FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

TDA n1 — Listas

[7541/9515] Algoritmos y Programación II

Segundo cuatrimestre de 2021

Alumno:	Vallejos, Matias
Número de padrón:	107924
Email:	matiavallejo@gmail.com
Email Institucional:	mvallejos@fi.uba.ar

1. Introducción

El objetivo de este TDA es implementar una solución simple al problema de agrupar elementos de diferentes datos, esto se soluciona mediante las listas, en este caso utilizando nodos enlazados, estas son estructuras dinámicas las cuales cada nodo contiene información sobre el elemento y el siguiente nodo. Mediante la implementación de la lista se nos simplifica la creación de colas y pilas.

2. Teoría

Respuestas a las preguntas teóricas

1. Se busco la creación de listas mediante nodos simplemente enlazados (Ver diagrama 1) se utilizó esta manera ya que si bien se puede hacer con vectores estáticos o dinámicos al realizarlas con nodos simples enlazados reducimos la complejidad del programa. En este proyecto contamos con 3 estructuras principales:

- **Lista** la cual se encarga de administrar toda la lista, su objetivo principal es saber cuántos elementos hay en total y donde está el principio y el fin de esta.
- **Nodo** Son los nodos que van a estar dentro de la lista ya que utilizamos el método de simplemente enlazados estos solo cuentan con un puntero hacia el elemento que vamos a insertar en el nodo y un puntero hacia el nodo siguiente

- **Lista Iterador** el objetivo de esta estructura como dice su nombre es ir iterando en la lista desde el principio hacia el fin o si es necesario utilizar una condición de corte el cual podría ser una función para que detenga su iteración. En este proyecto lo utilizamos tanto para la implementación de iterador externo como iterador interno.

2. Una **lista** es una agrupación de elementos de cualquier tipo de dato relacionados entre sí, cada lista tiene un sucesor y predecesor hay diferentes tipos de implementaciones para las listas como los vectores estáticos, vectores dinámicos y lista de nodos, en este caso utilizaremos lista de nodos debido a complejidad reducida, a su vez hay diferentes tipos de listas de nodos, entre ellas están:

- **Listas simplemente enlazadas** Es la que utilizamos en este proyecto, en este formato los nodos solo conocen al siguiente elemento. (Ver diagrama 1)
- **Listas doblemente enlazadas** En este formato los nodos contienen información tanto del siguiente elemento como el anterior.
- **Listas circulares simplemente enlazadas** En este formato el ultimo nodo de la lista tiene referencia hacia el primer nodo, es por eso por lo que se dice que son "Circulares" ya que parece que no tienen fin.
- **Listas circulares doblemente enlazadas** En este formato el ultimo nodo de la lista tiene referencia hacia el primer nodo como en las circulares simplemente enlazadas pero con la diferencia de que aquí el primer nodo tiene referencia hacia el ultimo.

> Para entender el metodo de insertar y remover de una fila doblemente enlazada esta el diagrama 1 y 2

3. Una **pila** es una estructura de datos que agrupa elementos, muy similar a las listas, pero con una diferencia mayor que estas cuentan con una estructura de tipo "Last in, First out"(LIFO) que traducido al español seria algo como "lo último que se apilo es lo primero que se usa". Al igual que las listas estas pueden implementarse con vectores estáticos, vectores dinámicos y lista de nodos, en este caso utilizaremos nuevamente la lista de nodos por su complejidad. También al igual que las listas tienen diferentes tipos de listas de nodos, pero utilizaremos la simplemente enlazadas. A diferencia de las listas en las pilas no se puede realizar cualquier tipo de operación, sino que para insertar y remover una pila siempre utilizamos el último elemento de la lista, o sea no podes insertar un elemento entre medio de la pila ni removerlo, en este proyecto cuenta con funciones para ver cuál es el último elemento, ver si esta vacía la pila, ver cuántos elementos contiene, etc.

> Para entender el metodo de insertar y remover de una pila esta el diagrama 3 y 4

4. Una **cola** es una estructura bastante parecida a la pila y también similar a las listas, pero con la diferencia de que cuentan con un tipo "First in, First out"(FIFO) que traducido al español seria algo como "lo primero que se encolo es lo primero que se usa". Se pueden implementar con vectores estáticos, vectores dinámicos y finalmente lista de nodos el cual utilizaremos debido a su complejidad. Las similitudes que tiene con la pila es que al igual que ellas no podemos realizar cualquier operación como en una lista sino que tenemos operaciones definidas para

seguir, a diferencia de las pilas estas operaciones son encolar y desencolar, la cual encolar se encarga de poner un elemento al final de la lista y desencolar se encarga de remover un elemento pero del principio de la lista, en este proyecto contamos con otras operaciones como ver cuál es el elemento al frente de la cola, o sea el primero de la lista, el tamaño, ver si esta vacia,etc.

