Notas para el ACM ICPC

Octavio Alberto Agustín Aquino Ricardo Omar Chávez García Fernando Said Ramírez García

17 de octubre de 2005

Índice

1. Trigonometría

Todo triángulo tiene un círculo inscrito tangente a sus lados e interior a él, cuyo centro es el punto de intersección de las bisectrices de los lados.

Todo triángulo tiene un círculo circunscrito que pasa por sus vértices. El punto de intersección de las medianas del triángulo es su centro de masa.

Sea el triángulo $\triangle ABC$ con ángulos A, B y C y sean a, b y c los lados opuestos a dichos ángulos, respectivamente. Sean h_c, t_c y m_c las longitudes de la altura, la bisectriz y la mediana que se originan en el vértice C, y sean r y R los sendos radios de los círculos inscrito y

circunscrito. Hagamos $s = \frac{1}{2}(a+b+c)$. Se satisfacen las siguientes relaciones.

$$
\begin{aligned}
c^2 &= a^2 + b^2 - 2ab\cos C && (1)\\
a &= b\cos C + c\cos B && (2)\\
\tfrac{a}{\operatorname{sen} A} &= \tfrac{b}{\operatorname{sen} B} = \tfrac{c}{\operatorname{sen} C} && (3)\\
area &= \tfrac{1}{2}ch_c = \tfrac{1}{2}ab\operatorname{sen} C && (4)\\
&= \tfrac{c^2 \operatorname{sen} A \operatorname{sen} B}{2\operatorname{sen} C} = rs = \tfrac{abc}{4R} && (5)\\
&= \sqrt{s(s-a)(s-b)(s-c)} && (6)\\
r &= c\operatorname{sen}\tfrac{1}{2}A \operatorname{sen}\tfrac{1}{2}B \sec\tfrac{1}{2}C && (7)\\
&= \tfrac{ab\operatorname{sen} C}{2s} = (s-c)\tan\tfrac{1}{2}C && (8)\\
&= \left(\tfrac{1}{h_a} + \tfrac{1}{h_b} + \tfrac{1}{h_c}\right)^{-1} && (9)\\
R &= \tfrac{c}{2\operatorname{sen} C} = \tfrac{abc}{4\,area} && (10)\\
h_c &= a\operatorname{sen} B = b\operatorname{sen} A = \tfrac{2\,area}{c} && (11)\\
t_c &= \tfrac{2ab}{a+b}\cos\tfrac{1}{2}C = \sqrt{ab\left(1 - \tfrac{c^2}{(a+b)^2}\right)} && (12)\\
m_c &= \sqrt{\tfrac{1}{2}a^2 + \tfrac{1}{2}b^2 - \tfrac{1}{2}c^2} && (13)\\
\cos(x \pm y) &= \cos(x)\cos(y) \mp \operatorname{sen}(x)\operatorname{sen}(y) && (14)\\
\operatorname{sen}(x \pm y) &= \cos(x)\operatorname{sen}(y) \pm \operatorname{sen}(x)\cos(y) && (15)\\
\tan 2x &= \tfrac{2\tan x}{1-\tan^2 x} && (16)\\
\operatorname{sen}\tfrac{x}{2} &= \pm\sqrt{\tfrac{1-\cos x}{2}} && (17)\\
\cos\tfrac{x}{2} &= \pm\sqrt{\tfrac{1+\cos x}{2}} && (18)\\
\tan\tfrac{x}{2} &= \pm\sqrt{\tfrac{1-\cos x}{1+\cos x}} = \tfrac{\operatorname{sen} x}{1+\cos x} = \tfrac{1-\cos x}{\operatorname{sen} x} && (19)
\end{aligned}
$$

2. Geometría analítica

2.1. Rectas, planos y círculos

Escribimos $[p_1, p_2, p_3] = \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$ donde $p_i = (x_i, y_i, z_i)$. El área signada de un triángulo con vértices en $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ y $p_3 = (x_3, y_3)$ es

$$K(\{p_i\}_{i=1}^3) = \frac{[q_1, q_2, q_3]}{2} = \frac{1}{2}\begin{vmatrix} x_2 - x_1 & y_2 - y_1 \\ x_3 - x_1 & y_3 - y_1 \end{vmatrix} \quad (20)$$

donde $q_i = (x_i, y_i, 1)$. La función K indica si el punto p_3 se encuentra a la izquierda ($K < 0$), a la derecha ($K > 0$) o es colineal ($K = 0$) con respecto al segmento dirigido $\overrightarrow{p_1p_2}$. Por lo tanto, la ecuación de la recta $Ax + By + C = 0$ que pasa por los puntos (x_1, y_1) y (x_2, y_2) puede escribirse de la siguiente manera:

$$\begin{vmatrix} x & y & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} = 0.$$

Cuadro 1: Ecuaciones de rectas

Condiciones	Ecuación
$\parallel B$, pasa por A	$(r - A) \times B = 0$
Pasa por A y B	$(r - A) \times (B - A) = 0$
$\perp B$, pasa por A	$(r - A) \cdot B = 0$

La distancia d de una recta $Ax + By + C = 0$ a un punto dado (x_1, y_1) es $d = \frac{|Ax_1 + By_1 + C|}{\sqrt{A^2+B^2}}$.

En general, el área de un polígono $\{p_i\}_{i=0}^{n-1}$ es

$$K(\{p_i\}_{i=0}^{n-1}) = \frac{1}{2}\sum_{i=0}^{n-1}\begin{vmatrix} x_i & y_i \\ x_{i+1} & y_{i+1} \end{vmatrix}$$

donde los índices se toman módulo el número de vértices n del polígono.

El siguiente determinante indica si un punto $p_4 = (x_4, y_4)$ está sobre ($D = 0$), dentro ($D < 0$) o fuera ($D > 0$) del círculo determinado por los puntos $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ y $p_3 = (x_3, y_3)$.

$$
\begin{aligned}
D &= \begin{vmatrix} x_1 & y_1 & x_1^2 + y_1^2 & 1 \\ x_2 & y_2 & x_2^2 + y_2^2 & 1 \\ x_3 & y_3 & x_3^2 + y_3^2 & 1 \\ x_4 & y_4 & x_4^2 + y_4^2 & 1 \end{vmatrix} \\
&= \begin{vmatrix} x_1 - x_4 & y_1 - y_4 & (x_1 - x_4)^2 + (y_1 - y_4)^2 \\ x_2 - x_4 & y_2 - y_4 & (x_2 - x_4)^2 + (y_2 - y_4)^2 \\ x_3 - x_4 & y_3 - y_4 & (x_3 - x_4)^2 + (y_3 - y_4)^2 \end{vmatrix}.
\end{aligned} \quad (21)
$$

2.2. Transformación de coordenadas

Si los ejes coordenados giran un ángulo θ en torno al origen en sentido horario, la relación entre el sistema original y el nuevo es

$$\begin{pmatrix} \xi \\ \eta \end{pmatrix} = \begin{pmatrix} \cos\theta & -\operatorname{sen}\theta \\ \operatorname{sen}\theta & \cos\theta \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}. \quad (22)$$

2.3. Cuadráticas

El polinomio de segundo grado

$$Q(x, y) = Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0,$$

en donde $B \neq 0$, puede transformarse en otro de la forma

$$Q'(\xi, \eta) = A'\xi^2 + C'\eta^2 + D'\xi + E'\eta + F' = 0$$

haciendo el cambio de variable (22) con parámetro

$$\theta = \begin{cases} \frac{1}{2}\arctan\frac{B}{A-C} & \text{si } A \neq C, \\ \frac{\pi}{4} & \text{si } A = C. \end{cases} \quad (23)$$

La forma polar de una cónica de excentricidad e, cuyo foco está en el polo y a p unidades de la directriz, es

$$r = \frac{ep}{1 + e\cos(\theta + \phi)},$$

donde ϕ es el ángulo que forman el eje polar y la línea que une el foco y el polo, tomado en sentido horario. Si $e = 1$, tenemos una párabola; si $e < 1$, una elipse, y si $e > 1$, una hipérbola.

3. Geometría computacional

Teorema 3.1 (Pick). *Sea P un polígono reticular, $I(P)$ el número de puntos reticulares interiores a él y $F(P)$ los puntos reticulares en su frontera. Entonces*

$$K(P) = I(P) + \frac{F(P)}{2} - 1.$$

3.1. Triangulación de polígonos

Una *oreja* de un polígono P son tres vértices consecutivos que determinan un triángulo completamente contenido en P. Todo polígono tiene al menos una oreja.

Algoritmo 3.1. Determina si tres puntos de un polígono son una oreja.

Entrada: Tres puntos p_i,p_j,p_k en un polígono P ordenado en sentido antihorario.
Salida: Verdadero, si los puntos conforman una oreja. Falso en otro caso.

función OREJA (p_i, p_j, p_k, P)
si $K(p_i, p_j, p_k) < 0$ **entonces**
  devolver falso.
para todo $m \neq i, j, k$ **hacer**
  si $p_m \in \triangle p_i p_j p_k$ **entonces**
    devolver falso.
devolver verdadero.

Algoritmo 3.2 (Otectomía). Triangula un polígono P de n vértices en tiempo $O(n^2)$.

Entrada: Un polígono $P = \{p_i\}_{i=0}^{n-1}$
Salida: Una triangulación T.

$T \leftarrow \emptyset$.
para todo $p_i \in P$ **hacer**
  $l_i \leftarrow (i-1) \mod n, r_i \leftarrow (i+1) \mod n$.
$i \leftarrow n - 1$.
mientras $|T| < n - 2$ **hacer**
  $i \leftarrow r_i$
  si OREJA(l_i, i, r_i, P) **entonces**
    $T \leftarrow T \cup \{p_{l_i}, p_i, p_{r_i}\}$.
    $l_{r_i} \leftarrow l_i, r_{l_i} \leftarrow r_i$.

3.2. Intersección de segmentos

Algoritmo 3.3. Determina si dos segmentos $\overrightarrow{p_0p_1}$ y $\overrightarrow{q_0q_1}$ se intersectan usando (20).

Entrada: Dos segmentos $\overrightarrow{p_0p_1}$ y $\overrightarrow{q_0q_1}$.
Salida: Verdadero si los segmentos se intersectan. Falso en caso contrario.

