LISTA ORDENADA

La operación que más frecuentemente se realiza en la mayoría de las aplicaciones informáticas que tratan con conjuntos de datos es la **búsqueda (***en inglés search***)** de un elemento dado en el conjunto. Por ejemplo cuando ingresamos al portal de la Universidad lo primero que debemos escribir es la matrícula; con este dato el algoritmo de búsqueda implementado, usualmente se denomina motor de búsqueda, busca si la matrícula ingresada coincide con alguna que tiene almacenada, Si es así mostrará la información correspondiente sino emitirá un mensaje de error. Si el conjunto se almacena en una lista lineal el algoritmo de búsqueda, como ya vimos anteriormente es O(n). Se puede mejorar el tiempo si los elementos se guardan en una ***lista ordenada***.

En una lista ordenada los elementos se encuentran ordenados de menor a mayo o de mayor a menor. Pero que significa "menor" o "mayor". Cuando los elementos son numéricos es claro, cuando es texto es el orden del diccionario, cuando son fechas es el orden cronológico.

Cuando son objetos más complejos debemos definir el orden. Por ejemplo la guía telefónica está ordenada por apellido, cuando hay varias personas con el mismo apellido por nombre y finalmente por dirección. En el ejemplo de los colectivos, éstos pueden estar ordenados por línea y número de interno:

Línea	Nro de interno	Cant. de asientos	Apto
64	5	35	No
64	12	40	Sí
64	32	38	Sí
105	28	40	No
105	30	36	Sí
152	4	29	Sí
152	28	31	No

Por consiguiente los elementos de una lista ordenada deben implementar el método ***compareTo()*** de la interfaz ***Comparable***. Si ***elem1*** y ***elem 2*** son dos objetos comparables entonces debemos implementar ***compareTo()*** tal que:

$$elem1.compareTo(elem2)\begin{cases} < 0 \ si \ elem1 \ es \ "menor" \ que \ elem2 \\ = 0 \ si \ elem1 \ es \ "igual" \ a \ elem2 \\ < 0 \ si \ elem1 \ es \ "mayor" \ que \ elem2 \end{cases}$$

El TAD Lista ordenada es muy similar al TAD Lista salvo que en lugar de insertar antes o después del elemento actual se deben realizar las inserciones manteniendo el orden. El método eliminar es algo distinto: en vez de eliminar el actual se pasa el elemento o parte del elemento a eliminar, generalmente la clave.
La lista ordenada puede ser con elementos repetidos.

Especificación del TAD Lista Ordenada

Nombre del TAD: ListaO

Representación:

Si la lista ordenada ***lst*** tiene por lo menos un elemento la representaremos como:

$$p = <e_1 ,e_2 ,e_3 ,. . . ,e_n>$$

con $e_i <= e_{i+1}$.

Si la lista ***lst*** no tiene elementos, entonces:

$$lst = \phi \quad ó \quad lst = <>$$

Constructoras:

- **ListaO: - → ListaO**
  - Construye una lista vacía, esto es solicita memoria para poder almacenar los elementos en la pila.
  - precondiciones: -
  - postcondciones: ListaO lst = ϕ

Modificadoras:

- **insertar : ListaO X Elemento → ListaO**
  - Agrega a la lista ordenada lst un nuevo elemento ***e*** manteniendo el orden
  - precondiciones: ListaO lst = $<e_1 ,e_2 ,e_3 . . . ,e_n>$ y elemento *e* tal que $e_i < e <= e_{i+1}$ ó
    ListaO lst = ϕ y elemento *e*
  - postcondciones: ListaO lst = $<e_1 ,e_2 ,e_3 . . . ,e_i , e, e_{i+1},....e_n >$ ó
    ListaO lst = $<e>$

- **eliminar: ListaO X Elemento → ListaO**
  - Saca de la lista lst el elemento indicado. Si el elemnto está repetido saca la primera aparición
  - precondiciones: ListaO lst = ϕ o lst = $<e_1 ,e_2 ,e_3 . ,e,. . ,e_n >$ y elemento e
  - postcondciones: Lista O = $<e_1 ,e_2 ,e_3 . . . ,e_{n-1} >$ o lanza excepción si la lista está vacia o el emento no esta en la lista.