> Para entender el metodo de insertar y remvoer de una cola esta el diagrama 5 y 6

3. Detalles de implementación

Detalles específicos de la implementación, como compilar, como ejecutar

1. `lista_crear`

   Reserva espacio en el heap para almacenar elementos de tipo lista_t y devuelve un puntero, esto se logra con calloc inicializando a los elementos en 0 o NULL. Devuelve la lista o NULL en caso de error.

2. `nodo_crear`

   Reserva espacio en el heap para almacenar elementos de tipo nodo_t y devuelve un puntero, esto lo logra con callloc inicializando todos los elementos en 0 o NULL. Devuelve el nodo o NULL en caso de error.

3. `lista_insertar`

   Inserta un elemento al final de la lista, tiene en cuenta si la lista está vacía así en vez de cambiar el nodo final cambia el nodo de inicio. Devuelve la lista o NULL en caso de error.

4. `posicion_nodo_anterior`

   Función recursiva para encontrar al nodo en la posición anterior. La utilizo en las funciones de insertar_en_posicion para insertar un elemento que no sea en el final o principio, en lista_quitar y lista_quitar_de_posicion para eliminar un elemento al final de la lista. Esta función devuelve un puntero de tipo nodo_t hacia el nodo anterior.

5. `lista_insertar_en_posicion`

   Inserta un elemento en una posición dada, lo hace mediante la reutilización de lista_insertar para insertar al inicio y posicion_nodo_anterior para crear un nuevo nodo y actualizar las referencias del nodo anterior. Si la posición dada es mayor al tamaño de la lista esta lo inserta al final de la lista. Devuelve la lista o NULL en caso de error.

6. `lista_quitar`

   Borra el último de la lista. Si hay un solo elemento comprueba que este se liberó. Devuelve el elemento removido o NULL en caso de error.

7. `lista_quitar_primer_elemento`

   Borra el primer elemento de la lista y actualiza las referencias. Este es utilizado en la cola. Devuelve el elemento removido o NULL en caso de error.

8. `lista_quitar_de_posicion`

   Utiliza las funciones de lista_quitar y lista_quitar_primer_elemento para los casos de remover al final o principio de la lista, si la posición dada es mayor al tamaño de la lista remueve el último elemento de la lista. En caso de querer eliminar un elemento del medio utiliza la función de posicion_nodo_anterior para actualizar las referencias y finalmente eliminar el nodo en la posición solicitada. Devuelve el elemento del nodo eliminado o NULL en caso de error. (Ver diagrama 2)

9. `lista_elemento_en_posicion`

   Devuelve el elemento de la posición indicada reutiliza las funciones lista_primero y lista_ultimo para devolver el elemento en el último nodo o primer nodo, si la posición es mayor al tamaño de la lista o la lista esta vacía devuelve NULL.

10. `lista_primero`

    Devuelve el primer elemento de la lista o NULL si esta vacía.

11. `lista_ultimo`

    Devuelve el último elemento de la lista o NULL si esta vacía.

12. `lista_vacia`

    Chequea si el tamaño de la lista es igual a 0, si lo es devuelve true sino false.

13. `lista_tamanio`

    Devuelve el tamaño de la lista el cual se actualiza en el struct de lista el elemento cantidad.

14. `lista_destruir`

    Libera todos los elementos de la lista con la función lista_quitar y al final libera la lista.

15. `lista_iterador_crear`

    Reserva espacio en el heap para almacenar un elemento de tipo lista_iterador_t y devuelve un puntero, lo realiza mediante el uso de calloc para iniciar todos los elementos en NULL. Finalmente apunta a la lista y a su primer elemento.

16. `lista_iterador_tiene_siguiente`

Si el elemento actual no es nulo significa que tiene un siguiente y puedo seguir avanzando. Retorna true si hay un siguiente y puede seguir avanzando o false si no puede seguir avanzando.

17. `lista_iterador_avanzar`

    Utiliza la función lista_iterador_tiene_siguiente para saber si puede avanzar, si esta le devuelve true avanza hacia el siguiente elemento.

