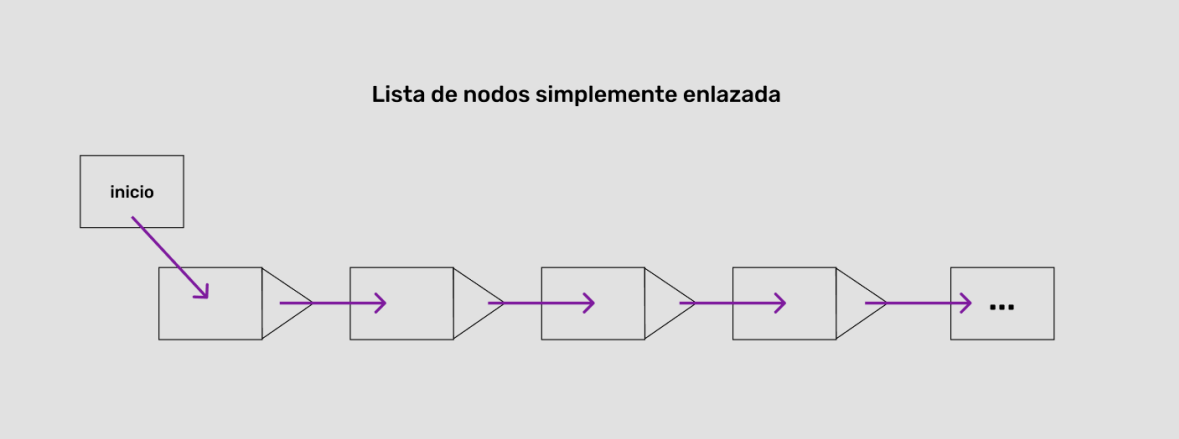
FACULTAD DE INGENIERIA
Universidad de Buenos Aires

TDA Lista y extras

[7541/9515] Algoritmos y Programación II



Segundo cuatrimestre de 2021

Alumna: CONVERSO, Lara Daniela

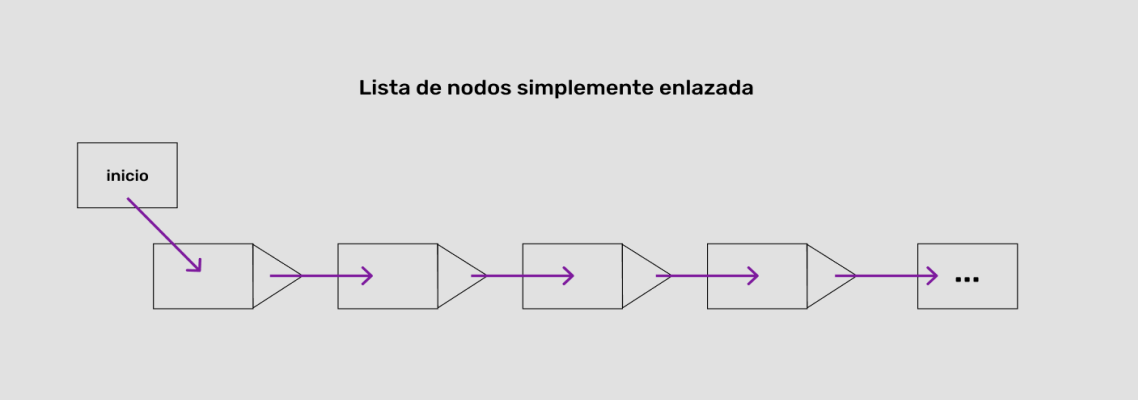
Número de padrón: 107632

Email: lconverso@fi.uba.ar

1. Detalles de implementación

Lista.c

En el archivo lista.c se encuentran implementadas las funciones presentadas en lista.h, que corresponden a la funcionalidad de la lista.

