Búsqueda Binaria Avanzada

Enzo Vivallo

enzovivallo@estudiante.uc.cl

¿Qué es la búsqueda binaria?

- Es un algoritmo para encontrar un valor en funciones booleanas monótonas.
- Funciona en $\log(n)$, con n el tamaño del dominio de la función.
- La estrategia es dividir el dominio en dos partes y descartar una de ellas.

Funciones booleanas monótonas

- Booleana o binaria significa que la función devuelve $true$ ✅ o $false$ ❌.
- Monótona significa que la función es creciente o decreciente, es decir, si $f(x) \leq f(y)$ entonces $x \leq y$ o $x \geq y$.
- El dominio de la función en la práctica puede ser una lista de valores explicitos o un rango de valores.
- Por ejemplo para $f(x) = x \geq 5$ el dominio puede ser un rango $x \in [1, 7]$ o también una lista de valores $x \in 1, 2, 7, 10$.

Ejemplo

El ejemplo clásico es buscar un elemento en una lista ordenada, si queremos buscan el elemento $target$ en la lista A, la función sería $f(x) = A[x] \geq target$ y el dominio sería los valores ordenados de la lista A.

Para la lista $A = [1, 3, 4, 5, 8, 9]$ y el $target = 5$:

x	1	3	4	5	8	9
f(x)	❌	❌	❌	✅	✅	✅

Ejemplo

La función raíz cuadrada entera o $\lfloor \sqrt{n} \rfloor$ también tiene ese comportamiento, si queremos encontrar la raíz cuadrada de un número entero n, la función sería $f(x) = x^2 \geq n$ y el dominio sería $x \in [0, n]$.

Para $n = 9$:

x	0	1	2	3	4	5	6	7	8	9
f(x)	❌	❌	❌	✅	✅	✅	✅	✅	✅	✅

¿Cómo funciona?

- Se define un rango inicial $[left, right]$ que contiene el valor buscado.
- Se calcula el punto medio $mid = \left\lfloor \frac{left+right}{2} \right\rfloor$.
- Se evalúa la función $f(mid)$.
  - Si $f(mid)$ es $true$, significa que el valor buscado está en el rango $[left, mid]$, por lo que se actualiza $right = mid$.
  - Si $f(mid)$ es $false$, significa que el valor buscado está en el rango $[mid + 1, right]$, por lo que se actualiza $left = mid + 1$.
- Se repite el proceso hasta que $left = right$.

¿Cómo funciona?

Para el primer ejemplo, si buscamos el $target = 5$ en la lista $A = [1, 3, 4, 5, 8, 9]$ con la función $f(x) = A[x] \geq target$:

$left = 0$ $\qquad$ $right = 5$

1	3	4	5	8	9

$$mid = \left\lfloor \frac{0+5}{2} \right\rfloor = 2 \longrightarrow 4 \geq 5 = false \longrightarrow left = 2 + 1 = 3$$

$left = 3$ $\qquad$ $right = 5$

✗	✗	✗	5	8	9

$$mid = \left\lfloor \frac{3+5}{2} \right\rfloor = 4 \longrightarrow 8 \geq 5 = true \longrightarrow right = 4 - 1 = 3$$

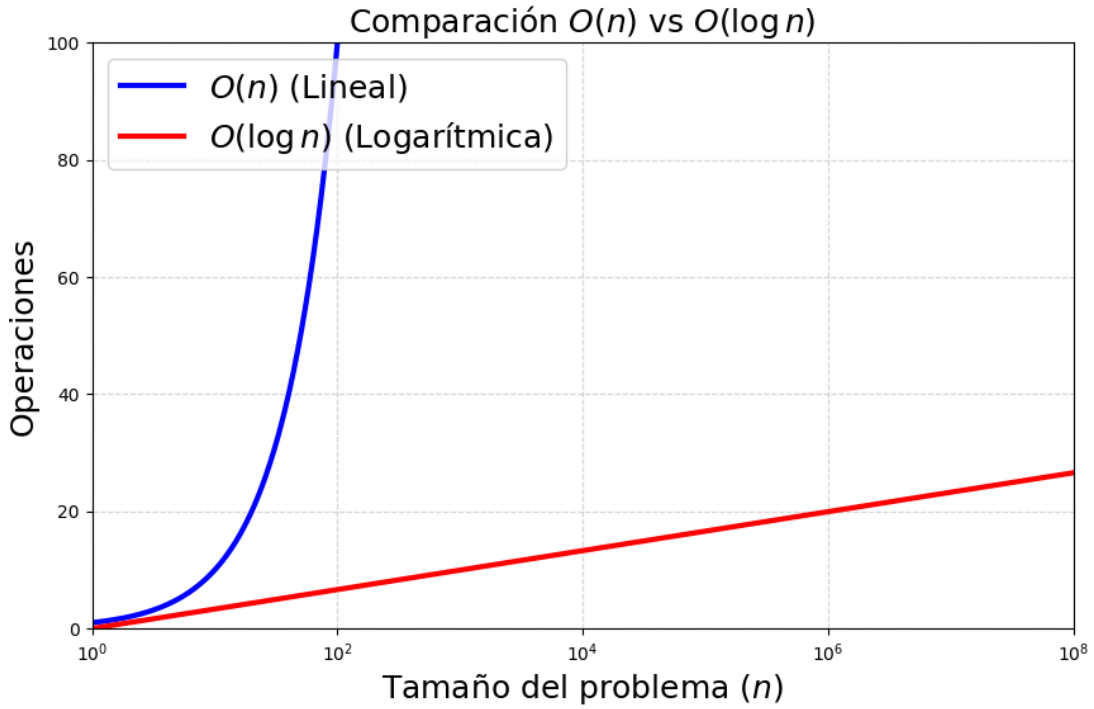
$left = right = 3$

✗	✗	✗	5	✗	✗

✅

Complejidad

Para un rango de tamaño n, lo que vamos haciendo es dividirlo a la mitad en cada iteración, por lo que el número de iteraciones es $\log(n)$. También ocupamos una función para evaluar si la condición se cumple, entonces si la complejidad de esa función es $O(f)$, la complejidad total del algoritmo es $O(f \times \log(n))$.



