ANALISIS DE ESTRUCTURAS DE DATOS

Por: Leonardo Hernández

Es frecuente que al elaborar programas de computador, se necesite representar conjuntos de datos completos en la memoria RAM, los cuales pueden llegar a tener muchos elementos. Ante la imposibilidad de asignarle una variable a cada ítem, aparecen las estructuras de datos, entre las cuales tenemos: el arreglo, la lista, la cola, la pila, el árbol, las tablas asociativas (hashing), el grafo y la partición

A Continuación se analizarán algunas de las estructuras mencionadas.

El arreglo unidimensional

Cuando se crea un arreglo unidimensional se reserva memoria para un número determinado de elementos. Este espacio es fijo y no puede ser incrementado o reducido posteriormente. Esto suele ser una desventaja porque se podría desperdiciar espacio en RAM. Por ejemplo se podría declarar un arreglo de 500 elementos, para posteriormente usar solamente 20. Este tipo de desperdicio es frecuente en programas que utilizan arreglos unidimensionales.

También podría suceder que no se reserve suficiente espacio en memoria. Por ejemplo, se podría declarar un arreglo de 100 elementos, y más adelante, durante la ejecución del programa, podría necesitarse espacio para 110 elementos.

Es verdad que el arreglo puede destruirse y crearse de nuevo con el tamaño adecuado, pero este es un proceso costoso en tiempo.

El espacio en RAM donde se almacene el arreglo debe ser contiguo, lo cual es otra desventaja del arreglo. Si hay fragmentos de memoria disponibles, pero más pequeños que el tamaño total del arreglo, no pueden utilizarse para el almacenamiento. Podría incluso producirse un error de desbordamiento de memoria, overflow, habiendo suficientes bytes para contener el arreglo.

El recorrido de un arreglo permite, por ejemplo, sumar los elementos del arreglo, hallar el máximo elemento del arreglo, contar los elementos iguales a cero del arreglo, inicializar el arreglo etc. Este recorrido está en $\Theta(n)$, ya que requiere un bucle sencillo, generalmente un bucle para.

El borrado de un ítem del arreglo, desplazando a la izquierda varios elementos también en un proceso en $\Theta(n)$, en los casos medio y peor, ya que requiere un bucle sencillo. En el caso mejo el proceso está en $\Theta(1)$.

La inserción de un ítem en el arreglo requiere un bucle sencillo para desplazar varios elementos y también es un proceso en $\Theta(n)$ en los casos medio y peor y en $\Theta(1)$ en el caso mejor.

La ubicación del k-ésimo elemento en un arreglo unidimensional está en $\Theta(1)$, ya que no se requiere ningún bucle para establecer qué está almacenado en la k-ésima posición. Esta es una ventaja importante. Usar arreglos unidimensionales es apropiado cuando se requiera utilizar aleatoriamente elementos de un conjunto de datos. Esto es típico en el problema de convertir números a letras, por ejemplo, obtener a partir del 123 la cadena "ciento ventitres".

La dirección del elemento k-ésimo del arreglo está dada por:

Direccion(k) = direccion (1) + (k – 1) * tamaño de un elemento.

Ejemplo: En el arreglo de la figura, donde se supone que cada elemento necesita 2 bytes de almacenamiento, la posición del cuarto elemento se halla con:

Direccion(4) = 1000 + (4 – 1) * 2 = 1006

En la figura se observa que esta respuesta es correcta.

	[10,	20,	40,	50,	30,	...]
Posición:	1	2	3	4	5	...
Dirección	1000	1002	2004	1006	1008	...

La dirección del elemento 101 está dada por:

Direccion(101) = 1000 + (101 – 1) * 2 = 1200

El arreglo bidimensional

El arreglo bidimensional tiene muchas características similares a las del arreglo unidimensional. El recorrido de un arreglo bidimensional de nxn elementos está en $\Theta(n^2)$.

La lista

Cuando se utiliza la estructura de datos lista, se reserva espacio a medida que se va necesitando. De igual manera, cuando ya no se necesita un ítem su espacio puede ser liberado. En esto la lista supera al arreglo. Por se podría representar adecuadamente con una lista datos de soldados en algún juego de estrategia, donde el número de soldados variara constantemente a medida que avanzara el juego, a veces aumentado o a veces disminuyendo. En este caso el arreglo no sería una elección adecuada.

El espacio en RAM que contiene la lista puede ser fragmentado. Pueden aprovecharse fragmentos de RAM, más pequeños que el tamaño total de la lista, para almacenar parte de la lista. En este aspecto la lista también supera al arreglo.

La lista requiere cierto espacio adicional en RAM, ya que para cada ítem se almacena la dirección del siguiente nodo y, muchas veces, la dirección del nodo anterior. Este espacio adicional no es requerido por el arreglo. Sin embargo en la lista no se presenta el caso de memoria reservada no utilizada, como si se presenta en el arreglo.

La ubicación del k-ésimo elemento de una lista está en $\Theta(n)$, ya que requiere un bucle sencillo llegar a un ítem determinado. Está es una desventaja de la lista con respecto al arreglo. La lista no sería una elección adecuada en el citado problema de expresar números con letras.

El recorrido de una lista está en $\Theta(n)$, ya que requiere un bucle sencillo.

La inserción o borrado de un ítem está en $\Theta(1)$, ya que no se requiere ningún bucle para estas acciones. En problemas, como en el mencionado juego de estrategia, donde se requiera con frecuencia, insertar elementos en un conjunto o removerlos de él, es adecuado usar la lista.

La inicialización de una lista está en $\Theta(1)$, en esto la lista supera al arreglo.

Hay diversas clase de lista, como la lista vinculada lineal, la doblemente vinculada, la lista circular y la lista circular doblemente vinculada[2], lo que hace que esta sea un estructura muy versátil.

Otras estructuras

La pila se suele implementar como una lista, por lo que tiene características similares. También se conoce como lista LIFO, por las iniciales de *last input first output.* Son operaciones básicas de la pila: *push, pop* y *empty*[2].

La cola también se suele implementar como una lista, por lo que tiene características similares. También se conoce como lista FIFO, por las iniciales de *first input first output.*

Existe una variación de la cola llamada *cola de prioridad* en donde la salida de la cola, no depende del orden de entrada sino de una prioridad asignada a cada ítem. Por ejemplo, un sistema operativo de un computador no debe realizar las tareas en el orden en que se le soliciten sino de acuerdo a su importancia e influencia en la estabilidad del sistema.

El árbol es una estructura de gran versatilidad y de innumerables usos. Se destacan los árboles B y B^+ de uso en bases de datos y un tipo especial de árbol binarios denominado *montículo* (heap). Los montículos se representan con un simple arreglo unidimensional, se usan en el curioso y eficiente algoritmo de ordenación *heapsort* u ordenación por montículo [1].

Las tablas asociativas son usadas en Bases de Datos, son un medio extremadamente eficiente para la recuperación de datos [1].

Las estructuras de datos son indispensables en la mayoría de los programas de computador. La elección de que estructura de datos usar, no es trivial, se deben tener claras las características de cada una, para elegir la más adecuada a la situación que se esté enfrentando. La elección entre usar una lista o un arreglo merece especial atención.

BIBLIOGRAFIA.

[1] BRASSARD, Gilles y BRATLEY, Paul. Fundamentos de Algoritmia. Prentice Hall.
[2] LANGSAM, AUGENSTEIN, TENENBAUM. Estructuras de datos. Prentices Hall.