Algoritmo de búsqueda

31 idiomas

* [Artículo](https://es.wikipedia.org/wiki/Algoritmo_de_b%C3%BAsqueda)
* [Discusión](https://es.wikipedia.org/wiki/Discusi%C3%B3n:Algoritmo_de_b%C3%BAsqueda)
* [Leer](https://es.wikipedia.org/wiki/Algoritmo_de_b%C3%BAsqueda)
* [Editar](https://es.wikipedia.org/w/index.php?title=Algoritmo_de_b%C3%BAsqueda&action=edit)
* [Ver historial](https://es.wikipedia.org/w/index.php?title=Algoritmo_de_b%C3%BAsqueda&action=history)

Herramientas

*Para otros usos de este término, véase*[*Búsqueda*](https://es.wikipedia.org/wiki/B%C3%BAsqueda)*.*

Un [algoritmo](https://es.wikipedia.org/wiki/Algoritmo) de búsqueda es un [conjunto de instrucciones](https://es.wikipedia.org/wiki/Conjunto_de_instrucciones) que están diseñadas para localizar un elemento con ciertas propiedades dentro de una [estructura de datos](https://es.wikipedia.org/wiki/Estructura_de_datos); por ejemplo, ubicar el registro correspondiente a cierta persona en una [base de datos](https://es.wikipedia.org/wiki/Base_de_datos), o el mejor movimiento en una partida de [ajedrez](https://es.wikipedia.org/wiki/Ajedrez).

La variante más simple del problema es la búsqueda de un número en un [vector](https://es.wikipedia.org/wiki/Vector_(programaci%C3%B3n)).

Búsqueda informada vs no informada

[[editar](https://es.wikipedia.org/w/index.php?title=Algoritmo_de_b%C3%BAsqueda&action=edit&section=1)]

Un problema típico de la [inteligencia artificial](https://es.wikipedia.org/wiki/Inteligencia_artificial) consiste en buscar un estado concreto entre un conjunto determinado, al que se le llama [espacio de estados](https://es.wikipedia.org/wiki/Espacio_de_estados). Imaginemos, por ejemplo, una habitación con baldosines en la que hay un libro. Un [robot](https://es.wikipedia.org/wiki/Robot) se desea desplazar por la habitación con el fin de llegar a dicho libro. ¿De qué manera lo hará? En este punto es donde entran en juego las estrategias y los algoritmos de búsqueda.

Cuando el sistema agente (en este caso, el robot) posee algún tipo de información del medio, se utilizan técnicas de búsquedas informadas; sin embargo, si carece de conocimiento alguno, se deberán emplear algoritmos de búsqueda no informadas. En nuestro ejemplo, y para este último caso, podemos imaginar un robot que no posea ningún tipo de [visión artificial](https://es.wikipedia.org/wiki/Visi%C3%B3n_artificial), que únicamente sea capaz de moverse en horizontal o vertical de un baldosín a otro y detectar si en el baldosín se halla el libro.

[Imagen en blanco y negro

El contenido generado por IA puede ser incorrecto.](https://commons.wikimedia.org/wiki/File:Grafico_Algoritmo_de_b%C3%BAsqueda.jpg)

De esta forma, los algoritmos de búsqueda pueden ser:

* Algoritmos [no informados](https://es.wikipedia.org/wiki/B%C3%BAsquedas_no_informadas) o ciegos: en general más ineficientes en tiempo y memoria que otros métodos.
* Algoritmos informados
  + Algoritmos [heurísticos](https://es.wikipedia.org/w/index.php?title=Heur%C3%ADstica_(inform%C3%A1tica)&action=edit&redlink=1): destacan las Búsquedas Primero el Mejor ([Algoritmo voraz o Greedy](https://es.wikipedia.org/wiki/Algoritmo_voraz) y [Algoritmo de búsqueda A\*](https://es.wikipedia.org/wiki/Algoritmo_de_b%C3%BAsqueda_A*)) y de Mejora Iterativa ([Algoritmo Escalada Simple -Hill Climbing-](https://es.wikipedia.org/wiki/Algoritmo_hill_climbing) y Escalada por Máxima Pendiente)
  + Algoritmos de Búsqueda con adversario: destacan el [Minimax](https://es.wikipedia.org/wiki/Minimax" \o "Minimax) y el [Poda alfa-beta](https://es.wikipedia.org/wiki/Poda_alfa-beta).

Búsqueda secuencial

[[editar](https://es.wikipedia.org/w/index.php?title=Algoritmo_de_b%C3%BAsqueda&action=edit&section=2)]

Consiste en ir comparando el elemento a buscar con cada elemento del vector hasta encontrarlo o hasta que se llegue al final, esto hace que la búsqueda sea secuencialmente (de ahí su nombre). La existencia se puede asegurar cuando el elemento es localizado, pero no podemos asegurar la no existencia hasta no haber analizado todos los elementos del vector. Se utiliza sin importar si el vector está previamente [ordenado](https://es.wikipedia.org/wiki/Orden_total) o no. A continuación se muestra el [pseudocódigo](https://es.wikipedia.org/wiki/Pseudoc%C3%B3digo) del algoritmo:[[1]](https://es.wikipedia.org/wiki/Algoritmo_de_b%C3%BAsqueda#cite_note-1)​

Datos de entrada:

vec: vector en el que se desea buscar el dato

tam: tamaño del vector. Los subíndices válidos van desde 0 hasta tam-1 inclusive. Puede representarse así: *vec[0...tam) o vec[0...tam-1]*.

dato: elemento que se quiere buscar.

