FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

TDA LISTA

Lucas Franciulli

[7541/9515] Algoritmos y Programación II

Segundo cuatrimestre 2021

Nombre:	Franciulli, Lucas
Padrón:	107059
Email:	lfranciulli@fi.uba.ar

Índice

1. Introducción

La idea de este trabajo es poder crear un tipo de dato abstracto el cual funcione como una lista simplemente enlazada. Este también sirvió para crear otros dos tipos de TDA, llamados pila y cola. El objetivo de este informe es poder explicar que es cada uno de estos TDAs, cómo funcionan, las implementaciones que tienen y alguna explicación de las funciones de los archivos ".c".

2. Teoría

Antes de explicar los distintos tipos de TDAs es necesario explicar el concepto de nodo. Este se trata de un conjunto de dos punteros los cuales uno hace referencia al elemento que se quiera almacenar y otro el cual apunta al próximo nodo.

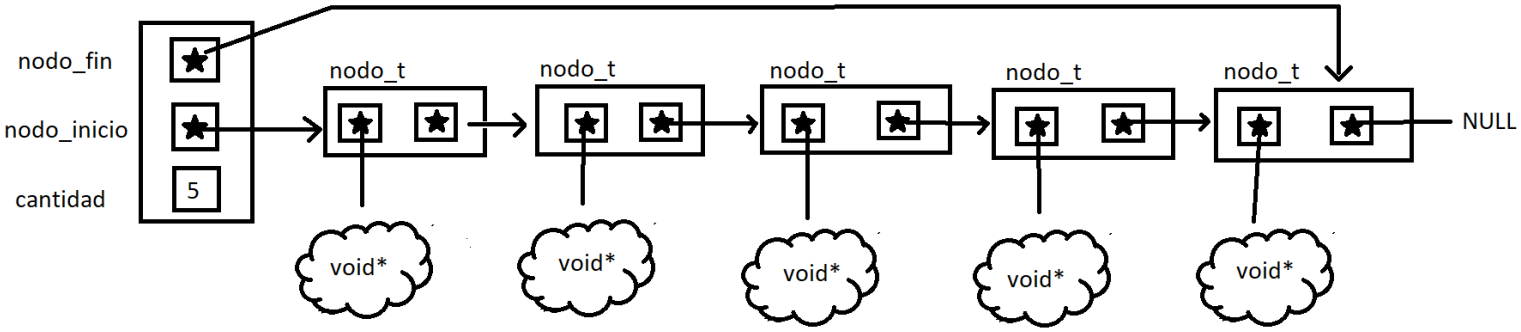
LISTA:

El primer TDA es lista. Este consiste en 3 elementos claves. Por un lado, está el nodo_inicio el cual apunta al primer nodo de la lista y que nos permite poder desplazarnos al siguiente. Luego esta el nodo_fin que apunta al último nodo de la lista y que su puntero al próximo nodo apunta a NULL. Y por último, pero no menos importante esta la cantidad de nodos o elementos que tiene la lista.

Este tipo de lista se conoce como lista simplemente enlazada ya que cada nodo esta enlazado solamente con su próximo nodo atreves del puntero siguiente. También existen las listas doblemente enlazadas las cuales usan

nodos diferentes que consisten en tener además del puntero siguiente, un puntero al nodo anterior.

Lista simplemente enlazada (utilizada en lista.c):



Otro caso de lista es la circular la cual es muy parecida a la lista simplemente enlazada, pero tiene una diferencia clave: el último nodo (nodo_fin) tiene como próximo nodo (nodo_fin->siguiente) al primero de todos, dando esta imagen de "circularidad".

Y el último tipo de lista es el de lista circular doblemente enlazada, que como bien dice el nombre, es una combinación entre la lista circular y la doblemente enlazada. Básicamente es una circular (ósea que el último nodo apunta al primero) pero usa los nodos de la doblemente enlazada, teniendo al primer nodo apuntando al último.

COLA:

El segundo TDA es la cola. Este es un caso particular de la lista simplemente enlazada pero que sigue el concepto de "FIFO" (First In First Out). Esto quiere decir que en el momento que nosotros queramos sacar un elemento de la cola, vamos a sacar si o si el primer elemento de la cola.

Existen otros casos de cola que no usan nodos, sino que usan un vector. Este puede ser estático (ósea que tiene un tamaño fijo) o dinámico (que puede pedir más memoria y agrandar el vector). Estos casos de cola también siguen el concepto de "FIFO".

PILA:

El tercer TDA es la pila. Este, al igual que la cola, es un caso especial de la lista simplemente enlazada pero a diferencia de la cola sigue el concepto de "LIFO" (Last In First Out). Este se basa en que en el momento que nosotros queramos sacar un elemento de la pila, si o si vamos a tener que sacar el último elemento agregado. Un ejemplo cotidiano de esto podría ser una pila de platos, si nosotros queremos sacar algún plato si o si tiene que ser el que está más arriba.

