MEMORIA ESTATICA Y DINAMICA

INTRODUCCIÓN

Tu ordenador probablemente usa ambas, memoria estática y memoria dinámica al mismo tiempo, pero las usa por diferentes razones debido al coste entre los dos tipos de memoria RAM. Si entiendes como los chips de RAM dinámica y RAM estática funcionan internamente, es fácil ver porqué el coste es diferente, y también podrás entender su nombre.

MEMORIA ESTÁTICA

Las técnicas de asignación de memoria estática son sencillas. La asignación de memoria puede hacerse en tiempo de compilación y los objetos están vigentes desde que comienza la ejecución del programa hasta que termina. En los lenguajes que permiten la existencia de subprogramas, y siempre que todos los objetos de estos subprogramas puedan almacenarse estáticamente se aloja en la memoria estática un registro de activación correspondiente a cada uno de los subprogramas. Estos registros de activación contendrán las variables locales, parámetros formales y valor devuelto por la función.

Consideraciones

- Error en tiempo de ejecución de índice fuera del rango.
- Se debe conocer con anticipación el tamaño de la estructura.
- Se guardan en memorias adyacentes.
- Vectores, matrices, cubos, registros, archivos.

Ventajas

- La velocidad de acceso es alta.
- Para retener los datos solo necesita estar energizada.
- Lógica simple.

Son más fáciles de diseñar.

Desventajas:

- No se puede modificar el tamaño de la estructura en tiempo de ejecucion.
- No es óptimo con grandes cantidades de datos.
- Desperdicio de memoria cuando no se utiliza en su totalidad del tamaño v[100] .
- Menor capacidad, debido a que cada celda de almacenamiento requiere más transistores.
- Mayor costo por *bit.*
- Mayor consumo de Potencia

MEMORIA DINÁMICA

La memoria dinámica es un espacio de almacenamiento que se solicita en tiempo de ejecución. De esa manera, a medida que el proceso va necesitando espacio para más líneas, va solicitando más memoria al sistema operativo para guardarlas. El medio para manejar la memoria que otorga el sistema operativo, es el puntero, puesto que no podemos saber en tiempo de compilación dónde nos dará huecos el sistema operativo (en la memoria de nuestro PC).

Un dato importante es que como tal este tipo de datos se crean y se destruyen mientras se ejecuta el programa y por lo tanto la estructura de datos se va dimensionando de forma precisa a los requerimientos del programa, evitándonos así perder datos o desperdiciar memoria si hubiéramos tratado de definirla cantidad de memoria a utilizar en el momento de compilar el programa. Cuando se crea un programa en el que es necesario manejar memoria dinámica el sistema operativo divide el programa en cuatro partes que son: texto, datos (estáticos), pila y una zona libre o heap. En el momento de la ejecución habrá tanto partes libres como partes asignadas al proceso por lo cual si no se liberan las partes utilizadas de la memoria y que han quedado inservibles es posible que se “agote” esta parte y por lo tanto la fuente de la memoria dinámica. También la pila cambia su tamaño dinámicamente, pero esto no depende del programador sino del sistema operativo.

VENTAJAS:

- Es posible disponer de un espacio de memoria arbitrario que dependa de información dinámica (disponible sólo en ejecución): Toda esa memoria que maneja es implementada por el programador cuando fuese necesario.

- Otra ventaja de la memoria dinámica es que se puede ir incrementando durante la ejecución del programa. Esto permite, por ejemplo, trabajar con arreglos dinámicos.

- Es memoria que se reserva en tiempo de ejecución. Su tamaño puede variar durante la ejecución del programa y puede ser liberado mediante la función free.

DESVENTAJAS:

- Es difícil de implementar en el desarrollo de un programa o aplicación.

- Es difícil implementar estructuras de datos complejas como son los tipos recursivos (árboles, grafos, etc.).Por ello necesitamos una forma para solicitar y liberar memoria para nuevas variables que puedan ser necesarias durante la ejecución de nuestros programas: Heap.

