El Bicho

DondeEstasCR7

19/02/2026

Índice

1. Algos

1.1. Binary Search

```
// binary search en array ordenado
int n, target; cin >> n >> target;
vector<int> a(n);
for (int i = 0; i < n; i++) cin >> a[i];

// encontrar primera posición >= target
int l = 0, r = n - 1, first_pos = n;
while (l <= r) {
  int mid = l + (r - l) / 2;
  if (a[mid] >= target) {
    first_pos = mid;
    r = mid - 1;
  } else {
    l = mid + 1;
  }
}

// encontrar última posición <= target
l = 0, r = n - 1;
int last_pos = -1;
while (l <= r) {
  int mid = l + (r - l) / 2;
  if (a[mid] <= target) {
    last_pos = mid;
    l = mid + 1;
  } else {
    r = mid - 1;
  }
}

// binary search en función monótona
function<bool(int)> check = [&](int x) {
  return true; // condición
};
l = 0, r = 1e9;
int ans = -1;
while (l <= r) {
  int mid = l + (r - l) / 2;
  if (check(mid)) {
    ans = mid;
    l = mid + 1; // o r = mid - 1 dependiendo del problema
  } else {
    r = mid - 1; // o l = mid + 1
  }
}
```

1.2. Fast Io

```
#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;
```

```cpp
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;
}
```

1.3. Sliding Window

```cpp
int n, k; cin >> n >> k;
vector<int> a(n);
for (int i = 0; i < n; i++) cin >> a[i];
```



```cpp
  }
}
vector<int> occurrences;
for (int i = pattern.size() + 1; i < combined.size(); i++) {
  if (z_combined[i] == pattern.size()) {
    occurrences.push_back(i - pattern.size() - 1);
  }
}
// z[i] = longitud del substring mas largo que empieza en i y es prefijo de s
// occurrences contiene las posiciones donde pattern aparece en text
```

```cpp
      cur = nodes[cur].next[idx];
    }
    return nodes[cur].count;
  }
};

// uso:
// Trie trie;
// int n; cin >> n;
// for (int i = 0; i < n; i++) {
//   string s; cin >> s;
//   trie.insert(s);
// }
// string query; cin >> query;
// bool exists = trie.search(query); // true si query existe en el trie
// int count = trie.count_prefix(query); // cantidad de strings que tienen query como prefijo

Trie trie;
trie.insert("hello");
trie.insert("hell");
bool found = trie.search("hello"); // true
int count = trie.count_prefix("hel"); // 2 (hello y hell)
```

11.3. Z Algorithm

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

string pattern, text;
cin >> pattern >> text;
string combined = pattern + "#" + text;
vector<int> z_combined(combined.size());
int l_combined = 0, r_combined = 0;
for (int i = 1; i < combined.size(); i++) {
  if (i <= r_combined) {
    z_combined[i] = min(r_combined - i + 1, z_combined[i - l_combined]);
  }
  while (i + z_combined[i] < combined.size() && combined[z_combined[i]] == combined[i +
      z_combined[i]]) {
    z_combined[i]++;
  }
  if (i + z_combined[i] - 1 > r_combined) {
    l_combined = i;
    r_combined = i + z_combined[i] - 1;
```



```cpp

// ventana deslizante de tamaño k
deque<int> dq;
for (int i = 0; i < n; i++) {
  while (!dq.empty() && dq.front() <= i - k) dq.pop_front();
  while (!dq.empty() && a[dq.back()] <= a[i]) dq.pop_back();
  dq.push_back(i);
  if (i >= k - 1) {
    // a[dq.front()] es el máximo en ventana [i-k+1, i]
  }
}

// mínimo en ventana de tamaño k
dq.clear();
for (int i = 0; i < n; i++) {
  while (!dq.empty() && dq.front() <= i - k) dq.pop_front();
  while (!dq.empty() && a[dq.back()] >= a[i]) dq.pop_back();
  dq.push_back(i);
  if (i >= k - 1) {
    // a[dq.front()] es el mínimo en ventana [i-k+1, i]
  }
}
```

1.4. Tablas Y Cotas

```
// 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
```



```
vector<int> pi_combined(combined.size());
for (int i = 1; i < combined.size(); i++) {
  int j = pi_combined[i - 1];
  while (j > 0 && combined[i] != combined[j]) {
    j = pi_combined[j - 1];
  }
  if (combined[i] == combined[j]) j++;
  pi_combined[i] = j;
}
vector<int> occurrences;
for (int i = pattern.size() + 1; i < combined.size(); i++) {
  if (pi_combined[i] == pattern.size()) {
    occurrences.push_back(i - 2 * pattern.size());
  }
}
// pi[i] = longitud del prefijo mas largo que es sufijo en s[0..i]
// occurrences contiene las posiciones donde pattern aparece en text
```

11.2. Trie

```
struct Trie {
  struct Node {
    vector<int> next;
    bool is_end;
    int count;
    Node() : next(26, -1), is_end(false), count(0) {}
  };
  vector<Node> nodes;
  Trie() { nodes.emplace_back(); }

  void insert(string& s) {
    int cur = 0;
    for (char c : s) {
      int idx = c - 'a';
      if (nodes[cur].next[idx] == -1) {
        nodes[cur].next[idx] = nodes.size();
        nodes.emplace_back();
      }
      cur = nodes[cur].next[idx];
      nodes[cur].count++;
    }
    nodes[cur].is_end = true;
  }

  bool search(string& s) {
    int cur = 0;
    for (char c : s) {
      int idx = c - 'a';
      if (nodes[cur].next[idx] == -1) return false;
      cur = nodes[cur].next[idx];
    }
    return nodes[cur].is_end;
  }

  int count_prefix(string& s) {
    int cur = 0;
    for (char c : s) {
      int idx = c - 'a';
      if (nodes[cur].next[idx] == -1) return 0;
```

