Estructuras de Datos #4

Arreglos. Parte 2

Arreglos

- Es la estructura de datos más popular
- Operaciones comunes
  - Inserción
  - Búsqueda
  - Eliminación
- Cómo se diferencian estas operaciones para un 'arreglo ordenado'?
- Cómo se diferencian estas operaciones para un arreglo que no permite duplicados?

Un arreglo ordenado

- Es un arreglo en que los datos son ordenados de manera ascendente o descendente

- ¿Por qué esto podría ser una característica deseable? ¿Qué operación podría ser ejecutada mas rápidamente?

Estas en lo correcto, la búsqueda!

- Aun podemos hacer búsqueda lineal
  - Ir paso a paso revisando cada elemento
- En el caso promedio, podría ser mas rápido que un arreglo desordenado?
- También podemos realizar lo que se llama búsqueda binaria, que es mucho mas rápida
  - Especialmente para grandes arreglos

Búsqueda Binaria: Idea

- ¿Has visto algún concurso de TV de búsqueda del precio justo?
  - La idea es suponer el precio de un artículo
  - Si la suposición es demasiado baja, el animador dice "más alto"
  - Si la suposición es demasiado alta, el animador dice "inferior"

- Esto puede funcionar si estamos usando arreglos ordenados
  - Compruebe el elemento medio
  - Si es demasiado bajo, restringir la búsqueda a la primera mitad de la matriz
  - Restringir de otro modo de búsqueda para la segunda mitad de la matriz
  - Y repetir.

Tenga en cuenta lo que esto puede ahorrar!

- Vemos un caso simple, donde buscamos un ítem en un arreglo de 100 elementos:
  - int[] arr = {1,2,3,4,5,6,.....,100}

- Para un arreglo desordenado donde realizamos busqueda linear, ¿cuantas comparaciones en promedio deben ser realizadas?

- ¿Qué hay sobre la búsqueda binaria en un arreglo ordenado? Busquemos el elemento 33.

Búsqueda binaria

- Arreglo tiene valores 1-100
- Primera búsqueda: Ver elemento 50
  - 50 > 33, entonces repita sobre la primera mitad (1-49)
- Segunda búsqueda: Ver elemento 25
  - 25 < 33, entonces repita sobre la segunda mitad (26-49)
- Tercera búsqueda: Ver elemento 37
  - 37 > 33, repita en la primera mita (26-36)
- Cuarta búsqueda: Ver elemento 31
  - 31 < 33, repita en la segunda mitad (32-36)
- Quinta búsqueda: Ver elemento 34
  - 34 > 33, repita en la primera mitad (32-33)
- Sexta búsqueda: Ver elemento 32
  - 32 < 33, repita en la segunda mitad (33)
- Séptima búsqueda: Ver elemento 33! Encontrado.
- Finalmente 7 comparaciones. Con búsqueda lineal, habrían sido 33.

Efecto de las operaciones

- Vimos que la búsqueda binaria mejora el rendimiento de la operación de búsqueda
- ¿Puede también mejorar la eliminación?

- ¿Que hay de la inserción, de un nuevo elemento en el arreglo ordenado?

Implementación

- Veamos la implementación en java
- En cualquier instante de tiempo:
  - lowerBound guarda el índice inferior del rango donde realizamos la búsqueda
  - upperBound guarda el índice superior del rango donde realizamos la búsqueda
  - curIn guarda el índice actual que estamos revisando
- ¿Qué pasa si el elemento no esta en el arreglo? ¿Qué sucede?

Ahora Implementemos la clase OrdArray

- Datos
  - Un arreglo
  - El numero de celdas ocupadas
- Métodos
  - Constructor
  - Size
  - Find (con búsqueda binaria)
  - Insert (con búsqueda binaria) (Propuesto)
  - Delete (con búsqueda binaria)
  - Display

Análisis

- ¿Qué hemos ganado con los arreglos ordenados?
- Es una búsqueda más rápida o más lenta?
- Es la inserción más rápida o más lenta?
- Es la eliminación más rápida o más lenta?

- Con todo, los arreglos ordenados son útiles en situaciones en las que la inserción / eliminación son poco frecuentes, pero la búsqueda es frecuente
  - Registros de empleados - la contratación / despido es menos frecuente que el acceso o la actualización de un registro de empleado

Arreglo ordenado: Conteo de Operaciones

- El máximo número de comparaciones para un arreglo ordenado de n elementos ejecutando una búsqueda binaria:

n	Comparaciones
10	4
100	7
1000	10
10000	14
100000	17
1000000	20

- ¿Cómo esto se compara con búsqueda lineal, para el caso particular de grandes arreglos? Uff.

Un análisis mas profundo

- ¿Cuantas comparaciones requiere un arreglo de 256 elementos? (2^8)
- ¿Uno de 512 (2^9)?
- ¿Crees que con 1024 debería ser (2^10)?
- Ven el patrón?

- Entonces para n valores, el número de comparaciones es $\log_2(n)+1$.
- Este es un ejemplo de un algoritmo que escala *logarítmicamente* con el tamaño de la entrada.
- Búsqueda lineal escala linealmente.

Calculando $\log_2 n$

- En una calculadora, si usas el botón “log”, usualmente es base 10. Si quieres convertirlo a base 2:
  - Multiplícalo por 3.322

- Los algoritmos que escalan logarítmicamente son preferibles a aquellos que escalan linealmente, debido a que el log de una función crece más lentamente que la misma función.

