Conceptos previos: funciones y algoritmos

Joaquim Borges
Robert Clarisó
Ramon Masià
Jaume Pujol
Josep Rifà
Joan Vancells
Mercè Villanueva

PID_00174685

UOC
Universitat Oberta de Catalunya

Los textos e imágenes publicados en esta obra están sujetos –excepto que se indique lo contrario– a una licencia de Reconocimiento-NoComercial-SinObraDerivada (BY-NC-ND) v.3.0 España de Creative Commons. Podéis copiarlos, distribuirlos y transmitirlos públicamente siempre que citéis el autor y la fuente (FUOC. Fundació per a la Universitat Oberta de Catalunya), no hagáis un uso comercial y no hagáis una obra derivada. La licencia completa se puede consultar en http://creativecommons.org/licenses/by-nc-nd/3.0/es/legalcode.es

Índice

Introducción

En este módulo se presentan un conjunto de conceptos básicos que necesitaremos en otros módulos de la asignatura. Son conceptos de matemática discreta que probablemente conozcáis de otras asignaturas. Hay, además, otros conceptos más sencillos que se supondrán conocidos: el concepto de conjunto, los razonamientos lógicos y las técnicas básicas de conteo.

En la primera parte del módulo repasaremos el concepto de función a partir de los conceptos básicos de producto cartesiano de dos conjuntos y el de relación. Así mismo, estudiaremos diferentes tipos de función y acabaremos describiendo los conjuntos finitos, numerables e infinitos. Precisaremos el concepto de contar y daremos su definición, que requiere determinar con exactitud la noción "el conjunto A tiene n elementos". Algunos de estos conceptos también los habréis estudiado previamente por ejemplo en las asignaturas de *Álgebra* o *Iniciación a las matemáticas para la ingeniería*.

En la segunda parte del módulo veremos una introducción a la complejidad computacional, concepto que posiblemente también habréis visto en la asignatura *Prácticas de programación*. Para empezar, introduciremos la idea intuitiva de cuándo un problema es fácil o es difícil, con la ayuda de ejemplos. En el caso de que el problema se pueda resolver mediante un algoritmo, el concepto de eficiencia del algoritmo traduce la idea de la dificultad para resolver el problema. Veremos cómo se puede estudiar la eficiencia de un algoritmo y que hay algoritmos (y problemas) que son intrínsecamente difíciles. Finalmente, veremos el concepto de complejidad algorítmica que permitirá expresar, de una manera fácil, esta idea de eficiencia de un algoritmo y comparar diferentes algoritmos que resuelven un mismo problema.

El concepto de complejidad será utilizado repetidamente en los módulos posteriores, concretamente en el estudio de los algoritmos que resuelven ciertos problemas asociados a la teoría de grafos. En el módulo "Complejidad computacional", se verá una exposición más extensa y formal de este concepto.

1. Funciones

Uno de los conceptos básicos y más utilizados en matemáticas y en ingeniería es el de función. Las funciones aparecen de forma natural en programación, bases de datos, comunicaciones y, en general, para expresar relaciones entre conjuntos.

Empezaremos este apartado con el producto cartesiano de dos conjuntos. A continuación introduciremos las relaciones y, finalmente, las funciones como un tipo especial de relación. Estudiaremos diferentes tipos de funciones y su relación con el concepto de cardinal de un conjunto. Veremos la diferencia entre conjuntos finitos, infinitos y numerables. Para acabar, utilizaremos las técnicas básicas de contar para calcular el número de funciones entre dos conjuntos finitos.

Estos conocimientos nos serán de utilidad en toda la asignatura en diferentes tareas, como para contabilizar el número de soluciones de un problema dado.

1.1. El producto cartesiano

Conjunto

Recordad que un conjunto es una colección de elementos sin orden y sin repeticiones. Si el conjunto no contiene elementos, se denomina **conjunto vacío** y se representa con el símbolo $\emptyset$.

Definición 1

Dados los conjuntos X e Y, el **producto cartesiano** $\mathbf{X \times Y}$ representa el conjunto de pares ordenados (x,y) en que x pertenece a X, e y pertenece a Y.

Notación

Representaremos los conjuntos por letras mayúsculas ($X,Y,A,...$) y sus elementos por letras minúsculas ($a,b,x,...$). Los elementos que pertenecen a un conjunto se escriben entre llaves y separados por comas: por ejemplo, $X = \{a,b,c\}$.

Ejemplo 1

Si $X = \{1,2,3\}$, $Y = \{a,b,c,d\}$, entonces el conjunto $X \times Y$ es $X \times Y = \{(1,a),(1,b),(1,c),(1,d),(2,a),(2,b),(2,c),(2,d),(3,a),(3,b),(3,c),(3,d)\}$.

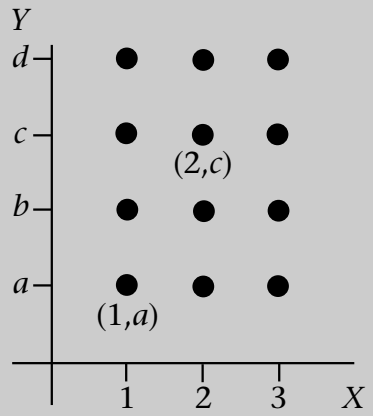
Ejercicio 1

Si $X = \{0,1\}$ y $Y = \{0,1,2\}$, escribir $X \times Y$ y $Y \times X$.

Solución: $X \times Y = \{(0,0),(0,1),(0,2),(1,0),(1,1),(1,2)\}$ y $Y \times X = \{(0,0),(0,1),(1,0),(1,1),(2,0),(2,1)\}$.

Nótese que, en general, son diferentes $X \times Y$ e $Y \times X$, tal y como se ha mostrado en el ejemplo anterior.

El producto cartesiano se puede representar gráficamente como un conjunto de puntos en el plano, donde cada eje representa uno de los dos conjuntos. Así, el producto cartesiano del ejemplo 1 se puede representar así:



1.2. Relaciones binarias

Definición 2

Dados los conjuntos X e Y, una **relación binaria** entre X e Y es un subconjunto de $X \times Y$.

Subconjunto

Recordad que un conjunto X es **subconjunto** de Y ($X \subseteq Y$) si todos los elementos de X pertenecen a Y. Si algún elemento de Y no pertenece a X, diremos que X es un **subconjunto propio** de Y ($X \subset Y$).

Ejemplo 2

Si $X = \{1,2,3\}$, $Y = \{a,b,c,d\}$, entonces el subconjunto de $X \times Y$, $R = \{(1,a),(2,b), (3,c),(3,d)\}$ es una relación binaria.

Ejemplo 3

Si $\mathbb{N}$ denota el conjunto de los números naturales, $\mathbb{N} = \{1,2,3,\ldots\}$, entonces definimos la relación binaria de igualdad $R = \{(x,y) \mid x = y\}$. Obsérvese que, en este caso, se puede escribir $x = y$ en lugar de $x\ R\ y$.

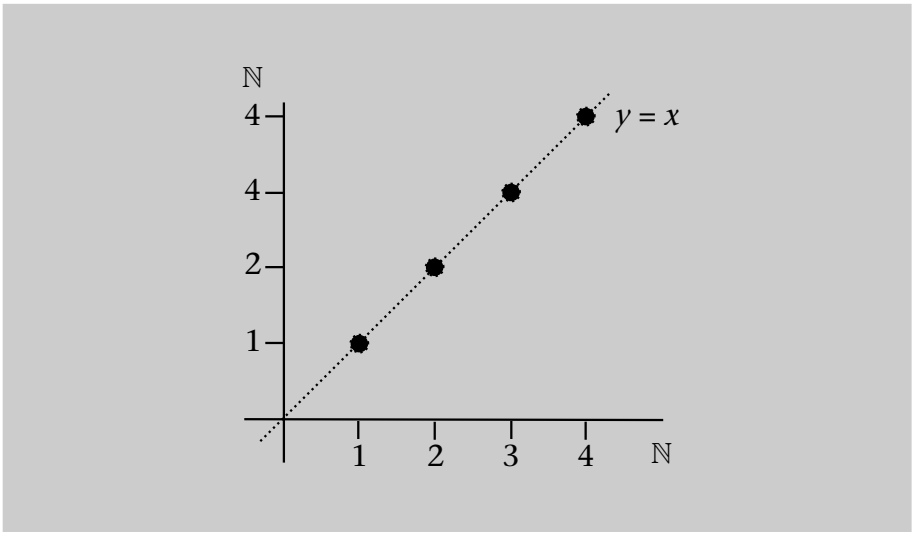
Propiedades

Recordad que un conjunto se puede definir como $A = \{x \mid p(x)\}$ donde p es una propiedad. Esto quiere decir que A contiene todos los elementos que satisfacen la propiedad p.

Notación

Si $(x,y) \in R$, entonces escribiremos $x\ R\ y$.

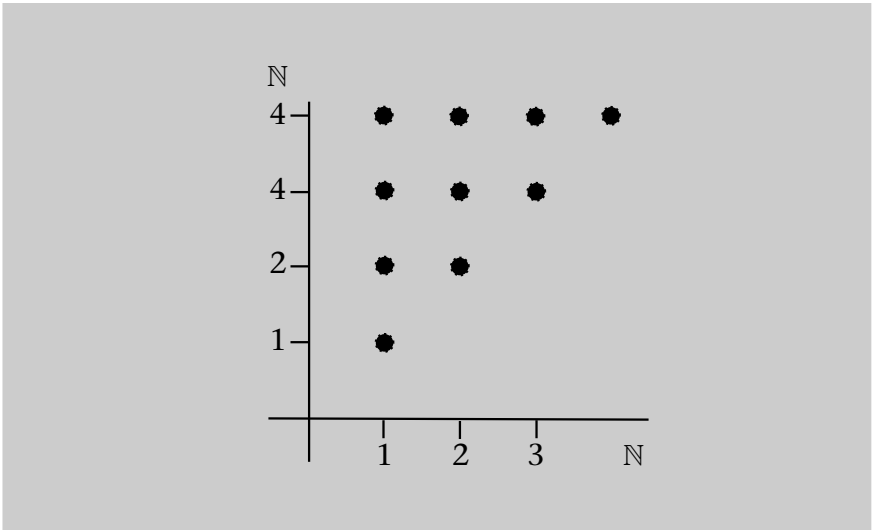
De la misma manera que hemos representado gráficamente el producto cartesiano, también podemos representar las relaciones como puntos del plano. Así, la relación binaria de igualdad sobre el conjunto de los números naturales del ejemplo 3 se puede representar como el conjunto de puntos de coordenadas naturales que están sobre la recta $y = x$.



Ejercicio 2

Definir la relación binaria "es menor o igual que" en el conjunto $\mathbb{N}$ de los números naturales y representarla gráficamente.

Solución: La relación binaria "es menor o igual que" será, $x\ R\ y$ si y sólo si $x \leq y$. Su representación gráfica será el conjunto de puntos que se hallan por encima de la recta $y = x$.



1.3. Funciones inyectivas, exhaustivas y biyectivas

Notación

Una función f de X en Y se denota también por $f : X \to Y$.

Definición 3

Dados los conjuntos X e Y, una **función** f de X en Y es una relación binaria f entre X e Y, tal que, dados dos elementos cualesquiera (x,y), (x',y') de f, si $x = x'$, entonces $y = y'$.

Si (x,y) pertenece a f, también se dice que $f(x) = y$. En este caso, el elemento y se llama **imagen** de x por f, y el elemento x, **antiimagen** de y por f. El conjunto de las antiimágenes se llama **dominio** de la función f, y el conjunto de las imágenes, **imagen** de f.

Ahora bien, en el contexto de la matemática discreta es mejor considerar sólo las funciones en que el dominio de f sea todo el conjunto X. A partir de ahora, pues, supondremos que una función debe cumplir esta propiedad adicional.

Ejemplo 4

Si $X = \{1,2\}$ e $Y = \{1,2,3,4,5\}$ entonces $f = \{(1,1),(2,4)\}$ es una función. Esta función es la relación binaria $f(x) = x^2$ entre X e Y.

La relación binaria $f(x) = x^3$ no es una función entre X e Y ya que sólo contiene el par (1,1). El par (2,8) no pertenece a la relación y, por tanto, f no cumpliría la condición que el dominio de f debe ser todo el conjunto X.

Ejercicio 3

Sean $X = \{1,2,3\}$ e $Y = \{a,b,c,d\}$. Comprobar que $f = \{(1,a),(2,c),(2,b),(3,d)\}$ no es una función de X en Y. Análogamente, $g = \{(1,a),(2,c),(3,e)\}$ tampoco es una función de X en Y.

Solución: f es una relación binaria pero no es una función de X en Y puesto que el elemento 2 tiene dos imágenes diferentes (c y b) en Y. g no es una relación binaria entre X e Y ya que $g(3) = e \notin Y$. Por lo tanto, g no puede ser una función.