$d_1 \leftarrow \operatorname{sgn} K(q_0, q_1, p_0), d_2 \leftarrow \operatorname{sgn} K(q_0, q_1, p_1)$.
$d_3 \leftarrow \operatorname{sgn} K(p_0, p_1, q_0), d_4 \leftarrow \operatorname{sgn} K(p_0, p_1, q_1)$. {$\operatorname{sgn}(x) = [x > 0] - [x < 0]$.}
si $(d_1 \cdot d_2 < 0) \vee (d_3 \cdot d_4 < 0)$ **entonces**
  devolver verdadero.
si no
  $u \leftarrow \min(p_{0_x}, p_{1_x}), U \leftarrow \max(p_{0_x}, p_{1_x})$.
  $v \leftarrow \min(p_{0_y}, p_{1_y}), V \leftarrow \max(p_{0_y}, p_{1_y})$.
  $s \leftarrow \min(q_{0_x}, q_{1_x}), S \leftarrow \max(q_{0_x}, q_{1_x})$.
  $t \leftarrow \min(q_{0_y}, q_{1_y}), T \leftarrow \max(q_{0_y}, q_{1_y})$.
  si $(d_1 = 0) \wedge (p_0 \in (s, S) \times (t, T))$ **entonces**
    devolver verdadero.
  si no si $(d_2 = 0) \wedge (p_1 \in (s, S) \times (t, T))$ **entonces**
    devolver verdadero.
  si no si $(d_3 = 0) \wedge (q_0 \in (u, U) \times (v, V))$ **entonces**
    devolver verdadero.
  si no si $(d_4 = 0) \wedge (q_1 \in (u, U) \times (v, V))$ **entonces**
    devolver verdadero.
devolver falso.

Sean s_1 y s_2 dos segmentos. Decimos que son *comparables* si la línea de barrido los intersecta, y que s_1 está por arriba de s_2 si su ordenada de intersección es la mayor de ambas.

Algoritmo 3.4. Determina si un conjunto de segmentos se intersecta en tiempo $O(n \log n)$.

Entrada: Un conjunto de segmentos S.
Salida: Verdadero si hay intersecciones en S. Falso en caso contrario.

$T \leftarrow \emptyset$.
Ordenar los segmentos de S de izquierda a derecha, de extremo izquierdo a derecho y de menor a mayor ordenada.
para todo $p \in P$ **hacer**
  si p es extremo izquierdo de un segmento s **entonces**
    INSERTAR(T, s).
    si ARRIBA$(T, s) \cap s \neq \emptyset$ o ABAJO$(T, s) \cap s \neq \emptyset$ **entonces**
      devolver verdadero.
  si p es extremo derecho de un segmento s **entonces**
    si ARRIBA$(T, s) \cap$ ABAJO$(T, s) \neq \emptyset$ **entonces**
      devolver verdadero.
    SACAR(T, s).
devolver falso.

3.3. Envolvente convexa

Algoritmo 3.5 (Graham). Calcula la envolvente convexa de un conjunto de puntos Q.

Entrada: Un conjunto de puntos $Q = \{p_i\}_{i=0}^{m} \subset \mathbb{R}^2$.

Salida: El conjunto de vértices de la envolvente convexa en la pila S.

1: Sea $p_0 \in Q$ el punto de menores coordenadas.
2: Sean $p_1 \leq \ldots \leq p_m$ los puntos restantes de Q ordenados por ángulo polar alrededor de p_0. Si hay empates, quitar todos los de mismo ángulo salvo el de mayor módulo.
3: Meter p_0, p_1, p_2 en la pila S.
4: **para** $i \leftarrow 3$ hasta m **hacer**
5: **mientras** $K(p_{tope(S)-1}, p_{tope(S)}, p_i) \leq 0$ **hacer**
6: SACAR(S).
7: METER(S, p_i).

3.4. Punto en polígono

Las operaciones en los índices del polígono son módulo el número de vértices del mismo.

Algoritmo 3.6. Determina si un punto está en el interior de un polígono simple. Identificamos al punto (x, y) con $(x, y, 1)$ y ∞ con $(1, 0, 0)$.

Entrada: Un punto $p = (x_0, y_0, 1)$ y un polígono P.

Salida: Verdadero si el punto está en el polígono. Falso en caso contrario.

1: $a \leftarrow$ falso.
2: **para todo** $\overrightarrow{p_i p_{i+1}} \in P$ **hacer**
3: **si** $([p, \infty, p_i] < 0) \neq ([p, \infty, p_{i+1}] < 0)$ **entonces**
4: **si** $([p_i, p_{i+1}, p] < 0) \neq ([p_i, p_{i+1}, \infty] < 0)$ **entonces**
5: $a \leftarrow \neg a$.
6: **si no si** $[p_i, p_{i+1}, p] = 0 \wedge x_i \leq x \leq x_{i+1}$ **entonces**
7: **devolver** verdadero.
8: **devolver** a.

Algoritmo 3.7. Determina si un punto está en el interior de un polígono.

Entrada: Un punto $p = (x_0, y_0)$ y un polígono P.

Salida: El índice g del polígono alrededor de p. Si $g = 0$, p está fuera del polígono.

1: $g \leftarrow 0$.
2: **para todo** $\overrightarrow{p_i p_{i+1}} \in P$ **hacer**
3: **si** $y_i \leq y_0$ **entonces**
4: **si** $y_{i+1} > y_0$ **entonces**
5: **si** $K(p_i, p_{i+1}, p_0) < 0$ **entonces**
6: $g \leftarrow g + 1$.
7: **si no**
8: **si** $y_{i+1} \leq y_0$ **entonces**
9: **si** $K(p_i, p_{i+1}, p_0) > 0$ **entonces**
10: $g \leftarrow g - 1$.
11: **devolver** g.

3.5. Mínimo círculo encapsulador

Algoritmo 3.8. Para hallar el centro y el radio del mínimo círculo encapsulador, invocamos a MINCIRC($P, \emptyset$).

Entrada: Un conjunto de puntos $P = \{p_i\}_{i=1}^{n}$.

Salida: El centro c y radio r del mínimo círculo encapsulador.

1: **función** MINCIRC(P, B)
2: **si** $|B| = 1$ **entonces**
3: **devolver** $c \leftarrow b_1$ y $r \leftarrow 0$.
4: **si no si** $|B| = 2$ **entonces**
5: **devolver** $c \leftarrow \frac{1}{2}(b_1 + b_2)$ y $r \leftarrow \frac{1}{2}d(b_1, b_2)$.
6: **si no si** $|B| = 3$ **entonces**
7: Hacer c el centro del círculo circunscrito del triángulo $\{b_1, b_2, b_3\}$
8: **devolver** c y $r \leftarrow d(c, b_1)$.
9: $c \leftarrow (\infty, \infty), r \leftarrow 0$.
10: **para todo** $p_i \in P$ **hacer**
11: **si** $d(p_i, c) > r$ **entonces**
12: $B \leftarrow B \cup \{p_i\}$.
13: MINCIRC(P, B).
14: **fin funcion**

4. Teoría de números

4.1. Factoriales

La mayor potencia de un primo p que divide a $n!$ es $\epsilon_p(n) = \sum_{k=1}^{\lfloor \log_p n \rfloor} \left\lfloor \frac{n}{p^k} \right\rfloor$. En consecuencia, el número Z de ceros al final de $n!$ es $Z = \sum_{k=1}^{\lfloor \log_5 n \rfloor} \left\lfloor \frac{n}{5^k} \right\rfloor$.

Teorema 4.1 (Wilson). *Se satisface* $(p-1)! \equiv -1 \pmod{p}$ *si, y sólo si, p es primo.*

4.2. Ternas pitagóricas

Las ternas pitagóricas son tripletes $(u, v, w) \in \mathbb{N}^3$ tales que $w^2 = u^2 + v^2$. Si $u \perp v \perp w$, entonces la tripleta se dice *primitiva*. Toda terna primitiva es de la forma $(u, v, w) = (x^2 - y^2, 2xy, x^2 + y^2)$, donde $y < x$ y $x \perp y$. Aunque hay una infinidad de ternas que tienen esta forma, no todas son primitivas.

4.3. Algoritmo de Euclides

Algoritmo 4.1 (Euclides). Calcula el máximo común divisor de a y b.

Entrada: $a, b \in \mathbb{Z}$.

Salida: El máximo común divisor de a y b.

1: **mientras** $a \neq 0$ y $b \neq 0$ **hacer**
2: **si** $|a| > |b|$ **entonces**
3: $a \leftarrow a \mod b$.
4: **si no**
5: $b \leftarrow b \mod a$.
6: **si** $a \neq 0$ **entonces**
7: **devolver** a.
8: **si no**
9: **devolver** b.

Algoritmo 4.2 (Blankinship). Calcula el m. c. d. (a, b) de a y b y la combinación lineal $ar + bs = (a, b)$.

Entrada: $a, b \in \mathbb{Z}$.
Salida: El m. c. d. de a y b y r y s tales que $ar + bs = (a, b)$.

1: **función** MCD(a,b)
2: $u \leftarrow 1, d \leftarrow a$.
3: **si** $b = 0$ **entonces**
4: **devolver** $(d, u, 0)$.
5: **si no**
6: $v_1 \leftarrow 0, v_3 \leftarrow b$.
7: **mientras** $v_3 \neq 0$ **hacer**
8: $q \leftarrow \lfloor d/v_3 \rfloor, t_3 \leftarrow d \mod v_3$.
9: $t_1 \leftarrow u - qv_1, u \leftarrow v_1, d \leftarrow v_3, v_1 \leftarrow t_1, v_3 \leftarrow t_3$.
10: **devolver** $(d, u, (d - au)/b)$.

Con el algoritmo anterior puede resolverse una congruencia de la forma $ax \equiv b \mod m$.

Algoritmo 4.3. Calcula una solución de $ax = b \mod m$.

Entrada: $a, b, m \in \mathbb{Z}$.
Salida: Si existe, x tal que $ax = b \mod m$. Si no, falso.

1: $(d, x', y') \leftarrow \text{MCD}(a, m)$.
2: **si** $d \backslash b$ **entonces**
3: $x \leftarrow x'(b/d)$
4: **para** $i = 0$ hasta $d - 1$ **hacer**
5: **devolver** $(x + i\frac{m}{d}) \mod m$
6: **si no**
7: **devolver** falso.

Obsérvese que podemos resolver la ecuación diofántica $ax + my = b$, obteniendo x con el algoritmo anterior y $y = \frac{b-ax}{d}$, siempre que $(a, m) \backslash b$.

4.3.1. Ciclos en sucesiones

Algoritmo 4.4 (Floyd). Encuentra un ciclo en una sucesión definida a través de $a_{i+1} = f(a_i)$, donde $f : S \to S$ y S es finito.

