Esquemas algorítmicos. Recorrido y búsqueda de una secuencia

PID_00268293

Autores que han participado colectivamente en esta obra
Lluís Beltrà
Kenneth Capseta
Maria Jesús Marco Galindo
Antonio Ponce

Idea y dirección de la obra
Maria Jesús Marco Galindo

Diseño y edición gráfica
Asunción Muñoz

Material docente de la UOC

Universitat Oberta de Catalunya

Tercera edición: febrero 2021

Av. Tibidabo, 39-43, 08035 Barcelona
Realización editorial: FUOC

Contenidos

Esquemas algorítmicos
Recorrido y búsqueda de una secuencia

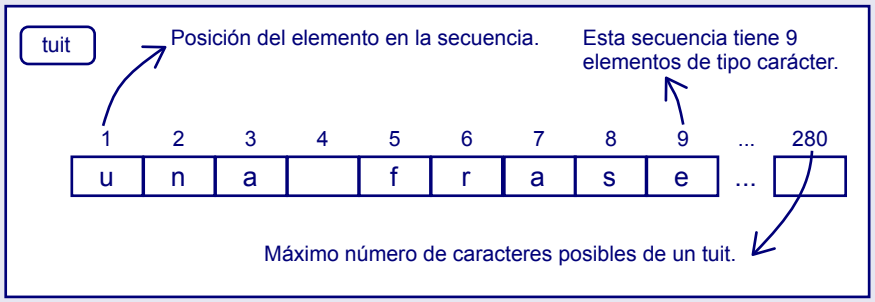
Para ser un buen programador, hay que saber cómo identificar patrones. Un patrón vendría a ser lo que se detecta cuando algo se repite una y otra vez del mismo modo, como por ejemplo el patrón cíclico según el cual se organiza el tiempo en una agenda: meses, semanas, días y horas. El reconocimiento de patrones aplicado a la resolución de problemas nos permite buscar semejanzas entre diversos problemas, para así poder aprovechar lo que sabemos sobre cómo solucionar uno de ellos para aplicarlo a la solución de otro que sea similar.. En el caso del código, detectar patrones nos hace ganar mucha eficiencia y eficacia, de aquí que sean muy importantes.

Recordad que el **reconocimiento de patrones** es uno de los elementos fundamentales del pensamiento computacional.

Ya conocemos la algorítmica básica, así que, con las tres estructuras (secuencial, alternativa e iterativa) y los tipos básicos y estructurados de datos (entero, carácter, real, cadena, booleano, tabla y tupla) podemos planificar cómo resolver cualquier problema que se pueda resolver con un ordenador (lo que denominamos un problema computable). No necesitamos nada más. Pero podemos hacerlo de un modo más sistemático y eficiente si usamos, por ejemplo, patrones. Los **esquemas algorítmicos** son patrones de código que nos ayudan a hacer el tratamiento de modo más sistemático y eficiente.

Ejemplo

Imaginemos que estamos escribiendo un tuit y queremos comprobar si hemos alcanzado el número máximo de caracteres que permite Twitter, que ahora es de 280. Nuestro tuit no es más que una **secuencia de datos** de tipo carácter que podemos representar con una tabla:



¿Cómo representamos el tuit algorítmicamente?

El tuit lo podemos representar como una frase de un máximo de 280 caracteres, que es lo que permite Twitter:

```
var
    tuit: tabla[280] de carácter;
fvar
```

¿Qué es una SECUENCIA de datos?

Una SECUENCIA de datos es una sucesión de elementos ordenados de un modo determinado. La secuencia puede ser de diferentes tipos de datos (carácter, entero, etc.) y siempre tiene un final (es finita).

Una de las estructuras de datos que mejor representa las secuencias es la tabla.

Entender el concepto de secuencia de datos es importante porque la inmensa mayoría de los algoritmos mínimamente complejos consisten en repetir un conjunto de cálculos (un patrón) en una secuencia de datos.

Un caso muy habitual de patrón que se repite cuando tratamos problemas que trabajan con secuencias de datos es el **recorrido**. Hacer un recorrido significa pasar por todos los elementos de la secuencia, uno tras otro. Otro patrón muy común es **la búsqueda**, que consiste en buscar un elemento determinado en una secuencia de datos.

