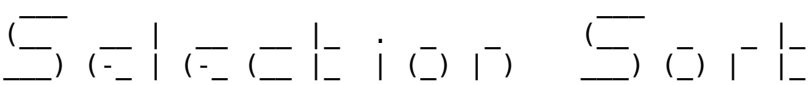
"Análisis de Algoritmos"
José Helo Guzmán



Contenido

▬▬▬ 1. Introducción.

En esta sección vamos a continuar con el uso de arreglos y listas. Estas son dos de las estructuras más básicas y utilizadas por todos los programadores.

Se presenta un algoritmo de ordenamiento conocido como el “selection sort”. Esta es una técnica de programación que corresponde al proceso de “decrease and conquer”.

Una gran cantidad de algoritmos funciona únicamente si sus datos se encuentran ordenados. El método de “selection sort”, es un paso en la dirección de otros algoritmos de ordenamiento más rápidos.

▬▬▬ 2. Conceptos Básicos de Arreglos y Listas Enlazadas.

Como probablemente saben, los arreglos utilizan posiciones contiguas de memoria. Por esta razón, se puede acceder a ellos utilizando un subíndice. Supondremos que conocemos la primera posición del arreglo, En este caso el vector inicia en la posición 1, por ejemplo:

Posición Memoria	Valor
[1]	10
[2]	20
[3]	30
[4]	40
⋮	:

De esta forma si se tiene el vector:

El punto de inicio del vector es la posición [0].

vector:[10, 20, 30, 40]

En general el vector se puede accesar como:

vector[inicio + desplazamiento]

Se tiene que vector[1] = vector[0+1] = 10

Se tiene que vector[2] = vector[0+2] = 20

Se tiene que vector[3] = vector[0+3] = 30

Si se desea agregar un nuevo elemento (en el caso que el lenguaje de programación lo permita) se debe mover todo el arreglo a una nueva posición de memoria que tenga el suficiente espacio.

Por ejemplo si se desea agregar un nuevo elemento con valor de 50 al vector anterior, y la posición [5] estuviera ocupada, se necesita buscar un nuevo espacio de memoria. El vector inicia ahora en la posición [21].

Posición Memoria	Valor
⋮	
[21]	10
[22]	20
[23]	30
[24]	40
[25]	50
⋮	

Este vector iniciará en la posición [20].

Cuando se indica

vector[1] = vector[20 + 1] = vector[22] = 10

vector[2] = vector[20 + 2] = vector[22] = 20

Las listas enlazadas trabajan de manera diferente, utilizan posiciones dinámicas de memoria. Conocemos la dirección del primer elemento, pero no de los demás elementos. Cada elemento indica donde se encuentra el siguiente elemento:

Posición Memoria	Valor	Sig
[1]	10	[3]
[2]	()	
[3]	20	[4]
[4]	30	[6]
[5]	()	
[6]	40	nulo
[7]	()	
[8]	()	

Si se desea agregar un nuevo elemento, es suficiente con poner el nuevo elemento y señalar que la lista continua.

Posición Memoria	Valor	Sig
[1]	10	[3]
[2]	()	
[3]	20	[4]
[4]	30	[6]
[5]	()	
[6]	40	[8]
[7]	()	
[8]	50	nulo

Por esta razón, cuando se tienen listas, nunca es necesario moverlas de la posición en la que se encuentran. Es fácil agregar nuevos elementos.

▬▬▬ 3. Comentarios sobre Arreglos y Listas Enlazadas.

Debido a lo anterior, el tiempo que se tarda en leer una posición del arreglo es un tiempo O(1), se puede acceder de forma inmediata.

El tiempo que tarda en leer un elemento de la lista es de tiempo O(n), pues hay que recorrer toda la lista para poner el nuevo elemento.

El tiempo para insertar un nuevo elemento en un arreglo es O(n), pues en este caso hay que mover todo el arreglo.

El tiempo para insertar un elemento en una lista es O(1), pues solo se necesita poner el nuevo elemento.

La siguiente tabla muestra estos tiempos:

Operación	Arreglos	Listas
Leer	O(1)	O(n)
Insertar	O(n)	O(1)

●● Insertar Elementos.

Supongamos que tenemos una lista, la cual debe mantenerse en un cierto orden. En caso que se desee insertar un nuevo elemento, por ejemplo en la mitad. Si usamos un arreglo tenemos que utilizar una nueva posición de memoria.

En el caso de las listas, podemos agregar nueva información, sin tener que cambiar de lugar la lista. Únicamente agregaremos una nueva posición.

●● Borrar Elementos.

De igual forma, si se desea borrar un elemento de un arreglo, se deben mover de lugar todos los elementos del arreglo para eliminar el elemento escogido. A diferencia de las inserciones, no es necesario mover el arreglo a una nueva posición de memoria.

En el caso de la lista, si se desea eliminar un elemento, es suficiente con cambiar los punteros de manera adecuada y de esta forma, ya no se tendrá el elemento.

●● Tiempos para las Operaciones.

Se tienen entonces los siguiente tiempos para las diferentes operaciones.