18. `lista_iterador_elemento_actual`

    Devuelve el elemento actual de la lista o NULL en caso de error.

19. `lista_iterador_destruir`

    Libera el iterador

20. `recorrer_iterador_interno`

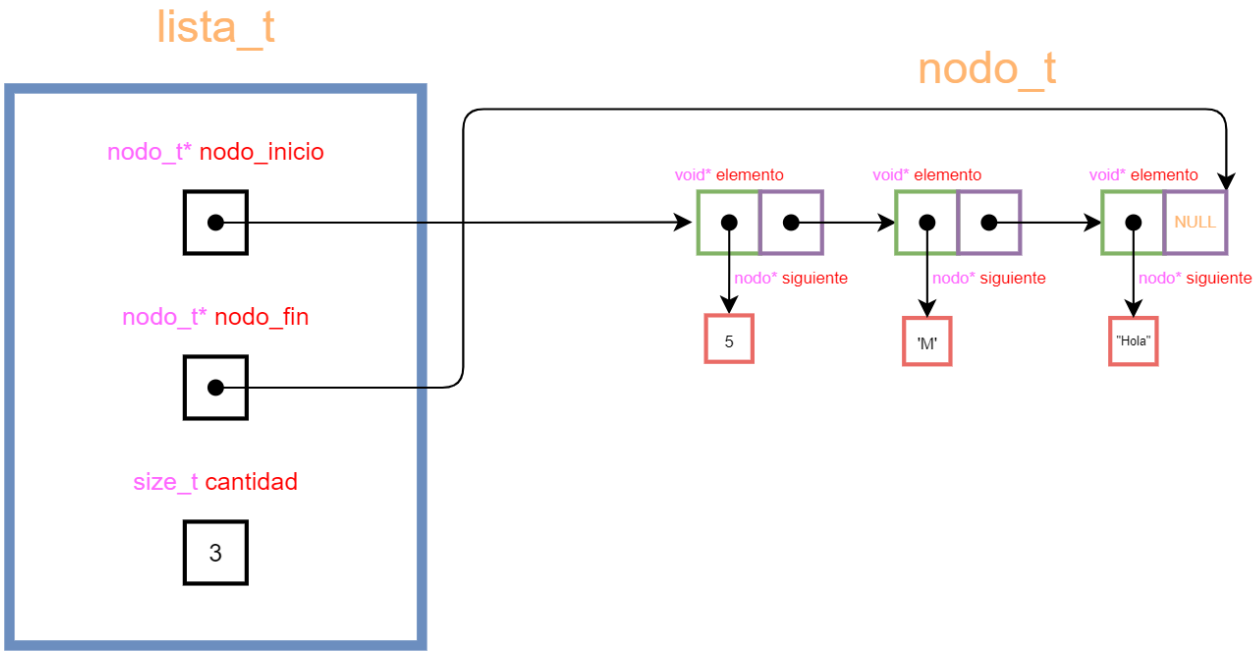
    Función recursiva es una función auxiliar para lista_con_cada_elemento, recorre la lista y va contando cuantos elementos lee, si llega al final de la lista o luego de evaluar el nodo en la función recibida por parámetro da false detiene el recorrido y devuelve la cantidad de elementos que recorrió.

21. `lista_con_cada_elemento`

    Se utiliza para el iterador interno, utiliza la función recorrer_iterador_interno para recorrer la lista hasta donde deba y devuelve la cantidad de elementos recorridos. Si algunos de los parámetros recibidos son NULL devuelve 0.