Código en C++

Siendo `f` una función que retorna un booleano para buscar el **primer valor donde la condición es verdadera**, un ejemplo de código sería:

```
int left = 0, right = n; // Puede ser un rango distinto

while (left < right) {
    int mid = left + (right - left) / 2; // Evitamos overflow

    if (f(mid))
      right = mid;
    else
      left = mid + 1;
}

// left es la variable que contiene el valor buscado y podemos revisar si cumple con f(left)

bool found = f(left);
```

Código en C++

Siendo `f` una función que retorna un booleano para buscar el **ultimo valor donde la condición es verdadera**, un ejemplo de código sería:

```
int left = 0, right = n; // Puede ser un rango distinto

while (left < right) {
    int mid = left + (right - left + 1) / 2; // Evitamos overflow

    if (f(mid))
      left = mid;
    else
      right = mid - 1;
}

// left es la variable que contiene el valor buscado y podemos revisar si cumple con f(left)

bool found = f(left);
```

Lower bound

El **lower bound** es el primer elemento de una lista que es mayor o igual a un valor dado. Se puede programar con búsqueda binaria o usar `std::lower_bound`.

Siendo `values` un vector ordenado de menor a mayor, el código sería:

```
int left = 0, right = values.size();

while (left < right) {1
    int mid = left + (right - left) / 2

    if (values[mid] >= target)
      right = mid;
    else
      left = mid + 1;
}

cout << "Lower bound index " << left << "\n";
```

Upper bound

El **upper bound** es el primer elemento de una lista que es estrictamente mayor a un valor dado. Se puede programar con búsqueda binaria o usar `std::upper_bound`. Siendo `values` un vector ordenado de menor a mayor, el código sería:

```
int left = 0, right = values.size();

while (left <= right) {
    int mid = left + (right - left) / 2;

    if (values[mid] > target)
      right = mid;
    else
      left = mid + 1;
}

cout << "Upper bound index " << left << "\n";
```

Búsqueda binaria con números reales

También la búsqueda binaria puede ser aplicada con números reales, como encontrar la raíz cuadrada de un número, pero debemos tener en consideración la presición. Por eso debemos de definir un valor de tolerancia que indica cuando podemos detener la búsqueda con la diferencia de los extremos del rango.

```
const double EPS = 1e-9; // Epsilon

double left = 0, right = n; // Puede ser un rango distinto

while (right - left > EPS) {
    double mid = left + (right - left) * 0.5;

    if (f(mid))
        left = mid;
    else
        right = mid;
}
```

Fight the Monster

Yang (**Y**) y un monstruo (**M**) tienen atributos: HP (vida), ATK (ataque) y DEF (defensa).

Cada segundo ocurren estas acciones simultáneamente:

- El monstruo pierde: $max(0, ATK_Y - DEF_M)$
- Yang pierde: $max(0, ATK_M - DEF_Y)$

Puede comprar mejoras antes de empezar: **+1** HP cuesta h, **+1** ATK cuesta a y **+1** DEF cuesta d.

Yang gana si el monstruo muere antes que él, se pide el costo mínimo para que Yang gane. Todos los valores son números enteros entre 0 y 100.

🔗 Link del problema: codeforces.com/problemset/problem/487/A
🔗 Link del código solución: miniurl.cl/l3y1gr

GukiZ hates Boxes

Profesor GukiZ quiere llegar a clases, pero hay n ($1 \leq n \leq 10^5$) pilas de cajas bloqueando su camino. La i-ésima pila tiene $a[i]$ cajas ($0 \leq a[i] \leq 10^9$). Hay m ($1 \leq m \leq 10^5$) estudiantes dispuestos a ayudar a removerlas.

Al segundo 0, todos los estudiantes están antes de la primera pila. Luego pueden realizar estas operaciones, cada una tomando 1 segundo:

- Moverse de la pila i a la pila $i + 1$ (si $i \neq n$).
- Quitar una caja de la pila en la que estén (si no está vacía).

Los estudiantes trabajan en paralelo. Se pide calcular el tiempo mínimo necesario para eliminar todas las cajas.

🔗 Link del problema: codeforces.com/contest/551/problem/C

Task Odašiljači

Debes conectar una red de n antenas distribuidas en un desierto representado como un plano 2D. Cada antena tendrá un radio de transmisión r (el mismo para todas). El alcance de una antena es el conjunto de puntos cuya distancia a ella es $\leq r$. Dos antenas pueden comunicarse directamente si sus alcances se superponen y la comunicación es transitiva.

Se te entregan n ($1 \leq n \leq 100$) puntos en un plano 2D con coordenadas enteras (x_i, y_i) ($-10^9 \leq x_i, y_i \leq 10^9$) representando las posiciones de las antenas.

Encuentra el radio mínimo r para que todas las antenas puedan comunicarse entre sí con precisión de 10^{-6}.

🔗 Link del problema: oj.uz/problem/view/COCI20_odasiljaci
🔗 Link del código solución: miniurl.cl/6zyp8r