Estructura de Datos y Algoritmos
Departamento de electrónica
Universidad Técnico Federico Santa María

Ayudantía 4: Listas simplemente enlazadas y doblemente enlazadas.

Objetivo Principal: Conocer cómo funcionan e implementar las listas simplemente enlazadas y doblemente enlazadas.
Autor: Kevin Pizarro Aguirre

---

1. Lista Simplemente Enlazada

Una lista simplemente enlazada es una secuencia de estructuras, las cuales están conectadas por enlaces (punteros) en una dirección. Cada una de las estructuras se les llama nodo, el primero de todos corresponde a la cabeza mientras que el último es la cola. Además, cada nodo contiene dos partes fundamentales: los datos "útiles" y el puntero al siguiente nodo, el último nodo debe apuntar a NULL tal como se ve en la figura 1.

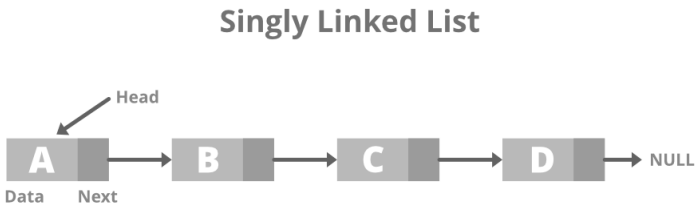

Figura 1: Esquema para lista simplemente enlazada.

Esta forma de estructurar los datos presenta ventajas y desventajas. Entre las ventajas se destaca que puede modificar su tamaño; además la inserción, eliminación de nodos/datos e implementación de la estructura es relativamente simple. Por contraparte se debe usar espacio para poder almacenar el puntero al nodo siguiente y que el acceso a un elemento específico no es directo, ya que se debe recorrer desde la cabeza a la cola hasta encontrar el nodo de interés.

Entre las operaciones más útiles en las listas simplemente enlazadas podemos encontrar: *push* (inserción), *eliminación* y *búsqueda* por valor. Todas las implementaciones pueden ser observadas en el código del archivo *7-ListaSimple.c*, el cual puede ser encontrado en el repositorio.

Además se podrían implementar más funciones según la necesidad, como por ejemplo determinar el largo de la lista, ordenar los datos ascendentemente o descendentemente, eliminar los valores duplicados, dividir una lista en dos, unir listas, etc. Aún así, se debe tener en cuenta la idea principal de la lista simplemente enlazada, para conocer sus limitaciones y ventajas respecto a otras.

2. Lista Doblemente Enlazada

Una lista doblemente enlazada es una secuencia de estructuras, las cuales están conectadas por enlaces (punteros) en ambas direcciones. Cada nodo contiene tres partes fundamentales: los datos "útiles", el puntero al siguiente nodo y el puntero al nodo previo. Tanto en el puntero al nodo previo a la cabeza como al nodo siguiente de la cola estará apuntando a NULL, tal como se puede ver en la figura 2.

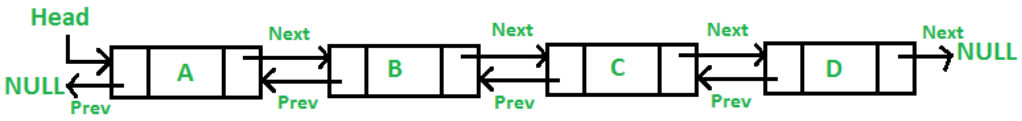

Figura 2: Esquema para lista doblemente enlazada.

La diferencia respecto a la lista simplemente enlazada es que ahora podremos movernos en ambos sentidos, avanzar al nodo siguiente y retroceder al nodo previo. Por ello mismo ahora se necesitará más espacio en la memoria para poder almacenar el puntero al nodo previo. Como ventaja son posibles más formas de añadir nodos, eliminar y otras operaciones útiles en la estructura.

Entre las operaciones más útiles en las listas doblemente enlazadas podemos encontrar: *push* (inserción), *eliminación*, *búsqueda* por valor y *reverse* (invertir la lista) . Algunas implementaciones pueden ser observadas en el código del archivo *8-ListaDoble.c*, el cual puede ser encontrado en el repositorio.