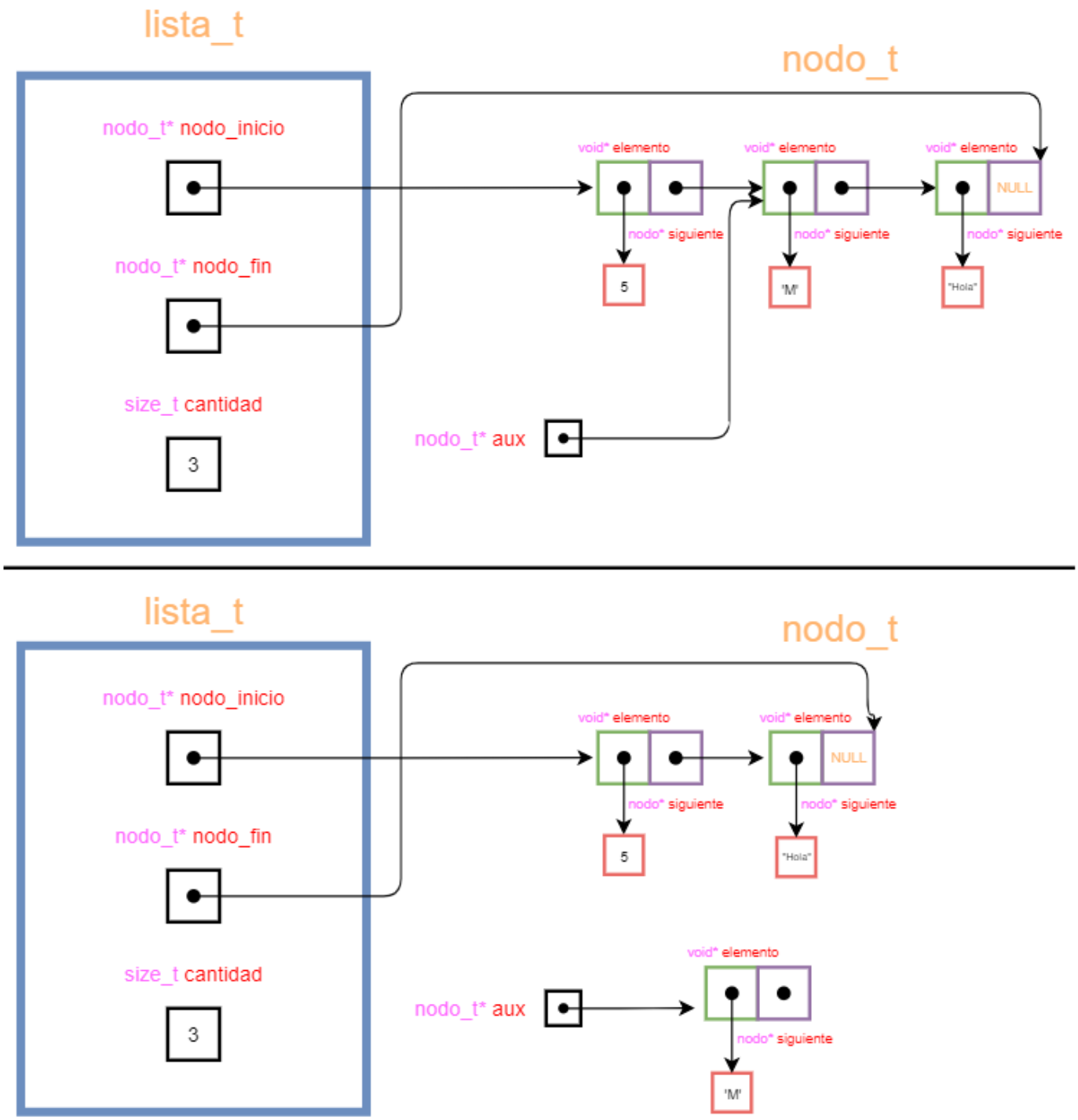
4. Diagramas

1. Diagrama1



En este caso podemos ver una lista de nodos simplemente enlazados con 3 elementos, nodo_inicio apunta al primer nodo, nodo_fin apunta al último nodo y en el último nodo como no hay siguiente el nodo* siguiente apunta a NULL