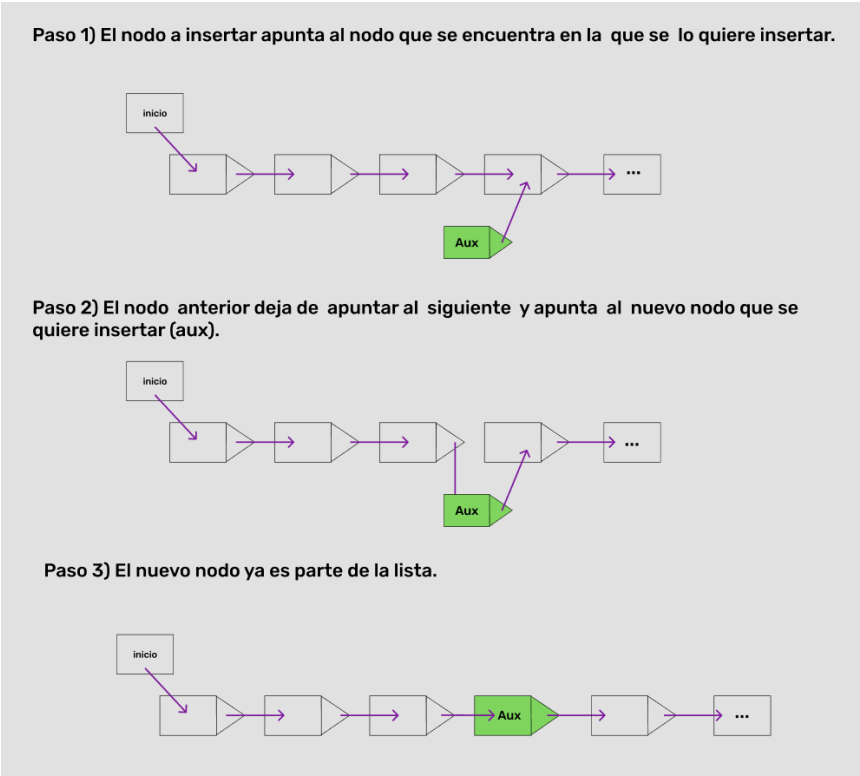


Esta es la representación grafica de una lista de nodos simplemente enlazada.

Funcionalidades de lista:

Lista_crear: se encarga de reservar memoria, y asi crear la lista, de ser error retorna NULL, ya que la operación es fallida.

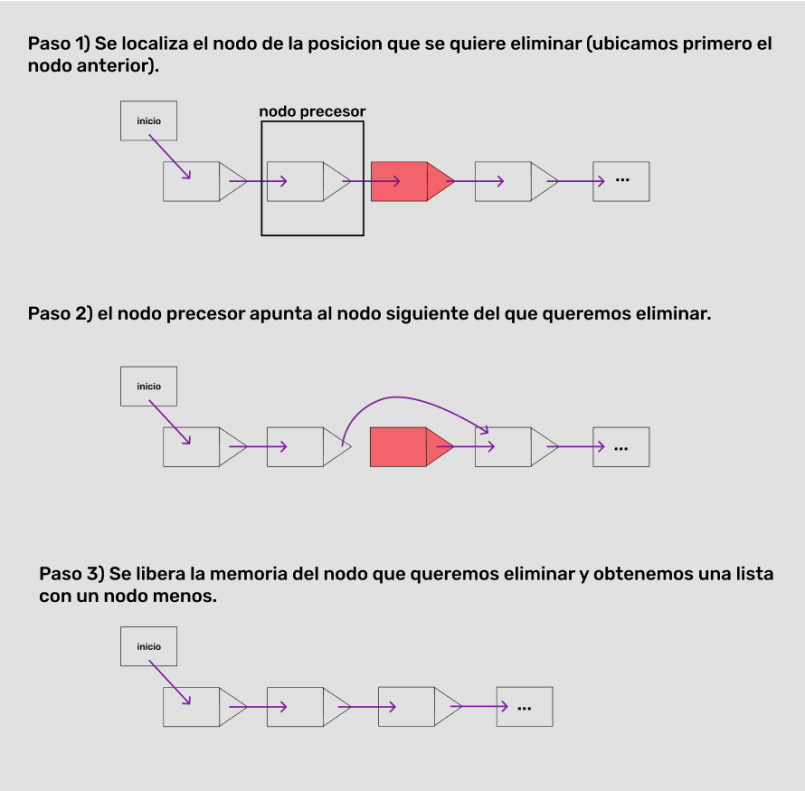
Lista_insertar: se encarga de insertar elementos en la lista, primero crea el nuevo nodo, reservando memoria para este mismo, y después en caso de que la lista se encuentre vacía, agrega un nodo al inicio, y en caso de ya tener elementos agrega es te nodo al final. En el caso de lista_insertar_en_posicion, al recibir una posición por parte del usuario el nodo debe agregarse en la posición que el usuario requiera. Su funcionamiento crea el nodo a ingresar, reservando memoria para este. En el caso de que la posición pasada por parámetro es 0 (la primera posición de la lista), el nuevo nodo debe ser agregado al principio de la lista, teniendo en cuenta que no se pierda la referencia con el nodo que ya se encuentra en la primera posición (si la lista tiene elementos), en el caso de que la posición pasada por parámetro es mayor a la cantidad de nodos en la lista se agrega al final de esta, si no es ninguno de estos casos, se busca el nodo anterior a la posición pasada por parámetro con una función auxiliar recursiva que cree (“encontrar_nodo”) que encuentra el nodo con la posición que es pasada por parámetro.



Representación grafica de la función insertar en posición en la lista.

Lista_quitar: se encarga de quitar el ultimo elemento de la lista, y de liberar la memoria reservada para este mismo. Nos retorna el elemento que quitamos, en caso de error retorna NULL.

En el caso de lista_quitar_de_posicion, se quita el elemento que se encuentre en la posición pasada por parámetro, si la posición es 0 (la primera posición de la lista), se encarga de mantener a los nodos enlazados con el uso de variables auxiliares y luego libera la memoria del elemento que se quiere quitar.



Representación grafica de la función quitar en posición en la lista.

Lista_elemento_en_posicion: utiliza la función propia, mencionada anteriormente, "encontrar_nodo" pasando por parámetro la posición de la cual queremos encontrar el elemento, y retorna el elemento de dicha posición.

Lista_primero, Lista_ultimo y Lista_tamanio: las tres funciones acceden al elemento que les corresponde.

Lista_destruir: se encarga de liberar la memoria, de cada uno de los nodos, para eso reutilice la función lista_quitar_de_posicion mientras se cumpla que la cantidad de elementos de la lista es mayor a cero.

Pila.c

En el archivo pila.c se encuentran implementadas las funciones presentadas en pila.h, que corresponden a la funcionalidad de la pila. Para su implementación reutilicé las funciones de lista.

Aclaraciones de las funcionalidades de pila:
En la implementación de las funciones apilar y desapilar use la funciones de lista_insertar_en_posicion y lista_quitar_de_posicion en donde el parámetro de la

posición es 0, dado que para cumplir con la funcionalidad de la pila (LIFO) explicado en las preguntas teóricas.
Para poder utilizar las funciones propias de lista se realizaron casteos para respetar los tipos de dato.

Cola.c

En el archivo cola.c se encuentran implementadas las funciones presentadas en cola.h, que corresponden a la funcionalidad de la cola. Para su implementación al igual que con pila, reutilicé las funciones de lista.

Aclaraciones de las funcionalidades de cola:
En la implementación de las funciones encolar reutilizo la función de lista_insertar dado a que cada vez que encolo un elemento, se debe encolar al final de esta. Por otro lado en lista desencolar, como el elemento que hay que sacar el que se encuentra en la primera posición, retutilize la función lista_quitar_de_posicion con el parámetro de posición en cero dado que es la primera.
Para poder utilizar las funciones propias de lista se realizaron casteos para respetar los tipos de dato.

Pruebas.c

En este archivo se encuentran las pruebas desarrolladas para comprobar el buen funcionamiento de los tres tipos de estructuras.
Las pruebas desarrolladas están organizadas por TDA, y función implementada. En la ejecución primero se prueban los casos en tener un TDA lista nulo, o inexistente.
Las pruebas de pila y cola se realizan dentro de tres bloques, el primero es de la creación del TDA, el segundo se centra en las pruebas con el TDA pila o cola NULL, y el tercer bloque prueba su funcionalidad.
Se crearon 123 pruebas propias para cumplir con el buen funcionamiento del TDA.

2. Compilación

Para compilar el TDA, se utiliza un makefile.
El comando utilizado para compilar el archivo con las pruebas es: "`make valgrind-pruebas`". Y para compilar con el archivo de ejemplo brindado por la catedra se puede utilizar "`make`". Ambos comandos también muestran el resultado de la herramienta Valgrind, que se utiliza para chequear el buen uso de la memoria dinámica.

3. Preguntas teóricas

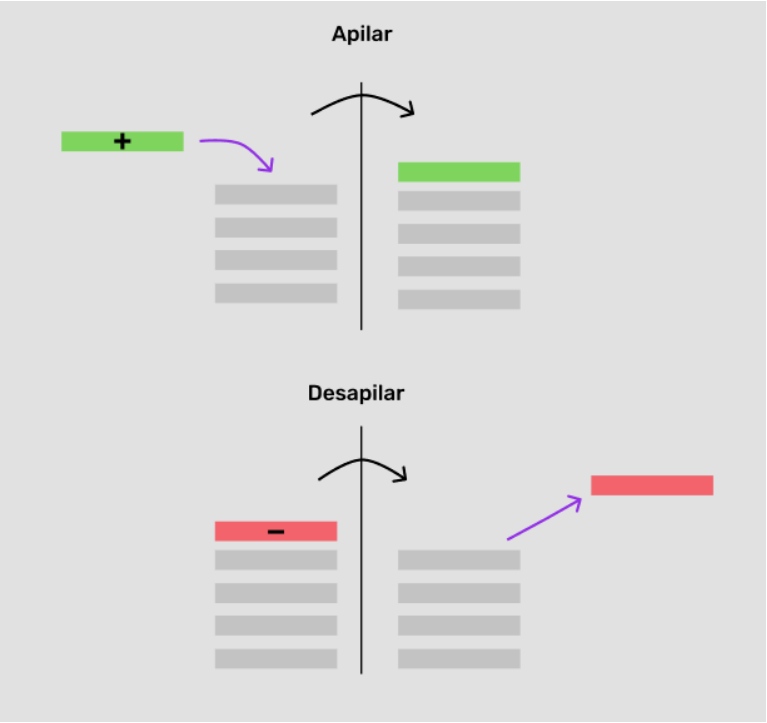
- **Explique que es una lista y como funciona. Explique que tipos diferentes de implementaciones conoce.**

  Una lista es un tipo de dato abstracto que nos permite almacenar elementos de manera ordenada y agrupada.
  La lista sirve para agrupar elementos, y cada elemento tiene un sucesor, menos el último, y un predecesor, menos el primero.
  Existen tres tipos de implementaciones, la primera, como un vector estático, en segundo lugar como un vector dinámico, y en tercer lugar con nodos. Esta última implementación, se subdivide en otros cuatro tipos, esto quiere decir que hay cuatro tipos de listas con nodos; La lista de nodos simplemente enlazada, la lista de nodos doblemente enlazada, la lista de nodos circular simplemente enlazada y por ultimo la lista circular doblemente enlazada.

  La lista implementada en este TDA es una lista de nodos simplemente enlazada.
  Este tipo de lista, se implementa con nodos, y cada uno tiene referencia hacia el nodo siguiente, exceptuando el último. La lista en todo momento tiene referencia al primer nodo.

- **Explique que es una pila y como funciona. Explique que tipos diferentes de implementaciones conoce.**

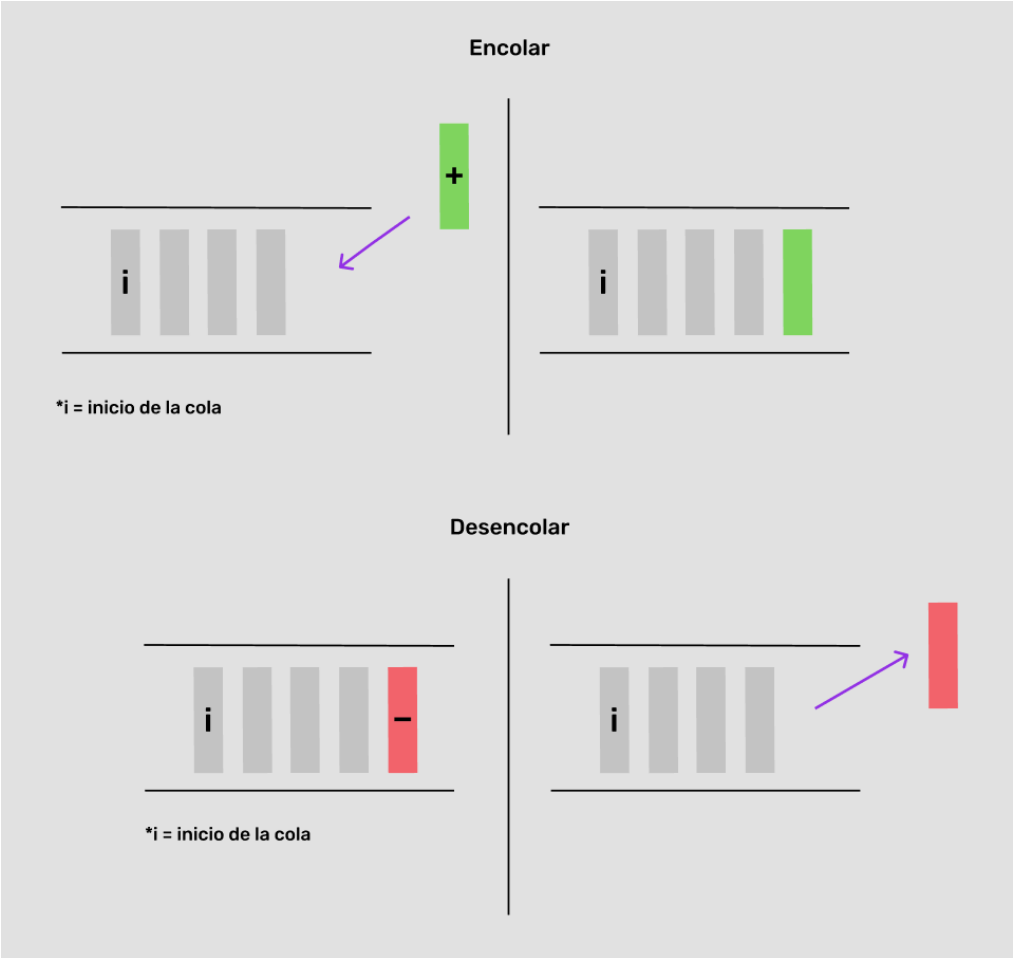
  Una pila es un tipo de dato abstracto, con la característica de que utiliza el modelo LIFO (Last in first out), que hace referencia a que se apilan y desapilan cosas desde la ultima posición, la analogía perfecta es una pila de platos, estos son capaces de apilarse un arriba de otro, pero al momento de desapilar siempre se saca el que se encuentra primero (o mas arriba).
  Se puede implementar con un vector estáticos, vector dinámico y con nodos.



Representación grafica de la función apilar y desapilar.

- **Explique que es una cola y como funciona. Explique que tipos diferentes de implementaciones conoce.**

  Una cola es un tipo de dato abstracto, con la característica de que usa el modelo FIFO(First in First Out), que hace referencia que se encolan cosas desde el final y se desencolan desde el principio. El ejemplo mas claro es la cola del supermercado, la fila se arma desde la caja hacia “atrás” y una vez que el que se encontraba por la primera posición pasa por la caja, ya no pertenece mas a la cola.
  Se puede implementar con un vector estáticos, vector dinámico y con nodos.



Representación grafica de las funciones encolar y desencolar.