Universidad Complutense de Madrid

I'm So Meta Even This Acronym

David Pérez, Pablo Hidalgo, Mingxiao Guo

ACM-ICPC SWERC 2017

26 de noviembre, 2017

Matemáticas (1)

1.1 Sumas

$$r^a + r^{a+1} + \cdots + r^b = \frac{r^{b+1} - r^a}{r-1}, r \neq 1$$

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$
$$\sum_{k=1}^{n} k^2 = \frac{n(2n+1)(n+1)}{6}$$
$$\sum_{k=1}^{n} k^3 = \left(\sum_{k=1}^{n} k^2\right)^2$$
$$\sum_{k=1}^{n} k^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

La suma de la progresión aritmética $a_n = a_1 + (n-1)d$ es

$$S_n = \frac{n}{2}(2a_1 + (n-1)d) = \frac{n}{2}(a_1 + a_n)$$

La suma de la progresión geométrica $a_n = ar^n$ es

$$\mathcal{S}_n = a\left(\frac{1-r^n}{1-r}\right), r \neq 1$$

1.2 Teoría de números

Ec. Diofántica lineal Sea $ax + by = c$ una ED, y $d = gcd(a, b)$. La ecuación tiene solución sii $d \mid c$, y esa solución es $(x, y) = (x_0 + tb/d, y_0 + ta/d)$, con t entero. Para encontrar x_0, y_0 vale con usar el algoritmo de Euclides extendido, para obtener $ax_0 + by_0 = d$ y multiplicar ambos valores resultantes por c/d.

Teorema chino de los restos El sistema $x \equiv a_i \pmod{m_i}$ con $i \in 1, \ldots, n$ y m_i primos dos a dos tiene una única solución módulo $M = m_1 m_2 \ldots m_n$. Si b_i es la inversa de $M/m_n \pmod{M}$ entonces la solución es $x \equiv a_1 b_1 M/m_1 + \cdots + a_n b_n M/m_n \pmod{M}$.

Suma y cuenta de divisores Si $x = p_1^{\alpha_1} \ldots p_k^{\alpha_k}$ entonces la suma de divisores es $S = (1 + p_1 + \cdots + p_1^{\alpha_1}) \cdots * (1 + p_k + \cdots + p_k^{\alpha_k})$ y su número de divisores es $C = (1 + \alpha_1) \ldots (1 + \alpha_k)$.

Función φ de Euler La función $\varphi(n)$ devuelve el número de números coprimos con n menores que n. Se calcula como $\varphi(p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}) = (p_1 - 1)p_1^{\alpha_1 - 1}(p_2 - 1)p_2^{\alpha_2 - 1} \ldots (p_k - 1)p_k^{\alpha_k - 1}$. Tiene la propiedad de que si a,n primos entre sí, $a^{\varphi(n)} \equiv 1 \pmod{n}$.

Acotación de primos $n/\ln(n) < prim(n) < 1.26n/\ln(n)$ Donde $prim(n)$ es el número de primos menores que n.

Grafo planar (fórmula de Euler)

$$v - e + f_1 = 2$$

$$v - e + f_2 - k = 1$$

donde

v = número de vértices
e = número de aristas
f_1 = número de caras internas
$f_2 = 1 + f_1$ ("fondo" + caras internas)
k = número de componentes conexas

teoriaDeNumeros.cpp

```
int mod(int a, int b) { return ((a%b) + b) % b; }

// devuelve q y r, el cociente al hacer la division entera de a entre b, tal que q*b
     + r = a, con r el unico numero positivo \in [0, b) que cumple la igualdad.
// funciona para todas las combinaciones de signos de a y b.
// es decir, divide como un matematico y no como un programador que trunca la
    division hacia el 0 (ademas el estandar de C++ no especifica siquiera esto).
div_t euc_div(int a, int d) {
    div_t divT = div(a, d);
    int I = divT.rem >= 0 ? 0 : (d > 0 ? 1 : -1);
    int qE = divT.quot - I;
    int rE = divT.rem + I * d;
    return div_t{qE, rE};
}

bitset<10000000> bs; // 10^7
vector<ll> primes;

// criba de Eratstenes
void sieve() {
  bs.set();
```

```
  bs[0] = bs[1] = 0;
  for (ll i = 2; i < bs.size(); i++)
  if (bs[i]) {
    for (ll j = i * i; j < bs.size(); j += i) bs[j] = 0;
    primes.push_back(i);
  }
}

//solo funciona para 0 <= N <= primes[primes.size()-1]^2
bool isPrime(ll N) {
  if (N < bs.size()) return bs[N];
  for (int i = 0; i < primes.size(); i++)
  if (N % primes[i] == 0) return false;
  return true;
}

//devuelve el vector de factores primos de N
vector<int> primeFactors(int N) {
  vector<int> factors;
  int PF_idx = 0, PF = primes[PF_idx];
  while (PF * PF <= N) {
    while (N % PF == 0) {
      N /= PF;
      factors.push_back(PF);
    }
    PF = primes[++PF_idx];
  }
  if (N != 1) factors.push_back(N); // N is prime
  return factors;
}

// Una simple variacion de la funcion anterior para solo contar el numero de
    factores primos
int numPF(ll N) {
  int PF_idx = 0;
  ll PF = primes[PF_idx], ans = 0;
  while (PF * PF <= N) {
    while (N % PF == 0) {
      N /= PF;
      ans++;
    }
    PF = primes[++PF_idx];
  }
  if (N != 1) ans++; // N is prime
  return ans;
}

// Devuelve el numero de divisores de n (inluyendo 1 y n) uva 11876, 294
int nod(int n) {
    int d = 1;
    for (int idx = 0, p = primes[idx]; p*p <= n; p = primes[++idx]) {
        int m = 0;
        while (n % p == 0) {
            n /= p;
            m++;
        }
        d *= m + 1;
    }
    if (n != 1) d *= 2;
    return d;
}

// devuelve la suma de los divisores de N (incluyendo N y 1)
ll sumDiv(ll N) {
    ll PF_idx = 0, PF = primes[PF_idx], sum = 1;
    while (PF * PF <= N) {
        ll p = 1;
        while (N % PF == 0) {
            N /= PF;
            p *= PF;
        }
        if (p != 1) sum *= ((p * PF) - 1) / (PF - 1);
        PF = primes[++PF_idx];
    }

    if (N != 1) sum *= N + 1;
    return sum;
}

ll EulerPhi(ll n) {
  ll idx = 0, p = primes[0], ans = n;
  while (n != 1 && (p*p <= n)) {
    if (n%p == 0) ans -= ans / p;
    while (n%p == 0) n /= p;
    p = primes[++idx];
  }
  if (n != 1) ans -= ans / n;
  return ans;
}

//Algoritmo de Euclides extendidio
//ax + by = d = gcd(a,b) Devuelve d.
ll eea(ll a, ll b, ll& x, ll& y) {
  ll xx = y = 0, yy = x = 1;
  while (b) {
    ll q = a / b, t = b; b = a%b; a = t;
    t = xx; xx = x - q*xx; x = t;
    t = yy; yy = y - q*yy; y = t;
  }
  return a;
}

//Encuentra z,M tal que z%x=a, z%y=b, unica mod M.
//Si no hay, M=-1;
ll_ll chinese(ll x, ll a, ll y, ll b) {
  ll s, t, d = eea(x, y, s, t);
  if (a%d != b%d) return ll_ll(0, -1);
  return ii(mod(s*b*x + t*a*y, x*y), x*y / d);//(z, M)
}
ll_ll chinese(const vector<ll> &x, const vector<ll> &a) {
  ll_ll ret(a[0], x[0]);
  for (int i = 1; ret.second != -1 && i<x.size(); i++)
    ret = chinese(ret.first, ret.second, x[i], a[i]);
  return ret;
}
```

```
ll fastPowMod(ll a, ll b, ll m) {
  ll aux;
  if (b == 0) return 1;
  if (b % 2 == 0) return aux = fastPowMod(a, b / 2, m), (aux*aux) % m;
  return (a*fastPowMod(a, b - 1, m)) % m;
}

//Miller Rabin: test Las Vegas de primalidad
//k=30 deberia ser suficiente casi siempre.
bool probablyPrime(ll n, int k = 35) {
  ll s = n - 1, r = 0, a, aux;
  while (s % 2 == 0) s /= 2, r++;
  while (k--) {
    a = rand() % (n - 1); a++;
    a = fastPowMod(a, s, n);
    if (a%n == 1) continue;
    for (int i = 0; i <= r; a = (a*a) % n, i++)
    if (i == r) return false; //esto es compuesto si o si.
    else if (a == n - 1) break;
  }
  return true; //Lo mas probable es que sea primo.
}

//Algoritmo Rho de factorizacion (montecarlo)
ll_ll pollardRho(ll n) {
  ll x = 2, y = 2, d = 1, a, b;
  while (d == 1) {
    x = (x*x + 1) % n;
    y = (y*y + 1) % n;
    y = (y*y + 1) % n;
    d = eea(abs(x - y), n, a, b);
  }
  return ll_ll(d, n / d);
}
```

1.3 Polinomios

horner.cpp
Descripción: Evalúa el polinomio `pol` $\equiv$ `[`$a_n, a_{n-1}, \ldots, a_1, a_0$`]` $\equiv a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ en el punto `x`.
Tiempo: $\mathcal{O}(n)$

```
double horner(vi &pol, double x) {
  double ev = pol[0];
  for (int n = 1; n < pol.size(); n++)
    ev = ev * x + pol[n];
  return ev;
}
```

aintken.cpp
Descripción: Interpolación de polinomios por el método de Aintken.

```
double interpola(vector<double> x, vector<double> y, double t) {
  int n = x.size(), i, j, l;
  double v[n][n];
  for (i = 0; i < n; ++i) v[i][i] = y[i];
  for (l = 2; l <= n; ++l)
  for (i = 0; (j = i + l - 1) < n; ++i)
    // Tengo el polinomio interpolador de [i..j - 1] y [i + 1..j]
    v[i][j] = ((t - x[i])*v[i + 1][j] + (x[j] - t)*v[i][j - 1]) / (x[j] - x[i]);
  return v[0][n - 1];
}
```

1.4 Álgebra

TODO: rotar y reflejar arrays2d.

matrices.cpp
Descripción: Operaciones básicas con matrices: suma, multiplicación, potencia
Tiempo: Suma: $\mathcal{O}(n^2)$, multiplicación: $\mathcal{O}(n^3)$, potencia: $\mathcal{O}(n^3 \log n)$

```
typedef double tipo; // puede ser util usar rac (de numteor.cpp)
typedef vector<tipo> row;
typedef vector<row> matrix;

matrix mult(const matrix &a, const matrix &b) {
  int i, j, k, m = a.size(), n = a[0].size(), p = b[0].size();
  matrix c(m);
  for (i = 0; i < m; i++) c[i].resize(p);
  for (i = 0; i < m; i++)
  for (j = 0; j < p; j++) {
    tipo r = 0;
    for (k = 0; k < n; k++)
      r += a[i][k] * b[k][j];
    c[i][j] = r;
  }
  return c;
}

matrix suma(const matrix &a, const matrix &b) {
  int i, j, n = a.size(), m = a[0].size();
  matrix c(n);
  for (i = 0; i < n; i++)
  for (j = 0; j < m; j++)
    c[i].push_back(a[i][j] + b[i][j]);
  return c;
}

matrix eleva(matrix &a, int n) {
  int l = a.size(), i;
  matrix b(l, row(l, 0));
  for (i = 0; i < l; i++) b[i][i] = 1;
  while (n) // el resultado es b*a^n
  if (n & 1) {
    b = mult(b, a);
    n ^= 1;
  }
  else {
    a = mult(a, a);
    n >>= 1;
  }
  return b;
}
```

gauss.cpp
Descripción: Resolución de sistemas de ecuaciones por el método de Gauss.
La función Gauss: hace gauss en m y devuelve el rango de la matriz sin ampliar, y en rangoamp el de la ampliada. De paso devuelve también el determinante de la matriz (si no es cuadrada pasar del valor devuelto) Usa abs de `cmath`, con `doubles`.
Función resuelve: Resuelve un sist de ec lineales, suponiendo que sea compatible determinado y que se haya hecho gauss anteriormente (m escalonada) (La ultima columna de m son los lados derechos de las igualdades)
Tiempo: $\mathcal{O}\left(x^2 * y\right)$ para una matriz $y \times x$.

```
#define error 1e-9

tipo gauss(matrix &m, int &rango, int &rangoamp) {
  int i, j, k, numec = m.size(), numvar = m[0].size() - 1;
  tipo c;

  rango = 0;
  tipo det = 1;
  for (j = 0; j <= numvar; j++) {
    tipo mayor = 0;
    int l = -1;
    // Elige el mayor pivote
    for (i = rango; i<numec; i++) if (l < 0 || abs(m[i][j]) > mayor) {
      mayor = abs(m[i][j]);
      l = i;
    }
    if (mayor > error) {
      c = m[l][j];
      if (l != rango) det = -det;
      det *= c;
      for (k = j; k <= numvar; k++) {
        swap(m[l][k], m[rango][k]);
        m[rango][k] = m[rango][k] / c;
      }
      for (i = rango + 1; i < numec; i++) {
        c = m[i][j];
        for (k = j; k <= numvar; k++)
          m[i][k] = m[i][k] - m[rango][k] * c;
      }
      if (j < numvar)
        rango++;
    }
  }
  rangoamp = rango;
  for (i = rango; i < numec; i++)
  if (abs(m[i][numvar]) > error) { rangoamp++; break; }
  if (rango != numec) det = 0;
  return det;
}

void resuelve(const matrix &m, row &sol) {
  int i, j, numec = m.size(), numvar = m[0].size() - 1;
  sol.resize(numvar);
  for (i = numec - 1; i >= 0; i--) {
    tipo c = m[i][numvar];
    for (j = i + 1; j < numvar; j++) c = c - m[i][j] * sol[j];
    sol[i] = c;
  }
}
```

