Informe TDA ~ Lista y extras

Padrón: 108951

Nombre: Luciano Andrés Lorenzo

TDA Lista

Una lista es un **conjunto de elementos**, donde cada elemento tiene **sucesor** (menos el último) y un **predecesor** (menos el primero).

En este trabajo fue implementada con **nodos** enlazados. Un nodo es una estructura que contiene un puntero a otro nodo y un puntero al elemento que guarda. El struct `lista_t` está compuesto por una referencia al nodo final, una al inicial y un contador con la cantidad de elementos.

Complejidad operaciones:

- `Crear`: **O(1)**
  - Devuelve la instancia de pila.
- `Insertar`: **O(N)**
  - Recorre los N elementos hasta llegar a la posición y allí agrega el nodo.
  - Aunque tiene 2 casos especiales de tiempo constante **O(1)**, agregar al principio y al final, ya que al tener las referencias a ambos nodos se puede agregar uno nuevo sin la necesidad de recorrer la lista.
- `Quitar`: **O(N)**
  - Recorre los N elementos hasta encontrar el anterior al que busca y le borra el siguiente (el que busco).
  - Este tiene solo un caso **O(1)**, y es cuando tengo que sacar el primero. Ya que cuando quiero sacar el último
- `Destruir`: **O(N)**
  - Hay que recorrer los N nodos de la lista para destruirlos a todos.
- `Iterador interno`: **O(N)**
  - Recorre y llama a la función como mucho a todos los elementos (puede parar antes), pero el peor caso es que tenga que recorrer todos, por lo que es **O(N)**.

Ya que las funciones quitar e insertar se encargan de insertar y quitar al final de la lista, decidí que solo se encargaran de llamar a la funcion `quitar_de_posicion` y `insertar_en_posicion` con la posicion final (de esta manera no hay codigo repetido).

TDA Pila

Es una estructura de un conjunto de elementos que imita objetos apilados. La cual sigue `LIFO` (*Last in first out*), el último que entra es el primero en salir.

Lo implemento en el TP con nodos enlazados, utilizando la implementacion de listas, donde el tope es el nodo_inicio, y la base el nodo_fin. Sobre la misma solo se puede insertar y quitar al principio de la lista, de esta manera ambas operaciones son de tiempo constante.

El usuario que utilice esta implementación en ningún momento se entera que es en verdad una lista enlazada, ya que toda función de la pila trabaja 'externamente' con un struct `pila_t`, pero en verdad para hacer todas las funcionalidades es casteado a `lista_t`.

Operaciones:

- `Crear`: **O(1)**
  - Devuelve la instancia de pila.
- `Insertar` (apilar): **O(1)**
  - Es un caso especial de insertar en lista, inserta en el primer nodo de la lista (que representa el tope)
- `Quitar` (desapilar): **O(1)**
  - Al igual que insertar, es un caso especial de quitar en lista, quita el primer nodo de la lista. Por lo que no hay que recorrer.
- `Destruir`: **O(N)**
  - Hay que recorrer los N nodos de la lista para destruirlos a todos.
- `Ver tope` : **O(1)**
  - Devolver el elemento del primer nodo. No hay que recorrer nodos, siempre es el primero.
- `Vacia`: **O(1)**
  - Comprueba si el primer nodo existe (el tope), por lo que no tiene que recorrer nodos.

TDA Cola

Es una estructura de un conjunto de elementos que imita objetos en cola. La cual sigue `FIFO` (*first in first out*), el primero que entra es el primero en salir.

Lo implemento en el TP con nodos enlazados, utilizando la implementacion de listas, donde el nodo inicio es el frente de la cola (donde desencolo) y el nodo fin es el final (donde encolo). Pero sobre la misma solo se puede insertar al final y quitar al principio de la lista, de esta manera ambas operaciones son de tiempo constante.

Al igual que con la pila, la cola trabaja 'externamente' con un struct `cola_t`, pero en verdad para hacer todas las funcionalidades es casteado a `lista_t`.

Operaciones:

- `Crear`, `Insertar` (encolar), `Quitar` (desencolar), `Destruir`, `vacía`:
  - Tienen la misma complejidad que las pilas por las mismas razones, siempre tengo referencias para donde quitar y donde insertar.
- `ver frente`: **O(1)**

- Al tener el frente en el primer elemento, no hay que recorrer más nodos.

Para implementar una cola con un **array estático** puedo pensarlo como un array circular, donde tengo referencias al frente y al final de la cola. La principal desventaja con respecto a la implementación con nodos enlazados, es que el array tiene un tamaño fijo por lo que no puede ser mayor a una cierta cantidad de elementos. Al encolar y desencolar elementos voy moviendo ambas referencias, cuando el final de la cola llega al último lugar del array (si es que ya se desencolo un elemento), paso la referencia del final al primer lugar del array. La cola esta llena cuando vuelven a encontrarse el final y el frente.