- Una desventaja de la memoria dinámica es que es más difícil de manejar.

- La memoria dinámica puede afectar el rendimiento. Puesto que con la memoria estática el tamaño de las variables se conoce en tiempo de compilación, esta información está incluida en el código objeto generado. Cuando se reserva memoria de manera dinámica,

- Se tienen que llevar a cabo varias tareas, como buscar un bloque de memoria libre y almacenar la posición y tamaño de la memoria asignada, de manera que pueda ser liberada más adelante. Todo esto representa una carga adicional, aunque esto depende de la implementación y hay técnicas para reducir su impacto.

Análisis de algoritmos.

Un algoritmo es una secuencia de pasos lógica para encontrar la solución de un problema.

Todo algoritmo debe contar con las siguientes características: preciso, definido y finito. Por Preciso, entenderemos que cada paso del algoritmo tiene una relación con el anterior y el siguiente; un algoritmo es Definido, cuando se ejecuta más de una vez con los mismos datos y el resultado es el mismo; y Finito, indica que el algoritmo cuenta con una serie de pasos definidos o que tiene un fin.

Hablando de estructuras de datos podemos decir que los algoritmos según su función se dividen en:

- Algoritmos de ordenamiento
- Algoritmos de búsqueda.

Un algoritmo de ordenamiento, es el que pone los elementos de una lista o vector en una secuencia (ascendente o descendente) diferente a la entrada, es decir, el resultado de salida debe ser una permutación (reordenamiento) de la entrada que satisfaga la relación de orden requerida.

Un algoritmo de búsqueda, es aquel que está diseñado para encontrar la solución de un problema booleano de existencia o no de un elemento en particular dentro de un conjunto finito de elementos (estructura de datos), es decir al finalizar el algoritmo este debe decir si el elemento en cuestión existe o no en ese conjunto, además, en caso de existir, el algoritmo podría proporcionar la localización del elemento dentro del conjunto.

Concepto de complejidad de algoritmos.

La mayoría de los problemas que se plantean en la actualidad se pueden resolver con algoritmos que difieren en su eficiencia. Dicha diferencia puede ser irrelevante cuando el número de datos es pequeño pero cuando la cantidad de datos es mayor la diferencia crece. Ejemplo: Suma de 4 y 10 primero números naturales.

1+2+3+4 = 10	3 tiempo	3	4*(4+1)/2 = 10
1+2+3+4+5+6+7+8+9+10 = 55	9	3	10*(10+1)/2 = 55

La *complejidad de un algoritmo* o *complejidad computacional*, estudia los recursos y esfuerzos requeridos durante el cálculo para resolver un problema los cuales se dividen en: *tiempo de ejecución* y *espacio en memoria*. El factor tiempo, por lo general es más

importante que el factor espacio, pero existen algoritmos que ofrecen el peor de los casos en un menor tiempo que el mejor de los casos, lo cual no es la mejor de las soluciones.

El factor tiempo de ejecución de un algoritmo depende de la cantidad de datos que se quieren procesar, una forma de apreciar esto es con las cuatro curvas que se presentan en las gráficas de la figura 1.1.

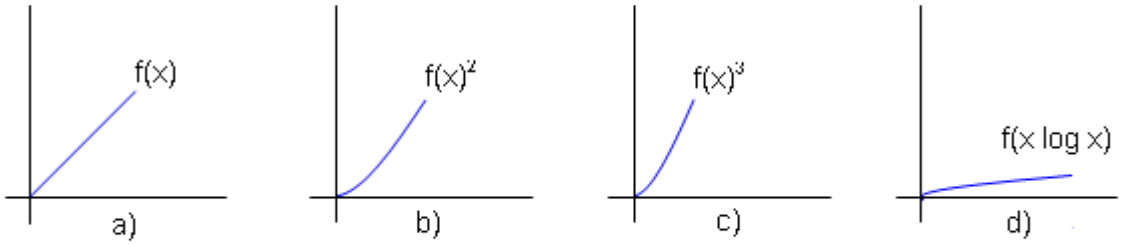


Figura 1.1 Funciones típicas para el análisis de un algoritmo.