Esquema de recorrido de una secuencia

Pensamos cómo diseñar un patrón, un esquema algorítmico, que nos permita recorrer todos los elementos de una secuencia e irlos tratando uno por uno de forma ordenada, desde el primero hasta el último...

Para recorrer toda la secuencia nos tendremos que situar en el primer elemento, es decir, en la primera posición de la secuencia. Después, deberemos acceder al siguiente elemento y así sucesivamente iremos pasando por todos los elementos de la secuencia hasta que lleguemos al último elemento. Necesitaremos pues una estructura iterativa.

Ejemplo

Imaginemos ahora que queremos contar las frases completas de un tuit. Sabemos que una frase termina porque encontramos un punto, por lo que deberemos recorrer todos los elementos (caracteres) de la tabla (que representa el tuit) e ir contando los puntos que encontraremos:

```
algoritmo cuentaFrasesTuit
var
    i, contador : entero;
    tuit:tabla[280] de caracter;
fvar

    i := 1;
    contador := 0;
    leerTuit(tuit);
    mientras i ≤ 280 hacer

        si tuit[i] = '.' entonces
            contador := contador + 1;
        fsi
        i := i + 1;
    fmientras
    escribirPantalla(contador);
falgoritmo
```

Inicializamos el índice de la tabla a 1 para situarlo en la primera posición de la tabla.

Inicializamos el contador de frases acabadas.

Separamos el problema de leer el tuit en otra función de la que ya nos ocuparemos más adelante. Estamos aplicando la **descomposición de problemas**.

Iteramos para recorrer las 280 posiciones que tiene la tabla.

Sabemos que se ha terminado una frase porque encontramos un punto.

Sumamos uno al contador de frases completas.

Avanzar al siguiente elemento de la tabla: consiste únicamente en incrementar el índice, la variable *i*.

Podemos leer el tuit del teclado con la siguiente función que recibe una variable de tipo tabla vacía como parámetro de entrada y la modifica rellenándola con el texto leido por teclado:

```
función leeTuit (t: tabla[280] de carácter)
var
    i: entero;
fvar
    i := 1;
    mientras i ≤ 280 hacer
    t[i] := leerTeclado();
    i := i+1;
    fmientras
ffunción
```

Esta función espera que se introduzcan por teclado 280 caracteres, ni más ni menos. No serviría para leer tuits de menos caracteres.

Fijaos que las funciones pueden modificar los parámetros que reciben siempre que sean de tipos estructurados. En este caso, se les llama **parámetros de entrada/salida**.

Puede ser que la última frase no la hayamos podido acabar porque hemos gastado los 280 caracteres y no nos ha cabido. Así que, nuestro algoritmo no contará esta última frase del tuit como terminada.

Sintaxis del esquema de recorrido

```
algoritmo esquemaRecorrido
        prepararSecuencia
        inicioTratamiento
        mientras no finSecuencia hacer
                tratarElemento
                avanzarSecuencia
        fmientras
        tratamientoFinal
falgoritmo
```

Nos situamos en el primer elemento de la secuencia, por ejemplo inicializando el índice si trabajamos con una tabla.

Inicializamos las variables que nos servirán para realizar los cálculos correspondientes al tratamiento (si es necesario).

Es una expresión que será cierta cuando estemos en el último elemento de la secuencia.

Acciones o cálculos a realizar sobre el elemento de la secuencia.

Acciones para avanzar al siguiente elemento de la secuencia, por ejemplo incrementar el índice en 1 si trabajamos con una tabla.

Si es necesario, acciones para obtener el resultado del algoritmo a partir del tratamiento a realizar.