Entrada: Una sucesión finita $\{a_i\}_{i=1}^{\infty}$.
Salida: La longitud t del ciclo de a.

1: $\alpha \leftarrow a_1, \beta \leftarrow a_1, t \leftarrow 0$.
2: **repetir**
3: $\alpha \leftarrow f(\alpha), \beta \leftarrow f(f(\beta)), t \leftarrow t + 1$.
4: **hasta que** $\alpha = \beta$

4.4. Divisores de un número

Teorema 4.2. *Sea $n \in \mathbb{N}$ y sea*

$$n = \prod_{i=1}^{s} p_i^{\alpha_i} \tag{24}$$

su descomposición en números primos. Entonces n tiene $N = \prod_{i=1}^{s} (\alpha_i + 1)$ divisores.

El menor divisor propio p de un número n es primo.

Algoritmo 4.5 (División exhaustiva). El siguiente algoritmo halla el menor divisor de un número entero.

Entrada: $n \in \mathbb{N}$.
Salida: El menor divisor d de n, o 1 si $n = 1$.

1: **si** $n \equiv 0 \pmod 2$ **entonces**
2: **devolver** 2.
3: **si** $n \equiv 0 \pmod 3$ **entonces**
4: **devolver** 3.
5: $\Delta_0 \leftarrow 2, \Delta_1 \leftarrow 4, d \leftarrow 5, i \leftarrow 0, q \leftarrow 25$.
6: **mientras** $n \mod d > 0 \wedge q \leq n$ **hacer**
7: $q \leftarrow q + 2\Delta_i d + 4 + 12i, d \leftarrow d + \Delta_i, i \leftarrow (i + 1) \mod 2$.
8: **si** $q > n$ **entonces**
9: **devolver** n.
10: **devolver** d.

Algoritmo 4.6 (Suma de divisores). Para obtener la suma de los divisores propios de un número, aprovechamos la identidad

$$\sigma(n) = \sum_{\substack{d \backslash n \\ 1<d<n}} d = \sum_{\substack{d \backslash n \\ 1<d^2 \leq n}} \left(d + [d^2 < n] \frac{n}{d} \right).$$

Entrada: $n \in \mathbb{N}$.
Salida: $S = \sigma(n)$.

1: $S \leftarrow 0, d \leftarrow 1, c \leftarrow 1$.
2: **repetir**
3: **si** $n \equiv 0 \mod d$ **entonces**
4: $S \leftarrow S + d$.
5: **si** $c < n$ **entonces**
6: $S \leftarrow S + \frac{n}{d}$
7: $c \leftarrow c + d + d + 1, d \leftarrow d + 1$.
8: **hasta que** $c > n$

Para la suma de todos los divisores de n, se puede usar $\sigma_1(n) = \prod_{i=1}^{s} \frac{p_i^{\alpha_i+1}-1}{p_i-1}$.

4.5. Números primos

Los números $M_p = 2^p - 1$, con p primo, se denominan *primos de Mersenne*.

Teorema 4.3 (Lucas-Lehmer). *Sea r_n la sucesión definida a través de $r_1 = 4, r_{n+1} = r_n^2 - 2$. Entonces M_p es primo si, y sólo si, M_p / r_{p-1}.*

Teorema 4.4 (Euclides). *Si M_p es primo, entonces $2^{p-1}(2^p - 1)$ es perfecto, y recíprocamente.*

Teorema 4.5. *El número de Fermat $F_n = 2^{2^n} + 1$ es primo si, y sólo si, $3^{\frac{F_n - 1}{2}} \equiv -1 \pmod{F_n}$.*

Del Teorema 4.7 se sigue que si p es un primo impar y $(a, p) = 1$ entonces

$$a^{p-1} \equiv 1 \pmod{p}. \tag{25}$$

Todo número que satisface (25) se dice un pseudoprimo para la base a. Los primos impares son pseudoprimos para todas sus bases coprimas. Los números compuestos que son pseudoprimos para todas sus bases coprimas se denominan *números de Carmichael*.

Teorema 4.6. *El entero $n > 1$ con descomposición* (24) *es un número de Carmichael si, y sólo si, $\alpha_i = 1$ y $(p_i - 1)/(n-1)$ para todo $i = 1, \ldots, s$.*

Algoritmo 4.7 (Eratóstenes). Obtención de los primos hasta una cota dada por medio de una criba.

Entrada: Una cota N para obtener todos los primos impares hasta $2N + 1$.
Salida: Un arreglo X tal que $X_k = [2k+1 \text{ no es primo}]$.

1: $X \leftarrow (0, \ldots, 0)$.
2: **para** $k \leftarrow 3$ hasta $\sqrt{2N+1}$ sumando 2 **hacer**
3: **si** $X_{(k-1)/2} = 0$ **entonces**
4: **para** $i \leftarrow k^2$ hasta $2N + 1$ sumando $2k$ **hacer**
5: $X_{(i-1)/2} \leftarrow 1$.

4.6. Potencias y logaritmos discretos

Teorema 4.7 (Euler). *Para todo $n > 1, a \in \mathbb{Z}_n$, $a^{\phi(n)} \equiv 1 \pmod{n}$, donde $\phi(n)$ es la cardinalidad del conjunto de números que son coprimos con n.*

La función ϕ satisface $\phi(n) = \prod_{i=1}^{s} p_i^{\alpha_i - 1}(p_i - 1) = n \prod_{p/n} \left(1 - \frac{1}{p}\right)$.

Teorema 4.8. *El grupo multiplicativo $\mathbb{Z}_n$ es cíclico si, y sólo si, $n = 2, 4, p^e$ o $2p^e$ para un primo impar p y algún $e \in \mathbb{Z}$ positivo.*

Teorema 4.9. *Si $\mathbb{Z}_n$ está generado por g, entonces $g^x \equiv g^y \mod n$ si, y sólo si, $x \equiv y \pmod{\phi(n)}$*

Algoritmo 4.8 (Exponenciación rápida). Calcula a^b en un semigrupo en tiempo $O(\log b)$.

Entrada: a en un semigrupo S y $b \in \mathbb{Z}$.
Salida: a^b.

1: $r \leftarrow 1$.
2: **mientras** $b > 0$ **hacer**
3: **si** $b \equiv 1 \mod 2$ **entonces**
4: $r \leftarrow a \cdot r$.
5: $a \leftarrow a \cdot a, b \leftarrow \left\lfloor \frac{b}{2} \right\rfloor$.
6: **devolver** r.

4.7. Teorema chino del residuo

Teorema 4.10. *Sean $a_1, \ldots, a_n, m_1, \ldots, m_n \in \mathbb{Z}$ tales que los elementos de $\{m_i\}_{i=1}^n$ son coprimos. Entonces el sistema $x \equiv a_k \pmod{m_k}$, $k = 1, \ldots, n$, tiene solución.*

Algoritmo 4.9 (Chino inductivo). Calcula una solución al sistema del Teorema 4.10.

Entrada: Dos conjunto de enteros: $\{m_i\}_{i=1}^n$ coprimos a pares y $\{x_i\}_{i=1}^n$.
Salida: x tal que $x \equiv x_i \pmod{m_i}$.

1: $m \leftarrow m_1, x \leftarrow x_1$.
2: **para** $i \leftarrow 2$ hasta k **hacer**
3: $(d, u, v) \leftarrow \text{MCD}(m, m_i)$.
4: $x \leftarrow umx_i + vm_i x, m \leftarrow mm_i, x \leftarrow x \mod m$
5: **devolver** x.

5. Combinatoria

Teorema 5.1 (Principio de inclusión y exclusión). *Sea $\{A_i\}_{i=1}^n$ una colección de conjuntos. Sea $\mathcal{P}_+(n)$ el conjunto de subconjuntos no vacíos I de $\{1, \ldots, n\}$. Entonces $\left|\bigcup_{i=1}^n A_i\right| = \sum_{I \in \mathcal{P}_+(n)} (-1)^{|I|+1} \left|\bigcap_{i \in I} A_i\right|$.*

Teorema 5.2. *Hay $\binom{n+k-1}{k-1}$ formas de colocar n objetos distinguibles en k cajas distinguibles.*

Sea G un grupo que actúa por la izquierda sobre un conjunto S. Para $g \in S$ consideramos el conjunto $F(g) = \{s \in S : gs = s\}$. La *órbita* de $s \in S$ es el conjunto $\text{orb}(s) = \{gs : g \in G\}$.

Teorema 5.3 (Burnside). *Sea G un grupo finito que actúa sobre el conjunto S. El número de órbitas de G en S es $\frac{1}{|G|} \sum_{g \in G} |F(g)|$.*

5.1. Objetos combinatorios

Aquí T es un segmento inicial de $\mathbb{N}$ de longitud n.

Algoritmo 5.1 (sucesor, código Gray). Aquí se considera el orden de cambio mínimo para los subconjuntos de T.

si $|T| \equiv 0 \mod 2$ **entonces**
  $U \leftarrow T\Delta\{n\}$.
si no
  $j \leftarrow n$.
  mientras $j \notin T$ y $j > 0$ **hacer**
    $j \leftarrow j - 1$.
  si $j = 1$ **entonces**
    devolver falso.
  $U \leftarrow T\Delta\{j - 1\}$.
$T = U$.
devolver verdadero.

Algoritmo 5.2 (colocador, código Gray). Se considera el orden de cambio mínimo para los subconjuntos de T.

$b \leftarrow r \leftarrow 0$.
para $i \leftarrow n - 1$ decreciendo hasta 0 **hacer**
  si $n - i \in T$ **entonces**
    $b \leftarrow 1 - b$.
  si $b = 1$ **entonces**
    $r \leftarrow r + 2^i$.
devolver r.

Algoritmo 5.3 (extractor, código Gray). Se considera el orden de cambio mínimo para los subconjuntos de T.

$T \leftarrow \emptyset, b' \leftarrow 0$.
para $i \leftarrow n - 1$ decreciendo hasta 0 **hacer**
  $b \leftarrow \left\lfloor \frac{r}{2^i} \right\rfloor$
  si $b \neq b'$ **entonces**
    $T \leftarrow T \cap \{n - i\}$.
    $b' \leftarrow b, r \leftarrow r - b2^i$.
devolver T.

Algoritmo 5.4 (sucesor, k-subconjuntos). Calcula el sucesor de un k-subconjunto (combinación) de T en orden lexicográfico.