```
    return 1;
}
```

10.4. Modular Slae

```
int gauss (vector < bitset<N> > a, int n, int m, bitset<N> & ans) {
    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        for (int i=row; i<n; ++i)
            if (a[i][col]) {
                swap (a[i], a[row]);
                break;
            }
        if (! a[row][col])
            continue;
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row && a[i][col])
                a[i] ^= a[row];
        ++row;
    }
        // The rest of implementation is the same as above
}
```

10.5. Simpson'S Integration

```
// Integration by Simpson's formula
const int N = 1000 * 1000; // number of steps (already multiplied by 2)

double simpson_integration(double a, double b){
    double h = (b - a) / N;
    double s = f(a) + f(b); // a = x_0 and b = x_2n
    for (int i = 1; i <= N - 1; ++i) { // Refer to final Simpson's formula
        double x = a + h * i;
        s += f(x) * ((i & 1) ? 4 : 2);
    }
    s *= h / 3;
    return s;
}
```

11. String

11.1. Kmp

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

string pattern, text;
cin >> pattern >> text;
string combined = pattern + "#" + text;
```



```
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)

// Límites 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
```

1.5. Two Pointers

```
int n, target; cin >> n >> target;
vector<int> a(n);
for (int i = 0; i < n; i++) cin >> a[i];

// encontrar subarray con suma = target
int l = 0, sum = 0;
for (int r = 0; r < n; r++) {
  sum += a[r];
  while (sum > target && l <= r) {
    sum -= a[l++];
  }
  if (sum == target) {
    // subarray [l, r] tiene suma = target
  }
}
```

```

// encontrar número de subarrays con suma <= target
l = 0, sum = 0;
long long count = 0;
for (int r = 0; r < n; r++) {
  sum += a[r];
  while (sum > target && l <= r) {
    sum -= a[l++];
  }
  count += r - l + 1;
}
// count = número de subarrays con suma <= target
```

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.

```
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 más 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 está 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))
// Número de ceros a la derecha (Posición del LSB, idx-0)
#define idxLastBit(x) __builtin_ctzll(x)
// Extrae el bit màs significativo 0100 (Most Significant Bit)
#define msb(S) (1LL << (63 - __builtin_clzll(S)))
// Posición del MSB (63 - ceros a la izquierda, idx-0)
#define idxFirstBit(x) (63 - __builtin_clzll(x))

#define countAllOnes(x) __builtin_popcountll(x)
// Apaga el último bit encendido (el menos significativo)
#define turnOffLastBit(S) ((S) & ((S) - 1))
// Enciende el último cero menos significativo
#define turnOnLastZero(S) ((S) | ((S) + 1))
// Apaga todos los bits encendidos más a la derecha consecutivos
#define turnOffLastConsecutiveBits(S) ((S) & ((S) + 1))
// Enciende los ceros consecutivos más a la derecha
```



```
    if (i != k)
        det = -det;
    det *= a[i][i];
    for (int j=i+1; j<n; ++j)
        a[i][j] /= a[i][i];
    for (int j=0; j<n; ++j)
        if (j != i && abs (a[j][i]) > EPS)
            for (int k=i+1; k<n; ++k)
                a[j][k] -= a[i][k] * a[j][i];
}

cout << det;
```

10.3. Linear Equations

```
// Gauss method for solving system of linear equations
const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be infinity or a big number

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

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

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

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

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

- **Suma de los primeros n cubos**

$$\sum_{i=1}^{n} i^3 = 1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$

- **Suma de cuadrados**

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

- **Suma de potencias cuartas**

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

- **Serie Geométrica (Finita)**

$$\sum_{i=0}^{n} a \cdot r^i = a + ar + ar^2 + \cdots + ar^n = \frac{a(r^{n+1}-1)}{r-1} \quad (r \neq 0, r \neq 1)$$

- **Serie Geométrica (Infinita)**

$$\sum_{i=0}^{\infty} a \cdot r^i = a + ar + ar^2 + \cdots = \frac{a}{1-r} \quad (|r| < 1)$$

- **Suma potencias de 1/2 (Infinita)**

$$\sum_{i=1}^{\infty} \left(\frac{1}{2}\right)^i = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots = 1$$

- **Suma $i \cdot 2^i$**

$$\sum_{i=1}^{n} i \cdot 2^i = 1 \cdot 2^1 + 2 \cdot 2^2 + 3 \cdot 2^3 + \cdots + n \cdot 2^n = (n-1)2^{n+1} + 2$$

- **Suma $i/2^i$**

$$\sum_{i=1}^{n} \frac{i}{2^i} = \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \cdots + \frac{n}{2^n} = 2 - \frac{n+2}{2^n}$$

10.2. Determinant Of A Matrix

```
// Calculating the determinant of a matrix by Gauss
const double EPS = 1E-9;
int n;
vector < vector<double> > a (n, vector<double> (n));

double det = 1;
for (int i=0; i<n; ++i) {
    int k = i;
    for (int j=i+1; j<n; ++j)
        if (abs (a[j][i]) > abs (a[k][i]))
            k = j;
    if (abs (a[k][i]) < EPS) {
        det = 0;
        break;
    }
    swap (a[i], a[k]);
```



```
#define turnOnLastConsecutiveZeroes(S) ((S) | ((S) - 1))

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

// Recorrer subconjuntos de un superconjunto (menos el vacío)
int b = 0b1011; // Representación 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

```
// 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; // aquí la división 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

```
// Devuelve el inverso modular de a mod MOD
// Usa el Teorema Pequeño de Fermat: a^(MOD-2) === a^(-1) (mod MOD)
// (válido 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);

```

```cpp
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;
}

// Permutación 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. Data Structures

4.1. Fenwick Tree

```cpp
struct FenwickTree {
  vector<long long> tree;
  int n;

  FenwickTree(int size) : n(size + 1) {
    tree.assign(n, 0);
  }

  void update(int idx, long long delta) {
    for (idx++; idx < n; idx += idx & -idx) {
      tree[idx] += delta;
    }
  }

  long long query(int idx) {
    long long sum = 0;
    for (idx++; idx > 0; idx -= idx & -idx) {
      sum += tree[idx];
    }
    return sum;
  }

  long long range_query(int l, int r) {
    return query(r) - query(l - 1);
  }
};