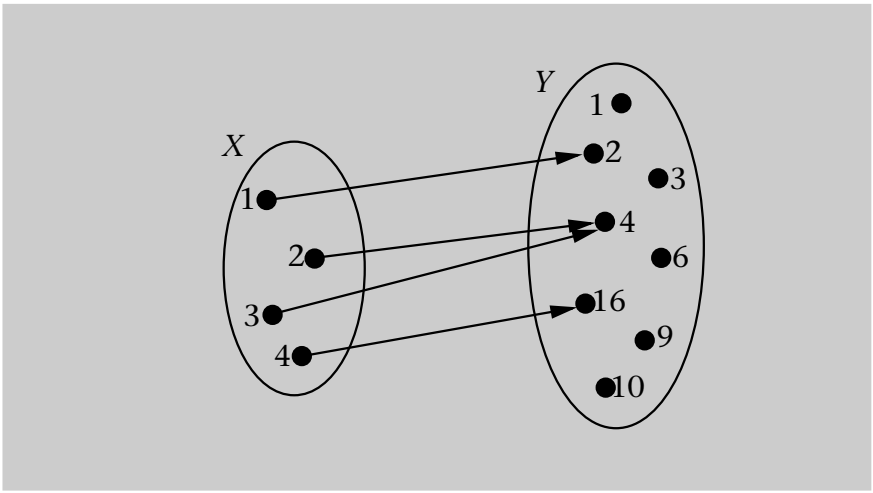
Recordad que $x \in A$ significa que x pertenece a A y $x \notin A$ significa que x no pertenece a A.

Ejercicio 4

Si $X = \{1,2,3,4\}$ e $Y = \{1,2,3,4,6,9,10,16\}$, considerar la función de X en Y, $f : X \to Y$, definida por $f(x) = x + 1$ si x es impar, y $f(x) = x^2$ si x es par. Describir la relación binaria f entre X e Y. Observar que $f(2) = f(3) = 4$. Hacer una lista de los elementos de Y con las antiimágenes correspondientes.

Solución: La función f se puede escribir como $f = \{(1,2),(2,4),(3,4),(4,16)\}$.

En este ejercicio podríamos hacer el siguiente gráfico para representar la función f:



Los elementos de X de los que sale una flecha son las antiimágenes de los elementos de Y a los que llega la flecha correspondiente. Los elementos de Y a los que llega una flecha son las imágenes de los elementos correspondientes de X. Así, $1 \in X$ es antiimagen de $2 \in Y$, 2 y $3 \in X$ son antiimágenes de $4 \in Y$, y $4 \in X$ es antiimagen de $16 \in Y$. Los elementos $1,3,6,9,10 \in Y$ no tienen antiimagen. En la siguiente tabla vemos un resumen:

Y	1	2	3	4	6	9	10	16
Antiimagen	ninguna	1	ninguna	2, 3	ninguna	ninguna	ninguna	4

Obsérvese que en una función todos los elementos de X deben ser antiimagen de algún elemento de Y y no necesariamente todos los elementos de Y deben ser imagen de alguno de X.

Definición 4

Una función $f : X \to Y$ es **exhaustiva** si cada elemento de Y tiene por lo menos una antiimagen (en X). Es **inyectiva** si cada elemento de Y tiene como máximo una antiimagen. Es **biyectiva** si cada elemento de Y tiene exactamente una antiimagen, es decir, si es inyectiva y exhaustiva a la vez.

Ejercicio 5

Comprobar si la función $f : \{1,2,3,4\} \to \{a,b,c,d\}$, definida por $f(1) = d$, $f(2) = a$, $f(3) = b$, $f(4) = c$, es biyectiva.

Solución: Cada elemento del conjunto $\{a,b,c,d\}$ tiene exactamente una antiimagen y, por lo tanto, es biyectiva.

Ejemplo 5

La función $f : \mathbb{N} \to \mathbb{N}$ definida por $f(x) = x^2$ es inyectiva pero no es exhaustiva. En efecto, f es inyectiva porque dos números naturales diferentes no pueden tener cuadrados iguales. No es exhaustiva puesto que, por ejemplo, 2 no tiene antiimagen (no existe ningún número natural x tal que $x^2 = 2$).

Es necesario observar que la función $g : \mathbb{Z} \to \mathbb{Z}$ definida por $g(x) = x^2$ no es inyectiva (2 y –2 tienen la misma imagen: 4) y tampoco es exhaustiva (por ejemplo, no hay ningún número entero que tenga cuadrado igual a 2).

El tipo de función

Tal y como se muestra en el ejemplo 5, que una función sea de un tipo determinado (exhaustiva, inyectiva o biyectiva) no sólo depende de la expresión que la define, sino también de los conjuntos entre los cuales está definida.

Ejemplo 6

La función identidad $j : X \to X$, o sea, la función de X en X que hace corresponder a cada elemento $x \in X$ el mismo elemento x ($j(x) = x$ para todo $x \in X$), es biyectiva.

Ejercicios

1. Encontrar todas las funciones biyectivas de $X = \{1,2,3\}$ en $Y = \{a,b,c\}$.

2. Encontrar todas las funciones exhaustivas de $X = \{1,2,3\}$ en $Y = \{a,b\}$.

3. Encontrar todas las funciones inyectivas de $X = \{1,2,3\}$ en $Y = \{a,b,c,d\}$.

4. Demostrar que la función $f : \mathbb{N} \to \mathbb{N}$ definida por $f(x) = x + 1$ es inyectiva pero no exhaustiva.

5. Demostrar que la función $f : \mathbb{Z} \to \mathbb{Z}$ definida por $f(x) = x + 1$ es biyectiva.

Soluciones

1. Si se utiliza la notación $f(1)f(2)f(3)$ para representar una función f, tenemos las seis biyecciones siguientes: *abc, acb, bac, bca, cab, cba*.

Obsérvese que con esta notación se obtiene una simplificación notable en la escritura. Así, *cab* es la función f definida por $f(1) = c$, $f(2) = a$, $f(3) = b$.

2. Con la misma notación anterior, tenemos seis funciones exhaustivas: *aab, aba, baa, abb, bab, bba*.

3. Hay veinticuatro funciones inyectivas, que son las siguientes: *abc, abd, acb, acd, adb, adc, bac, bad, bca, bcd, bda, bdc, cab, cad, cba, cbd, cda, cdb, dab, dac, dba, dbc, dca, dcb*.

4. La función f es inyectiva, puesto que si dos elementos diferentes, $x \neq y$, tuvieran la misma imagen podríamos escribir $x + 1 = y + 1$ y, entonces $x = y$, que no puede ser. La función no es exhaustiva, puesto que el número 1 no tiene ninguna antiimagen (de hecho, 1 es el único número sin ninguna antiimagen).

5. La función f es inyectiva, puesto que (igual que en el ejercicio 4) $x+1 = y+1$ implica que $x = y$. En este caso f es exhaustiva, puesto que cualquier número entero tiene antiimagen: la antiimagen de z es $z – 1$. Observar que esto no sería cierto si f estuviese definida sobre el conjunto de los números naturales, puesto que $z = 1$ no tendría antiimagen ($z – 1 = 0$ no pertenece a $\mathbb{N}$).

1.4. Conjuntos finitos, infinitos y numerables

Notación

$|A| = n$ indica que el conjunto A tiene cardinal n.
El cardinal del conjunto vacío se define igual a 0: $|\emptyset| = 0$.
$\mathbb{N}_n$ es el conjunto $\{1,2,3,\ldots,n\}$. Así, $\mathbb{N}_4 = \{1,2,3,4\}$.

Definición 5

Un conjunto A tiene n elementos si hay una función biyectiva de $\{1,2,3,\ldots,n\}$ en A. En este caso también decimos que el conjunto A tiene **cardinal** n.

Ejemplo 7

El conjunto $A = \{1,1/2,1/3,1/4,1/5,\ldots\}$ es tal que no existe ningún número natural n que permita construir una biyección (función biyectiva) de $\mathbb{N}_n$ en A. En estos casos se dice que A es un conjunto infinito (de cardinal infinito).

Definición 6

Un conjunto X es **finito** si es vacío o bien si tiene n elementos. De lo contrario, decimos que X es un conjunto **infinito**.

Como consecuencia de la definición, es necesario observar que un conjunto X es infinito si no es vacío y si no existe ningún número natural n para el cual se pueda construir una biyección de $\mathbb{N}_n$ en X.

A continuación se presentan una serie de resultados bastante intuitivos.

Proposición 1

- El conjunto de los números naturales $\mathbb{N}$ es un conjunto infinito.
- Un conjunto X es infinito si, y sólo si, se puede construir una función f de $\mathbb{N}$ en X que sea inyectiva.
- Si X es un subconjunto infinito de un conjunto Y, entonces Y es infinito.
- Dado un conjunto finito X y un subconjunto Y de X, entonces Y también es finito y se verifica $|Y| \leq |X|$.

Ejemplo 8

De acuerdo con la proposición 1, podemos afirmar, por ejemplo, que el conjunto $\mathbb{Z}$ de los números enteros es infinito, puesto que contiene un subconjunto (el conjunto $\mathbb{N}$) que ya sabemos que es infinito.

Definición 7

Un conjunto X es **numerable** si es finito o existe una biyección de $\mathbb{N}$ en X.

Por lo tanto, un conjunto X es numerable si podemos hacer una lista con los elementos de X tal que cualquier elemento de X sea un elemento de la lista, y dos elementos diferentes de la lista correspondan a elementos diferentes de X.

Ejemplo 9

El conjunto $\mathbb{Z}$ de los números enteros es numerable. En efecto, sólo es necesario considerar la función f de $\mathbb{N}$ en $\mathbb{Z}$ definida por

$$f(x) = \begin{cases} -\frac{x}{2} & \text{si } x \text{ es par} \\ \frac{x-1}{2} & \text{si } x \text{ es impar.} \end{cases}$$

Esta función f es biyectiva (comprobadlo), y por lo tanto podemos decir que $\mathbb{Z}$ es numerable. La función f nos permite escribir todos los números enteros en forma de lista: $0, 1, -1, 2, -2, 3, -3, \ldots$ (sin elementos repetidos).

Los conjuntos de números

Hay conjuntos infinitos que son numerables y otros que no lo son. El conjunto de los números reales no es numerable (este hecho se demuestra como un resultado básico en cualquier curso de análisis matemático o de cálculo).
El conjunto de los números enteros $\mathbb{Z}$ y el conjunto de los números racionales $\mathbb{Q}$ son ejemplos fundamentales de conjuntos numerables.

Es importante observar que la situación descrita en el ejemplo anterior es sorprendente, puesto que $\mathbb{N}$ es un **subconjunto propio** de $\mathbb{Z}$, es decir, hay números enteros que no son naturales; por otro lado, podemos decir que existen tantos números enteros como números naturales (hemos encontrado una biyección de $\mathbb{N}$ en $\mathbb{Z}$). Este hecho curioso no se puede dar en el ámbito de los conjuntos finitos: un conjunto finito no tiene ningún subconjunto propio que tenga el mismo número de elementos.

El cardinal del conjunto de números naturales se escribe como $\aleph_0$ (se pronuncia *alef subcero*) y es el cardinal más pequeño no finito que se conoce. El cardinal de los números reales se escribe como $\aleph_1$, y ya sabemos que $\aleph_0 \neq \aleph_1$ (puesto que los números reales no son numerables). En el campo de la matemática transfinita se demuestra que $2^{\aleph_0} = \aleph_1$. Se llama *hipótesis del continuo* al enunciado que dice que no hay ningún conjunto que tenga un cardinal intermedio entre $\aleph_0$ y $\aleph_1$.

George Cantor (1845-1918) demostró que $\aleph_0 \neq \aleph_1$. David Hilbert (1862-1943) formuló por primera vez la hipótesis del continuo como primer problema en la famosa lista que presentó en el congreso mundial de matemáticas que tuvo lugar en París el 1900.

En 1938 Kurt Gödel (1906-1978) y en 1963 Paul Cohen (1934-2007) demostraron que, con los axiomas habituales de la matemática, la hipótesis del continuo es indemostrable tanto en sentido positivo (Cohen) como negativo (Gödel).

Ejercicios

6. Demostrar que si X, Y son conjuntos finitos y no vacíos con el mismo cardinal, entonces si $f : X \to Y$ es inyectiva, f es también biyectiva.

7. Demostrar que si X, Y son conjuntos finitos y no vacíos con el mismo cardinal, entonces si $f : X \to Y$ es exhaustiva, f es también biyectiva.

8. Demostrar que el conjunto $\mathbb{N} \times \mathbb{N}$ es un conjunto numerable.

Soluciones

6. Probamos este resultado utilizando el método de inducción sobre el cardinal, n, de los conjuntos X y Y. Evidentemente si $n = 1$ el enunciado es cierto. Supongámoslo cierto para $n-1$ y veamos que lo es para n. Sea f una función inyectiva del conjunto X en el conjunto Y con $|X| = |Y| = n$. Escogemos $a \in X$ y sea $b = f(a)$. Consideramos $X' = X - a$ y $Y' = Y - b$. Entonces resulta que $|X'| = |Y'| = n - 1$, y la función f', restricción de la función f a X', es una función de X' en Y' que es inyectiva.

