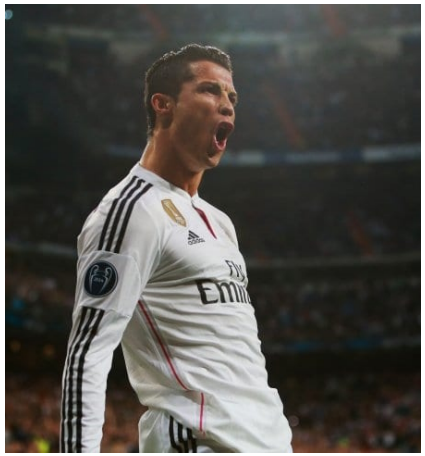
El Bicho

DondeEstasCR7



20/08/2025

Contents

1 Algos

1.1 Fast Io

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

#define cpu() ios::sync_with_stdio(false);cin.tie(nullptr);

using namespace std;
using namespace __gnu_pbds;
template <class T>
using ordered_set = tree<T, null_type, less_equal<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define pb push_back
#define sz(a) ((int)(a).size())
#define ff first
#define ss second
#define all(a) (a).begin(), (a).end()
#define allr(a) (a).rbegin(), (a).rend()
#define approx(a) fixed << setprecision(a)

template <class T> void read(vector<T> &v);
template <class F, class S> void read(pair<F, S> &p);
template <class T, size_t Z> void read(array<T, Z> &a);
template <class T> void read(T &x) {cin >> x;}
template <class R, class... T> void read(R& r, T&... t){read(r); read(t
    ...);};
template <class T> void read(vector<T> &v) {for(auto& x : v) read(x);}
template <class F, class S> void read(pair<F, S> &p) {read(p.ff, p.ss);}
template <class T, size_t Z> void read(array<T, Z> &a) { for(auto &x : a
    ) read(x); }

template <class F, class S> void pr(const pair<F, S> &x);
template <class T> void pr(const T &x) {cout << x;}
template <class R, class... T> void pr(const R& r, const T&... t) {pr(r)
    ; pr(t...);}
template <class F, class S> void pr(const pair<F, S> &x) {pr("{", x.ff,
    ",␣", x.ss, "}\n");}
void ps() {pr("\n");}
template <class T> void ps(const T &x) {pr(x); ps();}
template <class T> void ps(vector<T> &v) {for(auto& x : v) pr(x, '␣');
    ps();}
template <class T, size_t Z> void ps(const array<T, Z> &a) { for(auto &x
     : a) pr(x, '␣'); ps(); }
template <class F, class S> void ps(const pair<F, S> &x) {pr(x.ff, '␣',
    x.ss); ps();}
template <class R, class... T> void ps(const R& r,  const T &...t) {pr(r
    , '␣'); ps(t...);}

using ll = long long;
const double PI = 3.141592653589793;
const ll MX = 1e9 + 1;

void solve() {

}

int main() {
  cpu();

  int t = 1;
  //cin >> t;
  while (t--) {
    solve();
  }

  return 0;
}
```

1.2 Tablas Y Cotas

```cpp
// Primeros 180 Primos:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89
97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179
181 191 193 197 199 211 223 227 229 233 239 241 251 257 263 269 271
277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379
383 389 397 401 409 419 421 431 433 439 443 449 457 461 463 467 479
487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599
601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701
709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823
827 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937 941
947 953 967 971 977 983 991 997 1009 1013 1019 1021 1031 1033 1039
1049 1051 1061 1063 1069
```

```

// Primos cercanos a 10^n
9941 9949 9967 9973 10007 10009 10037 10039 10061 10067 10069 10079
99961 99971 99989 99991 100003 100019 100043 100049 100057 100069
999959 999961 999979 999983 1000003 1000033 1000037 1000039 9999943
9999971 9999973 9999991 10000019 10000079 10000103 10000121 99999941
99999959 99999971 99999989 100000007 100000037 100000039 100000049
999999893 999999929 999999937 1000000007 1000000009 1000000021
1000000033

