Colas con Prioridades

1. Definición

Una cola con prioridad es una estructura muy similar a una cola (ver capítulo de listas) pero, ahora, los elementos tendrán una prioridad y serán procesados atendiendo a esa prioridad. Lógicamente, a igual prioridad, se procesarán en orden de llegada a la cola.

Un ejemplo típico de cola con prioridad de la vida diaria es una guardia de un hospital. Si llega una persona con un infarto lo atenderán inmediatamente y no le dirán "espere que primero debemos atender al señor que tiene un resfriado que llegó antes que usted".
En informática, si se debe abrir un archivo se le dará una prioridad más alta que al escaneo de todos los discos en busca de algún posible virus.

Las operaciones que se deben realizar son las mismas que se definieron para las listas, pilas y colas. Es decir:
- Crear la estructura.
- Insertar un elemento.
- Retirar un elemento.

Alternativamente, se puede agregar *consultar el siguiente* y cualquier otra operación que se considere necesaria.

2. Implementación

Para su implementación se pueden utilizar las mismas estructuras de listas que se vieron con anterioridad, ya sean estáticas o dinámicas. Ahora bien, pensemos cómo se manejaría con una estructura lineal.
Si tenemos una estructura lineal de tipo lista y debemos insertar un elemento, podemos decidir hacerlo de dos formas:
- Insertamos al final como en una estructura de tipo cola.
- Insertamos en forma ordenada, en el lugar correspondiente.

En el primer caso, cuando se quisiera tomar al siguiente elemento, éste podría no estar al frente de la estructura. Entonces, ¿cómo saber cuál es el elemento que sigue? Esto se podría solucionar, fácilmente, indicando en algún campo extra su prioridad. La prioridad puede estar indicada mediante un número. Por ejemplo, si tiene prioridad 1 será la prioridad más alta, luego sigue el 2, etc. En la figura 1, se puede ver un ejemplo. El primer dato que llega a la cola es el dato C, con prioridad 2, luego llega el dato D con prioridad 3, luego el A y el H con prioridades 2 y 1, respectivamente.

Al momento de procesar un nodo, se deberá recorrer la cola hasta encontrar el dato que se debe procesar, es decir, el de prioridad más alta que se encuentre primero. En este ejemplo, el dato C tiene prioridad 2, si tuviera prioridad 1 no haría falta seguir recorriendo la cola, ya que no habrá otro con mayor prioridad. Luego, se analizará el dato D que, como tiene prioridad 3 se descartará, seguidamente, el dato A tiene prioridad 2, sin embargo, antes está el dato C que tiene la misma prioridad pero está posicionado con anterioridad. Finalmente, se analiza el dato H que, como tiene prioridad 1, será el que se procese.

Se hace notar que, si hubiera más nodos, no sería necesario seguir recorriendo la cola, teniendo en cuenta que el dato H tiene la prioridad más alta.

Por otra parte, si el dato H tuviera, por ejemplo, prioridad 2, se debería procesar el nodo que tiene el dato C, es decir, se necesitarían dos punteros o índices, uno para ir recorriendo la estructura y otro, que apunte al siguiente a procesar hasta ese momento.

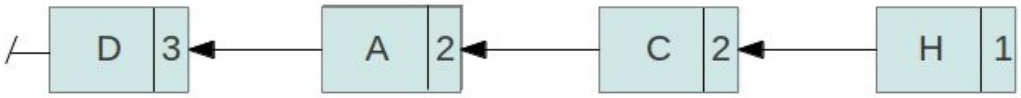
El orden de procesamiento de este ejemplo sería H, C, A y D.

En el segundo caso, si se inserta en forma ordenada, el siguiente siempre estará al frente, pero ¿cómo saber en qué lugar se debe insertar un elemento? También, en este caso, necesitamos indicar en algún campo extra la prioridad de cada elemento, para saber en qué lugar se debe insertar el mismo.

Utilizando el mismo ejemplo del caso anterior, en la figura 2 podemos observar cómo queda la estructura luego de insertar todos los nodos.

