foi) continue;
      e.flow += foi, g[e.to][e.rev].flow -= foi;
      return foi;
    }
    return 0;
  }
  ll max_flow(int s, int t) {
    for (lim = scaling ? (1<<30) : 1; lim; lim /= 2)
      while (bfs(s, t)) while (int ff = dfs(s, t)) F += ff;
    return F;
  }
};
vec<pair<int, int>> get_cut(Dinic& g, int s, int t) {
  g.max_flow(s, t);
  vec<pair<int, int>> cut;
  vec<int> vis(g.g.size(), 0), st = {s};
  vis[s] = 1;
  while (st.size()) {
    int u = st.back(); st.pop_back();
    for (auto e : g.g[u]) if (!vis[e.to] and e.flow < e.cap)
      vis[e.to] = 1, st.push_back(e.to);
  }
  for (int i = 0; i < g.g.size(); i++) for (auto e : g.g[i])
    if (vis[i] and !vis[e.to] and !e.res) cut.emplace_back(i, e.to);
  return cut;
}
```

4.6 Flow - MinCostMaxFlow

```
// O(nm + f * m log n)
// const ll oo = (ll)1e18;
template<typename T> struct mcmf {
  struct edge {
    int to, rev, flow, cap; // para, id da reversa, fluxo, capacidade
    bool res; // se eh reversa
    T cost; // custo da unidade de fluxo
    edge() : to(0), rev(0), flow(0), cap(0), cost(0), res(false) {}
    edge(int to_, int rev_, int flow_, int cap_, T cost_, bool res_)
      : to(to_), rev(rev_), flow(flow_), cap(cap_), res(res_), cost(
          cost_) {}
  };
  vec<vec<edge>> g;
  vec<int> par_idx, par;
  T inf;
  vec<T> dist;
  mcmf(int n) : g(n), par_idx(n), par(n), inf(numeric_limits<T>::max()
      /3) {}
  void add(int u, int v, int w, T cost) { // de u pra v com cap w e
      custo cost
    edge a = edge(v, g[v].size(), 0, w, cost, false);
    edge b = edge(u, g[u].size(), 0, 0, -cost, true);
    g[u].push_back(a);
    g[v].push_back(b);
  }
  vec<T> spfa(int s) { // nao precisa se nao tiver custo negativo
    deque<int> q;
    vec<bool> is_inside(g.size(), 0);
    dist = vec<T>(g.size(), inf);
    dist[s] = 0;
    q.push_back(s);
    is_inside[s] = true;
    while (!q.empty()) {
      int v = q.front();
      q.pop_front();
      is_inside[v] = false;
      for (int i = 0; i < g[v].size(); i++) {
        auto [to, rev, flow, cap, res, cost] = g[v][i];
        if (flow < cap and dist[v] + cost < dist[to]) {
          dist[to] = dist[v] + cost;

          if (is_inside[to]) continue;
          if (!q.empty() and dist[to] > dist[q.front()]) q.push_back(to)
              ;
          else q.push_front(to);
          is_inside[to] = true;
        }
      }
    }
    return dist;
  }
  bool dijkstra(int s, int t, vec<T>& pot) {
    priority_queue<pair<T, int>, vec<pair<T, int>>, greater<>> q;
    dist = vec<T>(g.size(), inf);
    dist[s] = 0;
    q.emplace(0, s);
```

```
    while (q.size()) {
      auto [d, v] = q.top();
      q.pop();
      if (dist[v] < d) continue;
      for (int i = 0; i < g[v].size(); i++) {
        auto [to, rev, flow, cap, res, cost] = g[v][i];
        cost += pot[v] - pot[to];
        if (flow < cap and dist[v] + cost < dist[to]) {
          dist[to] = dist[v] + cost;
          q.emplace(dist[to], to);
          par_idx[to] = i, par[to] = v;
        }
      }
    }
    return dist[t] < inf;
  }
  pair<int, T> min_cost_flow(int s, int t, int flow = (int)1e9) {
    vec<T> pot(g.size(), 0);
    pot = spfa(s); // mudar algoritmo de caminho minimo aqui
    int f = 0;
    T ret = 0;
    while (f < flow and dijkstra(s, t, pot)) {
      for (int i = 0; i < g.size(); i++)
        if (dist[i] < inf) pot[i] += dist[i];
      int mn_flow = flow - f, u = t;
      while (u != s){
        mn_flow = min(mn_flow,
          g[par[u]][par_idx[u]].cap - g[par[u]][par_idx[u]].flow);
        u = par[u];
      }
      ret += pot[t] * mn_flow;
      u = t;
      while (u != s) {
        g[par[u]][par_idx[u]].flow += mn_flow;
        g[u][g[par[u]][par_idx[u]].rev].flow -= mn_flow;
        u = par[u];
      }
      f += mn_flow;
    }
    return make_pair(f, ret);
  }
  // Opcional: retorna as arestas originais por onde passa flow = cap
  vec<pair<int,int>> recover() {
    vec<pair<int,int>> used;
    for (int i = 0; i < g.size(); i++) for (edge e : g[i])
      if(e.flow == e.cap && !e.res) used.push_back({i, e.to});
    return used;
  }
};
```

4.7 2 Sat

```
struct TwoSat {
    int n, v_n;
    vec<bool> vis, assign;
    vec<int> order, comp;
    vec<vec<int>> g, g_t;
    TwoSat(int n_): n(n_), v_n(2 * n_), vis(v_n) , assign(n_),  comp(v_n , - 1), g(v_n), g_t(v_n) {
        order.reserve(v_n);
    }
    void add_disj(int a, bool na, int b, bool nb) { // negated_a, negated_b
        a = 2 * a ^ na;
        b = 2 * b ^ nb;
        int neg_a = a ^ 1;
        int neg_b = b ^ 1;
        g[neg_a].pb(b);
        g[neg_b].pb(a);
        g_t[a].pb(neg_b);
        g_t[b].pb(neg_a);
    }
    void dfs1(int u){
        vis[u] = 1;
        for (int v: g[u]) if (!vis[v]) dfs1(v);
        order.pb(u);
    }
    void dfs2(int u, int cc) {
        comp[u] = cc;
        for (int v: g_t[u]) if (comp[v] == -1) dfs2(v, cc);
    }
    bool solve() {
        order.clear();
        vis.assign(v_n, 0);
        L(i,0, v_n) if (!vis[i]) dfs1(i);
        comp.assign(v_n, - 1);
```

```
        int cc = 0;
        L(i, 0, v_n) {
            int v = order[v_n - 1 - i];
            if (comp[v] == -1) dfs2(v, cc ++);
        }
        assign.assign(n, false);
        for (int i = 0;i < v_n; i += 2) {
            if (comp[i] == comp[i+1]) return false;
            assign[i / 2] = comp[i] > comp[i + 1];
        }
        return true;
    }
};
```

4.8 Euler Tour

```
// Directed version (uncomment commented code for undirected)
struct edge {
    int y;
//    list<edge>::iterator rev;
    edge(int y):y(y){}
};
list<edge> g[N];
void add_edge(int a, int b){
    g[a].push_front(edge(b));//auto ia=g[a].begin();
//    g[b].push_front(edge(a));auto ib=g[b].begin();
//    ia->rev=ib;ib->rev=ia;
}
vec<int> p;
void go(int x){
    while(g[x].size()){
        int y=g[x].front().y;
        //g[y].erase(g[x].front().rev);
        g[x].pop_front();
        go(y);
    }
    p.push_back(x);
}
vec<int> get_path(int x){ // get a path that begins in x
// check that a path exists from x before calling to get_path!
    p.clear();go(x);reverse(p.begin(),p.end());
    return p;
}
```

5 Trees

5.1 Heavy Light Decomposition

```
int ans[N], par[N], depth[N], head[N], pos[N];
vec<int> heavy(N, - 1);
int t = 0;
vec<int> g[N];
int dfs(int u) {
    int size = 1;
    int max_size = 0;
    for (int v: g[u]) if (v != par[u]) {
        par[v] = u;
        depth[v] = depth[u] + 1;
        int cur_size = dfs(v);
        size += cur_size;
        if (cur_size > max_size) {
            max_size = cur_size;
            heavy[u] = v;
        }
    }
    return size;
}
void decompose(int u, int h){
    head[u] = h;
    pos[u] = t ++;
    if (heavy[u] != -1){ decompose(heavy[u], h); }
    for (int v: G[u]) if (v != par[u] && v != heavy[u]) {
        decompose(v, v);
    }
}
int query(int a, int b) {
    int resp = -1;
    for (; head[a] != head[b]; b = par[head[b]]){ // Subi todo el heavy path y a su padre // Next
        if (depth[head[a]] > depth[head[b]]) swap(a, b);
        resp = max(resp, st.query(pos[head[b]], pos[b])); // pos[head[b]] < pos[b]
    }
    if (depth[a] > depth[b]) swap(a, b); // Una vez misma path(head) entonces es una query [a,b]
    resp = max(resp, st.query(pos[a], pos[b]));
    return resp;
```

```
}
dfs(root);
decompose(root, root);
```

5.2 Centroid

```
int sz[N];
bool removed[N];
int getSize(int u, int p){
    sz[u] = 1;
    for(int v: G[u]) if (v != p && !removed[v]){
        sz[u] += getSize(v, u);
    }
    return sz[u];
}
int centroid(int u, int p, int tz){
    for (int v: g[u])
        if (v != p && !removed[v] && sz[v] * 2 > tz) return centroid(v,
            u, tz);
    return u;
}
int build(int u){
    int c = centroid(u, -1, getSize(u, -1));
    removed[c] = 1;
    for (int v: G[c]) if (!removed[v]) { build(v); }
    return c;
}
```

5.3 LCA - Binary exponentiation

```
vec<int> g[N];
int K; // K should be (1<<K) > n
int jump[20][N];
int depth[N];

void dfs(int u, int p){
    for (int v: g[u]) if (v != p) {
        jump[0][v] = u;
        L(i, 1, K + 1) {
            jump[i][v] = -1;
            if (jump[i - 1][v] != -1) {
                jump[i][v] = jump[i - 1][jump[i - 1][v]];
            }
        }
        depth[v] = depth[u] + 1;
        dfs(v, u);
    }
}

int LCA(int u, int v){
    if (depth[u] < depth[v]) swap(u, v); // Make u the deepest
    for (int i = K; i >= 0; i --){ // make them same depth
        if (jump[i][u] != -1 && depth[jump[i][u]] >= depth[v]){
            u = jump[i][u];
        }
    }
    if (u == v) return u; // u is parent of v
    for (int i = K; i>= 0; i --){
        if (jump[i][u] != jump[i][v] && jump[i][u] != -1 && jump[i][v]
            != -1){
            u = jump[i][u];
            v = jump[i][v];
        }
    }
    return jump[0][u];
}
```

5.4 LCA - Const Time

```
struct LCA {
    vec<int> depth, in, euler;
    vec<vec<int>> g, st;
    int K, n;
    inline int Min(int i, int j) {return depth[i] <= depth[j] ? i : j;}
    void dfs(int u, int p) {
        in[u] = SZ(euler);
        euler.pb(u);
        for (int v: g[u]) if (v != p){
            depth[v] = depth[u] + 1;
            dfs(v, u);
            euler.pb(u);
        }
    }
    LCA(int n_): depth(n_), g(vec<vec<int>>(n_)), K(0), n(n_), in(n_) {
        euler.reserve(2 * n); }
    void add_edge(int u, int v) {g[u].pb(v);}
    void build(int root){
```

```
        dfs(root, -1);
        int ln = SZ(euler);
        while((1<<K)<=ln)K++;
        st = vec<vec<int>> (K, vec<int>(ln));
        L(i,0,ln) st[0][i] = euler[i];
        for (int i = 1; (1 << i) <= ln; i ++) {
            for (int j = 0; j + (1<<i) <= ln; j ++) {
                st[i][j] = Min(st[i-1][j], st[i-1][j + (1<<(i-1))]);
            }
        }
    }
    int get(int u, int v) {
        int su = in[u];
        int sv = in[v];
        if (sv < su) swap(sv, su);
        int bit = log2(sv - su + 1);
        return Min(st[bit][su], st[bit][sv - (1<<bit) + 1]);
    }
};
```

6 Dynamic Programming

6.1 Knapsack

```
int knapsack(vector<int>& values, vector<int>& weights, int W) {
    int n = values.size();
    vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));

    for(int i = 1; i <= n; i++) {
        for(int w = 0; w <= W; w++) {
            if(weights[i-1] <= w) {
                dp[i][w] = max(dp[i-1][w],
                               dp[i-1][w-weights[i-1]] + values[i-1]);
            } else {
                dp[i][w] = dp[i-1][w];
            }
        }
    }
    return dp[n][W];
}
```

6.2 LIS

```
vector<int> getLIS(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n + 1, INT_MAX);  // dp[i] = smallest value that ends an LIS of length i
    vector<int> len(n);              // Length of LIS ending at each position
    dp[0] = INT_MIN;
    for(int i = 0; i < n; i++) {
        int j = upper_bound(dp.begin(), dp.end(), arr[i]) - dp.begin();
        dp[j] = arr[i];
        len[i] = j;
    }
    // Find maxLen and reconstruct sequence
    int maxLen = 0;
    for(int i = n-1; i >= 0; i--) maxLen = max(maxLen, len[i]);
    vector<int> lis;
    for(int i = n-1, currLen = maxLen; i >= 0; i--) {
        if(len[i] == currLen) {
            lis.push_back(arr[i]);
            currLen--;
        }
    }
    reverse(lis.begin(), lis.end());
    return lis;
}
```

6.3 Edit Distance

```
int editDistance(string& s1, string& s2) {
    int n = s1.length(), m = s2.length();
    vector<vector<int>> dp(n + 1, vector<int>(m + 1));
    for(int i = 0; i <= n; i++) dp[i][0] = i;
    for(int j = 0; j <= m; j++) dp[0][j] = j;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= m; j++) {
            if(s1[i-1] == s2[j-1]) {
                dp[i][j] = dp[i-1][j-1];
            } else {
                dp[i][j] = 1 + min({dp[i-1][j],    // deletion
                                    dp[i][j-1],     // insertion
                                    dp[i-1][j-1]}); // replacement
            }
        }
```

```
    }
    return dp[n][m];
}
```

6.4 Kadane

```
pair<int, pair<int,int>> kadane(vector<int>& arr) {
    int maxSoFar = arr[0], maxEndingHere = arr[0];
    int start = 0, end = 0, s = 0;

    for(int i = 1; i < arr.size(); i++) {
        if(maxEndingHere + arr[i] < arr[i]) {
            maxEndingHere = arr[i];
            s = i;
        } else {
            maxEndingHere += arr[i];
        }

        if(maxEndingHere > maxSoFar) {
            maxSoFar = maxEndingHere;
            start = s;
            end = i;
        }
    }
    return {maxSoFar, {start, end}}; // max, l, r
}
```

7 Strings

7.1 Hashing

```
static constexpr ll ms[] = {1'000'000'007, 1'000'000'403};
static constexpr ll b = 500'000'000;
struct StrHash { // Hash polinomial con exponentes decrecientes.
  vec<ll> hs[2], bs[2];
  StrHash(string const& s) {
    int n = SZ(s);
    L(k, 0, 2) {
      hs[k].resize(n+1), bs[k].resize(n+1, 1);
      L(i, 0, n) {
        hs[k][i+1] = (hs[k][i] * b + s[i]) % ms[k];
        bs[k][i+1] =  bs[k][i] * b         % ms[k];
      }
    }
  }
  ll get(int idx, int len) const { // Hashes en `s[idx, idx+len)`.
    ll h[2];
    L(k, 0, 2) {
      h[k] = hs[k][idx+len] - hs[k][idx] * bs[k][len] % ms[k];
      if (h[k] < 0) h[k] += ms[k];
    }
    return (h[0] << 32) | h[1];
  }
};

pll union_hash(vec<pll> hs, vec<ll> lens){ //use arrays makes it slower
  ll len = 0;
  for(int i = hs.size()-1; i > 0; i--){
    len += lens[i];
    pll& [l1, l2] = hs[i];
    pll& [r1, r2] = hs[i-1];
    l1 = ((l1 * binpow(b, len, ms[0])) % ms[0] + r1) % ms[0];
    l2 = ((l2 * binpow(b, len, ms[1])) % ms[1] + r2) % ms[1];
  }

  return hs[0];
}
```

7.2 Trie

```
struct Trie {
    map<char, int> ch;
    bool eee;
    Trie(): eee(0) {}
};
vec<Trie> t;
void initTrie(){t.clear();t.pb(Trie());}
void insert(string &word) {
    int v = 0;
    for(char c : word) {
        if(!t[v].ch[c]) {
            t[v].ch[c] = SZ(t);
            t.pb(Trie());
        }
        v = t[v].ch[c];
    }
```

```
    t[v].eee = 1;
}
```

7.3 KMP

```
vec<int> KMP(const string &s){
    int n = SZ(s); vec<int> pi(n);
    L(i,1,n){
        int j = pi[i - 1];
        while(j>0&&s[i]!=s[j]) j = pi[j-1];
        if (s[i]==s[j])j++;
        pi[i]=j;
    }
    return pi;
}
```

7.4 LPS

```
vec<int> getLps(string pat){ //geek4geeks implementatio with some changes
  vec<int> lps(pat.length(), 0);
  int len = 0;
  int i = 1;
  lps[0] = 0;
  while(i < pat.length()){
    if(pat[i] == pat[len]){
      len++;
      lps[i] = len;
      i++;
    }
    else //pat[i] != pat[len]
    {
      lps[i] = 0;
      i++;
    }
  }
  return lps;
}
```

7.5 Z-FUNCTION

```
template<class Char=char>vec<int> zfun(const basic_string<Char>& w) {
  int n = SZ(w), l = 0, r = 0; vec<int> z(n);
  z[0] = w.length();
  L(i, 1, n) {
    if (i <= r) {z[i] = min(r - i + 1, z[i - l]);}
    while (i + z[i] < n && w[z[i]] == w[i + z[i]]) {++z[i];}
    if (i + z[i] - 1 > r) {l = i, r = i + z[i] - 1;}
  }
  return z;
}
```

7.6 Manacher

```
struct Manacher {
  vec<int> p;
  Manacher(string const& s) {
    int n = SZ(s), m = 2*n+1, l = -1, r = 1;
    vec<char> t(m); L(i, 0, n) t[2*i+1] = s[i];
    p.resize(m); L(i, 1, m) {
      if (i < r) p[i] = min(r-i, p[l+r-i]);
      while (p[i] <= i && i < m-p[i] && t[i-p[i]] == t[i+p[i]]) ++p[i];
      if (i+p[i] > r) l = i-p[i], r = i+p[i];
    }
  } // Retorna palindromos de la forma {comienzo, largo}.
  pii at(int i) const {int k = p[i]-1; return pair{i/2-k/2, k};}
  pii odd(int i) const {return at(2*i+1);} // Mayor centrado en s[i].
  pii even(int i) const {return at(2*i);} // Mayor centrado en s[i-1,i].
};
```

7.7 Aho-Corasick

```
bool vis[N], r[N];
struct ACvertex {
  map<char,int> next,go;
  int p,link;
  char pch;
  vec<int> leaf;
  ACACvertex(int p=-1, char pch=-1):p(p),pch(pch),link(-1){}
};
vec<ACvertex> t;
void aho_init(){ //do not forget!!
  t.clear();t.pb(ACvertex());
}
void add_string(string &s, int id){
  int v=0;
  for(char c:s){
    if(!t[v].next.count(c)){
```

```
      t[v].next[c]=t.size();
      t.pb(ACvertex(v,c));
    }
    v=t[v].next[c];
  }
  t[v].leaf.pb(id);
}
int go(int v, char c);
int get_link(int v){ // Failure link
  if(t[v].link<0)
    if(!v||!t[v].p)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 c){ // state = go(state, ch) this state is ACvertex id
  if(!t[v].go.count(c))
    if(t[v].next.count(c))t[v].go[c]=t[v].next[c];
    else t[v].go[c]=v==0?0:go(get_link(v),c);
  return t[v].go[c];
}
void proc(int x){
    if (x == - 1|| vis[x]) return;
    vis[x] = 1;
    L(i,0,SZ(t[x].leaf)) r[t[x].leaf[i]] = 1;
    proc(get_link(x));
}
```

7.8 Suffix-Array

```
#define RB(x) ((x) < n ? r[x] : 0)
void csort(vec<int>& sa, vec<int>& r, int k) {
  int n = SZ(sa);
  vec<int> f(max(255, n)), t(n);
  L(i,0, n) ++f[RB(i+k)];
  int sum = 0;
  L(i,0, max(255, n)) f[i] = (sum += f[i]) - f[i];
  L(i,0, n) t[f[RB(sa[i]+k)]++] = sa[i];
  sa = t;
}
vec<int> compute_sa(string& s){ // O(n*log2(n))
  int n = SZ(s) + 1, rank;
  vec<int> sa(n), r(n), t(n);
  iota(all(sa), 0);
  L(i,0, n) r[i] = s[i];
  for (int k = 1; k < n; k *= 2) {
    csort(sa, r, k), csort(sa, r, 0);
    t[sa[0]] = rank = 0;
    L(i, 1, n) {
      if(r[sa[i]] != r[sa[i-1]] || RB(sa[i]+k) != RB(sa[i-1]+k)) ++rank;
      t[sa[i]] = rank;
    }
    r = t;
    if (r[sa[n-1]] == n-1) break;
  }
  return sa; // sa[i] = i-th suffix of s in lexicographical order
}
vec<int> compute_lcp(string& s, vec<int>& sa){
  int n = SZ(s) + 1, K = 0;
  vec<int> lcp(n), plcp(n), phi(n);
  phi[sa[0]] = -1;
  L(i, 1, n) phi[sa[i]] = sa[i-1];
  L(i,0,n) {
    if (phi[i] < 0) { plcp[i] = 0; continue; }
    while(s[i+K] == s[phi[i]+K]) ++K;
    plcp[i] = K;
    K = max(K - 1, 0);
  }
  L(i,0, n) lcp[i] = plcp[sa[i]];
  return lcp; // lcp[i] = longest common prefix between sa[i-1] and sa[i]
}
```

8 Math

8.1 Euclidean Extended