// Cantidad de primos menores que 10^n
pi(10^1) = 4 -> [2, 3, 5, 7]
pi(10^2) = 25 -> [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    47, 53, 59, 61, 67, 71, 73, 79, 83, 89]
pi(10^3) = 168
pi(10^4) = 1.229
pi(10^5) = 9.592
pi(10^6) = 78.498
pi(10^7) = 664.579
pi(10^8) = 5.761.455
pi(10^9) = 50.847.534
pi(10^10) = 455.052.511
pi(10^11) = 4.118.054.813
pi(10^12) = 37.607.912.018

// Cantidad de divisores
sigma0(60) = 12 -> [1, 2, 3, 4, 6, 10, 12, 15, 20, 30, 60]
sigma0(120) = 16 -> [1, 2, 3, 4, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60,
    120]
sigma0(180) = 18 -> [1, 2, 3, 4, 5, 6, 9, 10, 12, 15, 18, 20, 30, 36,
    60, 90, 180]
sigma0(240) = 20 -> [1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24, 30,
    40, 60, 80, 120, 240]
sigma0(360) = 24
sigma0(720) = 30
sigma0(840) = 32
sigma0(1.260) = 36
sigma0(1.680) = 40
sigma0(10.080) = 72
sigma0(15.120) = 80
sigma0(50.400) = 108
sigma0(83.160) = 128
sigma0(110.880) = 144
sigma0(498.960) = 200
sigma0(554.400) = 216
sigma0(1.081.080) = 256
sigma0(1.441.440) = 288
sigma0(4.324.320) = 384
sigma0(8.648.640) = 448

// Suma de divisores
sigma1(96) = 252 -> [1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 96]
sigma1(108) = 280 -> [1, 2, 3, 4, 6, 9, 12, 18, 27, 36, 54, 108]
sigma1(120) = 360 -> [1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 24, 30, 40,
    60, 120]
sigma1(144) = 403 -> [1, 2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 36, 48, 72,
    144]
sigma1(168) = 480
sigma1(960) = 3.048
sigma1(1.008) = 3.224
sigma1(1.080) = 3.600
sigma1(1.200) = 3.844
sigma1(4.620) = 16.128
sigma1(4.680) = 16.380
sigma1(5.040) = 19.344
sigma1(5.760) = 19.890
sigma1(8.820) = 31.122
sigma1(9.240) = 34.560
sigma1(10.080) = 39.312
sigma1(10.920) = 40.320
sigma1(32.760) = 131.040
sigma1(35.280) = 137.826
sigma1(36.960) = 145.152
sigma1(37.800) = 148.800
sigma1(60.480) = 243.840
sigma1(64.680) = 246.240
sigma1(65.520) = 270.816
sigma1(70.560) = 280.098
sigma1(95.760) = 386.880
sigma1(98.280) = 403.200
sigma1(100.800) = 409.448
sigma1(491.400) = 2.083.200
sigma1(498.960) = 2.160.576
sigma1(514.080) = 2.177.280
sigma1(982.800) = 4.305.280
sigma1(997.920) = 4.390.848
```

```
sigma1(1.048.320) = 4.464.096
sigma1(4.979.520) = 22.189.440
sigma1(4.989.600) = 22.686.048
sigma1(5.045.040) = 23.154.768
sigma1(9.896.040) = 44.323.200
sigma1(9.959.040) = 44.553.600
sigma1(9.979.200) = 45.732.192

// Factoriales
0! = 1  (int)
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5.040
8! = 40.320
9! = 362.880
10! = 3.628.800
11! = 39.916.800
12! = 479.001.600   (int)
13! = 6.227.020.800   (ll)
14! = 87.178.291.200
15! = 1.307.674.368.000
16! = 20.922.789.888.000
17! = 355.687.428.096.000
18! = 6.402.373.705.728.000
19! = 121.645.100.408.832.000
20! = 2.432.902.008.176.640.000   (ll)
21! = 51.090.942.171.709.400.000  (__int128_t)

// Limites de enteros
max signed char = 127
max unsigned char = 255
max signed int = 2.147.483.647
max unsigned int = 4.294.967.295
max signed long long = 9.223.372.036.854.775.807
max unsigned long long = 18.446.744.073.709.551.615
max signed __int128_t =
    170.141.183.460.469.231.731.687.303.715.884.105.727
max unsigned __int128_t =
    340.282.366.920.938.463.463.374.607.431.768.211.456
```

2 Bit Manipulation

Técnicas para manipular bits individuales y operaciones a nivel de bit. Incluye macros útiles para competencias de programación.

2.1 Bits

Macros esenciales para manipulación de bits: verificar potencias de 2, establecer/limpiar bits, contar bits, y operaciones con LSB/MSB.

