Lógica

Universidad
Internacional
de Valencia

Máster Universitario
en Inteligencia Artificial

02MIAR | Matemáticas:
Matemáticas para la Inteligencia Artificial

Profesor:
Amílcar J. Pérez A.

De:
Planeta Formación y Universidades

Definición

Una **permutación sin repetición** *de un conjunto A de n elementos diferentes consiste en cualquier ordenación que se pueda hacer con éstos, teniendo en cuenta que el orden de secuenciación importa.*
El número de ordenaciones posible viene dado por la expresión

$$P_n = n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1 = n!.$$

Ejemplo

Sea $A = \{a, b, c, d\}$. Queremos cuántas permutaciones pueden realizarse con los elementos de A (por ejemplo, abcd, bdca y dabc son tres de ellas). Como $|A| = 4$, se tiene por la expresión anterior, tomando $n = 4$, que $P_4 = 4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$.

Definición

Dados n elementos de r tipos diferentes, con n_i el número de elementos de tipo i, $1 \leq i \leq r$, una **permutación** *con repetición consiste en una reordenación cualquiera de estos elementos, donde el el total de posibilidades viene dado por*

$$P_n^{n_1,n_2,\ldots,n_r} = \frac{n!}{n_1! \cdot n_2! \cdots n_r!}.$$

Ejemplo

Reordenaciones posibles con las letras de la palabra aabbbc. $r = 3$ elementos diferentes: las letras a, b y c. $n_1 = 2$. $n_2 = 3$. $n_3 = 1$. Por tanto, el resultado es $P_6^{2,3,1} = \frac{6!}{2! \cdot 3! \cdot 1!} = 60$.

Definición

Una **variación sin repetición** *consiste en tomar k elementos dentro de un conjunto de n elementos y contar las posibles elecciones, teniendo en cuenta que no pueden elegirse más de una vez un mismo elemento y que, a diferencia de las combinaciones, el orden de elección importa. El número de posibilidades es, en este caso,*

$$V_{n,k} = n \cdot (n-1) \cdot (n-2) \cdots (n-k+1) = \frac{n!}{(n-k)!}.$$

Ejemplo

Posibles números de dos cifras que pueden formarse con el conjunto de cifras $\{1, 2, 3, 4\}$*:* $V_{4,2} = \frac{4!}{2!} = 12$.

Definición

Una **variación con repetición** *consiste en tomar k elementos dentro de un conjunto de n elementos y contar las posibles elecciones, teniendo en cuenta que, en este caso, puede elegirse más de una vez un mismo elemento y que el orden de elección importa. El número de posibilidades es, en este caso,*

$$VR_{n,k} = \overbrace{n \cdot n \cdots n}^{k \text{ veces}} = n^k.$$

Ejemplo

Posibles números de dos cifras que pueden formarse con el conjunto de cifras $\{1, 2, 3, 4\}$*, pudiendo utilizarse cada una más de una vez:* $VR_{4,2} = 4^2 = 16$.

Definición

Una **combinación sin repetición** *consiste en tomar k elementos dentro de un conjunto de n elementos y contar las posibles elecciones, teniendo en cuenta que no se puede elegir más de una vez un mismo elemento y que, además, el orden de elección no importa. En este caso, el número de posibilidades es*

$$C_{n,k} = \binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}.$$

Ejemplo

Posibles grupos de 3 *letras dentro del conjunto de* 4 *letras* $\{a, b, c, d\}$*:*
$C_{4,3} = \binom{4}{3} = \frac{4!}{3! \cdot (4-3)!} = 4.$

Definición

Una **combinación con repetición** *consiste en tomar k elementos dentro de un conjunto de n elementos y contar las posibles elecciones, teniendo en cuenta que, en este caso, puede elegirse más de una vez un mismo elemento y que, de nuevo, el orden de elección no importa. En este caso, el número de posibilidades es*

$$CR_{n,k} = \binom{n+k-1}{k} = \frac{(n+k-1)!}{k! \cdot (n-1)!}.$$

Ejemplo

Posibles grupos de 3 *letras dentro del conjunto de* 4 *letras* $\{a, b, c, d\}$*, admitiendo repetición:* $CR_{4,3} = \binom{4+3-1}{3} = \binom{6}{3} = \frac{6!}{3! \cdot (6-3)!} = 20.$