Buscamos el esquema de recorrido en el ejemplo anterior donde contábamos las frases acabadas en un tuit:

```
algoritmo cuentaFrasesTuit
var
    i, contador : entero;
    tabla[280] de carácter;
fvar
        i := 1;
        contador := 0;
        leeTuit (tuit);
        mientras i ≤ 280 hacer

                si tuit[i] = '.' entonces
                        contador := contador + 1;
                fsi
                i := i + 1;
        fmientras
        escribirPantalla(contador);
falgoritmo
```

prepararSecuencia:
declaramos la variable *i* y la inicialicemos a 1 para situarnos en la primera posición de la secuencia. En este caso, la secuencia (los caracteres del tuit) están representados por una tabla.

inicioTratamiento:
Inicializamos la variable tuit con el *tuit* a analizar. También tendremos que declarar e inicializar a 0 una variable *contador* que guarde el número de frases acabadas que contiene el tuit.

finSecuencia:
el índice inicialmente vale 1 porque está en la primera posición, pero se irá incrementando de uno en uno para ir recorriendo todos los elementos de la tabla. Cuando valga 280, estará situado en el último elemento y, por lo tanto, ya no tendremos que seguir iterando.

Sabemos que se ha acabado una frase porque encontramos un punto.

tratarElemento:
debemos comprobar si el elemento i-ésimo es un punto y en este caso, incrementar el contador.

avanzarSecuencia:
consiste únicamente en aumentar el índice de la tabla, es decir, la variable *i*.

tratamientoFinal:
podríamos escribir el resultado por pantalla.

Como veis, con este algoritmo, hemos podido realizar un recorrido por todos los elementos de la tabla. El mismo algoritmo nos podría servir para leer cualquier secuencia, aunque tuviera miles de elementos. Por ejemplo, si se ampliase de nuevo la longitud máxima de un tuit solo se tendría que cambiar la longitud máxima de la tabla que ahora es de 280, pero el esquema sería el mismo. O, si queremos contar el número de letras 'a' del tuit, solo tenemos que cambiar '.' por 'a' y el esquema sigue siendo válido porque en todos los casos hacemos lo mismo: un recorrido por todos los elementos de la tabla.

Otro esquema de recorrido de una secuencia

Sabemos que las secuencias son siempre finitas, pero habrá ocasiones en las que no sabemos su número de elementos. En estos casos, lo que sí necesitamos saber siempre es cómo detectar que hemos llegado al final de la secuencia. Un modo de saberlo es conocer cuál será el último valor de la secuencia, el último elemento.

Ejemplo

En el ejemplo anterior hemos trabajado con una secuencia de la que conocíamos el número de elementos, la longitud máxima de un tuit (280 caracteres). Imaginemos ahora que tenemos una secuencia de números que queremos sumar. No sabemos cuántos, pero sí sabemos que la secuencia terminará con el número 0.

El algoritmo que crearemos se encargará de ir sumando los números que iremos leyendo del teclado (entrada estándar). Cuando queramos finalizar, simplemente tendremos que teclear un cero y así indicamos que hemos llegado al final de la secuencia:

```
algoritmo sumaNúmeros
var
    num, suma: entero;
fvar

    num := leerTeclado();
    suma := 0;

    mientras num ≠ 0 hacer

        suma := suma + num;

        num := leerTeclado();

    fmientras

        escribirPantalla(suma);

falgoritmo
```

prepararSecuencia:
declaramos la variable *num*, donde almacenaremos cada número que iremos introduciendo por el teclado.

prepararSecuencia:
En este caso, además, antes de entrar en la composición iterativa, debemos leer el primer elemento de la secuencia.

inicioTratamiento:
declaramos e inicializamos la variable *suma* donde iremos guardando el resultado de ir sumando los valores que iremos introduciendo.

finSecuencia:
comprobar si ya hemos llegado al final, es decir, si el número introducido es un 0.

tratarElemento:
acciones a realizar sobre cada elemento de la secuencia. En este caso, sumamos el elemento actual a la variable *suma*.

avanzarSecuencia:
consiste en leer el siguiente elemento de la secuencia, en este caso, el siguiente número que se introducirá por el teclado.

tratamientoFinal:
mostramos por pantalla el valor de la variable *suma* que contiene la suma de todos los elementos de la secuencia que hemos ido introduciendo.

Hemos podido realizar un recorrido por todos los elementos de la secuencia hasta llegar al último elemento (número 0).

En este caso no hemos representado la secuencia de números con una tabla, no siempre las secuencias de datos se representan con tablas.