```



```cpp
        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';
    }
}
```

9.8. Sum Of Divisors

```cpp
//* 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;
}
```

10. Numerical Methods

10.1. Sumas Notables

- **Suma de los primeros n números naturales (Gauss)**

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

- **Suma de los primeros n números pares**

$$\sum_{i=1}^{n} 2 \cdot i = 2 + 4 + 6 + \cdots + 2n = n(n+1)$$

- **Suma de los primeros n números impares**

$$\sum_{i=1}^{n} (2 \cdot i - 1) = 1 + 3 + 5 + \cdots + (2n - 1) = n^2$$

```cpp

//* 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;
        }
    }
}
```

9.5. Potenciacion Binaria

```cpp
using ll = long long;
const int MAXN = 1e6;   // límite 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;
}
```

9.6. Sieve

```cpp
// 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;
      }
    }
  }
}
```

9.7. Sieve Bitset

```cpp
// 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;
```



```cpp
// uso:
// int n; cin >> n;
// FenwickTree ft(n);
// for (int i = 0; i < n; i++) {
//   long long x; cin >> x;
//   ft.update(i, x);
// }
// ft.update(idx, delta); // actualizar elemento en idx
// long long sum = ft.range_query(l, r); // suma en rango [l, r]
```

4.2. Find Two Numbers

```cpp
// "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;
      }
    }
  }
};
```

4.3. Segment Tree

```cpp
// "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); }
};
```

4.4. Sparse Table

```
int n; cin >> n;
vector<long long> a(n);
for (int i = 0; i < n; i++) cin >> a[i];
int k = log2(n) + 1;
vector<vector<long long>> st(n, vector<long long>(k));
for (int i = 0; i < n; i++) st[i][0] = a[i];
for (int j = 1; j < k; j++) {
  for (int i = 0; i + (1 << j) <= n; i++) {
    st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
  }
}
function<long long(int, int)> query = [&](int l, int r) {
  int j = log2(r - l + 1);
  return min(st[l][j], st[r - (1 << j) + 1][j]);
};
// query(l, r) = mínimo en rango [l, r] en O(1)
// cambiar min por max para máximo
// cambiar min por gcd para GCD en rango
```

5. Dp

5.1. Digit Dp Pattern

```
string pattern; cin >> pattern; // ejemplo: "xxxxx3xxxx" donde x = dígito libre
int n = pattern.size();
long long k; cin >> k; // modulo

```



```
}
```

9.3. 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;
}
```

9.4. 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;
}
```

```
            else swap(p.x, p.y);
        }
    }
    return edges;
}
```

9. Number Theory

9.1. 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';
    }
  }
};
```

9.2. 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);
```



```
vector<vector<vector<long long>>> dp(n, vector<vector<long long>>(k, vector<long long>(2,
    -1)));

function<long long(int, int, bool, bool)> solve = [&](int pos, int rem, bool tight, bool
    started) {
  if (pos == n) {
    return (started && rem == 0) ? 1LL : 0LL;
  }
  if (started && !tight && dp[pos][rem][tight ? 1 : 0] != -1) {
    return dp[pos][rem][tight ? 1 : 0];
  }
  long long res = 0;
  if (pattern[pos] != 'x' && pattern[pos] != 'X') {
    int fixed_digit = pattern[pos] - '0';
    bool new_tight = tight && (fixed_digit == 9);
    bool new_started = started || (fixed_digit > 0);
    int new_rem = (rem * 10 + fixed_digit) % k;
    res += solve(pos + 1, new_rem, new_tight, new_started);
  } else {
    int limit = tight ? 9 : 9;
    int start_digit = (pos == 0) ? 1 : 0; // primer dígito no puede ser 0
    for (int d = start_digit; d <= limit; d++) {
      bool new_tight = tight && (d == limit);
      bool new_started = started || (d > 0);
      int new_rem = (rem * 10 + d) % k;
      res += solve(pos + 1, new_rem, new_tight, new_started);
    }
  }
  if (started && !tight) {
    dp[pos][rem][tight ? 1 : 0] = res;
  }
  return res;
};

long long result = solve(0, 0, true, false);
// result = cantidad de números que siguen el patrón y son divisibles por k
// ejemplo: pattern = "xxxxx3xxxx", k = 7
// cuenta números tipo 1234534567 que son divisibles por 7
// x o X = dígito libre, cualquier otro carácter = dígito fijo
```

5.2. Digit Dp

```
string s; cin >> s; // número como string (puede ser muy grande, tipo 10^100)
int n = s.size();
long long k; cin >> k; // modulo

vector<vector<vector<long long>>> dp(n, vector<vector<long long>>(k, vector<long long>(2,
    -1)));

function<long long(int, int, bool, bool)> solve = [&](int pos, int rem, bool tight, bool
    started) {
  if (pos == n) {
    return (started && rem == 0) ? 1LL : 0LL;
  }
  if (started && !tight && dp[pos][rem][tight ? 1 : 0] != -1) {
    return dp[pos][rem][tight ? 1 : 0];
  }
  long long res = 0;
  int limit = tight ? (s[pos] - '0') : 9;
```

```cpp
  for (int d = 0; d <= limit; d++) {
    bool new_tight = tight && (d == limit);
    bool new_started = started || (d > 0);
    int new_rem = (rem * 10 + d) % k;
    res += solve(pos + 1, new_rem, new_tight, new_started);
  }
  if (started && !tight) {
    dp[pos][rem][tight ? 1 : 0] = res;
  }
  return res;
};

long long result = solve(0, 0, true, false);
// result = cantidad de números <= s que son divisibles por k
// para contar en rango [a, b]: result_b - result_a-1
// ejemplo: s = "1000000", k = 7 -> contar números de 0 a 1000000 divisibles por 7
```