```cpp
using ull = unsigned long long;
const ull UNSIGNED_LL_MAX = 18'446'744'073'709'551'615;
// Verifica si S es potencia de dos (y distinto de cero)
#define isPowerOfTwo(S) ((S) && !((S) & ((S) - 1)))
// Retorna la potencia de dos mas cercana a S
#define nearestPowerOfTwo(S) (1LL << lround(log2(S)))
// Calcula S % N cuando N es potencia de dos
#define modulo(S, N) ((S) & ((N) - 1))

// Verifica si el bit esta encendido (bit en 1)
#define isOn(S, i) ((S) & (1LL<<(i)))
// Enciende el bit (Lo pone en 1)
#define setBit(S, i) ((S) |= (1LL<<(i)))
// Apaga el bit (Lo pone en 0)
#define clearBit(S, i) ((S) &= ~(1LL<<(i)))
// Invierte el estado del bit (0 <-> 1)
#define toggleBit(S, i) ((S) ^= (1LL<<(i)))
// Enciende los primeros 'n' bits (idx-0)
#define setAll(S, n) ((S) = ((n)>=64 ? ~0LL : (1LL << (n))-1))

// Extrae el bit menos significativo 0100 (Least Significant Bit)
#define lsb(S) ((S) & -(S))
// Numero de ceros a la derecha (Posicion del LSB, idx-0)
#define idxLastBit(x) __builtin_ctzll(x)
// Extrae el bit mas significativo 0100 (Most Significant Bit)
#define msb(S) (1LL << (63 - __builtin_clzll(S)))
// Posicion del MSB (63 - ceros a la izquierda, idx-0)
#define idxFirstBit(x) (63 - __builtin_clzll(x))

#define countAllOnes(x) __builtin_popcountll(x)
// Apaga el ultimo bit encendido (el menos significativo)
#define turnOffLastBit(S) ((S) & ((S) - 1))
// Enciende el ultimo cero menos significativo
#define turnOnLastZero(S) ((S) | ((S) + 1))
```

```cpp
// Apaga todos los bits encendidos mas a la derecha consecutivos
#define turnOffLastConsecutiveBits(S) ((S) & ((S) + 1))
// Enciende los ceros consecutivos mas a la derecha
#define turnOnLastConsecutiveZeroes(S) ((S) | ((S) - 1))

// Mascara de bits (mask -> subconjunto) O(2^N)
for (int mask = 0; mask < (1 << N); mask++)

// Recorrer subconjuntos de un superconjunto (menos el vacio)
int b = 0b1011; // Representacion binaria de un decimal en int
for (int i = b; i; i = (i - 1) & b) {
  cout << bitset<4>(i) << "\n";
}

void printBin(ll x) {
  // 63 -> unsigned ll, 62 -> ll, 31 -> unsigned int, 30 -> int
  for (ll i = 63; i >= 0; i--)
    cout << ((x >> i) & 1);
  cout << '\n';
}
```

3 Combinatory

3.1 Combi Brute Sin Mod

```cpp
// nCk brute force sin MOD n <= 20
long long nCk_bruteforce(long long n, long long k) {
    if (k < 0 || k > n) return 0;
    long long res = 1;
    for (long long i = 1; i <= k; i++) {
        res = res * (n - i + 1) / i; // aqui la division es exacta
    }
    return res;
}

// nPk brute force sin MOD n <= 20
long long nPk_bruteforce(long long n, long long k) {
    if (k < 0 || k > n) return 0;
    long long res = 1;
    for (long long i = 0; i < k; i++) {
        res *= (n - i);
    }
    return res;
}
```

3.2 Combinatory

OJO: Es necesario usar binpow con MOD primo

```cpp
// Devuelve el inverso modular de a mod MOD
// Usa el Teorema Pequeno de Fermat: a^(MOD-2) === a^(-1) (mod MOD)
// (valido solo si MOD es primo)
ll inv(ll a, ll p = MOD) {
    return binpow(a, p - 2, p);
}