```
ll extendedGCD(ll a, ll b, ll &x, ll &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    ll x1, y1;
    ll gcd = extendedGCD(b, a % b, x1, y1);
    x = y1;
```

```cpp
    y = x1 - (a / b) * y1;
    return gcd;
}

bool findSolutionWithConstraints(ll a, ll b, ll c, ll x_min, ll y_min, ll &x, ll &y) {
    ll g = extendedGCD(a, b, x, y);

    if (c % g != 0) return false;

    x *= c / g;
    y *= c / g;

    // Ajustamos las variables a/g y b/g para mover las soluciones
    a /= g;
    b /= g;

    if (x < x_min) {
        ll k = (x_min - x + b - 1) / b;  // Redondeo hacia arriba
        x += k * b;
        y -= k * a;
    } else if (x > x_min) {
        ll k = (x - x_min) / b;
        x -= k * b;
        y += k * a;
    }

    if (y < y_min) {
        ll k = (y_min - y + a - 1) / a;  // Redondeo hacia arriba
        x += k * b;
        y -= k * a;
    } else if (y > y_min) {
        ll k = (y - y_min) / a;
        x -= k * b;
        y += k * a;
    }

    return x >= x_min && y >= y_min;
}
```

8.2 Euler Totient

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;


vector<ll> compute_totients(ll n) {
    vector<ll> phi(n + 1);
    for (ll i = 0; i <= n; i++) {
        phi[i] = i;
    }

    for (ll i = 2; i <= n; i++) {
        if (phi[i] == i) { // i es primo
            for (ll j = i; j <= n; j += i) {
                phi[j] = phi[j] * (i - 1) / i;
            }
        }
    }

    return phi;
}
```

8.3 Josephus

```cpp
#include <iostream>
using namespace std;

typedef long long ll;

ll josephus_iterative(ll n, ll k) {
    ll result = 0;
    for (ll i = 2; i <= n; ++i) {
        result = (result + k) % i;
    }
    return result;
}


ll josephus_recursive(ll n, ll k) {

    if (n == 1)
        return 0;

    return (josephus_recursive(n - 1, k) + k) % n;
```

```cpp
}


ll josephus_power_of_2(ll n) {

    ll power = 1;
    while (power <= n) {
        power <<= 1;
    }
    power >>= 1;


    return 2 * (n - power);
}
```

8.4 Mobius

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;


vector<ll> compute_mobius(ll n) {
    vector<ll> mu(n + 1, 1);
    vector<bool> is_prime(n + 1, true);

    for (ll i = 2; i <= n; i++) {
        if (is_prime[i]) { // i es un primo
            for (ll j = i; j <= n; j += i) {
                mu[j] *= -1; // Multiplicamos por -1 para cada primo
                is_prime[j] = false;
            }
            for (ll j = i * i; j <= n; j += i * i) {
                mu[j] = 0; // Si tiene un cuadrado de un primo, se pone
                    en 0
            }
        }
    }

    return mu;
}


ll mobius(ll x) {
    ll count = 0;
    for (ll i = 2; i * i <= x; i++) {
        if (x % (i * i) == 0)
            return 0;
        if (x % i == 0) {
            count++;
            x /= i;
        }
    }

    if (x > 1) count++;

    return (count % 2 == 0) ? 1 : -1;
}
```

8.5 NTT

```cpp
#include <bits/stdc++.h>
using namespace std;
using cd = complex<double>;
typedef long long ll;
const ll mod = 998244353;
const ll root = 31;
const ll root_1 = inverse(root, mod);
const ll root_pw = 1 << 23;

ll inverse(ll a, ll m) {
    ll res = 1, exp = m - 2;
    while (exp) {
        if (exp % 2 == 1) res = (1LL * res * a) % m;
        a = (1LL * a * a) % m;
        exp /= 2;
    }
    return res;
}

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

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

```cpp
            j ^= bit;
        j ^= bit;

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

    for (int len = 2; len <= n; len <<= 1) {
        int wlen = invert ? root_1 : root;
        for (int i = len; i < root_pw; i <<= 1)
            wlen = (int)(1LL * wlen * wlen % mod);

        for (int i = 0; i < n; i += len) {
            int w = 1;
            for (int j = 0; j < len / 2; j++) {
                int u = a[i+j], v = (int)(1LL * a[i+j+len/2] * w % mod);
                a[i+j] = u + v < mod ? u + v : u + v - mod;
                a[i+j+len/2] = u - v >= 0 ? u - v : u - v + mod;
                w = (int)(1LL * w * wlen % mod);
            }
        }
    }

    if (invert) {
        int n_1 = inverse(n, mod);
        for (auto & x : a)
            x = (int)(1LL * x * n_1 % mod);
    }
}

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

    ntt(fa, false);
    ntt(fb, false);
    for (ll i = 0; i < n; i++)
        fa[i] = (fa[i] * fb[i]) % mod;
    ntt(fa, true);

    vector<ll> result(n);
    for (ll i = 0; i < n; i++)
        result[i] = fa[i];
    return result;
}
```

8.6 FFT

```cpp
typedef long long ll;
typedef complex<double> C;
typedef vector<double> vd;
typedef vector<ll> vll;
const double PI = acos(-1);

void fft(vector<C>& a) {
    int n = a.size(), L = 31 - __builtin_clz(n);
    static vector<C> R(2, 1);
    static vector<C> rt(2, 1);
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n); rt.resize(n);
        auto x = polar(1.0, PI / k);
        for (int i = k; i < 2 * k; i++)
            rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
    }
    vector<int> rev(n);
    for (int i = 0; i < n; i++) rev[i] = (rev[i / 2] | (i & 1) << L) /
        2;
    for (int i = 0; i < n; i++) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) for (int j = 0; j < k; j++) {
            auto x = (double*)&rt[j + k], y = (double*)&a[i + j + k];
            C z(x[0] * y[0] - x[1] * y[1], x[0] * y[1] + x[1] * y[0]);
            a[i + j + k] = a[i + j] - z;
            a[i + j] += z;
        }
}

vll multiply(const vll& a, const vll& b) {
    if (a.empty() || b.empty()) return {};
    vd fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int L = 32 - __builtin_clz(fa.size() + fb.size() - 1), n = 1 << L;
    vector<C> in(n), out(n);
```

```cpp

    for (int i = 0; i < a.size(); i++) in[i] = C(fa[i], 0);
    for (int i = 0; i < b.size(); i++) in[i].imag(fb[i]);

    fft(in);
    for (C& x : in) x *= x;
    for (int i = 0; i < n; i++) out[i] = in[-i & (n - 1)] - conj(in[i]);
          // Corregido aqui
    fft(out);

    vll res(a.size() + b.size() - 1);
    for (int i = 0; i < res.size(); i++) {
        res[i] = llround(imag(out[i]) / (4 * n));
    }
    return res;
}
```