Como se puede apreciar en las gráficas, entre mayor se al número de datos mayor tiempo se aplica en las gráficas a), b) y c), lo cual no ocurre con la gráfica d), por lo tanto podemos deducir que una función que se acerque más al eje de las x es más constante y eficiente en el manejo de grandes cantidades de datos.

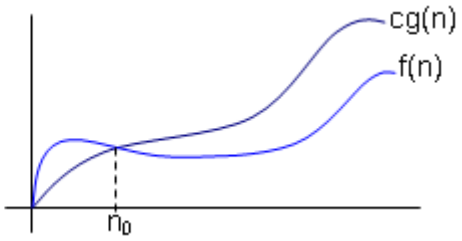
Aritmética de la notación O.

La notación asintótica “O” (grande) se utiliza para hacer referencia a la velocidad de crecimiento de los valores de una función, es decir, su utilidad radica en encontrar un límite superior del tiempo de ejecución de un algoritmo buscando el peor caso.

La definición de esta notación es la siguiente:

f(n) y *g(n)* funciones que representan enteros no negativos a números reales. Se dice que *f(n)* es *O(g(n))* si y solo si hay una constante real*c>0* y un entero constante n_0*>=1* tal que *f(n)<=cg(n)* para todo entero *n*>= n_0. Esta definición se ilustra en la figura 1.2.

Figura 1.2 Definición de la notación O. La ficción *f(n)* es *O(g(n))* para *f(n)<=c*g(n)* cuando *n>=n₀*



Nota: el orden de magnitud de una función es el orden del término de la función más grande respecto de n.

La notación asintótica “O” grande se utiliza para especificar una cota inferior para la velocidad de crecimiento de T(n), y significa que existe una constante c tal que T(n) es mayor o igual a c(g(n)) para un número infinito de valores n.

Regla para la notación O

Definición (O)T(n) es O(f(n)) si existen las constantes positivas c y n_0 tales que n >= n_0 se verifica T(n)<= cf(n).

Esta definición afirma que existe un punto inicial n_0 tal que para todos los valores de n después de ese punto; el tiempo de ejecución T(n) está acotado por algún múltiplo de f(n).

La expresión matemática de lo anterior es T(n) = O(f(n)) y el índice de crecimiento de T(n) es <= al crecimiento de f(n).

T(n) Tiempo de ejecución del algoritmo.

F(n) Tiempo al introducir los datos al algoritmo.

COMPLEGIDAD

Tiempo de ejecución de un algoritmo.

El tiempo de ejecución de un algoritmo, se refiere a la suma de los tiempos en los que el programa tarda en ejecutar una a una todas sus instrucciones, tomando en cuenta que cada instrucción requiere una unidad de tiempo, dicho tiempo se puede calcular en función de n (el número de datos), lo que se denomina T(n)

Si hacemos un análisis de forma directa al programa para determinar el tiempo de ejecución del mismo, debemos definir el conjunto de operaciones primitivas que son independientes del lenguaje de programación que se use. Algunas de las funciones primitivas son las siguientes:

- Asignación de un valor a una variable.
- Llamada a un método.
- Ejecución de una operación aritmética.

- Comparar dos números.
- Poner índices a un arreglo.
- Seguir una referencia de objeto.
- Retorno de un método.

