Estructuras de datos
Clase teórica 2

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MEDELLÍN

Contenido

- Arreglos estáticos
- Arreglos dinámicos
- Operaciones fundamentales y sus respectivas eficiencias

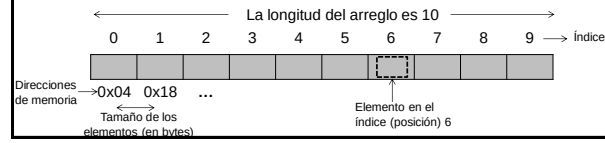
Material elaborado por: Julián Moreno

Facultad de Minas, Departamento de Ciencias de la Computación y la Decisión

Arreglo estático

Un arreglo estático es una estructura que permite almacenar un elementos de un mismo tipo. La longitud del arreglo (número elementos) se establece cuando este es creado y, a partir de allí, permanece fijo.

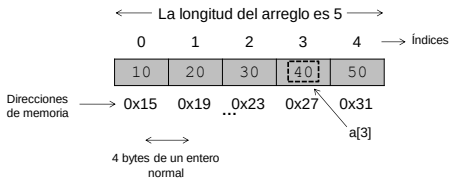
Una característica fundamental de un arreglo es que los elementos que almacena quedan guardados en posiciones consecutivas de memoria y pueden ser accedidos mediante un índice (empezando en cero).



Arreglo estático

Ejemplo en Java:

```
int[] a = new int[5];
for (int i=0; i<5; i++){
    a[i] = 10*(i+1);
}
```



Arreglo estático

Notemos en el ejemplo anterior, y en general cuando trabajamos con arreglos, que el compilador solo requiere saber tres cosas: el tipo, la dirección de memoria donde comienza a estar almacenado, y el tamaño (cantidad de elementos).

Así, acceder a un elemento del cual se conoce su índice (indexación) es sumamente fácil, basta con ir a:

Dirección donde comienza el arreglo +
(índice * tamaño en bytes del tipo que almacena)

Así por ejemplo, para un arreglo de tipo float de 1000 elementos que comienza en la dirección 0x03 (supongamos que las direcciones son números en sistema decimal), acceder al elemento 21, significa acceder al dato en la posición de memoria 87.

En otras palabras, independiente del tamaño del arreglo, la operación de indexación en un arreglo es *O(1)*

Arreglo estático

Ahora, respecto al proceso de búsqueda en un arreglo, debemos considerar dos escenarios:

- Si el arreglo está ordenado
- Si el arreglo no está ordenado

En el primer caso, lo mejor que podemos hacer es "revisar" elemento por elemento de izquierda a derecha:

```
static int busquedaLineal(int[] a, int x){
    for (int i=0; i < a.length; i++){
        if (a[i] == x)
            return i; //el primero que encuentre
    }
    return -1; //si no se encuentra
}
```

¿Cuál es la eficiencia de este algoritmo? $f = 4n+1$, por tanto $O(n)$

Arreglo estático

¿Por qué debemos considerar que el arreglo esté ordenado como un caso por aparte? ¿Será que en este caso el algoritmo de búsqueda puede ser mejor?

La respuesta es sí, mediante una búsqueda binaria:

```
static int busquedaBinaria(int[] a, int x){
    int ini = 0, fin = a.length, pos;
    while ((pos = (ini+fin)/2) >= 0){
        if (a[pos] == x)
            return pos;
        else if (x < a[pos])
            fin = pos;
        else
            ini = pos;
    }
    return -1;
}
```

¿Cuál es la eficiencia de este algoritmo? $f = 3+2\log_2(n)+1$, por tanto $O(\log_2(n))$

Arreglo estático

La búsqueda binaria es *O(log(n))*, pero hay una trampa … ¿cuánto "cuesta" ordenar? *O(n.log(n))* con un algoritmo eficiente

En otras palabras, cuando el arreglo no está ordenado, hacer una búsqueda en realidad "cuesta" *O(n.log(n))* + *O(log(n))* = *O(n.log(n))*, mientras que una búsqueda lineal en un arreglo no ordenado se puede hacer en *O(n)*

¿Se justifica entonces ordenar?