5.3. Edit Distance

```cpp
string s1, s2; cin >> s1 >> s2;
int n = s1.size(), m = s2.size();
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], dp[i][j - 1], dp[i - 1][j - 1]});
    }
  }
}
// dp[n][m] = edit distance (mínimo número de operaciones: insertar, eliminar, reemplazar)
// para convertir s1 en s2
```

5.4. Knapsack

```cpp
int n, capacity; cin >> n >> capacity;
vector<int> weight(n), value(n);
for (int i = 0; i < n; i++) {
  cin >> weight[i] >> value[i];
}
vector<long long> dp(capacity + 1, 0);
for (int i = 0; i < n; i++) {
  for (int w = capacity; w >= weight[i]; w--) {
    dp[w] = max(dp[w], dp[w - weight[i]] + value[i]);
  }
}
// dp[capacity] = valor máximo que se puede obtener con capacidad máxima
// para version con items ilimitados, cambiar el loop: for (int w = weight[i]; w <= capacity; w++)
```

5.5. Lcs

```cpp
string s1, s2; cin >> s1 >> s2;
int n = s1.size(), m = s2.size();
vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));
for (int i = 1; i <= n; i++) {
  for (int j = 1; j <= m; j++) {
```



```cpp
        for (int v : adj[u]) {
            if (--indeg[v] == 0) {
                pq.push(v);
            }
        }
    }
    if ((int)order.size() != n) { return {}; }
    return order;
}
```

8. Manhattan Distance

8.1. Farthest Pair Of Points

```cpp
long long ans = 0;
for (int msk = 0; msk < (1 << d); msk++) {
    long long mx = LLONG_MIN, mn = LLONG_MAX;
    for (int i = 0; i < n; i++) {
        long long cur = 0;
        for (int j = 0; j < d; j++) {
            if (msk & (1 << j)) cur += p[i][j];
            else cur -= p[i][j];
        }
        mx = max(mx, cur);
        mn = min(mn, cur);
    }
    ans = max(ans, mx - mn);
}
```

8.2. Nearest Neighbor In Each Octant

```cpp
// Nearest Neighbor in each Octant in O(n log n)
struct point {
    long long x, y;
};

// Returns a list of edges in the format (weight, u, v).
// Passing this list to Kruskal algorithm will give the Manhattan MST.
vector<tuple<long long, int, int>> manhattan_mst_edges(vector<point> ps) {
    vector<int> ids(ps.size());
    iota(ids.begin(), ids.end(), 0);
    vector<tuple<long long, int, int>> edges;
    for (int rot = 0; rot < 4; rot++) { // for every rotation
        sort(ids.begin(), ids.end(), [&](int i, int j){
            return (ps[i].x + ps[i].y) < (ps[j].x + ps[j].y);
        });
        map<int, int, greater<int>> active; // (xs, id)
        for (auto i : ids) {
            for (auto it = active.lower_bound(ps[i].x); it != active.end();
            active.erase(it++)) {
                int j = it->second;
                if (ps[i].x - ps[i].y > ps[j].x - ps[j].y) break;
                assert(ps[i].x >= ps[j].x && ps[i].y >= ps[j].y);
                edges.push_back({(ps[i].x - ps[j].x) + (ps[i].y - ps[j].y), i, j});
            }
            active[ps[i].x] = i;
        }
        for (auto &p : ps) { // rotate
            if (rot & 1) p.x *= -1;
```

```
      if(ind[i] == -1) dfs(i);

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

7.13. Topo Sort Dfs

```
// O(N + M). 0-indexed. Retorna cualquier toposort válido (no necesariamente
    lexicográficamente mínima)
vector<int> topo_sort(vector<vector<int>>& adj){
    int n = adj.size();
    bool cycle = false;
    vector<int> topo, color(n); // 0 = no visitado, 1 = visitando, 2 = terminado

    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; // ciclo detectado
        }
        color[u] = 2;
        topo.push_back(u);
    };

    for (int i = 0; i < n; i++){
        if (color[i] == 0 && !cycle) dfs(i);
    }
    if (cycle) return {}; // no existe toposort
    reverse(topo.begin(), topo.end());
    return topo;
}
```

7.14. Topo Sort Kahns Bfs

```
// O((N + M)*logN). 0-indexed. Topological sort (Kahn BFS) con min-heap (lexicográficamente
    mínimo)
vector<int> topo_sort(int n, const vector<vector<int>>& adj) {
    vector<int> indeg(n, 0); // in-degree de cada nodo
    for (int u = 0; u < n; u++) {
        for (int v : adj[u]) {
            indeg[v]++;
        }
    }
    // min-heap para siempre sacar el nodo de menor índice
    priority_queue<int, vector<int>, greater<int>> pq;
    for (int u = 0; u < n; u++) {
        if (indeg[u] == 0) pq.push(u);
    }

    vector<int> order;
    order.reserve(n);
    while (!pq.empty()) {
        int u = pq.top();
        pq.pop();
        order.push_back(u);
```



```
    if (s1[i - 1] == s2[j - 1]) {
      dp[i][j] = dp[i - 1][j - 1] + 1;
    } else {
      dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
    }
  }
}
// dp[n][m] = longitud de LCS (Longest Common Subsequence)