// Factoriales e inversos factoriales precomputados
// fact[n]  = n! mod MOD
// invf[n]  = (n!)^(-1) mod MOD
// Precomputa en O(n)
vector<ll> fact(MAXN + 1), invf(MAXN + 1);

void precompute_factorials() {
    fact[0] = 1;
    for (int i = 1; i <= MAXN; i++) {
        fact[i] = fact[i - 1] * i % MOD;
    }
    invf[MAXN] = inv(fact[MAXN]);
    for (int i = MAXN; i > 0; i--) {
        invf[i - 1] = invf[i] * i % MOD;
    }
}

// Combinatoria de n en k: nCk(n, k) para n <= 10^6
// "n choose k" = n! / (k! * (n-k)!) mod MOD
// Retorna 0 si k > n
ll nCk(ll n, ll k) {
    if (k < 0 || k > n) return 0;
    return fact[n] * invf[k] % MOD * invf[n - k] % MOD;
}

// Permutacion de n en k: nPk(n, k) para n <= 10^6
// Calcula permutaciones: "n permute k" = n! / (n-k)! mod MOD
// Retorna 0 si k > n
ll nPk(ll n, ll k) {
    if (k < 0 || k > n) return 0;
```

```
    return fact[n] * invf[n - k] % MOD;
}
```

4 Graph

Algoritmos de grafos: DFS, BFS, componentes fuertemente conexas, y otras estructuras de datos para problemas de grafos.

4.1 Bfs

```
vector<bool> vis(n+1);
queue<int> q;
function<void(int)> bfs = [&](int start) {
  vis[start] = true;
  q.push(start);
  while (!q.empty()) {
    int sz = q.size();
    while (sz--) {
      int u = q.front();
      q.pop();
      for (int& v : adj[u]) {
        if (vis[v]) continue;
        vis[v] = true;
        q.push(v);
      }
    }
  }
};

for (int u = 1; u <= n; u++) {
  if (vis[u]) continue;
  bfs(u);
}
```

4.2 Bipartite

```
int N, M; cin >> N >> M;
vector<vector<int>> adj(N + 1);
while (M--) {
  int u, v; cin >> u >> v;
  adj[u].push_back(v);
  adj[v].push_back(u);
}

vector<bool> vis(N + 1);
vector<int> col(N + 1, 0);
// bipartite graph
function<bool(int, int)> dfs = [&](int u, int c) {
  vis[u] = 1;
  col[u] = c;

  for (auto v : adj[u]) {
    if (vis[v] && col[u] == col[v]) return false;
    else if (!vis[v] && !dfs(v, c ^ 1)) return false;
  }
  return true;
};

for (int i = 1; i <= N; i++) {
  if (vis[i]) continue;
  if (dfs(i, 1) == false) {
    cout << "IMPOSSIBLE";
    return;
  }
}

for (int i = 1; i <= N; i++) cout << (col[i] ? 1 : 2) << '␣';
```

4.3 Dfs

```
vector<bool> vis(n+1);
function<void(int)> dfs = [&](int u) {
  vis[u] = true;
  for (int& v : adj[u]) {
    if (vis[v]) continue;
    dfs(v);
  }
};

for (int u = 1; u <= n; u++) {
  if (vis[u]) continue;
  dfs(u);
}
```

4.4 Dfs 2D

```cpp
int N, M; cin >> N >> M;
vector<vector<char>> grid(N, vector<char>(M));
for (int i = 0; i < N; i++) {
  for (int j = 0; j < M; j++) {
    cin >> grid[i][j];
  }
}

vector<vector<bool>> vis(N, vector<bool>(M));
vector<int> dx = {-1, 1, 0, 0}, dy = {0, 0, -1, 1};
function<void(int, int)> dfs = [&](int x, int y) {
  vis[x][y] = 1;

  for (int d = 0; d < 4; d++) {
    int nx = x + dx[d], ny = y + dy[d];
    if (0 <= nx && nx < N && 0 <= ny && ny < M && grid[nx][ny] == '.' &&
        !vis[nx][ny]) dfs(nx, ny);
  }
};

int comp = 0;
for (int i = 0; i < N; i++) {
  for (int j = 0; j < M; j++) {
    if (vis[i][j] || grid[i][j] == '#') continue;
    dfs(i, j);
    comp++;
  }
}

cout << comp;
```

4.5 Disjoint Set Union Dsu

```cpp
struct DSU{
  vector<int> p, size;
  DSU(int n){
    p.resize(n + 1), size.resize(n + 1,1);
    for(int i = 1; i <= n; i++) p[i] = i;
  }

  int find(int x){
    if(p[x] != x) p[x] = find(p[x]);
    return p[x];
  }

  void merge(int x, int y){
    x = find(x), y = find(y);
    if(x == y) return;
    if(size[x] < size[y]) swap(x, y);
    size[x] += size[y];
    p[y] = x;
  }
};
```

4.6 Djisktra