8.7 Rho

```cpp
ll mul(ll a, ll b, ll mod) {
    return (__int128)a * b % mod;
}

ll power(ll a, ll b, ll mod) {
    ll res = 1;
    while (b) {
        if (b & 1) res = mul(res, a, mod);
        a = mul(a, a, mod);
        b >>= 1;
    }
    return res;
}

bool isPrime(ll n) {
    if (n < 2) return false;
    for (ll p : {2, 3, 5, 7, 11, 13, 17, 19, 23}) {
        if (n % p == 0) return n == p;
    }
    ll d = n - 1, s = 0;
    while ((d & 1) == 0) d >>= 1, ++s;
    for (ll a : {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) {
        if (a % n == 0) continue;
        ll x = power(a, d, n);
        if (x == 1 || x == n - 1) continue;
        bool ok = false;
        for (int i = 0; i < s; ++i) {
            x = mul(x, x, n);
            if (x == n - 1) { ok = true; break; }
        }
        if (!ok) return false;
    }
    return true;
}

ll rho(ll n) {
    if (n % 2 == 0) return 2;
    while (true) {
        ll c = rand() % (n - 1) + 1;
        ll x = 2, y = 2, d = 1;
        while (d == 1) {
            x = (mul(x, x, n) + c) % n;
            y = (mul(y, y, n) + c) % n;
            y = (mul(y, y, n) + c) % n;
            d = std::gcd((x > y ? x - y : y - x), n);
        }
        if (d != n) return d;
    }
}

void fact(ll n, std::map<ll, int>& f) {
    if (n == 1) return;
    if (isPrime(n)) { f[n]++; return; }
    ll d = rho(n);
    if (d == n) {
        f[n]++;
        return;
    }
    fact(d, f);
    fact(n / d, f);
}
```

8.8 Get Divisors

```cpp
vector<ll> getDivisors(const map<ll, int>& f) {
    vector<ll> divisors = { 1 };
    for (auto [p, e] : f) {
```

```
        vector<ll> next;
        ll pe = 1;
        for (int i = 0; i <= e; i++) {
            for (ll d : divisors)
                next.push_back(d * pe);
            pe *= p;
        }
        divisors.swap(next);
    }
    sort(divisors.begin(), divisors.end());
    return divisors;
}
```

8.9 Simpson

```
ld simpsonRule(function<ld(ld)> f, ld a, ld b, int n) {
    // Asegurarse de que n sea par
    if (n % 2 != 0) {
        n++;
    }
    ld h = (b - a) / n;
    ld s = f(a) + f(b);

    // Suma de terminos interiores con los factores apropiados
    for (int i = 1; i < n; i++) {
        ld x = a + i * h;
        s += (i % 2 == 1 ? 4.0L : 2.0L) * f(x);
    }
    // Multiplica por h/3
    return (h / 3.0L) * s;
}
// Ejemplo: integrar la funcion x^2 entre 0 y 3
auto f = [&](ld x){ return x * x; };
ld a = 0.0L, b = 3.0L;
int n = 1000; // numero de subintervalos
ld resultado = simpsonRule(f, a, b, n);
```

8.10 Simplex

```
pair<ld, vec<ld>> simplex(vec<vec<ld>> A, vec<ld> b, vec<ld> c) {
    const ld EPS = (ld)1e-9;
    int n = SZ(b), m = SZ(c);

    vec<int> X(m), Y(n);
    L(j, 0, m) X[j] = j;
    L(i, 0, n) Y[i] = m + i;

    ld z = 0;

    auto pivot = [&](int x, int y) {
        swap(X[y], Y[x]);

        ld inv = (ld)1 / A[x][y];
        b[x] *= inv;
        L(j, 0, m) if (j != y) A[x][j] *= inv;
        A[x][y] = inv;

        L(i, 0, n) if (i != x && fabsl(A[i][y]) > EPS) {
            ld coef = A[i][y];
            b[i] -= coef * b[x];
            L(j, 0, m) if (j != y) A[i][j] -= coef * A[x][j];
            A[i][y] = -coef * A[x][y];
        }

        z += c[y] * b[x];
        L(j, 0, m) if (j != y) c[j] -= c[y] * A[x][j];
        c[y] = -c[y] * A[x][y];
        };

    while (true) {
        int x = -1, y = -1;
        ld mn = -EPS;
        L(i, 0, n) if (b[i] < mn) { mn = b[i]; x = i; }
        if (x < 0) break;
        L(j, 0, m) if (A[x][j] < -EPS) { y = j; break; }
        if (y < 0) {
            return { numeric_limits<ld>::quiet_NaN(), {} };
        }
        pivot(x, y);
    }

    while (true) {
        int y = -1, x = -1;
        ld mx = EPS;
        L(j, 0, m) if (c[j] > mx) { mx = c[j]; y = j; }
        if (y < 0) break;

```

```
        ld best = numeric_limits<ld>::infinity();
        L(i, 0, n) if (A[i][y] > EPS) {
            ld val = b[i] / A[i][y];
            if (val < best) { best = val; x = i; }
        }
        if (x < 0) {
            return { numeric_limits<ld>::infinity(), {} };
        }
        pivot(x, y);
    }

    vec<ld> sol(m, 0);
    L(i, 0, n) if (Y[i] < m) sol[Y[i]] = b[i];
    return { z, sol };
}
```

9 Geometry

9.1 Convex Hull

```
typedef pair<ll, ll> Point;
ll cross_product(Point O, Point A, Point B) {
    return (A.first - O.first) * (B.second - O.second) - (A.second - O.
        second) * (B.first - O.first);
}
vector<Point> convex_hull(vector<Point>& points) {
    sort(points.begin(), points.end());
    points.erase(unique(points.begin(), points.end()), points.end());
    vector<Point> hull;
    // Parte inferior
    for (const auto& p : points) {
        while (hull.size() >= 2 && cross_product(hull[hull.size() - 2],
            hull[hull.size() - 1], p) < 0)
            hull.pop_back();
        if (hull.empty() || hull.back() != p) {
            hull.push_back(p);
        }
    }
    // Parte superior
    int t = hull.size() + 1;
    for (int i = points.size() - 1; i >= 0; --i) {
        while (hull.size() >= t && cross_product(hull[hull.size() - 2],
            hull[hull.size() - 1], points[i]) < 0)
            hull.pop_back();
        if (hull.empty() || hull.back() != points[i]) {
            hull.push_back(points[i]);
        }
    }
    hull.pop_back();
    return hull;
}
```

9.2 Operations

```
ll cross_product(pair<ll, ll> P1, pair<ll, ll> P2, pair<ll, ll> P3) {
    ll x1 = P2.first - P1.first;
    ll y1 = P2.second - P1.second;
    ll x2 = P3.first - P1.first;
    ll y2 = P3.second - P1.second;
    return x1 * y2 - y1 * x2;
}
double distancia(pair<ll, ll> P1, pair<ll, ll> P2) {
    return sqrt((P2.first - P1.first) * (P2.first - P1.first) +
                (P2.second - P1.second) * (P2.second - P1.second));
}
ll dot_product(pair<ll, ll> P1, pair<ll, ll> P2, pair<ll, ll> P3) {
    ll x1 = P2.first - P1.first;
    ll y1 = P2.second - P1.second;
    ll x2 = P3.first - P1.first;
    ll y2 = P3.second - P1.second;
    return x1 * x2 + y1 * y2;
}
```

9.3 Polygon Area

```
typedef pair<ll, ll> Point;
double polygon_area(const vector<Point>& polygon) {
    ll area = 0;
    int n = polygon.size();
    for (int i = 0; i < n; ++i) {
        ll j = (i + 1) % n;
        area += (polygon[i].first * polygon[j].second - polygon[i].
            second * polygon[j].first);
    }
    return abs(area) / 2.0;
}
```

9.4 Ray Casting

```
typedef pair<ll, ll> Point;
bool is_point_in_polygon(const vector<Point>& polygon, Point p) {
    bool inside = false;
    int n = polygon.size();
    for (int i = 0, j = n - 1; i < n; j = i++) {
        if ((polygon[i].second > p.second) != (polygon[j].second > p.
            second) &&
            p.first < (polygon[j].first - polygon[i].first) * (p.second
                - polygon[i].second) /
                      (polygon[j].second - polygon[i].second) + polygon[
                          i].first) {
            inside = !inside;
        }
    }
    return inside;
}
```

