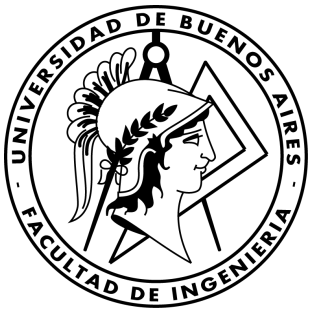

TDA Lista

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

Alumno:	de San Vicente, Hernán
Número de padrón:	108800
Email:	hdesanvicente@fi.uba.ar

Índice

1. Introducción

Idea general del TP, un breve resumen de que se pidió y de que forma.

2. Teoría

Respuestas a las preguntas teóricas (si no las hay podes borrar esta sección!)

1. TDA Lista y su implementación con nodos enlazados

   Una lista es una estructura que agrupa elementos, en donde cada uno de esos elementos tiene un sucesor(menos el último) y una predecesor(menos el primero). Se diferencia de los otros dos TDAs por no tener una política tan estricta de acceso a los elementos y ser mucho más flexible con su manejo. Tiene varias formas de ser implementado pero la que se va a explicar en este informe es la de nodos enlazados.

   Los nodos son estructuras mas pequeñas que adentro tienen información útil para cumplir con las políticas del TDA que los va a utilizar. Lo mas importante es el elemento que almacena cada nodo, pero también estos guardan una referencia al siguiente nodo o también, incluso, al anterior, dependiendo de si es un nodo simple (Figura 1) o doblemente (Figura 2) enlazado.

   La lista, en su interior, no guarda cada nodo que tiene (sino no tendrían sentido las referencias de los nodos y sería lo mismo que un vector) sino que guarda el primero y de ahí accede al resto siguiendo las referencias. Algunas cosas opcionales en esta implementación son la referencia al último nodo y la cantidad de nodos en la estructura, que simplemente agilizan algunas operaciones.

   Las operaciones más conocidas de este TDA (y su complejidad computacional con nodos enlazados) son:

   a) Crear: inicializa la lista vacía. CC: O(1)

   b) Insertar: mete un elemento al final de la lista. CC: O(1)

   c) Quitar: Saca el ultimo elemento de la lista. CC: O(1)

   d) Obtener: Obtiene un elemento en una posición específica. CC: O(n)

   e) Esta vacía: Verifica si la lista tiene elementos o no. CC: O(1)

   f) Destruir: Elimina la lista y cada uno de sus elementos. CC: O(n)

2. TDA Pila y su implementación con nodos enlazados

   La pila es, simplemente, una versión de la lista que tiene más restricciones o una política más rígida. En este TDA (Figura 3), no se puede acceder a otro elemento más que el último que se haya insertado (el tope) siguiendo la regla LIFO (Last In - First Out). Al ser una variación de la lista, la estructura puede ser la misma. Lo que cambia es la implementación de las operaciones que se pueden realizar (que aún así serían muy parecidas a las de la lista).

   Las operaciones más conocidas de este TDA (y su complejidad computacional con nodos enlazados) son:

   a) Crear: inicializa la pila vacía. CC: O(1)

   b) Apilar: mete un elemento al final de la pila. CC: O(1)

   c) Desapilar: Saca el ultimo elemento de la pila. CC: O(1)

   d) Obtener tope: Obtiene el ultimo elemento agregado a la pila. CC: O(1)

   e) Esta vacía: Verifica si la pila tiene elementos o no. CC: O(1)

   f) Destruir: Elimina la pila y cada uno de sus elementos. CC: O(n)

3. TDA Cola y su implementación con vectores estáticos (cola circular)

   La cola es el último de los TDAs que se plantean en este informe. También es una versión de la lista con más restricciones pero con políticas diferentes a las anteriores. En este TDA, tampoco se puede acceder a todos los elementos, sino que solo se puede ver el que esté primero o adelante de la cola. A diferencia de la pila, acá se sigue la regla FIFO (First In - First Out) que indica que se puede acceder solo al primer elemento que se haya insertado de los que se encuentran disponibles. La estructura también puede ser la misma que la de la lista y las operaciones también son parecidas. En esta caso, en vez de implementarlo con nodos enlazados, vamos a explicar cómo se implementa con un vector estático y siendo circular. Acá los elementos simplemente se guardan en posiciones de un vector, pero surge un problema al querer desencolar.

