TDA 1 - Lista

[7541/9515] Algoritmos y Programación II

Segundo Cuatrimestre - 2021

Alumno:	**González Calderón, Julián**
Padrón:	107938
Email:	gonzalezcalderonjulian@gmail.com

1. Introducción

La idea general del trabajo practico era implementar el TDA lista utilizando nodos simplemente enlazados. Además, a partir de las funciones definidas para la lista, debíamos definir también la pila y la cola.

Luego de haber definido la lista, debíamos implementar un nuevo TDA, el iterador externo. Se crea a partir de una lista y su trabajo es acceder a cada uno de los elementos de la lista de forma iterativa (de ahi su nombre).

Para hacer esto nosotros contábamos con un ***contrato*** que debíamos cumplir, este especificaba la funcionalidad pedida.

Además de implementar correctamente los TDA pedidos, teníamos que escribir nuestras propias pruebas a medida que desarrollábamos el código. A pesar de que esto pareció trabajoso al principio, rápidamente resulto muy útil a la hora de *debuggear* el código.

2. Teoría

- **¿Que es una Lista?**

  Una lista es una estructura de datos que consiste en un secuencia ordenada de datos

  La utilidad de la lista es poder insertar, eliminar, y acceder información de cualquier posicion de la misma.

  Además de la estructura mencionada, la lista cuenta con diversas funciones que nos permiten operar con ellas:

  - Crear lista.
  - Destruir lista.
  - Insertar elemento en cualquier posición.
  - Quitar elemento de cualquier posición.

- Acceder a un elemento en cualquier posición.
- Preguntar si es una lista vacía.
- Preguntar la cantidad de elementos.
- Aplicar a cada elemento de la lista una función determinada, con los parámetros necesarios.

- **¿Como se implementa una lista?**

  Hay tres formas de implementar una lista

  - **Vector Estático:** Es útil cuando sabemos de antemano el numero de elementos que vamos a necesitar, no es flexible.
  - **Vector Dinámico:** Es útil cuando no sabemos de antemano el numero de elementos que vamos a necesitar, se puede expandir.
  - **Nodos:** Consiste en el uso de nodos, donde cada nodo contiene la dirección a otros nodos. De esta forma no necesitamos reservar memoria contigua.
    - **Simplemente Enlazada:** Cada nodo contiene la dirección del nodo siguiente.
    - **Doblemente Enlazada:** Cada nodo contiene la dirección del nodo siguiente y el anterior.
    - **Lista Circular:** El nodo final guarda la referencia al primer nodo.

  La complejidad computacional de la lista es mencionada en los detalles de implementación.

- **¿Que es una Pila?**

  La pila es similar a la lista, pero cuenta con algunas restricciones. *(FIFO: First in, First out)*

  - Solo se pueden insertar elementos al final de la pila. (*apilar*)
  - Solo se pueden quitar elementos del final de la pila. (*desapilar*)

- **¿Como se implementa una pila?**

  Las implementaciones de la pila son parecidas a la lista

  - **Vector Estático:** Es útil cuando sabemos de antemano el numero de elemento que vamos a necesitar, no es flexible
  - **Vector Dinámico:** Es útil cuando no sabemos de antemano el numero de elementos que vamos a necesitar, se puede expandir
  - **Nodos Simplemente Enlazados:** Consiste en almacenar la información del nodo tope, y que cada nodo contenga la dirección del nodo siguiente. La ventaja de usar nodos es que no necesitamos memoria contigua.

- **¿Que es una Cola?**

  La cola es similar a la lista, pero cuenta con algunas restricciones. *(FILO: First in, Last out)*

  - Solo se pueden insertar elementos al final de la pila. (*encolar*)
  - Solo se pueden quitar elementos del inicio de la pila. (*desencolar*)

  La cola es ligeramente mas difícil de implementar que la pila (en el caso de usar un vector), esto se debe a que eliminamos elementos del principio y colocamos al final, por lo que eventualmente nos quedaremos sin espacio. Para solucionar esto tenemos dos opciones:

  - Desplazar los elementos a medida que agregamos nuevos.
  - Usar una cola circular, posicionando el tope en el elemento anterior al primero.