- Por lo que para grandes conjuntos de entrada, tendrás que ejecutar un número MUCHO menor de operaciones.

Como comparamos

- Es difícil simplemente decir: A es el doble de rápido que B
- Vimos con búsqueda lineal vs búsqueda binaria, que la comparación puede ser diferente cuando se cambia el tamaño de la entrada. Por ejemplo, para un arreglo de tamaño n:
- n = 16, las comparaciones de búsqueda lineal = 16, comparaciones búsqueda binaria = 5
  - La búsqueda binaria es 2x más rápido
- n = 32, las comparaciones de búsqueda lineal = 32, comparaciones búsqueda binaria = 6
  - La búsqueda binaria es 5.3x más rápido

Ejemplo: Inserción en un arreglo no Ordenado

- Supongamos que insertamos un elemento en la próxima posición disponible:
  - Posición en a[nElems]
  - Incrementar nElems

- Ambas operaciones son independiente del tamaño del arreglo n.

- Por lo que toma un tiempo, K, que no es función de n
- Decimos que esta operación es O(1), o de tiempo constante
- Lo que significa que el tiempo de ejecución es *proporcional* a 1.

Ejemplo: Búsqueda lineal

- Usted necesita un ciclo que se ejecuta en el peor de los casos n veces
- Cada vez, usted tiene que:
  - Incrementar el contador del ciclo
  - Comparar el contador del ciclo con n
  - Comparar el elemento actual con la llave
  - Cada una de estas operaciones toman un tiempo independiente de n, por lo que vamos a decir que consumen un tiempo total de K.
- Entonces, el algoritmo toma un tiempo total K * n
- A esto le decimos O (n).

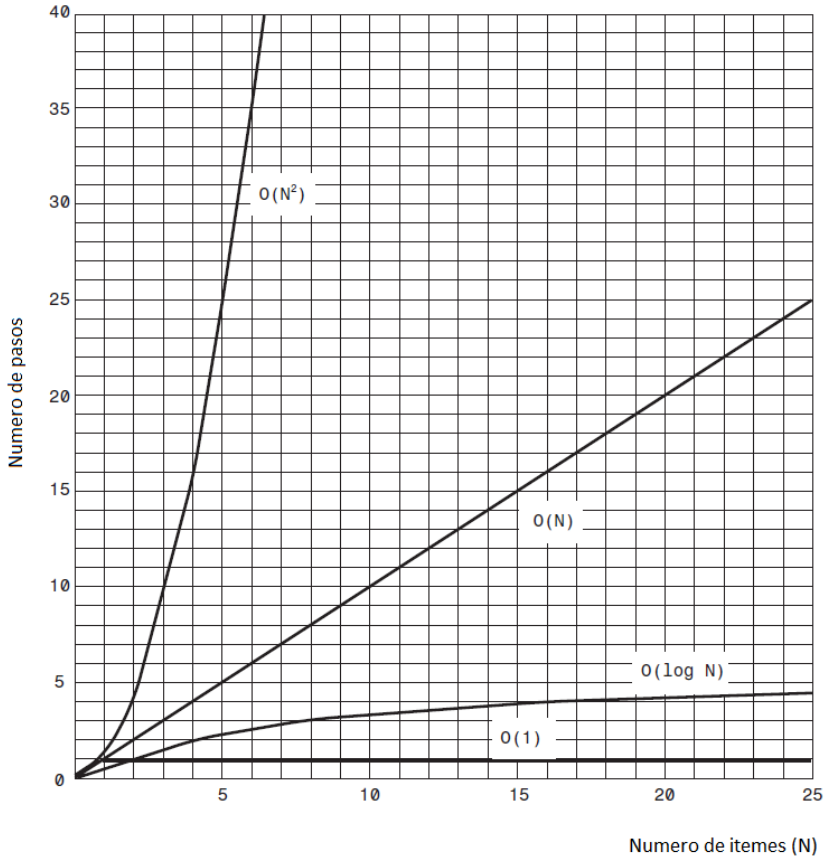
Ejemplo: Búsqueda Binaria

- Ya dijimos que para un arreglo de n elementos, necesitamos log(n)+1 comparaciones.
  - Cada comparación le toma un tiempo independiente de n, que llamamos K
- El tiempo total es entonces: K(log(n)+1) = K*log(n) + K
- Para un valor de n grande, este tiempo crece proporcionalmente a log(n), es decir, el primer termino domina.
- A esto le llamamos O(log n)

Los algoritmos que hemos discutido hasta aquí...

- Búsqueda lineal: O(n)
- Búsqueda binaria: O(log n)
- Inserción, arreglo desordenado: O(1)
- Inserción, arreglo ordenado: O(n)
- Eliminación, arreglo desordenado: O(n)
- Eliminación, arreglo ordenado: O(n)

Gráficos de la notación O()



Compensación arreglos ordenados/desordenados

- Desordenados
  - Inserción es rápida – O(1)
  - Búsqueda es lenta – O(n)
- Ordenado
  - Búsqueda es rápida – O(log n)
  - Inserción es lenta – O(n)
- Eliminación es igual – O(n)

Lo que veremos más adelante...

- Hay estructuras (árboles) en las que puedes insertar, eliminar y buscar en tiempo O(log n)
  - Por supuesto como se espera, estas son más complejas de analizar
- También aprenderemos de algunas estructuras flexibles al tamaño