ALGORTIMOS Y PROGRAMACION II, CURSO 04-MENDEZ.
INFORME DEL TDA#1: LISTA.

Alumno: Víctor Zacarías; Padrón: 107080; DNI: 95.979.449.

Mail: vzacarias@fi.uba.ar

-Descripción: En este informe se incluyen explicaciones teóricas breves sobre cada uno de los TDAs implementados (que son, como funcionan), así como esquemas y explicaciones sobre la funcionalidad de las funciones y estructuras utilizadas para poder implementar cada TDA.

-Introducción teórica:

1-Lista: Una lista es un tipo de dato abstracto que permite agrupar un conjunto de elementos como si se tratase de un vector, pero con la particularidad de que la lista provee funciones que hacen del trabajo con la información almacenada en ella una tarea sencilla, y que los elementos agrupados en la lista no se encuentran restringidos a un tipo de dato único como es el caso común en los vectores.

Cuando se habla de que la lista provee funciones que hacen del trabajo con la información almacenada en ella una tarea sencilla, se refiere a que, a diferencia de trabajar con un vector, el tener un TDA que incluya funciones que realizan las operaciones que se necesite que haga, deja de lado el trabajo manual que se tendría que hacer cada vez que se quiera almacenar información en un vector y luego hacer algo con esa información. La lista provee funciones que realizan estas tareas manuales sin necesidad de implementarlas (en *"cómo funciona una lista"* se especifican estas tareas).

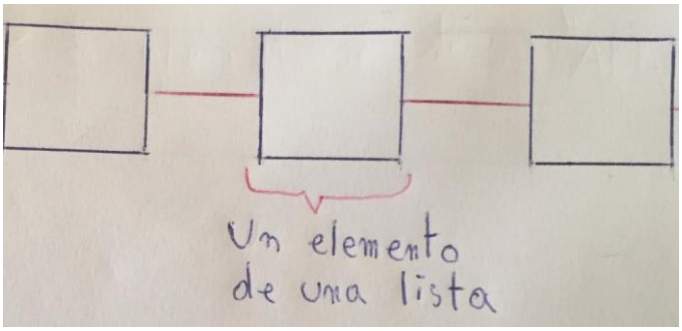
¿Cómo funciona una lista?: Para poder cumplir de la definición de lista, una lista debe poder agrupar un conjunto de elementos, permitiendo, insertarlos en una posición cualquiera de la misma, eliminarlos en una posición cualquiera de la misma, ver su tamaño, así como buscar un elemento en una posición cualquiera de la lista.

Existen diversas formas de implementar una lista, cada una con sus pros y contras (uso de memoria dinámica, complejidad algorítmica, limitación en cuanto a uso del stack), algunas de estas son:

1.1-Lista con vector estático (Se encuentra muy limitado por el tamaño del stack y por lo complejo de implementar algunas de las operaciones de la lista).

1.2-Lista con vector dinámico (Se encuentra muy limitado a la contigüidad de la memoria dinámica, y a lo complejo de algunas de sus operaciones).

1.3-Lista con nodos simplemente enlazados o doblemente enlazados (Su limitación se centra mayormente a cuanta memoria dinámica se dispone para hacer este tipo de implementación).



2-Pila: Una pila es un tipo de dato abstracto que permite agrupar un conjunto de elementos en una especie de pila (piénsese como una pila de platos), colocando uno sobre otro, donde el último elemento "apilado" se denomina tope, estos elementos no se encuentran restringidos a un único tipo de dato.

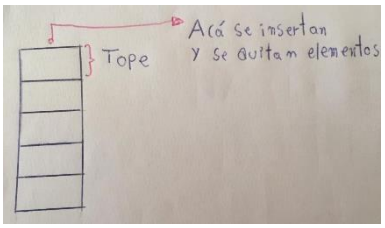
¿Cómo funciona una pila?: En una pila solo se pueden apilar o desapilar elementos, lo que quiere decir que, si se quiere acceder a una posición intermedia de la pila, habrá que desapilar los elementos necesarios hasta llegar al buscado (Se desapila el último elemento apilado, tope, una pila es una estructura tipo LIFO, donde el último elemento que entra es el primero que sale). Solo se pueden agregar elementos a una pila colocándolos sobre el tope (el último elemento agregado pasa a ser el nuevo tope). Solo se pueden eliminar elementos de una pila, desde el tope, una vez que se quita un elemento el tope, este es eliminado, y el tope actualizado.

