El Bicho

DondeEstasCR7

15/11/2025

Contents

1 Algos

1.1 Binary Search

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

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

// binary search en funcion monotona
function<bool(int)> check = [&](int x) {
  return true; // condicion
};
l = 0, r = 1e9;
int ans = -1;
while (l <= r) {
  int mid = (l + r) / 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

```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;
```

```cpp
 8  using namespace __gnu_pbds;
 9  template <class T>
10  using ordered_set = tree<T, null_type, less_equal<T>, rb_tree_tag,
        tree_order_statistics_node_update>;
11
12  #define pb push_back
13  #define sz(a) ((int)(a).size())
14  #define ff first
15  #define ss second
16  #define all(a) (a).begin(), (a).end()
17  #define allr(a) (a).rbegin(), (a).rend()
18  #define approx(a) fixed << setprecision(a)
19
20  template <class T> void read(vector<T> &v);
21  template <class F, class S> void read(pair<F, S> &p);
22  template <class T, size_t Z> void read(array<T, Z> &a);
23  template <class T> void read(T &x) {cin >> x;}
24  template <class R, class... T> void read(R& r, T&... t){read(r); read(t...);};
25  template <class T> void read(vector<T> &v) {for(auto& x : v) read(x);}
26  template <class F, class S> void read(pair<F, S> &p) {read(p.ff, p.ss);}
27  template <class T, size_t Z> void read(array<T, Z> &a) { for(auto &x : a) read(x); }
28
29  template <class F, class S> void pr(const pair<F, S> &x);
30  template <class T> void pr(const T &x) {cout << x;}
31  template <class R, class... T> void pr(const R& r, const T&... t) {pr(r); pr(t...);}
32  template <class F, class S> void pr(const pair<F, S> &x) {pr("{", x.ff, ",␣", x.ss, "}\n");}
33  void ps() {pr("\n");}
34  template <class T> void ps(const T &x) {pr(x); ps();}
35  template <class T> void ps(vector<T> &v) {for(auto& x : v) pr(x, '␣'); ps();}
36  template <class T, size_t Z> void ps(const array<T, Z> &a) { for(auto &x : a) pr(x, '␣'); ps
        (); }
37  template <class F, class S> void ps(const pair<F, S> &x) {pr(x.ff, '␣', x.ss); ps();}
38  template <class R, class... T> void ps(const R& r,  const T &...t) {pr(r, '␣'); ps(t...);}
39
40  using ll = long long;
41  const double PI = 3.141592653589793;
42  const ll MX = 1e9 + 1;
43
44  void solve() {
45
46  }
47
48  int main() {
49    cpu();
50
51    int t = 1;
52    //cin >> t;
53    while (t--) {
54      solve();
55    }
56
57    return 0;
58  }
```

1.3   Sliding Window

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

```cpp
       z_combined[i]]) {
28     z_combined[i]++;
29    }
30    if (i + z_combined[i] - 1 > r_combined) {
31      l_combined = i;
32      r_combined = i + z_combined[i] - 1;
33    }
34  }
35  vector<int> occurrences;
36  for (int i = pattern.size() + 1; i < combined.size(); i++) {
37    if (z_combined[i] == pattern.size()) {
38      occurrences.push_back(i - pattern.size() - 1);
39    }
40  }
41  // z[i] = longitud del substring mas largo que empieza en i y es prefijo de s
42  // occurrences contiene las posiciones donde pattern aparece en text
```

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

```cpp
// ventana deslizante de tamano 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 maximo en ventana [i-k+1, i]
  }
}

// minimo en ventana de tamano 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 minimo en ventana [i-k+1, i]
  }
}
```

1.4 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
```

```
34  pi(10^11) = 4.118.054.813
35  pi(10^12) = 37.607.912.018
36
37  // Cantidad de divisores
38  sigma0(60) = 12 -> [1, 2, 3, 4, 6, 10, 12, 15, 20, 30, 60]
39  sigma0(120) = 16 -> [1, 2, 3, 4, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60, 120]
40  sigma0(180) = 18 -> [1, 2, 3, 4, 5, 6, 9, 10, 12, 15, 18, 20, 30, 36, 60, 90, 180]
41  sigma0(240) = 20 -> [1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24, 30, 40, 60, 80, 120, 240]
42  sigma0(360) = 24
43  sigma0(720) = 30
44  sigma0(840) = 32
45  sigma0(1.260) = 36
46  sigma0(1.680) = 40
47  sigma0(10.080) = 72
48  sigma0(15.120) = 80
49  sigma0(50.400) = 108
50  sigma0(83.160) = 128
51  sigma0(110.880) = 144
52  sigma0(498.960) = 200
53  sigma0(554.400) = 216
54  sigma0(1.081.080) = 256
55  sigma0(1.441.440) = 288
56  sigma0(4.324.320) = 384
57  sigma0(8.648.640) = 448
58
59  // Suma de divisores
60  sigma1(96) = 252 -> [1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 96]
61  sigma1(108) = 280 -> [1, 2, 3, 4, 6, 9, 12, 18, 27, 36, 54, 108]
62  sigma1(120) = 360 -> [1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60, 120]
63  sigma1(144) = 403 -> [1, 2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 36, 48, 72, 144]
64  sigma1(168) = 480
65  sigma1(960) = 3.048
66  sigma1(1.008) = 3.224
67  sigma1(1.080) = 3.600
68  sigma1(1.200) = 3.844
69  sigma1(4.620) = 16.128
70  sigma1(4.680) = 16.380
71  sigma1(5.040) = 19.344
72  sigma1(5.760) = 19.890
73  sigma1(8.820) = 31.122
74  sigma1(9.240) = 34.560
75  sigma1(10.080) = 39.312
76  sigma1(10.920) = 40.320
77  sigma1(32.760) = 131.040
78  sigma1(35.280) = 137.826
79  sigma1(36.960) = 145.152
80  sigma1(37.800) = 148.800
81  sigma1(60.480) = 243.840
82  sigma1(64.680) = 246.240
83  sigma1(65.520) = 270.816
84  sigma1(70.560) = 280.098
85  sigma1(95.760) = 386.880
86  sigma1(98.280) = 403.200
87  sigma1(100.800) = 409.448
88  sigma1(491.400) = 2.083.200
89  sigma1(498.960) = 2.160.576
90  sigma1(514.080) = 2.177.280
91  sigma1(982.800) = 4.305.280
92  sigma1(997.920) = 4.390.848
```

