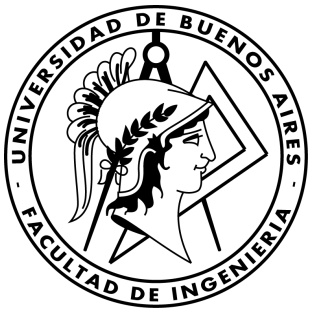

Trabajo Práctico — TDA Lista

[7541/9515] Algoritmos y Programación II
Primer cuatrimestre de 2022

Alumno:	Vainstein Aranguren, Tomás
Número de padrón:	109043
Email:	tvainstein@fi.uba.ar

Índice

1. Introducción

Para este TP se pidió armar el código de un TDA Lista que permita el ingreso y borrado de datos de tipo no determinado. Usando esta misma implementación de lista, se crearon funciones para el uso de TDA Pila y TDA Cola, para que el usuario pueda manipular la información dentro de los límites de cada uno de estos modelos abstractos. Aparte, se crearon pruebas para facilitar el desarrollo del código utilizando la biblioteca pa2m.h.

2. Teoría

1. Explique el TDA Pila y cómo se implementa utilizando nodos enlazados:

   Un TDA Pila es una colección de elementos ordenados de la cual se pueden agregar o quitar datos interactuando siempre con su posición final denominada tope. Para implementar una pila utilizando nodos enlazados nada más se utiliza el puntero que apunta a nodo_fin (tope) ya que es la única forma de acceder a una pila. Dado esto, en el caso de que se quiere insertar un elemento a la pila (apilar), se utilizará un nodo que contenga un puntero al elemento, el cual será apuntado por el campo "siguiente" del que previamente era el nodo_fin, siendo ahora el nodo con el elemento agregado el nuevo tope. Este procedimiento se repite cada vez que se quiera agregar un nuevo elemento. En el caso de querer desapilar un elemento, se encuentra la posición del tope y se libera la memoria del nodo, siendo el anterior a este ahora el nuevo tope apuntando a NULL.

2. Explique el TDA Cola y cómo se implementa utilizando vectores estáticos (cola circular):

   Un TDA Cola con vectores estáticos (cola circular) es una versión mejorada del TDA Cola simple. Al igual que la cola simple, la cola circular las operaciones de encolar y desencolar se dan por el final de la cola y el inicio respectivamente, por lo cual se pueden reutilizar estas funciones en el caso de que la cola se encuentre completamente llena o completamente vacía. La cola simple y circular tienen como diferencia que, al estar la cola circular implementada por vectores estáticos, en el caso de que la cola esté parcialmente llena, se debe verificar que la cantidad de memoria en el vector estático permite la inserción del nuevo elemento. Al desencolar un elemento en cola circular, la memoria liberada se puede reutilizar para insertar un nuevo elemento, haciendo esta versión más eficiente que la cola simple.

3. Explique el TDA Lista y cómo se implementa utilizando nodos enlazados:

   Un TDA Lista es una colección de elementos ordenados que, a diferencia de pila y cola, puede ser recorrido para insertar y quitar un elemento al principio, en medio o al final de la misma. Para agregar un elemento a una lista con nodos enlazados se reserva memoria para alocar un nuevo nodo que será insertado al principio, en medio (indicando la posición) o al final. En el caso de que la lista esté vacía, los nodos inicio y fin apuntarán a este nodo recientemente agregado. En el caso de que se quiera quitar un elemento, se apuntará un nodo auxiliar (sin reservar memoria) que ubicará el nodo que se quiere borrar para evitar perder la dirección de los nodos detrás o delante del mismo. En el caso de que haya un único elemento a quitar, los nodos inicio y fin apuntarán a dirección NULL una vez que este se haya borrado exitosamente.