Funciones booleanas

- Recordemos que $p \to q \equiv \overline{p} + q$.
- $p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$, luego $p \leftrightarrow q \equiv (\overline{p} + q) \cdot (p + \overline{q})$, por lo que simplificando $p \leftrightarrow q \equiv p \cdot q + \overline{p} \cdot \overline{q}$.
- Cualquier fórmula lógica puede expresarse empleando como únicos conectores la conjunción (producto), la disyunción (suma) y la negación (**álgebras de Boole**).

Definición

Una **función booleana** *es una aplicación* $f : \mathbb{B}^n \to \mathbb{B}$, *donde* $\mathbb{B} = \{0, 1\}$ *es el* **conjunto binario**. *Se denota* $\mathcal{F}_n$ *al conjunto formado por todas las funciones booleanas de n variables, es decir:*

$$\mathcal{F}_n = \{f : \mathbb{B}^n \to \mathbb{B} \mid f \text{ aplicación}\}.$$

Ejemplos

1. $f : \mathbb{B} \to \mathbb{B}$, $f(x) = x$.
2. $f : \mathbb{B} \to \mathbb{B}$, $f(x) = \overline{x}$.
3. $f : \mathbb{B}^2 \to \mathbb{B}$, $f(x, y) = x + y$.
4. $f : \mathbb{B}^2 \to \mathbb{B}$, $f(x, y) = x \cdot y$.
5. $f : \mathbb{B}^2 \to \mathbb{B}$, $f(x, y) = \overline{x} + y$.
6. $f : \mathbb{B}^2 \to \mathbb{B}$, $f(x, y) = x \cdot y + \overline{x} \cdot \overline{y}$.
7. $f : \mathbb{B}^3 \to \mathbb{B}$, $f(x, y, z) = \overline{x} \cdot z + y$.

Puertas lógicas

Definición

Una **puerta lógica** *es un operador que transforma uno o dos valores de entrada binarios,* x, y*, en un valor binario de salida, en función de* x, y*. Por tanto, una puerta lógica puede verse como elemento de* $\mathcal{F}_n$*, con* $n = 1$ *o* $n = 2$*.*

Existen un total de siete puertas lógicas:

- OR
- AND
- XOR
- NOT
- NOR
- NAND
- XNOR

Puertas lógicas

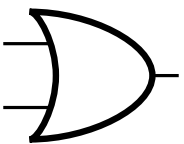
Puerta OR: se trata de la puerta lógica que combina un par de valores $x, y \in \mathbb{B}$ según la tabla de operaciones siguiente:

OR	$y = 0$	$y = 1$
$x = 0$	0	1
$x = 1$	1	1

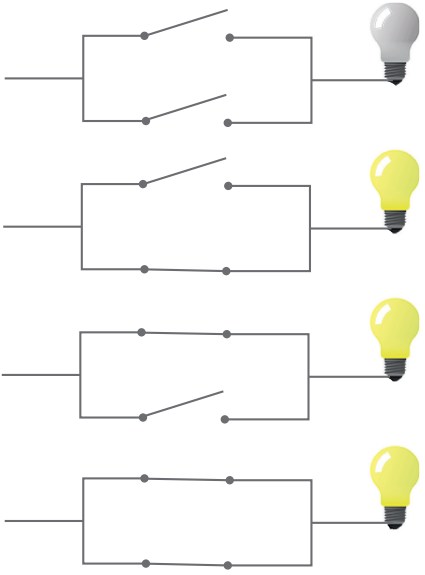
Esta puerta lógica puede interpretarse como la siguiente función booleana de $\mathcal{F}_2$:

$$f : \mathbb{B}^2 \to \mathbb{B}, \quad f(x, y) = x + y.$$

Se representa mediante el siguiente símbolo:



Puertas lógicas



Puertas lógicas

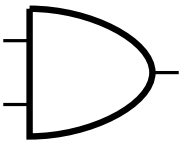
Puerta AND: se trata de la puerta lógica que combina un par de valores $x, y \in \mathbb{B}$ según la tabla de operaciones siguiente:

AND	$y = 0$	$y = 1$
$x = 0$	0	0
$x = 1$	0	1

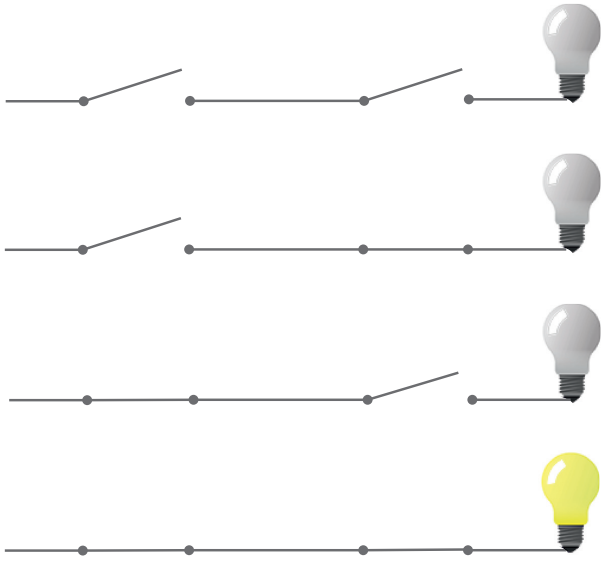
Esta puerta lógica puede interpretarse como la siguiente función booleana de $\mathcal{F}_2$:

$$f : \mathbb{B}^2 \to \mathbb{B}, \quad f(x, y) = x \cdot y.$$

Se representa mediante el siguiente símbolo:



Puertas lógicas



Puertas lógicas

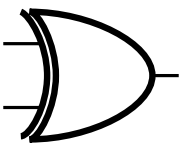
Puerta XOR: se trata de la puerta lógica que combina un par de valores $x, y \in \mathbb{B}$ según la tabla de operaciones siguiente:

XOR	$y = 0$	$y = 1$
$x = 0$	0	1
$x = 1$	1	0

Esta puerta lógica puede interpretarse como la siguiente función booleana de $\mathcal{F}_2$:

$$f : \mathbb{B}^2 \to \mathbb{B}, \quad f(x, y) = (\overline{x} + \overline{y}) \cdot (x + y).$$

Se representa mediante el siguiente símbolo:



Puertas lógicas

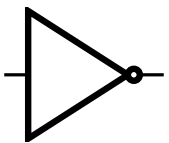
Puerta NOT: se trata de la puerta lógica que invierte el estado de un valor binario $x \in \mathbb{B}$:

NOT	
$x = 0$	1
$x = 1$	0

Esta puerta lógica puede interpretarse como la siguiente función booleana de $\mathcal{F}_1$:

$$f : \mathbb{B} \to \mathbb{B}, \quad f(x) = \overline{x}.$$

Se representa mediante el siguiente símbolo:



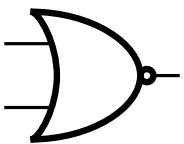
Puerta NOR: se trata de la puerta lógica que combina un par de valores $x, y \in \mathbb{B}$ según la tabla de operaciones siguiente:

NOR	$y = 0$	$y = 1$
$x = 0$	1	0
$x = 1$	0	0

Esta puerta lógica puede interpretarse como la siguiente función booleana de $\mathcal{F}_2$:

$$f : \mathbb{B}^2 \to \mathbb{B}, \quad f(x, y) = \overline{x} \cdot \overline{y}.$$

Se representa mediante el siguiente símbolo:



Puertas lógicas

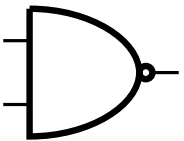
Puerta NAND: se trata de la puerta lógica que combina un par de valores $x, y \in \mathbb{B}$ según la tabla de operaciones siguiente:

NAND	$y = 0$	$y = 1$
$x = 0$	1	1
$x = 1$	1	0

Esta puerta lógica puede interpretarse como la siguiente función booleana de $\mathcal{F}_2$:

$$f : \mathbb{B}^2 \to \mathbb{B}, \quad f(x, y) = \overline{x} + \overline{y}.$$

Se representa mediante el siguiente símbolo:



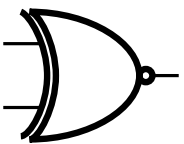
Puerta XNOR: se trata de la puerta lógica que combina un par de valores $x, y \in \mathbb{B}$ según la tabla de operaciones siguiente:

XNOR	$y = 0$	$y = 1$
$x = 0$	1	0
$x = 1$	0	1

Esta puerta lógica puede interpretarse como la siguiente función booleana de $\mathcal{F}_2$:

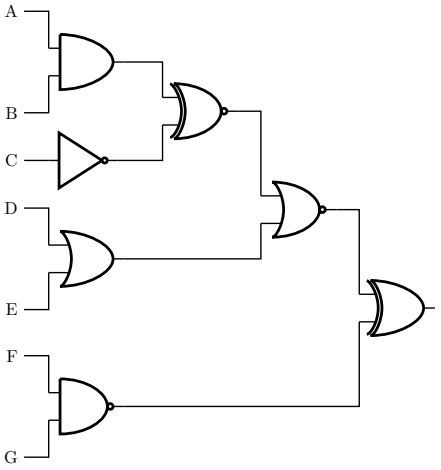
$$f : \mathbb{B}^2 \to \mathbb{B}, \quad f(x, y) = x \cdot y + \overline{x} \cdot \overline{y}.$$

Se representa mediante el siguiente símbolo:



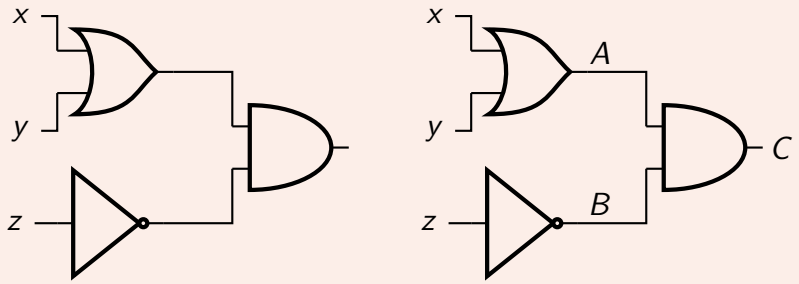
Circuitos lógicos

Las puertas lógicas pueden combinarse entre sí para formar **circuitos lógicos**, los cuales pueden interpretarse como elementos de $\mathcal{F}_n$.



Ejemplo

Consideremos el circuito lógico siguiente:



La función booleana asociada a este circuito es $f \in \mathcal{F}_3$ *dada por*

$$f(x, y, z) = (x + y) \cdot \overline{z}.$$

Redes Neuronales

Definición

Un **grafo** G *es una terna* (V, E, γ)*, donde* V *y* E *son conjuntos y* γ *es una aplicación*

$$\gamma : E \to \mathcal{E},$$

donde $\mathcal{E} = \{\{v, v'\} \mid v, v' \in V\}$

- *Los elementos del conjunto* V *reciben el nombre de* **vértices**.
- *Los elementos del conjunto* E *reciben el nombre de* **lados** *o* **aristas**.
- *La aplicación* γ *se conoce como* **aplicación de incidencia**.
- *Dicha aplicación identifica las aristas con el par correspondiente de vértices que unen.*

Todo grafo tiene su correspondiente representación gráfica, como veremos en los siguientes ejemplos.