Notación

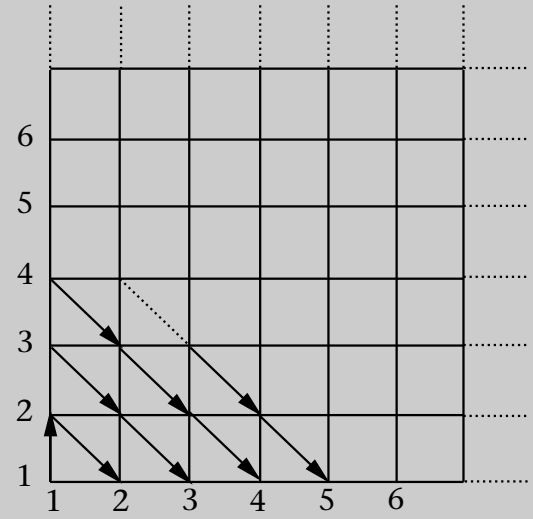
El conjunto $X - a$ es el conjunto obtenido eliminando el elemento a de X. El conjunto $X - a$ es un subconjunto de X.

Por lo tanto, aplicando la hipótesis de inducción podemos afirmar que f' es biyectiva (de X' en Y') y, consecuentemente, f es también biyectiva.

7. Suponemos f de X en Y exhaustiva con $|X| = |Y| = n$. Si f no fuera inyectiva se podría construir a partir de f (eliminando elementos de X con la misma imagen) una función f' biyectiva entre una parte propia de X y Y, lo cual es imposible puesto que X e Y deben tener el mismo número de elementos.

8. Para demostrar que $\mathbb{N} \times \mathbb{N}$ es numerable es suficiente con numerar los elementos de $\mathbb{N} \times \mathbb{N}$ de la manera siguiente: $1 \to (1,1)$, $2 \to (1,2)$, $3 \to (2,1)$, $4 \to (1,3)$, $5 \to (2,2)$, $6 \to (3,1)$, $7 \to (1,4)$, $8 \to (2,3)$, $9 \to (3,2)$, $10 \to (4,1)$, etc. O sea, en el orden que se muestra en la siguiente figura. Por lo tanto, acabamos de definir una función: $f : \mathbb{N} \longrightarrow \mathbb{N} \times \mathbb{N}$, donde $f(n) = (k,m)$.

Parece claro, examinando el diagrama, que la función es biyectiva (a cada número natural le corresponde un único punto de la red cuadrada, y a cada punto de la red le corresponde un único número natural).



1.5. Cálculo del número de funciones

Dado un conjunto X con n elementos, cada función f del conjunto $\mathbb{N}_r = \{1,2, \ldots, r\}$ en X se puede identificar por la lista $f(1),f(2),\cdots, f(r)$ de r elementos de X, y recíprocamente cada lista de r elementos (quizás con algunos de ellos repetidos) de X, $x_1,x_2,\cdots,x_r$, representa una función $f : \mathbb{N}_r \to X$ tal que $f(1) = x_1$, $f(2) = x_2$, $\ldots$, $f(r) = x_r$ (véase los ejercicios 1, 2 y 3).

Definición 8

La lista ordenada de elementos de X, $x_1,x_2,\cdots,x_r$, se denomina una *r*-**muestra ordenada con repetición**.

Así, el conjunto de las r-muestras ordenadas con repetición del conjunto X no es otra cosa que el conjunto de las funciones de $\mathbb{N}_r$ en X.

Ejemplo 10

La 3-muestra ordenada con repetición *aba* sobre $X = \{a,b\}$ puede ser interpretada como la función f de $\mathbb{N}_3 = \{1,2,3\}$ en $X = \{a,b\}$, tal que $f(1) = a$, $f(2) = b$, $f(3) = a$. Recíprocamente, a la función g de $\mathbb{N}_3 = \{1,2,3\}$ en $X = \{a,b\}$ definida por $g(1) = a$, $g(2) = b$, $g(3) = b$ le podemos asociar la 3-muestra ordenada con repetición $g(1)g(2)g(3) = abb$.

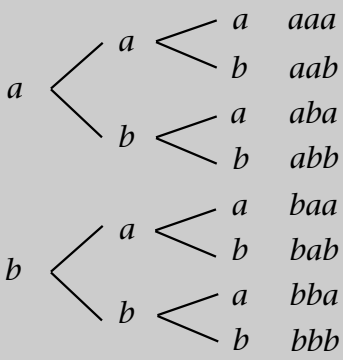
El conjunto de las 3-muestras ordenadas sin repetición del conjunto $X = \{a,b\}$ es, pues, el conjunto de las funciones de $\mathbb{N}_3$ en X.

Se puede construir el conjunto de las r-muestras ordenadas con repetición de un conjunto X de n elementos, mediante un diagrama en árbol. Supongamos que se desea escribir todas las 3-muestras ordenadas con repetición

***r*-muestras**

Recordad que el número de r-muestras ordenadas con repetición de un conjunto X de n elementos se denota per $VR(n,r)$. Coincide con el número de variaciones con repetición de un conjunto de n elementos tomados r a r.

de los elementos de $X = \{a,b\}$. A continuación se presenta el diagrama correspondiente:



Puesto que se han obtenido $VR(2,3) = 2^3 = 8$ muestras, hay 8 funciones de $\mathbb{N}_3$ en $X = \{a,b\}$.

Proposición 2

Si el conjunto X tiene n elementos, entonces podemos formar

$$VR(n,r) = n^r$$

r-muestras ordenadas con repetición del conjunto X y, por lo tanto, hay n^r funciones del conjunto $\mathbb{N}_r$ en X.

Demostración: Consultad Masià y otros (2007). ∎

Ejercicio 6

¿Cuántas funciones podemos construir de $\mathbb{N}_7$ en $X = \{0,1\}$?

Solución: El número pedido no es otro que $VR(2,7) = 2^7 = 128$. Hay un total de ciento veintiocho funciones (7-muestras ordenadas con repetición) de $\mathbb{N}_7$ en $X = \{0,1\}$.

Ejercicio 7

El número total de funciones de $\mathbb{N}_3$ en un conjunto B es 125. ¿Cuántos elementos tiene B?

Solución: Si B tiene cardinal n, se debe cumplir $n^3 = 125$. Por tanto, $n = 5$.

Ejercicio 8

¿Existe un conjunto X con el que se puedan definir quinientas doce funciones del conjunto $\mathbb{N}_3$ en X? ¿Y del conjunto $\mathbb{N}_4$ en X?

Solución: En el primer caso, $r = 3$ y $n^3 = 512$. Por lo tanto, debemos resolver la ecuación $n^3 = 512$, que tiene la solución entera positiva $n = 8$. Así, X puede ser cualquier conjunto de ocho elementos. En el segundo, $r = 4$ y deberíamos resolver la ecuación $n^4 = 512$, que no tiene solución entera. Por lo tanto, no existe tal conjunto X.

Ejercicios

9. Se tira una moneda cinco veces y se anotan los resultados ordenadamente: un 1 si sale cruz y un 0 si sale cara. ¿Cuántas listas ordenadas diferentes de cinco elementos pueden salir?

10. ¿Cuántos números de tres cifras impares podemos escribir?

11. En el juego del dominó, cada ficha se puede representar por el símbolo $[x|y]$, donde x, y son elementos del conjunto $\{0,1,2,3,4,5,6\}$. Representar las fichas del dominó como funciones entre dos conjuntos. ¿Cuántas fichas tiene el dominó? Razonar la respuesta.

12. Consideremos el alfabeto ternario $X = \{0,1,2\}$, ¿cuántas palabras no vacías hay de longitud no superior a 6? ¿Cuántas de estas palabras empiezan por cero?

Soluciones

9. Sea $X = \{0,1\}$ el conjunto de los resultados de las tiradas. Cada lista será una función del conjunto $\mathbb{N}_5$ en el conjunto X. Por lo tanto, tendremos $VR(2,5) = 2^5 = 32$ listas diferentes.

10. Sea $X = \{1,3,5,7,9\}$ el conjunto de las cifras impares. Cada número de tres cifras impares será una función del conjunto $\mathbb{N}_3$ en el conjunto X. Tendremos, $VR(5,3) = 5^3 = 125$ números de tres cifras impares.

11. A cada ficha de dominó le podemos asociar una función del conjunto $\mathbb{N}_2$ en el conjunto $X = \{0,1,2,3,4,5,6\}$. El número total de estas funciones es $VR(7,2) = 49$. No obstante, el juego del dominó sólo tiene 28 fichas. La diferencia es debida al hecho de que la ficha $[x|y]$ es la misma que la ficha $[y|x]$, es decir, debemos restar la mitad de las funciones $f : \mathbb{N}_2 \to X$ tales que $f(1) \neq f(2)$, que son 21.

12. Las palabras no vacías de longitud no superior a 6 tendrán longitudes 1, 2, 3, 4, 5 y 6. Cada longitud corresponde al número de funciones del conjunto $\mathbb{N}_r$ ($r = 1,2,3,4,5,6$) en X. El total será, $VR(3,1) + VR(3,2) + VR(3,3) + VR(3,4) + VR(3,5) + VR(3,6) = 3 + 3^2 + 3^3 + 3^4 + 3^5 + 3^6 = 1.092$.

La tercera parte empezará por 0, es decir, 364 palabras.

1.6. Cálculo del número de funciones inyectivas (biyectivas)

Dado un conjunto X con n elementos, cada función inyectiva f del conjunto $\mathbb{N}_r = \{1,2,\ldots,r\}$ en X se puede identificar por la lista $f(1),f(2),\cdots,f(r)$ de r elementos diferentes de X y recíprocamente cada lista de r elementos (sin repeticiones) de X, $x_1,x_2,\cdots,x_r$, repre-

senta una función inyectiva $f : \mathbb{N}_r \to X$ tal que $f(1) = x_1$, $f(2) = x_2$, $\ldots$, $f(r) = x_r$, ya que las funciones inyectivas son aquellas en las que cada elemento de X tiene como máximo una antiimagen.

Notación

Recordad que el número de r-muestras ordenadas sin repetición de un conjunto X de n elementos se denota por $V(n,r)$. Coincide con el número de variaciones (sin repetición) de un conjunto de n elementos tomados r a r.

Definición 9

La lista ordenada de r elementos diferentes de X, $x_1,x_2,\cdots,x_r$ es una r-**muestra ordenada sin repetición** de elementos de X.

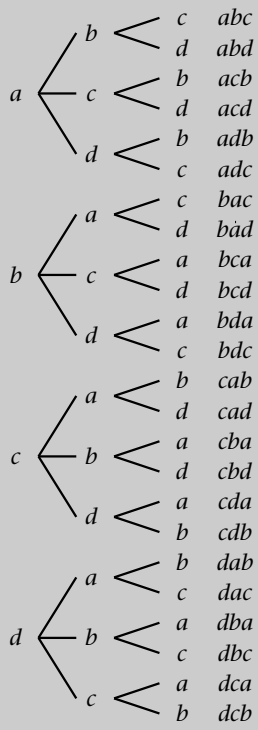
Así, el conjunto de las r-muestras ordenadas sin repetición del conjunto X es el conjunto de las funciones inyectivas de $\mathbb{N}_r$ en X.

Ejemplo 11

La 3-muestra ordenada sin repetición *abc* del conjunto $X = \{a,b,c,d\}$ puede ser interpretada como la función inyectiva f de $\mathbb{N}_3$ en X, definida por $f(1) = a$, $f(2) = b$, $f(3) = c$. Sobre el mismo conjunto, la 3-muestra ordenada sin repetición *bad* sería la función inyectiva g de $\mathbb{N}_3$ en X definida por $g(1) = b$, $g(2) = a$, $g(3) = d$. Recíprocamente, la función h de $\mathbb{N}_3$ en X definida por $h(1) = d$, $h(2) = c$, $h(3) = b$ se puede interpretar como la 3-muestra ordenada sin repetición *dcb* sobre X.

El conjunto de las 3-muestras ordenadas sin repetición del conjunto $X = \{a,b,c,d\}$ es, pues, el conjunto de las funciones inyectivas de $\mathbb{N}_3$ en X.

Como hemos hecho en el ejemplo 10, se puede construir todo el conjunto de 3-muestras ordenadas sin repetición de $X = \{a,b,c,d\}$ utilizando un diagrama de árbol:



El número de muestras se puede calcular a partir de los diferentes niveles del árbol. En el primer nivel hay 4 elementos, en el segundo 3 y en el último 2. Así hemos obtenido $V(4,3) = 4 \cdot 3 \cdot 2 = 24$ muestras ordenadas sin repetición, o sea hay 24 funciones inyectivas de $\mathbb{N}_3$ en $X = \{a,b,c,d\}$.

Observad que *r* no puede ser mayor que *n*.

Proposición 3

Si el conjunto X tiene n elementos, entonces podemos formar

$$V(n,r) = n \cdot (n-1) \cdot (n-2) \cdots (n-(r-1))$$

r-muestras ordenadas sin repetición del conjunto X y, por tanto, $V(n,r) = n \cdot (n-1) \cdot (n-2) \cdots (n-(r-1))$ funciones inyectivas del conjunto $\mathbb{N}_r$ en X.

Demostración: Consultad Masià y otros (2007). ∎

Ejercicio 9

¿Cuántas funciones inyectivas podemos construir de $\mathbb{N}_7$ en $X = \{a,b,c,d,e,f,g,h,i\}$?

Solución: El conjunto X tiene 9 elementos. Por tanto, el número de funciones inyectivas de $\mathbb{N}_7$ en X es $V(9,7) = 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 = 181.440$.