3. Detalles de implementación

Para comenzar a implementar el TP primero comencé a realizar todas las funciones de TDA Lista que luego servirían para implementar el TDA Pila y TDA Cola. La primera función que implementé fue la de crear_lista, la cual inicializa los campos de un struct de tipo lista_t en 0 y

NULL respectivamente. En el caso de que no sea posible la creación de la lista, la función devuelve NULL. Luego se implementaron las funciones de insertar, en el cual se tomaron en cuenta los casos en los que se podría encontrar la lista (vacía, con uno o con varios elemento) para la función de insertar al final o insertar en una posición (en el cual se considera el caso de que se ingrese un elemento en la primera posición si la posición recibida es 0). Lo mismo hice para las funciones de quitar, en donde se contemplaron los casos de una lista vacía, parcialmente llena o completamente llena en las funciones de quitar (la cual borra en la última posición) y quitar en posición. Para implementar las funciones de "lista elemento en posición" y "lista buscar elemento" utilicé un razonamiento similar basado en inicializar un nodo auxiliar en nodo_inicio que vaya siendo asignado a su posición siguiente con cada iteración, siendo la condición de corte en la primera función nombrada que se encuentre en la posición recibida por parámetro y en la segunda que se llegué hasta el último nodo de la lista, o bien, que se cumpla la condición de la función "comparador" para que la función de búsqueda tenga un resultado exitoso. En el caso de las funciones "lista_primero", "lista_ultimo" y "lista_tamanio" se utilizaron los valores en los cuales se encontraba la lista en el momento de utilizar la función. Para las funciones de "destruir" y "destruir todo" se utilizó una función que verifica que si la función recibida por parámetro era equivalente a NULL, entonces no se aplica esta función destructora a los elementos de la lista, por lo que en la función de destruir, se hace un llamado a la función de "lisa_destruir_todo" con segundo parámetro NULL. En las funciones del iterador, primero se crea reservando memoria e inicializando el campo corriente con el nodo_inicio y campo lista de este iterador con la lista recibida por parámetro. Por último, en el iterador interno (lista_con_cada_elemento) se utiliza un nodo auxiliar inicializado con el primer nodo de la lista el cual luego se utiliza para verificar si se deben seguir iterando los elementos de la lista o no. Para la implementación de pila, reutilicé las funciones de lista. Para el caso de apilar elementos implementé la función de lista_insertar ya que nada más se pueden insertar elementos al final de esta, y para desapilar la función de lista_quitar que se encarga de quitar elementos únicamente en la posición final. Lo mismo hice para el caso de cola, en la cual utilicé la función lista_insertar para encolar dado a que esta se encarga de agregar elementos únicamente en la posición final de la lista, y la función quitar_de_posicion recibiendo la posición 0 como parámetro, haciendo que nada más se puedan quitar elementos en la posición inicial.

Complejidad de cada función:

- crear: O(1)
- insertar: O(1)
- insertar en posicion: O(1) en el caso de que la posición recibida sea la primera o inexistente en la lista, O(n) si no.
- borrar: O(1)
- borrar de posicion: O(1) en el caso de que la posición recibida sea la primera o inexistente, O(n) si no.
- elemento en posicion: O(1) si la posición recibida es 0, O(n) si no.
- buscar: O(1) si hay un único elemento en la lista, O(n) si hay más.
- primero: O(1)
- ultimo: O(1)
- vacia: O(1)
- tamanio: O(1)
- destruir/destruir todo: O(1) si hay un único elemento en la lista, O(n) si hay más.

3.1. Detalles de insertar_en_posicion

La función insertar_en_posicion tiene como objetivo insertar un elemento a la lista creada. Para eso se contemplan tres casos: la posición puede ser inexistente o la última en la lista, la posición puede ser la primera en la lista, o la posición puede ser existente dentro de la lista pero ni la primera ni la última. En el caso de que la posición sea la última o inexistente, se llama a

la función lista_insertar que tiene como objetivo insertar el nodo al final de la lista. Dentro de esa función se verifica si la lista está vacía o no. En el caso de que lo esté, se asignan nodo_inicio y nodo_fin al nodo insertado. Si no sucede esto, el nodo siguiente del nodo_fin hacia el nuevo nodo insertado y luego el nodo_fin al nuevo elemento. En el caso de que la posición recibida sea la primera, simplemente se le asigna como nodo siguiente del nodo a agregar el que previamente estaba al principio de la lista y luego se posiciona el recientemente agregado como nodo_inicio. Por último, en el caso de que la posición sea una del medio, se utiliza un nodo auxiliar incializado en nodo_inicio que va avanzando por cada posición de la lista en cada iteración hasta posicionarse en la posición recibida por parámetro, donde luego se apunta el campo siguiente del nodo que será insertado al próximo nodo que se encontraba en la posición en la que ahora se va a ubicar el agregado, y el nodo que se encontraba en esa posición ahora tiene campo siguiente apuntando al nuevo nodo. Cada vez que se ingresa un nodo en la lista exitosamente, se suma un número a la cantidad de elementos en la lista.

3.2. Detalle de quitar_de_posicion

La función quitar_de_posicion tiene como objetivo borrar un elemento de la lista creada. En esta función, tal como en la de insertar, se contemplan varios casos. En el caso de que la posición a borrar sea inexistente o la última de la lista, se llama a la función lista_quitar, en la cual también se verifican dos casos más. En el caso de que haya un único elemento en la lista, se libera el último nodo y ambos nodo_inicio y nodo_fin apuntan a NULL. En el caso de que haya más de un elemento, se itera un nodo auxiliar inicializado en nodo_inicio hasta encontrar la última posición, borrando la misma y apuntando el nodo_fin al nodo que estaba en la posición anterior al borrado y el campo siguiente de este nodo a NULL. En el caso de que la posición recibida sea la primera de la lista, se ubica el nodo al principio de la misma y se apunta el nodo_inicio->siguiente al nodo que antes estaba en la segunda posición. Por último, si se recibe una posición del medio, se itera hasta encontrar esa posición y se le asigna la posición siguiente a un nodo auxiliar para no perder el enlace. Luego se hace que el campo siguiente del nodo auxiliar apunte al que estaba en la posición siguiente al que se está por eliminar y el campo siguiente del nodo anterior ahora apunte a este nodo auxiliar. Luego, se libera la memoria del nodo a quitar. Cada vez que se elimina un nodo de la lista exitosamente, se resta un número a la cantidad de elementos de la lista.

4. Diagramas

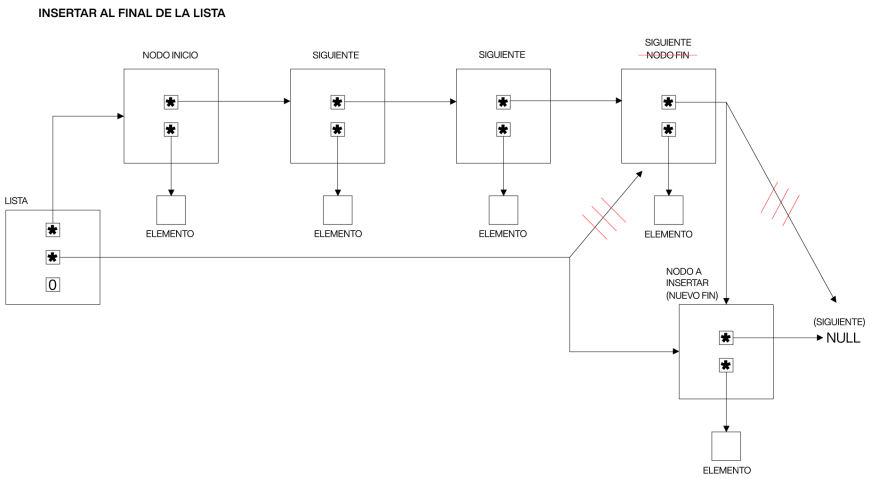

Figura 1: Insertar al final de la lista.

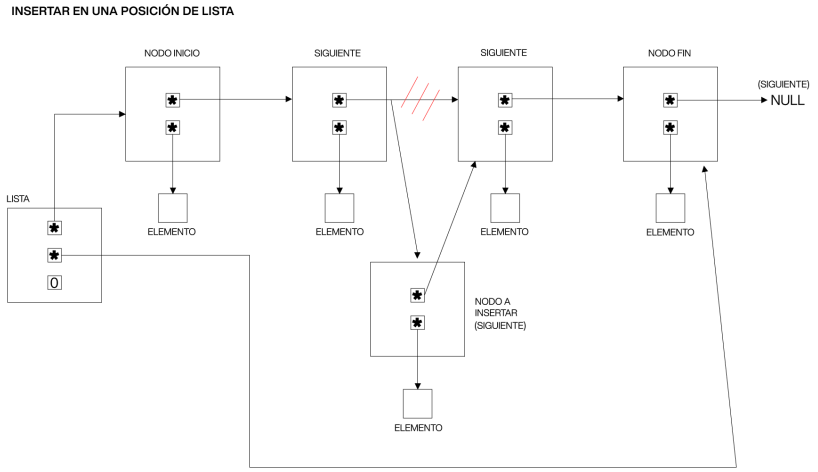

Figura 2: Insertar en posición de lista.

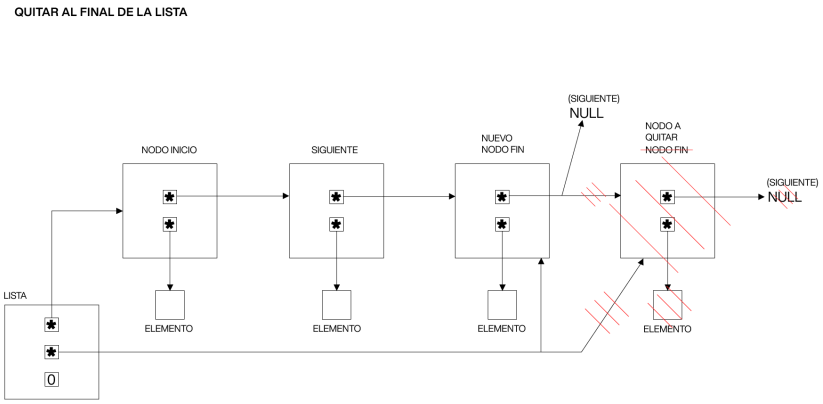

Figura 3: Quitar al final de la lista.

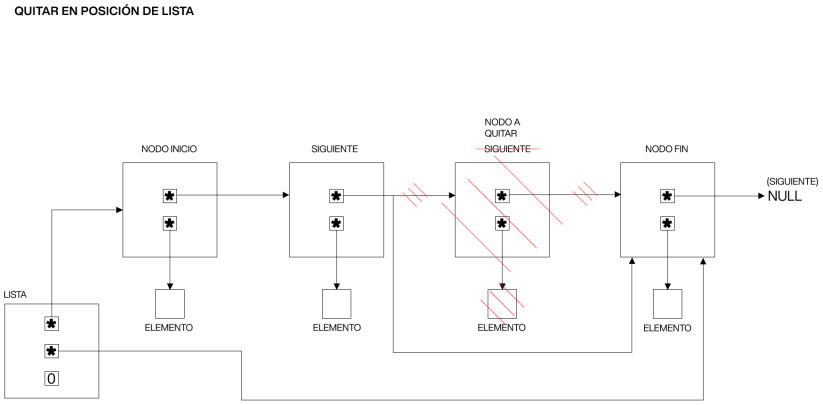

Figura 4: Quitar en posición de la lista.

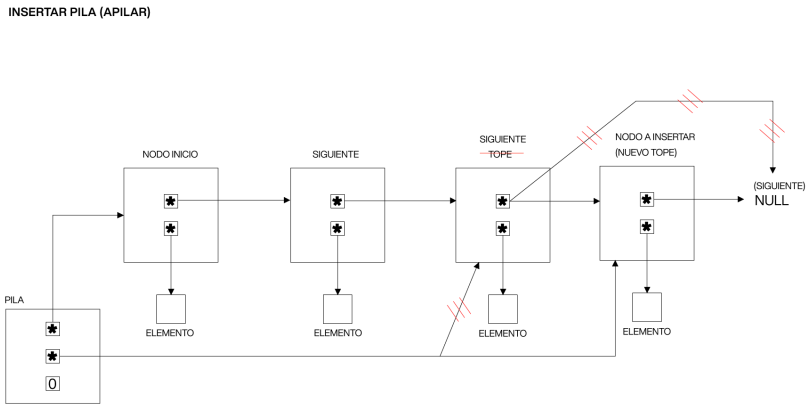

Figura 5: Apilar.

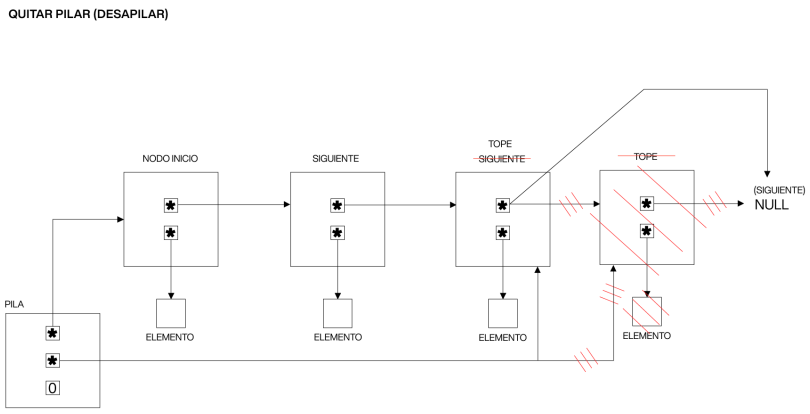

Figura 6: Desapilar.

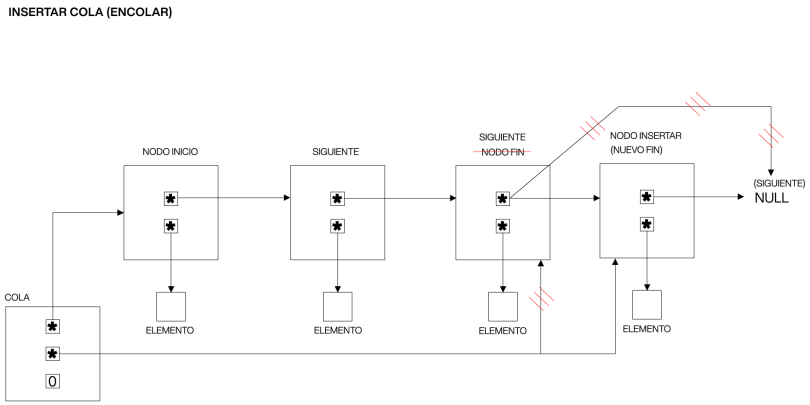

Figura 7: Encolar.

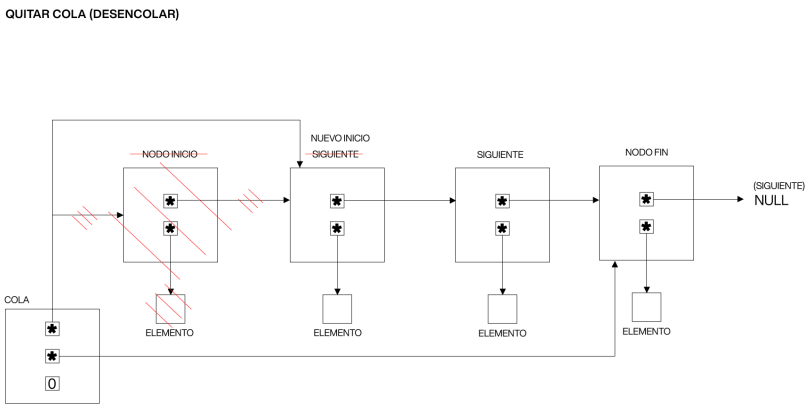

Figura 8: Desencolar.