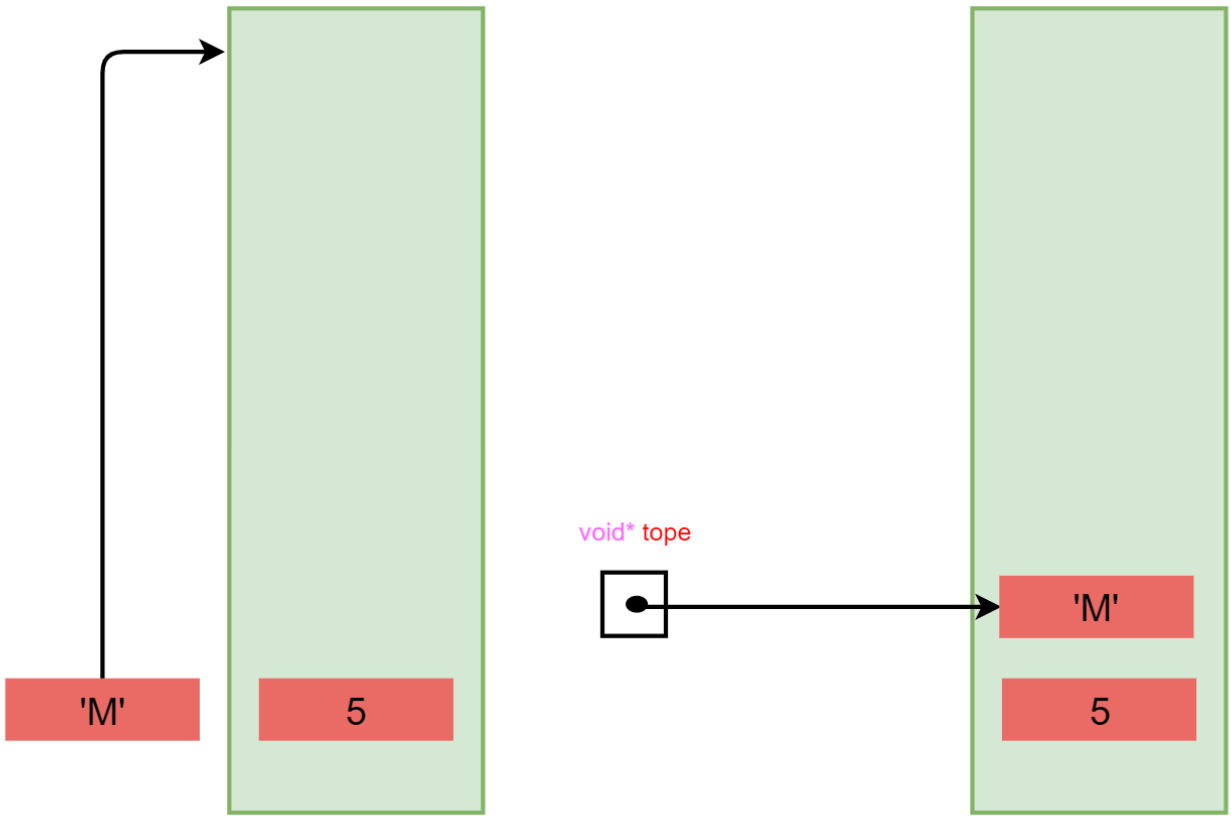
2. Diagrama2



En este caso podemos ver como se elimina un elemento de una lista, los pasos son los siguientes:

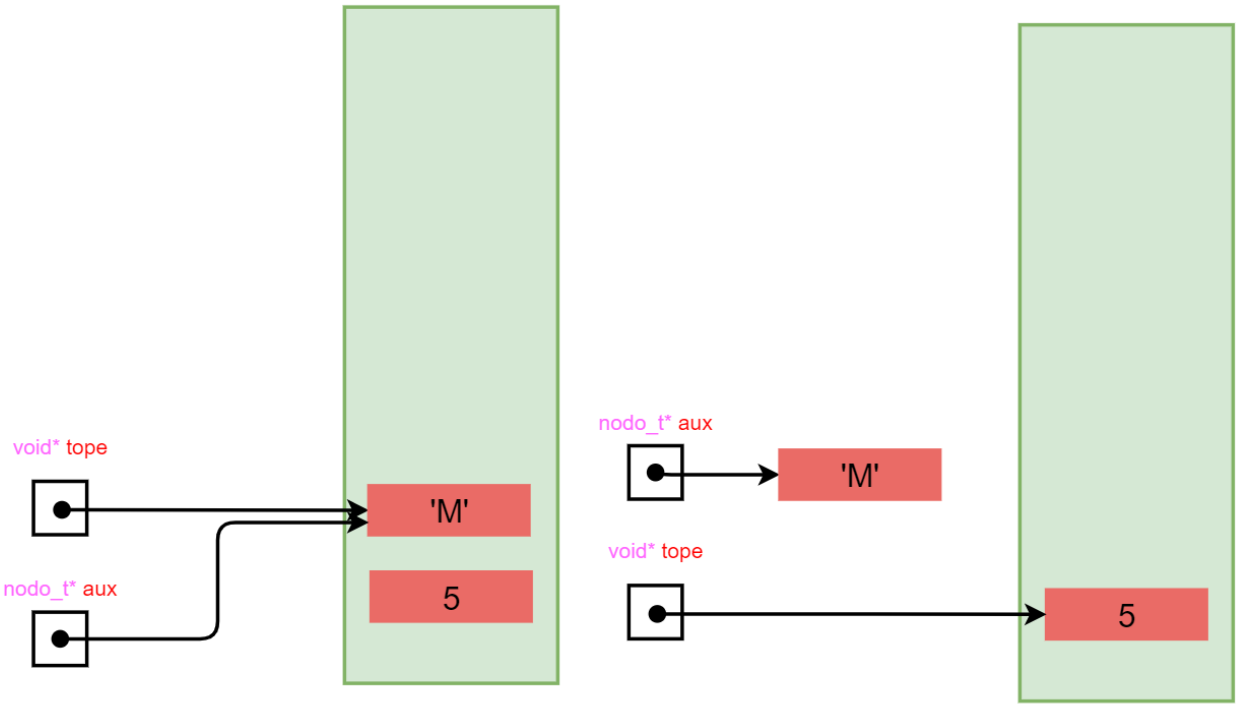
i. Apunto con el auxiliar al elemento que quiero remover
ii. Apunto el nodo anterior al que quiero mover al nodo siguiente.
iii. Borro el auxiliar para liberar el nodo que ya no quiero utilizar.