Observad que si *r* = 0, entonces *V*(*n*,0) = 1, y también que *V*(0,0) = 1.

Ejercicio 10

¿Cuántas funciones inyectivas se pueden construir del conjunto $Y = \{x,y,z\}$ en el conjunto $X = \{a,b,c,d\}$?

Solución: El conjunto Y contiene tres elementos. Por tanto, esto es equivalente a calcular el número de funciones inyectivas de $\mathbb{N}_3$ en $X = \{a,b,c,d\}$. El número pedido será 24, igual que en el ejemplo 11.

Ejercicio 11

El número total de funciones inyectivas de $\mathbb{N}_2$ en un conjunto A es 42. ¿Cuántos elementos tiene A?

Solución: Si A tiene cardinal n, debemos aplicar la fórmula de $V(n,r)$ cuando $r = 2$. Así, $V(n,2) = n(n-1) = 42$, y de aquí $n = 7$. Por lo tanto, el conjunto A tiene siete elementos.

A continuación analizaremos las funciones que, además de ser inyectivas, son exhaustivas: o sea, las funciones biyectivas. En este caso, dado un conjunto X con n elementos, cada función biyectiva f del conjunto $\mathbb{N}_n = \{1,2,\ldots,n\}$ en X (o del conjunto X en él mismo) se puede identificar como una n-muestra ordenada sin repetición de X. Recordemos que las funciones biyectivas son aquellas en que cada elemento de X tiene exactamente una antiimagen: o sea, cada

elemento de X debe aparecer en cada muestra exactamente una vez (las muestras deben contener, pues, n elementos).

Definición 10

Las n-muestras ordenadas sin repetición de un conjunto X de cardinal n se llaman **permutaciones** de los elementos del conjunto X y su número se denota por $P(n)$.

Así, el conjunto de las n-muestras ordenadas sin repetición del conjunto X es el conjunto de las funciones biyectivas de $\mathbb{N}_n$ en X (o de X en él mismo).

Proposición 4

Si el conjunto X tiene n elementos, entonces podemos formar

$$P(n) = V(n,n) = n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1$$

permutaciones de los elementos de X.

Por lo tanto, hay $P(n)$ funciones biyectivas de X en X.

Demostración: Consultad Masià y otros (2007). ∎

Además, por este motivo, una función f biyectiva de X en X también recibe el nombre de **permutación** de X.

Definición 11

El producto $n \cdot (n-1) \cdot (n-2) \cdots 1$ se denomina **factorial** de n (o n **factorial**) y se denota por $n!$ Por conveniencia se define $0! = 1$.

Así, $P(n) = n!$

Ejemplo 12

El número de funciones biyectivas de $X = \{x,y,z,t\}$ en él mismo es $4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$.

Ejercicio 12

Hay 120 funciones biyectivas de X en X, ¿cuál es el cardinal de X?

Solución: Puesto que 5! = 120 podemos afirmar que el cardinal de X es cinco.

Ejercicios

13. Demostrar la siguiente igualdad: $V(n,r) \cdot V(n-r,s-r) = V(n,s)$, donde n, r y s son enteros positivos tales que $n > s > r$.

14. ¿Cuántas palabras de cuatro letras se pueden formar con un alfabeto de diez símbolos, teniendo en cuenta que no hay restricciones de léxico, excepto que no puede haber letras repetidas en una misma palabra?

15. ¿Cuántos números menores de cien y con las dos cifras diferentes se pueden escribir, utilizando sólo las cifras 1, 3, 5 y 7?

16. En un concurso literario participan quince personas y se asignan tres premios: el primero, de 3.000 €, el segundo, de 2.000 € y el tercero, de 1.000 €. Si una misma persona no puede recibir más de un premio, ¿de cuántas maneras diferentes se pueden distribuir los premios?

17. Escribir el valor de $n!$ para $n = 1,2,\ldots,10$.

18. En una estantería hay seis libros diferentes. ¿De cuántas maneras se pueden ordenar?

19. Simplificar algebraicamente la siguiente expresión:

$$\frac{(m-1)!(a+b)!}{m!(a+b-1)!}.$$

20. Escribir en forma de cociente de dos factoriales el producto

$$n \cdot (n-1) \cdot (n-2) \cdots (n-r+1).$$

21. Utilizando un diagrama de árbol, escribir todos los números que se obtienen al permutar las cuatro cifras 1,2,3,4. Ordenarlos de menor a mayor. ¿Cuál será la suma de todos los números obtenidos?

Soluciones

13. $V(n,r) \cdot V(n-r,s-r) = [n \cdot (n-1) \cdots (n-r+1)] \cdot [(n-r) \cdot (n-r-1) \cdots (n-r-s+r+1)] = [n \cdot (n-1) \cdots (n-r+1)] \cdot [(n-r) \cdot (n-r-1) \cdots (n-s+1)] = V(n,s)$.

14. Nos piden el número de funciones inyectivas entre el conjunto $\mathbb{N}_4$ y el conjunto X formado por el alfabeto de 10 símbolos (4-muestras ordenadas sin repetición que se pueden hacer con un alfabeto de diez símbolos): es decir $V(10,4) = 10 \cdot 9 \cdot 8 \cdot 7 = 5.040$.

15. Nos piden el número de funciones inyectivas entre el conjunto $\mathbb{N}_2$ y el conjunto $X = \{1,3,5,7\}$. Hay $V(4,2) = 4 \cdot 3 = 12$ números con las condiciones

exigidas. La condición "menores de cien" es inútil ya que los números deben ser de dos cifras.

16. Si una misma persona no puede recibir más de un premio (no hay repetición) entonces nos piden el número de funciones inyectivas entre $\mathbb{N}_3$ y el conjunto de las 15 personas, los premios se pueden distribuir de $V(15,3) = 15 \cdot 14 \cdot 13 = 2.730$ maneras.

17. $1! = 1,\ 2! = 2,\ 3! = 6,\ 4! = 24,\ 5! = 120,\ 6! = 720,\ 7! = 5.040,\ 8! = 40.320,\ 9! = 362.880,\ 10! = 3.628.800$.

18. La respuesta es $P(6) = 6! = 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 720$. Así es equivalente a calcular el número de funciones biyectivas entre el conjunto $\mathbb{N}_6$ y el conjunto de les posiciones de la estantería.

19. $$\frac{(m-1)!(a+b)!}{m!(a+b-1)!} = \frac{(m-1)(m-2)\cdots 2\cdot 1\cdot (a+b)(a+b-1)\cdots 2\cdot 1}{m(m-1)\cdots 2\cdot 1\cdot (a+b-1)(a+b-2)\cdots 2\cdot 1}.$$

Eliminando los términos repetidos en el numerador y en el denominador, se obtiene $\frac{a+b}{m}$.

20. $$n\cdot(n-1)\cdot(n-2)\cdots(n-r+1) = \frac{n!}{(n-r)!}$$

21. Las $4! = 24$ permutaciones de 1, 2, 3, 4 son:

1234,1243,1324,1342,1423,1432,
2134,2143,2314,2341,2413,2431,
3124,3142,3214,3241,3412,3421,
4123,4132,4213,4231,4312,4321.

Hemos escrito las permutaciones a partir del diagrama en árbol correspondiente, y así los números quedan ordenados de menor a mayor.

Por lo que respecta a la suma de todos los números obtenidos, obsérvese que si sumamos las unidades obtenemos: $6 \cdot 1 + 6 \cdot 2 + 6 \cdot 3 + 6 \cdot 4 = 60$; la suma de las decenas es, también, 60, etc. La suma total será, por lo tanto, 66.660.

2. Algoritmos

Para realizar cualquier tarea, los computadores necesitan recibir información e instrucciones sobre cómo procesarla para obtener un resultado. La capacidad de memoria y el tiempo de cálculo que necesitará un ordenador para completar una tarea dependerán de varios factores: el volumen de información a tratar, la dificultad del problema a resolver y la habilidad para suministrar las instrucciones que permitan una utilización adecuada de los recursos disponibles. En este apartado, se estudian estos factores que nos darán una primera idea de la dificultad que pueden tener los problemas computacionales y la manera de evaluar nuestros programas. También se mostrará que algunas tareas que realizan los computadores son más complejas que otras.

2.1. Problemas fáciles y problemas difíciles

En este subapartado aprenderemos a distinguir entre problemas fáciles y problemas difíciles de resolver desde el punto de vista computacional.

Ved también

En este subapartado no daremos una definición precisa del concepto de problema, que pospondremos hasta el módulo "Complejidad computacional".

Los motivos que hacen que un problema computacional pueda considerarse difícil de resolver pueden ser diversos:

- El número de posibles soluciones es demasiado grande para encontrar la mejor solución utilizando la búsqueda exhaustiva entre todas estas posibles soluciones.
- El problema es tan complicado que nos vemos obligados a utilizar modelos simplificados, cuya solución está lejos de la solución del problema inicial.
- La función de evaluación utilizada para describir la calidad de una solución no es lo suficientemente buena o no se ajusta a las necesidades del problema.
- La persona que tiene que resolver el problema no está lo suficientemente preparada para encontrar la solución.

Ejemplo 13

El problema de hallar el número de soluciones de una ecuación con varias incógnitas puede parecer bastante difícil. Por ejemplo, el número de solu-

ciones enteras no negativas de la ecuación $x+y+z+t = 6$ es 84. Una solución sería $x = 1$, $y = 1$, $z = 2$ y $t = 2$. Otra solución sería $x = 1$, $y = 3$, $z = 1$ y $t = 1$, y podríamos seguir así hasta que las hallemos todas.

Este método no parece muy eficiente si hay muchas incógnitas. Por suerte, podemos calcular el número de soluciones si transformamos este problema (número de soluciones enteras no negativas de una ecuación) en otro problema que sea equivalente (número de muestras con repetición de un conjunto) del que conocemos su solución.

Podemos imaginarnos 6 platos iguales que queremos repartir en 4 pilas (x, y, z, t). Podemos representar la solución $x = 1$, $y = 1$, $z = 2$, $t = 2$ como las pilas:

$$\star|\star|\star\star|\star\star$$

donde $\star$ representa un plato y $|$ la separación entre las pilas. En total tenemos 9 posiciones donde colocar los 3 separadores de pilas. Por tanto, el número de soluciones coincidirá con el número de maneras de seleccionar 3 posiciones de un total de 9, o sea $\binom{9}{3} = 84$.

En general, el número de soluciones enteras no negativas de una ecuación de la forma $x_1 + x_2 + \cdots + x_r = n$ coincide con el número de $(r-1)$-muestras no ordenadas sin repetición de un conjunto de n elementos. Este número es $\binom{n+r-1}{r-1}$.

Intrínsecamente, pues, este problema se puede considerar fácil de resolver puesto que lo hemos podido transformar en otro problema del que conocíamos un método de resolución eficiente.

Número combinatorio

Recordad que el número $\binom{n}{r}$ se denomina **número combinatorio** o **binomial**. Se puede calcular con la fórmula, $\binom{n}{r} = \frac{n!}{r!(n-r)!}$.
El número $\binom{n}{r}$ nos indica la cantidad de maneras de seleccionar r elementos de un conjunto que tiene n. Alternativamente, el número $\binom{n}{r}$ indica la cantidad de r-muestras no ordenadas sin repetición que se pueden construir en un conjunto de n elementos.

Ejemplo 14

Un problema elemental en lógica binaria es el de encontrar alguna solución a una expresión booleana de la forma

$$(x_1 \vee \overline{x}_2) \wedge (\overline{x}_3 \vee x_4) \ldots = 1$$

donde las variables x_i toman valores 0 o 1. Por ejemplo, la expresión

$$(x_1 \vee \overline{x}_2) \wedge x_3 = 1$$

tiene como solución $x_1 = 1$, $x_2 = 0$ y $x_3 = 1$.

Aunque este problema es muy parecido al del ejemplo anterior, no se conoce ninguna transformación que nos permita saber, de una manera eficiente, si una ecuación como la anterior tiene solución, y en caso afirmativo, cuántas. El único método efectivo consiste en buscar las soluciones de manera exhaustiva.

Operadores lógicos

El operador $\vee$ representa la O lógica, mientras que el operador $\wedge$ representa la Y lógica. Además, la notación $\overline{x}$ se utiliza para indicar el valor contrario al de x, es decir, $x = 0 \Leftrightarrow \overline{x} = 1$.

Si la ecuación tiene pocas variables, entonces ocurre que el espacio de posibles soluciones es reducido y se puede encontrar la solución (en caso de que la haya) con pocos recursos. El espacio de posibles soluciones de la ecuación anterior contiene 2^3 elementos (todas las 3-muestras ordenadas con repetición que se pueden hacer con 2 elementos).

Si el número de variables es muy elevado entonces es prácticamente imposible buscar la solución de manera exhaustiva. Para una ecuación con cien variables, el espacio de posibles soluciones contiene 2^{100} elementos. Este problema se considera intrínsecamente difícil.

De manera general podríamos decir que la dificultad de un problema depende de aspectos inherentes al mismo (espacio de búsqueda, modelización, etc.) y de otros aspectos que dependen de la capacidad para encontrar un buen método o algoritmo de resolución.