10 Other

10.1 Mo's algorithm

```
const int BLOCK_SIZE = 450; using U64 = uint64_t;
struct query {int l, r, id;U64 order;};
U64 hilbertorder(U64 x, U64 y) {
    const U64 logn = __lg(max(x, y) * 2 + 1) | 1;
    const U64 maxn = (1ull << logn) - 1;
    U64 res = 0;
    for (U64 s = 1ull << (logn - 1); s; s >>= 1) {
        bool rx = x & s, ry = y & s;
        res = (res << 2) | (rx ? ry ? 2 : 1 : ry ? 3 : 0);
        if (!rx) {
            if (ry) x ^= maxn, y ^= maxn;
            swap(x, y);
        }
    }
    return res;
} // sort by this order
auto add = [&](int ix) { /* Add A[ix] to state*/};
auto rem = [&](int ix) { /* Remove A[ix] from state*/};
int c_l = 0, c_r = -1; // Cursors [0,-1] so r add 0 on first q
for(const auto &qr: qs){
    while(c_l > qr.l) add(--c_l);
    while(c_r < qr.r) add(++c_r);
    while(c_l < qr.l) rem(c_l++);
    while (c_r > qr.r) rem(c_r--);
    ans[qr.id] = /*State.Answer()*/;
}
```

11 Ecuations

11.1 Combinatorics

$$\binom{n}{k} = \binom{n}{n-k}$$

$$\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}, \quad (1 \le k \le n)$$

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n$$

$$\sum_{m=0}^{n} \binom{m}{k} = \binom{n+1}{k+1}, \quad (n \ge k \ge 0)$$

$$\sum_{k=0}^{m} \binom{n+k}{k} = \binom{n+m+1}{m}$$

$$\binom{n}{0}^2 + \binom{n}{1}^2 + \cdots + \binom{n}{n}^2 = \binom{2n}{n}$$

$$\sum_{k=1}^{n} k\binom{n}{k} = n2^{n-1}$$

$$F_n = \sum_{k=0}^{\lfloor \frac{n-1}{2} \rfloor} \binom{n-k-1}{k}$$

$$F_{n+1} = \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n-k}{k}$$

11.2 Discreta

Vandermonde convolution:

$$\sum_{k=0}^{n} \binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}$$

Multinomial theorem:

$$(x_1 + \cdots + x_m)^n = \sum_{\substack{a_1+\cdots+a_m=n \\ a_i \geq 0}} \frac{n!}{a_1! \cdots a_m!} x_1^{a_1} \cdots x_m^{a_m}$$

Binomial inversion (sequence form):

$$g(n) = \sum_{k=0}^{n} \binom{n}{k} f(k) \quad \Longleftrightarrow \quad f(n) = \sum_{k=0}^{n} (-1)^{n-k} \binom{n}{k} g(k)$$

Stars and bars (nonnegative):

$$x_1 + \cdots + x_k = n,\ x_i \geq 0 \ \Rightarrow\ \# = \binom{n+k-1}{k-1}$$

Positive parts:

$$x_1 + \cdots + x_k = n,\ x_i \geq 1 \ \Rightarrow\ \# = \binom{n-1}{k-1}$$

Compositions of n:

$$\#\{\text{ordered positive sum of } n \text{ into } k \text{ parts}\} = \binom{n-1}{k-1}, \quad \#\{\text{all compositions}\} = 2^{n-1}$$

Upper bounds via inclusion-exclusion:

$$x_1 + \cdots + x_k = n,\ 0 \leq x_i \leq u_i \Rightarrow \# = \sum_{S \subseteq \{1,\ldots,k\}} (-1)^{|S|} \binom{n - \sum_{i \in S}(u_i+1) + k - 1}{k-1}$$

$$(\text{toma } \binom{t}{k-1} = 0 \text{ si } t < k-1)$$

Multiset combinations:

$$\#\{k\text{-multicombinations from } n \text{ types}\} = \binom{n+k-1}{k}$$

Multiset permutations:

$$\#\{\text{perm of multiset with counts } m_1, \ldots, m_r\} = \frac{(m_1 + \cdots + m_r)!}{m_1! \cdots m_r!}$$

Circular permutations:

$$\#\{\text{distinct cyclic orders of } n \text{ items}\} = (n-1)!$$

Surjections count (onto functions):

$$\#\{f : [m] \to [n] \text{ onto}\} = \sum_{j=0}^{n} (-1)^j \binom{n}{j} (n-j)^m = n!\, S(m,n)$$

Derangements:

$$!n = n! \sum_{i=0}^{n} \frac{(-1)^i}{i!} \quad \text{and} \quad !n \approx \frac{n!}{e}$$

Stirling numbers (second kind):

$$S(n,k) = k\, S(n-1,k) + S(n-1,k-1), \quad S(0,0) = 1$$

Stirling numbers (first kind, unsigned):

$$c(n,k) = c(n-1,k-1) + (n-1)\,c(n-1,k), \quad c(0,0) = 1$$

Expansions with falling powers:

$$x^{\underline{n}} = \sum_{k=0}^{n} s(n,k)\,x^k, \qquad x^n = \sum_{k=0}^{n} S(n,k)\,x^{\underline{k}}$$

(here $x^{\underline{k}} = x(x-1)\cdots(x-k+1)$, $s(n,k) = (-1)^{n-k}c(n,k)$)

Bell numbers:

$$B_n = \sum_{k=0}^{n} S(n,k), \qquad \sum_{n\geq 0} B_n \frac{x^n}{n!} = \exp(e^x - 1)$$

Cayley trees:

$$\#\{\text{labeled trees on } n \text{ vertices}\} = n^{\,n-2}$$

Perfect matchings in complete graph:

$$\#\{\text{perfect matchings in } K_{2n}\} = (2n-1)!! = \frac{(2n)!}{2^n n!}$$

Grid shortest paths:

$$\#\{\text{monotone paths from } (0,0) \text{ to } (a,b)\} = \binom{a+b}{a}$$

Ballot (Bertrand special case):

$$p > q \Rightarrow \#\{\text{prefix-wise leading sequences}\} = \frac{p-q}{p+q}\binom{p+q}{q}$$

Alternating binomial sums:

$$\sum_{k=0}^{n} (-1)^k \binom{n}{k} = 0 \ (n \geq 1), \qquad \sum_{k=0}^{n} (-1)^k \binom{n}{k} k^m = 0 \ (0 \leq m < n)$$

Lucas theorem (mod prime p):

$$n = \sum n_i p^i, \quad k = \sum k_i p^i \ \Rightarrow \ \binom{n}{k} \equiv \prod_i \binom{n_i}{k_i} \pmod{p}$$