```
10    pi[i] = j;
11  }
12
13  string pattern, text;
14  cin >> pattern >> text;
15  string combined = pattern + "#" + text;
16  vector<int> pi_combined(combined.size());
17  for (int i = 1; i < combined.size(); i++) {
18    int j = pi_combined[i - 1];
19    while (j > 0 && combined[i] != combined[j]) {
20      j = pi_combined[j - 1];
21    }
22    if (combined[i] == combined[j]) j++;
23    pi_combined[i] = j;
24  }
25  vector<int> occurrences;
26  for (int i = pattern.size() + 1; i < combined.size(); i++) {
27    if (pi_combined[i] == pattern.size()) {
28      occurrences.push_back(i - 2 * pattern.size());
29    }
30  }
31  // pi[i] = longitud del prefijo mas largo que es sufijo en s[0..i]
32  // occurrences contiene las posiciones donde pattern aparece en text
```

11.2   Trie

```
1   struct Trie {
2     struct Node {
3       vector<int> next;
4       bool is_end;
5       int count;
6       Node() : next(26, -1), is_end(false), count(0) {}
7     };
8     vector<Node> nodes;
9     Trie() { nodes.emplace_back(); }
10
11    void insert(string& s) {
12      int cur = 0;
13      for (char c : s) {
14        int idx = c - 'a';
15        if (nodes[cur].next[idx] == -1) {
16          nodes[cur].next[idx] = nodes.size();
17          nodes.emplace_back();
18        }
19        cur = nodes[cur].next[idx];
20        nodes[cur].count++;
21      }
22      nodes[cur].is_end = true;
23    }
24
25    bool search(string& s) {
26      int cur = 0;
27      for (char c : s) {
28        int idx = c - 'a';
29        if (nodes[cur].next[idx] == -1) return false;
30        cur = nodes[cur].next[idx];
31      }
32      return nodes[cur].is_end;
33    }
```

```
37        if (abs (sum - a[i][m]) > EPS)
38            return 0;
39    }
40
41    for (int i=0; i<m; ++i)
42        if (where[i] == -1)
43            return INF;
44    return 1;
45 }
```

10.3   Gauss Solving Modular Slae

```
1  int gauss (vector < bitset<N> > a, int n, int m, bitset<N> & ans) {
2      vector<int> where (m, -1);
3      for (int col=0, row=0; col<m && row<n; ++col) {
4          for (int i=row; i<n; ++i)
5              if (a[i][col]) {
6                  swap (a[i], a[row]);
7                  break;
8              }
9          if (! a[row][col])
10             continue;
11         where[col] = row;
12
13         for (int i=0; i<n; ++i)
14             if (i != row && a[i][col])
15                 a[i] ^= a[row];
16         ++row;
17     }
18         // The rest of implementation is the same as above
19 }
```

10.4   Integration By Simpson'S Formula

```
1  const int N = 1000 * 1000; // number of steps (already multiplied by 2)
2
3  double simpson_integration(double a, double b){
4      double h = (b - a) / N;
5      double s = f(a) + f(b); // a = x_0 and b = x_2n
6      for (int i = 1; i <= N - 1; ++i) { // Refer to final Simpson's formula
7          double x = a + h * i;
8          s += f(x) * ((i & 1) ? 4 : 2);
9      }
10     s *= h / 3;
11     return s;
12 }
```

11   String

11.1   Kmp

```
1  string s; cin >> s;
2  int n = s.size();
3  vector<int> pi(n);
4  for (int i = 1; i < n; i++) {
5    int j = pi[i - 1];
6    while (j > 0 && s[i] != s[j]) {
7      j = pi[j - 1];
8    }
9    if (s[i] == s[j]) j++;
```

```
93  sigma1(1.048.320) = 4.464.096
94  sigma1(4.979.520) = 22.189.440
95  sigma1(4.989.600) = 22.686.048
96  sigma1(5.045.040) = 23.154.768
97  sigma1(9.896.040) = 44.323.200
98  sigma1(9.959.040) = 44.553.600
99  sigma1(9.979.200) = 45.732.192
100
101 // Factoriales
102 0! = 1   (int)
103 1! = 1
104 2! = 2
105 3! = 6
106 4! = 24
107 5! = 120
108 6! = 720
109 7! = 5.040
110 8! = 40.320
111 9! = 362.880
112 10! = 3.628.800
113 11! = 39.916.800
114 12! = 479.001.600    (int)
115 13! = 6.227.020.800    (ll)
116 14! = 87.178.291.200
117 15! = 1.307.674.368.000
118 16! = 20.922.789.888.000
119 17! = 355.687.428.096.000
120 18! = 6.402.373.705.728.000
121 19! = 121.645.100.408.832.000
122 20! = 2.432.902.008.176.640.000    (ll)
123 21! = 51.090.942.171.709.400.000    (__int128_t)
124
125 // Limites de enteros
126 max signed char = 127
127 max unsigned char = 255
128 max signed int = 2.147.483.647
129 max unsigned int = 4.294.967.295
130 max signed long long = 9.223.372.036.854.775.807
131 max unsigned long long = 18.446.744.073.709.551.615
132 max signed __int128_t = 170.141.183.460.469.231.731.687.303.715.884.105.727
133 max unsigned __int128_t = 340.282.366.920.938.463.463.374.607.431.768.211.456
```

1.5   Two Pointers

```
1  int n, target; cin >> n >> target;
2  vector<int> a(n);
3  for (int i = 0; i < n; i++) cin >> a[i];
4
5  // encontrar subarray con suma = target
6  int l = 0, sum = 0;
7  for (int r = 0; r < n; r++) {
8    sum += a[r];
9    while (sum > target && l <= r) {
10     sum -= a[l++];
11   }
12   if (sum == target) {
13     // subarray [l, r] tiene suma = target
14   }
15 }
```