Ejemplos

1. *Consideremos el grafo* $G = (V, E, \gamma)$, *con* $V = \{v_1, v_2, v_3\}$, $E = \{e_1, e_2\}$, *siendo* $\gamma : E \to \mathcal{E}$ *dada por:*

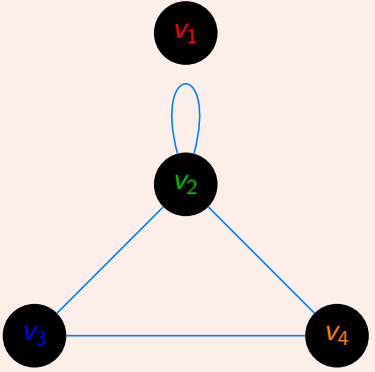
$\gamma(e_1) = \{v_1, v_2\}$

$\gamma(e_2) = \{v_2, v_3\}$

Ejemplos

2. *Consideremos el grafo* $G = (V, E, \gamma)$, *con* $V = \{v_1, v_2, v_3, v_4\}$, $E = \{e_1, e_2, e_3, e_4\}$, *siendo* $\gamma : E \to \mathcal{E}$ *dada por:*

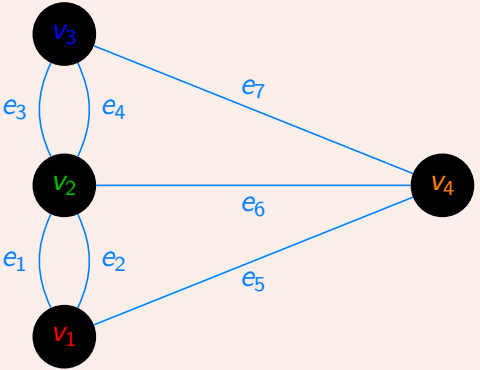
$\gamma(e_1) = \{v_2, v_2\}$
$\gamma(e_2) = \{v_2, v_3\}$
$\gamma(e_3) = \{v_3, v_4\}$
$\gamma(e_4) = \{v_4, v_2\}$



Ejemplos

3. *Consideremos el grafo* $G = (V, E, \gamma)$, *con* $V = \{v_1, v_2, v_3, v_4\}$, $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$, *siendo* $\gamma : E \to \mathcal{E}$ *dada por:*

$$\gamma(e_1) = \{v_1, v_2\}$$
$$\gamma(e_2) = \{v_1, v_2\}$$
$$\gamma(e_3) = \{v_2, v_3\}$$
$$\gamma(e_4) = \{v_2, v_3\}$$
$$\gamma(e_5) = \{v_1, v_4\}$$
$$\gamma(e_6) = \{v_2, v_4\}$$
$$\gamma(e_7) = \{v_3, v_4\}$$



Grafos dirigidos

Definición

Un **grafo dirigido** *G es una terna (V, E, γ), donde V y E son conjuntos (de vértices y de aristas o lados, respectivamente) y γ es una aplicación de la forma*

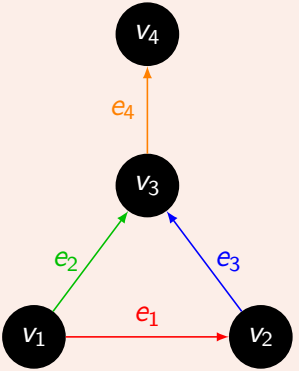
$$\gamma : E \to V \times V.$$

Los grafos dirigidos habitualmente se representan gráficamente dibujando las aristas con una flecha en la dirección válida para recorrerla, siendo la primera componente del par ordenado el punto de partida y la segunda componente el punto de llegada.

Ejemplo

Sea $G = (V, E, \gamma)$ *el grafo dirigido dado por el conjunto de vértices* $V = \{v_1, v_2, v_3, v_4\}$, *el conjunto de aristas* $E = \{e_1, e_2, e_3, e_4\}$ *y* $\gamma : E \to V \times V$ *dada por:*

$\gamma(e_1) = (v_1, v_2)$,
$\gamma(e_2) = (v_1, v_3)$,
$\gamma(e_3) = (v_2, v_3)$,
$\gamma(e_4) = (v_3, v_4)$.