Operación	Arreglos	Listas
Leer	O(1)	O(n)
Insertar	O(n)	O(1)
Eliminar	O(n)	O(1)

En general es mejor usar los arreglos cuando se tiene una cantidad conocida de elementos que van a cambiar poco.

Otro elemento importante para escoger esta estructura es el tipo de acceso. Si se necesita un acceso aleatorio o un acceso secuencial.

Como su nombre lo indica un acceso aleatorio significa que se puede necesitar cualquiera de los elementos, por ejemplo si se necesita el elemento con el índice [10] se puede ir directamente a esa posición.

Un acceso secuencial es que los elementos se van a utilizar en una secuencia, uno después del otro. De esta forma si se necesita ir a la posición [10], se deben recorrer las primeras 9 posiciones antes de llegar a la posición indicada.

●● Lenguajes de Programación Actuales.

Muchos lenguajes de programación modernos, permiten tener una misma estructura a la cual se puede tener acceso como un vector o como una lista. En estos casos, para mantener los mejores tiempos posibles se suele usar una gran cantidad de punteros, y por lo tanto se gasta más memoria, para permitir el uso de vector/lista.

▬▬▬ 4. Selection Sort.

En este caso estamos suponiendo que estamos tratando de ordenar los elementos de menor a mayor. El “selection sort” busca ordenar un arreglo, utilizando la técnica de extraer de forma repetida el menor elemento.

Se tratan de mantener dos listas, una con los elementos no ordenados, y otra con los elementos ordenados.

Por ejemplo veamos el siguiente ejemplo. Supongamos que tenemos una lista con los siguientes valores de las canciones que tienen el mayor número de reproducciones en Spotify. Deseamos ordenar la lista del menos escuchado al más escuchado.

>>> Para ello se usan dos listas:

lista: [156, 141, 35, 94, 88, 61, 111]

resultado: []

>>> Primer paso.

Se encuentra el menor de la lista: 35.
Se borra de la lista.
Se agrega al resultado.

lista: [156, 141, 94, 88, 61, 111]

resultado: [35]

>>> Segundo paso.

Se encuentra el menor de la lista: 61.
Se borra de la lista.
Se agrega al resultado.

lista: [156, 141, 94, 88, 111]

resultado: [35, 61]

```
>>> Tercer paso.

Se encuentra el menor de la lista: 88.
Se borra de la lista.
Se agrega al resultado.

lista: [156, 141, 94, 111]

resultado: [35, 61, 88]



>>> Cuarto paso.

Se encuentra el menor de la lista: 94.
Se borra de la lista.
Se agrega al resultado.

lista: [156, 141, 111]

resultado: [35, 61, 88, 94]



>>> Quinto paso.

Se encuentra el menor de la lista: 111.
Se borra de la lista.
Se agrega al resultado.

lista: [156, 141]

resultado: [35, 61, 88, 94, 111]



>>> Sexto paso.

Se encuentra el menor de la lista: 141.
Se borra de la lista.
Se agrega al resultado.

lista: [156]

resultado: [35, 61, 88, 94, 111, 141]



>>> Sétimo paso y último.

Se encuentra el menor de la lista: 156.
Se borra de la lista.
Se agrega al resultado.

lista: []

resultado: [35, 61, 88, 94, 111, 141, 156]
```

▬▬▬ 5. Tiempo de Ejecución.

Cada vez que se busca el mínimo en la lista, se debe recorrer, en el peor de los casos, toda la lista. Esto se puede observar con claridad al tener una lista como la que se presenta a continuación.

>>> Para el primer mínimo:

lista: [156, 141, 111, 94, 88, 61, 35]

resultado: []

Para encontrar el valor mínimo 35 se necesitan 7 comparaciones.

>>> Si se tiene:

lista: [156, 141, 111, 94, 88, 61]

resultado: [35]

Para encontrar el valor mínimo que es 61 se necesitan 6 comparaciones.

>>> Si se tiene:

lista: [156, 141, 111, 94, 88]

resultado: [35, 61]

Para encontrar el valor mínimo de 88 se necesitan 5 comparaciones.

>>> Y así sucesivamente hasta llegar a:

lista: []

resultado: [35, 61, 88, 94, 111, 141, 156]

En este momento se han realizado la cantidad de comparaciones

$$7 + 6 + 5 + 4 + 3 + 2 + 1 + 0 = \left(\frac{7*(7+1)}{2} \right)$$

$$7 + 6 + 5 + 4 + 3 + 2 + 1 + 0 = 28$$

De forma general, si la lista tiene un tamaño de “n” elementos se realizan:

$$n + (n-1) + (n-2) + \ldots + 1 + 0 = \left(\frac{n*(n+1)}{2} \right)$$

$$\left(\frac{n*(n+1)}{2} \right) = \left(\frac{n^2 + n}{2} \right) = \left(\frac{n^2}{2} + \frac{n}{2} \right)$$

Que es un polinomio de grado 2 y su orden es de $O(n^2)$

●● Usos de los Algoritmos de Ordenamiento.

Los algoritmos de ordenamiento son de gran utilidad, se pueden usar para:

Ordenar los nombres en un libro o palabras en un diccionario.

