Niñas PRO(grama doras)

Niñas PRO(grama doras)

Búsqueda Binaria

Niñas PRO(grama doras)

Temario

- ★ Analizar el costo.
- ★ Búsqueda binaria.
- ★ Buscar el máximo elemento en un arreglo ordenado y en uno no ordenado.

Análisis de Costo

Cuando vamos a diseñar una solución, siempre debemos fijarnos en cuánto se demoraría nuestro programa.

A la derecha puedes ver el código de **búsqueda lineal en un arreglo ordenado**. En el peor de los casos, en que tengamos que recorrer todo el arreglo, tendremos que realizar **n** iteraciones... ¿qué pasa cuando **n es muy grande**?

En el peor de los casos tenemos que realizar **n** iteraciones

```
for (int i = 0; i < n;  i++)
{
  if (arreglo[i] > elemento)
    return -1;  // No está
  if (arreglo[i] == elemento)
    return i;
}
// Búsqueda lineal en un arreglo
ordenado
```

Análisis de **costo**

El costo se mide en la **cantidad de iteraciones que se deben realizar en el peor de los casos**.

En el ejemplo anterior, este consistía en **n** iteraciones, lo que denotamos como $O(n)$.

Actividad

Indica el costo de búsqueda lineal en los siguientes casos.

1. `int matriz3d[m][n]`
2. `int matriz3d[m][n][o]`
3. `string palabra, con n caracteres.`

Niñas PRO(grama doras)

Búsqueda Binaria

Busquemos el elemento **33** en el siguiente arreglo:

-123	0	12	20	33	49	90
0	1	2	3	4	5	6

Búsqueda Binaria

Nos posicionamos en la mitad del arreglo:

```
int mitad = 7/2 = 3
```

-123	0	12	20	33	49	90
0	1	2	3	4	5	6

Búsqueda Binaria

Como 33 > 20 (el elemento de la mitad), descartamos la mitad izquierda y nos quedamos con la mitad derecha:

-123	0	12	20	33	49	90
0	1	2	3	4	5	6

Búsqueda Binaria

Busquemos el elemento **33** en el siguiente arreglo:

-123	0	12	20	33	49	90
0	1	2	3	4	5	6

Esta vez, 49 es el elemento de la mitad. Como 33 < 49, nos quedamos con la mitad izquierda.

Búsqueda Binaria

Busquemos el elemento **33** en el siguiente arreglo:

-123	0	12	20	33	49	90
0	**1**	**2**	**3**	**4**	**5**	**6**

¡El arreglo tiene un único elemento que justamente es 33! Así que retornamos **4**. En caso que no hubiesen sido iguales, simplemente retornamos -1.

Código Búsqueda Binaria


```
using namespace std;

int busquedaBinaria(vector<int> arreglo int l, int r, int x)
{
    if (r >= l) {
        int mitad = l + (r - l) / 2;

        // Si el elemento está presente al medio
        if (arreglo[mitad] == x)
            return mitad;

        // Si el elemento es menor que el de la mitad, entonces está en la mitad izquierda
        if (arreglo[mitad] > x)
            return busquedaBinaria(arreglo, l, mitad - 1, x);

        // En caso contrario, está en la derecha
        return busquedaBinaria(arreglo, mitad + 1, r, x);
    }
    // Si llegamos aquí, es porque el elemento no está en el arreglo
    return -1;
}
```

Costo Búsqueda Binaria

A continuación mostramos el análisis del costo de realizar búsqueda binaria en un arreglo ordenado de n elementos.

Como vimos en el ejemplo anterior, el algoritmo de **búsqueda binaria** parte por la mitad el arreglo en cada iteración.
Así que, ¿cuántas veces tenemos que dividir por 2 hasta tener un solo elemento?

$\frac{n}{2^k} = 1$, con k la cantidad de veces que hay que dividir.

Podemos reescribir la expresión anterior como:

$$2^k = n$$

Aplicando logaritmo a ambos lados, obtenemos:

$$k = log_2(n)$$

Luego, el orden del algoritmo es la cantidad de veces que hay que dividir, o sea, la cantidad de iteraciones:

$$O(k) = O(log_2 n)$$

Niñas PRO(grama doras)

¿Qué veremos la próxima clase?

- ★ Algoritmos de ordenamiento con criterio mayor a menor, menor a mayor.
- ★ Algoritmos de ordenamiento con criterio alternativo.

Línea Gráfica

Lorena Gonzalez - Diseñadora
@soygonzaleztambien