```cpp
template <class T> using pq = priority_queue<T>;
template <class T> using pqg = priority_queue<T, vector<T>, greater<T>>;

void solve() {
  int n, m; cin >> n >> m;
  vector<vector<pair<int, ll>>> adj(n+1);
  while (m--) {
    int u, v; ll w; cin >> u >> v >> w;
    adj[u].push_back({v, w});
  }

  vector<ll> dist(n+1, MX);
  pqg<pair<ll, int>> q;
  q.push({0LL, 1});
  dist[1] = 0LL;
  while (!q.empty()) {
    auto [d, u] = q.top();
    q.pop();
    if (dist[u] < d) continue;
    for (auto [v, w] : adj[u]) {
      ll new_d = d + w;
      if (new_d < dist[v]) {
        dist[v] = new_d;
        q.push({dist[v], v});
      }
    }
  }

  for (int u = 1; u <= n; u++) cout << dist[u] << ' ';
  cout << '\n';
```

```
}
```

4.7 Lowest Common Ancestor Lca

```
struct LCA{
  int n, l, timer = 0;
  vector<vector<int>> up, adj;
  vector<int> depth, in, out;

  LCA(int _n) {
    n = _n + 1;
    l = ceil(log2(n));
    up.resize(n, vector<int>(l + 1));
    adj.resize(n);
    depth.resize(n);
    in.resize(n);
    out.resize(n);
  }

  void add_edge(int p, int u){
    adj[p].push_back(u);
    adj[u].push_back(p);
  }

  void dfs(int u = 1, int p = 1){
    up[u][0] = p;
    depth[u] = depth[p] + 1;
    in[u] = ++timer;
    for(int level = 1; level <= l; level++){
      up[u][level] = up[up[u][level - 1]][level - 1];
    }
    for(int v : adj[u]){
      if(v == p) continue;
      dfs(v, u);
    }
    out[u] = ++timer;
  }

  bool is_ancestor(int p, int u){
    return in[p] <= in[u] && out[p] >= out[u];
  }

  int query(int u, int v){
    if(is_ancestor(u, v)) return u;
    if(is_ancestor(v, u)) return v;

    for(int bit = l; bit >= 0; bit--){
      if(is_ancestor(up[u][bit], v)) continue;
      u = up[u][bit];
    }
    return up[u][0];
  }

  int ancestor(int u, int k){
    if(depth[u] <= k) return -1;
    for(int bit = 0; bit <= l; bit++){
      if(k >> bit & 1) u = up[u][bit];
    }
    return u;
  }

  int distance(int u, int v){
    return depth[u] + depth[v] - 2 * depth[query(u, v)];
  }
};
```

4.8 Scc

Algoritmo de Tarjan para encontrar componentes fuertemente conexas (SCC) en un grafo dirigido.

```
// "These works to find a componente fuertemente conexa that it's in
    directed graph"
struct SCC{
  int N = 0, id;
  vector<vector<int>> adj;
  vector<int> ind, low;
  stack<int> s;
  vector<bool> in_stack;
  vector<vector<int>> components;
  vector<int> component_id;