En ocasiones las listas doblemente enlazadas pueden ser usadas para implementar otras estructuras. Cabe destacar que también puede ser utilizado para implementar el sistema de un navegador para poder movernos hacia adelante y atrás, en algunos sistemas operativos para el manejo de hilos, registros con MRU (Most Recently Used) o LRU (Last Recently Used) page replacement, etc.

3. Ejercicios

3.1. Lista simplemente enlazada. Squid Game.

1. Introducción a listas simplemente enlazadas.

   a) Cree una estructura llamada Player cuyos miembros sean: name, id_player, debt y next. Los miembros name deben ser de tipo char[50]; id_player y debt del tipo int; el miembro next debe ser de tipo (struct Player *).

   b) Defina una función con prototipo: Player * new_player (char * name, int id_player, int debt). Ayúdese de la función strcpy para copiar los strings entregados a su función. Simplemente crea el jugador, aún no es añadido a la lista.

2. Insertando nodos/jugadores al comienzo de la lista.

   a) Defina una función con prototipo: void push (Player ** list, Player * player). Esta función debe agregar el jugador player al comienzo de la lista.

3. Lectura de la lista de jugadores.

   a) Defina una función con prototipo: Player * get (product * list, int X). Esta función debe retornar el nodo en la posición X de la lista.

   b) Defina una función con prototipo: void print_list(Player * list). Esta función debe imprimir todos los jugadores que están en la lista, con su información pertinente.

4. Eliminar jugador y limpieza de memoria.

   a) Defina una función con prototipo: void delete (Player ** list, int X). Esta función debe eliminar el elemento X de la lista list. Use la función free y asegurese de que no hay fugas de memoria con Valgrind.

   b) Antes de terminar el programa se deben eliminar todos los elementos en la memoria HEAP.

5. Interacción con administrador.

   a) Permita que todas las funciones implementadas anteriormente puedan ser utilizadas por un usuario, el cual añade jugadores, elimina y chequea la información a través de la consola.

3.2. Lista Doblemente enlazada

1. Introducción a listas doblemente enlazadas.

   a) Se desea modelar la fila frente a la caja de un banco. Cree una estructura llamada persona que represente a una persona en la fila. La estructura debe permitir construir una lista doblemente enlazada, con los atributos struct persona *prev, *next, su edad y la cantidad de dinero que traen.

   b) En su función main cree un puntero a la primera persona de la fila, esto es la persona que está más cerca de la caja. ¿Hacia donde apuntará este puntero si aún no hemos creado personas?

   c) Escriba una función que le permita crear personas. Las personas deben ser creadas con edad aleatoria entre 18 y 200 años y cantidad de dinero aleatoria entre 0 y 232 unidades monetarias. ¿De qué tipo tiene que ser la variable usada para almacenar la cantidad de dinero?

2. Inserción al final de la lista

   a) Escriba una función void ins_persona (persona **final, persona *nuevo elemento) que le permita insertar personas al final de la fila.

   b) En su función main cree un un bucle while que se ejecute una vez cada 10 segundos.

   c) Al comenzar el bucle while debe eliminarse a la primera persona de la fila. Esto simulará al usuario que acaba de ser atendido y sale de la fila. La memoria que ocupa este usuario debe ser liberada.

   d) Después de quitar al primer usuario deben agregarse aleatoriamente entre 0 y 3 nuevos usuarios al final de la fila.

3. Cantidad de elementos/personas en una lista.

   a) Para cada iteración del bucle while imprima por pantalla todos los usuarios que se encuentran en la fila indicando su edad y cantidad de dinero. Además imprima la cantidad total de personas esperando en la fila.

4. Búsqueda en una lista

   a) Para cada iteración del bucle while busque a todas las personas mayores de 70 años y permita que pasen al comienzo de la fila. No es necesario que las personas que reasigne al principio de la fila sigan un orden determinado. Esta búsqueda debe ser realizada desde la persona que está más cerca de la caja a la que está más lejos.

5. Inserción en una lista

   a) Suponga que su fila tampoco está libre de gente que quiera “colarse”. Para simular este comportamiento, cree una función void insertar(persona ** fila, persona * interpuesta, int posicion). Tenga precaución con los casos en que fila o interpuesta sean nulos, y la posición (contada desde el principio de la fila) exceda los límites de esta fila.

   b) Incluya esta función en el bucle while que hizo en la parte 2. La probabilidad de que una persona se “cuele” en la fila debe ser del 25 %, aleatoriamente en cualquier posición de la fila.