$U \leftarrow T, i \leftarrow k$
mientras $((i \geq 1) \wedge (t_i = n - k + 1))$ **hacer**
  $i \leq i - 1$.
si $i = 0$ **entonces**
  devolver falso.
si no
  para $j \leftarrow i$ hasta k **hacer**
    $u_j \leftarrow t_i + 1 + j - i$.
  devolver verdadero.

Algoritmo 5.5 (colocador, k-subconjuntos). Coloca un k-subconjunto (combinación) de T en la posición r.

$t_0 \leftarrow r \leftarrow 0$.
para $i \leftarrow n - 1$ hasta k **hacer**
  si $t_{i-1} + 1 \leq t_i - 1$ **entonces**
    $b \leftarrow 1 - b$.
    para $j \leftarrow t_{i-1} + 1$ hasta $t_i - 1$ **hacer**
      $r \leftarrow r + \binom{n-j}{k-i}$
devolver r.

Algoritmo 5.6 (extractor, k-subconjuntos). Extrae de la posición r el k-subconjunto (combinación) de T.

$x \leftarrow 1$
para $i \leftarrow 1$ hasta k **hacer**
  mientras $\binom{n-x}{k-i} \leq r$ **hacer**
    $r \leftarrow r - \binom{n-x}{k-i}, x \leftarrow x + 1$.
  $t_i \leftarrow x, x \leftarrow x + 1$.
devolver T.

Algoritmo 5.7 (extractor, permutaciones). Extrae de la posición r una permutación de T en orden lexicográfico.

$\pi_n \leftarrow 1$.
para $j \leftarrow 1$ hasta $n - 1$ **hacer**
  $d \leftarrow \lfloor \frac{r \mod (j-1)!}{j!} \rfloor, r \leftarrow r - d \cdot j!, \pi_{n-j} \leftarrow d + 1$
  para $i \leftarrow n - j + 1$ hasta n **hacer**
    si $\pi_i > d$ **entonces**
      $\pi_i \leftarrow \pi_i + 1$.
devolver π.

Algoritmo 5.8 (colocador, permutaciones). Coloca en la posición r una permutación de T en orden lexicográfico.

$r \leftarrow 0, \rho \leftarrow \pi$.
para $j \leftarrow 1$ hasta n **hacer**
  $r \leftarrow r + (\rho_j - 1)(n - j)!$
  para $i \leftarrow j + 1$ hasta n **hacer**
    si $\rho_i > \rho_j$ **entonces**
      $\rho_i \leftarrow \rho_i - 1$.
devolver r.

5.2. Números combinatorios

1. Subconjuntos de k elementos de un conjunto de n elementos: $\binom{n}{k}$.

   Fórmula explícita: $\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$.
   Recurrencia: $\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}$.
   Valores iniciales: $\binom{n}{0} = \binom{n}{n} = 1, n \geq 0$.

2. Permutaciones de n letras que tienen exactamente k ciclos: $\begin{bmatrix} n \\ k \end{bmatrix}$.

   Recurrencia: $\begin{bmatrix} n \\ k \end{bmatrix} = \begin{bmatrix} n-1 \\ k-1 \end{bmatrix} + (n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix}$
   Valores iniciales: $\begin{bmatrix} 0 \\ k \end{bmatrix} = [k = 0], \begin{bmatrix} n \\ 0 \end{bmatrix} = [n = 0]$.

3. Permutaciones de n letras que tienen exactamente k ascendentes: $\left\langle {n \atop k} \right\rangle$.

   Recurrencia: $\left\langle {n \atop k} \right\rangle = (k+1)\left\langle {n-1 \atop k} \right\rangle + (n-k)\left\langle {n-1 \atop k-1} \right\rangle$
   Valores iniciales: $\left\langle {0 \atop k} \right\rangle = [k=0]$.

4. Particiones de un conjunto de n elementos en k clases: $\left\{ {n \atop k} \right\}$.

   Recurrencia: $\left\{ {n \atop k} \right\} = \left\{ {n-1 \atop k-1} \right\} + k\left\{ {n-1 \atop k} \right\}$.
   Valores iniciales: $\left\{ {0 \atop k} \right\} = [k=0], \left\{ {n \atop 0} \right\} = [n=0]$.

5. Particiones $p(n,k)$ de n cuya mayor parte es k.

   Recurrencia: $p(n,k) = p(n-1,k-1) + p(n-k,k)$.
   Valores iniciales: $p(n,k) = 0$ si $n \le 0$, $k \le 0$ o $k > n$ y $p(1,1) = 1$.

6. Números de Catalan C_n.

   Fórmula explícita: $C_n = \frac{1}{n+1}\binom{2n}{n}$.
   Recurrencias: $C_{n+1} = \frac{2(2n+1)}{n+2} C_n = \sum_{k=0}^{n} C_k C_{n-k}$.

6. Métodos numericos

6.1. Interpolación

Dados los puntos $\{(x_i, y_i)\}_{i=0}^{n}$, la recurrencia de Neville

$$Q_{i,j}(x) = \frac{(x - x_{i-j})Q_{i,j-1}(x) - (x - x_i)Q_{i-1,j-1}(x)}{x_i - x_{j-1}}$$

calcula el valor de la interpolación polinomial de dichos puntos, usando como valores iniciales $Q_{i,0} = y_i$.

6.2. Diferenciación

La fórmula central de f' para una función $f \in C^3[a,b]$ con $x-h, x, x+h \in [a,b]$ es

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} =: f_c(x,h),$$

con error $O(h^2)$. La fórmula central de f' para una función $f \in C^5[a,b]$ con $x-2h, x-h, x, x+h, x+2h \in [a,b]$ es

$$f'(x) \approx 8f_c(x,h) - 2f_c(x,2h),$$

con error $O(h^4)$.

6.3. Integración

Algoritmo 6.1 (Método de Simpson adaptativo). En el siguiente algoritmo, definimos $S(a,b,c,h) = \frac{h}{3}(f(a) + 4f(c) + f(b))$.

Entrada: Un intervalo de integración $[a,b]$, una función $f \in C^4[a,b]$ y una tolerancia ϵ.

Salida: El valor de $\int_a^b f(x)\,dx$ con la tolerancia establecida.

1: **función** SIMP($f, [a,b], \epsilon$)
2: $h \leftarrow \frac{b-a}{2}$, $c = \frac{b+a}{2}$.
3: $a_1 \leftarrow a, b_1 \leftarrow a_2 \leftarrow c, b_2 \leftarrow b$.
4: $c_1 = \frac{b_1+a_1}{2}, c_2 = \frac{b_2+a_2}{2}$.
5: $L = \frac{S(a_1,b_1,c_1,h)+S(a_2,b_2,c_2,h)}{2}$.
6: **si** $\frac{1}{10}|L - S(a,b)| < \epsilon$ **entonces**
7: **devolver** L.
8: **si no**
9: **devolver** SIMP($f, [a_1,b_1], \frac{\epsilon}{2}$)+SIMP($f, [a_2,b_2], \frac{\epsilon}{2}$).

Sea $f : \mathbb{R} \to \mathbb{R}$. Supongamos que $J \ge 1$ y que los puntos $\{x_k = a + kh\}$ subdividen al intervalo $[a,b]$ en $2^J = 2M$ intervalos de anchura $h = \frac{b-a}{2^J}$. Haciendo $T(0) = \frac{h}{2}(f(a) + f(b))$, la fórmula trapezoidal recurrente es

$$T(J) = \frac{T(J-1)}{2} + h\sum_{k=1}^{M} f(x_{2k-1}).$$

Iniciando la tabla $R_{J,0} = T(J)$, tenemos la recursión de Richardson

$$R_{J,K} = \frac{4^K R_{J,K-1} - R_{J-1,K-1}}{4^K - 1},$$

que aproxima a $\int_a^b f(x)\,dx$ con un error de truncamiento de orden $O(h^{2K+2})$ para $R(J,K)$ cuando $f \in C^{2K+2}[a,b]$.

6.4. Transformación rápida de Fourier

Algoritmo 6.2. Transformación rápida de Fourier.

1: **función** TRF(a)
2: $n \leftarrow |a|$ {n es una potencia de 2}
3: **si** $n = 1$ **entonces**
4: **devolver** a.
5: **para** $j \leftarrow 1$ hasta $n/2$ **hacer**
6: $b_j \leftarrow a_{2j-1}$, $c_j \leftarrow a_{2j}$
7: $b^* \leftarrow$ TRF(b), $c^* \leftarrow$ TRF(c).
8: $\omega_n \leftarrow \exp\left(\frac{2\pi}{n}\right)$, $\omega_n \leftarrow \exp\left(\frac{2\pi}{n}\right)$.
9: $\omega \leftarrow 1$.
10: **para** $j \leftarrow 1$ hasta $n/2$ **hacer**
11: $a_j^* \leftarrow b_j^* + \omega c_j^*$, $a_{j+n/2+1}^* \leftarrow b_j^* - \omega c_j^*$.
12: $\omega \leftarrow \omega\omega_n$ {Usar $\overline{\omega}_n$ para la inversa}

Algoritmo 6.3. Multiplicación de polinomios con TRF.

Entrada: Dos sucesiones $a = \{a_k\}_{k=0}^{n-1}$ y $b = \{b_k\}_{k=0}^{m-1}$ que son los coeficientes de los polinomios $p(x) = \sum_{k=0}^{n-1} a_k x^k$ y $q(x) = \sum_{k=0}^{m-1} b_k x^k$.

Salida: Una sucesión $c = \{c_k\}_{k=0}^{n-1}$ de los coeficientes del polinomio $p(x) \cdot q(x)$.

1: **función** MULTPOLINOMIO(a, b)

2: $\ell \leftarrow \lceil \log_2(n+m) \rceil$
3: **para** $j \leftarrow n$ hasta $2^\ell - 1$ **hacer**
4: $\quad a_j \leftarrow 0$.
5: **para** $j \leftarrow m$ hasta $2^\ell - 1$ **hacer**
6: $\quad b_j \leftarrow 0$.
7: $\alpha = \text{TRF}(a), \beta = \text{TRF}(b)$
8: **para** $k = 0$ hasta $k = n - 1$ **hacer**
9: $\quad \gamma_k \leftarrow \alpha_k \cdot \beta_k$.
10: **devolver** $c \leftarrow \text{ITRF}(\gamma)$.

6.5. Álgebra lineal

Algoritmo 6.4. Encuentra la descomposición LUP de una matriz $A \in \mathcal{M}_{m\times n}$.