```
16
17   // encontrar numero de subarrays con suma <= target
18   l = 0, sum = 0;
19   long long count = 0;
20   for (int r = 0; r < n; r++) {
21     sum += a[r];
22     while (sum > target && l <= r) {
23       sum -= a[l++];
24     }
25     count += r - l + 1;
26   }
27   // count = numero 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.

```
1    using ull = unsigned long long;
2    const ull UNSIGNED_LL_MAX = 18'446'744'073'709'551'615;
3    // Verifica si S es potencia de dos (y distinto de cero)
4    #define isPowerOfTwo(S) ((S) && !((S) & ((S) - 1)))
5    // Retorna la potencia de dos mas cercana a S
6    #define nearestPowerOfTwo(S) (1LL << lround(log2(S)))
7    // Calcula S % N cuando N es potencia de dos
8    #define modulo(S, N) ((S) & ((N) - 1))
9
10   // Verifica si el bit esta encendido (bit en 1)
11   #define isOn(S, i) ((S) & (1LL<<(i)))
12   // Enciende el bit (Lo pone en 1)
13   #define setBit(S, i) ((S) |= (1LL<<(i)))
14   // Apaga el bit (Lo pone en 0)
15   #define clearBit(S, i) ((S) &= ~(1LL<<(i)))
16   // Invierte el estado del bit (0 <-> 1)
17   #define toggleBit(S, i) ((S) ^= (1LL<<(i)))
18   // Enciende los primeros 'n' bits (idx-0)
19   #define setAll(S, n) ((S) = ((n)>=64 ? ~0LL : (1LL << (n))-1))
20
21   // Extrae el bit menos significativo 0100 (Least Significant Bit)
22   #define lsb(S) ((S) & -(S))
23   // Numero de ceros a la derecha (Posicion del LSB, idx-0)
24   #define idxLastBit(x) __builtin_ctzll(x)
25   // Extrae el bit mas significativo 0100 (Most Significant Bit)
26   #define msb(S) (1LL << (63 - __builtin_clzll(S)))
27   // Posicion del MSB (63 - ceros a la izquierda, idx-0)
28   #define idxFirstBit(x) (63 - __builtin_clzll(x))
29
30   #define countAllOnes(x) __builtin_popcountll(x)
31   // Apaga el ultimo bit encendido (el menos significativo)
32   #define turnOffLastBit(S) ((S) & ((S) - 1))
33   // Enciende el ultimo cero menos significativo
34   #define turnOnLastZero(S) ((S) | ((S) + 1))
35   // Apaga todos los bits encendidos mas a la derecha consecutivos
36   #define turnOffLastConsecutiveBits(S) ((S) & ((S) + 1))
37   // Enciende los ceros consecutivos mas a la derecha
```

```
8      for (int j=i+1; j<n; ++j)
9        if (abs (a[j][i]) > abs (a[k][i]))
10         k = j;
11     if (abs (a[k][i]) < EPS) {
12       det = 0;
13       break;
14     }
15     swap (a[i], a[k]);
16     if (i != k)
17       det = -det;
18     det *= a[i][i];
19     for (int j=i+1; j<n; ++j)
20       a[i][j] /= a[i][i];
21     for (int j=0; j<n; ++j)
22       if (j != i && abs (a[j][i]) > EPS)
23         for (int k=i+1; k<n; ++k)
24           a[j][k] -= a[i][k] * a[j][i];
25   }
26
27   cout << det;
```

10.2   Gauss Method For Solving System Of Linear Equations

```
1    const double EPS = 1e-9;
2    const int INF = 2; // it doesn't actually have to be infinity or a big number
3
4    int gauss (vector < vector<double> > a, vector<double> & ans) {
5      int n = (int) a.size();
6      int m = (int) a[0].size() - 1;
7
8      vector<int> where (m, -1);
9      for (int col=0, row=0; col<m && row<n; ++col) {
10       int sel = row;
11       for (int i=row; i<n; ++i)
12         if (abs (a[i][col]) > abs (a[sel][col]))
13           sel = i;
14       if (abs (a[sel][col]) < EPS)
15         continue;
16       for (int i=col; i<=m; ++i)
17         swap (a[sel][i], a[row][i]);
18       where[col] = row;
19
20       for (int i=0; i<n; ++i)
21         if (i != row) {
22           double c = a[i][col] / a[row][col];
23           for (int j=col; j<=m; ++j)
24             a[i][j] -= a[row][j] * c;
25         }
26       ++row;
27     }
28
29     ans.assign (m, 0);
30     for (int i=0; i<m; ++i)
31       if (where[i] != -1)
32         ans[i] = a[where[i]][m] / a[where[i]][i];
33     for (int i=0; i<n; ++i) {
34       double sum = 0;
35       for (int j=0; j<m; ++j)
36         sum += ans[j] * a[i][j];
```

```
1   // Hasta N = 10^8 aprox en 1s
2   const int MAX_V = 1e7 + 5;
3   bitset<MAX_V> composite;
4   void sieve() {
5       composite[0] = composite[1] = true;
6       for (int i = 2; i * i < MAX_V; i++) {
7           if (composite[i]) continue;
8           for (int j = i * i; j < MAX_V; j += i) {
9               composite[j] = true;
10          }
11      }
12  }
13
14  int main() {
15      sieve();
16      for (int i = 2; i < 100; i++) {
17          cout << i << "␣is_primes␣:␣" << !composite[i] << '\n';
18      }
19  }
```

9.8   Sum Of Divisors

```
1   //* Sum of divs
2   long long SumOfDivisors(long long num) {
3       long long total = 1;
4
5       for (int i = 2; (long long)i * i <= num; i++) {
6           if (num % i == 0) {
7               int e = 0;
8               do {
9                   e++;
10                  num /= i;
11              } while (num % i == 0);
12
13              long long sum = 0, pow = 1;
14              do {
15                  sum += pow;
16                  pow *= i;
17              } while (e-- > 0);
18              total *= sum;
19          }
20      }
21      if (num > 1) {
22          total *= (1 + num);
23      }
24      return total;
25  }
```

10   Numerical Methods

10.1   Calculating The Determinant Of A Matrix By Gauss

```
1   const double EPS = 1E-9;
2   int n;
3   vector < vector<double> > a (n, vector<double> (n));
4
5   double det = 1;
6   for (int i=0; i<n; ++i) {
7       int k = i;
```

```
38  #define turnOnLastConsecutiveZeroes(S) ((S) | ((S) - 1))
39
40  // Mascara de bits (mask -> subconjunto) O(2^N)
41  for (int mask = 0; mask < (1 << N); mask++)
42
43  // Recorrer subconjuntos de un superconjunto (menos el vacio)
44  int b = 0b1011; // Representacion binaria de un decimal en int
45  for (int i = b; i; i = (i - 1) & b) {
46    cout << bitset<4>(i) << "\n";
47  }
48
49  void printBin(ll x) {
50      // 63 -> unsigned ll, 62 -> ll, 31 -> unsigned int, 30 -> int
51      for (ll i = 63; i >= 0; i--)
52          cout << ((x >> i) & 1);
53      cout << '\n';
54  }
```

3   Combinatory

3.1   Combi Brute Sin Mod

```
1   // nCk brute force sin MOD n <= 20
2   long long nCk_bruteforce(long long n, long long k) {
3       if (k < 0 || k > n) return 0;
4       long long res = 1;
5       for (long long i = 1; i <= k; i++) {
6           res = res * (n - i + 1) / i; // aqui la division es exacta
7       }
8       return res;
9   }
10
11  // nPk brute force sin MOD n <= 20
12  long long nPk_bruteforce(long long n, long long k) {
13      if (k < 0 || k > n) return 0;
14      long long res = 1;
15      for (long long i = 0; i < k; i++) {
16          res *= (n - i);
17      }
18      return res;
19  }
```

3.2   Combinatory

OJO: Es necesario usar binpow con MOD primo

```
1   // Devuelve el inverso modular de a mod MOD
2   // Usa el Teorema Pequeno de Fermat: a^(MOD-2) === a^(-1) (mod MOD)
3   // (valido solo si MOD es primo)
4   ll inv(ll a, ll p = MOD) {
5       return binpow(a, p - 2, p);
6   }
7
8   // Factoriales e inversos factoriales precomputados
9   // fact[n]  = n! mod MOD
10  // invf[n]  = (n!)^(-1) mod MOD
11  // Precomputa en O(n)
12  vector<ll> fact(MAXN + 1), invf(MAXN + 1);
13
```

```
14  void precompute_factorials() {
15      fact[0] = 1;
16      for (int i = 1; i <= MAXN; i++) {
17          fact[i] = fact[i - 1] * i % MOD;
18      }
19      invf[MAXN] = inv(fact[MAXN]);
20      for (int i = MAXN; i > 0; i--) {
21          invf[i - 1] = invf[i] * i % MOD;
22      }
23  }
24
25  // Combinatoria de n en k: nCk(n, k) para n <= 10^6
26  // "n choose k" = n! / (k! * (n-k)!) mod MOD
27  // Retorna 0 si k > n
28  ll nCk(ll n, ll k) {
29      if (k < 0 || k > n) return 0;
30      return fact[n] * invf[k] % MOD * invf[n - k] % MOD;
31  }
32
33  // Permutacion de n en k: nPk(n, k) para n <= 10^6
34  // Calcula permutaciones: "n permute k" = n! / (n-k)! mod MOD
35  // Retorna 0 si k > n
36  ll nPk(ll n, ll k) {
37      if (k < 0 || k > n) return 0;
38      return fact[n] * invf[n - k] % MOD;
39  }
```

4   Data Structures

4.1   Fenwick Tree

```
1   struct FenwickTree {
2     vector<long long> tree;
3     int n;
4
5     FenwickTree(int size) : n(size + 1) {
6       tree.assign(n, 0);
7     }
8
9     void update(int idx, long long delta) {
10      for (idx++; idx < n; idx += idx & -idx) {
11        tree[idx] += delta;
12      }
13    }
14
15    long long query(int idx) {
16      long long sum = 0;
17      for (idx++; idx > 0; idx -= idx & -idx) {
18        sum += tree[idx];
19      }
20      return sum;
21    }
22
23    long long range_query(int l, int r) {
24      return query(r) - query(l - 1);
25    }
26  };
27
```

```
9       }
10    }
11    if(n > 1) {
12        ans -= ans / n;
13    }
14    return ans;
15  }
16
17  //* phi(n) -> complex: O(log(log(n)))
18  void phi_1_to_n(int n) {
19      vector<int> phi(n + 1);
20      for (int i = 0; i <= n; i++)
21          phi[i] = i;
22
23      for (int i = 2; i <= n; i++) {
24          if (phi[i] == i) {
25              for (int j = i; j <= n; j += i)
26                  phi[j] -= phi[j] / i;
27          }
28      }
29  }
```

9.5   Potenciacion Binaria

```
1   using ll = long long;
2   const int MAXN = 1e6;    // limite superior de n
3   const ll MOD = 1e9 + 7; // primo grande
4
5   // Potenciacion binaria modular a^b mod p
6   ll binpow(ll a, ll b, ll m = MOD) {
7       a %= m;
8       ll res = 1;
9       while (b > 0) {
10          if (b & 1)
11              res = res * a % m;
12          a = a * a % m;
13          b >>= 1;
14      }
15      return res;
16  }
```

9.6   Sieve

```
1   // Criba de Eratostenes: Hasta N = 10^6
2   void sieve(vector<bool>& is_prime) {
3     int N = (int) is_prime.size();
4     if (!is_prime[0]) is_prime.assign(N+1, true);
5     is_prime[0] = is_prime[1] = false;
6     for (int p = 2; p * p <= N; p++) {
7       if (is_prime[p]) {
8         for (int i = p * p; i <= N; i += p) {
9           is_prime[i] = false;
10        }
11      }
12    }
13  }
```

9.7   Sieve Bitset

```
4        if (a == 0) return b;
5        return gcd(b % a, a);
6    }
7
8    // Minimo comun multiplo (LCM): Calculado con GCD
9        int lcm(int a, int b) {
10       return (a * b) / gcd(a, b);
11   }
```

9.3   Number Theory

```
1    // Divisores de N: Hasta N = 10^6
2    vector<int> divisores(int N) {
3      vector<int> divs;
4      for (int d = 1; d * d <= N; d++) {
5        if (N % d == 0) {
6          divs.push_back(d);
7          if (N / d != d) divs.push_back(N / d);
8        }
9      }
10     return divs;
11   }
12
13   // Factorizacion de N: Hasta N = 10^6
14   vector<pair<int, int>> factorizar(int N) {
15     vector<pair<int, int>> facts;
16     for (int p = 2; p * p <= N; p++) {
17       if (N % p == 0) {
18         int exp = 0;
19         while (N % p == 0) {
20           exp++;
21           N /= p;
22         }
23         facts.push_back({ p, exp });
24       }
25     }
26     if (N > 1) facts.push_back({ N, 1 });
27     return facts;
28   }
29
30   // Primalidad: Hasta N = 10^6 - O(sqrt(N))
31   bool isPrime(int N) {
32     if (N < 2) return false;
33     for (int d = 2; d * d <= N; d++) {
34       if (N % d == 0) return false;
35     }
36     return true;
37   }
```

9.4   Phi Euler

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

```
1    int phi(int n) {
2        int ans = n;
3        for(int i = 2; i * i <= n; i++) {
4            if(n % i == 0) {
5                while (n % i == 0) {
6                    n /= i;
7                }
8                ans -= ans / i;
```

```
28   // uso:
29   // int n; cin >> n;
30   // FenwickTree ft(n);
31   // for (int i = 0; i < n; i++) {
32   //    long long x; cin >> x;
33   //    ft.update(i, x);
34   // }
35   // ft.update(idx, delta); // actualizar elemento en idx
36   // long long sum = ft.range_query(l, r); // suma en rango [l, r]
```

4.2   Find Two Numbers

```
1    // "find two number where the sum is x, and gcd(a, b) > 1" b
2    auto find = [&](ll x){
3      for(int d = 2; d <= x / 2; d++){
4        if(x % d == 0){
5          ll m = 1, n = (x / d) - 1;
6          ll a = d * m, b = d * n;
7          if(__gcd(a, b) > 1){
8            cout<< a << '␣' << b;
9            ps();
10           return;
11         }
12       }
13     }
14   };
```

4.3   Segment Tree

```
1    // "This segment_tree I understand better how it works"
2    template<typename T>
3    struct seg_tree {
4        int N;
5        T Z = 0;
6        vector<T> tree;
7
8        seg_tree(int N) : N(N) {
9            tree.resize(4 * N);
10       }
11
12       seg_tree(vector<T>& A) {
13           N = (int)A.size();
14           tree.resize(4 * N);
15           build(A, 1, 0, N-1);
16       }
17
18   private:
19       T op(T a, T b) {
20           return a + b;
21       }
22
23       void build(vector<T>& a, int node, int left, int right) {
24           if(left == right) {
25               tree[node] = a[left];
26               return;
27           }
28           int mid = (left + right) >> 1;
29           build(a, 2 * node, left, mid);
30           build(a, 2 * node + 1, mid + 1, right);
31           tree[node] = op(tree[2 * node], tree[2 * node + 1]);
```

```cpp
32        }
33
34     void modify(int pos, T value, int node, int left, int right) {
35         if(left == right) {
36             tree[node] = value;
37             return;
38         }
39         int mid = (left + right) >> 1;
40         if(pos <= mid)
41             modify(pos, value, 2 * node, left, mid);
42         else
43             modify(pos, value, 2 * node + 1, mid + 1, right);
44         tree[node] = op(tree[2 * node], tree[2 * node + 1]);
45     }
46
47     T query(int l, int r, int node, int left, int right) {
48         if(r < left || l > right) return Z;
49         if(l <= left && right <= r) return tree[node];
50         int mid = (left + right) >> 1;
51         T leftSum = query(l, r, 2 * node, left, mid);
52         T rightSum = query(l, r, 2 * node + 1, mid + 1, right);
53         return op(leftSum, rightSum);
54     }
55
56 public:
57     void build(vector<T>& a) { build(a, 1, 0, N-1); }
58     void modify(int pos, T value) { modify(pos, value, 1, 0, N-1); }
59     T query(int l, int r) { return query(l, r, 1, 0, N-1); }
60 };
```

4.4　Sparse Table

```cpp
1 int n; cin >> n;
2 vector<long long> a(n);
3 for (int i = 0; i < n; i++) cin >> a[i];
4 int k = log2(n) + 1;
5 vector<vector<long long>> st(n, vector<long long>(k));
6 for (int i = 0; i < n; i++) st[i][0] = a[i];
7 for (int j = 1; j < k; j++) {
8   for (int i = 0; i + (1 << j) <= n; i++) {
9     st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
10   }
11 }
12 function<long long(int, int)> query = [&](int l, int r) {
13   int j = log2(r - l + 1);
14   return min(st[l][j], st[r - (1 << j) + 1][j]);
15 };
16 // query(l, r) = minimo en rango [l, r] en O(1)
17 // cambiar min por max para maximo
18 // cambiar min por gcd para GCD en rango
```

5　Dp

5.1　Digit Dp Pattern

```cpp
1 string pattern; cin >> pattern; // ejemplo: "xxxxx3xxxx" donde x = digito libre
2 int n = pattern.size();
3 long long k; cin >> k; // modulo
4
```

```cpp
22             assert(ps[i].x >= ps[j].x && ps[i].y >= ps[j].y);
23             edges.push_back({(ps[i].x - ps[j].x) + (ps[i].y - ps[j].y), i, j});
24         }
25         active[ps[i].x] = i;
26     }
27     for (auto &p : ps) { // rotate
28         if (rot & 1) p.x *= -1;
29         else swap(p.x, p.y);
30     }
31   }
32   return edges;
33 }
```

9　Number Theory

9.1　Euler Toliente

```cpp
1 class EulerTotiente {
2   public:
3   //* metodo en O(sqrt(n))
4   template <typename T>
5   T euler_classic(T n) {
6     T result = n;
7     for(T i = 2; i * i <= n; i++) {
8       if(n % i == 0) {
9         while(n % i == 0) n /= i;
10        result -= result / i;
11      }
12    }
13    if(n > 1) {
14      result -= result / n;
15    }
16    return result;
17  }
18
19  //* metodo en O(nlog(log(n)))
20  void euler_faster(int n) {
21    vector<int> phi(n + 1);
22    for(int i = 0; i <= n; i++) {
23      phi[i] = i;
24    }
25    for(int i = 2; i <= n; i++) {
26      if(phi[i] == i) {
27        for(int j = i; j <= n; j += i) {
28          phi[j] -= phi[j] / i;
29        }
30      }
31    }
32    for(int i = 1; i <= n; i++) {
33      cout << i << '␣' << phi[i] << '\n';
34    }
35  }
36 };
```

9.2　Gcd Lcm

```cpp
1 // Maximo comun divisor (GCD): Algoritmo de Euclides
2 int gcd(int a, int b) {
3     if (a > b) swap(a, b);
```

```cpp
    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;
}
```

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;
```

```cpp
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 digito 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 numeros que siguen el patron y son divisibles por k
// ejemplo: pattern = "xxxxx3xxxx", k = 7
// cuenta numeros tipo 1234534567 que son divisibles por 7
// x o X = digito libre, cualquier otro caracter = digito fijo
```

5.2   Digit Dp

```cpp
string s; cin >> s; // numero 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;
```

```
16    for (int d = 0; d <= limit; d++) {
17      bool new_tight = tight && (d == limit);
18      bool new_started = started || (d > 0);
19      int new_rem = (rem * 10 + d) % k;
20      res += solve(pos + 1, new_rem, new_tight, new_started);
21    }
22    if (started && !tight) {
23      dp[pos][rem][tight ? 1 : 0] = res;
24    }
25    return res;
26  };
27
28  long long result = solve(0, 0, true, false);
29  // result = cantidad de numeros <= s que son divisibles por k
30  // para contar en rango [a, b]: result_b - result_a-1
31  // ejemplo: s = "1000000", k = 7 -> contar numeros de 0 a 1000000 divisibles por 7
```

5.3   Edit Distance

```
1  string s1, s2; cin >> s1 >> s2;
2  int n = s1.size(), m = s2.size();
3  vector<vector<int>> dp(n + 1, vector<int>(m + 1));
4  for (int i = 0; i <= n; i++) dp[i][0] = i;
5  for (int j = 0; j <= m; j++) dp[0][j] = j;
6  for (int i = 1; i <= n; i++) {
7    for (int j = 1; j <= m; j++) {
8      if (s1[i - 1] == s2[j - 1]) {
9        dp[i][j] = dp[i - 1][j - 1];
10     } else {
11       dp[i][j] = 1 + min({dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1]});
12     }
13   }
14 }
15 // dp[n][m] = edit distance (minimo numero de operaciones: insertar, eliminar, reemplazar)
16 // para convertir s1 en s2
```

5.4   Knapsack

```
1  int n, capacity; cin >> n >> capacity;
2  vector<int> weight(n), value(n);
3  for (int i = 0; i < n; i++) {
4    cin >> weight[i] >> value[i];
5  }
6  vector<long long> dp(capacity + 1, 0);
7  for (int i = 0; i < n; i++) {
8    for (int w = capacity; w >= weight[i]; w--) {
9      dp[w] = max(dp[w], dp[w - weight[i]] + value[i]);
10   }
11 }
12 // dp[capacity] = valor maximo que se puede obtener con capacidad maxima
13 // para version con items ilimitados, cambiar el loop: for (int w = weight[i]; w <= capacity
        ; w++)
```

5.5   Lcs

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

```
8    vector<vector<int>> components;
9    vector<int> component_id;
10
11   //1-indexed
12   SCC(int n = 0){ N = n + 1, adj.assign(N, {}); }
13   SCC(const vector<vector<int>> & _adj){ adj = _adj, N = adj.size(); }
14
15   void add_edge(int from, int to){
16     adj[from].push_back(to);
17   }
18
19   void dfs(int u){
20     low[u] = ind[u] = id++;
21     s.push(u);
22     in_stack[u] = true;
23     for(int v : adj[u]){
24       if(ind[v] == -1){
25         dfs(v);
26         low[u] = min(low[u], low[v]);
27       }else if(in_stack[v]){
28         low[u] = min(low[u], ind[v]);
29       }
30     }
31     if(low[u] == ind[u]){
32       components.emplace_back();
33       vector<int> & comp = components.back();
34       while(true){
35         assert(!s.empty());
36         int x = s.top(); s.pop();
37         in_stack[x] = false;
38         component_id[x] = components.size() - 1;
39         comp.push_back(x);
40         if(x == u) break;
41       }
42     }
43   }
44
45   vector<vector<int>> get(){
46     ind.assign(N, - 1); low.assign(N, -1); component_id.assign(N, -1);
47     s = stack<int>();
48     in_stack.assign(N, false);
49     id = 0;
50     components = {};
51     for(int i = 1; i < N; i++)
52       if(ind[i] == -1) dfs(i);
53
54     // reverse(components.begin(), components.end()); return components; // SCC in
          topological order
55     return components; // SCC in reverse topological order
56   }
57 };
```

7.13   Topological Sort

```
1  vector<int> top_sort(vector<vector<int>>& adj){
2    int n = adj.size();
3    bool cycle = false;
4    vector<int> sorted, color(n);
```

```cpp
40        if(is_ancestor(u, v)) return u;
41        if(is_ancestor(v, u)) return v;
42
43        for(int bit = l; bit >= 0; bit--){
44          if(is_ancestor(up[u][bit], v)) continue;
45          u = up[u][bit];
46        }
47        return up[u][0];
48      }
49
50      int ancestor(int u, int k){
51        if(depth[u] <= k) return -1;
52        for(int bit = 0; bit <= l; bit++){
53          if(k >> bit & 1) u = up[u][bit];
54        }
55        return u;
56      }
57
58      int distance(int u, int v){
59        return depth[u] + depth[v] - 2 * depth[query(u, v)];
60      }
61    };
```

7.11   Prim

```cpp
1   int n, m; cin >> n >> m;
2   vector<vector<pair<int, long long>>> adj(n + 1);
3   for (int i = 0; i < m; i++) {
4     int u, v; cin >> u >> v;
5     long long w; cin >> w;
6     adj[u].emplace_back(v, w);
7     adj[v].emplace_back(u, w);
8   }
9   vector<bool> vis(n + 1);
10  pqg<pair<long long, int>> pq;
11  pq.push({0LL, 1});
12  long long mst_cost = 0;
13  while (!pq.empty()) {
14    auto [w, u] = pq.top(); pq.pop();
15    if (vis[u]) continue;
16    vis[u] = true;
17    mst_cost += w;
18    for (auto [v, weight] : adj[u]) {
19      if (!vis[v]) {
20        pq.push({weight, v});
21      }
22    }
23  }
24  // mst_cost = costo del MST (Minimum Spanning Tree)
```

7.12   Scc

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

```cpp
1   // "These works to find a componente fuertemente conexa that it's in directed graph"
2   struct SCC {
3     int N = 0, id;
4     vector<vector<int>> adj;
5     vector<int> ind, low;
6     stack<int> s;
7     vector<bool> in_stack;
```

```cpp
6         if (s1[i - 1] == s2[j - 1]) {
7           dp[i][j] = dp[i - 1][j - 1] + 1;
8         } else {
9           dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
10        }
11      }
12    }
13    // dp[n][m] = longitud de LCS (Longest Common Subsequence)
14
15    string reconstruct_lcs() {
16      string lcs = "";
17      int i = n, j = m;
18      while (i > 0 && j > 0) {
19        if (s1[i - 1] == s2[j - 1]) {
20          lcs += s1[i - 1];
21          i--, j--;
22        } else if (dp[i - 1][j] > dp[i][j - 1]) {
23          i--;
24        } else {
25          j--;
26        }
27      }
28      reverse(lcs.begin(), lcs.end());
29      return lcs;
30    }
31    // lcs = string de la LCS
```

6   Geometry

6.1   Convex Hull

```cpp
1   struct Point {
2     long long x, y;
3     Point(long long x = 0, long long y = 0) : x(x), y(y) {}
4     Point operator-(const Point& p) const { return Point(x - p.x, y - p.y); }
5     long long cross(const Point& p) const { return x * p.y - y * p.x; }
6     long long cross(const Point& a, const Point& b) const { return (a - *this).cross(b - *this
          ); }
7     bool operator<(const Point& p) const { return x < p.x || (x == p.x && y < p.y); }
8     bool operator==(const Point& p) const { return x == p.x && y == p.y; }
9   };
10
11  vector<Point> convex_hull(vector<Point>& points) {
12    int n = points.size();
13    if (n <= 1) return points;
14    sort(points.begin(), points.end());
15    vector<Point> hull;
16    for (int i = 0; i < n; i++) {
17      // Si quiero incluir coords colineales ( < 0), si no lo quiero ( <= 0)
18      while (hull.size() >= 2 && hull[hull.size() - 2].cross(hull.back(), points[i]) <= 0) {
19        hull.pop_back();
20      }
21      hull.push_back(points[i]);
22    }
23    int lower = hull.size();
24    for (int i = n - 2; i >= 0; i--) {
25      // Si quiero incluir coords colineales ( < 0), si no lo quiero ( <= 0)
26      while (hull.size() > lower && hull[hull.size() - 2].cross(hull.back(), points[i]) <= 0)
              {
```

```
27        hull.pop_back();
28      }
29      hull.push_back(points[i]);
30    }
31    hull.pop_back();
32    return hull;
33 }
34
35 // uso:
36 // int n; cin >> n;
37 // vector<Point> points(n);
38 // for (int i = 0; i < n; i++) {
39 //    cin >> points[i].x >> points[i].y;
40 // }
41 // vector<Point> hull = convex_hull(points);
42 // hull contiene los puntos del convex hull en orden counter-clockwise (CCW / Antihorario)
43 // reverse(hull) para obtenerlo en orden CW / horario
```

6.2   Point Operations

```
1  struct Point {
2    long long x, y;
3    Point(long long x = 0, long long y = 0) : x(x), y(y) {}
4    Point operator+(const Point& p) const { return Point(x + p.x, y + p.y); }
5    Point operator-(const Point& p) const { return Point(x - p.x, y - p.y); }
6    Point operator*(long long k) const { return Point(x * k, y * k); }
7    long long dot(const Point& p) const { return x * p.x + y * p.y; }
8    long long cross(const Point& p) const { return x * p.y - y * p.x; }
9    long long cross(const Point& a, const Point& b) const { return (a - *this).cross(b - *this
          ); }
10   long long norm2() const { return x * x + y * y; }
11   double norm() const { return sqrt(norm2()); }
12   bool operator<(const Point& p) const { return x < p.x || (x == p.x && y < p.y); }
13   bool operator==(const Point& p) const { return x == p.x && y == p.y; }
14 };
15
16 long long orientation(Point a, Point b, Point c) {
17   return (b - a).cross(c - a);
18 }
19
20 bool collinear(Point a, Point b, Point c) {
21   return orientation(a, b, c) == 0;
22 }
23
24 bool cw(Point a, Point b, Point c) {
25   return orientation(a, b, c) < 0;
26 }
27
28 bool ccw(Point a, Point b, Point c) {
29   return orientation(a, b, c) > 0;
30 }
31
32 // uso:
33 // Point p1(1, 2), p2(3, 4), p3(5, 6);
34 // long long area = abs((p2 - p1).cross(p3 - p1)) / 2; // area del triangulo
35 // bool clockwise = cw(p1, p2, p3); // true si p1->p2->p3 es clockwise
36 // bool counter_clockwise = ccw(p1, p2, p3); // true si p1->p2->p3 es counter-clockwise
```

```
20 int n, m; cin >> n >> m;
21 vector<tuple<long long, int, int>> edges;
22 for (int i = 0; i < m; i++) {
23   int u, v; cin >> u >> v;
24   long long w; cin >> w;
25   edges.emplace_back(w, u, v);
26 }
27 sort(edges.begin(), edges.end());
28 DSU dsu(n);
29 long long mst_cost = 0;
30 for (auto [w, u, v] : edges) {
31   if (dsu.find(u) != dsu.find(v)) {
32     dsu.merge(u, v);
33     mst_cost += w;
34   }
35 }
36 // mst_cost = costo del MST (Minimum Spanning Tree)
```

7.10   Lowest Common Ancestor Lca

```
1  struct LCA{
2    int n, l, timer = 0;
3    vector<vector<int>> up, adj;
4    vector<int> depth, in, out;
5
6    LCA(int _n) {
7      n = _n + 1;
8      l = ceil(log2(n));
9      up.resize(n, vector<int>(l + 1));
10     adj.resize(n);
11     depth.resize(n);
12     in.resize(n);
13     out.resize(n);
14   }
15
16   void add_edge(int p, int u){
17     adj[p].push_back(u);
18     adj[u].push_back(p);
19   }
20
21   void dfs(int u = 1, int p = 1){
22     up[u][0] = p;
23     depth[u] = depth[p] + 1;
24     in[u] = ++timer;
25     for(int level = 1; level <= l; level++){
26       up[u][level] = up[up[u][level - 1]][level - 1];
27     }
28     for(int v : adj[u]){
29       if(v == p) continue;
30       dfs(v, u);
31     }
32     out[u] = ++timer;
33   }
34
35   bool is_ancestor(int p, int u){
36     return in[p] <= in[u] && out[p] >= out[u];
37   }
38
39   int query(int u, int v){
```

```
 6      }
 7
 8      int find(int x){
 9        if(p[x] != x) p[x] = find(p[x]);
10        return p[x];
11      }
12
13      void merge(int x, int y){
14        x = find(x), y = find(y);
15        if(x == y) return;
16        if(size[x] < size[y]) swap(x, y);
17        size[x] += size[y];
18        p[y] = x;
19      }
20    };
```

7.8   Floyd Warshall

```
 1  int n; cin >> n;
 2  vector<vector<long long>> dist(n + 1, vector<long long>(n + 1, inf));
 3  for (int i = 1; i <= n; i++) dist[i][i] = 0;
 4  int m; cin >> m;
 5  for (int i = 0; i < m; i++) {
 6    int u, v; cin >> u >> v;
 7    long long w; cin >> w;
 8    dist[u][v] = min(dist[u][v], w);
 9    dist[v][u] = min(dist[v][u], w);
10  }
11  for (int k = 1; k <= n; k++) {
12    for (int i = 1; i <= n; i++) {
13      for (int j = 1; j <= n; j++) {
14        if (dist[i][k] != inf && dist[k][j] != inf) {
15          dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
16        }
17      }
18    }
19  }
20  // dist[i][j] = distancia minima entre nodo i y nodo j
```

7.9   Kruskal

```
 1  struct DSU {
 2    vector<int> p, size;
 3    DSU(int n) {
 4      p.resize(n + 1), size.resize(n + 1, 1);
 5      for (int i = 1; i <= n; i++) p[i] = i;
 6    }
 7    int find(int x) {
 8      if (p[x] != x) p[x] = find(p[x]);
 9      return p[x];
10    }
11    void merge(int x, int y) {
12      x = find(x), y = find(y);
13      if (x == y) return;
14      if (size[x] < size[y]) swap(x, y);
15      size[x] += size[y];
16      p[y] = x;
17    }
18  };
19
```

7   Graph

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

7.1   Bellman Ford

```
 1  int n, m; cin >> n >> m;
 2  vector<tuple<int, int, long long>> edges;
 3  for (int i = 0; i < m; i++) {
 4    int u, v; cin >> u >> v;
 5    long long w; cin >> w;
 6    edges.emplace_back(u, v, w);
 7  }
 8  vector<long long> dist(n + 1, inf);
 9  dist[1] = 0;
10  for (int i = 0; i < n - 1; i++) {
11    for (auto [u, v, w] : edges) {
12      if (dist[u] != inf) {
13        dist[v] = min(dist[v], dist[u] + w);
14      }
15    }
16  }
17  bool has_negative_cycle = false;
18  for (auto [u, v, w] : edges) {
19    if (dist[u] != inf && dist[u] + w < dist[v]) {
20      has_negative_cycle = true;
21      break;
22    }
23  }
24  // dist[u] = distancia minima desde nodo 1 hasta u
25  // has_negative_cycle = true si hay ciclo negativo alcanzable desde nodo 1
```

7.2   Bfs

```
 1  vector<bool> vis(n+1);
 2  queue<int> q;
 3  function<void(int)> bfs = [&](int start) {
 4    vis[start] = true;
 5    q.push(start);
 6    int levels = 1;
 7    while (!q.empty()) {
 8      int sz = q.size();
 9      levels++;
10      while (sz--) {
11        int u = q.front();
12        q.pop();
13        for (int& v : adj[u]) {
14          if (vis[v]) continue;
15          vis[v] = true;
16          q.push(v);
17        }
18      }
19    }
20  };
21
22  for (int u = 1; u <= n; u++) {
23    if (vis[u]) continue;
24    bfs(u);
```

```
25 }
```

7.3   Bipartite

```
1  int N, M; cin >> N >> M;
2  vector<vector<int>> adj(N + 1);
3  while (M--) {
4    int u, v; cin >> u >> v;
5    adj[u].push_back(v);
6    adj[v].push_back(u);
7  }
8
9  vector<bool> vis(N + 1);
10 vector<int> col(N + 1, 0);
11 // bipartite graph
12 function<bool(int, int)> dfs = [&](int u, int c) {
13   vis[u] = 1;
14   col[u] = c;
15
16   for (auto v : adj[u]) {
17     if (vis[v] && col[u] == col[v]) return false;
18     else if (!vis[v] && !dfs(v, c ^ 1)) return false;
19   }
20   return true;
21 };
22
23 for (int i = 1; i <= N; i++) {
24   if (vis[i]) continue;
25   if (dfs(i, 1) == false) {
26     cout << "IMPOSSIBLE";
27     return;
28   }
29 }
30
31 for (int i = 1; i <= N; i++) cout << (col[i] ? 1 : 2) << '␣';
```

7.4   Dfs

```
1  vector<bool> vis(n+1);
2  function<void(int)> dfs = [&](int u) {
3    vis[u] = true;
4    for (int& v : adj[u]) {
5      if (vis[v]) continue;
6      dfs(v);
7    }
8  };
9
10 for (int u = 1; u <= n; u++) {
11   if (vis[u]) continue;
12   dfs(u);
13 }
```

7.5   Dfs 2D

```
1  int N, M; cin >> N >> M;
2  vector<vector<char>> grid(N, vector<char>(M));
3  for (int i = 0; i < N; i++) {
4    for (int j = 0; j < M; j++) {
5      cin >> grid[i][j];
6    }
```

```
7  }
8
9  vector<vector<bool>> vis(N, vector<bool>(M));
10 int dx[4] = {-1, 1, 0, 0}, dy[4] = {0, 0, -1, 1};
11 function<void(int, int)> dfs = [&](int x, int y) {
12   vis[x][y] = 1;
13
14   for (int d = 0; d < 4; d++) {
15     int nx = x + dx[d], ny = y + dy[d];
16     if (0 <= nx && 0 <= ny && nx < N && ny < M && grid[nx][ny] == '.' && !vis[nx][ny]) {
17       dfs(nx, ny);
18     }
19   }
20 };
21
22 int comp = 0;
23 for (int i = 0; i < N; i++) {
24   for (int j = 0; j < M; j++) {
25     if (vis[i][j] || grid[i][j] == '#') continue;
26     dfs(i, j);
27     comp++;
28   }
29 }
30
31 cout << comp;
```

7.6   Dijkstra

```
1  int N, M; cin >> N >> M;
2  vector<vector<pair<int, long long>>> adj(N + 1);
3  for (int i = 0; i < M; i++) {
4    int u, v; cin >> u >> v;
5    long long w; cin >> w;
6    adj[u].emplace_back(v, w);
7  }
8  vector<long long> dis(N + 1, inf);
9  pqg<pair<long long, int>> pq;
10 dis[1] = 0;
11 pq.push({0LL, 1});
12 while (!pq.empty()) {
13   auto [d, node] = pq.top(); pq.pop();
14   if (dis[node] != d) continue;
15   for (auto [v, w] : adj[node]) {
16     if (d + w < dis[v]) {
17       dis[v] = d + w;
18       pq.push({dis[v], v});
19     }
20   }
21 }
22 for (int u = 1; u <= N; u++) {
23   cout << dis[u] << "␣";
24 }
```

7.7   Disjoint Set Union Dsu

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