Analizadoras:

- **buscar: ListaO X Elemento → Elemento**
  - Devuelve el elemento que se pasa de parámetro. Generalmente se pasa la clave y devuelve el objeto completo
  - precondiciones**:** ListaO lst = ϕ o lst = $<e_1 ,e_2 ,e_{3,} . e. . ,e_n >$ y elemento c
  - postcondciones: Elemento *e cuyo valor* de clave coincide con c

- **estaEn: ListaO X Elemento → boolean**
  - Verifica si el elemento que se pasa de parámetro está o no en la lista. Generalmente se pasa la clave
  - precondiciones**:** ListaO lst = $<e_1 ,e_2 ,e_{3,} . e. . ,e_n >$ y elemento c
  - postcondciones: Verdadero si hay algún elemento cuyo *valor* de clave coincide con c, falso en caso contrario.

- **recupear: ListaO X Entero → Elemento**
  - Devuelve el elelmnto que se encuentra en la posición k de la lista
  - precondiciones**:** ListaO lst = $<e_1, e_2, e_3, .e_k. . ,e_n>$ y 0 < k <= longitud
  - postcondciones: e_k.

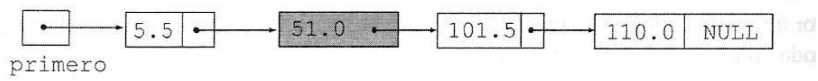
Los métodos irSiguiente(), irAnterior(), irA (int n), longitud(), esVacio(), finLista() se definen en forma similar a la lista no ordenada.

La implemnatción de la lista ordenada puede ser estática o dinámica. La estática tiene la ventaja que los métodos buscar() y estaEn() son de orden logarítmico pero las inserciones y eliminaciones son de orden O(n) ya que hay que hacer los corrimientos.

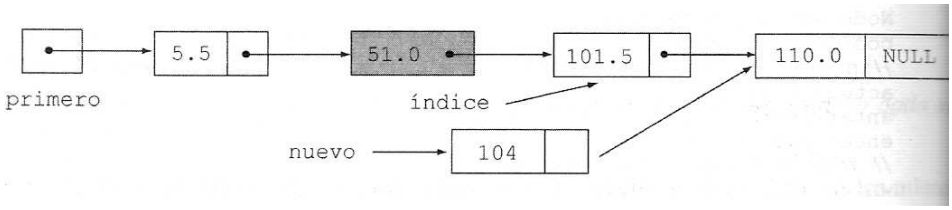
En la implementación dinámica las búsquedas son O(n) pero más eficientes (generalmente la mitad del tiempo) que en la lista no ordenada ya que no es necesario llegar al final de la lista para saber si un eleemnto se encuentra presente en ella.

La eficiencia de la búsqueda puede ser elementos de una lista tienen la propiedad de estar ordenados de forma lineal según las posiciones que ocupan en la misma.
Se dice que $\mathbf{n_i}$ precede a $\mathbf{n_{i+1}}$ para **i=1.. n-1** y que $\mathbf{n_i}$ sucede a $\mathbf{n_{i-1}}$ **para i=2..**
Lista ordenada: También es posible mantener **una lista enlazada de número reales ordenada según el dato asociado a cada nodo.**

Ejemplo: Lista enlazada ordenada de forma creciente



La siguiente figura muestra **la inserción de un elemento en dicha lista manteniendo el orden creciente de la misma.**
La forma de **insertar un elemento en una lista ordenada** es determinar en primer lugar**, la posición de inserción** y, a continuación **ajustar los enlaces.**



Para insertar el **104** necesito ***recorrer la lista hasta*** el nodo **110.0**, que es el ***nodo inmediatamente mayor.***
El ***puntero índice*** se ***queda con la dirección del nodo anterior***, a partir del cual se enlaza el nuevo nodo.

El método **insertaOrden()** crea una lista ordenada:

- Parte *de una lista vacia.*
- **Añade** a la lista vacia **nuevos elementos** de tal forma que en todo momento los **elementos están ordenados en orden creciente**.
- La **inserción del primer nodo de la lista** consiste en **crear el nodo** y **asignar su referencia a la cabeza** de la lista.
- El **segundo elemento** se ha de insertar **antes o después del primero**, dependiendo que sea **mayor o menor**.

En general para insertar en una lista ordenada, se busca primero la posición de inserción en la lista actual, o sea el nodo a partir del cual se ha de enlazar el nuevo nodo para que la lista mantenga la ordenación.
Los datos de una lista ordenada han de ser de tipo ordinal (tipo a los que se puede aplicar = =, > y <)
O bien objetos de clases que tengan definidos métodos de comparación (equals(), compareTo( ), …).