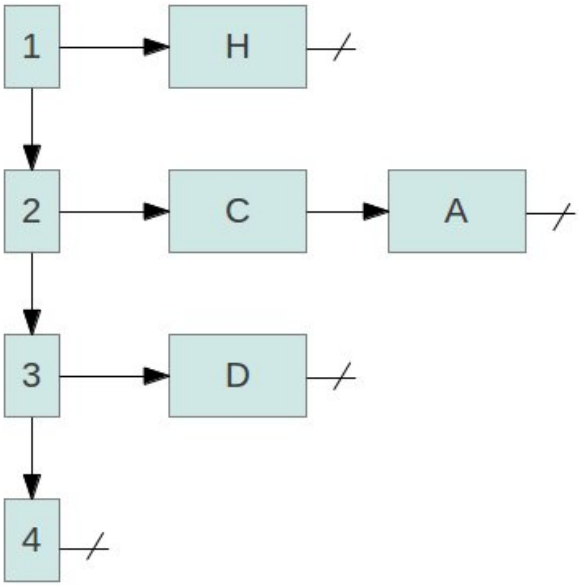
En primer lugar, cuando se va a insertar el nodo con el dato C, la estructura está vacía, por lo que no presenta ningún problema de decisión. Luego, cuando se

inserte el nodo con el dato D, como tiene prioridad 3, se inserta después del nodo que tiene al dato C, ya que este último tiene prioridad 2. En tercer lugar viene el dato A, este dato tiene prioridad 2, pero va detrás del dato C, ya que a igual prioridad se debe respetar el orden de llegada. Sin embargo, se inserta antes que el dato D, ya que su prioridad (la de A) es más alta. Finalmente, el dato H, tiene prioridad 1, por lo que va delante de todo.
De esta forma, la complicación es al momento de inserción, pero al momento de procesar, es inmediato, ya que el siguiente elemento siempre se encuentra al principio de la estructura.



Otras formas de implementación contemplan estructuras que no son lineales. Por ejemplo, una estructura del tipo lista de listas. De esta manera podríamos tener tantas colas como prioridades tengamos, sabiendo que se deberán procesar, en primer lugar, los elementos que están en la cola de mayor prioridad, luego los de la siguiente cola, etc. En este caso, no sería necesario almacenar con el dato la prioridad ya que el pertenecer a una lista u otra es lo que nos indicaría qué prioridad tiene el elemento. Obviamente, en el momento de inserción debemos saber la prioridad, para saber en cuál lista se inserta el elemento, pero no es un dato que debamos guardar en el nodo. En la figura 3 vemos el mismo ejemplo de los casos anteriores tratado de esta manera. Se asume que hay 4 prioridades, aunque no es necesario crear una lista vacía para corresponder a cierta prioridad si no se tiene ningún elemento que asignar (en este ejemplo podríamos prescindir de la lista para las prioridades número 4).
El modo de inserción es muy sencillo, en primer lugar, tenemos todas las listas (las cuatro) vacías. Cuando llega el dato C, nos deberán indicar, mediante algún parámetro, cuál es su prioridad. Como es de prioridad 2, nos dirigimos a la lista correspondiente e insertamos el nodo con el dato C (su prioridad no nos interesa, ya que nos la indica la propia lista). Luego, con el dato D, procedemos de la misma manera. Cuando llega el dato A, como es de la misma prioridad que el dato C, se agregará en la lista correspondiente. Es decir, cada una de las estructuras, se comporta como una cola tradicional, en donde se agrega al final y se toma del principio.

Para su procesamiento, se deberán ir recorriendo las listas en orden, desde el principio hasta el final.



Por supuesto, todas las implementaciones mencionadas, las podemos manejar tanto con estructuras estáticas (vectores) como dinámicas (nodos enlazados), variantes que ya hemos visto en el capítulo de listas.

Analizando el orden de los algoritmos en las implementaciones vistas tenemos (N es la cantidad de elementos en la cola):
Primera implementación. Orden de inserción 1 (ya que se inserta al final, suponiendo que tenemos un puntero o índice indicando el último nodo). Orden de procesado N (ya que se podría necesitar recorrer toda la lista para ubicar el siguiente nodo).
Segunda implementación. En esta implementación, los órdenes son contrarios a los de la anterior. La inserción tiene orden N y el procesamiento orden 1.
Tercera implementación. Tanto la inserción como el procesamiento tendrían orden K, en donde K es la cantidad de prioridades que manejamos, ya que, luego, teniendo un puntero al primer nodo de cada lista y al último, las operaciones son inmediatas.

Una estructura muy útil para el manejo de colas con prioridad es la estructura llamada *heap* (ver el capítulo correspondiente). Imaginemos un vector con datos, en donde alguno de sus campos nos dará un orden, ya sea el campo prioridad, la clave, etc. Una estructura de tipo heap, si bien puede tener el vector desordenado, siempre tendrá en el primer lugar el valor mínimo (o máximo). En este caso, tendría en primer lugar el dato con mayor prioridad. Al momento de retirar este dato, habrá que ocupar ese lugar con el siguiente de la lista pero, por las características de la estructura de tipo heap, se puede realizar esto con un muy bajo costo computacional. No tendríamos más de log2(N) movimientos, en donde N es la cantidad de elementos del vector. Pensemos que en un vector de 10 millones de elementos, a lo sumo, haríamos 24 movimientos.