  //1-indexed
  SCC(int n = 0){ N = n + 1, adj.assign(N, {}); }
  SCC(const vector<vector<int>> & _adj){ adj = _adj, N = adj.size(); }

  void add_edge(int from, int to){
```

```
    adj[from].push_back(to);
  }

  void dfs(int u){
    low[u] = ind[u] = id++;
    s.push(u);
    in_stack[u] = true;
    for(int v : adj[u]){
      if(ind[v] == -1){
        dfs(v);
        low[u] = min(low[u], low[v]);
      }else if(in_stack[v]){
        low[u] = min(low[u], ind[v]);
      }
    }
    if(low[u] == ind[u]){
      components.emplace_back();
      vector<int> & comp = components.back();
      while(true){
        assert(!s.empty());
        int x = s.top(); s.pop();
        in_stack[x] = false;
        component_id[x] = components.size() - 1;
        comp.push_back(x);
        if(x == u) break;
      }
    }
  }

  vector<vector<int>> get(){
    ind.assign(N, - 1); low.assign(N, -1); component_id.assign(N, -1);
    s = stack<int>();
    in_stack.assign(N, false);
    id = 0;
    components = {};
    for(int i = 1; i < N; i++)
      if(ind[i] == -1) dfs(i);

    // reverse(components.begin(), components.end()); return components;
        // SCC in topological order
    return components; // SCC in reverse topological order
  }
};
```

4.9 Topological Sort

```
vector<int> top_sort(vector<vector<int>>& adj){
  int n = adj.size();
  bool cycle = false;
  vector<int> sorted, color(n);
  function<void(int)> dfs = [&](int u){
    color[u] = 1;
    for(int v : adj[u]){
      if(color[v] == 0 && !cycle) dfs(v);
      else if(color[v] == 1) cycle = true;
    }
    color[u] = 2;
    sorted.push_back(u);
  };
  for(int i = 1; i < n; i++){
    if(color[i] == 0 && !cycle) dfs(i);
  }
  if(cycle){return {};}
  reverse(sorted.begin(), sorted.end());
  return sorted;
}
```

5 Number Theory

5.1 Euler Toliente

```
class EulerTotiente {
  public:
  //* metodo en O(sqrt(n))
  template <typename T>
  T euler_classic(T n) {
    T result = n;
    for(T i = 2; i * i <= n; i++) {
      if(n % i == 0) {
        while(n % i == 0) n /= i;
        result -= result / i;
      }
    }
    if(n > 1) {
      result -= result / n;
    }
    return result;
```

```
  }

  //* metodo en O(nlog(log(n))
  void euler_faster(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;
        }
      }
    }
    for(int i = 1; i <= n; i++) {
      cout << i << ' ' << phi[i] << '\n';
    }
  }
};
```

5.2 Gcd Lcm

```
// Maximo comun divisor (GCD): Algoritmo de Euclides
int gcd(int a, int b) {
    if (a > b) swap(a, b);
    if (a == 0) return b;
    return gcd(b % a, a);
}

// Minimo comun multiplo (LCM): Calculado con GCD
    int lcm(int a, int b) {
    return (a * b) / gcd(a, b);
}
```

5.3 Number Theory

```
// Divisores de N: Hasta N = 10^6
vector<int> divisores(int N) {
  vector<int> divs;
  for (int d = 1; d * d <= N; d++) {
    if (N % d == 0) {
      divs.push_back(d);
      if (N / d != d) divs.push_back(N / d);
    }
  }
  return divs;
}

// Factorizacion de N: Hasta N = 10^6
vector<pair<int, int>> factorizar(int N) {
  vector<pair<int, int>> facts;
  for (int p = 2; p * p <= N; p++) {
    if (N % p == 0) {
      int exp = 0;
      while (N % p == 0) {
        exp++;
        N /= p;
      }
      facts.push_back({ p, exp });
    }
  }
  if (N > 1) facts.push_back({ N, 1 });
  return facts;
}

// Primalidad: Hasta N = 10^6 - O(sqrt(N))
bool isPrime(int N) {
  if (N < 2) return false;
  for (int d = 2; d * d <= N; d++) {
    if (N % d == 0) return false;
  }
  return true;
}
```

5.4 Phi Euler

Phi(n) = contar la cantidad de numero coprimos entre 1 a n

```
int phi(int n) {
    int ans = n;
    for(int i = 2; i * i <= n; i++) {
        if(n % i == 0) {
            while (n % i == 0) {
                n /= i;
            }
            ans -= ans / i;
        }
```

```
    }
    if(n > 1) {
        ans -= ans / n;
    }
    return ans;
}

//* phi(n) -> complex: O(log(log(n)))
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;
        }
    }
}
```

5.5 Potenciacion Binaria

```
using ll = long long;
const int MAXN = 1e6;   // limite superior de n
const ll MOD = 1e9 + 7; // primo grande