  Debido a estos inconvenientes, es mejor utilizar la implementación con nodos simplemente enlazados, nuevamente no nos sirve utilizar nodos doblemente enlazados (ya que no queremos recorrer la cola).

- **¿Como se implementa una cola?**

  Las implementaciones de la cola son parecidas a la lista

  - **Vector Estático:** Es útil cuando sabemos de antemano el numero de elemento que vamos a necesitar, no es flexible
  - **Vector Dinámico:** Es útil cuando no sabemos de antemano el numero de elementos que vamos a necesitar, se puede expandir

  Estas implementaciones tienen un inconveniente, como insertamos del final y agregamos del comienzo nos vamos a quedar sin espacio eventualmente. Para solucionar esto tenemos dos opciones:

  - Desplazar los elementos a medida que agregamos nuevos.
  - Usar una cola circular, posicionando el tope en el elemento anterior al primero.

  Debido a estos inconvenientes, es mejor utilizar la implementación con **nodos simplemente enlazados.**

3. Detalles de implementación

Algunas funciones importantes merecen ser explicadas de forma detalla.

1. `lista_insertar()`

   Esta función reserva memoria para el nuevo nodo a introducir, conecta el nodo final con este, y asigna el nodo final de la pila al nuevo nodo.

   La complejidad de esta función es $O(1)$, ya que solo debemos modificar el ultimo elemento.

2. `lista_insertar_en_posicion()`

   Esta función es similar a la anterior. Reserva memoria para el nuevo nodo a introducir, busca el nodo *original* de la posición buscada y el *anterior*. A

continuación conecta el nodo *a insertar* con el *original*, y el nodo *anterior* con el nuevo.

Si se desea insertar en la primera posición, entonces simplemente reserva memoria para el nuevo nodo, apunta este nodo al nodo inicial, y asigna el nodo inicial de la lista al nuevo nodo.

La complejidad de esta función es $O(n)$, ya que debemos recorrer la lista para encontrar la posición en la que queremos insertar.

3. `lista_quitar()`

   Esta función busca el nodo que esta en la posición anterior a la final. Luego libera el ultimo nodo, apunta el nodo *anterior* a *NULL* y asigna el nodo final de la lista a este mismo nodo (*anterior*).

   La complejidad de esta función es $O(n)$, esto se debe a que al no usar nodos doblemente enlazados, para acceder a la anteultima posición debemos recorrer la lista.

4. `lista_quitar_de_posicion()`

   Esta función es similar a la anterior. Busca los nodos que están en la posición anterior y siguiente al nodo a eliminar, luego (guardando referencia del nodo a *eliminar*) apunta el nodo *anterior* al nodo *siguiente*. Finalmente libera el nodo a *eliminar*.

   Si se desea quitar un elemento de la primera posición, entonces se guarda referencia al nodo a *eliminar*, luego se apunta el nodo inicial de la lista al nodo siguiente al primero, y se libera el nodo a *eliminar*.

   La complejidad de esta función es $O(n)$, ya que debemos recorrer la lista para encontrar la posición en la que queremos eliminar.

5. `lista_elemento_en_posicion()`

   Esta función simplemente recorre la lista, empezando por el primer elemento y accediendo a el siguiente elemento hasta llegar a la posición buscada.

   La complejidad de esta función es $O(n)$, ya que debemos recorrer la lista para encontrar la posición a la que queremos acceder.

6. `lista_destruir()`

   Esta función destruye el nodo inicial de la lista, y asigna el nuevo nodo inicial al siguiente. Luego vuelve a llamar a la misma función con la lista actualizada, si el nodo inicial de la lista es *NULL* quiere decir que ya libero todos los elementos, por lo que debe libera la lista en si.

   La complejidad de esta función es $O(n)$, ya que iteramos la lista.

7. `lista_con_cada_elemento()`

   Esta función itera cada elemento de la lista, partiendo del primero y accediendo a siguiente hasta que este esa *NULL, a*plica una función a cada elemento. Si esta función devuelve false, entonces corta el ciclo.

   La complejidad de esta función es $O(n)$, ya que iteramos la lista.