11.3 Trigonometry

$$\sin(-x) = -\sin x, \qquad \cos(-x) = \cos x, \qquad \tan(-x) = -\tan x$$
$$\sin^2 x + \cos^2 x = 1, \qquad 1 + \tan^2 x = \sec^2 x, \qquad 1 + \cot^2 x = \csc^2 x$$

$$\sin(\alpha \pm \beta) = \sin\alpha\cos\beta \pm \cos\alpha\sin\beta$$
$$\cos(\alpha \pm \beta) = \cos\alpha\cos\beta \mp \sin\alpha\sin\beta$$
$$\tan(\alpha \pm \beta) = \frac{\tan\alpha \pm \tan\beta}{1 \mp \tan\alpha\tan\beta}$$

$$\sin(2x) = 2\sin x\cos x$$
$$\cos(2x) = \cos^2 x - \sin^2 x = 1 - 2\sin^2 x = 2\cos^2 x - 1$$
$$\tan(2x) = \frac{2\tan x}{1 - \tan^2 x}$$
$$\sin(3x) = 3\sin x - 4\sin^3 x, \qquad \cos(3x) = 4\cos^3 x - 3\cos x$$
$$\sin^2\frac{x}{2} = \frac{1-\cos x}{2}, \qquad \cos^2\frac{x}{2} = \frac{1+\cos x}{2}, \qquad \tan\frac{x}{2} = \frac{\sin x}{1+\cos x} = \frac{1-\cos x}{\sin x}$$

$$\sin\alpha\sin\beta = \tfrac{1}{2}\big[\cos(\alpha-\beta) - \cos(\alpha+\beta)\big]$$
$$\cos\alpha\cos\beta = \tfrac{1}{2}\big[\cos(\alpha-\beta) + \cos(\alpha+\beta)\big]$$
$$\sin\alpha\cos\beta = \tfrac{1}{2}\big[\sin(\alpha+\beta) + \sin(\alpha-\beta)\big]$$
$$\sin\alpha \pm \sin\beta = 2\sin\frac{\alpha\pm\beta}{2}\,\cos\frac{\alpha\mp\beta}{2}$$
$$\cos\alpha + \cos\beta = 2\cos\frac{\alpha+\beta}{2}\,\cos\frac{\alpha-\beta}{2}$$
$$\cos\alpha - \cos\beta = -2\sin\frac{\alpha+\beta}{2}\,\sin\frac{\alpha-\beta}{2}$$

$$\sum_{k=0}^{n-1} \cos(a+kd) = \frac{\sin(\frac{nd}{2})}{\sin(\frac{d}{2})}\cos\Big(a + \tfrac{(n-1)d}{2}\Big)$$
$$\sum_{k=0}^{n-1} \sin(a+kd) = \frac{\sin(\frac{nd}{2})}{\sin(\frac{d}{2})}\sin\Big(a + \tfrac{(n-1)d}{2}\Big)$$

Law of sines: $\dfrac{a}{\sin A} = \dfrac{b}{\sin B} = \dfrac{c}{\sin C} = 2R$

Law of cosines: $c^2 = a^2 + b^2 - 2ab\cos C$ (and cyclic)

Area: $\Delta = \frac{1}{2}ab\sin C = \frac{1}{2}bc\sin A = \frac{1}{2}ca\sin B$

$$(x', y') = (x\cos\theta - y\sin\theta,\ x\sin\theta + y\cos\theta)$$

$$e^{ix} = \cos x + i\sin x, \qquad \cos x = \tfrac{e^{ix}+e^{-ix}}{2}, \quad \sin x = \tfrac{e^{ix}-e^{-ix}}{2i}$$

$$\pi \text{ rad} = 180^\circ, \qquad 1^\circ = \frac{\pi}{180} \text{ rad}$$

11.4 Catalan Numbers

Recursive definition:

$$C_0 = C_1 = 1$$

$$C_n = \sum_{k=0}^{n-1} C_k\, C_{n-1-k}, \quad n \geq 2$$

Closed form:

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

Combinatorial equivalent:

$$C_n = \binom{2n}{n} - \binom{2n}{n-1} = \frac{1}{n+1}\binom{2n}{n}, \quad n \geq 0$$

Combinatorial meaning:
Number of ways to: (i) arrange n balanced parenthesis pairs; (ii) full binary trees with $n+1$ leaves; (iii) Dyck paths of length $2n$ that never cross the diagonal.

Generalized form (k):

$$C_n^{(k)} = \frac{k+1}{n+k+1}\binom{2n+k}{n}$$

Extended recurrence:

$$C_n^{(k)} = \sum_{a_1+\cdots+a_k=n} C_{a_1} C_{a_2} \cdots C_{a_k}, \qquad C_0 = 1$$

Efficient recurrence (for computation):

$$C_n = \frac{2(2n-1)}{n+1}\, C_{n-1}, \quad n \geq 1$$

Generating function:

$$C(x) = \sum_{n=0}^{\infty} C_n x^n = \frac{1-\sqrt{1-4x}}{2x}$$

Asymptotic behavior:

$$C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$$

Examples:

$$C_0 = 1,\ C_1 = 1,\ C_2 = 2,\ C_3 = 5,\ C_4 = 14,\ C_5 = 42$$

11.5 Geometry

Rectangle:

$$A = b\,h$$

Area with base b and height h.

Triangle:

$$A = \tfrac{1}{2}\, b\,h$$

$$A = \sqrt{s(s-a)(s-b)(s-c)}, \quad s = \tfrac{a+b+c}{2} \quad \text{(Heron)}$$

Base–height or Heron using side lengths a, b, c.

Parallelogram & rhombus:

$$A_{\text{parallelogram}} = b\,h, \qquad A_{\text{rhombus}} = \tfrac{D\,d}{2}$$

D, d are diagonals of a rhombus.

Trapezoid:

$$A = \tfrac{(B+b)}{2}\, h$$

B and b are the parallel sides (bases).

Regular n-gon:

$$A = \tfrac{1}{2}\,P\,a = \tfrac{n\,l\,a}{2}$$

P perimeter, l side, a apothem.

Circle:

$$A = \pi r^2, \qquad C = 2\pi r$$

Circular sector (angle in radians):

$$A = \tfrac{1}{2} r^2\theta, \qquad \text{arc length } L = r\theta$$

Circular segment (height h):

$$A = r^2 \arccos\left(\frac{r-h}{r}\right) - (r-h)\sqrt{2rh - h^2}$$

Region cut by a chord; $0 < h < 2r$.

Annulus (circular crown):

$$A = \pi(R^2 - r^2)$$

Difference of two concentric disks ($R > r$).

Ellipse:

$$A = \pi ab$$

a, b are semi-axes.

Polygon by coordinates (shoelace):

$$A = \tfrac{1}{2}\left|\sum_{i=1}^{n}(x_i y_{i+1} - x_{i+1} y_i)\right|, \qquad (x_{n+1}, y_{n+1}) = (x_1, y_1)$$

Works for any simple polygon in the plane.

Triangle by coordinates:

$$A = \tfrac{1}{2}\left|x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)\right|$$

Triangle from sides and circumradius/inradius:

$$A = \tfrac{abc}{4R}, \qquad A = r\,s$$

R circumradius, r inradius, s semiperimeter.

Lune (difference of two circular sectors):

$$A_{\text{lune}} = \tfrac{1}{2} r_1^2 \theta_1 - \tfrac{1}{2} r_2^2 \theta_2$$

Two sectors overlapping with angles θ_1, θ_2 matching the same chord.

Lens (two equal circles radius r, center distance d):

$$A = 2r^2 \arccos\left(\frac{d}{2r}\right) - \frac{d}{2}\sqrt{4r^2 - d^2}, \quad 0 < d < 2r$$

Intersection of two equal disks.

Spherical cap (radius R, height h):

$$A_{\text{cap}} = 2\pi Rh$$

Surface area of the cap on a sphere. (Volume: $V = \frac{\pi h^2}{3}(3R - h)$)

11.6 Useful math

Arithmetic series:

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

Squares & cubes:

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

Geometric series ($r \neq 1$):

$$\sum_{i=0}^{n-1} r^i = \frac{r^n - 1}{r - 1}$$

For mod prime p: multiply by $(r-1)^{-1} \equiv (r-1)^{p-2} \bmod p$.