También existen casos de pilas que no usan nodos y en vez de eso usan vectores ya sea dinámicos (que pueden agrandarse pidiendo memoria) o estáticos (con un tamaño fijo).

3. Funciones

3.1 lista_insertar ()

La función lista_insertar() se encarga de agregar un nuevo elemento al final de la lista. Para esto pide memoria para un nodo auxiliar el cual guarda un puntero al elemento que queremos agregar. Luego de esto pueden ocurrir dos cosas:

1) Si la lista esta vacía iguala el nodo_incio y nodo_fin al nodo auxiliar.

2) Si la lista no esta vacía simplemente iguala el puntero del nodo_fin al nodo auxiliar y marca el nodo auxiliar como el último elemento igualándolo con el nodo_fin.

Al terminar uno de estos dos caminos aumenta la cantidad de elementos sumándole 1 a lista->cantidad y devuelve la lista con el elemento nuevo agregado.

3.2 lista_insertar_en_posicion ()

La función lista_insertar_en_posicon() se encarga de insertar un nuevo elemento en la posición especificada. Para poder lograr esto pide memoria para un nodo auxiliar el cual almacena un puntero al elemento que se quiera agregar. Luego de esto el problema se divide en 3.

1) Si la posición en la cual se quiera agregar un nuevo elemento es cero entonces significa que tenemos que agregar el nodo auxiliar al principio de todo. Para lograrlo iguala nodo_aux->siguiente al nodo_inicio y después iguala nodo_inicio al nodo_aux. Si aparte de esto la lista estaba vacía, iguala el nodo_fin al nodo_aux.
2) El segundo caso es que la posicon sea menor a la cantidad de elementos de la lista. Para lograrlo crea un nuevo nodo llamado actual el cual va a iterar las veces necesarias para pararse en el nodo anterior a la posicon que queremos insertar el nodo auxiliar. Una vez logrado esto simplemente iguala nodo_aux->siguiente al siguente del actual e iguala el siguente del actual al nodo_aux.
3) El tercer y último caso se da cuando la posicion no existe o sea justo la ultima posicion. Por ende se insertara el elemento al final de la

lista. Para lograrlo se diferencia entre dos casos: si la lista estaba vacia simplemente iguala el nodo_incio y nodo_fin al nodo_aux. Si la lista no estaba vacia iguala el siguente del nodo_fin al nodo_aux e iguala nodo_fin al nodo_aux, haciendolo el último nodo de la lista.

Una vez terminado alguno de estos 3 casos aumenta la cantidad de elementos de la lista por uno y devuelve la lista con el elemento agregado.

3.3 lista_quitar ()

La función lista_quitar() se encarga de eliminar el último elemento de la lista. Para poder lograr esto crea un nodo llamado actual, que se iguala al primer nodo (nodo_inicio), el cual se va a usar para poder navegar la lista. Una vez hecho esto la función se divide en dos casos:

1) Si la función tiene mas de un elemento, actual va a moverse hasta el anteúltimo nodo. Luego se guarda el elemento del último nodo en un puntero aparte llamada resultado. Después se iguala el nodo_fin al actual y se libera el próximo del actual.
2) Si la función tiene un elemento se iguala el puntero resultado al elemento y después se libera el único nodo que existe.

Una vez realizado uno de estos dos casos la función devuelve el puntero resultado.

3.4 lista_quitar_de_posicion ()

La función lista_quitar_de_posicion() se encarga de eliminar un elemento de una posición especificada. Para lograrlo crea un puntero llamado

resultado en el cual se va almacenar el elemento que se quiere eliminar. Luego de esto el problema se divide en 3 casos:

1) Si la posicon en la que se quiere eliminar es un valor igual o mayor que la cantidad de elementos simplemente se llama a la funcion lista_quitar() que se encargue de eliminar el último elemento.
2) Si la posicion es cero significa que tiene que eliminar el primer nodo de todos. Para esto crea un puntero nodo llamado actual el cual apunte al primer nodo. Luego se iguala nodo_inicio al su proximo nodo. Se iguala el elemento en actual al puntero resultado y se libera el puntero actual. Por último se disminuye por uno la cantidad de elementos de la lista.
3) Si la posicon esta entre el primer y último elemento lo que se hace es crear un puntero nodo llamado actual que apunte al primer nodo de todos. Después de esto se desplaza hasta una posiocn anterior de la que se quiere eliminar. Se crea otro puntero nodo llamado proximo_al_actual que se iguala al siguiente del nodo que se quiere eliminar. Luego de esto se iguala el puntero resultado al elemento del nodo a eliminar y se libera el nodo. Al hacerlo se conecta entre el actual->siguiente y el proximo_al_actual. Por último se disminuye por uno la cantidad de elementos de la lista.

Al terminar alguno de estos 3 casos la funcion devuelve el puntero resultado.