string reconstruct_lcs() {
  string lcs = "";
  int i = n, j = m;
  while (i > 0 && j > 0) {
    if (s1[i - 1] == s2[j - 1]) {
      lcs += s1[i - 1];
      i--, j--;
    } else if (dp[i - 1][j] > dp[i][j - 1]) {
      i--;
    } else {
      j--;
    }
  }
  reverse(lcs.begin(), lcs.end());
  return lcs;
}
// lcs = string de la LCS
```

6. Geometry

6.1. 2D Geometry

Cookbook Geometría 2D - Operaciones con Puntos, Vectores, Líneas y Polígonos

PROBLEMAS TÍPICOS Y SUS SOLUCIONES

- **¿Un segmento toca un rectángulo?**

  Construir 4 lados del rectángulo como `Line`, usar `segIntersect(seg, lado_i)` para $i = 1..4$, y verificar con `between`/extremos dentro para cubrir el caso "segmento completamente dentro".

  Funciones clave: `segIntersect`, `between`, `Point/Line`.

- **¿Dos segmentos se cruzan?**

  Usar `segIntersect(l1, l2)` si contar colineales y tocar endpoints, o `segStrictlyIntersect(l1, l2)` si solo quieres cruce estricto.

- **Distancia mínima entre dos segmentos**

  `Real d = segDist(l1, l2);` Si $d = 0$ entonces se tocan o cruzan.

- **Distancia mínima de un punto a un segmento**

  `Real d = pointToSegDist(p, seg);`

- **Distancia mínima de un punto a una recta infinita**

  `Real d = pointToLineDist(p, line);`

- **¿Punto dentro de un polígono cualquiera?**

  `bool inside = pointInPoly(p, poly);`

- **Área de un polígono (ordenado CCW o CW)**

  `T twice = area(poly); Real A = fabsl((Real)twice) / 2.0;`

- **¿Cuándo un triángulo es degenerado?**

  Un triángulo con vértices a, b, c es degenerado (área $= 0$) si: (1) sus puntos son colineales `sign(cross(b-a, c-a)) == 0`, o (2) con lados a, b, c, falla la desigualdad triangular estricta ($a+b > c, a+c > b, b+c > a$), i.e., $a + b = c$ (se aplana).

- **¿Punto sobre el borde de un polígono?**

  Recorrer lados $[i, i+1]$, verificar si `pointOnSeg(p, Line(ai, aj))`.

- **¿Dos rectas infinitas se cruzan? Dame el punto**

  Primero verificar que no sean paralelas: `sign(cross(direction(l1), direction(l2))) != 0`, luego `P<Real>inter = lineIntersection(l1, l2);`

- **Ordenar vectores/puntos por ángulo alrededor del origen**

  `sort(v.begin(), v.end(), polar<i64>);` o usar lambda con `up()` y `cross()`.

- **Vector normal a un segmento**

  `P<T>d = direction(seg); P<T>n = rotate90(d);` (normal 90° CCW) o `P<Real>nu = normal(d);` (normal unitaria).

- **Detectar si un punto está a la izquierda/derecha de un borde**

  `int s = side(p, a, b);` donde $s > 0$ izquierda (CCW), $s < 0$ derecha (CW), $s = 0$ colineal.

- **Clasificar orientación de un polígono**

  `T twice = area(poly);` Si `twice >0` es CCW, si `twice <0` es CW.

- **Reflexión de un punto respecto a una recta**

  `P<Real>pr = reflection(p, line);`

- **Proyección de un punto sobre una recta**

  `P<Real>proj = projection(p, line);`

- **Intersección de rayo con segmento**

  `rayIntersect(rayo1, rayo2)` si ambos modelados como `Line` con origen en `l.l[0]` y dirección `l.l[1]-l.l[0]`. Para rayo vs segmento, combinar con `segIntersect` o rediseñar con `side()` y `dot()`.

- **Rotar un punto/vector alrededor del origen**

  `P<Real>rotado = rotate(p, angulo_en_radianes);` Para 90° CCW rápido: `rotate90(p);`

- **Calcular ángulo de un vector respecto al eje X**

  `Real ang = angle(p);` devuelve `atan2(p.y, p.x)` en radianes.

TIPS IMPORTANTES

- Usa `Point = P<i64>` para coords enteras, evita overflow en `cross/dot`
- Usa `P<Real>` (long double) para resultados con decimales (distancias, intersecciones)
- `EPS = 1e-9` para comparaciones de flotantes
- `sign()` y `cmp()` manejan tolerancia automáticamente
- Para convex hull y polígonos, mantén puntos en sentido CCW
- Verifica casos especiales: colineales, segmentos degenerados (mismo punto)
- Al leer entrada: `Point p; cin >>p;` (operador sobrecargado)


```
    }
  }
}
// mst_cost = costo del MST (Minimum Spanning Tree)
```

7.12. 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++)
```

```cpp
      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)];
  }
};
```