Power sum of base a:

$$1 + a + \cdots + a^n = \frac{a^{n+1}-1}{a-1} \ (a \neq 1), \quad = n+1 \ (a=1)$$

Harmonic numbers:

$$H_n = \sum_{k=1}^{n} \frac{1}{k} \approx \ln n + \gamma + \frac{1}{2n}$$

$\gamma \approx 0.57721$ (Euler–Mascheroni). Useful for estimates.

Basic mod rules:

$$(a \pm b) \bmod m = ((a \bmod m) \pm (b \bmod m)) \bmod m$$

$$(a \cdot b) \bmod m = ((a \bmod m) \cdot (b \bmod m)) \bmod m$$

Fermat little theorem (prime p):

$$a^{p-1} \equiv 1 \pmod{p} \quad \text{if } p \nmid a, \qquad a^{-1} \equiv a^{p-2} \pmod{p}$$

Euler theorem:

$$a^{\varphi(m)} \equiv 1 \pmod{m} \text{ if } \gcd(a,m) = 1$$

Chinese remainder (pairwise coprime):

$$x \equiv a_i \pmod{m_i} \ \Rightarrow \ x \equiv \sum_i a_i M_i y_i \pmod{M}$$

$M = \prod m_i$, $M_i = M/m_i$, $y_i \equiv M_i^{-1} \pmod{m_i}$.

gcd/lcm relation:

$$\operatorname{lcm}(a,b) = \frac{|ab|}{\gcd(a,b)}$$

Binomial theorem:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$$

Stars and bars (non-neg.):

$$x_1 + \cdots + x_k = n, \ x_i \geq 0 \ \Rightarrow \ \# = \binom{n+k-1}{k-1}$$

Permutations & combinations:

$$P(n,k) = \frac{n!}{(n-k)!}, \qquad \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Derangements (approx):

$$!n = \left\lfloor \frac{n!}{e} + \frac{1}{2} \right\rfloor, \qquad !n = n! \sum_{i=0}^{n} \frac{(-1)^i}{i!}$$

Stirling approximation:

$$n! \ \approx \ \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

Inclusion-Exclusion (finite):

$$\left|\bigcup_{i=1}^{m} A_i\right| = \sum_i |A_i| - \sum_{i<j} |A_i \cap A_j| + \cdots + (-1)^{m+1} \left|A_1 \cap \cdots \cap A_m\right|$$

Dot and cross (2D):

$$u \cdot v = u_x v_x + u_y v_y = |u||v| \cos\theta, \quad u \times v = u_x v_y - u_y v_x$$

$|u \times v| = 2\times$ triangle area(u,v). Orientation by sign of $u \times v$.

Distance point to line AB:

$$\operatorname{dist}(P, AB) = \frac{|(B-A) \times (P-A)|}{|B-A|}$$

Projection length on AB:

$$\operatorname{proj}_{AB}(P) = \frac{(P-A) \cdot (B-A)}{|B-A|}$$

AM-GM (non-neg.):

$$\frac{x_1+\cdots+x_n}{n} \ \geq \ (x_1\cdots x_n)^{1/n}$$

Cauchy–Schwarz:

$$\left(\sum_i a_i b_i\right)^2 \leq \left(\sum_i a_i^2\right)\left(\sum_i b_i^2\right)$$

Log rules:

$$\log_a b = \frac{\ln b}{\ln a}, \qquad \log(ab) = \log a + \log b$$

Fast exponent splits:

$$a^{x+y} = a^x a^y, \qquad a^{2^k} = \underbrace{(\cdots(a^2)^2\cdots)^2}_{k \text{ times}}$$

Divisor count/sum (multiplicative):

$$n = \prod p_i^{e_i} \Rightarrow \tau(n) = \prod(e_i+1), \qquad \sigma(n) = \prod \frac{p_i^{e_i+1}-1}{p_i-1}$$

$\tau(n)$ = number of divisors, $\sigma(n)$ = sum of divisors.

Linearity of expectation:

$$\mathbb{E}\Big[\sum_i X_i\Big] = \sum_i \mathbb{E}[X_i] \quad \text{(no independence needed)}$$

Binomial distribution:

$$X \sim \mathrm{Bin}(n,p) \Rightarrow \mathbb{E}[X] = np,\ \mathrm{Var}(X) = np(1-p)$$

11.7 Mobius

Mobius function mu:

$$\mu(1) = 1$$

$$\mu(n) = 0 \text{ if } \exists p^2 \mid n, \qquad \mu(n) = (-1)^k \text{ if } n \text{ is square-free with } k \text{ distinct primes.}$$

Basic convolutions:

$$\sum_{d|n} \mu(d) = \begin{cases} 1, & n = 1, \\ 0, & n > 1, \end{cases} \qquad (\mu * \mathbf{1})(n) = \varepsilon(n).$$

Mobius inversion (divisor-sum):
If

$$g(n) = \sum_{d|n} f(d),$$

then

$$f(n) = \sum_{d|n} \mu(d)\, g\Big(\frac{n}{d}\Big) = \sum_{d|n} \mu\Big(\frac{n}{d}\Big)\, g(d).$$

Inversion over multiples:
If

$$G(n) = \sum_{k\geq 1} f(k\,n),$$

then

$$f(n) = \sum_{k\geq 1} \mu(k)\, G(k\,n).$$

Euler totient via mu:

$$\varphi(n) = \sum_{d|n} \mu(d)\,\frac{n}{d}.$$

Counting coprimes up to x:

$$\#\{1 \leq m \leq x : \gcd(m,n) = 1\} = \sum_{d|n} \mu(d) \left\lfloor \frac{x}{d} \right\rfloor.$$

Square-free count up to x:

$$Q(x) = \#\{n \leq x : n \text{ square-free}\} = \sum_{k \leq \sqrt{x}} \mu(k) \left\lfloor \frac{x}{k^2} \right\rfloor.$$

GCD=1 k-tuples:

$$\#\{1 \le x_1, \ldots, x_k \le n : \gcd(x_1, \ldots, x_k) = 1\} = \sum_{d=1}^{n} \mu(d) \left\lfloor \frac{n}{d} \right\rfloor^k.$$

Dirichlet series:

$$\sum_{n=1}^{\infty} \frac{\mu(n)}{n^s} = \frac{1}{\zeta(s)}, \qquad \Re(s) > 1.$$

Mertens function:

$$M(x) = \sum_{n \le x} \mu(n).$$

11.8 Burnside

Necklaces under cyclic C_n (gcd form):

$$N_{\text{rot}} = \frac{1}{n} \sum_{k=0}^{n-1} m^{\gcd(n,k)}.$$

Equivalent divisor form (for C_n):

$$N_{\text{rot}} = \frac{1}{n} \sum_{d \mid n} \varphi(d)\, m^{\,n/d}.$$

Necklaces/bracelets under dihedral D_n (gcd form):

$$N = \begin{cases} \dfrac{1}{2n} \left(\displaystyle\sum_{k=0}^{n-1} m^{\gcd(n,k)} + n\, m^{(n+1)/2} \right), & n \text{ odd}, \\ \dfrac{1}{2n} \left(\displaystyle\sum_{k=0}^{n-1} m^{\gcd(n,k)} + \tfrac{n}{2}\, m^{n/2} + \tfrac{n}{2}\, m^{n/2+1} \right), & n \text{ even}. \end{cases}$$

Here gcd is the greatest common divisor. A rotation by k positions has $\gcd(n,k)$ cycles.