Tanto en un caso como en otro, sería interesante disponer de una "medida" de su dificultad. En los problemas que se pueden resolver mediante un algoritmo, se utiliza el concepto de *eficiencia* (cantidad de recursos que utiliza un algoritmo) como medida de su dificultad.

Eficiencia de un algoritmo

Observad que la eficiencia de un algoritmo es una medida de su dificultad centrada en la cantidad de recursos que utiliza el algoritmo.

Ved también

En los subapartados siguientes intentaremos precisar más el concepto de *eficiencia* que aplicaremos repetidamente en los siguientes módulos.

Ejercicios

22. Supongamos que queremos encontrar, de manera exhaustiva, todas las soluciones enteras, no negativas, de la ecuación $x_1 + x_2 + \cdots + x_r = n$, ¿cuál es el cardinal del conjunto de todas las posibles soluciones? ¿Cuántas soluciones enteras no negativas tiene esta ecuación?

23. Calcular el cardinal del conjunto de todas las soluciones posibles de la ecuación booleana

$$x \wedge \overline{x} \wedge y \wedge z = 1$$

y detallar todas las soluciones.

24. Consideremos una ecuación booleana con cien variables y supongamos que un ordenador es capaz de verificar mil posibles soluciones por segundo. Calcular el tiempo necesario para verificar las posibles soluciones de la ecuación.

Soluciones

22. Cada variable puede tomar valores x_i, donde $0 \leq x_i \leq n$. Así, el conjunto de posibles soluciones tendrá cardinal $(n+1)^r$. El número de soluciones, tal como hemos visto en el ejemplo 13, será $\binom{n+r-1}{r-1}$.

23. El cardinal del conjunto de soluciones es $2^3 = 8$ pero no habrá ninguna solución, ya que si $x = 1$ necesariamente $\overline{x} = 0$.

24. El espacio de posibles soluciones tiene $2^{100} \cong 10^{30}$ elementos. Si en cada segundo verificamos mil soluciones, necesitaríamos

$$\frac{10^{30}}{10^3} = 10^{27} \text{ segundos} = 0.3 \cdot 10^{20} \text{ años}$$

La edad del universo es aproximadamente $15 \cdot 10^9$ años.

2.2. Introducción al análisis de algoritmos

Definición 12

Un **algoritmo** es una secuencia de instrucciones, claramente especificadas, que hay que seguir para resolver un problema.

Algoritmo

El término *algoritmo* proviene originalmente del matemático y científico Al-Khoresmi (780 dC. Khoreme, actual Khiva, Uzbekistán), considerado uno de los padres del álgebra.

Una vez definido un algoritmo, para resolver un problema es importante determinar la cantidad de recursos (como el tiempo o el almacenamiento) que el algoritmo consume.

Si un algoritmo tarda un año en resolver un problema, seguramente no será de mucha utilidad. Del mismo modo, si un algoritmo necesita un petabyte de almacenamiento, seguramente tampoco será muy útil.

Para un problema concreto se pueden encontrar diferentes algoritmos que lo resuelvan. Naturalmente, interesa escoger aquel que sea más *eficiente*, es decir, aquel que utilice menos recursos.

Ejemplo 15

Los siguientes algoritmos intercambian el contenido de dos variables:

algoritmo *Intercambio1*(*a*,*b*)
inicio
$a \leftarrow a + b$
$b \leftarrow a - b$
$a \leftarrow a - b$
fin

algoritmo *Intercambio2*(*a*,*b*)
inicio
$c \leftarrow a$
$a \leftarrow b$
$b \leftarrow c$
fin

Recursos utilizados por cada algoritmo:

Intercambio1	Intercambio2
2 variables	3 variables
3 asignaciones	3 asignaciones
3 operaciones aritméticas	

Podemos concluir que el primer algoritmo utiliza más recursos temporales (hace más operaciones) y el segundo más recursos espaciales (más memoria).

Unidades

Según el estándar ISO/IEC 80000, las unidades utilizadas para describir el almacenamiento de información son el byte (8 bits), el kilobyte (10^3 bytes), el megabyte (10^3 kilobytes), el gigabyte (10^3 megabytes), el terabyte (10^3 gigabytes), el petabyte (10^3 terabytes), el exabyte, el zettabyte, el yottabyte, etc.
Si se quiere utilizar el sistema binario y referirse a $2^{10} = 1.024$ bytes, debe decirse kibibyte o mebibyte (2^{20} bytes), gibibyte (2^{30} bytes), tebibyte (2^{40} bytes), pebibyte (2^{50} bytes), exbibyte (2^{60} bytes), zebibyte (2^{70} bytes), yobibyte (2^{80} bytes)

Esta idea de eficiencia, interpretada como cantidad de recursos, nos permitirá distinguir entre problemas fáciles y problemas difíciles de una manera objetiva. Es importante, antes de empezar, ver cómo podemos calcular la cantidad de recursos utilizados por un algoritmo.

Nos centraremos principalmente en el estudio de los recursos temporales de un algoritmo, es decir, en el tiempo de ejecución del algoritmo. Dado que este tiempo de ejecución depende del sistema de cálculo (ordenador) utilizado, supondremos, para simplificar el análisis, que todas las operaciones elementales tardan el mismo tiempo*.

***En el próximo subapartado veremos que esta simplificación no representa ninguna debilidad en el análisis.**

En lo que queda de este subapartado, cuando hagamos referencia al coste de un algoritmo entenderemos, por defecto, que nos referimos al coste temporal.

El ejemplo siguiente muestra los principios del análisis del tiempo de ejecución de un algoritmo.

Ejemplo 16

Este algoritmo calcula la suma de los n ($n \geq 1$) primeros números naturales.

```
1 función Suma(n)
2   inicio
3     parcial ← 0
4     para i ← 1 hasta n
5           parcial ← parcial + i
6     finpara
7     retorno (parcial)
8   fin
```

Las líneas 1, 2 y 8 forman parte de la declaración del algoritmo y se considera que no consumen recursos. Las líneas 3 y 7 consumen una unidad de tiempo cada una (según nuestra suposición que todas las operaciones elementales tardan lo mismo). La línea 5 consta de una suma y una asignación, y se ejecuta n veces. En total tarda $2n$ unidades de tiempo. Finalmente, la línea 4 consta de una inicialización, $n+1$ comparaciones ($i \leq n$) y n incrementos de la variable i. En total, $2n+2$ unidades de tiempo. Sumando el tiempo que se tarda en ejecutar todo el algoritmo, obtenemos un total de $2 + 2n + 2n + 2 = 4n + 4$ unidades de tiempo. Diremos, también, que el cálculo de la suma de los n primeros números naturales es un problema que tiene un *coste* $T(n) = 4n + 4$ usando el algoritmo propuesto.

Así, pues, en general podemos definir el coste de un algoritmo de la siguiente manera.

Coste de un algoritmo

El coste de un algoritmo es una función no decreciente entre números naturales, que depende del *tamaño* del problema que se quiere resolver. El objetivo del análisis es estudiar la variación del coste de un problema cuando varía su tamaño.

Definición 13

El **coste** de un algoritmo es una función $T : \mathbb{N} \to \mathbb{N}$, no decreciente, que calcula la cantidad de recursos (temporales y espaciales) que utiliza el algoritmo.

Podemos destacar que:

- El coste de un algoritmo (que incluye el tiempo de ejecución, el espacio de memoria utilizado, etc.) es una función no decreciente, ya que no es posible que el tiempo de ejecución o la memoria disminuya cuando aumenta el número de pasos. En el ejemplo anterior, $T(n) = 4n + 4$.
- La función T es una función que depende del *tamaño* del problema que se quiere resolver (en el ejemplo, la cantidad n de números naturales que queremos sumar).
- El objetivo de este análisis consiste en estudiar cómo varía el coste asociado a la resolución de un problema cuando varía su tamaño. En nuestro ejemplo, si el tamaño se duplica, entonces el coste también se duplica. Es decir, hay una dependencia *lineal* entre el coste de la solución y el tamaño del problema.

A veces no es necesario describir completamente un algoritmo para calcular (de manera aproximada) el coste asociado a la resolución de un problema. Veámoslo con un ejemplo.

Ejemplo 17

Para calcular una potencia entera y positiva, x^n, de un número necesitamos hacer $n-1$ multiplicaciones utilizando el método elemental que consiste en multiplicar x por x, el resultado por x, y así sucesivamente hasta obtener x^n.

Así, podemos decir que este algoritmo elemental, para calcular x^n, necesitaría $n-1$ unidades de tiempo o que el problema de calcular la potencia x^n con este algoritmo tiene un coste $T(n) = n-1$. Por ejemplo, el cálculo de 2^{62} tendría un coste de 61 con este algoritmo.

¿Podemos calcular esta potencia, x^n, con menos multiplicaciones? Observar que x^5 se puede calcular como $x \cdot x \cdot x \cdot x \cdot x$ o como $x^2 \cdot x^2 \cdot x$. En el primer caso, necesitamos cuatro multiplicaciones, y en el segundo sólo tres. Para calcular x^{62} podríamos proceder de la manera siguiente:

$$\begin{aligned} x^{62} &= x^{31} \cdot x^{31} \\ x^{31} &= x^{30} \cdot x \\ x^{30} &= x^{15} \cdot x^{15} \\ x^{15} &= x^{14} \cdot x \\ x^{14} &= x^{7} \cdot x^{7} \\ x^{7} &= x^{6} \cdot x \\ x^{6} &= x^{3} \cdot x^{3} \\ x^{3} &= x^{2} \cdot x \\ x^{2} &= x \cdot x \end{aligned}$$

En total hacemos nueve multiplicaciones en lugar de las sesenta y una iniciales. Llamaremos a esta técnica el método de *multiplicar y elevar*.

Para contar el número de multiplicaciones en función de n, tenemos que distinguir dos tipos de valores: si n es par hay una multiplicación de la forma $x^{n/2} \cdot x^{n/2}$; si n es impar hay una multiplicación de la forma $x^{n-1} \cdot x$ y otra de la forma $x^{(n-1)/2} \cdot x^{(n-1)/2}$. Así, el número mínimo de multiplicaciones se obtendrá cuando n sea par y potencia de 2. El coste será $T_{min}(n) = \log_2 n$. El número máximo de multiplicaciones se obtendrá cuando en cada paso n sea impar, es decir, cuando n sea de la forma $2^k - 1$. En este caso, $T_{max}(n) = 2\lfloor \log_2 n \rfloor$.

Logaritmos

Recordad que $\log_a n = x$ significa que $a^x = n$. El valor a se denomina **base** del logaritmo y en esta asignatura utilizaremos principalmente el logaritmo en base 2 ($\log_2$).

Este ejemplo pone de manifiesto que:

- Para calcular el coste de un problema no hace falta describir todas las operaciones de un algoritmo. A menudo, será suficiente calcular el coste de las operaciones que consumen más recursos aunque el valor que se obtenga no sea aplicable en todos los casos. En el ejemplo, esta operación es la multiplicación.
- El coste de un algoritmo no siempre se puede expresar de una forma unívoca. Sin embargo, siempre se pueden calcular los valores máximo y mínimo de la función de coste y los valores de entrada para los cuales estos valores máximo y mínimo se alcanzan. En el ejemplo, el valor mínimo (*mejor caso*) es $\log_2 n$ que se

logra cuando n es una potencia de 2 y el valor máximo (*peor caso*) es $2\lfloor \log_2 n \rfloor$ que se logra cuando n es una potencia de 2 menos 1.

Acabaremos este subapartado con otro ejemplo de cálculo del coste de un algoritmo.

Ejemplo 18

Uno de los métodos básicos de clasificación es el *método de clasificación por intercambio (burbuja)*. En este método una secuencia de n números almacenados en una tabla se ordenan de menor a mayor, comparando pares de números consecutivos. Por ejemplo, para ordenar la secuencia 4,2,3,1 procedíriamos a comparar la primera posición con la segunda, o sea el 4 con el 2. Puesto que el 2 es más pequeño que el 4, intercambiaremos las posiciones del 4 y del 2. Nos quedará una nueva secuencia 2,4,3,1. A continuación, compararemos la segunda y la tercera posición, o sea el 4 con el 3 y, de nuevo, procediremos a intercambiar el 4 y el 3. Finalmente, compararemos la tercera con la cuarta posición, o sea el 4 con el 1 y también los intercambiaremos:

$$\underbrace{4,2}\,,3,1 \rightarrow 2,4,3,1$$
$$2,\underbrace{4,3}\,,1 \rightarrow 2,3,4,1$$
$$2,3,\underbrace{4,1} \rightarrow 2,3,1,4$$

Al final de este proceso (*paso*) tendremos el 4 correctamente situado después de haber hecho tres comparaciones y tres intercambios. Si repetimos este proceso dos veces más (dos pasos más) conseguiremos ordenar toda la secuencia inicial. Observar que, en cada paso, el número de comparaciones disminuye en una unidad y que no siempre una comparación da lugar a un intercambio.

Procederemos a calcular el número total, $T(n)$, de comparaciones que hemos hecho (es la operación central del algoritmo) para ordenar n números. Llamemos $T(n)$ al número total de comparaciones que se necesitan para ordenar los n números. Tras $n-1$ comparaciones habremos situado el valor más grande al final de la secuencia y nos quedará una secuencia de $n-1$ números para ordenar en el paso siguiente. La ordenación de esta secuencia necesitará $T(n-1)$ comparaciones. Así, podemos escribir la ecuación,

$$T(n) = T(n-1) + n - 1, \quad (n \geq 1)$$

además, $T(1) = 0$ puesto que con un solo elemento no hace falta hacer ninguna comparación.