Existen diversas formas de implementar una pila, cada una con sus pros y contras (uso de memoria dinámica, complejidad algorítmica, limitación en cuanto a uso del stack), alguna de estas formas son:

2.1-Pila con vector estático: Se limita exclusivamente al tamaño del stack para definir vectores donde implementar la lista.

2.2-Pila con vector dinámico: Se limita exclusivamente a la contigüidad de la memoria dinámica en el heap, un vector dinámico ocupa posiciones de memoria contiguas al igual que uno estático.

2.3-Pila con nodos: Se limita exclusivamente al límite que se puede tener al usar memoria dinámica.



3-Cola: Una cola es un tipo de dato abstracto que permite agrupar un conjunto de elementos en una estructura que visualmente se podría imaginar como un vector, pero que, a diferencia de él, solo permite operar sobre el inicio o fin de la cola.

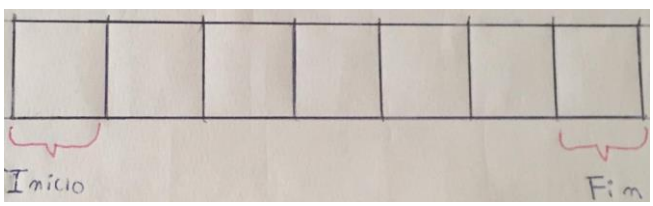
¿Cómo funciona una cola?: Una cola puede imaginarse como una fila de supermercado, donde se entra a ella desde el final de esta, y el primero en entrar es el primero en salir (una estructura de tipo FIFO). A estas dos acciones se reducen las operaciones de una cola, ingresar un elemento desde el final, y sacar el primer elemento.

Existen diversas formas de implementar una cola, cada uno con sus pros y contras (uso de memoria dinámica, complejidad algorítmica, limitación en cuanto a uso del stack), alguna de estas formas son:

3.1-Cola con vector estático: Su limitación está en la limitación en cuanto al tamaño del stack, y que al desencolar (sacar) algún elemento, es necesario reorganizar el vector para no dejar espacios vacíos.

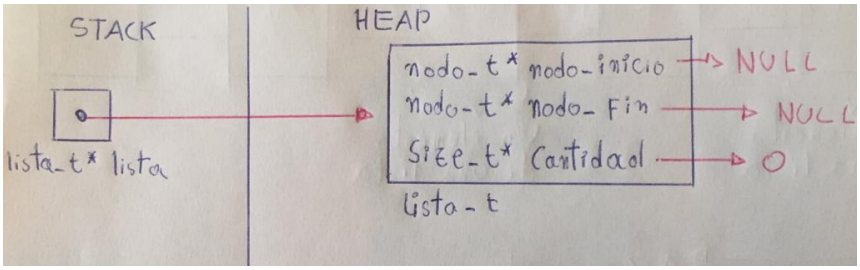
3.2-Cola con vector dinámico: Su limitación es la misma que la del vector estático al desencolar, con el añadido de que se debe disponer de memoria contigua en el heap al momento de hacer esta implementación.

3.3-Cola con nodos: Su limitación está en el uso del heap.

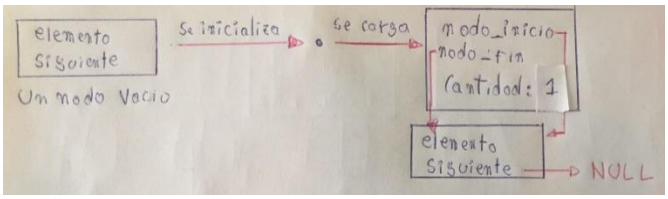


-Implementación del TDA lista: A continuación, se detallarán las implementaciones tomadas en el TDA lista, el mismo fue hecho a partir de nodos enlazados.

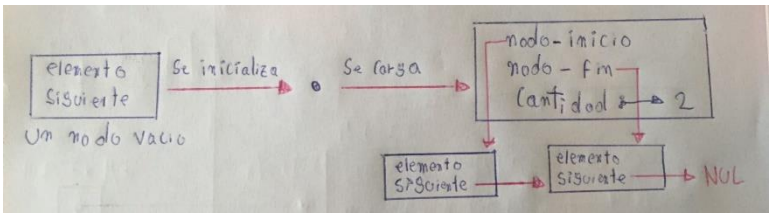
1-Funcion lista_crear: Crea una lista vacía, e inicializa sus campos en NULL o cero (0) según corresponda.