Todas las secuencias que podemos tratar deben ser finitas y siempre deberemos tener algún modo de saber cuándo hemos llegado al final, ya sea porque sabemos cuántos elementos contiene la secuencia o porque conocemos cuál el último elemento (que evidentemente tiene que ser un elemento distintivo y diferente de todos los anteriores).

¿Para qué sirve...?

Concepto clave

Un **esquema algorítmico** es un patrón que podemos seguir para resolver un problema prototípico habitual. Así no tenemos que pensarlo de nuevo y reinventar cada vez la solución. Con esto, ganamos eficiencia y eficacia ya que si aplicamos bien el esquema seguro que la solución funcionará correctamente.

El **esquema de recorrido** se puede usar para resolver problemas en los que, para encontrar la solución, se necesita recorrer (revisar, tratar) todos y cada uno de los datos de una secuencia. Esta secuencia de datos debe ser finita y siempre hay que tener el modo de saber cuándo hemos llegado al final, ya sea porque sabemos cuántos elementos contiene la secuencia o porque conocemos cuál es el último elemento que, evidentemente, debe ser un elemento distintivo y diferente de todos los anteriores.

El esquema siempre sigue una misma estructura:

1. Obtener el primer elemento.
2. Inicializar las variables.
3. Iterar hasta llegar al final y dentro de la iteración:
   - **3.1** Hacer el tratamiento que sea necesario a cada elemento para resolver el problema.
   - **3.2** Pasar al siguiente elemento.
4. Si hace falta, calcular y mostrar el resultado final una vez tratados todos los datos de la secuencia.

```
algoritmo esquemaRecorrido
    prepararSecuencia
    inicioTratamiento
    mientras no finSecuencia hacer
        tratarElemento
        avanzarSecuencia
    fmientras
    tratamientoFinal
falgoritmo
```

Una **secuencia** es una sucesión ordenada de datos. A veces la representaremos algorítmicamente con una tabla, pero no siempre tiene que ser así. Lo que sí es imprescindible para que podamos aplicar este esquema es que sea finita y que tengamos algún modo de saber cuándo hemos llegado al final, bien porque ya sabemos su longitud, bien porque tenemos el modo de identificar el último elemento que es distintivo y diferente del resto (este elemento se llama centinela).

Esquema de búsqueda de una secuencia

¿Y si el problema que tenemos implica buscar un elemento dentro de una secuencia de datos?

Imaginemos que somos unos apasionados de las olimpiadas y queremos saber si el año en el que cumplimos 40 años podremos celebrarlo yendo a ver las olimpiadas. Sabemos la secuencia prevista de los años olímpicos de las próximas dos décadas: 2021, 2024, 2028, 2032, 2036, 2040. ¿Estará el año de nuestro 40.º aniversario en esta secuencia?

Podemos pensar en realizar un recorrido por la secuencia, donde pasando por y tratando todos sus elementos. Pero fijémonos que tal vez no sea necesario recorrer todos los elementos; por ejemplo, si cumplimos los 40 años en 2028, cuando lleguemos a encontrar el año 2028 ya sabemos que nuestro 40 aniversario lo podremos celebrar en unas olimpiadas y no tendremos que seguir revisando el resto de elementos de la secuencia hasta el final.

Lo que estamos haciendo es **buscar** un elemento en una secuencia, por lo que cuando lo encontremos ya no será necesario continuar hasta el final de la secuencia. Una búsqueda sigue, también un esquema, un patrón.

Ejemplo

El siguiente algoritmo determina si el año 2008 fue un año olímpico o no. Partimos del supuesto de que tenemos una secuencia de números correspondientes a los últimos años en los que ha habido olimpiadas:

	1	2	3	4	5	6
Años	1996	2000	2004	2008	2012	2016

Podemos representarlo como una tabla de enteros:

```
var
    añosolímpicos : tabla [6] de entero;
fvar
```

```
algoritmo buscaAñoOlímpico
var
    i: entero;
    añosolímpicos: tabla [6] de entero;
    encontrado: booleano;
fvar
    i := 1;
    encontrado := falso;
    añosolímpicos[1]:1996;
    añosolímpicos[2]:2000;
    añosolímpicos[3]:2004;
    añosolímpicos[4]:2008;
    añosolímpicos[5]:2012;
    añosolímpicos[6]:2016;

    mientras i ≤ 6 y no encontrado hacer
      encontrado := (añosolímpicos[i] = 2008)
      si encontrado entonces
        escribirPantalla("En el año 2008 se celebraron
        unas olimpiadas");
      fsi

      i := i + 1;
    fmientras

    si no encontrado entonces
      escribirPantalla("En el año 2008 no hubo
      olimpiadas");
    fsi
falgoritmo
```

