BUSQUEDA

La búsqueda de un elemento dentro de un conjunto de elementos es una de las operaciones más importantes en el procesamiento de la información y permite la recuperación de datos previamente almacenados.

El tipo de búsqueda se puede clasificar como interna o externa, según el lugar en el que esté almacenada la información (en memoria o en dispositivos externos).

Todos los algoritmos de búsqueda tienen dos finalidades:

- Determinar si el elemento buscado se encuentra en el conjunto en el que se busca.
- Si el elemento está en el conjunto, hallar la posición en la que se encuentra.

En la búsqueda interna los principales algoritmos son: la búsqueda secuencial, la binaria y la búsqueda utilizando tablas de hash.

Búsqueda secuencial

Consiste en recorrer y examinar cada uno de los elementos del conjunto de datos (listas, array, arreglos o vectores) hasta encontrar el o los elementos buscados, o hasta que se han revisado todos los elementos de la lista.

```
i = 0
while i < N:
     if lista[i] == elemento:
          print("Encontrado")
     i = i + 1
```

Este algoritmo se puede optimizar cuando la lista está ordenada, en cuyo caso la condición de salida cambiaría a:

while (i < N) and (lista[i] != elemento):

o cuando sólo interesa conocer la primera ocurrencia del elemento en el array:

En este último caso, cuando sólo interesa la primera posición, se puede utilizar un centinela, esto es, dar a la posición siguiente al último elemento de array el valor del elemento, para estar seguro de que se encuentra el elemento, y no tener que comprobar a cada paso si seguimos buscando dentro de los límites del array:

```
i = 0
while i < N:
     if lista[i] == elemento:
          print("Encontrado")
          i = N
     i = i + 1
```

Si al acabar el bucle, i vale N es que no se encontraba el elemento. El número medio de comparaciones que hay que hacer antes de encontrar el elemento buscado es de (N+1)/2.

Búsqueda binaria o dicotómica

Para utilizar este algoritmo, el array debe estar ordenado. La búsqueda binaria consiste en dividir la lista por su elemento medio en dos sublistas más pequeñas, y comparar el elemento con el del centro. Si coinciden, la búsqueda se termina. Si el elemento es menor, debe estar (si está) en la primera sublista, y si es mayor está en la segunda. Por ejemplo, para buscar el elemento 3 en la lista [1,2,3,4,5,6,7,8,9] se realizarían los siguientes pasos:

Se toma el elemento central y se divide la lista en dos:

`[1,2,3,4]-5-[6,7,8,9]`

Como el elemento buscado (3) es menor que el central (5), debe estar en la primera sublista:
`[1,2,3,4]`

Se vuelve a dividir el array en dos:

`[1]-2-[3,4]`

Como el elemento buscado es mayor que el central, debe estar en la segunda sublista:

`[3,4]`

Se vuelve a dividir en dos:

`[]-3-[4]`

Como el elemento buscado coincide con el central, lo hemos encontrado.

Si al final de la búsqueda todavía no lo hemos encontrado, y la sublista a dividir está vacía `[]` el elemento no se encuentra en el array.

```
# Incorporar la solución entregada por los estudiantes.
```

En general, este método realiza $\log_2(N+1)$ comparaciones antes de encontrar el elemento, o antes de descubrir que no está. Este número es muy inferior que el necesario para la búsqueda lineal para casos grandes.

Este método también se puede implementar de forma recursiva, siendo la función recursiva la que divide al array en dos más pequeños.

```
# Agregar la implementación recursiva entregada por los estudiantes
```

Búsqueda mediante transformación de claves (hashing)

Es un método que aumenta la velocidad de búsqueda, pero que no requiere que los elementos estén ordenados.

Consiste en asignar a cada elemento un índice mediante una transformación del elemento. Esta correspondencia se realiza mediante una función de conversión, llamada función hash.

La correspondencia más sencilla es la identidad, esto es, al número 0 se le asigna el índice 0, al elemento 1 el índice 1, y así sucesivamente.

Pero si los números a almacenar son demasiado grandes esta función es inservible. Por ejemplo, se quiere guardar en una lista la información de los **1000** usuarios de una empresa, y se elige el número de **RUT** como elemento identificativo.

Es inviable hacer un array de **100.000.000** elementos, sobre todo porque se desaprovecha demasiado espacio.