1.4.1 Resolución de recurrencias

gaussInZM.cpp
Descripción: Otro método de Gauss para resolver sistemas de ecuaciones, esta vez en módulo M. Resuelve el sistema $ax = b$ con n ecuaciones y `col` incógnitas.
Tiempo: $\mathcal{O}\left(n^2\texttt{col}\right)$

```
void gauss(vector<vector<int> > & a, vector<int> & b, int n, int col) {
  for (int row = 0; row < n && row < col; row++) {
    int best = row;
    for (int i = row + 1; i < n; i++)
      if (fabs(a[best][row]) < fabs(a[i][row]))
        best = i;
    swap(a[row], a[best]);
    swap(b[row], b[best]);
    for (int i = row + 1; i < col; i++)
      a[row][i] = ((a[row][i] * inv[a[row][row]]) % M + M) % M;
    b[row] = ((b[row] * inv[a[row][row]]) % M + M) % M;
    // a[row][row] = 1;
    for (int i = 0; i < n; i++) {
      int x = a[i][row];
      if (i != row && x != 0) {
        // row + 1 instead of row is an optimization
        for (int j = row + 1; j < col; j++)
          a[i][j] = ((a[i][j] - a[row][j] * x) % M + M) % M;
        b[i] = ((b[i] - b[row] * x) % M + M) % M;
      }
    }
  }
  for (int i = 0; i < b.size(); ++i) {
    b[i] = ((b[i] % M) + M) % M;
  }
  // b is the solution
}
```

recurrencias.cpp
Descripción: Calcula x[i], donde x esta definida por la recurrencia x[i] = ini[i], para
0 ≤ i < n = ini.size() = coef.size();
x[i+1] = coef[0] * x[i] + coef[1] * x[i-1] + ... + coef[n-1] * x[i-n+1]
Tiempo: $\mathcal{O}\left(n^3 \log i\right)$

```
tipo recur(vector<int> ini, vector<int> coef, int i) {
  int n = ini.size(), j;
  if (i < n) return ini[i];
  matrix m(n, row(n));
  m[0].assign(coef.begin(), coef.end());
  for (j = 1; j < n; j++) m[j][j - 1] = 1;
  m = eleva(m, i - n + 1);
  tipo res = 0;
  for (i = 0; i < n; i++) res += m[0][i] * ini[n - 1 - i];
  return res;
```

```
}
```

1.5 Búsqueda de ciclos

ciclos.cpp
Descripción: Dada una función f y un valor inicial x_0, encontrar μ, λ, que son los menores números que cumplen $x_\mu = x_{\mu+\lambda}$. En este sentido, encontramos el ciclo que produce esta función: μ es el comienzo del ciclo y λ, su longitud.
Se presentan dos algoritmos: Floyd y Brent (un poco más eficiente en teoría)
Memoria: $\mathcal{O}(1)$
Tiempo: $\mathcal{O}(\mu + \lambda)$

```
// Mu es el comienzo, lambda el ciclo.
ii floydCycleFinding(int x0) {
  int tortoise = f(x0), hare = f(f(x0));
  while (tortoise != hare)
    tortoise = f(tortoise), hare = f(f(hare));

  // 2nd part: finding mu, hare and tortoise move at the same speed
  int mu = 0; hare = x0;
  while (tortoise != hare)
    tortoise = f(tortoise), hare = f(hare), mu++;

  // 3rd part: finding lambda, hare moves, tortoise stays
  int lambda = 1; hare = f(tortoise);
  while (tortoise != hare)
    hare = f(hare), lambda++;
  return ii(mu, lambda);
}

void brent(int x0) {
  int power = lambda = 1;
  int tortoise = x0, hare = f(x0);
  while (tortoise != hare) {
    if (power == lambda) {
      tortoise = hare;
      power *= 2;
      lambda = 0;
    }
    hare = f(hare);
    lambda++;
  }

  mu = 0;
  tortoise = hare = x0;
  for (int i = 0; i < lambda; i++)
    hare = f(hare);

  while (tortoise != hare) {
    tortoise = f(tortoise);
    hare = f(hare);
    mu++;
  }
}
```

1.6 Combinatoria

Fibonnacci Tienen pinta: $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$. Se pueden calcular por DP, o por la fórmula de Binet: $F_n = \frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}}$ con $\phi = \frac{1+\sqrt{5}}{2}$. También por la fórmula matricial $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$.
T^a (Zeckendorf). *Todos los naturales se pueden escribir de manera única como suma de fibonaccis distintos sin tener dos consecutivos.*
T^a (Pisano). *Los últimos un/dos/tres/cuatro dígitos de la serie se repiten con frecuencias 60/300/1500/15000.*

Coeficientes binomiales Tienen pinta $\binom{n}{k} = \frac{n!}{(n-k)!k!}$. Se calculan por dp: $\binom{n}{0} = \binom{n}{n} = 1$, $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$. Representan formas de tomar k elementos de un conjunto con n. Además:
Prop. *Número de formas de tomar un multiconjunto de tamaño k de un conjunto de n:* $\binom{n+k-1}{k}$
Prop. *Número de tuplas de n enteros positivos con suma s:* $\binom{s-1}{n-1}$. *Con no negativos:* $\binom{s+n-1}{n-1}$

Coeficientes multinomiales $\binom{n}{k_1,\ldots,k_m} = \frac{n!}{k_1!\ldots k_m!} = \prod_{i=1}^{m} \binom{\sum_{j=1}^{i} k_j}{k_i}$

Números de Catalan Tienen pinta $C_n = \binom{2n}{n}/(n+1)$. Se calculan por DP: $C_n = C_{n-1}\frac{2n(2n-1)}{(n+1)n}$. C_n es el:

- # de formas de poner n pares de paréntesis equilibrados.
- # de árboles binarios con $n+1$ elementos.
- # de triangulaciones de un polígono de $n+2$ lados.
- # de caminos que van de esquina a esquina sin cruzar la diagonal principal.
- # de permutaciones de de longitud n que no contienen una subsecuencia creciente de longitud 3.

Derangements Permutaciones de n elementos sin puntos fijos. $D_0 = 1$, $D_1 = 0$, $D_n = (n-1)(D_{n-1} + D_{n-2})$

Fórmula de Cayley Hay n^{n-2} árboles de recubrimiento en un grafo completo de n vértices. En un grafo bipartito completo son $m^{n-1}n^{m-1}$.

1.7 Fast Fourier Transform

FFT.cpp

Descripción: Calcula el transformado y el inverso de un polinomio. Para multiplicar dos polinomios: $a * b = F^{-1}\{F\{a\} * F\{b\}\}$
Uso: `fft(a, 1)` $\equiv F\{a\}$; `fft(a, -1)` $\equiv F^{-1}\{a\}$. `Incluye <complex>`
Tiempo: $\mathcal{O}(n \log n)$

```
void fft(vector<complex<double> > &a, int sign = 1) {
  int n = a.size(); // n should be a power of two
  float theta = 8 * sign * atan(1.0) / n;
  for (int i = 0, j = 1; j < n - 1; ++j) {
    for (int k = n >> 1; k >(i ^= k); k >>= 1);
    if (j < i) swap(a[i], a[j]);
  }
  for (int m, mh = 1; (m = mh << 1) <= n; mh = m) {
    int irev = 0;
    for (int i = 0; i < n; i += m) {
      complex<double>  w = exp(complex<double>(0, theta*irev));
      for (int k = n >> 2; k >(irev ^= k); k >>= 1);
      for (int j = i; j < mh + i; ++j) {
        int k = j + mh;
        complex<double>  x = a[j] - a[k];
        a[j] += a[k];
        a[k] = w * x;
      }
    }
  }
  if (sign == -1) {
    for (i = 0; i < n; i++)
      a[i] /= n;
  }
  return;
}
```

Geometría (2)

2.1 Trigonometría

$$\sin(\alpha \pm \beta) = \sin\alpha\cos\beta \pm \cos\alpha\sin\beta$$
$$\cos(\alpha \pm \beta) = \cos\alpha\cos\beta \mp \sin\alpha\sin\beta$$
$$\tan(\alpha \pm \beta) = \frac{\tan\alpha \pm \tan\beta}{1 \mp \tan\alpha\tan\beta}$$
$$\sin\alpha + \sin\beta = 2\sin\frac{\alpha+\beta}{2}\cos\frac{\alpha-\beta}{2}$$
$$\cos\alpha + \cos\beta = 2\cos\frac{\alpha+\beta}{2}\cos\frac{\alpha-\beta}{2}$$

2.2 Triángulos

En un triángulo de lados a, b, c, alturas h_a, h_b, h_c, medianas m_a, m_b, m_c, semiperímetro $s = \frac{a+b+c}{2}$ y área A.

Heron-like area formulae

$$\begin{aligned} A &= \sqrt{s(s-a)(s-b)(s-c)} \\ &= \left(4\sqrt{\eta(\eta-h_a^{-1})(\eta-h_b^{-1})(\eta-h_c^{-1})}\right)^{-1} \\ &= \frac{4}{3}\sqrt{s(s-m_a)(s-m_b)(s-m_c)} \end{aligned}$$

donde $\eta = \frac{1}{2}(h_a^{-1} + h_b^{-1} + h_c^{-1})$

Circunradio $R = \frac{abc}{4A}$

Inradio $r = \frac{A}{s}$

Medianas dividen en dos subtriángulos de igual área.

$$m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2} \qquad \text{(teorema de Apolonio)}$$

Bisectrices dividen a los ángulos en dos partes iguales.

$$b_a = \sqrt{bc\left(1 - \left(\frac{a}{b+c}\right)^2\right)}$$

Teorema del seno $\frac{\sin\alpha}{a} = \frac{\sin\beta}{b} = \frac{\sin\gamma}{c} = \frac{1}{2R}$

Teorema del coseno $a^2 = b^2 + c^2 - 2bc\cos\alpha$

Teorema de la tangente $\frac{a+b}{a-b} = \frac{\tan\frac{\alpha+\beta}{2}}{\tan\frac{\alpha-\beta}{2}}$

2.3 Cuadriláteros

2.3.1 Cuadriláteros cíclicos

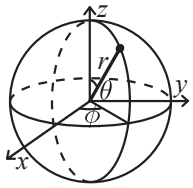
Sea un cuadrilátero **cíclico** de lados a, b, c, d y diagonales e, f. Entonces $ef = ac + bd$ y el área A se puede calcular con la

Fórmula de Brahmagupta

$$A = \sqrt{(s-a)(s-b)(s-c)(s-d)}$$

donde $s = \frac{a+b+c+d}{2}$

2.4 Coordenadas esféricas



$$x = r\sin\theta\cos\phi \qquad r = \sqrt{x^2+y^2+z^2}$$
$$y = r\sin\theta\sin\phi \qquad \theta = \texttt{acos}(z/\sqrt{x^2+y^2+z^2})$$
$$z = r\cos\theta \qquad \phi = \texttt{atan2}(y,x)$$

