6. Listas, colas y pilas

Listas

En una lista es una secuencia ordenada de elementos. Se dice que es ordenada, no porque sea relevante el orden, sino porque cada elemento que forma parte de la lista tiene una posicion en la misma.

Se puede decir que una lista es homogenea cuando todos sus elementos son del mismo tipo. C++, por defecto, solamente permite hacer listas homogeneas.

En una lista se pueden insertar y eliminar elementos en una posicion. Una lista con N elementos tiene N+1 posiciones. Esto no es muy informatico, por asi decirlo, no tiene tanto que ver con como se indexa. Esto se entiende como que si tengo tres elementos, puedo insertar uno nuevo en 4 lugares (al comienzo, entre el 1 y 2, entre el 2 y 3, o, al final).

La primera posicion se conoce como **front**, la ultima posicion se conoce como **back**. En STL, las posiciones son representadas por iteradores. El metodo `begin()` refiere a la primera posicion, es decir, al espacio antes del primer elemento. El metodo `end()` hace referencia a la ultima posicion. De nuevo, esto no es el ultimo elemento, sino que es la posicion despues del ultimo elemento. Estas no son posiciones de memoria. Lo que se hace, en el fondo, es usar un puntero, pero en el caso de `end()` no debo intentar acceder a la posicion porque me estaria yendo del arrego. En distintos entornos puede estar implementado de maneras distintas (con o sin punteros). Es importante remarcar que, en el caso de `end()` queda totalmente a criterio del programador decidir lo que quiere hacer con la posicion que hace referencia a la direccion de memoria siguiente al ultimo elemento de la lista. Se puede hacer alusion a la misma pero puede no ser lo mas correcto, desreferenciarla.

Listas con arreglo

Una lista con arreglo es una lista implementada con un arreglo
Las listas implementadas con arreglos se caracterizan por ser un bloque **contiguo** de memoria. En STL, pueden ser `vector` y `array`. La diferencia entre estos es que `vector` es de longitud variable, mientras que `array` no lo es. `vector` siempre guarda un poco mas de memoria de la que se pidio, para no tener que hacer un `resize()` cada vez que se quiera insertar un elemento. *See* `vector.reserve()` *and* `vector.size()`.

- La complejidad computacional de crear una lista con arreglo vacio es $O(1)$.
- La complejidad de crear una lista con N elementos es $O(n)$. El `malloc()` tiene complejidad $O(1)$ pero luego hay que llamar al constructor de cada objeto, por lo que se vuelve $O(n)$.

- La insercion y eliminacion en en la ultima posicion es $O(1)$, y en una posicion cualquiera es $O(n)$.

De mas esta decir que las listas con arreglos ocupan en la memoria `nro de elementos * tamano de un elemento`.

Deque

Double-ended queue
Es similar a una lista con arreglo con la ventaja de poseer insercion y eliminacion rapida en `front`. Es mas lenta en el acceso a sus elementos. Suelen usarse con una lista de arreglo de punteros a listas con arreglo. Estas estructuras tambien son contiguas en memoria.

> *Se prefieren las listas con arreglo (incluso deques) cuando necesitamos acceso aleatorio rapido.*

Listas simplemente enlazadas

Es una secuencia de nodos (de algun tipo de dato) no necesariamente contigua. Cada nodo contiene un puntero al siguiente nodo hasta el ultimo, que contiene un puntero a `NULL`. Las listas simplemente enlazadas permiten insertar y remover de manera muy eficiente. Sin embargo, no son la mejor opcion para acceso aleatorio.

En STL, las listas simplemente enlazadas estan implementadas en `forward_list`.

`forward_list` suele utilizar un *nodo centinela* o nodo *header*. Este se ubica en la primera posicion y su contenido se ignora (salvo el puntero al siguiente elemento). Tiene el fin de simplificar la logica de la lista ya que me garantiza un nodo al cual apuntar para acceder a la lista. *Nota: esto a Marc no le gusta.*

Listas doblemente enlazadas

Estas son como las anteriores pero con dos punteros. Uno de los cuales apunta al siguiente elemento (`next`) y otro al elemento anterior (`prev`). Esto otorga la ventaja de poder recorrer la estructura tanto para adelante como para atras.

Estas listas ademas del nodo header, tienen un nodo trailer. Cumple la misma funcion que en nodo header de las listas simplemente enlazadas.

Las listas enlazadas son $O(1)$ en la insercion pero son $O(n)$ en el acceso por indice.
Una lista con N elementos tienen N punteros (recordar el overhead de los nodos centinela)

> *Se prefieren las listas enlazadas para situaciones que requieren realizar muchas operaciones de insercion y eliminacion*

Manejo de elementos

Una idea interesanta para manejar listas de cosas es usar una lista de punteros a elementos. De esta manera si quiero repetir un mismo elemento, puedo hacerlo apuntando al mismo lugar. Ademas, cuando borre la lista, los elementos siguen existiendo, se estan borrando los punteros.

Listas heterogeneas

En C++ las listas son homoegeneas porque se crean usando template. Para crear listas heterogeneas se usa herencia con metodos virtuales. La lista consiste de una serie de punteros a la clase base. Luego, se pueden insertar objetos que deriven de la clase base.

Colas

Una cola es una lista en la que los elementos se encolan por una punta (normalmente designada como `back`) y se desencolan por la otra (generalmente nombrada `front`). Las colas son FIFO. Suelen usarse para encolar tareas.

En STL las colas se implementan en el adaptador `queue`. Un adaptador es un patron de diseno de softawre que permite transformar la interfaz de una clase en otra. Es, en otras palabras, un wrapper. En el caso de `queue` se esta haciendo un wrapper de `deque` por defecto, o de `list`. Esto presenta una nueva interfaz. Habra que elegir de que manera es mejor implementar la cola. Un `deque` va a tener la ventaja de usar menos memoria (por no tener punteros) mientras que `list` va a tener la ventaja de ser un poquito mas rapido para insertar y eliminar elementos.

Colas de prioridad

Es una cola que se ordena segun la prioridad de sus elementos. Los elementos de mayor prioridad quedan al comienzo de la lista y los de menor prioridad al final. Esto tambien es un wrapper, en este caso de `vector` o `deque`.

Pilas

Una pila es una lista en la que los datos se apilan por una punta (generalmente `back`) y se desapilan por la misma punta. Las pilas son LIFO. Se suelen usar para almacenar tareas parcialmente completadas, evaluar expresoines aritmeticas e implementar la funcionalidad undo/redo. Son un stack.