7.11. Prim

```cpp
int n, m; cin >> n >> m;
vector<vector<pair<int, long long>>> adj(n + 1);
for (int i = 0; i < m; i++) {
  int u, v; cin >> u >> v;
  long long w; cin >> w;
  adj[u].emplace_back(v, w);
  adj[v].emplace_back(u, w);
}
vector<bool> vis(n + 1);
pqg<pair<long long, int>> pq;
pq.push({0LL, 1});
long long mst_cost = 0;
while (!pq.empty()) {
  auto [w, u] = pq.top(); pq.pop();
  if (vis[u]) continue;
  vis[u] = true;
  mst_cost += w;
  for (auto [v, weight] : adj[u]) {
    if (!vis[v]) {
      pq.push({weight, v});
```



```cpp
// ==================== Tipos base ====================
// - i64: reemplazo al long long
// - Real: para cálculos con flotantes de más precisión (distancias, intersecciones, etc.).
using i64  = long long;
using Real = long double;
constexpr Real EPS = 1e-9;

// ==================== sign / cmp ====================
//  - sign(x): devuelve -1, 0, 1 (según x < 0, x == 0, x > 0)
//  - cmp(a, b): compara a y b con tolerancia (para Real).
//    cmp(a, b) == 0 -> a "igual" a b, cmp(a, b) < 0 -> a < b, etc.
template <typename T>
int sign(T x) {
  return (x > 0) - (x < 0);
}
int sign(Real x) {
  return (x > EPS) - (x < -EPS);
}

template <typename T>
int cmp(T a, T b) {
  return sign(a - b);
}

// ==================== Punto y Línea ====================
//  - con T = i64 es usual en problemas de coordenadas enteras.
//  - P<T>: punto/vector en 2D con componentes de tipo T.
//  - L<T>: línea o segmento definido por dos puntos (l[0], l[1]).
template <typename T>
struct P {
  T x = 0, y = 0;
  P(T x = 0, T y = 0) : x(x), y(y) {}
  friend istream& operator>>(istream &is, P &p) { return is >> p.x >> p.y; }
  friend ostream& operator<<(ostream &os, P p) { return os << p.x << ' ' << p.y; }
  friend bool operator==(P a, P b) { return cmp(a.x, b.x) == 0 && cmp(a.y, b.y) == 0; }
  friend bool operator!=(P a, P b) { return !(a == b); }
  P operator-() const { return P(-x, -y); }
  P& operator+=(P a) {
    x += a.x; y += a.y;
    return *this;
  }
  P& operator-=(P a) {
    x -= a.x; y -= a.y;
    return *this;
  }
  P& operator*=(T d) {
    x *= d; y *= d;
    return *this;
  }
  P& operator/=(T d) {
    x /= d; y /= d;
    return *this;
  }
  friend P operator+(P a, P b) { return P(a) += b; }
  friend P operator-(P a, P b) { return P(a) -= b; }
  friend P operator*(P a, T d) { return P(a) *= d; }
  friend P operator/(P a, T d) { return P(a) /= d; }
  friend bool operator<(P a, P b) {
    int sx = cmp(a.x, b.x);
```

```
    return (sx != 0 ? sx == -1 : cmp(a.y, b.y) == -1);
  }
};

template <typename T>
struct L {
  array<P<T>, 2> l;
  L(P<T> a = {}, P<T> b = {}) : l{a, b} {}
};

// ==================== Operaciones vectoriales básicas ====================
//  - dot(a, b): producto escalar.
//  - cross(a, b): producto cruzado escalar (a.x * b.y - a.y * b.x).
//  - cross(p, a, b): cross(a - p, b - p)  orientación de p respecto a ab.
//  - square(a): |a|^2.
//  - dist2(a, b): |a-b|^2, sin sqrt.
//  - length(a): |a|.
//  - dist(a, b): distancia euclidiana entre a y b.
template <typename T>
T dot(P<T> a, P<T> b) { return a.x * b.x + a.y * b.y; }
template <typename T>
T cross(P<T> a, P<T> b) { return a.x * b.y - a.y * b.x; }
template <typename T>
T cross(P<T> p, P<T> a, P<T> b) { return cross(a - p, b - p); }
template <typename T>
T square(P<T> a) { return dot(a, a); }
template <typename T>
T dist2(P<T> a, P<T> b) { return square(a - b); }
template <typename T>
Real length(P<T> a) { return sqrtl(square(a)); }
template <typename T>
Real dist(P<T> a, P<T> b) { return length(a - b); }

// ==================== Direcciones, ángulos, normales ====================
//  - normal(a): vector unitario en dirección de a.
//  - up(a): true si el vector está en el semiplano "de arriba" (para ordenar por ángulo).
//  - polar(a, b): criterio de orden por ángulo polar (para sort).
//  - parallel(a, b): vectores paralelos.
//  - sameDirection(a, b): vectores paralelos y apuntando en misma dirección.
//  - angle(p): atan2(y, x).
//  - rotate90(p): rota 90° CCW (útil para una normal rápida).
//  - rotate(p, ang): rota un vector por un ángulo ang en radianes.
template <typename T>
P<Real> normal(P<T> a) {
  Real len = length(a);
  return P<Real>(a.x / len, a.y / len);
}
template <typename T>
bool up(P<T> a) {
  return sign(a.y) > 0 || (sign(a.y) == 0 && sign(a.x) > 0);
}
// 3 colinear? recuerda remover (0,0) si lo usas en ordenamientos polares
template <typename T>
bool polar(P<T> a, P<T> b) {
  bool ua = up(a), ub = up(b);
  return ua != ub ? ua : sign(cross(a, b)) == 1;
}
template <typename T>
bool parallel(P<T> a, P<T> b) {
```



```
  }
  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;
  }
};

int n, m; cin >> n >> m;
vector<tuple<long long, int, int>> edges;
for (int i = 0; i < m; i++) {
  int u, v; cin >> u >> v;
  long long w; cin >> w;
  edges.emplace_back(w, u, v);
}
sort(edges.begin(), edges.end());
DSU dsu(n);
long long mst_cost = 0;
for (auto [w, u, v] : edges) {
  if (dsu.find(u) != dsu.find(v)) {
    dsu.merge(u, v);
    mst_cost += w;
  }
}
// mst_cost = costo del MST (Minimum Spanning Tree)
```

7.10. 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++){
```

```cpp
    }
  }
}
for (int u = 1; u <= N; u++) {
  cout << dis[u] << " ";
}
```

7.7. 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;
  }
};
```

7.8. Floyd Warshall

```cpp
int n; cin >> n;
vector<vector<long long>> dist(n + 1, vector<long long>(n + 1, inf));
for (int i = 1; i <= n; i++) dist[i][i] = 0;
int m; cin >> m;
for (int i = 0; i < m; i++) {
  int u, v; cin >> u >> v;
  long long w; cin >> w;
  dist[u][v] = min(dist[u][v], w);
  dist[v][u] = min(dist[v][u], w);
}
for (int k = 1; k <= n; k++) {
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
      if (dist[i][k] != inf && dist[k][j] != inf) {
        dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
      }
    }
  }
}
// dist[i][j] = distancia minima entre nodo i y nodo j
```

