Listas, pilas y colas

David Ayuso, Vladimir Rico

Introducción listas

Una lista es una secuencia finita de n elementos de un tipo:

- Por lo general cuando hablemos de listas, las consideraremos homogéneas, es decir, sus elementos son del mismo tipo.
- Es posible encontrar listas heterogéneas, sus elementos son de tipo diferente entre sí.

Introducción a listas

- Cuando hablamos de una lista de n elementos, n es el **tamaño** de la lista.
- Cuando hablamos de la **capacidad** de una lista es el **tamaño máximo** que puede llegar a poseer.

Introducción a listas

Una **lista** puede poseer una diversa gama de funcionalidades que nos permite operar con la misma. Las comunes podrían ser:

- **Obtener un elemento** por posición o correspondencia
- **Insertar un elemento** en la posición n, se añade un elemento que se sitúa en esa posición.
- **Borrar un elemento** cuya posición corresponde a la posición n.
- **Borrar un elemento que pasamos por parámetro**. La lista busca y elimina el elemento que corresponda al proporcionado.
- Comprobar **si un elemento existe** en la lista.
- Obtener el **tamaño o longitud de la lista**.

Introducción a listas contiguas

Llamaremos **lista contigua**, a aquella cuyos elementos se almacenan en memoria de forma consecutiva.

- Puede tratarse de una lista **estática**, su capacidad y espacio en memoria **no varían**. (ej: Arrays o Vectores)
- Puede tratarse de una lista **dinámica/redimensionable**, su capacidad y espacio en memoria **pueden variar**.

Introducción a listas contiguas dinámicas

Su capacidad y espacio en memoria pueden ser alterados a lo largo de la ejecución del programa. Al variar su capacidad pueden darse una serie de casos:

- Al aumentar el espacio en memoria: hay memoria libre consecutiva a la que ya tenemos que nos brinda el espacio deseado. Sencillamente se expande la memoria ya reservada.
- Al aumentar el espacio en memoria: no tenemos memoria libre consecutiva o no tenemos la suficiente. El programa solicita un espacio de memoria libre suficiente, reserva y copia toda nuestra lista en la nueva reserva.

Introducción a listas contiguas dinámicas

Su capacidad y espacio en memoria puede ser alterado a lo largo de la ejecución del programa. Al variar su capacidad pueden darse una serie de casos:

- Disminuimos la capacidad: no hace falta realojar la memoria, simplemente el programa deja de tener reservada la memoria sobrante.

Introducción a listas contiguas dinámicas

En el peor caso, cada vez que tengamos que ampliar la capacidad de una lista, se tendría que copiar su contenido a una nueva reserva de memoria ya que no hay espacio contiguo suficiente.

Existen algunas estrategias para disminuir la frecuencia de reservar memoria y copiar el contenido a la misma, por ejemplo: cada vez que se tenga que aumentar la capacidad de la lista en n, lo hacemos en n + incremento, así nos evitamos futuros realojamientos.

Introducción a listas contiguas

Ventajas:

- El acceso a un elemento en la posición n es rápido.
- Se pueden ordenar con mucha eficiencia.
- Ocupa la cantidad casi justa de memoria, dependiendo si hemos seguido la estrategia del incremento u otra.

Desventajas:

- Se precisa de grandes zonas de memorias libres y contiguas, un problema si la memoria se encuentra fragmentada.
- Insertar y eliminar elementos son operaciones muy costosas.

Introducción a listas contiguas

En java podemos usar la clase **ArrayList<E>** donde E es el tipo de elemento que deseamos almacenar en la lista.

- Implementa la interfaz **List<E>** por lo que ya implementa las funcionalidades básicas de las listas, además de otras propias.
- Implementa la interfaz **Iterable<E>** lo que nos permite usar el bucle “foreach”.

Documentación

Introducción a listas no contiguas

Llamamos lista contigua a aquella cuyos elementos no es necesario que ocupen regiones de memoria contiguas entre sí.

- Sus elementos se almacenan en unas estructuras que llamaremos **Nodos.**
  - Estos Nodos se pueden situar en cualquier región de memoria que tenga capacidad para almacenarlos, lo que elimina la necesidad de contigüidad.
  - El concepto de **capacidad** ya no existe.
- La lista es, básicamente, una lista de Nodos.
- Sus operaciones son naturalmente recursivas.

Introducción a listas no contiguas

Un **Nodo** una estructura que posee una serie de características:

- Almacena un elemento de la lista.
- Apunta al siguiente Nodo en el orden de la lista.

La lista no contigua en sí, será la que realice las operaciones necesarias con estos nodos, por lo que el usuario se abstrae de la existencia de los mismos.

- Esta lista posee una referencia al primer nodo de la lista, que será NULL si la lista se encuentra vacía.
- Puede interesar llegar a tener una referencia al último nodo de la lista también.

Introducción a listas no contiguas

Tipos de listas no contiguas:

- Lista enlazada simple:
  - Cada nodo apunta al nodo subsiguiente.
  - El último nodo de la lista apunta a NULL, ya que no hay otro nodo después de él.
- Lista doblemente enlazada:
  - Cada nodo apunta al nodo subsiguiente y al anterior.
  - El primer nodo apunta a NULL, cuando nos referimos a su anterior, y el último nodo apunta a NULL, cuando nos referimos al subsiguiente.
- Lista enlazada simple circular:
  - El último nodo apunta al primero de la lista.
- Lista doblemente enlazada circular:
  - El último nodo tiene como subsiguiente, el primero y este tiene como anterior el último.

Introducción a listas no contiguas

Operaciones con listas enlazadas:

- **Obtener un elemento:** se van recorriendo todos los nodos hasta hallar aquel que estamos buscando.
  - Se requiere pasar por todos los nodos anteriores a él.
  - Una vez obtenido podemos realizar operaciones sobre el mismo.