Grafos simples

Definición

Un **grafo simple** $G = (V, E, \gamma)$ *es un grafo que cumple dos restricciones adicionales:*

1. *No hay aristas diferentes que unan el mismo par de vértices, es decir, si* $e \neq e'$, *entonces* $\gamma(e) \neq \gamma(e')$ *(*γ *es inyectiva).*
2. *No contiene* **bucles** *o* **loops**, *es decir, no tiene aristas que unen un vértices consigo mismo; equivalentemente,* $\gamma(e) = \{v_e, v'_e\}$, *con* $v_e \neq v'_e$, $\forall e \in E$.

Nota

En adelante, y por comodidad, en caso de trabajar con grafos simples, omitiremos la existencia de la aplicación γ *y denotaremos las aristas directamente por el correspondiente conjunto o par ordenado de vértices.*

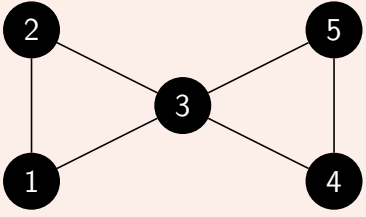
Grafos conexos

Definición

Un grafo G es **conexo** *si para cada par de vértices diferentes existe algún* **camino** *(sucesión de aristas adyacentes) que los une.*

Ejemplos

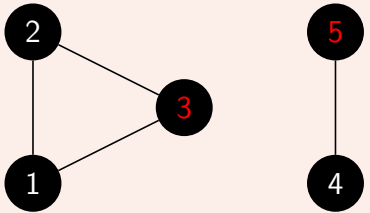
1. *Sea G un grafo de* 5 *vértices dado por las aristas* $\{1,2\},\{1,3\},\{2,3\},\{3,4\},\{3,5\},\{4,5\}$. *Gráficamente:*



G es conexo.

Ejemplos

2. *Sea G un grafo de 5 vértices cuyas aristas son $\{1,2\},\{1,3\},\{2,3\},\{4,5\}$. Gráficamente:*



Entonces G no es conexo.

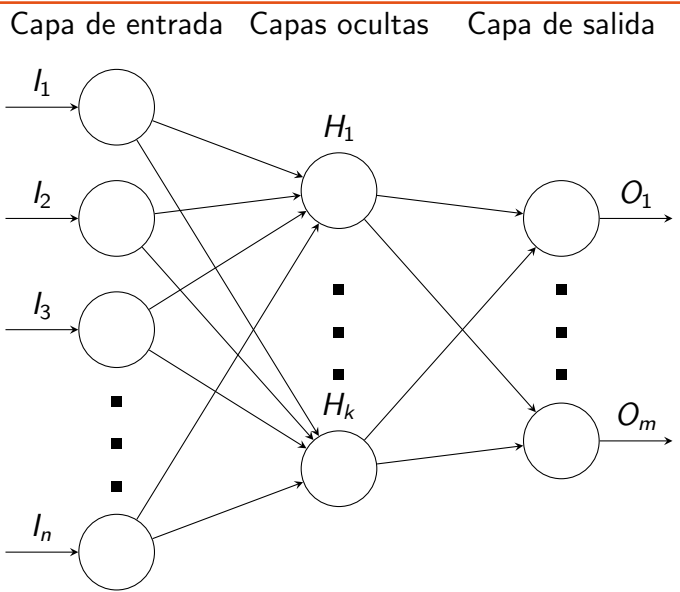
Definición

Una **red neuronal** *es un grafo dirigido* (V, E)*, con* $V = \{1, 2, \ldots, n\}$ *y* $E \subseteq V \times V$ *con los elementos adicionales siguientes:*

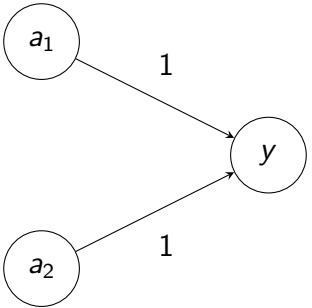
- $\forall i \in V$ *tiene asociada una variable de estado* $a_i \in [0, 1]$*.*
- $\forall (i, j) \in E$ *tiene asociado un peso* $\omega_{i,j} \in \mathbb{R}$ *(***grafo ponderado***).*
- $\forall i \in V$ *tiene asociado un* **sesgo** b_i*.*
- $\forall i \in V$ *le corresponde una* **función de activación** $f_i : \mathbb{R} \to [0, 1]$ *de la forma*

$$f_i\left(\sum_{j=1}^{n} \omega_{j,i} a_j + b_i\right).$$

Redes neuronales

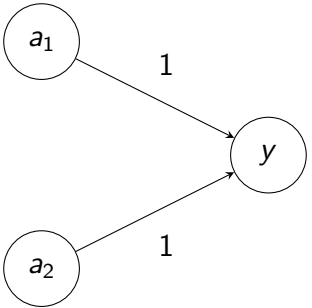


Ejemplo 1: neurona AND



$$y = f(a_1 + a_2 - 1.5), \quad f(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$$

Ejemplo 2: neurona OR



$$y = f(a_1 + a_2 - 0.5), \quad f(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$$

Complejidad computacional

La teoría de **complejidad computacional** estudia y clasifica los algoritmos y problemas computacionales en función de su coste y dificultad de resolver.

Algorithm Compute $p = u \cdot v$ for $u, v \in \mathbb{R}^n$

Require: $\text{length}(u) = \text{length}(v)$
Ensure: $p = u \cdot v$
  $p \leftarrow u_1 v_1$
  for $k \leftarrow 2$ to n **do**
    $p \leftarrow p + u_k v_k$
  end for
  return p

- **Operaciones:** $n - 1$ sumas, n productos.
- **Total operaciones:** $2n - 1$ operaciones (coste **lineal**).

Complejidad computacional

Algorithm Compute $C = A + B$ for $A, B \in \mathbb{R}^{n \times n}$

Require: $\text{size}(A) = \text{size}(B)$

Ensure: $C = A + B$

```
C ← 0_{n×n}
for i ← 1 to n do
    for j ← 1 to n do
        C_{i,j} ← A_{i,j} + B_{i,j}
    end for
end for
return C
```

- **Operaciones:** n^2 sumas.
- **Total operaciones:** n^2 operaciones (coste **cuadrático**).

Complejidad computacional

Algorithm Compute $C = A \cdot B$ for $A, B \in \mathbb{R}^{n \times n}$

Require: size(A) = size(B)

Ensure: $C = A \cdot B$

```
C ← 0_{n×n}
for i ← 1 to n do
    for j ← 1 to n do
        for k ← 1 to n do
            C_{i,j} ← C_{i,j} + A_{i,k} B_{k,j}
        end for
    end for
end for
return C
```

- **Operaciones:** n^3 sumas, n^3 productos.
- **Total operaciones:** $2n^3$ operaciones (coste **cúbico**).

Complejidad computacional

Universidad Internacional de Valencia

Resolución de un sistema de ecuaciones lineal $n \times n$ con matriz de coeficientes triangular inferior:

$$\left.\begin{array}{rcl} a_{11}x_1 & & = b_1 \\ a_{21}x_1 + a_{22}x_2 & & = b_2 \\ & \ddots & \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n & & = b_n \end{array}\right\}$$

Matricialmente: $AX = B$, donde $A \in \mathbb{R}^{n\times n}$, $X \in \mathbb{R}^{n\times 1}$ y $B \in \mathbb{R}^{n\times 1}$:

$$A = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}, X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, B = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}.$$

Complejidad computacional

Universidad Internacional de Valencia

Algorithm Solve $AX = B$ for $A \in \mathbb{R}^{n \times n}$, $X, B \in \mathbb{R}^{n \times 1}$

Require: $\det(A) \neq 0$
Ensure: $AX = B$
  for $i \leftarrow 1$ to n **do**
    $X_i \leftarrow B_i$
    for $j \leftarrow 1$ to $i - 1$ **do**
      $X_i \leftarrow X_i - A_{i,j} X_j$
    end for
    $X_i \leftarrow \dfrac{X_i}{A_{i,i}}$
  end for
  return X

- **Operaciones:** $(n-1)n/2$ sumas, $(n-1)n/2$ productos, n divisiones.
- **Total operaciones:** n^2 operaciones (coste **cuadrático**).