7.9. Kruskal

```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;
```



```cpp
  return sign(cross(a, b)) == 0;
}
template <typename T>
bool sameDirection(P<T> a, P<T> b) {
  return sign(cross(a, b)) == 0 && sign(dot(a, b)) == 1;
}
template <typename T>
Real angle(P<T> p) {
  return atan2((Real)p.y, (Real)p.x);
}
template <typename T>
P<T> rotate90(P<T> p) {
  return P<T>(-p.y, p.x);
}
P<Real> rotate(P<Real> p, Real ang) {
  return P<Real>(p.x * cosl(ang) - p.y * sinl(ang),
                 p.x * sinl(ang) + p.y * cosl(ang));
}

// ==================== Dirección de una línea ====================
//  - direction(l): vector l.l[1] - l.l[0] (dirección del segmento/recta).
//  - parallel(l1, l2) / sameDirection(l1, l2): igual que para vectores pero con líneas.
template <typename T>
P<T> direction(L<T> l) {
  return l.l[1] - l.l[0];
}
template <typename T>
bool parallel(L<T> l1, L<T> l2) {
  return sameDirection(direction(l1), direction(l2));
}
template <typename T>
bool sameDirection(L<T> l1, L<T> l2) {
  return sameDirection(direction(l1), direction(l2));
}

// ==================== Proyección, reflexión, distancias a recta ====================
//  - projection(p, l): proyección ortogonal de p sobre la recta (infinita) l.
//  - reflection(p, l): reflejo de p respecto a la recta l.
//  - pointToLineDist(p, l): distancia mínima de p a la recta infinita que pasa por l.
P<Real> projection(P<Real> p, L<Real> l) {
  auto d = direction(l);
  return l.l[0] + d * (dot(p - l.l[0], d) / (Real)square(d));
}
P<Real> reflection(P<Real> p, L<Real> l) {
  return projection(p, l) * 2 - p;
}
template <typename T>
Real pointToLineDist(P<T> p, L<T> l) {
  if (l.l[0] == l.l[1]) return dist(p, l.l[0]);
  return fabsl(cross(l.l[0] - l.l[1], l.l[0] - p)) / length(direction(l));
}

// ==================== Intersección de líneas (rectas infinitas) ====================
//  - lineIntersection(l1, l2): punto de intersección de las rectas infinitas
//    definidas por l1 y l2.
//  - OJO: no chequea si son paralelas; debes verificar antes que cross != 0.
template <typename T>
P<Real> lineIntersection(L<T> l1, L<T> l2) {
  auto d1 = direction(l1);
```

```cpp
  auto d2 = direction(l2);
  auto num = (Real)cross(d2, l1.l[0] - l2.l[0]);
  auto den = (Real)cross(d2, d1);
  return P<Real>(l1.l[0]) - d1 * (num / den);
}

// ==================== Side / Between ====================
//  - side(p, a, b): orientación de p respecto al vector ab.
//      > 0: izquierda (CCW), < 0: derecha (CW), 0: colineal.
//  - side(p, l): igual que antes pero con línea l.
//  - between(m, l, r): true si m está entre l y r (incluyendo bordes).
template <typename T>
int side(P<T> p, P<T> a, P<T> b) {
  return sign(cross(p, a, b));
}
template <typename T>
int side(P<T> p, L<T> l) {
  return side(p, l.l[0], l.l[1]);
}
template <typename T>
bool between(T m, T l, T r) {
  return cmp(l, m) == 0 || cmp(m, r) == 0 || (l < m) != (r < m);
}

// ==================== Puntos sobre segmento ====================
//  - pointOnSeg(p, l): true si p está sobre el segmento l (incluye endpoints).
//  - pointStrictlyOnSeg(p, l): true si p está sobre el segmento pero no en los endpoints.
template <typename T>
bool pointOnSeg(P<T> p, L<T> l) {
  return side(p, l) == 0 &&
         between(p.x, l.l[0].x, l.l[1].x) &&
         between(p.y, l.l[0].y, l.l[1].y);
}
template <typename T>
bool pointStrictlyOnSeg(P<T> p, L<T> l) {
  if (side(p, l) != 0) return false;
  auto d = direction(l);
  return sign(dot(p - l.l[0], d)) * sign(dot(p - l.l[1], d)) < 0;
}

// ==================== Solapamiento de intervalos ====================
//  - overlap(l1, r1, l2, r2): true si [l1, r1] y [l2, r2] se solapan (1D).
template <typename T>
bool overlap(T l1, T r1, T l2, T r2) {
  if (l1 > r1) swap(l1, r1);
  if (l2 > r2) swap(l2, r2);
  return cmp(r1, l2) != -1 && cmp(r2, l1) != -1;
}

// ==================== Intersección de segmentos / rayos ====================
//  - segIntersect(l1, l2): true si los segmentos se tocan o cortan
//     (incluye colineales solapados y tocar en vértices).
//  - segStrictlyIntersect(l1, l2): true si se cortan estrictamente
//     (no cuenta tocar solo en un endpoint).
//  - rayIntersect(l1, l2): considera l1 y l2 como rayos, intersectan "hacia adelante"
//     (no cuenta si solo coincide el origen).
template <typename T>
bool segIntersect(L<T> l1, L<T> l2) {
  auto [p1, p2] = l1.l;
```



```cpp

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

7.5. 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));
int dx[4] = {-1, 1, 0, 0}, dy[4] = {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 && 0 <= ny && nx < N && 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;
```

7.6. Dijkstra

```cpp
int N, M; cin >> N >> M;
vector<vector<pair<int, long long>>> adj(N + 1);
for (int i = 0; i < M; i++) {
  int u, v; cin >> u >> v;
  long long w; cin >> w;
  adj[u].emplace_back(v, w);
}
vector<long long> dis(N + 1, inf);
pqg<pair<long long, int>> pq;
dis[1] = 0;
pq.push({0LL, 1});
while (!pq.empty()) {
  auto [d, node] = pq.top(); pq.pop();
  if (dis[node] != d) continue;
  for (auto [v, w] : adj[node]) {
    if (d + w < dis[v]) {
      dis[v] = d + w;
      pq.push({dis[v], v});
```

```cpp
      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);
}
```