Por eso, se realiza una transformación al número de RUT para que nos dé un número menor, por ejemplo escoger las 3 últimas cifras para guardar a los empleados en una lista de 1000 elementos. Para buscar a uno de ellos, bastaría con realizar la transformación a su RUT y ver si está o no en la lista.

La función de hash ideal debería ser biyectiva, esto es, que a cada elemento le corresponda un índice, y que a cada índice le corresponda un elemento, pero no siempre es fácil encontrar esa función, e incluso a veces es inútil, ya que se puede no saber el número de elementos a almacenar. La función de hash depende de cada problema y de cada finalidad, y se pueden utilizar con números o string, pero las más utilizadas son:

Restas sucesivas: esta función se emplea con claves numéricas entre las que existen espacios (gap) de tamaño conocido, obteniéndose direcciones consecutivas.

Por ejemplo, si el número de matricula de un estudiante universitario está formado por el año de entrada en la universidad, seguido de un número identificativo de tres cifras, y suponiendo que entran un máximo de 400 estudiantes al año, se le asignarían las claves:

```
2023-000 --> 0 = 2023000-2023000
2023-001 --> 1 = 2023001-2023000
2023-002 --> 2 = 2023002-2023000
     ...
2023-399 --> 399 = 2023399-2023000
2024-000 --> 400 = 2024000-2024000+400
     ...
yyyy-nnn --> N = yyyynnn-2023000+(400*(yyyy-2023))
```

Aritmética modular: el índice de un número es resto de la división de ese número entre un número N prefijado, preferentemente primo. Los números se guardarán en las direcciones de memoria de 0 a N-1. Este método tiene el problema de que cuando hay N+1 elementos, al menos un índice es señalado por dos elementos (teorema del palomar).

A este fenómeno se le llama colisión, y es tratado más adelante. Si el número N es el 13, los números siguientes quedan transformados en:

```
13000000 --> 0
12345678 --> 7
13602499 --> 1
71140205 --> 6
73062138 --> 6
```

Mitad del cuadrado: consiste en elevar al cuadrado la clave y coger las cifras centrales. Este método también presenta problemas de colisión:

```
123*123=15129  --> 51
136*136=18496  --> 84
730*730=532900 --> 29
301*301=90601  --> 06
625*625=390625 --> 06
```

Truncamiento: consiste en ignorar parte del número y utilizar los elementos restantes como índice. También se produce colisión. Por ejemplo, si un número de 8 cifras se debe ordenar en un array de 1000 elementos, se pueden coger la primer, la tercer y la última cifras para formar un nuevo número:

```
13000000 --> 100
12345678 --> 138
13602499 --> 169
71140205 --> 715
73162135 --> 715
```

Plegamiento: consiste en dividir el número en diferentes partes, y operar con ellas (normalmente con suma o multiplicación). También se produce colisión. Por ejemplo, si dividimos los número de 8 cifras en 3, 3 y 2 cifras y se suman, dará otro número de tres cifras (y si no, se cogen las tres últimas cifras):

```
13000000 --> 130=130+000+00
12345678 --> 657=123+456+78
71140205 --> 118 --> 1118=711+402+05
13602499 --> 259=136+024+99
25000009 --> 259=250+000+09
```

Tratamiento de colisiones

Pero ahora se nos presenta el problema de qué hacer con las colisiones, qué pasa cuando a dos elementos diferentes les corresponde el mismo índice. Pues bien, hay tres posibles soluciones:

Cuando el índice correspondiente a un elemento ya está ocupada, se le asigna el primer índice libre a partir de esa posición. Este método es poco eficaz, porque al nuevo elemento se le asigna un índice que podrá estar ocupado por un elemento posterior a él, y la búsqueda se ralentiza, ya que no se sabe la posición exacta del elemento.

También se pueden reservar unos cuantos lugares al final del array para alojar a las colisiones. Este método también tiene un problema: ¿Cuánto espacio se debe reservar? Además, sigue la lentitud de búsqueda si el elemento a buscar es una colisión.

Lo más efectivo es, en vez de crear una lista de números, crear una lista de listas donde cada índice de la lista original señala el principio de una nueva lista (denominada enlazada). Así, cada elemento que llega a un determinado índice se pone en el último lugar de la lista de ese índice. El tiempo de búsqueda se reduce considerablemente, y no hace falta poner restricciones al tamaño del array, ya que se pueden añadir nodos dinámicamente a la lista