Complejidad computacional

Universidad Internacional de Valencia

Definición

Diremos que el coste computacional de un algoritmo es **polinomial** *si el número de operaciones en función del tamaño de la entrada n, es un polinomio en n.*

Definición

Diremos que el coste computacional de un algoritmo con $c(n)$ operaciones es $\mathcal{O}(n^k)$, $k \in \mathbb{N}$, si

$$\exists \lim_{n \to +\infty} \frac{c(n)}{n^k} = C, \quad C \in (0, +\infty).$$

Ejemplos

1. *El coste computacional del producto escalar, con $2n-1$ operaciones, es $\mathcal{O}(n)$:*

$$\lim_{n \to +\infty} \frac{2n-1}{n} = 2.$$

Ejemplos

2. *El coste computacional de la suma de matrices cuadradas, con n^2 operaciones, es $\mathcal{O}(n^2)$:*
$$\lim_{n\to+\infty} \frac{n^2}{n^2} = 1.$$
3. *El coste computacional del producto de matrices cuadradas, con $2n^3$ operaciones, es $\mathcal{O}(n^3)$:*
$$\lim_{n\to+\infty} \frac{2n^3}{n^3} = 2.$$
4. *El coste computacional de la resolución de un sistema triangular, con n^2 operaciones, es $\mathcal{O}(n^2)$:*
$$\lim_{n\to+\infty} \frac{n^2}{n^2} = 2.$$

Complejidad computacional

Universidad Internacional de Valencia

Algorithm Compute $\det(A)$ for $A \in \mathbb{R}^{n \times n}$

Require: A is a square matrix
Ensure: $d = \det(A)$
  $n \leftarrow \dim(A)$
  if $n = 1$ **then**
    $d \leftarrow A_{1,1}$
  else
    $d \leftarrow 0$
    for $j \leftarrow 1$ to n **do**
      $d \leftarrow d + (-1)^{1+j} A_{1,j} \det(A_{\overline{\{1\}},\overline{\{j\}}})$
    end for
  end if
  return d

$2 \cdot n!$ operaciones.

Complejidad computacional

- Se tiene que para todo $k \in \mathbb{N}$ fijado:

$$\lim_{n\to+\infty} \frac{2 \cdot n!}{n^k} = \infty \quad \text{pues} \quad \frac{n!}{n^k} = \frac{n!}{n^k(n-k)!}(n-k)! < (n-k)!, \quad \text{si } n > k$$

- Necesitamos ampliar la noción de $\mathcal{O}$.

Definición

Sea $f : \mathbb{N} \to \mathbb{R}$ y $c(n)$ el número de operaciones requeridas por un algoritmo. Diremos que el coste computacional del algoritmo es $\mathcal{O}\big(f(n)\big)$ si

$$\lim_{n\to+\infty} \frac{c(n)}{f(n)} = C, \quad C \in]0, +\infty[.$$

Ejemplos

1. *El coste computacional del algoritmo que calcula el determinante es* $\mathcal{O}(n!)$*:*

$$\lim_{n\to+\infty} \frac{2 \cdot n!}{n!} = 2.$$

2. *El coste computacional de un algoritmo con* $2^n + n^6 - 2n^4 + 3n + 7$ *operaciones es* $\mathcal{O}(2^n)$*:*

$$\lim_{n\to+\infty} \frac{2^n + n^6 - 2n^4 + 3n + 7}{2^n} = 1.$$

3. *Algoritmos de ordenación eficientes:* $\mathcal{O}\big(n \log(n)\big)$.

¡Muchas gracias!

Universidad
Internacional
de Valencia

Contacto:

amilcar.perez@professor.universidadviu.com