2-Funcion lista_insertar: Se encarga de insertar elementos al final de una lista, si la lista está vacía entonces inserta el elemento al principio de ella. Para esas operaciones se hace uso de un nodo_auxiliar, sobre el cual se carga la información del elemento a insertar y posteriormente se inserta a la lista. Si hay algún error en la función, la misma devuelve NULL, de lo contario devuelve la lista.

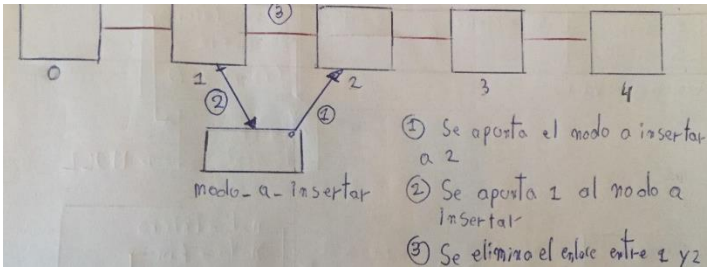


Se inserta un elemento a una lista vacía.



Se inserta un elemento al final de una lista con un elemento.

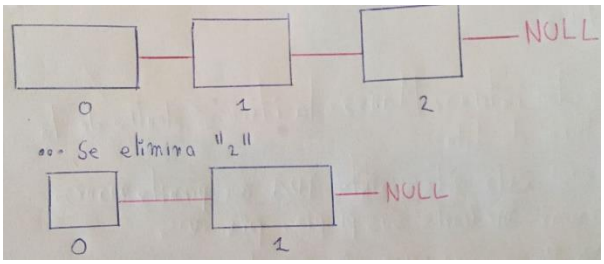
3-Funcion lista_insertar_en_posicion: Se encarga de insertar un elemento a través de un nodo en una posición cualquiera de la lista. Si la lista está vacía, se hace uso de la función lista_insertar para insertar el elemento al principio. Si la posición en la que se quiere insertar el elemento es inexistente, este se inserta al final de la lista, haciendo uso de la función lista_insertar. Fuera de estos casos borde, la función hace uso de un nodo_a_insertar sobre el que se carga la información que se quiere agregar a la lista, y de un nodo_auxiliar que permite insertar el nodo_a_insertar sin perder acceso al resto de los nodos de la lista. La función recorre la lista hasta la posición en donde se quiere insertar el nuevo elemento, guardando la información del nodo anterior al de dicha posición en nodo_auxiliar para hacer la inserción. En caso de error devuelve NULL, o la lista si todo salió bien.



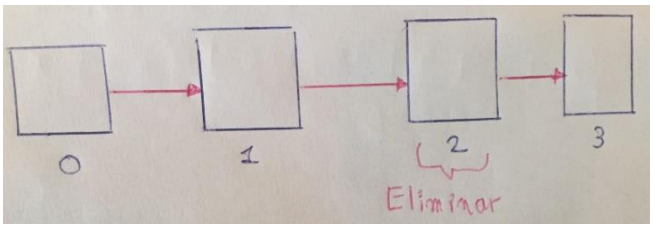
Para este ejemplo, nodo_auxiliar es el de la posición 1, se hace uso de el porqué, es el quien tiene información sobre quien es el siguiente nodo, esto permite insertar un elemento sin "romper" la estructura de la lista.

4-Funcion lista_quitar: Se encarga de eliminar el último elemento de la lista, si la lista no existe, entonces no se hace nada. La función devuelve el elemento eliminado o NULL en caso de error.

Para realizar la eliminación del último elemento de la lista, se recorre la lista hasta el penúltimo elemento (y se lo guarda como auxiliar), quien va a pasar a ser el nuevo último elemento de la lista una vez hecha la eliminación (para esto sirve guardarlo como auxiliar).



5-Funcion lista_quitar_de_posicion: Se encarga de eliminar un elemento cualquiera de la lista. Si la lista está vacía no hace nada. Si la lista tiene un único elemento, se hace uso de lista_quitar para eliminarlo. Si la posición que se desea eliminar es inexistente, se elimina la última posición de la lista haciendo uso de lista_quitar. Fuera de estos casos bordes, la función se encarga de recorrer la lista hasta una posición antes del elemento a eliminar, guardando la información del elemento a eliminar y del elemento anterior a elemento eliminar, esto ultimo se hace ya que el elemento a eliminar tiene información sobre quien es su siguiente, y para poderse eliminar correctamente sin romper son la estructura de lista, es necesario apuntar el elemento siguiente del nodo anterior al nodo eliminar al siguiente del nodo a eliminar. Devuelve el elemento eliminado o NULL en caso de error.

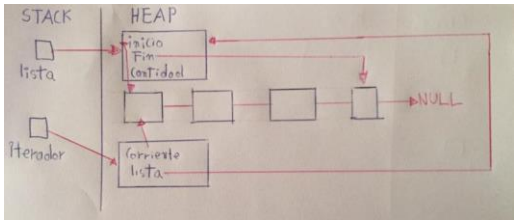


Para poder eliminar el 2, es necesario tener variables auxiliares que guarden a 1 y a 2, si se elimina directamente el 2, se pierde el acceso el resto de la lista. Hacer esto es posible ya que los nodos de una lista tienen información sobre quiénes son sus siguientes.

6-Funcion lista_elemento_en_posicion: Se encarga de devolver un elemento de una posición de una lista, si la posición no existe devuelve NULL. Para hacer esto se hace uso de una estructura iterativa.

7-Funcion lista_destruir: Se encarga de liberar toda la memoria ocupada por la lista, esto es la estructura que permite acceder a sus nodos, y cada uno de sus nodos. Para hacer esto ultimo hace uso de una estructura iterativa que recorre todos los nodos de la lista liberándolos.

8-Funcion lista_iterador_crear: Crea un iterador, que es una estructura que permite iterar sobre cada uno de los elementos de la lista. El iterador contiene en sus campos a la lista y al elemento actual de la misma, todo iterador comienza con su campo corriente (elemento actual de la lista) apuntando a la primera posición de la lista.



Un iterador creado con la función lista_iterador_crear

9-Funcion lista_iterador_tiene_siguiente: Se encarga de analizar si el iterador puede iterar a otro elemento o no. La función devuelve false si el iterador no existe o si el elemento actual es NULL (lo que quiere decir que no hay mas elementos para iterar), en otro caso devuelve true.

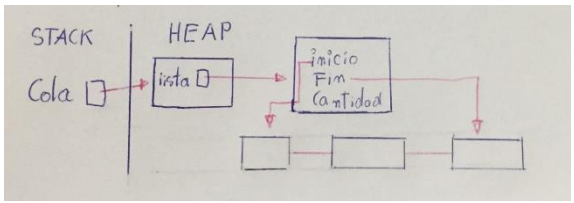
10-Funcion lista_iterador_avanzar: Se encarga de avanzar el iterador a la siguiente posición de una lista. Devuelve true o false dependiendo del resultado de la acción. Devuelve true si se avanzó a una posición valida de la lista, devuelve false si no se pudo avanzar o si se avanzo a NULL (NULL es el siguiente de la última posición de la lista)

11-Funcion lista_iterador_elemento_acutal: Devuelve el elemento actual de la lista apuntado por el iterador.

12-Funcion lista_con_cada_elemento: Es un iterador interno. Le aplica una función a cada uno de los elementos de la lista, esto a través de una estructura iterativa que recorre la lista de inicio a fin mientras que la función aplicada a cada elemento devuelva true, en caso de que se recorra toda la lista o la función devuelve false, se sale de la estructura iterativa, y la función lista_con_cada_elemento devuelve la cantidad elementos leídos por la función mandada como parámetro a lista_cona_cada_elemento.

NOTA: Cada una de estas funciones hace verificaciones sobre los parámetros mandados, que en caso de ser inválidos devuelven algún valor que dan a entender que hubo algún error. Estas verificaciones incluyen todas las verificaciones necesarias al momento de trabajar con memoria dinámica (detalles sobre esto no fueron incluidos en las explicaciones anteriores por cuestiones didácticas).

-Implementación del TDA cola: Para la misma, fueron reutilizadas en su totalidad las funciones implementadas en lista; El comportamiento de ambas estructuras, y la cola no es más que una abstracción de la lista.



La estructura de una cola.

1-Funcion cola_crear: Crea una cola a partir de una lista (ver imagen anterior).

2-Funcion cola_encolar: Agrega elementos a la cola. Hay que recordar que en una cola solo se pueden agregar elementos al final de esta, por lo que se hace uso de la función lista_insertar, para insertarlos al final y respetar la definición de cola. Si todo salió bien, devuelve un puntero a la cola, de lo contrario devuelve NULL.

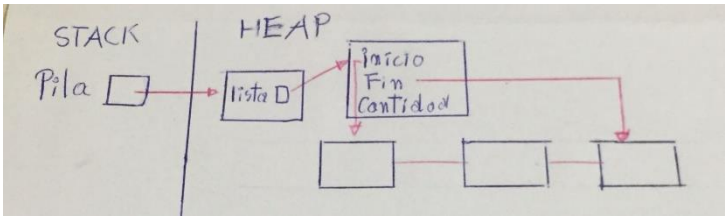
3-Funcion cola desencolar: Elimina elementos de la cola. Hay que recordar que en una cola los elementos solo se pueden eliminar desde el principio (el primer elemento en entrar es el primero que sale, como una fila del supermercado). Para poder funcionar bien se hace uso de la función lista_quitar_de_posicion con la posición 0, de esta forma siempre que se quiera desencolar un elemento a partir de una estructura de lista, el resultado será el esperado. Devuelve el elemento eliminado, o NULL en caso de error.

4-Funcion cola_frente: Devuelve el primer elemento insertado en la cola, o NULL en caso de error (Pensar nuevamente como una fila de supermercado, el frente es la primera persona que llego a la fila.

5-Funcion cola destruir: Se encarga de liberar toda la memoria necesaria para la creación de la cola, es decir, libera toda la memoria usada para la lista (cada uno de los nodos de ella), el puntero a la lista y finalmente el puntero a la cola.

NOTA: No se incluyen detalles sobre las funciones cuya implementación es clara y consta de pocas líneas, o aquellas que no necesitan de una aclaración teóricas sobre el comportamiento de una cola. Para estas funciones excluidas basta con entender que para implementar una cola se hizo una abstracción de una lista, entonces: el tamaño de una cola es lo mismo que el tamaño de una lista, y así, si una cola esta vacía, entonces una lista también.

-Implementación del TDA pila: Para su implementación fueron reutilizadas en su totalidad las funciones utilizadas para crear la lista, esto a través de una abstracción de lo que es una lista y su semejanza con una pila.



Estructura de una pila, implementada a partir de una cola.

1-Funcion pila_crear: Crea una pila a partir de una lista (ver imagen anterior).

2-Funcion pila_apilar: Agrega elementos en el tope de una pila. Hay que recordar que en una pila solo se pueden insertar elementos al final de esta, esta última posición es la que se denomina tope. Para poder hacer esto a través de una lista, se hace uso de la función lista_insertar (que solo inserta elementos al final de una lista), de forma que cada vez que se inserte un elemento se inserte al final.

3-Funcion pila_desapilar: Desapila elementos del tope de una pila. Hay que recordar que una pila solo se pueden quitar elementos desde el tope de esta (seria como tener una lista y solo poder quitarle el ultimo elemento). Para poder hacer esto a través de una lista, se hace uso de la función lista_quitar quien elimina los elementos del final de una lista.

4-Funcion pila_tope: Devuelve el tope de una fila. Para poder implementar eso a través de una lista, se hace uso de la función lista_ultimo, que devuelve el último elemento de esta.

5-Funcion pila_destuir: Libera toda la memoria reservada para la creación de una pila, esto es: la estructura de pila, el campo lista de la pila, y cada uno de los nodos de la lista (el orden de liberación es el siguiente: nodos de la lista, lista, pila).

NOTA: No se incluyen explicaciones sobre funciones cuyo funcionamiento consta de pocas líneas y es claro. Así como de aquellas funciones que no requieren de una explicación teórica que de a entender que abstracción de la lista se tuvo que realizar para poder implementar la cola correctamente. Con esto se hace referencia a:

El tamaño de una pila es el mismo que el de una lista, y si una pila esta vacía, entonces una lista también.

NOTA FINAL: La lista tiene un comportamiento muy similar al de una pila o de una cola, es por esto que fue posible realizar una abstracción de la misma para poder implementar la pila y la cola.