7.3. Bipartite

```cpp
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) << ' ';
```

7.4. Dfs

```cpp
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);
  }
};
```



```cpp
  auto [q1, q2] = l2.l;
  return overlap(p1.x, p2.x, q1.x, q2.x) &&
        overlap(p1.y, p2.y, q1.y, q2.y) &&
        side(p1, l2) * side(p2, l2) <= 0 &&
        side(q1, l1) * side(q2, l1) <= 0;
}
template <typename T>
bool segStrictlyIntersect(L<T> l1, L<T> l2) {
  auto [p1, p2] = l1.l;
  auto [q1, q2] = l2.l;
  return side(p1, l2) * side(p2, l2) < 0 &&
        side(q1, l1) * side(q2, l1) < 0;
}
template <typename T>
bool rayIntersect(L<T> l1, L<T> l2) {
  auto v1 = direction(l1);
  auto v2 = direction(l2);
  int x = sign(cross(v1, v2));
  return x != 0 && side(l1.l[0], l2) == x && side(l2.l[0], l1) == -x;
}

// ==================== Distancias punto-segmento / segmento-segmento ====================
//  - pointToSegDist(p, l): distancia mínima de p al segmento l.
//  - segDist(l1, l2): distancia mínima entre dos segmentos (0 si se intersectan).
template <typename T>
Real pointToSegDist(P<T> p, L<T> l) {
  auto d = direction(l);
  if (sign(dot(p - l.l[0], d)) >= 0 && sign(dot(p - l.l[1], d)) <= 0) {
    return pointToLineDist(p, l);
  } else {
    return min(dist(p, l.l[0]), dist(p, l.l[1]));
  }
}
template <typename T>
Real segDist(L<T> l1, L<T> l2) {
  if (segIntersect(l1, l2)) return 0;
  return min({
    pointToSegDist(l1.l[0], l2),
    pointToSegDist(l1.l[1], l2),
    pointToSegDist(l2.l[0], l1),
    pointToSegDist(l2.l[1], l1)
  });
}

// ==================== Área de polígono y punto en polígono ====================
//  - area(a): devuelve 2 * área con signo del polígono a (ordenado).
//      >0  CCW, <0  CW, abs(area)/2.0  área real.
//  - pointInPoly(p, a): true si p está dentro o sobre el borde del polígono a
//      (no necesariamente convexo).
template <typename T>
T area(vector<P<T>> a) {
  T res = 0;
  int n = (int)a.size();
  for (int i = 0; i < n; i++) {
    res += cross(a[i], a[(i + 1) % n]);
  }
  return res;
}
template <typename T>
```

```
bool pointInPoly(P<T> p, vector<P<T>> a) {
  int n = (int)a.size(), res = 0;
  for (int i = 0; i < n; i++) {
    P<T> u = a[i], v = a[(i + 1) % n];
    if (pointOnSeg(p, L<T>(u, v))) return true;
    if (cmp(u.y, v.y) <= 0) swap(u, v);
    if (cmp(p.y, u.y) > 0 || cmp(p.y, v.y) <= 0) continue;
    res ^= cross(p, u, v) > 0;
  }
  return res;
}

// ==================== Aliases finales ====================
//  - Point  = P<i64>    puntos con coordenadas enteras.
//  - Line   = L<i64>    segmentos/líneas con endpoints enteros.
//  - Usa Real (long double) para distancias si necesitas precisión extra.
using Point = P<i64>;
using Line  = L<i64>;

// ejemplo de uso rápido:
// Point a, b; cin >> a >> b;
// Line seg(a, b);
// if (segIntersect(seg, Line(Point(0,0), Point(10,0)))) { ... }
```

6.2. Convex Hull

```
struct Point {
  long long x, y;
  Point(long long x = 0, long long y = 0) : x(x), y(y) {}
  Point operator-(const Point& p) const { return Point(x - p.x, y - p.y); }
  long long cross(const Point& p) const { return x * p.y - y * p.x; }
  long long cross(const Point& a, const Point& b) const { return (a - *this).cross(b - *this
      ); }
  bool operator<(const Point& p) const { return x < p.x || (x == p.x && y < p.y); }
  bool operator==(const Point& p) const { return x == p.x && y == p.y; }
};

vector<Point> convex_hull(vector<Point>& points) {
  int n = points.size();
  if (n <= 1) return points;
  sort(points.begin(), points.end());
  vector<Point> hull;
  for (int i = 0; i < n; i++) {
    // Si quiero incluir coords colineales ( < 0), si no lo quiero ( <= 0)
    while (hull.size() >= 2 && hull[hull.size() - 2].cross(hull.back(), points[i]) <= 0) {
      hull.pop_back();
    }
    hull.push_back(points[i]);
  }
  int lower = hull.size();
  for (int i = n - 2; i >= 0; i--) {
    // Si quiero incluir coords colineales ( < 0), si no lo quiero ( <= 0)
    while (hull.size() > lower && hull[hull.size() - 2].cross(hull.back(), points[i]) <= 0)
        {
      hull.pop_back();
    }
    hull.push_back(points[i]);
  }
```



```
  hull.pop_back();
  return hull;
}

// uso:
// int n; cin >> n;
// vector<Point> points(n);
// for (int i = 0; i < n; i++) {
//   cin >> points[i].x >> points[i].y;
// }
// vector<Point> hull = convex_hull(points);
// hull contiene los puntos del convex hull en orden counter-clockwise (CCW / Antihorario)
// reverse(hull) para obtenerlo en orden CW / horario
```

7. Graph

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

7.1. Bellman Ford

```
int n, m; cin >> n >> m;
vector<tuple<int, int, long long>> edges;
for (int i = 0; i < m; i++) {
  int u, v; cin >> u >> v;
  long long w; cin >> w;
  edges.emplace_back(u, v, w);
}
vector<long long> dist(n + 1, inf);
dist[1] = 0;
for (int i = 0; i < n - 1; i++) {
  for (auto [u, v, w] : edges) {
    if (dist[u] != inf) {
      dist[v] = min(dist[v], dist[u] + w);
    }
  }
}
bool has_negative_cycle = false;
for (auto [u, v, w] : edges) {
  if (dist[u] != inf && dist[u] + w < dist[v]) {
    has_negative_cycle = true;
    break;
  }
}
// dist[u] = distancia minima desde nodo 1 hasta u
// has_negative_cycle = true si hay ciclo negativo alcanzable desde nodo 1
```

7.2. Bfs

```
vector<bool> vis(n+1);
queue<int> q;
function<void(int)> bfs = [&](int start) {
  vis[start] = true;
  q.push(start);
  int levels = 1;
  while (!q.empty()) {
    int sz = q.size();
    levels++;
    while (sz--) {
```