3. Diagrama3



En este caso podemos ver como se ingresa un elemento en una pila, este se cola al final y el tope apunta hacia este ultimo elemento

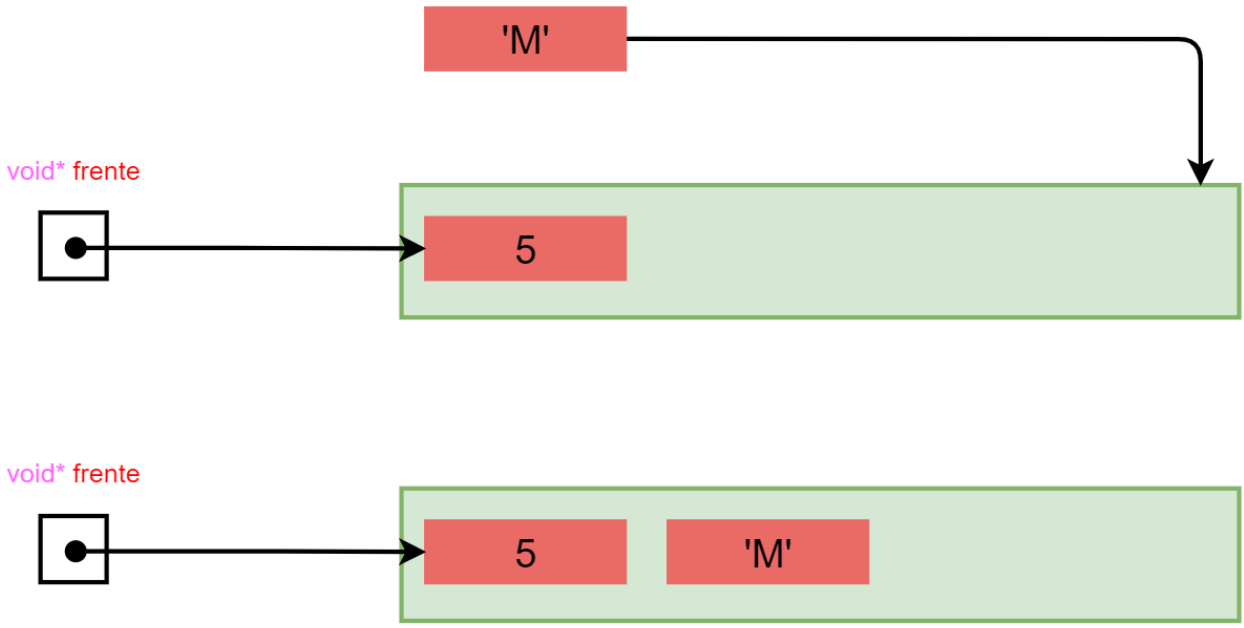
4. Diagrama4



En este caso podemos ver como se remueve un elemento de una pila, los pasos son los siguientes:

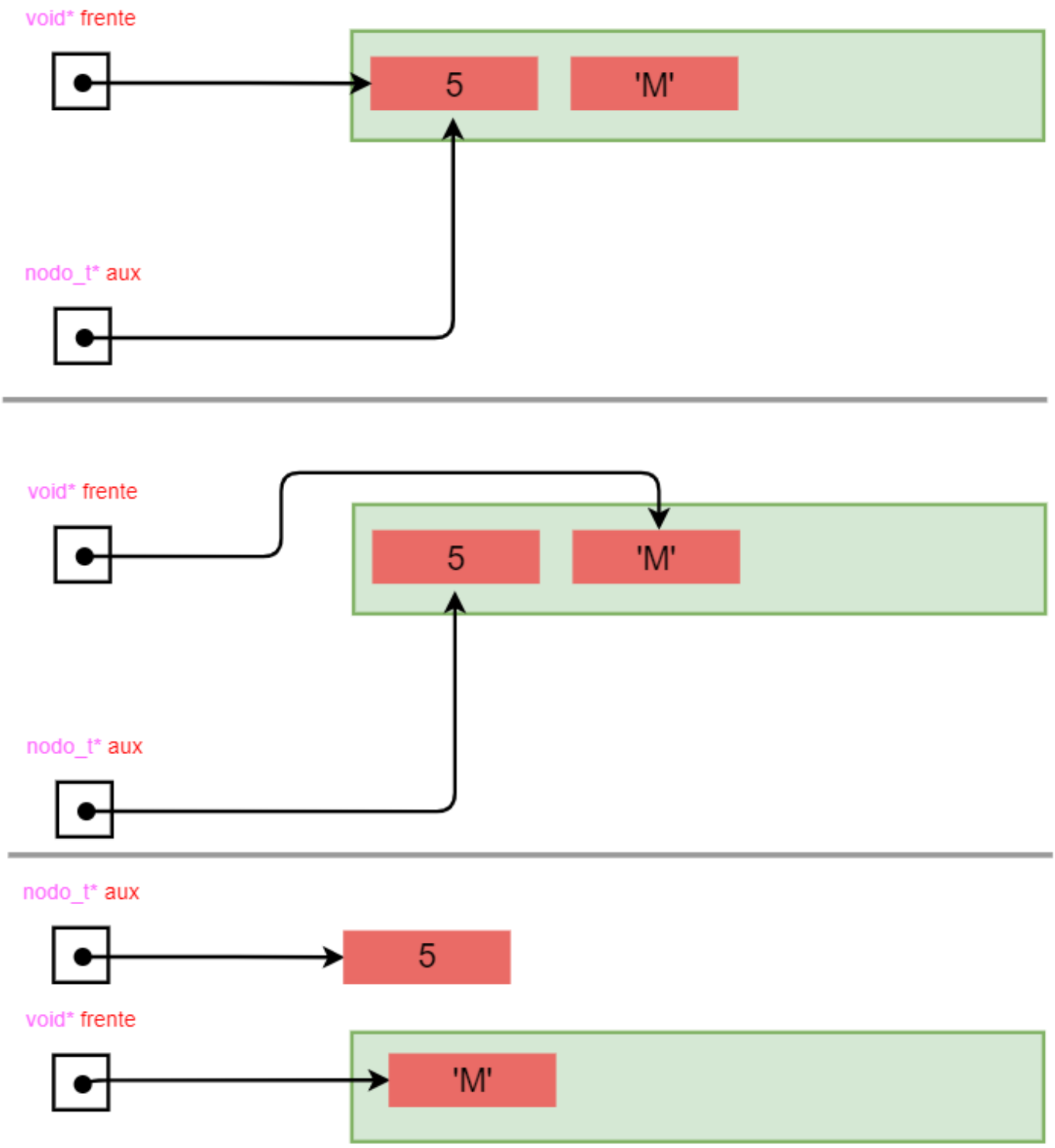
i. Apunto con un auxiliar al ultimo elemento de la pila ya que es el unico que puedo remover (El tope)
ii. Cambio hacia donde apunta el tope ya que como este no va a ser mas el ultimo elemento lo apunto al anterior, osea el 5
iii. Borro el auxiliar y libero el nodo.

5. Diagrama5



En este caso podemos ver como se inserta un elemento en una cola, podemos desctacar que el frente no cambia.

6. Diagrama6



En este caso podemos ver como se remueve un elemento de una cola, los pasos son los siguientes:

i. Apunto el auxiliar al primer elemento de la cola ya que es el unico que puedo remover (El frente)
ii. Cambio hacia adonde apunta el frente ya que este no va a ser mas el primer elemento, lo apunto al siguiente
iii. Borro el auxiliar y libero el nodo