   Hacer la cola circular es una de las dos soluciones a ese problema (la otra siendo mover todos los elementos hacia adelante, que es muy costoso). En esta implementación, se necesita una variable llamada principio, que indique la posición del elemento que se encuentra adelante, y el tope que va siguiendo a esa variable. Se puede ver un ejemplo de esto en la Figura 4

   Las operaciones más conocidas de este TDA (y su complejidad computacional con vectores estáticos) son:

   a) Crear: inicializa la cola vacía. CC: O(1)

   b) Encolar: mete un elemento al final de la cola. CC: O(1)

   c) Desencolar: Saca el primer elemento de la cola. CC: O(1) o O(n) dependiendo de la implementación

   d) Obtener primero: Obtiene el elemento que se encuentra adelante en la cola. CC: O(1)

   e) Esta vacía: Verifica si la cola tiene elementos o no. CC: O(1)

   f) Destruir: Elimina la cola y cada uno de sus elementos. CC: O(n)

3. Detalles de implementación

Este trabajo se compila con al comando "gcc src/*.c ejemplo.c -o ejemplo" y se ejecuta con el comando "./ejemplo". En caso de querer correr con valgrind se ejecuta el comando "make" y si se quieren correr las pruebas se ejecuta "make valgrind pruebas".

3.1. Implementación de Lista

En este trabajo, la lista se implementó con nodos simplemente enlazados y, además de las funciones primitivas de una lista, se implementó un iterador, para recorrer la lista mas efectivamente, con sus respectivas funciones y algunas funciones extras que puede llegar a ser útiles (como lista_ultimo y lista_primero)

1. Crear, destruir y destruir todo

   Las funciones de crear y destruir simplemente reservan y liberan la memoria de la lista respectivamente. Destruir todo, por otro lado, recibe una función por parámetro y la ejecuta a cada uno de los elementos de la lista, liberando la lista por completo al final.

2. Insertar y quitar (En posiciones especificas o al final)

   Las funciones insertar y quita son simples gracias a que la estructura del TDA guarda el nodo del final. Cada una tiene sus diferentes precauciones tomadas en los casos de que las lista esté vacía, tenga un solo elemento o tenga mas de uno. En el caso que se quiera insertar o quitar un elemento en una posición específica, las precauciones a tomar son más y más complejas. Se tiene mucho cuidado para no perder la referencia a ningún nodo, ya sea usando auxiliares o no.

3. Buscar por elemento o por posición

   Las funciones de buscar iteran por la cadena de nodos hasta encontrarse con la condición dada de cada una. En el caso de buscar por elemento, se busca la igualdad del elemento pasado y el actual siendo iterado, y en el caso de buscar por posición se recorre la lista recursivamente usando un contador.

4. Lista vacía y tamaño

   Lista vacía solo verifica si el primer elemento es nulo o no y el tamaño está dado por la variable cantidad de la estructura.

5. Iteradores

   Para el TDA lista se creó un TDA auxiliar que permite recorrer la misma de forma mas eficiente. Esta, a su vez, tiene diferentes funciones.

   a) Crear y destruir

   La función crear simplemente reserva espacio en memoria para la estructura e inicializa sus variables, y destruir solo libera esa memoria.

   b) Avanzar y tiene siguiente

   La función de avanzar el iterador verifica que los nodos no sean nulos (excepto cuando se llega al final de la lista) y asigna el próximo nodo al actual, mientras que la función booleana de tiene siguiente simplemente se fija si el próximo elemento es nulo o no y devuelve true o false.

   c) Obtener elemento actual

   Esta función simplemente accede al nodo y devuelve el elemento.

   d) Iterador interno

   Este iterador no usa el TDA auxiliar, pero tiene una función parecida. Ésta recibe un lista y una función que ejecuta en cada uno de los elementos de esa lista, retornando la cantidad de elementos por los que se iteró

3.2. Implementación de Pila

El TDA pila se implementó usando la estructura de la lista e incluso llamando a las mismas funciones pero con ciertos cambios y casteando la variable que contiene a la pila a un lista_t. Las únicas funciones que hacen algo particular son apilar, desapilar y tope, el resto llaman a la función de lista con el mismo nombre.

1. Apilar: utiliza la función lista_insertar_en_posicion mandándole 0 como posición, así siempre se inserta .ªrriba".

2. Desapilar: utiliza la función lista_quitar_de_posicion mandándole 0 como posición, así siempre se quita el último elemento insertado.