A lo mejor las acciones de la estructura iterativa se ven más claras así:

```
mientras i ≤ 6 y no encontrado hacer
    si (añosolímpicos[i] = 2008) entonces
        escribirPantalla("En el año 2008 se celebraron unas olimpiadas");
        encontrado := cierto;
    si no
        encontrado := falso;
    fsi
        i := i + 1;
fmientras
```

Ya hemos encontrado el valor que estábamos buscando y podemos aplicar las acciones que nos interesen. En este caso simplemente escribiremos por pantalla que en 2008 se celebraron unas olimpiadas.

Pensad si este algoritmo se podría optimizar. Por ejemplo, si buscamos el año 2007, cuando encontremos el 2008, ¿hace falta seguir buscando? No, en este caso, si sabemos con certeza que la secuencia está ordenada, ya podemos acabar aunque no hayamos recorrido aún todos los elementos ni encontrado el que buscamos porque ya sabemos que no lo encontraremos. Ahora bien, si la secuencia no está ordenada, sí que se debe seguir buscando hasta el final mientras no encontremos el año 2007.

Sintaxis del esquema de búsqueda

```
algoritmo esquemaBúsqueda
        var
          encontrado: booleano;
        fvar
        prepararSecuencia
        encontrado := falso;
        inicioTratamiento
        mientras no finSecuencia y no encontrado hacer
            actualizarEncontrado
            si no encontrado entonces
               tratarElemento
               avanzarSecuencia
            fsi
        fmientras
        tratamientoFinal
falgoritmo
```

Obtenemos el primer elemento de la secuencia, nos situamos en el primer elemento de la secuencia.

Iniciamos las variables que nos servirán para realizar los cálculos correspondientes al tratamiento (si es necesario).

Es una expresión que será cierta cuando estemos en el último elemento de la secuencia.

Acciones para determinar si el elemento actual es el que estamos buscando. En caso afirmativo, la variable *encontrado* toma el valor de cierto.

Acciones a realizar sobre un elemento de la secuencia.

Acciones que avanzarán hacia el segundo elemento.

Acciones finales para obtener el resultado del algoritmo a partir de la búsqueda realizada (si es necesario).

A diferencia del esquema de recorrido, donde recorremos y tratamos todos los elementos de la secuencia, con este nuevo esquema recorremos la secuencia únicamente hasta que encontramos un elemento que cumpla una determinada condición o que ya tengamos claro que no lo encontraremos, como pasaba en el ejemplo de los años olímpicos si la secuencia estaba ordenada.

Hay que tener presente que esta condición no tiene por qué referirse a un elemento con un valor concreto (como por ejemplo, cuando queremos encontrar la letra 'a' en una frase), sino que puede ser una condición más general (encontrar el primer número par de una secuencia de enteros, la primera palabra capicúa de un texto, etc.).

Buscamos el esquema de búsqueda en el ejemplo anterior donde estábamos buscando si el año 2008 era olímpico:

```
algoritmo buscaAñoolímpico
var
      i: entero;
      añosolímpicos: tabla [6] de entero;
      encontrado: booleano;
fvar
      i := 1;
      encontrado := falso;
      añosolímpicos[1] := 1996; añosolímpicos[2] := 2000; añosolímpicos[3] := 2004;
      añosolímpicos[4] := 2008; añosolímpicos[5] := 2012; añosolímpicos[6] := 2016;

      mientras i ≤ 6 y no encontrado hacer
         encontrado := (añosolímpicos[i] = 2008)
         si encontrado entonces
         escribirPantalla("En el año 2008 se celebraron unas olimpiadas");
         fsi
         i := i + 1;
      fmientras

         si no encontrado entonces
         escribirPantalla("En el año 2008 no hubo olimpiadas");
         fsi
falgoritmo
```

prepararSecuencia:
Declaramos la variable *i* y la inicializamos en 1 para situarnos en la primera posición de la tabla.