Introducción a listas no contiguas

Operaciones con listas enlazadas:

- **Insertar un elemento:**
  - Creamos el nodo que contiene el nuevo elemento.
  - Obtenemos el nodo en la posición i.
  - Sustituimos la referencia siguiente del nodo i-1 por una hacia el nuevo elemento.
  - Añadimos la referencia del nodo i al nuevo nodo.
    - Si la lista es doblemente enlazada tendríamos que sustituir también las referencias de los nodos anteriores.
    - Pueden existir casos especiales como insertar al principio o al final de la lista o que la misma está vacía.

Introducción a listas no contiguas

Operaciones con listas enlazadas:

- **Borrar un elemento:**
  - Se obtiene el nodo i a eliminar.
  - Hacemos que el campo siguiente del nodo i-1 apunte al nodo i+1
  - Eliminamos el nodo i
    - Si la lista es doblemente enlazada tendremos que cambiar la referencia de los nodos anteriores.
    - Pueden existir casos especiales como borrar el primer elemento o el último.

Introducción a listas no contiguas

Ventajas de las listas enlazadas:

- No es necesario disponer de grandes bloques de memoria contigua.
  - Muy significativo con listas de gran tamaño.
- Insertar y eliminar son eficientes.
- La capacidad no es un problema.

Introducción a listas no contiguas

Ventajas de las listas enlazadas:

- No es necesario disponer de grandes bloques de memoria contigua.
  - Muy significativo con listas de gran tamaño.
- Insertar y eliminar son eficientes.
- La capacidad no es un problema.

Inconvenientes:

- Búsqueda de un elemento menos eficiente que en las listas contiguas.
  - Se requiere pasar por los nodos anteriores.
- Ocupan más memoria que las listas contiguas ya que se ha de crear y almacenar nodos.

Introducción a listas no contiguas

En java podemos la clase **LinkedList<E>** donde E es el tipo de elemento que deseamos almacenar en la lista.

- Implementa la interfaz **List<E>** por lo que ya implementa las funcionalidades básicas de las listas, además de otras propias.
- Implementa la interfaz **Iterable<E>** lo que nos permite usar el bucle “foreach”.

Documentación

Comparativa

¿Cuando usar listas contiguas?

- Cuando tengamos muchos más accesos que inserciones o eliminaciones.
- Cuando queramos ahorrar memoria.
  - Pues la lista ocupa justo lo necesario para sus elementos.
- Cuando tengamos grandes porciones de memoria contiguas.
  - La necesitamos para guardar la lista.
- Cuando el tamaño de la lista no sea muy grande.
  - Será probable encontrar zonas de memoria contiguas.
- Cuando necesitemos ordenar o buscar con eficiencia.

Comparativa

¿Cuando usar listas no contiguas?

- Cuando haya muchas más inserciones/eliminaciones que accesos
  - Las inserciones/eliminaciones ahora son más eficientes que en las listas contiguas.
- No requieren memoria adicional.
- Cuando no nos importe desperdiciar memoria.
  - Pues la lista ocupa más, al tener que guardar los punteros para apuntar al nodo anterior y siguiente.
- Desde otro punto de vista, insertar/eliminar no requieren memoria adicional, por lo cual funcionan mejor en entornos con poca memoria.
- Cuando no tengamos grandes porciones de memoria contiguas.
  - Cada nodo se puede guardar en cualquier lugar de la memoria, no necesariamente contiguo al anterior.
- Insertar/eliminar no requieren memoria adicional contigua (ni no contigua).
- Cuando el tamaño de la lista sea muy grande.
  - Al ser muy grande, es imposible encontrar grandes trozos de memoria contigua, y por lo tanto no nos queda más remedio que usar una enlazada
- Cuando no necesitemos ordenar o buscar con mucha eficiencia.

Introducción a pilas y colas

Una pila es una lista en la cual:

- Cuando añadimos un elemento (**push**), este se introduce por uno de los extremos de la lista (ej: el final).
- Cuando obtenemos un elemento (**pop**) este se obtiene del mismo extremo por donde añadimos los elementos.
  - Este extremo se denomina la **“cima”** de la pila.
- Se sigue una estrategia LIFO (Last Input First Out).

Introducción a pilas y colas

Una cola es una lista en la cual:

- Cuando añadimos un elemento (**push**), este se introduce por uno de los extremos de la lista (ej: el final).
- Cuando obtenemos un elemento (**pop**) este se obtiene del extremo **opuesto** por donde añadimos los elementos.
- Se sigue una estrategia FIFO (First Input First Out).

Introducción a pilas y colas

Tipos de cola:

- Colas simples:
  - Los elementos se añaden al final de la cola.
  - Los elementos se extraen del principio de la cola.
- Colas circulares:
  - El último elemento conecta con el primero.
- Colas de prioridad:
  - Más parecida a un tipo especial de lista ordenada.
  - El primer elemento en ser extraído es el que cuente con mayor prioridad (Si hay varios con la misma prioridad el primero que llegó)

Introducción a pilas y colas

Al final las pilas y colas se pueden implementar a partir de cualquier tipo de lista que deseemos.

Introducción a pilas y colas

En java se nos proporciona la interfaz **Queue<E>** que implementan diversas clases de lista para que podamos usarlas como una cola.

Documentación

Para la pila java implementa la clase **Stack<E>** que incorpora las funcionalidades básicas de una pila.

Documentación

Sin embargo, la propia documentación nos advierte que esta última clase no es consistente, ya que hereda de **Vector<E>** y nos invita a usar la interfaz **Deque<E>.**

Documentación