2.5 Puntos y vectores

Point.h

Descripción: `struct` para representar puntos y vectores en el plano. `T` puede ser `double` o `ll`, por ejemplo (por lo general evita `int`). Si usas enteros, asegúrate de que aunque los datos te los den en coordenadas enteras, todos los cálculos que hagas con ellos también.

```
template <class T>
struct Point {
  typedef Point P;
  T x, y;
  explicit Point(T x=0, T y=0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
  bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
  P operator+(P p) const { return P(x+p.x, y+p.y); }
  P operator-(P p) const { return P(x-p.x, y-p.y); }
  P operator*(T d) const { return P(x*d, y*d); }
  P operator/(T d) const { return P(x/d, y/d); }
  T dot(P p) const { return x*p.x + y*p.y; }
  T cross(P p) const { return x*p.y - y*p.x; }
  T cross(P a, P b) const { return (a-*this).cross(b-*this); }
  T dist2() const { return x*x + y*y; }
  double dist() const { return sqrt((double)dist2()); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this/dist(); } // makes dist()=1
  P perp() const { return P(-y, x); } // rotates +90 degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around the origin
  P rotate(double a) const {
    return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
```

pointVec.cpp

Descripción: `structs` para representar puntos y vectores en el plano como en el Halim, con algunas operaciones básicas.

```
#define EPS 1e-9

struct point {
    double x, y;
    point() {x = y = 0.0;}
    point(double _x, double _y) : x(_x), y(_y) {}
    // to sort points lexicographically
    bool operator < (point other) const {
        if (fabs(x - other.x) > EPS)
            return x < other.x;
        return y < other.y;
    }
    bool operator != (const point other) const {
        return x != other.x || y != other.y;
    }
};

struct vec {
    double x, y;
    vec(double _x, double _y) : x(_x), y(_y) { }
};

vec toVec(point a, point b) { return vec(b.x-a.x, b.y-a.y); }

vec scale(vec v, double s) { return vec(v.x*s, v.y*s);}

point translate(point p, vec v) { return point(p.x + v.x, p.y + v.y);}

double dot(vec a, vec b) { return a.x*b.x + a.y*b.y;}

double norm_sq(vec v) { return v.x*v.x + v.y*v.y;}

double dist(point p1, point p2) { return hypot(p1.x - p2.x, p1.y - p2.y);}

double distSq(point p1, point p2) {
    return (p1.x - p2.x)*(p1.x - p2.x)+ (p1.y - p2.y)*(p1.y - p2.y);
}

double angle(point a, point o, point b) { // returns angle aob in rad
   vec oa = toVec(o, a), ob = toVec(o, b);
   return acos(dot(oa, ob) / sqrt(norm_sq(oa) * norm_sq(ob))); }

double cross(vec a, vec b) { return a.x * b.y - a.y * b.x; }

// para aceptar puntos colineales cambia a >=
// returns true if point r is on the left side of line pq
bool ccw(point p, point q, point r) {
    return cross(toVec(p, q), toVec(p, r)) > 0;
}

bool collinear(point p, point q, point r) {
    return fabs(cross(toVec(p, q)), toVec(p, r)) < EPS;
}

int insideCircle(point p, point c, double r) {
    double eucSq = distSq(p, c);
    double rSq = r*r;
    if (fabs(eucSq - rSq) > EPS && eucSq < rSq) return 0; // inside
    else if (fabs(eucSq - rSq) < EPS) return 1; // border
    else return 2; // outside
}
```

2.6 Rectas y segmentos

lines.cpp

Descripción: `struct` para representar rectas en el plano $ax + by + c = 0$, con algunas operaciones básicas.

```
struct line {
    double a, b, c; // ax + by + c = 0

    bool operator== (const line &l2) {
        return (a == l2.a && b == l2.b && c == l2.c);
    }
};

bool areParallel(line l1, line l2) {
    return l1.a == l2.a && l1.b == l2.b;
}

line pointsToLine(point p, point q) {
    line l;
    if (p.x == q.x) {
        l.a = 1; l.b = 0; l.c = -p.x;
    } else {
        l.a = (q.y - p.y) / (p.x - q.x);
        l.b = 1;
        l.c = -(l.a * p.x) - p.y;
    }
    return l;
}
```

lineIntersect.cpp

Descripción: Devuelve `true` si las rectas `l1` y `l2` no son paralelas, y en tal caso devuelve en `p` el punto de intersección.

```
// Solves the linear system to find the intersection of two lines
bool areIntersect(line l1, line l2, point &p) {
    if (areParallel(l1, l2)) return false; // no intersection
    p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a * l2.b);
    if (fabs(l1.b) > EPS) p.y = -(l1.a * p.x + l1.c);
    else                  p.y = -(l2.a * p.x + l2.c);
    return true;
}
```

segmentIntersect.cpp

Descripción: Devuelve `true` si los segmentos `a` y `b` intersecan.
Si los segmentos intersecan, devuelve en `x` un punto de intersección.
Si los segmentos intersecan, devuelve en `collinear` si los segmentos son colineales. Si lo son, hay o bien un único punto de intersección (los segmentos se tocan en un extremo), o infinitos (los segmentos se solapan). Si no son colineales, el punto de intersección es único.
Un segmento está representado por $p + \lambda\vec{r}, \lambda \in [0, 1]$.

```
struct segment {
    point p;
    vec r;
    segment() { }
    segment(point a, point b) {
        p = a;
        r = toVec(a, b);
    }
};

bool intersect(segment a, segment b, point &x, bool &collinear) {
    vec r = a.r, s = b.r;
    point p = a.p, q = b.p;
    vec pq = toVec(p, q);
    double pqxr = cross(pq, r), rxs = cross(r, s);
    bool parallel = eq(rxs, 0);
    collinear = parallel && eq(pqxr, 0);
    if (collinear) {
        double t0 = dot(pq, r) / dot(r, r);
        double t1 = t0 + dot(s, r) / dot(r, r);
        if (lt(dot(s, r), 0)) swap(t0, t1);

        return leq(max(t0, 0.0), min(t1, 1.0));
    } else if (parallel) return false;
    else {
        double t = cross(pq, s) / cross(r, s), u = cross(pq, r) / cross(r, s);
        x = translate(p, scale(r, t));

        return leq(0, t) && leq(t, 1) && leq(0, u) && leq(u, 1);
    }
}
```

distToLine.cpp

Descripción: Devuelve la distancia desde `p` a la línea que pasa por los puntos **distintos** `a` y `b`. El punto más cercano a `p` sobre la línea se guarda en `c`.

```
double distToLine(point p, point a, point b, point &c) {
    vec ap = toVec(a, p), ab = toVec(a, b);
    double u = dot(ap, ab) / norm_sq(ab);
    c = translate(a, scale(ab, u));
    return dist(p, c);
}
```

distToSegment.cpp

Descripción: Devuelve la distancia desde `p` al segmento que pasa por los puntos **distintos** `a` y `b`. El punto más cercano a `p` sobre el segmento se guarda en `c`.

```
double distToSegment(point p, point a, point b, point &c) {
    vec ap = toVec(a, p), ab = toVec(a, b);
    double u = dot(ap, ab) / norm_sq(ab);
    if (u < 0.0) {
        c = point(a.x, a.y);
        return dist(p, a);
    } else if (u > 1.0) {
        c = point(b.x, b.y);
        return dist(p, b);
    } else return distToLine(p, a, b, c);
}
```

2.7 Círculos

Circles.cpp
Descripción: Construcción de círculos notables: con dos puntos y el radio, circunferencias circunscrita e inscrita. A veces se necesitará el área: usa la fórmula de Herón.

```
// Devuelve por referencia el centro de un circulo que pasa por 2 puntos y tiene un
    radio dado
bool circle2PtsRad(point p1, point p2, double r, point &c) {
  double d2 = (p1.x - p2.x) * (p1.x - p2.x) +
              (p1.y - p2.y) * (p1.y - p2.y);
  double det = r * r / d2 - 0.25;
  if (det < 0.0) return false;
  double h = sqrt(det);
  c.x = (p1.x + p2.x) * 0.5 + (p1.y - p2.y) * h;
  c.y = (p1.y + p2.y) * 0.5 + (p2.x - p1.x) * h;
  return true;
}

// Radio de la circunferencia inscrita dadas las longitudes de los lados
double rInCircle(double ab, double bc, double ca) {
  return area(ab, bc, ca) / (0.5 * perimeter(ab, bc, ca)); }

// Devuelve el centro y el radio de la circ. inscrita de un triangulo (por ref)
bool inCircle(point p1, point p2, point p3, point &ctr, double &r) {
  r = rInCircle(p1, p2, p3);
  if (fabs(r) < EPS) return false; // no inCircle center
  line l1, l2;// compute these two angle bisectors
  double ratio = dist(p1, p2) / dist(p1, p3);
  point p = translate(p2, scale(toVec(p2, p3), ratio / (1 + ratio)));
  pointsToLine(p1, p, l1);
  ratio = dist(p2, p1) / dist(p2, p3);
  p = translate(p1, scale(toVec(p1, p3), ratio / (1 + ratio)));
  pointsToLine(p2, p, l2);
  areIntersect(l1, l2, ctr);// get their intersection point
  return true;
}

// Radio de la circunferencia circunscrita dadas las longitudes de los lados
double rCircumCircle(double ab, double bc, double ca) {
  return ab * bc * ca / (4.0 * area(ab, bc, ca)); }

// Devuelve el centro y el radio de la circ. circunscrita de un triangulo (por ref)
bool circumCircle(point p1, point p2, point p3, point &ctr, double &r){
  double a = p2.x - p1.x, b = p2.y - p1.y, c = p3.x - p1.x, d = p3.y - p1.y;
  double e = a * (p1.x + p2.x) + b * (p1.y + p2.y);
  double f = c * (p1.x + p3.x) + d * (p1.y + p3.y);
  double g = 2.0 * (a * (p3.y - p2.y) - b * (p3.x - p2.x));
  if (fabs(g) < EPS) return false;
  ctr.x = (d*e - b*f) / g; ctr.y = (a*f - c*e) / g;
  r = dist(p1, ctr);  // r = distance from center to 1 of the 3 points
  return true;
}

// Devuelve el ortocentro de un triangulo. r no tiene significado
bool orthocenter(point a, point b, point c, point & ctr, double & r){
  return circumcenter(a+b-c,b+c-a,c+a-b, ctr, r);
}
```

tangentPoints.cpp
Descripción: Dado un punto `p`, fuera del círculo $x^2 + y^2 = r^2$, devuelve en el array `pt` los dos puntos de tangencia de las rectas tangentes al círculo desde `p`.

```
void tangentPoints(point p, double r, point pt[]) {
    double p0 = p.x, p1 = p.y;
    double den = p0*p0 + p1*p1;
    double dis = sqrt(p0*p0 + p1*p1 - r*r);
    pt[0] = point((p0*r*r - dis*p1*r)/den, (p1*r*r + dis*p0*r)/den);
    pt[1] = point((p0*r*r + dis*p1*r)/den, (p1*r*r - dis*p0*r)/den);
}
```

circle3pts.cpp
Descripción: Dados tres puntos no alineados, devuelve el centro del único círculo que pasa por esos tres puntos.

```
point center(point p1, point p2, point p3) {
    double c1n = 0.5*((p1.y - p3.y)*p2.y*p2.y + p1.x*p1.x*p3.y - p2.x*p2.x*p3.y + p1
        .y*p1.y*p3.y + (p2.x*p2.x - p3.x*p3.x - p3.y*p3.y)*p1.y - (p1.x*p1.x - p3.x
        *p3.x + p1.y*p1.y - p3.y*p3.y)*p2.y);
    double c1d = (p2.x - p3.x)*p1.y - (p1.x - p3.x)*p2.y + p1.x*p3.y - p2.x*p3.y;

    double c2n = -0.5*((p1.x - p3.x)*p2.x*p2.x + p1.x*p1.x*p3.x - (p2.x - p3.x)*p1.y
        *p1.y + (p1.x - p3.x)*p2.y*p2.y - (p3.x*p3.x + p3.y*p3.y)*p1.x - (p1.x*p1.x
         - p3.x*p3.x - p3.y*p3.y)*p2.x);
    double c2d = (p2.x - p3.x)*p1.y - (p1.x - p3.x)*p2.y + p1.x*p3.y - p2.x*p3.y;

    return point(c1n / c1d, c2n / c2d);
}
```

2.8 Polígonos

Teorema de Pick Sea un polígono simple cuyos vértices tienen coordenadas enteras. Si B es el número de puntos enteros en el borde, I el número de puntos enteros en el interior del polígono, entonces el área $A = I + \frac{B}{2} - 1$.