Ordenar las fechas de viajes en aplicaciones.

Ordenar “emails” del más nuevo al más viejo o viceversa.

Ordenar los directorios de una computadora.

▬▬▬ 6. Resumen.

• El manejo de la memoria de la computadora obliga a usar diferentes tipos de datos.

• Cuando se utiliza un arreglo, todos los elementos se almacenan de forma continua en la memoria.

• Cuando se utiliza una lista enlazada, los elementos no tienen que estar de forma continua. Cada elemento indica donde se encuentra el siguiente.

• Los arreglos permiten leer con mucha rapidez.

• Las listas enlazadas permiten hacer inserciones y borrados de forma rápida.

▬▬▬ 7. Ejercicios.

●● Ejercicio 1.

Suponga que está programando una aplicación para mantener un registro de sus gastos. Cada día pone en la lista la actividad en que gastó dinero. Por ejemplo

1. Comida $10.
2. Ir al cine $15.
3. Membrecía de SFBC $12.

Cada día que pasa, se agregan más actividades. Al final del mes, revisa su lista de gastos y suma todos los gastos para saber cuánto dinero gastó. De esta forma se tienen una gran cantidad de “inserts” y no tantas operaciones de “read”.

Qué clase de estructura será mejor para este proceso, un arreglo o un vector?

●● Ejercicio 2.

Supongamos que deseamos construir una aplicación para tomar órdenes de restaurantes. La aplicación debe almacenar las órdenes conforme se van generando.

Cada vez que se encarga una nueva comida, se debe agregar una orden al final de la fila. Cuando una orden se envía esta se quita del inicio de la fila.

Qué clase de estructura será mejor para este proceso, un arreglo o un vector?

●● Ejercicio 3.

Suponga que la aplicación de Facebook tiene una lista con todos sus usuarios. Cuando alguien trata de entrar a Facebook, se realiza una búsqueda para encontrar el nombre. Si el nombre está en la lista puede hacer el “log in”. Las personas entran y salen de Facebook muy a menudo, muchas veces al día, así que hay una gran cantidad de búsquedas. Supongamos que se usa un algoritmo de búsqueda binaria. La búsqueda binaria necesita acceso aleatorio, se debe poder llegar a la mitad de la lista de forma instantánea. Sabiendo esto, implementaría esta lista de usuarios con un arreglo o con una lista enlazada?

●● Ejercicio 4.

Mucha gente suele hacer un “sign up” en Facebook también. Suponga que decide utilizar un arreglo para guardar la lista de usuarios. Qué desventajas se tienen al utilizar un arreglo cuando van a ocurrir tantos “inserts”? Se puede utilizar una búsqueda binaria con esta estructura?

●● Ejercicio 5.

En la realidad Facebook no utiliza ni un arreglo ni una lista enlazada para mantener la información de sus usuarios. Se utiliza una estructura híbrida, un arreglo de listas enlazadas. De esta forma se tiene un arreglo con 29 campos, las letras del alfabeto, y en cada uno de estos campos se tiene un puntero hacia una lista enlazada.

Por ejemplo, el primer campo de este arreglo, tiene un puntero señalando todos los usuarios que inician con la letra “a”, el segundo campo tiene un puntero señalando todos los usuarios que inician con la letra “b”, y así sucesivamente.

Arreglo Listas Enlazadas

[1][A] ------>[Ana]-->[Abel]-->[Alicia] --> ...

[2][B] ------>[Brenda]-->[Beatriz]-->[Bruno]--> ...

[3][C] ------>[Carlos]-->[Camilo]-->[Carla]-->[Cecilia] --> ...

Supongamos que Adela hace un “sign up” y se desea agregar a la lista. Se debe ir al slot[1], que tiene el puntero a la lista con todos los nombres que inician con “A”. Ahí se agrega el nombre de Adela al final de la lista.

Supongamos que deseamos buscar a Zakhir. Se debe de ir a la posición 29 del arreglo, el slot [29][Z], y se debe buscar de forma secuencial a lo largo de la lista de nombres.

Compare este sistema híbrido con un sistema de arreglos y listas enlazadas.
Es más lento o más rápido?
No tiene que calcular el tiempo de la Gran O, pero justifique su respuesta para indicar si es más lento o más rápido.

●● Ejercicio 6.

Se puede programar el “selection sort” utilizando un único arreglo.
Una parte del arreglo está ordenada y otra parte no lo está.
Para ello se busca el menor elemento y se intercambia con el elemento al inicio del arreglo.

```
[156, 141,  35,  94,  88,  61, 111]
^
[ 35, 141, 156,  94,  88,  61, 111]
    ^
[ 35,  61, 156,  94,  88, 141, 111]
        ^
[ 35,  61,  88,  94, 156, 141, 111]
           ^
[ 35,  61,  88,  94, 156, 141, 111]
              ^
[ 35,  61,  88,  94, 111, 141, 156]
                  ^
[ 35,  61,  88,  94, 111, 141, 156]
                      ^
[ 35,  61,  88,  94, 111, 141, 156]
                          ^
```

Cuál es el valor del O(n) de este algoritmo?