En forma específica, una operación primitiva corresponde a una instrucción en el lenguaje de bajo nivel, cuyo tiempo de ejecución depende del ambiente de hardware y software, pero es constante. Ejemplo. Método que retorna el número mayor de un arreglo de n elementos.

```
 public int Mayor()
{
        int may=arr[0];
        for(ind=0; ind<arr.length; ind++)
                if(arr[ind]>may)
                        may=arr[ind];
        return may;
}
```

Para este ejemplo se pueden encontrar dos fórmulas que determinen el tiempo de ejecución, la primera representa el peor de los casos y la segunda el mejor de los casos. Para ser creación se sigue el programa:

- La inicialización de la variable may=arr[0], corresponde a dos unidades de tiempo.
- La inicialización del ciclo for agrega otra unidad de tiempo.
- La condición del ciclo for se ejecuta desde 1 hasta el tamaño del arreglo lo cual agrega el número de unidades del tamaño del arreglo.
- El cuerpo del ciclo for se ejecuta el tamaño del arreglo - 1 veces, para este caso el número de operaciones del cuerpo del ciclo pueden ser 6 o 4 (condición del if dos, asignación a may dos e incremento y asignación dos) en el peor o mejor de los casos respectivamente. Por consiguiente el cuerpo del ciclo contribuye con 4(tamaño del arreglo - 1) o 6(tamaño del arreglo - 1) unidades de tiempo.
- Y el retorno de may aporta una unidad de tiempo.

Con todo lo anterior se logra obtener las siguientes formulas (tamaño del arreglo o arr.length se cambian por n):

T(n) = 2+1+n+6(n-1)+1 = 7n-2 Peor de los casos.

T(n) = 2+1+n+4(n-1)+1 = 5n Mejor de los casos.

Complejidad en espacio.

La *complejidad de espacio*, se refiere a la memoria que utiliza un programa para su ejecución; es decir el espacio de memoria que ocupan todas las variables propias del programa. Dicha memoria se divide en *Memoria estática* y *Memoria dinámica*.

Para calcular la memoria estática, se suman la cantidad de memoria que ocupa cada una de las variables declaradas en el programa.

Tomando en cuenta los tipos de datos primitivos del lenguaje de programación java podemos determinar el espacio que requiere cada una de las variables de un programa, de acuerdo a lo siguiente:

Tipo de dato primitivo	Tamaño en bits	Tamaño en Bytes
byte	8	1
char	16	2
short	16	2
int	32	4
float	32	4
long	64	8
double	64	8

El cálculo de la memoria dinámica, no es tan simple ya que depende de cada ejecución del programa o algoritmo y el tipo de estructuras dinámicas que se estén utilizando.

Selección de un algoritmo.

Una de las características primordiales en la selección de un algoritmo es que este sea sencillo de entender, calcular, codificar y depurar, así mismo que utilice eficientemente los recursos de la computadora y se ejecute con la mayor rapidez posible con un eficaz uso de memoria dinámica y estática.

También para seleccionar correctamente el mejor algoritmo es necesario realizar estas preguntas:

¿Qué grado de orden tendrá la información que vas a manejar?

Si la información va a estar casi ordenada y no quieres complicarte, un algoritmo sencillo como el ordenamiento burbuja será suficiente. Si por el contrario los datos van a estar muy desordenados, un algoritmo poderoso como Quicksort puede ser el más indicado. Y si no puedes hacer una presunción sobre el grado de orden de la información, lo mejor será elegir un algoritmo que se comporte de manera similar en cualquiera de estos dos casos extremos.

¿Qué cantidad de datos vas a manipular?

Si la cantidad es pequeña, no es necesario utilizar un algoritmo complejo, y es preferible uno de fácil implementación. Una cantidad muy grande puede hacer prohibitivo utilizar un algoritmo que requiera de mucha memoria adicional.

¿Qué tipo de datos quieres ordenar?

Algunos algoritmos sólo funcionan con un tipo específico de datos (enteros, enteros positivos, etc.) y otros son generales, es decir, aplicables a cualquier tipo de dato.

¿Qué tamaño tienen los registros de tu lista?

Algunos algoritmos realizan múltiples intercambios (burbuja, inserción). Si los registros son de gran tamaño estos intercambios son más lentos.

BIBLIOGRAFÍA

https://sites.google.com/site/estdatjiq/home/unidad-vii

http://adrian-estructuradedatos.blogspot.mx/2011/04/memoria-estatica-y-dinamica.html