Ecuaciones recurrentes

Las ecuaciones del tipo $x_n = ax_{n-1} + f(n)$ se denominan **ecuaciones recurrentes lineales** de primer orden.

Esta es una ecuación recurrente lineal de primer orden que se puede resolver de la siguiente manera. Substituyendo $T(n–1)$ por su valor, $T(n–2)+n–2$, etc., llegamos a la expresión: $T(n) = 1 + 2 + 3 + \cdots + (n-1)$. El resultado de esta suma vale

$$T(n) = \frac{n(n-1)}{2}$$

tal como calcularemos en un próximo ejercicio.

Este resultado indica que el coste de ordenar una secuencia de n números enteros por el método de la burbuja es $n(n-1)/2$.

Ejercicios

25. ¿Cuál de las dos versiones del algoritmo de intercambio del ejemplo 15 parece más adecuada para intercambiar el contenido de dos variables cualesquiera?

26. ¿Cuántas unidades de tiempo se precisan para sumar los sesenta primeros números naturales utilizando el algoritmo del ejemplo 16? ¿Sabríais escribir un algoritmo más eficiente para calcular la suma de los n ($n \geq 1$) primeros números naturales?

27. Calcular el número de multiplicaciones necesarias para hallar el valor de 2^{11}, 2^{15}, 2^{16}, 2^{20} siguiendo el algoritmo de *multiplicar y elevar* del ejemplo 17. Comprobar, en cada caso, que este número de multiplicaciones tiene como cota inferior y superior $\lfloor \log_2 n \rfloor$ y $2\lfloor \log_2 n \rfloor$, respectivamente.

28. Hacer una tabla comparativa del número de multiplicaciones necesarias para calcular una potencia siguiendo los dos métodos del ejemplo 17. Tomar los valores de $n = 1,2,3,4,5,6$. Deducir para qué valores de n es preferible utilizar cada uno de los métodos.

29. Demostrar por inducción que la solución de la ecuación recurrente $T(n) = T(n-1) + n - 1$ con la condición inicial $T(1) = 0$ es $T(n) = \frac{n(n-1)}{2}$.

Soluciones

25. Siguiendo el ejemplo 16, la primera versión consume 2 unidades de almacenamiento y tiene un coste temporal $T_1 = 6$. La segunda versión consume 3 unidades de almacenamiento y tiene un coste temporal $T_2 = 3$. Si queremos ahorrar recursos de almacenamiento escogeríamos la primera versión. Si quisiéramos ahorrar tiempo elegiríamos la segunda versión.

Hay otras diferencias que se pueden considerar para elegir la versión más adecuada, la primera versión sólo se puede utilizar para tipos numéricos de datos. La segunda versión se puede utilizar para cualquier tipo simple de datos. Por lo tanto, la segunda versión se puede considerar más general que la primera.

26. Para sumar los sesenta primeros números naturales necesitaríamos $4n+4 = 4 \cdot 60 + 4 = 244$ unidades de tiempo.

Carl Friedrich Gauss (1777-1855), a la temprana edad de 7 años, obtuvo la suma de uno hasta sesenta de manera inmediata al observar que $1 + 60 = 2+59 = \ldots = 30+31$ y podía calcular la suma como $30 \cdot (30+31) = 1.830$. Había descubierto la conocida fórmula:

$$1 + \cdots + n = \frac{(n+1)n}{2}$$

El siguiente algoritmo

```
función Suma(n)
  inicio
    parcial ← n · (n + 1)/2
    retorno (parcial)
  fin
```

calcula la suma de los n primeros números naturales en cuatro unidades de tiempo (cinco si contamos el **retorno**).

27. La siguiente tabla resume el resultado

n	$\lfloor \log_2 n \rfloor$	*multiplicar y elevar*	$2\lfloor \log_2 n \rfloor$
11	3	5	6
15	3	6	6
16	4	4	8
20	4	5	8

28. La tabla sería

n	multiplicar	*multiplicar y elevar*
1	0	0
2	1	1
3	2	2
4	3	2
5	4	3
6	5	3

y demuestra que para $n \leq 3$ es indiferente utilizar uno u otro método. A partir de $n = 4$ es mejor el método de multiplicar y elevar.

29. Para $n = 1$ se tiene $T(1) = 0$ y la hipótesis es correcta.

Consideremos la fórmula válida hasta $n = k - 1$. Entonces,

$$\begin{aligned} T(k) &= T(k-1) + (k-1) && \text{igualdad válida utilizando la ecuación de partida} \\ &= \frac{(k-1)(k-2)}{2} + (k-1) && \text{hipótesis de inducción} \\ &= (k-1)\left(\frac{k-2}{2} + 1\right) \\ &= \frac{k(k-1)}{2} \end{aligned}$$

de manera que la hipótesis era correcta.

Por lo tanto, $T(n) = \frac{n(n-1)}{2}$.

2.3. La notación *O*

Observación

Es muy importante conocer la evolución del número de operaciones que hace un algoritmo, a medida que aumenta el tamaño del problema.

En el análisis de un algoritmo es importante saber exactamente cuántas operaciones hace este algoritmo, pero todavía es más importante saber cómo evoluciona este número de operaciones cuando aumenta el tamaño del problema.

En el algoritmo del ejemplo 16 habíamos visto que el número de operaciones necesarias para sumar los n primeros números naturales era $4n + 4$. En este caso, podemos decir, simplemente, que el tiempo de ejecución (coste) es proporcional a n o que crece linealmente con n. En cambio, si aplicamos directamente la fórmula de la suma (ejercicio 26) para sumar los n primeros números naturales, entonces el número de operaciones es constante (exactamente 4) independientemente del valor de n.

El algoritmo de multiplicar y elevar (ejemplo 17) es un poco más difícil de analizar dado que el número de operaciones no depende únicamente del tamaño. En estas situaciones, el coste toma el valor del peor caso. Así, en el ejemplo, podemos decir que el coste será proporcional a $\lfloor \log_2 n \rfloor$ o que crece como el logaritmo de n.

Finalmente, el algoritmo de clasificación por el método de la burbuja (ejemplo 18) tiene un coste

$$\frac{(n-1)n}{2} = \frac{1}{2}(n^2 - n)$$

y diremos que su coste es proporcional a n^2 o que crece cuadráticamente.

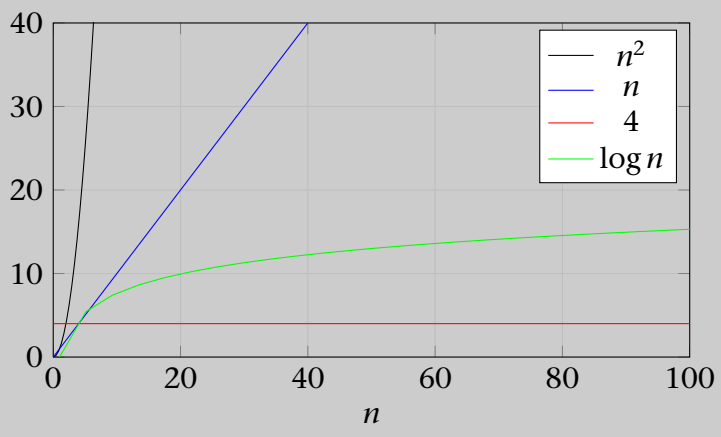
Definición 14

La **tasa de crecimiento** de la función de coste es el término de la función que crece más rápidamente a medida que aumenta el tamaño, n, del problema. La tasa de crecimiento nos da el orden de magnitud de la complejidad de un problema.

La figura siguiente nos muestra el comportamiento de algunas tasas de crecimiento asociadas a las funciones de coste estudiadas anteriormente. Podemos destacar que, conforme n aumenta, los gráficos se separan y la diferencia entre éstos se hace mayor. Así, podemos decir que la función $\frac{1}{2}(n^2-n)$ es la que tiene mayor *tasa de crecimiento* (n^2). A continuación viene la función $4n + 4$, después la función $\lfloor \log_2 n \rfloor$ y, finalmente, la función constante 4 es la que tiene menor tasa de crecimiento.

Complejidad logarítmica

Observad que las características especiales de la función logarítmica hacen deseable buscar algoritmos de complejidad logarítmica para resolver problemas.



La tabla siguiente muestra, numéricamente, diversas tasas de crecimiento. Observar que para valores pequeños de n las diferencias son relativamente pequeñas, pero, conforme n aumenta, las diferencias se hacen mayores. En particular, la columna de 2^n es la que representa la mayor tasa de crecimiento.

$\log n$	n	n^2	2^n
0	1	1	2
1	2	4	4
2.3	5	25	32
3.3	10	100	1024
3.9	15	225	32768
4.3	20	400	1048576
5.6	50	2500	1125899906842620
6.6	100	10000	1267650600228230000000000000000

Desde el punto de vista del análisis de los algoritmos, todas las funciones que tienen la misma tasa de crecimiento se consideran equivalentes. Las funciones n, $3n+2$, $n-1000$ tienen todas la misma tasa de crecimiento y diremos que son del **orden** de n o que tienen una **complejidad del orden de** n. El conjunto de todas las funciones del orden de n se representa por $O(n)$.

Por ejemplo, si el coste temporal de un cierto algoritmo es $T(n) = 4n^2 + n + 1$ diremos que la tasa de crecimiento de esta función es del orden de n^2 o que la complejidad computacional es $O(n^2)$.

Nota

Observad que las expresiones "tasa de crecimiento", "orden" y "complejidad de orden" son prácticamente equivalentes.

Definición 15

Un algoritmo se dice que tiene una **complejidad del orden de** f, y se escribe $O(f)$, si el coste (temporal, espacial, etc.) del algoritmo es una función que tiene la misma tasa de crecimiento que f.

Ejemplo 19

1) El algoritmo para sumar los n primeros números naturales tiene una complejidad $O(n)$ si utilizamos el algoritmo del ejemplo 16. Si aplicamos la fórmula de Gauss (algoritmo del ejercicio 26), entonces tendrá una complejidad constante que se representa por $O(1)$.

2) Para calcular una potencia de exponente entero, el algoritmo elemental (ejemplo 17) tiene una complejidad lineal, $O(n)$, mientras que el algoritmo de multiplicar y elevar (ejemplo 17) tiene una complejidad logarítmica, $O(\log n)$.

3) Finalmente, el algoritmo de clasificación por intercambio tiene una complejidad cuadrática, puesto que tiene una complejidad del orden de n^2, o sea, $O(n^2)$.

Observemos que $O(f)$ es un conjunto de funciones. Exactamente, contiene todas aquellas funciones que tienen la misma tasa de crecimiento que f. Por ejemplo, $O(n^2)$ contendrá las funciones n^2, n^2+1, $3n^2+n$, $-n^2+2n-1$, etc. Así, podemos escribir $n^2+1 \in O(n^2)$.

Calcular la complejidad de un algoritmo a partir del coste de las operaciones elementales (como hemos hecho en el ejemplo 16) puede ser una tarea complicada si el algoritmo conlleva un gran número de operaciones. Por suerte, hay algunas reglas que nos pueden facilitar este cálculo.

Teorema 5

1) Si el coste de un algoritmo es constante, es decir, $T(n) = k$, $k \in \mathbb{N}$ entonces el algoritmo tiene una complejidad $O(1)$.

2) Si el coste de un algoritmo es un polinomio de grado $k \in \mathbb{N}$ en n, o sea $T(n) = a_k n^k + \cdots + a_1 n + a_0$, entonces el algoritmo tiene una complejidad $O(n^k)$.

3) Si $T_1(n) \in O(f(n))$ y $T_2(n) \in O(g(n))$, entonces $T_1(n) + T_2(n) \in \max\{O(f(n)), O(g(n))\}$.

4) Si $T_1(n) \in O(f(n))$ y $T_2(n) \in O(g(n))$, entonces $T_1(n) \cdot T_2(n) \in O(f(n) \cdot g(n))$.

Demostración: Consultad Cormen y otros (2001). ∎

Estas reglas para calcular el coste total de un algoritmo son bastante intuitivas. Si varias instrucciones del programa se ejecutan secuencialmente, el coste total será la suma del coste de cada una de ellas. Por otro lado, si se trata de una instrucción condicional su coste será el coste de evaluar la condición más el máximo de los costes asociados a la ejecución de cada una de las ramas. Finalmente, para contabilizar el coste de las instrucciones que se hallen dentro de un bucle, se deberá multiplicar su coste por el número de iteraciones del bucle y añadir el coste de la condición de final de bucle.

Ejemplo 20

Dados dos números enteros x, n, el algoritmo siguiente calcula $x^n/n!$.

```
1 función potencia (x,n)
2   inicio
3     pot ← multiplicar_elevar(x,n)
4     fact ← factorial(n)
5     retorno (pot/fact)
6   fin
```

Sólo tenemos que analizar las líneas 3, 4 y 5. En la línea 3 llamamos al algoritmo de multiplicar y elevar que ya sabemos que tiene una complejidad $O(\log n)$. La asignación de esta línea es constante y, por lo tanto, tiene una complejidad $O(1)$ (regla 1). Ya que primero se ejecuta el algoritmo de multiplicar y elevar y, a continuación, la asignación, podemos aplicar la regla 3 y obtenemos, para la línea 3, una complejidad total $O(\log n)$.