La respuesta, como nos daremos cuenta muchas veces en este curso, es: depende! ¿de qué? Pues del problema al que nos enfrentemos. Miremos por ejemplo el siguiente caso:

	Sin ordenar	Ordenando
Realizar una búsqueda en un arreglo de 100 elementos	100	$100\log_2 100 + \log_2 100 \approx 671$
Realizar 10 búsquedas en un arreglo de 100 elementos	$10*100 = 1000$	$100\log_2 100 + 10*\log_2 100 \approx 730$
Realizar 50 búsquedas en un arreglo de 100 elementos	$50*100 = 5000$	$100\log_2 100 + 50*\log_2 100 \approx 997$

Arreglo dinámico

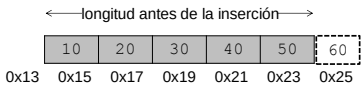
Un arreglo dinámico es …
lo mismo que uno estático, solo que dinámico

Lo anterior significa que es una estructura que permite almacenar un elementos de un mismo tipo, los cuales quedan guardados en posiciones consecutivas de memoria y pueden ser accedidos mediante un índice. Pero además, luego de creado, un arreglo dinámico puede aumentar o disminuir de tamaño (agregando o eliminando elementos) según se requiera.

Entendiéndolo así, las operaciones de indexación y búsqueda son exactamente iguales que en un arreglo estático, lo que cambia es que un arreglo dinámico tiene dos operaciones más: inserción y borrado.

Arreglo dinámico

Analicemos primero la inserción, y hagámoslo para el escenario más sencillo: se ingresa un nuevo elemento al final del arrreglo



Durante esta operación el compilador solicita la memoria adicional y puede pasar una de dos cosas:

- Que dicha memoria (en el ejemplo anterior 0x25 y 0x26) esté desocupada
- Que esté ocupada

El primer caso es el favorable pues lo único que hay que hacer es reservar ese espacio de memoria y luego guardar allí el elemento, es decir, en este caso la operación es *O(1)*

Arreglo dinámico

El segundo caso es el problemático porque el nuevo elemento no se puede pedir en otro espacio de memoria (dejaría de ser un arreglo: elementos en posiciones contiguas de memoria) ¿Qué puede hacer entonces el compilador?

La única alternativa es mover todo el arreglo a una nueva dirección de memoria donde quepa todo lo que ya tenía, más lo nuevo.

En otras palabras tiene que:

1. Buscar un espacio de memoria suficiente
2. Reservar dicho espacio de memoria
3. "Trastear" los elementos que ya estaban y agregar el nuevo

Lo cual, en total, implica *O(1)* + *O(1)* + *O(n)* = *O(n)*

Arreglo dinámico

Analicemos ahora un escenario más complejo: ingresar un elemento en la posición *i* del arreglo (*0 ≤ i ≤ n-1*)

En el caso favorable, es decir, cuando hay memoria disponible a la derecha, habría que:

1. Reservar ese espacio de memoria
2. Mover los elementos *n-1* hasta el *i* una posición a la derecha
3. Guardar el elemento nuevo en la posición *i*

El costo final de esta operación sería entonces: *O(1)* + *O(n)* + *O(1)* = *O(n)*

Arreglo dinámico

Mientras que en el caso desfavorable habría que:

1. Reservar ese espacio de memoria
2. Trastear los elementos *1* hasta el *i-1* a la nueva memoria
3. Trastear los elementos *i* hasta el *n-1* a la nueva memoria
4. Guardar el elemento nuevo en la posición *i* de la nueva memoria

El costo final de esta operación sería entonces: *O(1)* + *O(n)* + *O(1)* = *O(n)*

Arreglo dinámico

Finalmente, analicemos cómo se borra un elemento. Para esto hay que analizar dos casos:

- Si se conoce el índice del elemento
- Si no se conoce

En el primer caso hay que:

1. Mover los elementos *i+1* hasta el *n-1* una posición a la izquierda
2. Liberar el espacio de memoria n-1

El costo de esta operación sería entonces: *O(n) + O(1) = O(n)*

En el segundo caso hay que, antes de los dos pasos anteriores, buscar la primera ocurrencia del elemento. Esto, como ya vimos, se puede hacer en O(n)

Así, el costo total sería: *O(n) + O(n) + O(1) = O(n)*

Tabla resumen

Recapitulando la clase de hoy tenemos que (en general esto es lo que haremos durante todo el curso con las diferentes estructuras):

Estructura	Inserción	Indexación	Búsqueda	Borrado
Arreglo estático	No Aplica	*O(1)*	*O(log(n))* si el arreglo está ordenado* *O*(*n*) en caso contrario	No Aplica
Arreglo dinámico	*O(n)*	*O(1)*	*O(log(n))* si el arreglo está ordenado* *O*(*n*) en caso contrario	*O(n)*

*En caso que se requiera, ordenar "cuesta" *O(n.log(n))*