En lo que sigue, representamos un polígono por un vector de puntos dados en sentido antihorario en el que el primer punto y el último punto son el mismo (i.e. `P[0] = P[n - 1]`).

areaPerimeter.cpp
Descripción: Calcula el área y el perímetro del polígono `P`.

```
double area(const vector<point> &P) {
    int result = 0, x1, y1, x2, y2;
    for (int i = 0; i < P.size()-1; i++) {
        x1 = P[i].x; x2 = P[i+1].x;
        y1 = P[i].y; y2 = P[i+1].y;
        result += (x1 * y2 - x2 * y1);
    }
    return ((double) abs(result)) / 2;
}

double perimeter(vector<point> &P) {
```

```
    double result = 0.0;
    for (int i = 0; i < P.size()-1; i++) // remember that P[0] == P[n-1]
        result += dist(P[i], P[i+1]);
    return result;
}
```

inPolygon.cpp

Descripción: Devuelve `true` si el punto `pt` está dentro del polígono `P`. ¿Qué pasa si `pt` está sobre una arista del polígono?

Tiempo: $\mathcal{O}(n)$

```
bool inPolygon(point pt, const vector<point> &P) {
    if (P.size() <= 3) return false;
    double sum = 0;
    for (int i = 0; i < P.size() - 1; i++) {
        if (ccw(pt, P[i], P[i + 1])) sum += angle(P[i], pt, P[i + 1]);
        else sum -= angle(P[i], pt, P[i + 1]);
    }
    return fabs(fabs(sum) - 2*PI) < EPS; // sum sera negativo si los vertices se
        recorrieron clockwise
}
```

inConvexPolygon.cpp

Descripción: Devuelve `true` si el punto `q` está en dentro del polígono **convexo** (dado en sentido antihorario).

Tiempo: $\mathcal{O}(\log n)$

```
double cross(point a, point b, point c) { return cross(toVec(a, b), toVec(b, c)); }

// O(log n) Solo para convexos (ordenados ccw)!
bool inConvexPolygon(point q, vector<point> const &p) {
  if (cross(p[0], p[1], q) < 0 || cross(p[p.size() - 2], p[0], q)<0)
    return false;
  int ini = 1, fin = p.size() - 2, mid;
  while (ini != fin - 1) {
    mid = (ini + fin) / 2;
    if (cross(p[0], p[mid], q) < 0) fin = mid;
    else ini = mid;
  }
  if (cross(p[ini], p[fin], q)<0) return false;
  return true;
}
```

cutPolygon.cpp

Descripción: Corta el polígono `Q` con la recta dada por los puntos `a` y `b`. Devuelve la parte izquierda del corte (`a` y `b` formarán parte del polígono a devolver).

Para hacer la programación más simple se hace uso de `lineIntersectSeg`, que interseca un segmento con una recta.

```
// line segment p-q intersect with line A-B.
point lineIntersectSeg(point p, point q, point A, point B) {
    double a = B.y - A.y, b = A.x - B.x, c = B.x * A.y - A.x * B.y;
    double u = fabs(a * p.x + b * p.y + c), v = fabs(a * q.x + b * q.y + c);
    return point((p.x * v + q.x * u) / (u+v), (p.y * v + q.y * u) / (u + v));
}
```

```
// cuts polygon Q along the line formed by point a -> point b. Q[0]==Q.back()
vector<point> cutPolygon(point a, point b, const vector<point> &Q) {
    vector<point> P;
    for (int i = 0; i < Q.size(); i++) {
        double left1 = cross(toVec(a, b), toVec(a, Q[i])), left2 = 0;
        if (i != Q.size() - 1) left2 = cross(toVec(a, b), toVec(a, Q[i + 1]));
        if (left1 > -EPS) P.push_back(Q[i]);       // Q[i] is on the left of ab
        if (left1 * left2 < -EPS)         // edge (Q[i], Q[i+1]) crosses line ab
            P.push_back(lineIntersectSeg(Q[i], Q[i + 1], a, b));
    }
    if (!P.empty() && !(P.back() == P.front()))
        P.push_back(P.front());        // make P's first point = P's last point
    return P;
}
```

grahamScan.cpp

Descripción: Calcula la envolvente convexa de una nube de puntos.

No da el resultado correcto si `P` son tres puntos alineados.

Da excepción en ejecución si `P` empieza con cuatro puntos alineados.

Precisa de `angleCmp` para ordenar los puntos según el ángulo que formen con el eje x.

Tiempo: $\mathcal{O}(n \log n)$

```
bool angleCmp(point a, point b) {
    if (collinear(pivot, a, b))
        return distSq(pivot, a) < distSq(pivot, b); // cuidado con el overflow de
            distSq, cambiar a hypot si crees que puede haber overflow

    int d1x = a.x - pivot.x, d1y = a.y - pivot.y;
    int d2x = b.x - pivot.x, d2y = b.y - pivot.y;
    return (atan2(d1y, d1x) - atan2(d2y, d2x)) < 0;
}

vector<point> graham(vector<point> &P) { // the content of P may be reshuffled
    int n = P.size();
    if (n <= 3) {
        if (P[Ca0] != P[n-1]) P.push_back(P[0]);
        return P; // CH is P itself
    }

    // first, find P0 = point with lowest Y and if tie: rightmost X
    int P0 = 0;
    for (int i = 1; i < n; i++)
        if (P[i].y < P[P0].y || (P[i].y == P[P0].y && P[i].x > P[P0].x))
            P0 = i;

    point temp = P[0]; P[0] = P[P0]; P[P0] = temp; // swap P[P0] with P[0]

    // second, sort points by angle w.r.t pivot P0
    pivot = P[0];
    sort(++P.begin(), P.end(), angleCmp); // we do not sort P[0]

    // third, the ccw tests
    vector<point> S;
    S.push_back(P[n-1]);
    S.push_back(P[0]);
    S.push_back(P[1]);
```

```
    int i = 2;
    while (i < n) {
        int j = S.size()-1;
        if (ccw(S[j-1], S[j], P[i])) S.push_back(P[i++]); // left turn, accept
        else S.pop_back(); // or pop the top of S until we have a left turn
    }

//     add these two lines if you want P0 to be the first point in the convex hull
//    S.erase(S.begin());
//    S.push_back(*S.begin());

    return S;
}
```

andrewsMonotoneChain.cpp
Descripción: Devuelve un polígono con la envolvente convexa de una nube de puntos.
Tiempo: $\mathcal{O}(n \log n)$

```
vector<point> andrew(vector<point> &P) {
    int n = P.size(), k = 0;
    vector<point> H(2*n);

    sort(P.begin(), P.end());

    // Build lower hull
    for (int i = 0; i < n; i++) {
        while (k >= 2 && !ccw(H[k-2], H[k-1], P[i])) k--;
        H[k++] = P[i];
    }

    // Build upper hull
    for (int i = n-2, t = k+1; i >= 0; i--) {
        while (k >= t && !ccw(H[k-2], H[k-1], P[i])) k--;
        H[k++] = P[i];
    }

    H.resize(k);
    return H;
}
```

2.9 Misc.

TODO: Apotema, teorema de stewart, volúmenes de pirámides, cone frustums...

closestPoints.cpp
Descripción: Devuelve la closest pair distance de una nube de puntos. Esquema útil para algoritmos de divide y vencerás.
Tiempo: $\mathcal{O}(n \log n)$

```
double cp(int l, int r) {
    if (l >= r) return DBL_MAX;

    int mid = (l + r) / 2;
    double d = min(cp(l, mid), cp(mid + 1, r));

    vector<point> strip;
    for (int i = l; i <= r; i++)
        if (fabs(p[i].x - p[mid].x) <= d)
            strip.push_back(p[i]);

    sort(strip.begin(), strip.end(), cmpY);

    for (int i = 0; i < strip.size(); i++)
        for (int j = i+1; j < strip.size() && (strip[j].y - strip[i].y) <= d; j++)
            d = min(d, dist(strip[i], strip[j]));

    return d;
}
```

2.9.1 Great Circle distance

sphericalDistance.cpp

```
double sphericalDistance(double f1, double t1,
  double f2, double t2, double radius) {
  double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
  double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
  double dz = cos(t2) - cos(t1);
  double d = sqrt(dx*dx + dy*dy + dz*dz);
  return radius*2*asin(d/2);
}
```

SphericalDistance.cpp
Descripción: Calcula la distancia entre 2 puntos sobre una esfera dados sus ángulos ϕ y θ y el radio de la esfera. Para ello, pasa primero a cartesianas.

```
double sphericalDistance(double f1, double t1,
  double f2, double t2, double radius) {
  double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
  double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
  double dz = cos(t2) - cos(t1);
  double d = sqrt(dx*dx + dy*dy + dz*dz);
  return radius*2*asin(d/2);
}
```

Estructuras de datos (3)

3.1 UFDS

ufds.cpp
Tiempo: $\mathcal{O}(\alpha(n))$

```
struct ufds {
    int numSets;
    vector<int> p;

    ufds(int N) : numSets(N), p(N, -1) { }
```

```
    int findSet(int i) { // finds and path compresses if possible
        return (p[i] < 0) ? i : (p[i] = findSet(p[i]));
    }

    bool isSameSet(int i, int j) {
        return findSet(i) == findSet(j);
    }

    void unionSet(int i, int j) {
        int x = findSet(i), y = findSet(j);
        if (x == y) return;
        if (p[x] < p[y]) {
            p[x] += p[y];
            p[y] = x;
        } else {
            p[y] += p[x];
            p[x] = y;
        }
        --numSets;
    }

    int sizeSet(int i) { // returns size of the set to which element i belongs
        return -p[findSet(i)];
    }
};
```

3.2 Fenwick Tree

FenwickTree.cpp
Tiempo: $\mathcal{O}(\log n)$

```
#define LSOne(S) (S&&(-S))

struct FenwickTree {
  vi ft;
  FenwickTree(int n) {ft.assign(n + 1, 0);}
  int rsq(int b) {
    int t = 0;
    for (; b; b -= LSOne(b))
      t += ft[b];
    return t;
  }
  int rsq(int a, int b){return rsq(b) - (a == 1 ? 0 : rsq(a - 1));}
  void update(int k, int v) {
    for (; k < ft.size(); k += LSOne(k))
      ft[k] += v;
  }
  int lower_bound(int sum) {// min pos st sum of [0 , pos ] >= sum
  // Returns n i f no sum is >= sum, or ..1 i f empty sum is .
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1 << 25; pw; pw >>= 1) {
      if (pos + pw <= s.size() && s[pos + pw-1] < sum)
        pos += pw, sum -= s[pos-1];
    }
    return pos;
  }
};
```

3.3 Segment Tree

SegmentTree.cpp
Descripción: El rsq es prescindible, se hace con Fenwick
Tiempo: $\mathcal{O}(\log n)$

```
#include <vector>
using namespace std;

typedef vector<int> vi;

// Dynamic answers to RangeMinimumQuery
class SegmentTree {
private :
  vi A, st; // A->data, st->tree (stores indices)
  int n; // length of the interval
  int left(int p) { return p << 1; }
  int right(int p) { return (p << 1) + 1; }

  void build(int p, int L, int R) { // O(n)
    if (L == R)
      st[p] = L;
    else {
      build(left(p), L, (L + R) / 2);
      build(right(p), (L + R) / 2 + 1, R);
      int p1 = st[left(p)], p2 = st[right(p)];
      st[p] = (A[p1] <= A[p2]) ? p1 : p2;
    }
  }

  int rmq(int p, int L, int R, int i, int j) { // O(log n)
    // [L, R] -> interval of the node of the tree in which we are (p)
    // [i, j] -> interval of the query
    if (i > R || j < L) return -1; // totally outside
    if (L >= i && R <= j) return st[p]; // query is larger

    // if (st[p]>=i && st[p]<=j) return st[p]; better?
    // Si es el minimo de un segmento mas largo, lo sera de uno contenido en ese

    int p1 = rmq(left(p), L, (L + R) / 2, i, j);
    int p2 = rmq(right(p), (L + R) / 2 + 1, R, i, j);

    if (p1 == -1) return p2;
    if (p2 == -1) return p1;
    return (A[p1] <= A[p2]) ? p1 : p2;
  }

  int updatePoint(int p, int L, int R, int idx, int new_value) { // O(log n)
    if (idx < L || idx > R) return st[p]; // no change
    if (L == idx && R == idx) { // leaf
      A[idx] = new_value; // we change it here to go up changing
      return st[p];
    }
    int p1 = updatePoint(left(p), L, (L + R) / 2, idx, new_value);
    int p2 = updatePoint(right(p), (L + R) / 2 + 1, R, idx, new_value);
```

```
    return st[p] = (A[p1] <= A[p2]) ? p1 : p2;
  }

public :
  SegmentTree(const vi & a) {
    A = a; n = (int)a.size();
    st.assign(4 * n, 0);
    build(1, 0, n - 1);
  }

  int rmq(int i, int j) {
    return rmq(1, 0, n - 1, i, j);
  }

  void updatePoint(int idx, int new_value) {
    updatePoint(1, 0, n - 1, idx, new_value);
  }
};

#define INF 1e9

// Dynamic answers to RangeSumQuery
class SegmentTree2 {
private:
  vi A, st; // A->data, st->tree
  int n; // length of the interval
  int left(int p) { return p << 1; }
  int right(int p) { return (p << 1) + 1; }

  void build(int p, int L, int R) { // O(n)
    if (L == R)
      st[p] = A[L];
    else {
      build(left(p), L, (L + R) / 2);
      build(right(p), (L + R) / 2 + 1, R);
      st[p] = st[left(p)] + st[right(p)];
    }
  }

  int rsq(int p, int L, int R, int i, int j) { // O(log n)
    // [L, R] -> interval of the node of the tree in which we are (p)
    // [i, j] -> interval of the query
    if (i > R || j < L) return -INF; // totally outside
    if (L >= i && R <= j) return st[p]; // query is larger

    int p1 = rsq(left(p), L, (L + R) / 2, i, j);
    int p2 = rsq(right(p), (L + R) / 2 + 1, R, i, j);

    if (p1 == -INF) return p2;
    if (p2 == -INF) return p1;
    return p1 + p2;
  }

  // Real update! not only adding, it is modifying! (better than Fenwick)
  int updatePoint(int p, int L, int R, int idx, int new_value) { // O(log n)
    if (idx < L || idx > R) return st[p]; // no change
    if (L == idx && R == idx) { // leaf
      A[idx] = new_value; // we change it here to go up changing
      return st[p] = new_value;
    }
    int p1 = updatePoint(left(p), L, (L + R) / 2, idx, new_value);
    int p2 = updatePoint(right(p), (L + R) / 2 + 1, R, idx, new_value);
    return st[p] = p1 + p2;
  }

public:
  SegmentTree2(const vi & a) {
    A = a; n = (int)a.size();
    st.assign(4 * n, 0);
    build(1, 0, n - 1);
  }

  int rsq(int i, int j) {
    return rsq(1, 0, n - 1, i, j);
  }

  void updatePoint(int idx, int new_value) {
    updatePoint(1, 0, n - 1, idx, new_value);
  }
};
```