En la línea 4 tenemos una función y una asignación. El cálculo del factorial de un número entero n necesita de $n-1$ multiplicaciones. Por lo tanto, tendrá una complejidad $O(n)$ ya que $n-1$ es un polinomio de grado 1 (regla 2). La asignación tiene una complejidad $O(1)$. En total, la línea 4 tiene una complejidad $O(n)$ (regla 3).

Finalmente, la línea 5 consta de una división y una operación de retorno del algoritmo. Las dos juntas se ejecutan una única vez. Si aplicamos la regla 1 obtenemos una complejidad $O(1)$.

En resumen, tenemos tres líneas que se ejecutan secuencialmente con complejidades $O(\log n)$, $O(n)$ y $O(1)$. Si aplicamos la regla 3 obtenemos una complejidad para todo el algoritmo de $O(n)$, puesto que $f(n) = n$ es la función que tiene la mayor tasa de crecimiento de las tres.

Ejercicios

30. Ordenar las funciones siguientes según su tasa de crecimiento (utilizar, si hace falta, una calculadora):

$$2^n, \quad \log n, \quad n^2, \quad (\log n)^2, \quad n^3, \quad \sqrt{n}, \quad n!, \quad n\log n, \quad 6, \quad n$$

31. Para cada una de las funciones siguientes indicar a qué orden de complejidad pertenecen (utilizar, si hace falta, las reglas de cálculo de la complejidad):

$$n + n^5 - 5n^3, \quad 5^n + n^5, \quad 5 + 10^6, \quad \sqrt{n^4 - 2}, \quad n^2 \cdot (\log n + n)$$

32. Calcular la complejidad de cada uno de los fragmentos de algoritmo siguientes:

a)
```
1 suma ← 0
2 para i ← 1 hasta n
3       suma ← suma + 1
4 finpara
```

b)
```
1 suma ← 0
2 para i ← 1 hasta n
3       para j ← 1 hasta n
4             suma ← suma + 1
5       finpara
6 finpara
```

c)
```
1 suma ← 0
2 para i ← 1 hasta n
3       para j ← 1 hasta n^2
4             suma ← suma + 1
5       finpara
6 finpara
```

Soluciones

30. Utilizaremos el símbolo '$\ll$' para indicar que la tasa de crecimiento es menor:

$$6 \ll \log n \ll (\log n)^2 \ll \sqrt{n} \ll n \ll n\log n \ll n^2 \ll n^3 \ll 2^n \ll n!$$

31. $n + n^5 - 5n^3 \in O(n^5)$

$5^n + n^5 \in O(5^n)$

$5 + 10^6 \in O(1)$

$\sqrt{n^4 - 2} \in O(n^2)$

$n^2 \cdot (\log n + n) \in O(n^3)$

32. La inicialización de la variable *suma* tiene un coste 1 y, por lo tanto, una complejidad $O(1)$. La instrucción *suma* ← *suma*+1 tiene un coste de 2 unidades que, también, es $O(1)$. Así, la complejidad en cada caso depende de los bucles **para**.

a) La línea 2 tiene una complejidad $O(n)$. Como la línea 3 tiene una complejidad $O(1)$, si aplicamos la regla 4 obtenemos una complejidad $O(n \cdot 1) = O(n)$ para todo el bucle **para**. Si aplicamos la regla 3 obtenemos una complejidad $\max\{O(1),O(n)\} = O(n)$ para todo el algoritmo.

b) La línea 2 tiene una complejidad $O(n)$, la línea 3 tiene una complejidad $O(n)$. Si aplicamos la regla 4 y la 3 obtenemos, para todo el algoritmo $\max\{O(1),O(n \cdot n \cdot 1)\} = O(n^2)$.

c) La línea 2 tiene una complejidad $O(n)$ y la línea 3, $O(n^2)$. Si aplicamos la regla 3, todo el algoritmo tendrà una complejidad, $\max\{O(1),O(n \cdot n^2 \cdot 1)\} = O(n^3)$.

Los resultados obtenidos sobre la complejidad de un algoritmo nos permitirán precisar la idea de problema intrínsecamente "difícil". De momento, de forma breve, podemos decir que se consideran "fáciles" todos aquellos problemas que se pueden resolver mediante algoritmos de complejidad polinómica. Diremos que estos algoritmos son *eficientes*.

Ved también

Estas ideas sobre problemas fáciles y difíciles se desarrollarán en el módulo "Complejidad computacional".

Algoritmo eficiente

Recordad que calificamos de eficiente un algoritmo que tiene una complejidad polinómica (y el problema asociado se considerará "fàcil"). Por otro lado, dados dos algoritmos, el más eficiente es el que consume menos recursos (coste menor).

Ejemplo 21

La mayoría de problemas que hemos analizado en este apartado 2 (suma de los n primeros números naturales, cálculo de la potencia entera de un número, clasificación de una secuencia de números, etc.) se pueden considerar "fáciles", puesto que para todos ellos hemos sabido encontrar un algoritmo que tiene una complejidad polinómica.

En cambio, el problema del ejemplo 14, llamado *SAT*, es un problema que se considera intrínsecamente "difícil". Para resolver este problema con n variables tendríamos que probar todas las posibilidades, y un algoritmo

que lo hiciera así tendría una complejidad $O(2^n)$, que no es polinómico. No se conoce ningún algoritmo que pueda resolver este problema en un tiempo polinómico. En cambio, si en lugar de considerar el problema genérico sólo consideramos una instancia concreta del mismo, por ejemplo $x_1 = 0$, $x_2 = 1$, ..., $x_n = 0$, entonces es fácil (problema de complejidad polinómica) resolver el problema y saber si esta instancia lo satisface o no.

Más adelante se verán ejemplos de problemas complejos con costes no polinómicos del estilo $O(2^n)$, $O(n!)$ o $O(n^n)$.

Ejercicios de autoevaluación

1. Encontrar todas las funciones de $X = \{1,2,3\}$ en $Y = \{a,b\}$. Clasificarlas en dos clases: exhaustivas y no exhaustivas.

2. Encontrar todas las funciones de $X = \{1,2\}$ en $Y = \{a,b,c\}$. Clasificarlas en dos clases: inyectivas y no inyectivas.

3. Construir una función inyectiva de $\mathbb{N}$ en el conjunto $X = \{2,4,6,8,10,\ldots\}$ que nos demuestre que X es infinito. ¿Es biyectiva la función que habéis construido?

4. Demostrar que el conjunto $\mathbb{Z}$ de los números enteros es numerable a partir de la función

$$f(x) = \begin{cases} \frac{x}{2} & \text{si } x \text{ es par} \\ \frac{-(x-1)}{2} & \text{si } x \text{ es impar} \end{cases}$$

5. Sabiendo que un byte es una palabra binaria de longitud 8, ¿cuántos bytes diferentes hay? ¿Cuántos contienen al menos dos unos?

6. ¿De cuántas maneras se puede responder un test de quince preguntas si cada pregunta está formulada en términos de cierto-falso?

7. Una comisión de diez personas ha de elegir un presidente, un secretario y un tesorero. Suponiendo que cada persona puede ostentar un cargo como máximo, ¿de cuántas maneras se pueden elegir los tres cargos?

8. Demostrar que el número de maneras diferentes de colocar cuatro personas alrededor de una mesa circular es seis. ¿Y si hemos de colocar n personas?

9. ¿Cuántas funciones hay del conjunto $A = \{x,y,z,t\}$ en un conjunto B de siete elementos? ¿Cuántas de estas funciones son inyectivas? ¿Y biyectivas?

10. Calcular la complejidad de cada uno de los fragmentos de algoritmo siguientes:

a)
```
1 suma ← 0
2 para i ← 1 hasta n
3       para j ← 1 hasta i
4             suma ← suma + 1
5       finpara
6 finpara
```

b)
```
1 suma ← 0
2 para i ← 1 hasta n
3       para j ← 1 hasta i^2
4             suma ← suma + 1
5       finpara
6 finpara
```

c)
```
1 suma ← 0
2 para i ← 1 hasta n
3       para j ← 1 hasta i^2
4             si j mod i ≠ 0
5                entonces suma ← suma + 1
6             finsi
7       finpara
8 finpara
```

Operador módulo

Recordad que el operador módulo (a mod b) calcula el residuo de la división de a por b. Por ejemplo, $7 \bmod 5 = 2$. Si $a \bmod b = 0$, entonces a es múltiplo de b.

11. Los problemas que se pueden resolver mediante algoritmos de complejidad logarítmica son considerados muy eficientes. Esto se debe a las propiedades particulares de la función logaritmo.

a) Demostrar que todas las funciones logarítmicas, $\log_a n$, tienen la misma tasa de crecimiento. Es decir, $\log_a n \in O(\log n)$ para cualquier base de logaritmos que tomemos.

b) Demostrar que la tasa de crecimiento del logaritmo de cualquier función potencial, $\log(n^a)$ ($a \in \mathbb{N}$), es la misma que la del $\log n$, es decir, $\log(n^a) \in O(\log n)$.

12. Las características especiales de la función logarítmica hacen deseable buscar algoritmos de complejidad logarítmica para resolver problemas.

Consideremos una secuencia $a_1, a_2, \ldots, a_n$ de números enteros, donde queremos buscar una determinada información b. La manera usual de proceder consiste en comparar consecutivamente b con cada elemento de la secuencia a_i, $i = 1,2,\ldots$ hasta encontrarlo. ¿Cuál será el número mínimo y el máximo de comparaciones que habrá que hacer para encontrar el número b dentro la secuencia? ¿Cuál es la complejidad de este algoritmo?

Si la secuencia está previamente ordenada, $a_1 \leq a_2 \leq \ldots \leq a_n$, entonces podemos proceder de la manera siguiente: comparamos b con $a_{\lfloor n/2 \rfloor}$. Si $b = a_{\lfloor n/2 \rfloor}$ hemos acabado la búsqueda. De lo contrario, si $b < a_{\lfloor n/2 \rfloor}$ seguimos la búsqueda en la subsecuencia $a_1, \ldots, a_{\lfloor n/2 \rfloor - 1}$. Si $b > a_{\lfloor n/2 \rfloor}$ continuaremos la búsqueda en la subsecuencia $a_{\lfloor n/2 \rfloor + 1}, \ldots, a_n$. ¿Cuál será, en este caso, el número mínimo y máximo de comparaciones que habrá que hacer? ¿Cuál será la complejidad de este algoritmo?

13. Para calcular la potencia entera x^n de un número, conocemos un algoritmo muy eficiente que se llama algoritmo de multiplicar y elevar (ver el ejemplo 17).

a) ¿Cuántas multiplicaciones hacen falta para calcular x^8, x^{11}, x^{14}, x^{15}?

b) Obtener la representación binaria de los números enteros 8, 11, 14 y 15.

c) ¿Qué relación hay entre la representación binaria obtenida en el punto anterior y el número de multiplicaciones calculadas en el primer punto?

d) Deducir, en función de n, una cota superior para el número de multiplicaciones necesarias para calcular x^n.

e) ¿Cuál es la complejidad del algoritmo de multiplicar y elevar?

14. Tal como hemos hecho en el ejemplo 18, determinar una ecuación recurrente lineal de primer orden que calcule el número mínimo de comparaciones que hacen falta para localizar el número más pequeño de un vector de n números. Deducir, iterativamente, la solución de la ecuación y demostrar –por inducción– que la solución es correcta.

15. Sea C un conjunto formado por $N = 2^n$ números enteros, donde $n \geq 1$. Responder las siguientes cuestiones, que nos van a permitir calcular la cantidad de comparaciones necesarias para determinar los valores máximo y mínimo en C:

a) Si $n = 1$, ¿cuántas comparaciones son necesarias para encontrar el máximo y el mínimo?

b) Si hacemos una partición de C en dos subconjuntos, $C = C_1 \cup C_2$ de manera que C_1 y C_2 contengan la mitad de los elementos de C y $T(n-1)$ es el número de comparaciones necesarias para encontrar el máximo y el mínimo en C_1 (y en C_2), encontrar una ecuación recurrente de primer orden que relacione $T(n)$ y $T(n-1)$.

c) Resolver la ecuación recurrente obtenida en el punto anterior y deducir la complejidad del algoritmo para determinar los valores máximo y mínimo en *C*.

Soluciones

1. De acuerdo con la notación introducida en el ejercicio 1 las funciones son: *aaa*, *bbb*, *aab*, *aba*, *baa*, *abb*, *bab*, *bba*.

La clase de las exhaustivas es: *aab*, *aba*, *baa*, *abb*, *bab*, *bba*. La clase de las no exhaustivas es: *aaa*, *bbb*.

2. Las funciones son: *aa*, *ab*, *ac*, *ba*, *bb*, *bc*, *ca*, *cb*, *cc*.

La clase de las inyectivas es: *ab*, *ac*, *ba*, *bc*, *ca*, *cb*. La clase de las no inyectivas es: *aa*, *bb*, *cc*.