// Potenciacion binaria modular a^b mod p
ll binpow(ll a, ll b, ll m = MOD) {
    a %= m;
    ll res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a % m;
        a = a * a % m;
        b >>= 1;
    }
    return res;
}
```

5.6 Sieve

```
// Criba de Eratostenes: Hasta N = 10^6
void sieve(vector<bool>& is_prime) {
  int N = (int) is_prime.size();
  if (!is_prime[0]) is_prime.assign(N+1, true);
  is_prime[0] = is_prime[1] = false;
  for (int p = 2; p * p <= N; p++) {
    if (is_prime[p]) {
      for (int i = p * p; i <= N; i += p) {
        is_prime[i] = false;
      }
    }
  }
}
```

5.7 Sieve Bitset

```
// Hasta N = 10^8 aprox en 1s
const int MAX_V = 1e7 + 5;
bitset<MAX_V> composite;
void sieve() {
    composite[0] = composite[1] = true;
    for (int i = 2; i * i < MAX_V; i++) {
        if (composite[i]) continue;
        for (int j = i * i; j < MAX_V; j += i) {
            composite[j] = true;
        }
    }
}

int main() {
    sieve();
    for (int i = 2; i < 100; i++) {
        cout << i << " is_primes : " << !composite[i] << '\n';
    }
}
```

5.8 Sum Of Divisors

```
//* Sum of divs
long long SumOfDivisors(long long num) {
    long long total = 1;

    for (int i = 2; (long long)i * i <= num; i++) {
        if (num % i == 0) {
```

```
            int e = 0;
            do {
                e++;
                num /= i;
            } while (num % i == 0);

            long long sum = 0, pow = 1;
            do {
                sum += pow;
                pow *= i;
            } while (e-- > 0);
            total *= sum;
        }
    }
    if (num > 1) {
        total *= (1 + num);
    }
    return total;
}
```

6 Segment Tree

6.1 Find Two Numbers

```
// "find two number where the sum is x, and gcd(a, b) > 1" b
auto find = [&](ll x){
  for(int d = 2; d <= x / 2; d++){
    if(x % d == 0){
      ll m = 1, n = (x / d) - 1;
      ll a = d * m, b = d * n;
      if(__gcd(a, b) > 1){
        cout<< a << ' ' << b;
        ps();
        return;
      }
    }
  }
};
```

6.2 Segment Tree Recursivo

```
template<typename T>
struct segment_tree{
  int N;
  T Z = 0;
  vector<T> tree;
  segment_tree(int N) : N(N) {
    tree.resize(2 * N);
  }

  segment_tree(vector<T>& A){
    N = (int) A.size();
    tree.resize(2 * N);
    build(A, 1, 0, N - 1);
  }

  auto& operator[](size_t i) { return tree[i]; } // this funciton works
      for get element int this position
private:

  T op(T& a, T& b){ return a + b; }
  // O (n)
  void build(vector<T>& values, int node, int l, int r){
    // if l and r are equal both are leaf node
    // left node = [l, m]
    // m = (l + r) / 2
    // left and right are nodes
    // left interval = [l, m], right intervla = [m + 1, r]
    // after complete fill nodes of left and right, we need to fill the
        [l, r] node
    if(l == r){
      tree[node] = values[l];
      return;
    }
    int m = (l + r) >> 1;
    int left = node + 1;
    int right = node + 2 * (m - l + 1);

    build(values, left, l, m);
    build(values, right, m + 1, r);

    tree[node] = op(tree[left], tree[right]);
  }

  // O (log N)
  void modify(int pos, T value, int node,  int l,  int r){
```

```
    // if l and r are equal, we found our node and update it
    if(l == r){
      tree[node] = value;
      return;
    }
    int m = (l + r) >> 1; // we get the mid
    int left = node + 1;
    int right = node + 2 * (m - l + 1);

    if(pos <= m) modify(pos, value, left, l, m);
    else modify(pos, value, right, m + 1, r);

    tree[node] = op(tree[left], tree[right]);
  }

  void update(int pos, T value, int node,  int l,  int r){
    // if l and r are equal, we found our node and update it
    if(l == r){
      tree[node] = op(tree[node], value);
      return;
    }
    int m = (l + r) >> 1; // we get the mid
    int left = node + 1;
    int right = node + 2 * (m - l + 1);