Grafos (4)

4.1 Recorridos

dfs.cpp
Tiempo: $\mathcal{O}(V+E)$

```
void dfs(int u, vvi &adjList, vi &dfs_num, vi &topo) {
    dfs_num[u] = 1;
    for (int v : adjList[u]) {
        if (dfs_num[v] == 0)
            dfs(v, adjList, dfs_num, topo);
    }
    topo.push_back(u); // Read topo in reverse order.
}
```

bfs.cpp
Descripción: Recorrido en anchura. También resuelve SSSP si el grafo es sin pesos. Multi-source BFS: mete todos los nodos fuentes a una `queue`.
Tiempo: $\mathcal{O}(V+E)$

```
void bfs(vvi &adjList, int u, vi &dist) {
    dist[u] = 0;
    queue<int> q;
    q.push(u);
    while (!q.empty()) {
        u = q.front(); q.pop();
        for (auto v : adjList[u]) {
            if (dist[v] == INT_MAX) {
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
```

```
    }
}
```

floodfill.cpp
Descripción: Hace floodfill de la componente conexa del vértice (`r`, `c`) de color `c1` coloreándola de color `c2`.

```
// N, E, S, W
int dr[] = {-1,0,1,0};
int dc[] = {0,1,0,-1};

// S, SE, E, NE, N, NW, W, SW
int dr[] = {1,1,0,-1,-1,-1,0,1};
int dc[] = {0,1,1,1,0,-1,-1,-1};

int floodfill(int r, int c, int c1, int c2) {
    if (r < 0 || r >= R || c < 0 || c >= C) return 0;
    if (grid[r][c] != c1) return 0;
    int ans = 1;
    grid[r][c] = c2;
    for (int d = 0; d < 8; d++)
        ans += floodfill(r + dr[d], c + dc[d], c1, c2);
    return ans;
}
```

4.1.1 Graph Check

graphCheck.cpp
Descripción: Clasificación de tipos de aristas: tree edge (una arista del DFS spanning tree), back edge (una arista que forma parte de un ciclo) y forward cross edge (una arista de un vértice `EXPLORED` a un vértice `VISITED`).

```
bool graphCheck(int u, vvi &adjList, vi &dfs_num, vi &dfs_parent) {
    bool cycle = false;
    dfs_num[u] = EXPLORED;
    for (int j = 0; j < adjList[u].size(); j++) {
        int v = adjList[u][j];
        if (dfs_num[v] == UNVISITED) {
            dfs_parent[v] = u;
            // Pon cycle |= si quieres parar de explorar el grafo en cuanto se
                detecte un ciclo.
            cycle = graphCheck(v, adjList, dfs_num, dfs_parent) || cycle;
        } else if (dfs_num[v] == EXPLORED)
            if (v != dfs_parent[u]) // Back edge, graph is cyclic (quita esta linea
                si el grafo es dirigido).
                cycle = true; // Haz return si quieres parar de explorar el grafo en
                     cuanto se detecte un ciclo.
        else if (dfs_num[v] == VISITED) {
            // Forward cross edge.
        }
    }
    dfs_num[u] = VISITED;

    return cycle;
}
```

bipartiteGraphCheck.cpp
Descripción: Intenta bicolorear un grafo para detectar si es bipartito.

```
vi color;

bool is_bipartite(vvi &adjList, int s, int &white, int &black) {
    queue<int> q;
    q.push(s);
    color[s] = 0;
    black = 1;
    white = 0;
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int v : adjList[u]) {
            if (color[v] == -1) {
                color[v] = 1 - color[u];
                if (color[v] == 0) black++;
                else white++;
                q.push(v);
            } else if (color[v] == color[u]) return false;
        }
    }
    return true;
}
```

4.1.2 Articulation points and bridges en grafos no dirigidos

En un grafo no dirigido G, un *articulation point* es un vértice que al ser eliminado (junto con sus aristas) desconecta G.
Un *bridge* es una arista que al ser eliminada desconecta G.

articulationPointsAndBridges.cpp
Descripción: Encuentra articulation points and bridges. `dfs_num[u]` es el número de iteración en el que se visita u *por primera vez*. `dfs_low[u]` es el valor de `dfs_num[v]` más bajo de un vértice v alcanzable desde el subárbol de expansión del recorrido DFS que empieza en u.
Caso especial: la raíz del árbol DFS (`dfsRoot`) es un articulation point si y sólo si tiene más de un hijo (`rootChildren > 1`).
Tiempo: $\mathcal{O}(V+E)$

```
vvi adjList;
set<ii> bridges;
vi dfs_num, dfs_low, dfs_parent;
vector<bool> ap;
int dfsCounter, dfsRoot, rootChildren;

void findAp(int u) {
    dfs_num[u] = dfs_low[u] = dfsCounter++;
    for (int j = 0; j < adjList[u].size(); j++) {
        int v = adjList[u][j];
        if (dfs_num[v] == UNVISITED) {
            dfs_parent[v] = u;
            if (u == dfsRoot) rootChildren++;
```

```
            findAp(v);

            ap[u] = ap[u] | (dfs_low[v] >= dfs_num[u]);
            dfs_low[u] = min(dfs_low[u], dfs_low[v]);
        } else if (dfs_parent[v] != u) // A back edge.
            dfs_low[u] = min(dfs_low[u], dfs_num[v]);
    }
}

void findBridges(int u) {
    dfs_num[u] = dfs_low[u] = dfsCounter++;
    for (int j = 0; j < adjList[u].size(); j++) {
        int v = adjList[u][j];
        if (dfs_num[v] == UNVISITED) {
            dfs_parent[v] = u;

            findBridges(v);

            if (dfs_low[v] > dfs_num[u])
                bridges.insert(ii(min(u, v), max(u, v)));
            dfs_low[u] = min(dfs_low[u], dfs_low[v]);
        } else if (v != dfs_parent[u]) // A back edge.
            dfs_low[u] = min(dfs_low[u], dfs_num[v]);
    }
}

// Como usarlos
int main() {
    dfsCounter = 0;
    dfs_num.assign(V, UNVISITED); dfs_low.assign(V, 0); dfs_parent.assign(V, 0);
    for (int u = 0; u < V; u++)
        if (dfs_num[u] = UNVISITED) {
            dfsRoot = u; rootChildren = 0; findAp(u);
            ap[dfsRoot] = rootChildren > 1;
        }
}
```

4.2 SSSP

dijkstra.cpp
Descripción: SSSP desde el vértice `s` en un grafo dirigido con pesos *sin ciclos de coste negativo*. `adjList[u][j]` es un par (`w`, `v`) donde `w` es el peso de la arista $u \to v$.
Tiempo: $\mathcal{O}((V+E)\log V)$, probablemente TLE si $V, E > 300K$.

```
void dijkstra(int s, vector<vii> const& grafo, vi &dist) {
    dist.assign(adjList.size(), numeric_limits<int>::max());
    dist[s] = 0;
    priority_queue<ii, vii, greater<ii>> pq;
    pq.push({0,s});
    while (!pq.empty()) {
        ii front = pq.top(); pq.pop();
        int d = front.first, u = front.second;
        if (d > dist[u]) continue;
        for (auto v : grafo[u]) {
            if (dist[u] + v.first < dist[v.second]) {
                dist[v.second] = dist[u] + v.first;
               pq.push({dist[v.second], v.second});
            }
        }
    }
}
```

bellmanFord.cpp
Descripción: SSSP desde el vértice `s` en un grafo dirigido con pesos. Devuelve si existe al menos un ciclo de coste negativo.
Si quieres saber los vértices que están en un ciclo de coste negativo, puedes encontrar las SCCs de los vértices v con `relax[v] = 1`.
Si quieres saber los vértices hasta los que puedes llegar con coste negativo, recorre el grafo desde los vértices v con `relax[v] = 1`.
Tiempo: $\mathcal{O}(VE)$, probablemente TLE si $VE > 10^6$.

```
bool bellman_ford(int s, vector<vector<pair<int,int>>> const& grafo,
                  vector<int> & dist, vector<bool> & en_ciclo) {
   dist.assign(grafo.size(), INF);
   en_ciclo.assign(grafo.size(), false);
   dist[s] = 0;
   bool changed = true;
   for (int i = 0; i < grafo.size() && changed; ++i) {
      changed = false;
      for (int u = 0; u < grafo.size(); ++u) {
         for (auto & a : grafo[u]) {
            if (dist[a.first] > dist[u] + a.second) {
               dist[a.first] = dist[u] + a.second;
               changed = true;
               if (i == grafo.size() - 1) en_ciclo[a.first] = true;
            }
         }
      }
   }
   return changed;
}
```

4.3 APSP

floyd.cpp
Descripción: APSP en un grafo dirigido con pesos en matriz de adyacencia `adjMat` de `N` vértices.
Asegúrate de inicializar `adjMat[i][i] = 0` y `adjMat[i][j] = INF` si no existe la arista $i \to j$ (no uses `INT_MAX` por overflow).
La matriz `pi` sirve para imprimir el camino mínimo: `pi[i][j]` es el vértice anterior a j en un camino mínimo de i a j; `i` $\to \ldots \to$ `pi[i][j]` $\to$ `j`.
Tiempo: $\mathcal{O}(V^3)$, probablemente TLE si $V > 400$

```
const int INF = 1000000000;

void floyd() {
    for (int k = 0; k < N; k++)
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                if (adjMat[i][k] + adjMat[k][j] < adjMat[i][j]) {
                    adjMat[i][j] = adjMat[i][k] + adjMat[k][j];
```

```
                pi[i][j] = pi[k][j];
            }
}
```

4.4 MST

kruskal.cpp
Descripción: Añade aristas vorazmente al MST hasta que todos los vértices están en el MST.
Ordena `edgeList`. Devuelve `INT_MAX` si el grafo no es conexo.
Tiempo: $\mathcal{O}(E \log E) \approx \mathcal{O}\left(E \log V^2\right) = \mathcal{O}(2E \log V) = \mathcal{O}(E \log V)$

```
int kruskal(vector<pair<int, ii>> &edgeList) {
    sort(edgeList.begin(), edgeList.end());
    int mst_cost = 0;
    ufds uf(V);

    for (int i = 0; i < E && uf.numSets != 1; i++) {
        pair<int, ii> front = edgeList[i];
        if (!uf.isSameSet(front.second.first, front.second.second)) {
            mst_cost += front.first;
            uf.unionSet(front.second.first, front.second.second);
        }
    }

    if (uf.numSets != 1) return INT_MAX; // this graph is not connected
    return mst_cost;
}
```

Problema del *minimax* (análogamente se define *maximin*): dados dos vértices de un grafo i y j, considera todos los caminos entre ellos, y asocia a cada camino un coste: el peso máximo de entre todas las aristas que forman el camino. ¿Cuál es el menor coste de estos caminos? Solución: un camino con coste mínimo es el que une i y j en el MST del grafo.

4.5 SCC

tarjan.cpp
Descripción: Calcula las SCCs de un grafo dirigido y las guarda en `sccs`. `cur_scc[v]` $= 1 \iff v$ está en la SCC que se está explorando actualmente, cuyos vértices se guardan en la "pila" `S`. Un vértice v es el "inicio" (según el recorrido del DFS) de una SCC $\iff$ `dfs_low[v]` $=$ `dfs_num[u]`.
Tiempo: $\mathcal{O}(E + V)$

```
const int UNVISITED = -1;
vector<int> S, dfs_num, dfs_low, cur_scc, component;
vector<vector<int>> grafo, sccs;
int V, dfsCounter = 0;

void tarjanSCC(int u) {
   dfs_num[u] = dfs_low[u] = dfsCounter++;
   cur_scc[u] = 1;
   S.push_back(u);
   for (int v : grafo[u]) {
      if (dfs_num[v] == UNVISITED)
         tarjanSCC(v);
      if (cur_scc[v] == 1) // El nodo v ha sido visitado, pero esta en la misma SCC
          que u? Podria tratarse de un forward edge a otra SCC.
         dfs_low[u] = min(dfs_low[u], dfs_low[v]);
   }

   // inserta aqui u en un vector para generar un reverse toposort si quieres.

   if (dfs_low[u] == dfs_num[u]) {
      vector<int> scc;
      int v, nc = (int)sccs.size();
      do {
         v = S.back(); S.pop_back();
         cur_scc[v] = 0;
         scc.push_back(v);
         component[v] = nc;
      } while (u != v);
      sccs.push_back(scc);
   }
}

// hallamos las SCCs (en resuelve())
dfsCounter = 0;
dfs_num.assign(V, UNVISITED); dfs_low.assign(V, 0); cur_scc.assign(V, 0);
component.assign(V, 0); S.clear(); sccs.clear();
for (int i = 0; i < V; ++i)
   if (dfs_num[i] == UNVISITED)
      tarjanSCC(i);
```

kosaraju.cpp
Descripción: Calcula el número de SCCs de un grafo dirigido. Dos vértices u y v están en la misma SCC si y sólo si se puede llegar desde u a v en el grafo original y desde v a u en el grafo traspuesto mediante un DFS. Para contar las SCC bien se inicia el recorrido en el grafo traspuesto desde los vértices que van "antes" en el grafo original, por lo que se calcula un toposort antes (no es un toposort estricto pues el grafo podría ser cíclico).
Tiempo: $\mathcal{O}(2(E + V))$

```
int kosaraju(vector<vector<int>> &adjList) {
   vector<int> dfs_num(adjList.size(), 0), topo;

   for (int u = 0; u < adjList.size(); u++)
      if (dfs_num[u] == 0)
         dfs(u, adjList, dfs_num, topo);

   // compute transpose graph
   vector<vector<int>> adjListT(adjList.size(), {});
   for (int u = 0; u < adjList.size(); u++)
      for (int v : adjList[u])
         adjListT[v].push_back(u);

   int scc = 0;
   dfs_num.assign(adjListT.size(), 0);
   for (int i = topo.size() - 1; i >= 0; i--) {
```

```
      if (dfs_num[topo[i]] == 0) {
         dfs(topo[i], adjListT, dfs_num, topo);
         scc++;
      }
   }
   return scc;
}
```

4.6 TopoSort

Véase DFS.

kahn.cpp
Descripción: Calcula un toposort de un grafo dirigido sin pesos. Devuelve `true` si el grafo es cíclico.
Tiempo: $\mathcal{O}(V+E)$

```
bool kahn(vvi &adjList, vi &topo) {
    vi inDeg(adjList.size(), 0);

    for (int u = 0; u < adjList.size(); u++)
        for (int v = 0; v < adjList[u].size(); v++)
            inDeg[adjList[u][v]]++;

    queue<int> q;
    for (int u = 0; u < inDeg.size(); u++)
        if (inDeg[u] == 0) q.push(u);

    while (!q.empty()) {
        int u = q.front(); q.pop();
        topo.push_back(u);
        for (int v = 0; v < adjList[u].size(); v++)
            if (--inDeg[adjList[u][v]] == 0) q.push(adjList[u][v]);
    }

    for (int u = 0; u < inDeg.size(); u++)
        if (inDeg[u] != 0) return true;
    return false;
}
```

4.7 MaxFlow

edmondsKarp.cpp
Descripción: Calcula el flujo máximo en una red de flujo con un único source node `s` y único sink node `t` implementando el método de Ford Fulkerson: usa BFS para encontrar un augmenting path. La matriz `res` contiene las capacidades residuales y se inicializa con las capacidades de las aristas. Pon en `adjList` las aristas en ambos sentidos.
Tiempo: $\mathcal{O}(VE^2)$

```
int V, f, s, t;  // global variables
vector<vector<int>> res; // matriz de adyacencia, con capacidades
vector<vector<int>> grafo; // listas de adyacentes, tambi n los back edges
vector<int> p; // parent

void augment(int v, int minEdge) {  // traverse BFS spanning tree from s to t
   if (v == s) { f = minEdge; return; }  // record minEdge in global var f
   else if (p[v] != -1) {
      augment(p[v], min(minEdge, res[p[v]][v])); // recursive
      res[p[v]][v] -= f; res[v][p[v]] += f;   // update
   }
}

// Edmonds Karp Max Flow O(VE^2)
int edmondsKarp() {
   int mf = 0;    // mf stands for max_flow
   while (1) {
      f = 0;
      // BFS
      vector<bool> marcado(V, false); marcado[s] = true;
      queue<int> q; q.push(s);
      p.assign(V, -1);  // record the BFS spanning tree, from s to t!
      while (!q.empty()) {
         int u = q.front(); q.pop();
         if (u == t) break;      // immediately stop BFS if we already reach sink t
         for (auto v : grafo[u])
            if (res[u][v] > 0 && !marcado[v]) {
               marcado[v] = true;
               q.push(v);
               p[v] = u;
            }
      }
      augment(t, INF);   // find the min edge weight 'f' along this path, if any
      if (f == 0) break;      // we cannot send any more flow ('f' = 0), terminate
      mf += f;                // we can still send a flow, increase the max flow!
   }
   return mf;
}
```

4.8 Grafos eulerianos

Un grafo no dirigido

- tiene un **ciclo euleriano** si y sólo si todos los vértices tienen grado par y todos los vértices con grados no nulo pertenecen a la misma componente conexa.
- tiene un **ciclo euleriano** si y sólo si se puede descomponer en edge-disjoint cycles y todos los vértices con grado no nulo pertenecen a la misma componente conexa.
- tiene un **camino euleriano** si y sólo si tiene exactamente cero o dos vértices con grado impar y los vértices con grado no nulo pertenecen a la misma componente conexa.

Un grafo dirigido

- tiene un **ciclo euleriano** si y sólo si cada vértice tiene `inDeg` = `outDeg` y todos los vértices con grado no nulo pertenecen a la misma componente fuertemente conexa.
- tiene un **ciclo euleriano** si y sólo si se puede descomponer en edge-disjoint cycles y todos los vértices con grado no nulo pertenecen a la misma componente conexa.
- tiene un **camino euleriano** si y sólo si tiene a lo sumo un vértice con `outDeg - inDeg` = 1, a lo sumo un vértice con `inDeg - outDeg` = 1, los demás vértices tienen `inDeg` = `outDeg`, y todos los vértices con grado no nulo pertenecen a la misma componente conexa del grafo no dirigido subyacente.

eulerTour.cpp

```
void eulerTour(list<int>::iterator it, int u) {
    for (int j = 0; j < adjList[u].size(); j++) {
        ii v = adjList[u][j];
        if (v.second == 1) {
            adjList[u][j].second = 0;
            for (int k = 0; k < adjList[v.first].size(); k++) {
                ii e = adjList[v.first][k];
                if (e.second == 1 && e.first == u) {
                    adjList[v.first][k].second = 0;
                    break;
                }
            }
            eulerTour(cyc.insert(it, u), v.first);
        }
    }
}
```

4.9 Matchings

Matching Un matching (o independent edge set) de un grafo G es un conjunto de aristas que no tienen vértices en común.

Augmenting path Dado un matching M en un grafo G, un augmenting path (camino de M-aumento) es un camino que empieza y acaba en unmatched vertices y cuyas aristas alternan entre aristas que están y no esan en M.

Lema de Berge Un matching M es máximo (contiene la mayor cantidad de aristas) si y sólo si no tiene un augmenting path.

Vertex cover Un vertex cover es un conjunto de vértices tales que cada arista del grafo es incidente a al menos un vértice del conjunto.

Conjunto independiente de vértices (independent set) Es un conjunto de vértices tales que no existe ninguna arista entre ellos.

4.9.1 Grafos bipartitos

Teorema de König En un grafo bipartito, el número de aristas en un matching máximo es igual al número de vértices en un recubrimiento mínimo de vértices (i.e. MaxCBM = MinVC).

MaxIS + MaxCBM = V en grafos bipartitos.

bergeMcbm.cpp
Descripción: Calcula un matching máximo en un grafo bipartito usando el lema de Berge. `M` es el tamaño de la partición izquierda del grafo, `N` la de la derecha. `adjList` es un grafo dirigido **de tamaño `M`** con las aristas dirigidas desde la partición izquierda a la derecha. `match` indica con qué vértices de la partición izquierda están emparejados los vértices de la partición derecha; tiene tamaño `N + M` pero sólo se indexa con los vértices de la partición derecha.
Tiempo: $\mathcal{O}(VE)$

```
int M, N; // M parte izquierda, N parte derecha
vector<unordered_set<int>> grafo; // dirigido, tama o M   sets para poder borrar
vector<int> match, vis;

int aug(int l) { // Devuelve 1 si encuentra un augmenting path para el matching M
    representado en match.
    if (vis[l]) return 0;
    vis[l] = 1;
    for (auto r : grafo[l]) {
         if (match[r] == -1 || aug(match[r])) {
            match[r] = l;
            return 1;
        }
    }
    return 0;
}

int berge_mcbm() {
    int mcbm = 0;
    match.assign(N + M, -1);
    vis.assign(M, 0);
    for (int l = 0; l < M; l++) {
        vis.assign(M, 0);
        mcbm += aug(l);
    }
    return mcbm;
}
```

DP (5)

5.1 Recurrencias de problemas clásicos

LIS 1- $LIS(0) = 1$. 2- $LIS(i) = max(LIS(j) + 1), \forall j \in [0..i-1]$ and $A[j] < A[i]$

Mochila 1- $val(id, 0) = 0$, cannot take anything else
2- $val(n, remW) = 0$, considered all items

3- if $W[id] > remW$, we have no choice but to ignore this item:
$val(id, remW) = val(id+1, remW)$
4- if $W[id] \leq remW$, we have two choices: ignore or take this item; we take the maximum:
$val(id, remW) = max(val(id+1, remW), V[id] + val(id+1, remW - W[id]))$

Coin Change 1- $change(0) = 0$. 2- $change(< 0) = \infty$
3- $change(value) = 1 + min(change(value - coinValue[i]))\forall i \in [0..n-1]$

Ways Coin Change 1- $ways(type, 0) = 1$, one way, use nothing
2- $ways(type, < 0) = 0$, no way, we cannot reach negative value
3- $ways(n, value) = 0$, no way, we have considered all coin types $\in [0..n-1]$
4- $ways(type, value) =$
$ways(type+1, value) + ways(type, value - coinValue[type])$, take or ignore this coin type

TSP State: tsp(currentPosition, visited), $\mathcal{O}\left(n^2 2^n\right)$
1- $tsp(pos, 2n-1) = dist[pos][0]$, return to starting city
2- $tsp(pos, mask) = min(dist[pos][nxt] + tsp(nxt, mask|(1 << nxt)))$,
$\forall nxt \in [0..n-1], nxt \neq pos$, and $(mask\&(1 << nxt))$ is '0' (turned off)

5.2 LIS

lis.cpp
Descripción: Devuelve una LIS de la secuencia dada en el vector `a`.
En concreto, devuelve la última que aparece.
`L[i]` es el valor más pequeño en el que acaban todas las LISes de longitud $i+1$.
Si quieres saber la longitud de la LIS en el intervalo $[0, i]$, lleva otro array, `lisl[]`, y en el bucle haz `lisl[i] = pos + 1`.
Si quieres saber la LDS puedes multiplicar el vector por -1 o leerlo del final hacia el principio.
Si quieres saber la longest non-decreasing subsequence, cambia `lower_bound` por `upper_bound` restando uno (o algo así).
Tiempo: $\mathcal{O}(n \log k)$, donde k es la longitud de una LIS.

```
vi lis(const vi &a) {
    vi L, L_id, P;
    L.assign(a.size(), 0);
    L_id.assign(a.size(), 0);
    P.assign(a.size(), -1);

    int lis_len = 0, lis_end = 0;
    for (int i = 0; i < a.size(); i++) {
        int pos = lower_bound(L.begin(), L.begin() + lis_len, a[i]) - L.begin();
        L[pos] = a[i];
        L_id[pos] = i;
        P[i] = pos ? L_id[pos-1] : -1;
        if (pos == lis_len) {
            lis_len++;
            lis_end = i;
        }
    }

    vi sol;
    stack<int> s;
    int x = lis_end;
    for (; P[x] >= 0; x = P[x]) s.push(a[x]);
    sol.push(a[x]);
    for (; !s.empty(); s.pop()) sol.push(s.top());

    return sol;
}
```

5.3 Kadane

kadane.cpp
Descripción: Halla la suma máxima de un intervalo en un array (1D) o de un subrectángulo en un rectángulo (2D). La solución naíf de hallar las sumas acumuladas ($\mathcal{O}\left(n^2\right)$) y luego mover cuatro índices ($\mathcal{O}\left(n^4\right)$) suele ser suficiente si $n \leq 100$.
Tiempo: 1D: $\mathcal{O}(n)$, 2D: $\mathcal{O}\left(n^3\right)$

```
int kadane1D(vi & arr, int& st, int& end, int n){
    int sum = 0, i, maxSum = -INF, ls = 0;
    end = -1; //Es simplemente un valor inicial
    for (i = 0; i < n; ++i){
        sum += arr[i];
        if (sum < 0)
            sum = 0, ls = i+1;
        else if (sum > maxSum)
            maxSum = sum, st = ls, end = i;
    }
    if (end != -1)  return maxSum; //Habia al menos un numero >=0
    maxSum = arr[0]; st=end=0;//else: todos <0, buscamos el maximo
    for (i = 1; i < n; i++)
        if (arr[i] > maxSum)
            maxSum = arr[i], st = end = i;
    return maxSum;
}

int fleft, fright, ftop, fbottom;//valores finales
int kadane2D(vvi m, int r, int c){
    int maxSum = -INF, left, right, i, sum, st, end;
    vi temp(r, 0);
    for (left = 0; left < c; ++left){
    fill (temp.begin(), temp.end(), 0);
        for (right = left; right < c; ++right){
            for (i = 0; i < r; ++i)
                temp[i] += m[i][right];
            sum = kadane1D(temp, st, end, r);
            if (sum > maxSum){
                maxSum = sum;
                fleft = left; fright = right;
                ftop = st; fbottom = end;
  }}}
  return maxSum;
}
```

Strings (6)

6.1 KMP

KMP.cpp
Descripción: Encuentra un patrón en un texto.
Uso: `p = patrón; t=texto; kmpPreprocess(); kmpSearch();`
Tiempo: $\mathcal{O}(n+m)$

```
string t, p; vi b;
int n, m; // n=t.size, m=p.size

void kmpPreprocess() {
  int i = 0, j = -1; b[0] = -1;
  while (i < m) {
    while (j >= 0 && p[i] != p[j]) j = b[j];
    b[++i]=++j;
  }
}
vi kmpSearch() {
    int i = 0, j = 0;
    vi sol;
  while (i<n) {
    while (j >= 0 && t[i] != p[j]) j = b[j];
    i++; j++;
    if (j==m)
      sol.push_back(i-j), j = b[j];
  }
  return sol;
}
```

6.2 Suffix Array

SuffixArray.cpp
Descripción: Construcción del Suffix Array y varias funcionalidades

```
#define _CRT_SECURE_NO_WARNINGS

#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;

typedef pair<int, int> ii;

#define MAX_N 100010 // O(n log n) -> up to 100 K
char T[MAX_N]; int n;  // input string and length
int RA[MAX_N], tempRA[MAX_N], SA[MAX_N], tempSA[MAX_N], c[MAX_N];

void countingSort(int k) { // O(n)
  int i, sum, maxi = max(300, n); // up to 255 ASCII chars or length of n
  memset(c, 0, sizeof c);
  for (i = 0; i < n; i++)
    c[i + k < n ? RA[i + k] : 0]++;
  for (i = sum = 0; i < maxi; i++) {
    int t = c[i]; c[i] = sum; sum += t;
  }
  for (i = 0; i < n; i++)
    tempSA[c[SA[i] + k < n ? RA[SA[i] + k] : 0]++] = SA[i];
  for (i = 0; i < n; i++)
    SA[i] = tempSA[i];
}

// Builds Suffix Array for string T. T should end with sth like $
void constructSA() { // O(n log n), can go up to 100000 characters
  int i, k, r;
  for (i = 0; i < n; i++) RA[i] = T[i];
  for (i = 0; i < n; i++) SA[i] = i;
  for (k = 1; k < n; k <<= 1) {
    countingSort(k);
    countingSort(0);
    tempRA[SA[0]] = r = 0;
    for (i = 1; i < n; i++)
      tempRA[SA[i]] =
      (RA[SA[i]] == RA[SA[i - 1]] && RA[SA[i] + k] == RA[SA[i - 1] + k]) ? r : ++r;
    for (i = 0; i < n; i++)
      RA[i] = tempRA[i];
    if (RA[SA[n - 1]] == n - 1) break; // optimization
  }
}

int Phi[MAX_N], PLCP[MAX_N], LCP[MAX_N];

// Longest Common Prefix: Para cada i, halla la longitud del prefijo mas largo
// que comparte con algun otro sufijo, y lo guarda en LCP[i]
// Para hacerlo en O(n), usa el orden inicial de los sufijos, no el del suffix array
    ,
// y de ahi surgen los arrays auxiliares Phi y PLCP
void computeLCP() { // O(n)
  int i, L;
  Phi[SA[0]] = -1;
  for (i = 1; i < n; i++)
    Phi[SA[i]] = SA[i - 1];
  for (i = L = 0; i < n; i++) {
    if (Phi[i] == -1) { PLCP[i] = 0; continue; }
    while (T[i + L] == T[Phi[i] + L]) L++;
    PLCP[i] = L;
    L = max(L - 1, 0);
  }
  for (i = 0; i < n; i++)
    LCP[i] = PLCP[SA[i]];
}

// Longest Repeated Substring: por la definicion de LCP, es el valor maximo de LCP
ii LRS() { // O(n), returns a pair (the LRS length and its index in the SA)
  int i, idx = 0, maxLCP = -1;
  for (i = 1; i < n; i++)
    if (LCP[i] > maxLCP)
      maxLCP = LCP[i], idx = i;
  return ii(maxLCP, idx);
}

char P[MAX_N]; /*pattern*/ int m; /*length of pattern*/
```

```
// Compara T a partir del indice id con P
int comp(int id) {
  for (int i = 0; i < m; ++i) {
    if (id + i >= n || T[id + i] < P[i]) return -1; // P mayor
    if (T[id + i] > P[i]) return 1; // P menor
  }
  return 0;
}

// Devuelve dos extremos (l, u): todos los S[i], i en [l, u]
// son indices donde aparece el patron P
ii stringMatching() { // O(m log n)
  int lo = 0, hi = n - 1, mid = lo;
  while (lo < hi) { // binary search lower bound
    mid = (lo + hi) / 2;
    int res = strncmp(T + SA[mid], P, m);
    // int res = comp(SA[mid]) if working with strings
    if (res >= 0) hi = mid;
    else          lo = mid + 1;
  }
  if (strncmp(T + SA[lo], P, m) != 0) return ii(-1, -1); // if not found
  ii ans; ans.first = lo;
  lo = 0; hi = n - 1; mid = lo;
  while (lo < hi) { // if lower bound is found, binary search upper bound
    mid = (lo + hi) / 2;
    int res = strncmp(T + SA[mid], P, m);
    // int res = comp(SA[mid]) if working with strings
    if (res > 0) hi = mid;
    else         lo = mid + 1;
  }
  if (strncmp(T + SA[hi], P, m) != 0) hi--;                  // special case
  ans.second = hi;
  return ans;
} // return (lowerbound, upperbound)

int owner(int idx) { return (idx < n - m - 1) ? 1 : 2; }

// Longest Common Substring (entre 2 strings): n = T1.size(), m = T2.size()
// Primero las concatenamos: T = T1$T2#. Construimos el suffix array de esta string.
// Llamamos al LCP, y despues a esto -> coste total O(n log n)
ii LCS() { // O(n), returns  (l: the LCS length, i: its index in the SA), i.e.,
    // solution = T[SA[i]] T[SA[i] + 1] ... T[SA[i] + l - 1]
  int i, idx = 0, maxLCP = -1;
  for (i = 1; i < n; i++)
    if (owner(SA[i]) != owner(SA[i - 1]) && LCP[i] > maxLCP)
      maxLCP = LCP[i], idx = i;
  return ii(maxLCP, idx);
}
```

6.3 Aho-Corasick

Aho-Corasick.cpp
Descripción: Para encontrar varios patrones en un mismo texto. También construye un autómata.
Uso: `searchWords(patterns, text)`
Tiempo: Lineal en el tamaño de la entrada

```
const int MAXC = 26; // Size of input alphabet
unordered_map<int, int> out, f; // i-th bit of out[j] on => i-th pattern found at
    index j
struct myhash {
  template <typename T, typename U>
  size_t operator()(const std::pair<T, U> &x) const {
    return hash<T>()(x.first) ^ hash<U>()(x.second); // xor, for instance
  } };
unordered_map<ii, int, myhash> g;

int buildMatchingMachine(vector<string> const & arr){//patterns
  int k = arr.size(), states = 1; // Only state 0
  for (int i = 0; i < k; ++i) { // Fill g, o , i.e., build trie
    const string &word = arr[i];
    int currentState = 0;
    for (int j = 0; j < word.size(); ++j) {
      int ch = word[j] - 'a';//Change if not only chars in [a, ,z]
      if (g.find(ii(currentState, ch)) == g.end())
        g[ii(currentState,ch)] = states++;
      currentState = g[ii(currentState, ch)];
    }
    if (out.find(currentState) == out.end()) out[currentState] = 0;
    out[currentState] |= (1 << i); // Add to output
  }
  queue<int> q; // We fill failure function with a BFS
  for (int ch = 0; ch < MAXC; ++ch) {
    if (g.find(ii(0, ch)) == g.end()) // Initial state: special
      g[ii(0, ch)] = 0;
    else { // depth = 1 => failure = 0
      f[g[ii(0, ch)]] = 0;
      q.push(g[ii(0, ch)]);
    }
  }
  while (q.size()) {
    int state = q.front(); q.pop();
    for (int ch = 0; ch <= MAXC; ++ch) {
      if (g.find(ii(state,ch)) != g.end()) {
        int failure = f[state];
        while (g.find(ii(failure,ch)) == g.end())
          failure = f[failure];
        failure = g[ii(failure, ch)];
        f[g[ii(state,ch)]] = failure;
        out[g[ii(state, ch)]] |= out[failure]; // Merge outputs
        q.push(g[ii(state, ch)]);
      }
    }
  }
  return states; // Number of states in the automaton
}

int findNextState(int currentState, char nextInput) {
  int answer = currentState;
  int ch = nextInput - 'a';
  while (g.find(ii(answer, ch)) == g.end())
    answer = f[answer];
  return g[ii(answer,ch)];
}
```

```
// Find patterns (arr) in a line (text)
void searchWords(vector<string> arr, string text) {
  buildMatchingMachine(arr); // Preprocess
  int currentState = 0;
  for (int i = 0; i < text.size(); ++i) {
    currentState = findNextState(currentState, text[i]);
    if (out[currentState] != 0) // Match found
      for (int j = 0; j < arr.size(); ++j)
        if (out[currentState] & (1 << j)) {
          //Word arr[j] appears from i - arr[j].size() + 1 to i
        }
  }
}
```

6.4 Manacher

Manacher.cpp
Descripción: Procesa un string, encontrando toda la información sobre los palíndromos que tiene dentro. Si alrededor de la posición i hay un pal. de longitud impar n: p[1][i] = (n-1)/2. Si alrededor del hueco entre las posiciones i, i+1 hay un palíndromo de longitud par n: p[0][i] = n.
Tiempo: $\mathcal{O}(n)$

```
void manacher(const string& s) {
  int n = s.size();
  vi p[2] = {vi(n+1), vi(n)};
  for(int z=0; z<2; ++z)
    for (int i=0,l=0,r=0; i < n; i++) {
      int t = r-i+!z;
      if (i<r) p[z][i] = min(t, p[z][l+t]);
      int L = i-p[z][i], R = i+p[z][i]-!z;
      while (L>=1 && R+1<n && s[L-1] == s[R+1])
        p[z][i]++, L--, R++;
      if (R>r) l=L, r=R;
    }
}
```

Bits (7)

bits.cpp
Descripción: Operaciones básicas con bitsets y bitmasks.

```
#include <bitset>

bs.set(/*pos, val = true*/);  // Set bit at pos to val. Si no pones pos ni val se
    pone todo a 1.
bs.reset(/*pos*/);            // Reset bit at pos to 0 .
bs.all();                   // Return whether all bits are set.
bs.test(pos);                 // Return value of bit at pos.
bs.to_ullong();               // Return unsigned long long.

// Mascaras de bits
#define isOn(S, j) (S & (1 << j))
#define setBit(S, j) (S |= (1 << j))
#define clearBit(S, j) (S &= ~(1 << j))
#define toggleBit(S, j) (S ^= (1 << j))
#define lowOne(S) (S & (-S))
#define lowZero(S) ~S & (S+1)
#define setLowZero(S) S | (S+1)
#define setAll(S, n) (S = (1 << n) - 1)

// Mascaras de 64 bits
// Hay que poner LL.   ej. (1LL << n)
S &= ~(1LL << k) // Clear bit k version 64bits
S & (1LL << k)) // Check if bit k is on version 64bits

// Una mascara con los N bits menos significativos a 1
~(~(0LL) << N)

// Tu error esta aqui
if (x & (1 << i) == 0) { } // esta mal, tiene precedencia ==
(1ULL << n) // shiftea 1 n veces (si no pones ULL no puedes shiftearlo mas de 32
    bits y te dara cero)

// Devuelve n con su msb puesto a cero. n es un entero de 32 bits sin signo.
uint clear_msb(uint n) {
    uint mask = n;
    mask |= mask >> 1;
    mask |= mask >> 2;
    mask |= mask >> 4;
    mask |= mask >> 8;
    mask |= mask >> 16;
    mask = mask >> 1;

    return n & mask;
}

// Funciones guay en g++.

// Returns the number of leading 0-bits in x, starting at the most significant bit
    position. If x is 0, the result is undefined (el MSB lo puedes sacar haciendo
    32 - __builtin_clz(x), si x != 0).
int __builtin_clz (unsigned int x);

// Returns the number of trailing 0-bits in x, starting at the least significant bit
     position. If x is 0, the result is undefined.
int __builtin_ctz (unsigned int x);

// Devuelve el numero de bits a uno en un long long.
__builtin_popcountll(n)

// Al realizar la operacion complemento a 2 con el numero i obtienes -i
i & (-i) // Te da todo ceros, salvo un uno en el bit a 1 menos significativo de i
```

Misc (8)

cositas.cpp
Descripción: Cositas varias de C++

```
// Strings
```

```
string firstletter(n, str[0]); // firstletter es n copias del char str[0]
    concatenadas.
s.insert(0, "012"); // Inserta en s en la posicion 0 el string "012".

// Convertir de enteros <-> string
int i = stoi("1893"); // 1893
string s = i.to_string(); // "1893"

// Ejemplos printf y similares
printf("%.5f", d); // Imprimir un double con 5 digitos tras el punto decimal. Usa
    %.5Lf para long double.
printf("%s\n", s.c_str()); // Imprimir un string de C++.
printf("%lld\n", unlonglong); // %lld para long long
printf("%0.5d / %0.5d\n", a, b); // Escribe los ints a, b con cinco cifras, paddeado
     con ceros a la izquierda si es necesario.
printf("%5d", x); // Imprimir un int con anchura 5. Sera paddeado por la izquierda
    con espacios.
sscanf(s.c_str(), "%s %s", &s, &t); // Parsea el string s en "%s %t"

// Imprimir un string alineado a la izquierda con anchura 21. El string sera
    paddeado por la derecha con espacios
// si su longitud es menor que 21. Por ejemplo, si p[i] = "San Bernardino",
    imprimira
// 'San Bernardino       '
printf("%-21s", p[i].c_str());

// Estructuras de datos
priority_queue<di, vector<di>, greater<di>> pq; // min priority queue de (double,
    int)

// Funciones guays de la STL
auto ai = find(perms.begin(), perms.end(), a); // Busca en perms el elemento a y
    devuelve un iterador apuntando al elemento, o el iterador perms.end() si no lo
    encuentra.
transform(t.begin(), t.end(), t.begin(), ::tolower); // Convierte el string t a
    lower.
bills.erase(--bills.end()); // Borrar el ultimo elemento de un contenedor como set,
    multiset, map etc.
value.erase(value.begin(), std::find_if(value.begin(), value.end(), std::bind1st(std
    ::not_equal_to<char>(), ' '))); // Removes leading spaces from value.
// Elimina todos los elementos salvo el primero de cada grupo de elementos iguales
    en el rango.
// Por ejemplo: 10 20 20 20 30 30 20 20 10 ==> 10 20 30 20 10 ?  ?  ?  ?
// Devuelve un iterador al elemento que sigue al ultimo elemento no eliminado, en el
     ejemplo apuntaria a la primera interrogacion.
auto it = unique (myvector.begin(), myvector.end());
// lower_bound(first, last, val) ==> Returns an iterator pointing to the first
    element in the range [first,last) which does not compare less than val.
// upper_bound(first, last, val) ==> Returns an iterator pointing to the first
    element in the range [first,last) which compares greater than val.
// Usa distance(v.begin(), it) para convertir un iterador a una posicion dentro del
    contenedor

// Si los casos de prueba son lineas de texto y antes te viene el numero de casos de
     prueba T, asegurate de hacer cin.ignore() para consumir hasta el proximo \n
// (vease UVa 10374 para un ejemplo en el que cin.get() no basta porque meten
    espacios tras T).
int T;
cin >> T; cin.ignore();
while (T--) {
    string s;
    getline(cin, s);
}

// Leer de un stringstream splitteando por comas
stringstream st(t);
string token;
while (getline(st, token, ',')) {
    // hacer cosas con token
}

// Custom sorting comparator de un tipo. Usa tie() para structs que en realidad son
    tuplas.
// Pon const antes de { si estas definiendo operator < en un struct.
struct comp {
    bool operator() (const ii &x, const ii &y) {
        return x.first * y.second < x.second * y.first;
    }
};
sort(scores.begin(), scores.end(), cmp()); // Y asi se usa.
set<ii, comp> Fn; // Y asi lo puedes usar en un contenedor ordenado.

// Reverse un string en java.
new StringBuilder(hi).reverse().toString()

double DOUBLE_MAX = numeric_limits<double>::max(); // necesita #include <limits>
```

8.1 Calendario

calendar.cpp

Descripción: Funciones para trabajar con fechas. Los meses se expresan como enteros del 1 al 12, los días del 1 al 31 y los años del 0 al 9999.

```
int dateToInt(int d, int m, int y) {
    return
            1461*(y+4800+(m-14)/12)/4+
            367*(m-2-(m-14)/12*12)/12-
            3*((y+4900+(m-14)/12)/100)/4+
            d-32075;
}

void intToDate(int jd, int &d, int &m, int &y) {
    int x, n, i, j;
    x = jd+68569; n = 4*x/146097;
    x -= (146097*n+3)/4;
    i = (4000*(x+1))/1461001;
    x -= 1461*i/4-31; j = 80*x/2447;
    d = x-2447*j/80;
    x = j/11; m = j+2-12*x;
    y = 100*(n-49)+i+x;
}
```

8.2 Greedy

intervalCovering.cpp
Descripción: Esquema de solución para resolver el interval covering problem: dada una colección de intervalos, recubrir el intervalo $[0, M]$ con una cantidad mínima. Solución: ordenar por increasing left endpoint and, if ties arise, decreasing right endpoint (función `cmp()`); tomar el intervalo que cubre más a la derecha que solape con el anterior que has cogido.
Tiempo: $\mathcal{O}(n)$

```
sort(v.begin(), v.end(), cmp());

int i = 0;
while (i < v.size() && v[i].second < 0) i++;

vii sol;
ii p = ii(0, 0);
while (i < v.size() && p.second < M) {
    ii q = v[i];

    if (q.first > p.second)
        break;

    while (i < v.size() && v[i].first <= p.second) {
        if (v[i].second > q.second)
            q = v[i];

        i++;
    }

    sol.push_back(q);
    p = q;
 }

if (sol.size() > 0 && sol[sol.size() - 1].second >= M) { // OK
 } else { // No se puede recubrir el intervalo [0, M]
```

8.3 2SAT

2SAT.cpp
Descripción: Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type $(a \vee b) \wedge (\neg a \vee c) \wedge (d \vee \neg b) \wedge \ldots$ becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit inversions (~x).
Uso:
```
TwoSat ts(number of boolean variables);
ts.either(0,  3); Var 0 is true or var 3 is false
ts.set value(2);, Var 2 is true
ts.at most one(0, 1,2); <= 1 of vars 0,  1 and 2 are true
ts.solve(); Returns true iff it is solvable
ts.values(0..N-1) holds the assigned values to the vars
```
Tiempo: $\mathcal{O}(N + E)$, where N is the number of boolean variables, and E is the number of clauses.

```
#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define trav(a, x) for(auto& a : x)

struct TwoSat {
  int N;
  vector<vi> gr;
  vi values; // 0 = false , 1 = true
  TwoSat(int n = 0) : N(n), gr(2*n) {}
  int add_var() { // (optional )
    gr.emplace_back();
    gr.emplace_back();
    return N++;
  }
  void either(int f, int j) {
    f = (f >= 0 ? 2*f : -1-2*f);
    j = (j >= 0 ? 2*j : -1-2*j);
    gr[f^1].push_back(j);
    gr[j^1].push_back(f);
  }
  void set_value(int x) { either(x, x); }
  void at_most_one(const vi& li) { // (optional )
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i,2,li.size()) {
      int next = add_var();
      either(cur, ~li[i]);
      either(cur, next);
      either(~li[i], next);
      cur = ~next;
    }
    either(cur, ~li[1]);
  }
  vi val, comp, z; int time = 0;
  int dfs(int i) {
    int low = val[i] = ++time, x; z.push_back(i);
    trav(e, gr[i]) if (!comp[e])
      low = min(low, val[e] ?: dfs(e));
    ++time;
    if (low == val[i]) do {
      x = z.back(); z.pop_back();
      comp[x] = time;
      if (values[x>>1] == -1)
        values[x>>1] = !(x&1);
    } while (x != i);
    return val[i] = low;
  }
  bool solve() {
    values.assign(N, -1);
    val.assign(2*N, 0); comp = val;
    rep(i,0,2*N) if (!comp[i]) dfs(i);
    rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
    return 1;
  }
};
```

Java (9)

Main.java

Descripción: Un template para problemas en Java si usas `BigInteger` o `BigDecimal`. `modPow()` is a godsend.

```
import java.math.BigInteger;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        while (sc.hasNextInt()) {
            int b1 = sc.nextInt();
            int b2 = sc.nextInt();
            String s = sc.next();
            try {
                BigInteger x = new BigInteger(s, b1);
                System.out.println(s + " base " + b1 + " = " + x.toString(b2).
                    toUpperCase() + " base " + b2);
            } catch (NumberFormatException e) {
                System.out.println(s + " is an illegal base " + b1 + " number");
            }
        }
    }
}
```

Anexo (10)

10.0.1 Tablas

Primos hasta potencias de 10
$prim(10^1) = 4$
$prim(10^2) = 25$
$prim(10^3) = 168$
$prim(10^4) = 1.229$
$prim(10^5) = 9.592$
$prim(10^6) = 78.498$
$prim(10^7) = 664.579$
$prim(10^8) = 5.761.455$
$prim(10^9) = 50.847.534$

Mayores primos hasta 10^k: $10^1 - 3$, $10^2 - 3$, $10^3 - 3$, $10^4 - 27$, $10^5 - 9$, $10^6 - 17$, $10^7 - 9$, $10^8 - 11$, $10^9 - 63$, $10^{10} - 33$, $10^{11} - 23$, $10^{12} - 11$, $10^{13} - 29$, $10^{14} - 27$, $10^{15} - 11$.

10.0.2 Tu error está aquí

- ¿Inicializas todas las variables al principio de cada caso de prueba?
- Fíjate en el formato de la salida. Además, ¿no te falta/sobra un `endl`?
- ¿Seguro que no necesitas `long long int`?
- ¿Seguro que no tienes una variable local y otra global con el mismo nombre?
- `if (a = b)`