Las funciones restantes, tienen complejidad computacional $O(1)$

- `lista_primero`
- `lista_ultimo`
- `lista_vacia`
- `lista_tamanio`
- `lista_iterador_crear`
- `lista_iterador_avanzar`
- `lista_iterador_elemento_actual`
- `lista_iterador_destruir`

3.1 Pila y Cola

Las funciones de la implementación de pila y cola con nodos simplemente enlazados tienen complejidad computacional $O(1)$ en todas las funciones, excepto en la función `destruir()`

Para cumplir con esto se utilizan las funciones de la lista normalmente, excepto en el caso de desapilar y apilar de una pila.

Para implementar estas funciones con complejidad computacional $O(1)$, entonces invertí la pila. Los elementos se insertan y se quitan de la primera posición, de esta forma, la funcionalidad es la misma pero no debe recorrer el vector para llegar al final.

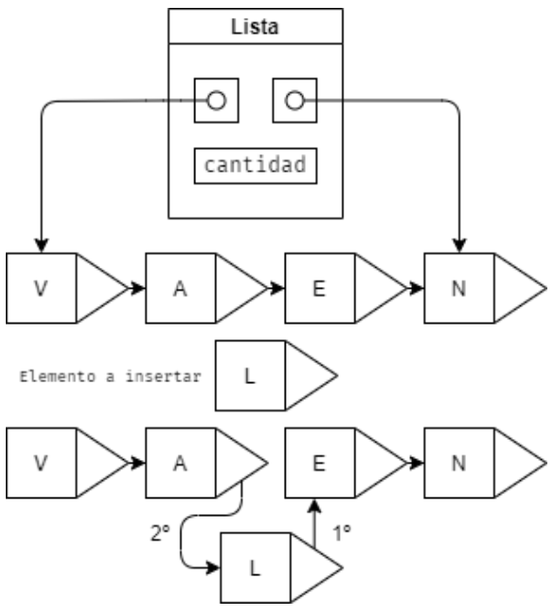
4. Diagramas

Algunos diagramas para profundizar la explicación los algoritmos utilizados.

Insertar Elemento

En este diagrama se puede ver el algoritmo de inserción de elementos en una lista:

1. Buscamos los nodos que están en la posición anterior, y en la posición a insertar.
2. Conectamos el nodo a insertar con el nodo siguiente.
3. Conectamos el nodo anterior con el nodo a insertar.

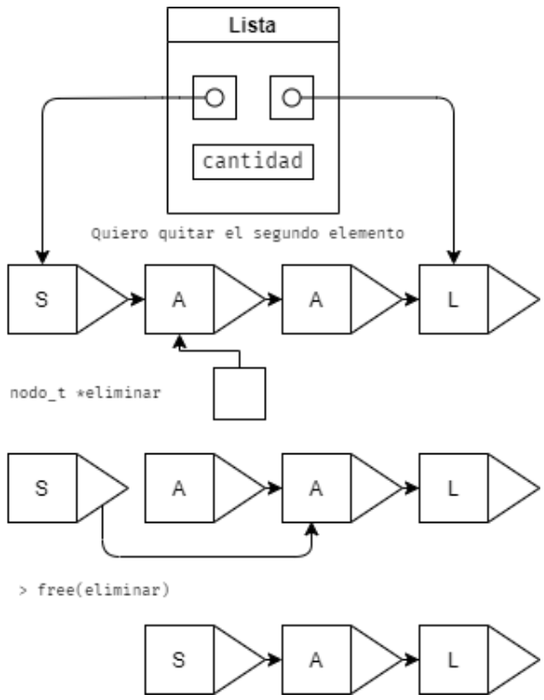


Eliminar Elemento

En este diagrama se puede ver el algoritmo de eliminación de elementos de una lista

1. Buscamos los nodos que están en la posición anterior, y en la siguiente al nodo a eliminar.

2. Guardamos la referencia del nodo que debemos eliminar.
3. Conectamos el nodo anterior con el nodo siguiente.
4. Liberamos el nodo a eliminar.



Buscar Posición

En este diagrama se puede ver el algoritmo de buscar un elemento de la lista

1. Partimos del nodo inicial
2. Accedemos al nodo siguiente n veces (si queremos la posición 0, entonces no avanzamos)
3. Devolvemos el elemento del nodo actual