3. Tope: No hay una función con el mismo nombre en lista, pero la función lista_primero tiene la misma función.

3.3. Implementación de Cola

Para el TDA cola se hizo exactamente lo mismo que con pila, tanto para la estructura como para las funciones. Lo único que se hizo diferente fue la función encolar, que en vez de usar lista_insertar_en_posicion se usó lista_insertar, ya que los elementos se insertan al final

4. Diagramas

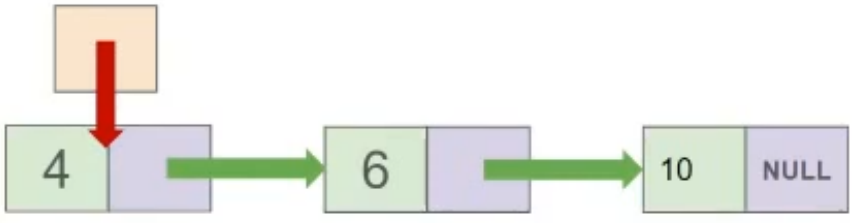

Figura 1: Lista de nodos simplemente enlazada.

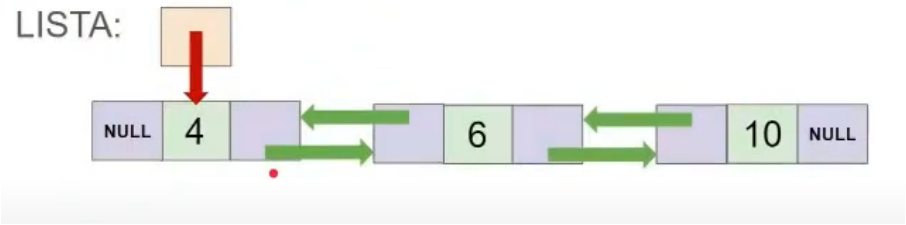

Figura 2: Lista de nodos doblemente enlazada.

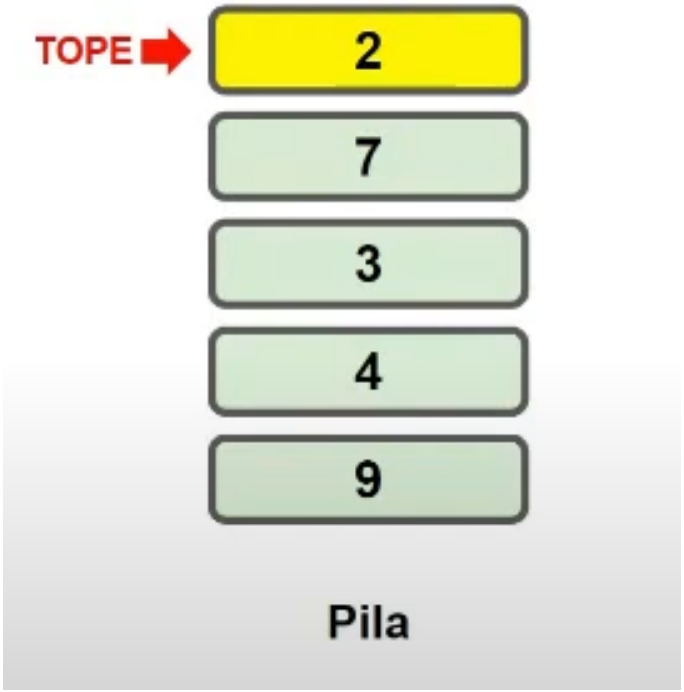

Figura 3: Pila.

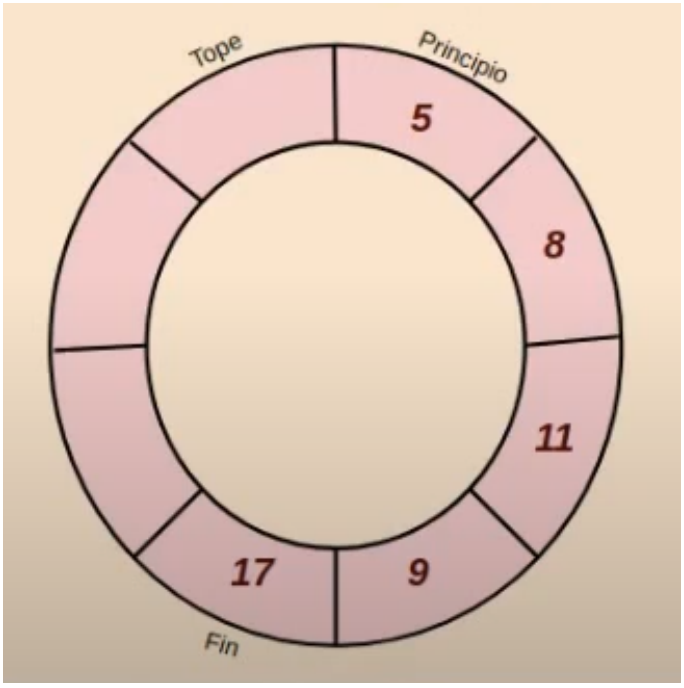

Figura 4: Cola circular.