Variables

pos: posición actual en el vector

pos = 0

while pos < tam:

if vec[pos] == dato:

Retorne *verdadero* y/o *pos*,

else:

pos = pos + 1

Fin (while)

Retorne *falso*,

* [C](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n))

int busquedaSimple(int vector[n], int n, int dato) {

int i;

for(i=0; i<n; i++){

if(dato==vector[i]) {

return i;

}

}

return -1;

}

Búsqueda dicotómica (binaria)

[[editar](https://es.wikipedia.org/w/index.php?title=Algoritmo_de_b%C3%BAsqueda&action=edit&section=3)]

Se utiliza cuando el vector en el que queremos determinar la existencia de un elemento está previamente ordenado. Este algoritmo reduce el tiempo de búsqueda considerablemente, ya que disminuye [exponencialmente](https://es.wikipedia.org/wiki/Crecimiento_exponencial) el número de iteraciones necesarias. En el peor de los casos el número máximo de comparaciones es ⌊log2⁡n+1⌋, donde n es el número de los elementos en el vector. Por ejemplo, en uno conteniendo 50.000.000 elementos, el algoritmo realiza como máximo 26 comparaciones.

Para implementar este algoritmo se compara el elemento a buscar con un elemento cualquiera del vector (normalmente el elemento central): si el valor de este es mayor que el del elemento buscado se repite el procedimiento en la parte del vector que va desde el inicio de este hasta el elemento tomado, en caso contrario se toma la parte del vector que va desde el elemento tomado hasta el final. De esta manera obtenemos intervalos cada vez más pequeños, hasta que se obtenga un intervalo indivisible. Si el elemento no se encuentra dentro de este último entonces se deduce que el elemento buscado no se encuentra en todo el vector.

A continuación se presenta el pseudocódigo del algoritmo, tomando como elemento inicial el elemento central del vector.

Datos de entrada:

vec: vector en el que se desea buscar el dato

tam: tamaño del vector. Los subíndices válidos van desde 0 hasta tam-1 inclusive.

dato: elemento que se quiere buscar.

Variables

centro: subíndice central del intervalo

inf: límite inferior del intervalo

sup: límite superior del intervalo

inf = 0

sup = tam-1

Mientras inf <= sup:

centro = ((sup - inf) / 2) + inf // División entera: se trunca la fracción

Si vec[centro] == dato devolver *verdadero* y/o *pos*, de lo contrario:

Si dato < vec[centro] entonces:

sup = centro - 1

En caso contrario:

inf = centro + 1

Fin (Mientras)

Devolver *Falso*

* [C](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n))

int busquedaBinaria(int vector[], int n, int dato) {

int centro,inf=0,sup=n-1;

while(inf<=sup){

centro=((sup-inf)/2)+inf;

if(vector[centro]==dato) return centro;

else if(dato < vector[centro]) sup=centro-1;

else inf=centro+1;

}

return -1;

}

Implementación recursiva en C++

* [C++](https://es.wikipedia.org/wiki/C%2B%2B)

#include *<vector>*

bool busqueda\_dicotomica(const vector<int> &v, int principio, int fin, int &x){

bool res;

if(principio <= fin){

int m = ((fin - principio)/2) + principio;

if(x < v[m]) res = busqueda\_dicotomica(v, principio, m-1, x);

else if(x > v[m]) res = busqueda\_dicotomica(v, m+1, fin, x);

else res = true;

}else res = false;

return res;

}

*/\*{Post: Si se encuentra devuelve true, sino false}\*/*

Implementación recursiva en Python

* [Python](https://es.wikipedia.org/wiki/Python)

def busquedaBinaria (numeros, inicio, fin, elemento):

if (inicio == fin): return numeros [inicio] == elemento

centro = ((fin - inicio) // 2) + inicio

if (elemento < numeros [centro]):

return busquedaBinaria (numeros, inicio, centro - 1, elemento)

elif (elemento > numeros [centro]):

return busquedaBinaria (numeros, centro + 1, fin, elemento)

else: return True

def busqueda (numeros, elemento):

if (numeros == None) or (numeros == []):

return False

else: return busquedaBinaria (numeros, 0, len (numeros) - 1, elemento)

Implementación recursiva en Python3

* [Python](https://es.wikipedia.org/wiki/Python)

def bin(a,x,low,hi):

ans = -1

if low==hi: ans = -1

else:

mid = (low+((hi-low)//2))

if x < a[mid]: ans = bin(a,x,low,mid)

elif x > a[mid]: ans = bin(a,x,mid+1,hi)

else: ans = mid

return ans

*# Así se hace el llamado: print(bin(Lista, numero\_a\_buscar, 0, len(Lista)))*

*# Retorna el índice que coincide con 'numero\_a\_buscar', si no está retorna -1*

*# Tiempo: (log n)*

Implementación iterativa en Python3

* [Python](https://es.wikipedia.org/wiki/Python)

def bin(a, c):

ans = -1

if a[0] >= c: ans = -1

else:

low, hi = 0, len(a)

while low+1 != hi:

mid = low + ((hi-low)//2)

if a[mid] < c: low = mid

else: hi = mid

ans = low

return ans

*# Así se hace el llamado: print(bin(lista(), numero\_a\_buscar))*

*# Retorna el índice que coincide con 'numero\_a\_buscar', si no está retorna -1*

*# Tiempo: (log n)*