1: **para** $i = 1$ hasta n **hacer**
2: $\quad \pi[i] \leftarrow i$
3: **para** $k \leftarrow 1$ hasta n **hacer**
4: $\quad p \leftarrow 0$
5: $\quad$ **para** $i \leftarrow k$ hasta n **hacer**
6: $\quad\quad$ **si** $|a_{ik}| > p$ **entonces**
7: $\quad\quad\quad p \leftarrow |a_{ik}|, k' \leftarrow i$.
8: $\quad\quad$ **si** $p = 0$ **entonces**
9: $\quad\quad\quad$ **error** Matriz singular.
10: $\quad\quad$ Intercambiar $\pi[k] \leftrightarrow \pi[k']$.
11: $\quad\quad$ **para** $i \leftarrow 1$ hasta n **hacer**
12: $\quad\quad\quad$ Intercambiar $a_{k1} \leftrightarrow a_{k'i}$.
13: $\quad\quad$ **para** $i \leftarrow k+1$ hasta n **hacer**
14: $\quad\quad\quad a_{ik} \leftarrow a_{ik}/a_{kk}$.
15: $\quad\quad\quad$ **para** $j \leftarrow k+1$ hasta n **hacer**
16: $\quad\quad\quad\quad a_{ij} \leftarrow a_{ij} - a_{ik}a_{kj}$.

Algoritmo 6.5. Resuelve el sistema $Ax = b$ con la descomposición LUP de $A \in \mathcal{M}_{m\times n}$.

1: **para** $i = 1$ hasta n **hacer**
2: $\quad y_i \leftarrow b_{\pi[i]} - \sum_{j=1}^{i-1} L_{ij}y_j$.
3: **para** $i = n$ hasta 1 **hacer**
4: $\quad x_i \leftarrow \frac{y_i - \sum_{j=i+1}^{n} U_{ij}x_j}{U_{ij}}$.

7. Probabilidad

Teorema 7.1 (Desigualdad de Markov). *Sea $t > 0$ y X una variable aleatoria no negativa. Entonces $P(X \geq t) \leq \frac{E(X)}{t}$.*

Teorema 7.2 (Desigualdad de Chebyshev). *Sea $t > 0$ y X una variable aleatoria no negativa. Entonces $P(|X - E(X)| \geq t) \leq \frac{V(X)}{t^2}$.*

Sean dos eventos A y B, con sendas probabilidades $P(A)$ y $P(B)$.

1. La probabilidad condicional $P(A|B)$ satisface $P(A|B) = \frac{P(A\cap B)}{P(B)}$.
2. Los eventos A y B son independientes si $P(A|B) = P(A)$. Además, $P(AB) = P(A) \cdot P(B)$.
3. Se satisface $P(A \cup B) = P(A) + P(B) - P(A \cap B)$.

Si X e Y son variables aleatorias independientes, con distribuciones f y g respectivamente, entonces la distribución de $Z = X + Y$ es $f * g = \int_{-\infty}^{\infty} f(t)g(x-t)\,dt$.

8. Ordenamiento y búsqueda

8.1. Algoritmos basados en comparaciones

En los algoritmos de esta sección N es la cardinalidad del conjunto L a ordenar.

Algoritmo 8.1 (Ordenamiento por inserción). Puede calcular el número mínimo de intercambios para ordenar a L. En cada iteración del ciclo externo los elementos $L_0, \ldots L_i$ forman una lista ordenada.

1: $\mu \leftarrow 0$.
2: **para** $i \leftarrow 1$ hasta $i < N$ **hacer**
3: $\quad t \leftarrow L_i, j \leftarrow i - 1$.
4: $\quad$ **mientras** $L_j > t$ y $j \geq 0$ **hacer**
5: $\quad\quad L_{j+1} \leftarrow L_j,\ j \leftarrow j - 1$.
6: $\quad\quad L_{j+1} \leftarrow t, \mu \leftarrow \mu + 1$.
7: **devolver** μ.

Algoritmo 8.2 (Ordenamiento por fusión). Puede calcular el número mínimo de intercambios para ordenar a L en tiempo $O(n \log n)$.

1: $\mu \leftarrow 0$.
2: **función** PARTIR(i, j).
3: **si** $i \neq j$ **entonces**
4: $\quad p \leftarrow \lfloor \frac{i+j}{2} \rfloor$.
5: $\quad$ PARTIR(i, p), PARTIR$(p+1, j)$.
6: $\quad a \leftarrow 0, b \leftarrow 0, c \leftarrow 0$.
7: $\quad$ **mientras** $(a < (p+i-1)) \wedge (b < (j-p))$ **hacer**
8: $\quad\quad$ **si** $L_{i+a} < L_{p+1+b}$ **entonces**
9: $\quad\quad\quad \Lambda_c \leftarrow L_{i+a}, a \leftarrow a + 1$.
10: $\quad\quad$ **si no**
11: $\quad\quad\quad \Lambda_c \leftarrow L_{p+i+b}, b \leftarrow b + 1$.
12: $\quad\quad\quad \mu \leftarrow \mu + p + 1 - (a + i)$. {Intercambio.}
13: $\quad\quad c \leftarrow c + 1$.
14: $\quad$ **mientras** $i + a \leq p$ **hacer**
15: $\quad\quad \Lambda_c \leftarrow L_{i+a}, a \leftarrow a + 1, c \leftarrow b + 1$.
16: $\quad$ **mientras** $p + 1 + b \leq j$ **hacer**
17: $\quad\quad \Lambda_c \leftarrow L_{p+1+b}, b \leftarrow b + 1, c \leftarrow b + 1$.
18: $\quad$ **para todo** $k, 0 \leq k \leq j - i$ **hacer**
19: $\quad\quad L_{i+k} = \Lambda_i$.

Algoritmo 8.3. Algoritmo de ordenación rápida de Hoare. Se invoca con ORDRAP$(0, N-1)$.

```
función OrdRap(i, j)
d ← L_j
λ ← i − 1, κ ← j, c ← 1
si i ≥ j entonces
    devolver ∅.
mientras c > 0 hacer
    mientras L_λ < d hacer
        λ ← λ + 1.
    mientras L_κ > d hacer
        κ ← κ − 1.
    si λ < κ entonces
        Intercambiar L_λ ↔ L_κ
    si no
        c ← 0.
Intercambiar L_λ ↔ L_κ
OrdRap(i, λ − 1), OrdRap(λ + 1, j).
```

8.2. Algoritmos lineales

Algoritmo 8.4 (Conteo).

Entrada: Un arreglo A a ordenar, y una cota para los datos k.

```
para i ← 1 hasta k hacer
    C[i] ← 0.
para j ← 1 hasta longitud(A) hacer
    C[A[j]] ← C[A[j]] + 1.
para i ← 2 hasta k hacer
    C[i] ← C[i] + C[i − 1]
para j = |A| decreciendo hasta 1 hacer
    B[C[A[j]]] ← A[j]
    C[A[j]] ← C[A[j]] − 1
```

Algoritmo 8.5 (Cubeta). Ordena $\{A_i\}_{i=1}^n \subset [0, 1)$ usando n subintervalos iguales de $[0, 1)$.

```
para i ← 1 hasta n hacer
    Insertar A[i] en la lista B[⌊nA[i]⌋].
para i ← 0 hasta n − 1 hacer
    Ordenar la lista B[i] con inserción.
Concatenar las listas B[0], . . . , B[n − 1] en orden.
```

8.3. Búsqueda binaria

Algoritmo 8.6 (búsqueda binaria). Algoritmo que busca una llave en un conjunto ordenado en tiempo $O(\log_2 n)$.

Entrada: La llave K para buscar, el límite inferior i y superior j donde se hará la búsqueda.

Salida: k tal que $L_k = K$ o falso si no existe.

```
mientras j − i > 1 hacer
    p = ⌊(j+i)/2⌋.
    si L_p < K entonces
        i ← p.
    si no
        j ← p.
si L_i = K entonces
    devolver i.
si no si L_j = K entonces
    devolver j.
si no
    devolver falso.
```

9. Programación dinámica

9.1. Suma de subconjunto

Algoritmo 9.1. Dado un conjunto de enteros positivos $\{a_1, \ldots, a_n\}$, queremos saber cuántos subconjuntos de tal conjunto suman B. Definimos la recurrencia

$$m(1, j) = [j = a_1],$$
$$m(i, j) = m(i - 1, j) + [j > a_i]m(i - 1, j - a_i) + [j = a_i],$$

para $2 \le i \le n$ y $1 \le j \le B$. El valor de $m(n, B)$ nos da la respuesta, calculando m tiempo $O(nB)$.

```
para j ← 1 hasta B hacer
    m(1, j) ← [j = a_1].
para i ← 2 hasta n hacer
    para j ← 1 hasta B hacer
        m(i, j) ← m(i − 1, j) + [j = a_i].
        si j > a_i entonces
            m(i, j) ← m(i, j) + m(i − 1, j − a_i)
```

Algoritmo 9.2 (Recuperación de un subconjunto). La tabla generada anteriormente da información suficiente para encontrar un subconjunto en tiempo $O(B)$, llamando a Secuencia(n, B).

```
función Secuencia(i, B)
si m(i, B) = 0 entonces
    imprime "No hay solución".
si no si B = a_i entonces
    imprime a_i.
si no si B > a_i entonces
    imprime a_i.
    Secuencia(i − 1, B − a_i).
```

9.2. Estructuras de Catalan óptimas

Algoritmo 9.3 (Mínimo genérico). Se le ajusta la relajación apropiada al siguiente algoritmo.

```
función Mínimo
para i ← 1 hasta n − 1 hacer
    M_{i,i+1} ← 0.
para j ← 2 hasta n − 1 hacer
```

5: **para** $i \leftarrow 1$ hasta $n - d$ **hacer**
6: RELAJA(i, j).
7: **devolver** $M_{1,n}$

Algoritmo 9.4 (Relajación). Si el problema es de triangulación óptima usamos:

$$F(i,j,k) = \begin{cases} \frac{1}{2}|K(p_i, p_j, p_k)| & \text{(Área)}, \\ \|p_i, p_j\| + \|p_j, p_k\| + \|p_k, p_i\| & \text{(Perímetro)}. \end{cases}$$

donde $\|\cdot,\cdot\|$ es una métrica adecuada. Si es de multiplicación óptima de n matrices, usamos

$$F(i,j,k) = D(i) \cdot D(j) \cdot D(k)$$

donde la matriz i-ésima es de dimensión $D(i) \times D(i+1)$ y $1 \leq i \leq n$. Para el problema de secuencia óptima de corte, tenemos

$$F(i,j,k) = C(j) - C(i),$$

donde $C(i)$ es el corte i-ésimo.

1: **función** RELAJA(i, j)
2: $M_{i,j} \leftarrow \infty$.
3: **para** $k \leftarrow i + 1$ hasta $j - 1$ **hacer**
4: $t \leftarrow F(i,j,k) + M_{i,k} + M_{k,j}$.
5: **si** $M_{i,j} > t$ **entonces**
6: $M_{i,j} \leftarrow t, s_{i,j} \leftarrow k$

Algoritmo 9.5 (Recuperación de la secuencia). Para recuperar la secuencia de pasos para obtener el mínimo, invocamos a SECUENCIA$(s, 1, n)$.

1: **función** SECUENCIA(s, i, j)
2: **si** $i = j$ **entonces**
3: **imprime** "A"+i.
4: **si no**
5: **imprime** "(".
6: SECUENCIA$(s, i, s_{i,j})$.
7: SECUENCIA$(s, s_{i,j} + 1, j)$.
8: **imprime** ")".

9.3. Mayor subsecuencia común

Algoritmo 9.6 (Longitud de la mayor subsecuencia común).

Entrada: Cadenas X e Y de sendas cardinalidades m y n.
Salida: La longitud de la M. S. C. en $c_{m,n}$.

1: **para** $i \leftarrow 1$ hasta m **hacer**
2: $c_{i,0} \leftarrow 0$.
3: **para** $j \leftarrow 1$ hasta n **hacer**
4: $c_{0,j} \leftarrow 0$.
5: **para** $i \leftarrow 1$ hasta m **hacer**
6: **para** $j \leftarrow 1$ hasta n **hacer**
7: **si** $X_{i-1} = Y_{j-1}$ **entonces**
8: $c_{i,j} \leftarrow c_{i-1,j-1} + 1, b_{i,j} \leftarrow 1$.
9: **si no si** $c_{i-1,j} \geq c_{i,j-1}$ **entonces**
10: $c_{i,j} \leftarrow c_{i-1,j}, b_{i,j} \leftarrow 2$.
11: **si no**
12: $c_{i,j} \leftarrow c_{i,j-1}, b_{i,j} \leftarrow 3$.
13: **devolver** $c_{m,n}$

Algoritmo 9.7 (Recuperación de la subsecuencia). Requiere que se haya ejecutado previamente el algoritmo anterior.

1: **función** SUBSECUENCIA(i, j, X, b)
2: **si** $i = 0 \vee j = 0$ **entonces**
3: **devolver** $\emptyset$.
4: **si** $b_{i,j} = 1$ **entonces**
5: SUBSECUENCIA$(i - 1, j - 1, X, b)$
6: **imprime** X_{i-1}.
7: **si no si** $b_{i,j} = 2$ **entonces**
8: SUBSECUENCIA$(i - 1, j, X, b)$
9: **si no**
10: SUBSECUENCIA$(i, j - 1, X, b)$

9.4. Distancia de edición

Sean m la matriz de costos mínimos de transformacion y p la matriz de predecesores. Considerando que la operacion 1 es cambiar, la 2 es insertar y la 3 es borrar, tenemos las siguientes condiciones iniciales

$$d_{0,0} = 0,\ d_{0,j} = j, d_{i,0} = i,$$
$$p_{0,0} = -1,\ p_{0,j} = 1, p_{i,0} = 2,$$

para $0 \leq i \leq |s_1|$ y $0 \leq j \leq |s_2|$. Definamos

$$T_1(i,j) = d_{i-1,j-1} + [s_i \neq s_j], \quad \text{(Cambiar)}$$
$$T_2(i,j) = d_{i,j-1} + 1, \quad \text{(Insertar)}$$
$$T_3(i,j) = d_{i-1,j} + 1. \quad \text{(Borrar)}$$

Las relaciones de recurrencia son:

$$d_{i,j} = \min_k\{T_k(i,j) : 1 \leq k \leq 3\}, \quad p_{i,j} = k.$$

Tenemos que $d_{|s_1|,|s_2|}$ es la distancia de edición.

9.5. Mayor subsecuencia creciente o decreciente

Algoritmo 9.8. Dada la sucesión $\{A_i\}_{i=1}^n$, encuentra una subsucesión $\{A_{i_k}\}_{k=1}^m$ que sea estrictamente creciente o decreciente de máxima cardinalidad.

Entrada: Una sucesión $\{A_i\}_{i=1}^n$.
Salida: Un conjunto de longitudes máximas ℓ y un conjunto de predecesores π.

1: **para** $i \leftarrow 1$ hasta $n-1$ **hacer**
2: **para** $j \leftarrow i+1$ hasta n **hacer**
3: **si** $A_j > A_i$ y $\ell_i + 1 > \ell_j$ **entonces**
4: $\ell_j \leftarrow \ell_i + 1, \pi_j \leftarrow i$.

9.6. Conteo de cambio

Algoritmo 9.9.

Entrada: Un conjunto $M = \{m_i\}_{i=0}^n \subset \mathbb{N}$ y $W \in \mathbb{N}$.

Salida: El número de submulticonjuntos de M que suman W.

1: $F_0 \leftarrow 1$
2: **para** $i \leftarrow 0$ hasta $n-1$ **hacer**
3: $c \leftarrow M_i$
4: **para** $j \leftarrow c$ hasta W **hacer**
5: $F_j \leftarrow F_j + F_{j-c}$
6: **devolver** F_W.

10. Rastreo hacia atrás

Solución de un problema por rastreo hacia atrás.

Algoritmo 10.1. Esquema para una solución.

procedimiento Ensayar(paso : TipoPaso)
repetir
Seleccionar candidato.
si aceptable **entonces**
Anotar candidato.
si solución incompleta **entonces**
Ensayar(paso siguiente)
si no acertado **entonces**
Borrar candidato.
si no
Anotar solución.
Hacer acertado ← cierto.
hasta que acertado=cierto o no haya candidatos
fin procedimiento

Algoritmo 10.2. Esquema para todas las soluciones.

procedimiento Ensayar(paso : TipoPaso)
para todo candidato **hacer**
Seleccionar candidato.
si aceptable **entonces**
Anotar candidato.
si solución incompleta **entonces**
Ensayar(paso siguiente).
si no
Almacenar solución.
Borrar candidato.
fin procedimiento

11. Algoritmos voraces

Algoritmo 11.1 (Mochila continua).

Entrada: Un conjunto $\{(s_i, w_i)\}_{i=1}^n$ de pares ordenados de reales positivos y un objetivo S.

Salida: Un conjunto $\{x_i\}_{i=1}^n$ tal que $S = \sum_{i=1}^n x_i s_i$ y $\sum_{i=1}^n x_i w_i$ es mínimo.

1: Ordenar $\{(s_i, w_i)\}_{i=1}^n$ según w_i/s_i en forma no decreciente.
2: $s \leftarrow S, i \leftarrow 1$.
3: **mientras** $s_i \leq s$ **hacer**
4: $x_i \leftarrow 1, s = s - s_i, i \leftarrow i+1$.
5: $x_i \leftarrow s/s_i$.
6: **para** $j \leftarrow i+1$ hasta n **hacer**
7: $x_j \leftarrow 0$.

Algoritmo 11.2 (Mochila 0-1). Si tenemos n objetos con pesos W_i y valores V_i con $1 \leq i \leq n$ y una mochila con capacidad w. ¿Qué objetos debemos meter en la mochila para maximizar el valor de lo que contiene? La solución se puede encontrar en $C_{n,w}$.

Entrada: Los pesos $\{W_i\}_{i=1}^n$ y los valores $\{V_i\}_{i=1}^n$ de los n objetos.

Salida: El máximo valor que puede contener una mochila de capacidad w.

1: **para** $i \leftarrow 0$ hasta N **hacer**
2: $C_{0,w} \leftarrow C_{i,0} \leftarrow 0$
3: **para** $i \leftarrow 1$ hasta N **hacer**
4: **para** $j \leftarrow 1$ hasta w **hacer**
5: **si** $W_i > w$ **entonces**
6: $C_{i,j} = C_{i-1,j}$.
7: **si no**
8: $C_{i,j} = \max(C_{i-1,j}, C_{i-1,j-W_i} + V_i)$.

12. Teoría de grafos

Para un grafo G con $V(G) = \{v_i\}_{i=1}^p$, la sucesión $\{\operatorname{grad} v_i\}_{i=1}^p$ ordenada en forma decreciente se denomina *sucesión de grados* de G. Una sucesión de enteros no negativos se dice *gráfica* si es la sucesión de grados de algún grafo.

Teorema 12.1 (Havel-Hakimi). *Una sucesión $\{s_i\}_{i=1}^p$ de enteros no negativos, con $p \geq 2$ y $d_i \geq 1$, es gráfica si, y sólo si, la sucesión $d_2 - 1, d_3 - 1, \ldots, d_{d_1+1} - 1, d_{d_1+2}, d_{d_1+3}, \ldots, d_p$ es gráfica.*

Algoritmo 12.1. Creación de un conjunto.

1: **función** HazConjunto(x)
2: $p(x) \leftarrow x, r(x) \leftarrow 0$.

Algoritmo 12.2.

1: **función** HALLA(x)
2: **mientras** $x \neq p(x)$ **hacer**
3: $x \leftarrow p(x)$.
4: **devolver** $p(x)$

Algoritmo 12.3.

1: **función** ENLAZA(x, y)
2: **si** $r(x) > r(y)$ **entonces**
3: $x \leftrightarrow y$.
4: **si** $r(x) = r(y)$ **entonces**
5: $r(y) \leftarrow r(y) + 1$.
6: $p(x) \leftarrow y$.
7: **devolver** y.

Algoritmo 12.4. Une conjuntos considerando el peso.

1: **función** UNIÓN(x, y)
2: ENLAZA(HALLA(x),HALLA(y)).

12.1. Circuitos eulerianos

Teorema 12.2. *Un grafo conexo contiene un circuito euleriano si, y sólo si, a) cada vértice es de grado par; b) las aristas pueden particionarse en ciclos ajenos.*

Algoritmo 12.5 (Hierholzer). Halla un circuito euleriano pegando circuitos.

Entrada: Un grafo conexo con todos sus vértices de grado par.
Salida: Un circuito euleriano C de G.

1: Sea $v \in V$. Construir un ciclo C empezando en v.
2: **mientras** $E(C) \neq E(G)$ **hacer**
3: Sea $w \in C, u \in V$ tal que $\{w, u\} \notin E(C)$.
4: Construir un ciclo C^* empezando en w en el grafo $G - E(C)$.
5: Insertar a C^* en lugar de w.

12.2. Recorridos

Algoritmo 12.6. Al hacer recorrido por profundidad, si el grafo no es conexo, marcar cada vértice visitado apropiadamente.

Entrada: Un grafo $G = (V, E)$ con $|V| = n$.
Salida: Una función de profundidad o anchura p y el número c de componentes del grafo.

1: **para todo** $0 \leq i \leq n$ **hacer**
2: $p(i) \leftarrow \infty$.
3: $M \leftarrow \{1, \ldots, n\}, c \leftarrow 0$.
4: **mientras** VACÍO(M) = falso **hacer**
5: $c \leftarrow c + 1, k \leftarrow$ mín(M), $p(k) \leftarrow 0$.
6: METER(v_k, Q). {Si la búsqueda es en profundidad, Q es una pila. En otro caso, es una cola.}
7: **mientras** $Q \neq \emptyset$ **hacer**
8: $w \leftarrow$SACAR(Q).
9: **para todo** $\{v_i, w\} \in E$ tal que $p(i) = \infty$ **hacer**
10: METER(v_i, Q), $M \leftarrow M \backslash \{i\}$, $p(i) \leftarrow p(w)+1$.

Algoritmo 12.7. Halla vértices de separación y puentes.

Entrada: Un grafo $G = (V, E)$.
Salida: Un arreglo s tal que $s_u = 1 \iff u$ es de separación y p tal que $p_{uv} = 1 \iff uv$ es un puente.

1: **para todo** $u \in V$ **hacer**
2: $c_u \leftarrow W, \pi_u \leftarrow \infty, a_u \leftarrow 0, s_u \leftarrow 0$.
3: **para todo** $uv \in E$ **hacer**
4: $p_{uv} \leftarrow 0$.
5: $t \leftarrow 0$.
6: **para todo** $u \in V$ **hacer**
7: **si** $c_u = W$ **entonces**
8: BPP(u).
9: **si** $a_u \leq 1$ **entonces**
10: $s_u = 0$

Algoritmo 12.8. Búsqueda a primera profundidad para el algoritmo anterior.

1: **función** BBP(u)
2: $c_u \leftarrow G, t \leftarrow t + 1, d_u \leftarrow L_u \leftarrow t$.
3: **para todo** $uv \in E \wedge v \neq \pi_u$ **hacer**
4: **si** $c_v = W$ **entonces**
5: $\pi_v \leftarrow v, a_u \leftarrow a_u + 1$.
6: BPP(v).
7: $L_u \leftarrow$ mín(L_u, L_v).
8: **si** $L_v > d_u$ **entonces**
9: $p_{uv} \leftarrow 1$.
10: **si** $L_v \geq d_u$ **entonces**
11: $s_u \leftarrow 1$.
12: **si no si** $d_v < d_u$ **entonces**
13: $L_u \leftarrow$ mín(L_u, d_v).
14: $c_u \leftarrow B, t \leftarrow t + 1, f_u \leftarrow t$.
15: **fin funcion**

12.3. Caminos mínimos

En esta sección, $G(E, V, W)$ es un grafo con el conjunto de aristas E y el conjunto de vértices V, con una función de peso $W : E \to \mathbb{R}$. La arista que conecta a los vértices $u, v \in V$ se escribe como $e_{uv} \in E$.

12.3.1. Con una sola fuente

Se desea encontrar todas las distancias $d(u)$ de cualquier vértice a otro distinguido s.

Algoritmo 12.9 (Inicialización). Establece el estado inicial de conocimiento de distancias y rutas más cortas.

1: **función** INICIALIZA(s)
2: $d(s) \leftarrow 0, \pi(s) \leftarrow 0$.
3: **para todo** $v \in V$ con $v \neq s$ **hacer**
4: $\quad d(v) \leftarrow \infty, \pi(v) \leftarrow$ NULL.

La arista e_{uv} se dice *tensa* si $d(u) + W(e_{uv}) < d(v)$.

Algoritmo 12.10 (Relajación). Si se encuentra una arista tensa, se *relaja* con la siguiente función.

1: **función** RELAJA(e_{uv})
2: **si** $d(v) > d(u) + w(e_{uv})$ **entonces**
3: $\quad d(v) \leftarrow d(u) + w(e_{uv}), \pi(v) \leftarrow u$.
4: $\quad$ **devolver** verdadero.
5: **si no**
6: $\quad$ **devolver** falso.

Algoritmo 12.11 (Dijkstra). Aquí B una cola de prioridad de los vértices con prioridad d sobre el mínimo. Siendo así, el tiempo de ejecución del algoritmo es $O(|E| + |V| \log |V|)$.

Entrada: Un vértice $s \in V(G)$ como fuente.
Salida: Una función de distancias mínimas d y de predecesores π.

1: **función** DIJKSTRA($G(E, V, W), d, \pi, s$)
2: INICIALIZA(S)
3: METER(B, s).
4: **mientras** $B \neq \emptyset$ **hacer**
5: $\quad u =$SACAR(B).
6: $\quad$ **para todo** $e_{uv} \in E(G)$ **hacer**
7: $\quad\quad$ **si** RELAJA(e_{uv}) **entonces**
8: $\quad\quad\quad$ METER(B, v).

Algoritmo 12.12 (Moore-Bellman-Ford). Encuentra los caminos mínimos con una sola fuente en un grafo pesado $G(E, V, w)$ en tiempo $O(|E||V|)$

1: **función** BELLMANFORD($G(E, V, W), d, \pi, s$).
2: INICIALIZA(s).
3: **para** $i \leftarrow 1$ hasta $|V|$ **hacer**
4: $\quad$ **para todo** $e_{uv} \in E(G)$ **hacer**
5: $\quad\quad$ RELAJA(e_{uv});
6: **para todo** $e_{uv} \in E(G)$ **hacer**
7: $\quad$ **si** $d(v) > d(u) + w(e_{uv})$ **entonces**
8: $\quad\quad$ **devolver** falso.
9: **devolver** verdadero.

12.3.2. Entre todos los pares

Algoritmo 12.13 (Johnson). Este algoritmo corre en tiempo $O(|V||E| + |V|^2 \log |V|)$.

Salida: Una función de distancias mínimas d_v y de predecesores π_v para $v \in V(G)$.

1: $V = V \cup \{s\}, E' = E, W' = W$
2: **para todo** $v \in V$ **hacer**
3: $\quad E' = E' \cup \{e_{sv}, e_{vs}\}$
4: $\quad W(e_{sv}) \leftarrow 0, W(e_{vs}) \leftarrow \infty$.
5: **si** BELLMANFORD($G(E', V', W'), d_s, \pi_s, s$) es falso **entonces**
6: $\quad$ **devolver** "Hay un ciclo negativo".
7: **para todo** $e_{uv} \in E$ **hacer**
8: $\quad W''(e_{uv}) \leftarrow d_s(u) + w(e_{uv}) - d_s(v)$.
9: **para todo** $v \in V$ **hacer**
10: $\quad$ DIJKSTRA($G(E, V, W''), d_v, \pi_v, v$).
11: **para todo** $u \in V$ **hacer**
12: $\quad d_v(u) \leftarrow d_s(v) + d_v(u) - d_s(u)$.

Algoritmo 12.14 (Floyd). Encuentra las distancias mínimas entre todos los vértices de un grafo pesado $G = (V, E, w)$ en tiempo $O(|V|^3)$

Entrada: Matriz de pesos w de un grafo con n vértices.
Salida: Matriz de distancias mínimas d y de caminos p.

1: **para** $i \leftarrow 0$ hasta $n - 1$ **hacer**
2: $\quad$ **para** $j \leftarrow 0$ hasta $n - 1$ **hacer**
3: $\quad\quad d_{i,j} \leftarrow w_{i,j}, p_{i,j} \leftarrow i$ {Inicialización}
4: **para** $i \leftarrow 0$ hasta $n - 1$ **hacer**
5: $\quad d_{i,i} \leftarrow 0$.
6: **para** $k \leftarrow 0$ hasta $n - 1$ **hacer**
7: $\quad$ **para** $i \leftarrow 0$ hasta $n - 1$ **hacer**
8: $\quad\quad$ **para** $j \leftarrow 0$ hasta $n - 1$ **hacer**
9: $\quad\quad\quad d_{i,j} \leftarrow \min(d_{i,j}, d_{i,k} + d_{k,j}), p_{i,j} \leftarrow p_{k,j}$.

12.4. Ordenamiento topológico

Algoritmo 12.15. Algoritmo de ordenamiento topológico. Corre en tiempo $O(|V|+|E|)$.

Entrada: Un grafo dirigido acíclico $G = (V, E)$.
Salida: Un ordenamiento lineal de V tal que $y \succ x$ si, sólo si, existe un camino dirigido de x hacia y.

1: Meter todos los vértices cuya valencia de entrada es cero en una cola Q.
2: **mientras** $Q \neq \emptyset$ **hacer**
3: $\quad$ Sacar a n de Q.
4: $\quad$ **devolver** n.
5: $\quad$ **para todo** vértice m con una arista e_{nm} **hacer**
6: $\quad\quad$ Quitar a e del grafo.
7: $\quad\quad$ **si** m no tiene otras aristas incidentes **entonces**
8: $\quad\quad\quad$ Meter m a Q.

12.5. Mínimo árbol generador

Algoritmo 12.16 (Kruskal). Encuentra en tiempo $O(|E| \log |V|)$ el mínimo árbol generador de un grafo co-

nexo pesado no trivial G

Salida: El mínimo árbol generador F.

1: Ordenar E por peso y hacer $F \leftarrow \emptyset$.
2: **para todo** $v \in V$ **hacer**
3: HazConjunto(v).
4: **para** $i \leftarrow 1$ hasta $|E|$ **hacer**
5: $\{u, v\} \leftarrow e_i$.
6: **si** Halla(u) $\neq$ Halla(v) **entonces**
7: Unión(u, v), $F \leftarrow F \cup \{u, v\}$.

12.6. Grafos hamiltonianos

Algoritmo 12.17. Para determinar un ciclo hamiltoniano de bajo peso en un grafo que satisface la desigualdad del triángulo.

Entrada: Un grafo completo pesado K de orden $p \geq 3$.

Salida: Un ciclo hamiltoniano de peso menor que el doble del ciclo hamiltoniano mínimo.

1: Encontrar el mínimo árbol generador T de G.
2: Recorrer T en una búsqueda a primero profundidad obteniendo la secuencia de vértices $v_{i_1}, \ldots, v_{i_p}$.
3: **devolver** $v_{i_1}, \ldots, v_{i_p}, v_{i_1}$.

Teorema 12.3 (Dirac). *Sea G un grafo de orden $p \geq 3$. Si $\operatorname{grad} v \geq p/2$ para cada $v \in V(G)$, entonces G es hamiltoniano.*

12.7. Flujo máximo

Algoritmo 12.18 (Ford-Fulkerson). Este algoritmo sólo funciona si las capacidades de las aristas son racionales. Si Q es una cola, entonces el algoritmo corre en tiempo $O(|V||E|(|V| + |E|))$.

Entrada: Una red $G(E, V, C)$, una fuente s, un destino t y un flujo factible f.

Salida: Un flujo máximo f.

1: Etiquetar la fuente con $(\emptyset, \emptyset, \epsilon(s) = \infty)$.
2: Meter(Q, s).
3: **mientras** $Q \neq \emptyset$ **hacer**
4: Tomar $u =$ Sacar(Q) con etiqueta $(w, \pm, \epsilon(u))$.
5: **para todo** $e_{uv} \in E(G)$ con v no etiquetado y tal que $f(e_{uv}) < c(e_{uv})$ **hacer**
6: Etiquetar a v con $(u, +, \min(\epsilon(u), c(e_{uv}) - f(e_{uv})))$.
7: Meter(Q, v).
8: **para todo** $e_{vu} \in E(G)$ con v no etiquetado y tal que $f(e_{uv}) > 0$ **hacer**
9: Etiquetar a v con $(u, -, \min(\epsilon(u), f(e_{uv})))$.
10: Meter(Q, v).
11: **si** t está etiquetado **entonces**
12: $\epsilon \leftarrow \epsilon(t)$.
13: **mientras** $t \neq s$ **hacer**
14: **si** la etiqueta de t es $(u, +, \epsilon(t))$ **entonces**
15: $f(e_{u,t}) \leftarrow f(e_{u,t}) + \epsilon$
16: **si no si** la etiqueta de t es $(u, -, \epsilon(t))$ **entonces**
17: $f(e_{t,u}) \leftarrow f(e_{t,u}) - \epsilon$
18: $t \leftarrow u$.
19: Vaciar(Q) y Meter(Q, s).

13. Localización de patrones

13.1. Algoritmo KMP

Algoritmo 13.1 (Knuth-Morris-Pratt). Localiza una cadena en otra.

Entrada: Dos cadenas T y S, con $|T| = n$ y $|P| = m$.

Salida: Una posición k tal que $T_k \cdots T_{k+n-1} = P$, o falso si no existe.

1: $j \leftarrow 1$.
2: **para** $i \leftarrow 1$ hasta n **hacer**
3: **mientras** $j > 0$ y $T_i \neq P_j$ **hacer**
4: $j = \Phi(j)$.
5: **si** $j = m$ **entonces**
6: **devolver** $i - m + 1$
7: $j \leftarrow j + 1$.
8: **devolver** falso.

Algoritmo 13.2. Calcula la función Φ requerida en el algoritmo anterior.

Entrada: Una cadena P con $|P| = m$

Salida: La función Φ.

1: $j \leftarrow 0$.
2: **para** $i \leftarrow 1$ hasta m **hacer**
3: **si** $P_i = P_j$ **entonces**
4: $\Phi(i) \leftarrow \Phi(j)$.
5: **si no**
6: $\Phi(i) \leftarrow j$.
7: **mientras** $j > 0$ y $P_i \neq P_j$ **hacer**
8: $j \leftarrow \Phi(j)$.
9: $j \leftarrow j + 1$.

13.2. Detección de periodicidades

El *borde* de una cadena $C = c_0 \ldots c_{n-1}$ es la subcadena propia de longitud máxima $B = c_0 \ldots c_{b-1}$ de C tal que $B = c_{n-b} \ldots c_{n-1}$.

Algoritmo 13.3. Cálculo en tiempo $O(n)$ de las longitudes de los bordes de los prefijos de la cadena $C = c_0 \ldots c_{n-1}$.

Entrada: Una cadena C con $|C| = n$

Salida: Un arreglo de las longitudes de los bordes B tal que B_i es la longitud del borde de la subcadena $c_0 \ldots c_i$.

1: $B_0 \leftarrow 0$.
2: **para** $i \leftarrow 0$ hasta $n-1$ **hacer**
3: $b \leftarrow B_i$.
4: **mientras** $b > 0$ y $C_{i+1} \neq C_b$ **hacer**
5: $b \leftarrow B_{b+1}$.
6: **si** $C_{i+1} = C_b$ **entonces**
7: $B_{i+1} \leftarrow b+1$.
8: **si no**
9: $B_{i+1} \leftarrow 0$.

Sea b la cardinalidad del borde B de C. Entonces C consta de $\lfloor \frac{n}{p} \rfloor$ cadenas iguales de longitud p, donde $n = |C|$ y $p = n - b$, y p es el menor número con esa propiedad. Si $p \backslash n$, entonces la cadena es periódica con periodo p.

14. Cuestiones misceláneas

double modf(double d, double *p) Devuelve la parte fraccionaria de d, coloca la parte entera en *p.

std::atoi(cadena.c_str()) Convertir una cadena a un numero.

string.erase(posicion,nelementos) Borrar un subconjunto de la cadena.

string.replace(posicion,nelementos,subcadena) Reemplaza un subconjunto de una cadena.

string::size_type s1.find(s2) Devuelve la posición de la cadena s2 en s1. Si falla devuelve **npos**. Si **npos** está ocupado, se lanza un **range_error**.

string.substr(posicion,nelementos) Extraer un subconjunto de una cadena.

void erase(iterator pos) Elimina el elemento apuntado por pos en un conjunto.

size_type erase(const key_type& x) Elimina el elemento x de un conjunto.

Aritmética de precisión arbitraria implementada con cadenas.

```
string convertir(long long n) {
 string c("");
 do {
  c += (char)(n%10+'0'); n /= 10;
  }while(n);
 reverse(c.begin(),c.end());
 return c; }

string borrar_ceros(string a) {
 int i=0;
 while(a[i]=='0'&&i<a.size()-1) i++;
 return a.substr(i,a.size()-i); }

bool menor(string a, string b) {
 a=borrar_ceros(a); b=borrar_ceros(b);
 if(a.size()<b.size()) return true;
 else if(a.size()>b.size())
  return false;
 else return a<b; }

string suma(string a, string b) {
 string ans(""); int k=0;
 if(a.size()<b.size()) swap(a,b);
 int j = a.size()-1, i = b.size()-1;
 for(; j>=0; j--,i--){
   int u = a[j]-'0';
   if(i>=0){
    ans += (u+(b[i]-'0')+k)%10+'0';
    k = (u+(b[i]-'0')+k)>=10; }
 /* ans += (u-(b[i]-'0')+k+10)%10+'0';
    k = 0-((u-(b[i]-'0')+k)<0);} */
   else{
    ans += (u+k)%10+'0';
    k = (u+k)>=10; } }
 /* ans += (u+k+10)%10+'0';
    k = 0-((u+k)<0); } } */
 if(k) ans += '1';
 reverse(ans.begin(),ans.end());
 return ans; }

string mult(string a, string b){
 int n = a.size(), m = b.size();
 int t,k,i; string ans(m+n,'0');
 for(int j = m; j>0; j--){
   for(i = n,k=0; i>0; i--){
    t = ((a[i-1]-'0')*(b[j-1]-'0'));
    t += (ans[i+j-1]-'0')+k;
    ans[i+j-1] = (t%10)+'0';
    k = t/10;}
   ans[j-1]=k+'0'; }
 return borrar_ceros(ans); }

string divide_d(string a, int d) {
 string temp("");
 int N,i,res=0; N = a.size();
 temp+=((a[0]-'0')/d)+'0';
 res = ((int)(a[0]-'0'))%d;
 for(i=1;i<N;i++) {
   res =(res*10)+(a[i]-'0');
   temp +=(res/d+'0'); res = res%d; }
 return borrar_ceros(temp); }

string divide(string u, string v) {
 string d(""),ans(""),parcial("");
 string temp1(""),temp2("");
 vector <string> mul;
```

```
 mul.clear();
 if(v.size()==1)
  return divide_d(u,v[0]-'0');
 int m,q=0,a1,a2,a3,a4,a5,j,inc;
 d += (10/((v[0]-'0')+1)+'0');
 u = mult(d,u); v = mult(d,v);
 u.insert(u.begin(),'0'); j = 0;
 mul.push_back("0"); mul.push_back(v);
 for(int k=2;k<10;k++)
   mul.push_back(suma(mul[k-1],v));
 m = u.size()-v.size()-1;
 while(j<=m)
  {
   a1 = u[j]-'0'; a2 = u[j+1]-'0';
   a3 = u[j+2]-'0'; a4 = v[0]-'0';
   a5 = v[1]-'0';
   if(a1==a4) q = 9;
   else q = (a1*10+a2)/a4;
   while(q*a5>((a1*10+a2-q*a4)*10+a3))
    q--;
   parcial.erase();
   for(int l=j;l<j+v.size()+1;l++)
    parcial += u[l];
   if(menor(parcial,mul[q])) q--;
   temp2 = resta(parcial,mul[q]);
   for(int l=j;l<j+v.size()+1;l++)
    u[l] = temp2[l-j];
   ans += (char)(q+'0'); j++;
  }
 return borrar_ceros(ans); }
```

Extracción de datos listados en una sola fila cuando no se especifica su número.

```
cin.getline(conjuntos, 1000);
ptr = strtok(conjuntos," ");
while(ptr!=NULL) {
 numero=atoi(ptr); B.insert(numero);
 ptr = strtok(NULL, " "); }
```

Leer datos cuando no especifican cuántos son.

```
#include <sstream>
int i; double f; string cad,temp;
// Delimitador del '.'
// getline(cin, cad,'.');

//Para leer muchas cadenas
getline(cin, cad);

istringstream in(cad);
// Pero como esta esta cosa
// lee primero un entero.
in >>i; cout<<i*10<<endl;

//Lee cadenas y las pasa a temp.
while(in >> temp)
cout << temp << endl;
```