3. La función f de $\mathbb{N}$ en $X = \{2,4,6,8,10,\ldots\}$ definida por $f(x) = 2x$ es inyectiva (sólo es necesario ver que $2x = 2y$ implica que $x = y$). Esto demuestra que el conjunto X es infinito.

Además, esta función es biyectiva (todo elemento de X tiene antiimagen: la antiimagen del 2 es 1, la antiimagen del 4 es 2, la antiimagen del 6 es 3, etc.) y, por lo tanto, podemos decir que el conjunto X es numerable.

4. Veamos que f es inyectiva. Empezamos observando que la imagen por f de un número natural par es un entero positivo y la imagen de un número natural impar es 0 o bien un entero negativo. Así, si suponemos $f(x) = f(y)$ deberán ser x,y pares o bien x,y impares y, para cualquiera de estos casos, se deduce $x = y$.

Falta ver que f es exhaustiva. Sea z un entero cualquiera. Si $z = 0$ entonces tiene antiimagen 1, si z es positivo entonces tiene antiimagen $2z$, y si z es negativo entonces tiene antiimagen $1 - 2z$.

Puesto que f es inyectiva y exhaustiva entonces será una función biyectiva y, por tanto, $\mathbb{Z}$ es un conjunto numerable.

5. El alfabeto es $X = \{0,1\}$ y el número de bytes coincide con el número de funciones de $\mathbb{N}_8$ en X y también con el número de 8-muestras ordenadas con repetición de los elementos del conjunto X, es decir, $2^8 = 256$. De estos 256 bytes hay uno que no tiene unos: 00000000. Y hay ocho que tienen exactamente un uno: 10000000, 01000000, 001000004, ... ,00000001. Por lo tanto, hay $256 - 9 = 247$ bytes que contienen al menos dos unos.

6. Si representamos cierto por C y falso por F, entonces hemos de calcular el número de funciones del conjunto $\mathbb{N}_{15}$ en el conjunto $X = \{C,F\}$, o sea el número de 15-muestras ordenadas con repetición de los elementos del conjunto X. El número es $2^{15} = 32.768$.

7. La elección se puede hacer de tantas maneras como el número de funciones inyectivas del conjunto $X = \{presidente,secretario,tesorero\}$ en el conjunto $\mathbb{N}_{10}$ de las 10 personas, suponiendo que una misma persona no puede ostentar más de un cargo. Así serán: $V(10,3) = 10 \cdot 9 \cdot 8 = 720$.

8. Para empezar, supongamos $n = 4$. Con cuatro personas podemos hacer $4! = 24$ permutaciones. Con cada una de estas permutaciones tenemos una manera de colocar a las cuatro personas alrededor de una mesa circular. Por ejemplo, la permutación *abcd* corresponde a la distribución siguiente: a la derecha de la persona *a* se coloca la persona *b*, a la derecha de *b* se coloca *c* y a la derecha de *c* se coloca *d*. Pero la permutación *dabc* define la misma colocación, lo mismo pasa con *cdab*, etc. Por lo tanto, diferentes permutaciones pueden dar la misma distribución alrededor de la mesa. Continuando con el caso $n = 4$

podemos hacer seis grupos, de cuatro permutaciones cada uno, que definen la misma distribución:

abcd, *dabc*, *cdab*, *bcda*
abdc, *cabd*, *dcab*, *bdca*
acbd, *dacb*, *bdac*, *cbda*
acdb, *bacd*, *dbac*, *cdba*
adbc, *cadb*, *bcad*, *dbca*
adcb, *badc*, *cbad*, *dcba*

Obsérvese que lo que hemos hecho es fijar una persona (*a*), considerar una permutación que empieza por *a* (*abcd*) y hacer los desplazamientos cíclicos de esta permutación (*dabc*, *cdab*, *bcda*). La permutación *abcd* y sus desplazamientos cíclicos forman el primer grupo. A continuación consideramos la permutación *abdc* y hacemos lo mismo, etc. Claro está que el número de maneras diferentes de colocar las cuatro personas alrededor de una mesa circular es, por lo tanto, seis (tantas como grupos hemos podido formar). Este número 6 es el resultado de dividir 4! (el total de permutaciones) entre 4 (número de permutaciones dentro de cada grupo = número de personas).

Generalizando lo que hemos dicho, podemos afirmar que hay $n!/n = (n-1)!$ maneras de colocar n personas alrededor de una mesa circular.

9. El número de funciones de un conjunto A de cuatro elementos en un conjunto B de siete elementos coincide con el número de 4-muestras ordenadas con repetición del conjunto B. O sea, $VR(7,4) = 7^4 = 2.401$.

De las anteriores, el número de funciones que son inyectivas coincide con el número de 4-muestras ordenadas sin repetición del conjunto B. O sea, $V(7,4) = 7 \cdot 6 \cdot 5 \cdot 4 = 840$.

De funciones biyectivas del conjunto $A = \{x,y,z,t\}$ en el conjunto B de siete elementos, no hay ninguna, puesto que el cardinal de estos conjuntos es diferente.

10. En cada algoritmo contaremos el número de veces que se ejecuta la instrucción *suma* ← *suma* + 1.

a) El bucle más interno se ejecuta un número variable de veces, dependiendo del valor de la variable *i*. Cuando $i = 1$, se ejecuta una vez, cuando $i = 2$, dos veces, etc. En total se ejecutará $1+2+\cdots+n = \dfrac{(n+1)n}{2}$ veces. La complejidad será de $O(n^2)$.

b) Cuando $i = 1$ el bucle j se ejecuta una vez. Cuando $i = 2$ el bucle j se ejecuta cuatro veces, etc. En total, se ejecutará:

$$1 + 4 + 9 + \cdots + n^2 = \sum_{s=1}^{n} s^2 = \frac{n(n+1)(2n+1)}{6}.$$

La complejidad será $O(n^3)$.

c) Este caso es como el anterior pero tenemos que restar aquellos valores de j que son múltiplos de *i*. Entre i y i^2 hay i valores que son múltiplos de *i*. Por ejemplo, cuando $i = 1$ la línea 5 se ejecuta una vez, cuando $i = 2$ la línea 5 se ejecuta $4 - 2$ veces, cuando $i = 3$ la línea 5 se ejecuta $9 - 3$ veces, etc. En total, se ejecutará:

$$1 + (4-2) + (9-3) + \cdots + (n^2 - n) = 1 + (1-1) + (4-2) + (9-3) + \cdots + (n^2 - n) =$$

$$1 + \sum_{s=1}^{n} s^2 - \sum_{s=1}^{n} s = 1 + \frac{n(n+1)(2n+1)}{6} - \frac{n(n+1)}{2} = 1 + \frac{n(n+1)(n-1)}{3}.$$

La complejidad será $O(n^3)$.

11. La función logarítmica es la inversa de la función exponencial y tiene una tasa de crecimiento muy lenta que la hace muy útil en informática.

a) Todas las funciones logarítmicas tienen la misma tasa de crecimiento independientemente de la base de los logaritmos. Esto es así ya que $\log a$ es una constante y,

$$\log_a n = \frac{\log n}{\log a} = \frac{1}{\log a} \log n \in O(\log n).$$

b) Es suficiente darse cuenta de que

$$\log(n^a) = a \cdot \log n \in O(\log n).$$

12. El primer tipo de búsqueda se llama *búsqueda secuencial*. El número mínimo de comparaciones será 1, cuando el valor b que buscamos esté en la primera posición de la secuencia. El máximo número de comparaciones se obtendrá cuando b esté al final o, simplemente, no esté en la secuencia. Habrá que hacer n comparaciones. La complejidad de este algoritmo es $O(n)$.

Si la secuencia está ordenada podemos proceder con el segundo método llamado *búsqueda binaria*. En este caso, el número mínimo de comparaciones será 1 y se obtendrá cuando b esté en la posición $\lfloor n/2 \rfloor$.

El número máximo de comparaciones es un poco más difícil de calcular. Observemos que cada vez que hacemos una comparación negativa también hacemos una división de la secuencia por la mitad. Naturalmente, el número máximo de comparaciones se obtendrá cuando b no esté en la secuencia, es decir, cuando hagamos el máximo posible de divisiones. Contemos, pues, el número de divisiones que habrá que hacer. La tabla siguiente muestra el número de elementos que nos quedan por investigar tras cada división:

Divisiones	Elementos
Inicialmente	n
1	$\lfloor n/2 \rfloor$
2	$\lfloor n/4 \rfloor$
$\vdots$	$\vdots$
k	$\lfloor n/2^k \rfloor$

El proceso de divisiones se acabará cuando quede un solo elemento para investigar, es decir, cuando

$$\lfloor \frac{n}{2^k} \rfloor = 1 \Rightarrow k = \lfloor \log_2 n \rfloor.$$

Por lo tanto, el número de comparaciones será $\log_2 n$ y, por el ejercicio anterior, tendremos una complejidad $O(\log n)$.

Observemos que el algoritmo de búsqueda binaria es mucho más eficiente que el algoritmo de búsqueda secuencial. Observar, también, que esta mejora se obtiene por el hecho de que sabemos que la secuencia ha estado previamente ordenada. Por ello, es conveniente ordenar previamente cualquier secuencia sobre la que haga falta hacer búsquedas: listas de alumnos, listas de teléfonos, archivos en un disco, etc.

13. En el algoritmo de multiplicar y elevar hacemos una multiplicación si n es impar y elevamos al cuadrado si n es par.

a) El número de multiplicaciones es 3, 5, 5, 6.

b) Las representaciones binarias son 1000, 1011, 1110 y 1111.

c) Observar que el número de multiplicaciones coincide con $\lfloor \log_2 n \rfloor$ más el número de unos en la representación binaria de n menos 1. Por ejemplo, para calcular x^8 tenemos que hacer 3 multiplicaciones que coincide con $\log_2 8 + 1 - 1$, puesto que la representación binaria de 8 sólo contiene un 1.

Del mismo modo, para calcular x^{11} tenemos que hacer cinco multiplicaciones, o sea, $\lfloor \log_2 11 \rfloor + 3 - 1$.

Observar que la parte $\lfloor \log_2 11 \rfloor$ corresponde al número de veces que elevamos al cuadrado y, $3 - 1$, al número de veces que multiplicamos.

d) El número de multiplicaciones está acotado por $\lfloor 2 \log_2 n \rfloor$.

e) Si aplicamos la regla 4 y el primer punto del problema 10, deducimos que el algoritmo tiene una complejidad $O(\log n)$.

14. Si encontramos el valor más pequeño de entre $n-1$ números y añadimos otro número, sólo tenemos que hacer una comparación más entre el mínimo que ya habíamos encontrado y el nuevo valor. Si sabemos cuántas comparaciones, $T(n-1)$, son necesarias para encontrar el valor más pequeño de un vector de $n-1$ coordenadas entonces

$$T(n) = T(n-1) + 1 \qquad \forall n \geq 2$$

con $T(1) = 0$, puesto que con un solo número no hace falta hacer ninguna comparación.

Para encontrar la solución, desarrollamos los primeros términos

$$\begin{aligned} T(2) &= T(1) + 1 = 1 \\ T(3) &= T(2) + 1 = 2 \\ T(4) &= T(3) + 1 = 3 \\ &\ldots \end{aligned}$$

así, podemos tomar como hipótesis

$$T(n) = n - 1.$$

Procedamos a demostrar por inducción que esta hipótesis es cierta. Para $n = 1$ tenemos $T(1) = 0$. Si consideremos la hipótesis cierta hasta $n = k$ podemos escribir

$$T(k+1) = T(k) + 1 = k - 1 + 1 = k$$

lo que demuestra que la hipótesis era correcta.

15. En este ejercicio expresaremos el coste del algoritmo como una ecuación recurrente que nos ayudará a calcular la complejidad.

a) Si $n = 1$ entonces C contiene dos elementos y se necesita una sola comparación para determinar el máximo y el mínimo.

b) Si conocemos el máximo (y mínimo) de C_1 y el máximo (y mínimo) de C_2 necesitamos una comparación más para determinar el máximo (y mínimo) de C. La ecuación recurrente será

$$T(n) = 2T(n-1) + 2 \quad (n \geq 2), \quad T(1) = 1.$$

c) Esta ecuación se puede resolver desarrollando los primeros términos

$$\begin{aligned} T(2) &= 2T(1) + 2 = 4 \\ T(3) &= 2T(2) + 2 = 10 \\ T(4) &= 2T(3) + 2 = 22 \\ T(5) &= 2T(4) + 2 = 46 \\ &\ldots \end{aligned}$$

Por inducción se puede demostrar que la solución es $T(n) = 3 \cdot 2^{n-1} - 2$. Para hacer el cálculo de la complejidad observemos que, aunque hemos calculado el número de comparaciones en función de n, el tamaño del conjunto C es $N = 2^n$. El número de comparaciones será $\frac{3}{2}N - 2$ que tiene una complejidad $O(N)$, es decir, tiene una tasa de crecimiento lineal en el tamaño del conjunto C.

Bibliografía

Cormen, T. H. y otros (2001). *Introduction to Algorithms*. Cambridge: MIT Press.

Masià, R.; Pujol, J.; Rifà, J.; Villanueva, M. (2007). *Matemática discreta*. Barcelona: Fundació per a la Universitat Oberta de Catalunya.