    if(pos <= m) update(pos, value, left, l, m);
    else update(pos, value, right, m + 1, r);

    tree[node] = op(tree[left], tree[right]);
  }

  // O(log N)
  T query(int ql, int qr, int node, int l, int r){
    if(r < ql || l > qr) return Z; // CHECK
    if(ql <= l && r <= qr) return tree[node];
    int m = (l + r) >> 1;
    int left = node + 1;
    int right = node + 2 * (m - l + 1);
    T ansL = query(ql, qr, left, l, m);
    T ansR = query(ql, qr, right, m + 1, r);
    return op(ansL, ansR);
  }
public:
  void build(vector<T>& values){ build(values, 1, 0, N - 1); }

  void modify(int pos, T value){ modify(pos, value, 1, 0, N - 1); }

  void update(int pos, T value){ update(pos, value, 1, 0, N - 1); }

  T query(int ql, int qr){ return query(ql, qr, 1, 0, N - 1); }
};
```

6.3 Segment Tree V2

```
// "This segment_tree I understand better how it works"
template<typename T>
struct seg_tree {
    int N;
    T Z = 0;
    vector<T> tree;

    seg_tree(int N) : N(N) {
        tree.resize(4 * N);
    }

    seg_tree(vector<T>& A) {
        N = (int)A.size();
        tree.resize(4 * N);
        build(A, 1, 0, N-1);
    }

private:
    T op(T a, T b) {
        return a + b;
    }

    void build(vector<T>& a, int node, int left, int right) {
        if(left == right) {
            tree[node] = a[left];
            return;
        }
        int mid = (left + right) >> 1;
        build(a, 2 * node, left, mid);
        build(a, 2 * node + 1, mid + 1, right);
        tree[node] = op(tree[2 * node], tree[2 * node + 1]);
    }
```

```
    void modify(int pos, T value, int node, int left, int right) {
        if(left == right) {
            tree[node] = value;
            return;
        }
        int mid = (left + right) >> 1;
        if(pos <= mid)
            modify(pos, value, 2 * node, left, mid);
        else
            modify(pos, value, 2 * node + 1, mid + 1, right);
        tree[node] = op(tree[2 * node], tree[2 * node + 1]);
    }

    T query(int l, int r, int node, int left, int right) {
        if(r < left || l > right) return Z;
        if(l <= left && right <= r) return tree[node];
        int mid = (left + right) >> 1;
        T leftSum = query(l, r, 2 * node, left, mid);
        T rightSum = query(l, r, 2 * node + 1, mid + 1, right);
        return op(leftSum, rightSum);
    }

public:
    void build(vector<T>& a) { build(a, 1, 0, N-1); }
    void modify(int pos, T value) { modify(pos, value, 1, 0, N-1); }
    T query(int l, int r) { return query(l, r, 1, 0, N-1); }
};
```

6.4 Segment Tree V3

```
// snippet seg_tree_2 "Description" b
template<class T>
struct segment_tree{
    int n;
    vector<T> tree;

    segment_tree(int n){
        this -> n = n;
        tree.resize(2 * n);
    }

    segment_tree(vector<T>& values){
        this -> n = values.size();
        tree.resize(2 * n);
        for(int i = 0; i < n; i++) upd(i, values[i]);
    }

    //CHANGE
    T compare(T a, T b){
        return a + b;
    }

    void modify(int index, T value){
        index += n;
        tree[index] = value;
        for(index >>= 1; index >= 1; index >>= 1) tree[index]= compare(
            tree[2 * index], tree[2 * index + 1]);
    }

    void upd(int index, T value){
        index += n;
        tree[index] = compare(tree[index], value);
        for(index >>= 1; index >= 1; index >>= 1) tree[index]= compare(
            tree[2 * index], tree[2 * index + 1]);
    }

    //BOTTOM - TOP
    T query(int first, int last){
        first += n, last += n;
        T ans = 0;
        while(first <= last){
            if(first % 2 == 1) ans = compare(ans, tree[first++]);
            if(last % 2 == 0) ans = compare(ans, tree[last--]);
            first >>= 1, last >>= 1;
        }
        return ans;
    }
};
```