inicioTratamiento:
declaramos e inicializamos la variable *años* con los valores indicados anteriormente. También deberemos declarar e inicializar en a **falso** una variable *encontrado* que nos indicará si ya se ha encontrado el valor buscado o no.

finSecuencia:
el índice inicialmente vale 1 porque está en la primera posición, pero se irá incrementando de uno a uno para ir recorriendo todos los elementos de la tabla hasta llegar al final o hasta encontrar el valor 2008 que estamos buscando. Si se cumple una de estas dos condiciones, saldremos de la composición iterativa y dejaremos de recorrer la secuencia.

actualizarEncontrado
comprobamos si el valor actual es el que estamos buscando. En caso afirmativo, la variable *encontrado* pasa a tener valor **cierto**, de modo que en la siguiente iteración ya no se cumplirán las condiciones para seguir iterando y dejaremos de recorrer la tabla.

avanzarSecuencia:
consiste únicamente en incrementar la variable *i*.

tratamientoFinal:
si al salir de la composición iterativa el valor de la variable *encontrado* sigue siendo **falso**, significa que no hemos encontrado el valor que buscábamos dentro de la tabla.

¿Para qué sirve...?

Concepto clave

El **esquema de búsqueda** se utiliza para resolver problemas en los que la solución implique encontrar y tratar un elemento (o conjunto de elementos) concreto en una secuencia. Es de hecho una variación del esquema de recorrido en el que es posible que no haya que tratar necesariamente toda la secuencia de datos.

El esquema sigue siempre una misma estructura:

1 Obtener el primer elemento.

2 Inicializar las variables.

3 Iterar hasta encontrar el elemento que buscamos o hasta llegar al final:

3.1 Hacer el tratamiento que sea necesario para resolver el problema.

3.2 Pasar al siguiente elemento si aún no hemos encontrado lo que buscamos.

4 Si es necesario, calcular y mostrar el resultado final una vez hayamos encontrado lo que buscamos dentro de la secuencia o no si llegamos al final.

```
algoritmo EsquemaBúsqueda
        var
                encontrado: booleano;
        fvar
        prepararSecuencia
        encontrado := falso;
        inicioTratamiento
        mientras no finSecuencia y no encontrado hacer
                actualizarEncontrado
                si no encontrado entonces
                        tratarElemento
                        avanzarSecuencia
                fsi
        fmientras
        tratamientoFinal
falgoritmo
```

Fijaos que si el elemento que estamos buscando resulta que es el último de la secuencia, estaremos haciendo, de facto, un recorrido por toda la secuencia. Por eso decimos que el esquema de búsqueda es un caso particular del esquema de recorrido.

Recorrido y búsqueda de una secuencia en JS

¿Cómo serían estos algoritmos en JavaScript?

Ejemplo

Contar frases terminadas de un tuit.

```
var i = 0;
var contador = 0;
//Frase célebre de Grouxo Marx
var tuit = "Estos son mis principios. Si no les gustan, tengo otros.";
// En JavaScript las cadenas de texto son accesibles como si fueran
// un array  e incluso tienen la propiedad length para saber su extensión

while (i < tuit.length){
    console.log(i, tuit[i]);
    if(tuit[i]=="."){
        contador++;
    }
    i++;
}
console.log(contador);
```

Sumar una secuencia de números.

```
var suma = 0;
console.log(suma);
var i = 0;
var numerosSuma = [3, 1, 2, 3, 0, 5, 8];
while (numerosSuma[i]!=0) {
  suma = suma + numerosSuma[i];
  i++;
}
console.log(suma);
```

Averiguar si un año es olímpico.

```
var i = 0;
var años = [1996, 2000, 2004, 2008, 2012, 2016];
var encontrado = false;
while (i < 6 && !encontrado){
    encontrado = (años[i] == 2008);
    i++;
    if (encontrado){
        console.log("El año 2008 se disputaron una olimpiadas.");
    }
}
if (!encontrado){
    console.log("El año 2008 no se disputaron una olimpiadas.");
}
```

Podéis probar estos códigos en la web de PythonTutor: