Algoritmos y estructuras de datos

Estructuras Lineales

CEIS

Escuela Colombiana de Ingeniería

2021-2

Agenda

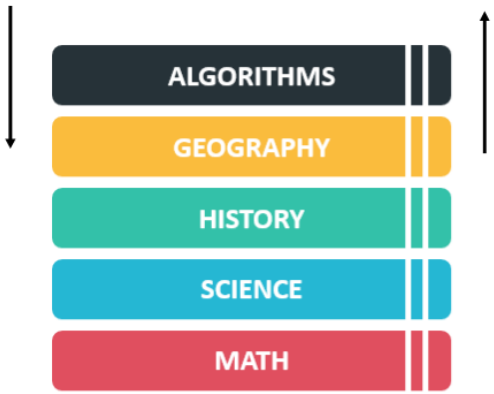
1. Pilas y Colas
   - Pilas
   - Colas
2. Listas Encadenadas
   - Centinela
3. Aspectos finales
   - Ejercicios

Agenda

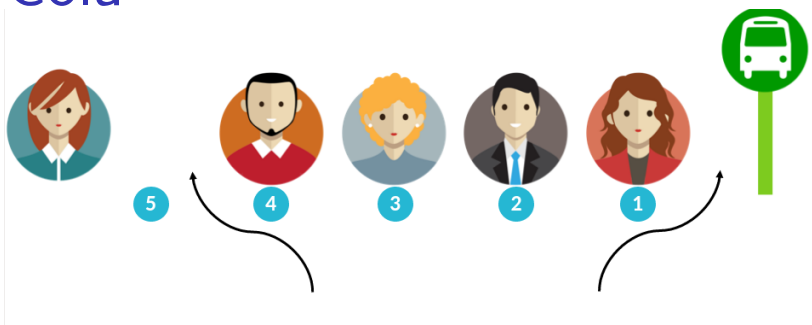
Pilas y Colas

Las pilas y colas son estructuras lineales dinámicas en los cuales el elemento que se removerá por la operación de `DELETE` está predefinido.

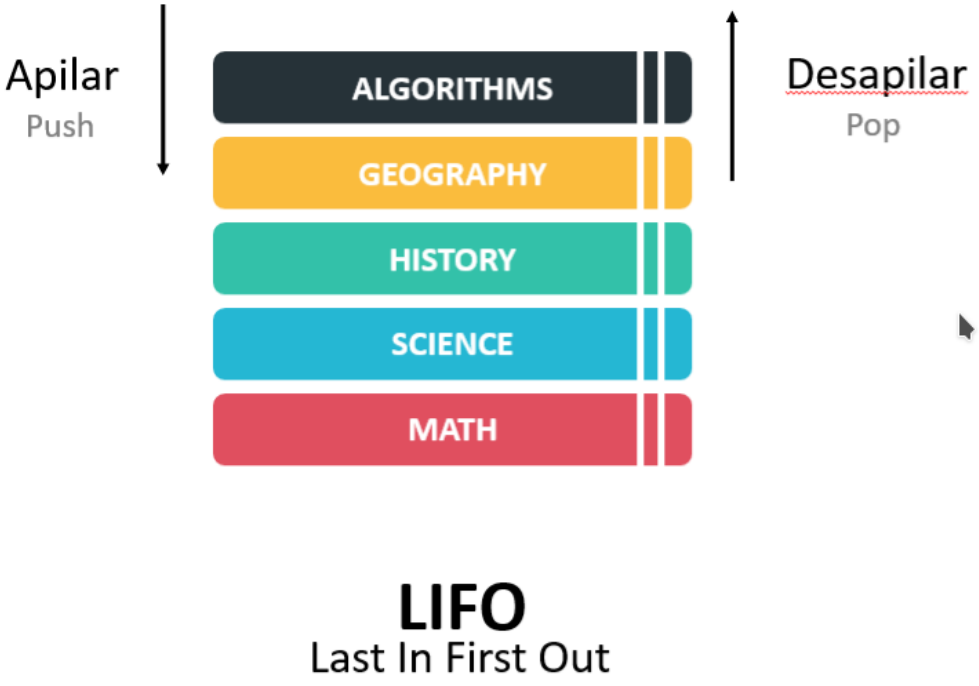
Pila



Cola

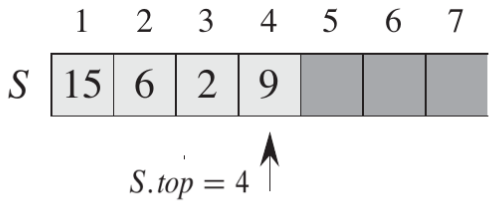


Pilas



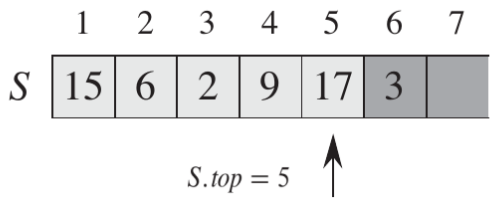
- En una pila (*stack*) el elemento eliminado es el más reciente en ser insertado: implementa una política LIFO.
- La operación de inserción se denomina `PUSH` y la operación de eliminación se denomina `POP`.

Pilas

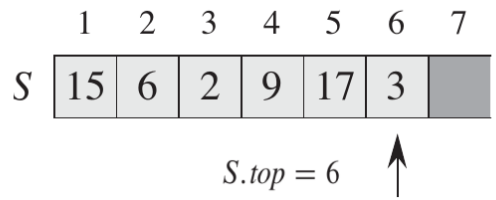


PUSH$(S, 17)$

PUSH$(S, 3)$

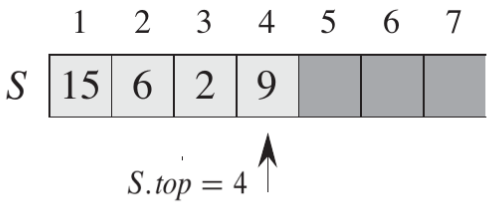


POP(S)

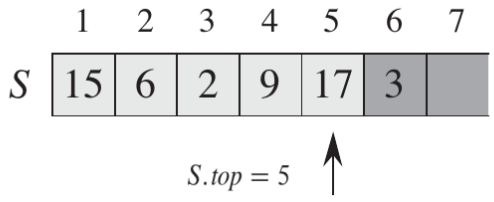


- Se puede implementar un *stack* de hasta n elementos con un arreglo S[1..n]. El arreglo tiene un atributo S.top que da el índice del último elemento insertado.
- Cuando S.top $= 0$, la pila no contiene elementos y se dice que está vacia. El stack consiste de elementos S[1..S.top]

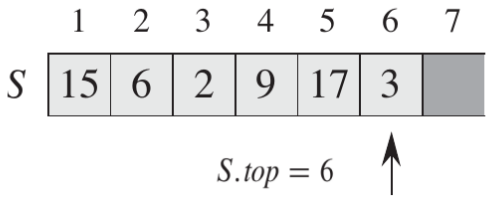
Pilas



PUSH$(S, 17)$
PUSH$(S, 3)$



POP(S)

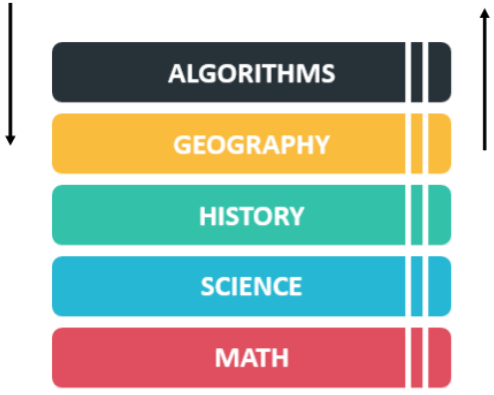



```
STACK-EMPTY(S)
1  if S.top == 0
2       return TRUE
3  else return FALSE
```

```
PUSH(S, x)
1  S.top = S.top + 1
2  S[S.top] = x
```

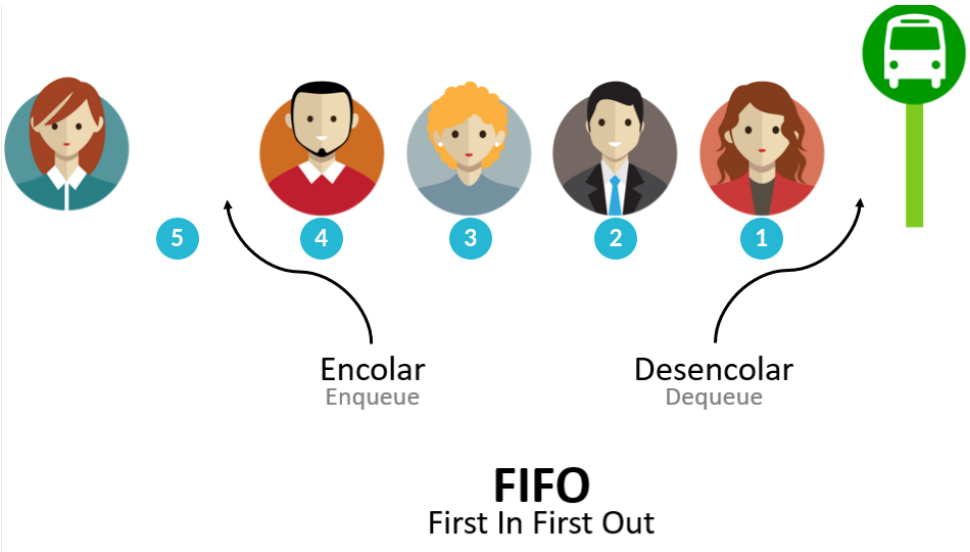
```
POP(S)
1  if STACK-EMPTY(S)
2       error "underflow"
3  else S.top = S.top − 1
4       return S[S.top + 1]
```

Pilas



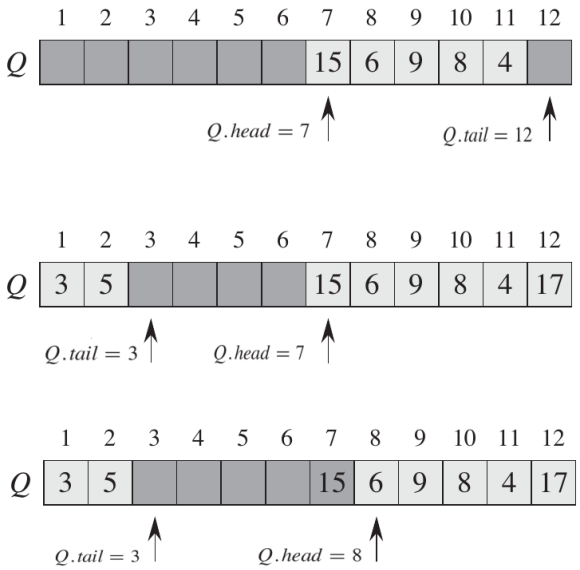

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack
[3, 4, 5, 6]
>>> stack.pop()
6
>>> stack
[3, 4, 5]
```

Cola



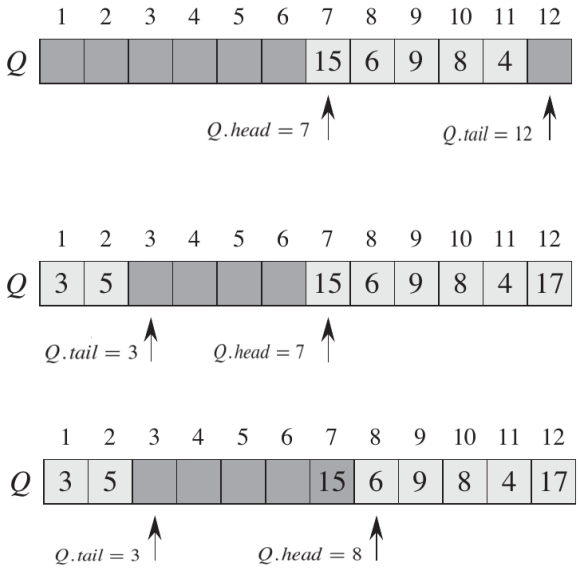
- En una cola (*queue*) el elemento eliminado es el que lleva más tiempo en ella: implementa una una política FIFO.
- La operación de inserción se denomina `ENQUEUE` y la operación de eliminación se denomina `ENQUEUE`.

Colas



- La cola tiene un head y un tail. Cuando un elemento es encolado, este toma su posición en el tail de la cola. El elemento decolado, siempre es el elemento en la cabeza(head) de la cola.
- La cola tiene un atributo Q.head que apunta a su cabeza. El atributo Q.tail, apunta al lugar donde se almacenará un nuevo elemento. Cuando Q.head = Q.tail, la cola esta vacía. Si, Q.head = Q.tail + 1, la cola esta totalmente llena.

Colas



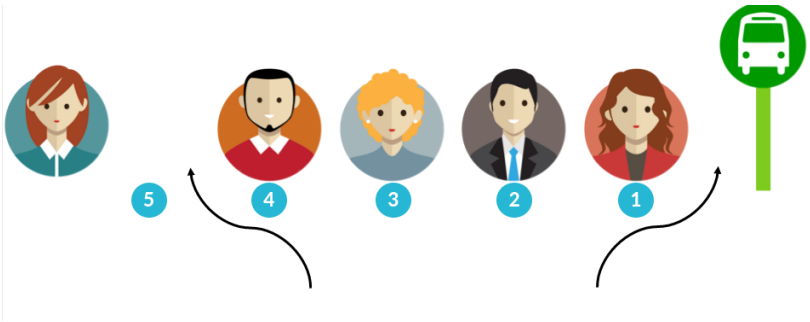
ENQUEUE(Q, x)

```
1  Q[Q.tail] = x
2  if Q.tail == Q.length
3        Q.tail = 1
4  else Q.tail = Q.tail + 1
```

DEQUEUE(Q)

```
1  x = Q[Q.head]
2  if Q.head == Q.length
3        Q.head = 1
4  else Q.head = Q.head + 1
5  return x
```

Colas



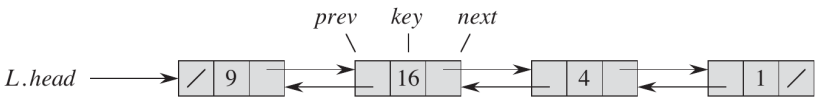

```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")
>>> queue.append("Graham")
>>> queue.popleft()
'Eric'
>>> queue.popleft()
'John'
>>> queue
deque(['Michael', 'Terry', 'Graham'])
```

Agenda

Listas Encadenadas

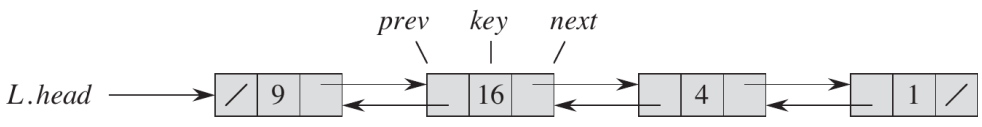
- Una lista encadenada es una estructura en donde los objetos están arreglados de una manera lineal.
- A diferencia un arreglo, en el cuál el orden lineal esta determinado por los índices del arreglo, el orden de una lista encadenada esta determinado por un apuntador en cada objeto.

Listas Encadenadas



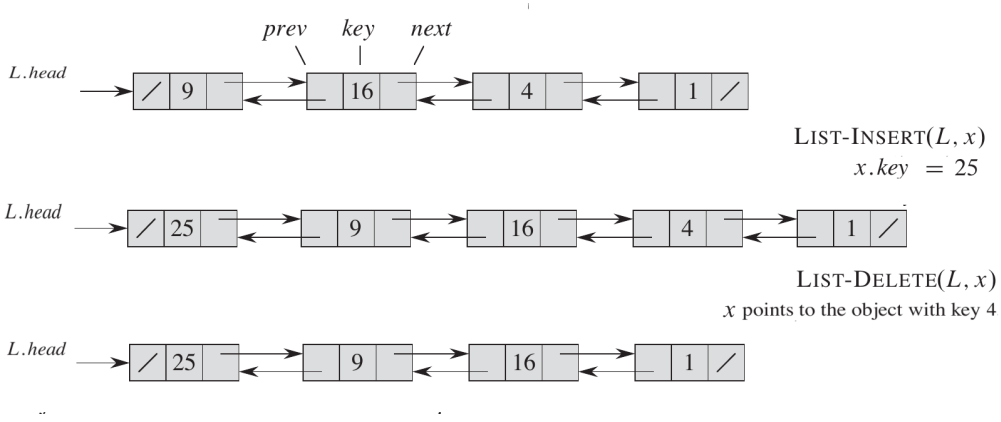
- Cada elemento en una lista L doblemente encadenada es un objeto con un atributo key y dos atributos apuntadores: next y prev.
- Dado un elemento x en la lista, x.next apunta al sucesor en la lista, y x.prev apunta a su predecesor.
- Si x.prev = NIL, el elemento x no tiene predecesor y es el primer elemento. Si x.next = NIL, el elemento x no tiene sucesor y es por tanto el último elemento.
- Un atributo L.head apunta al primer elemento de la lista. Si L.head = NIL, la lista es vacia.

Listas Encadenadas



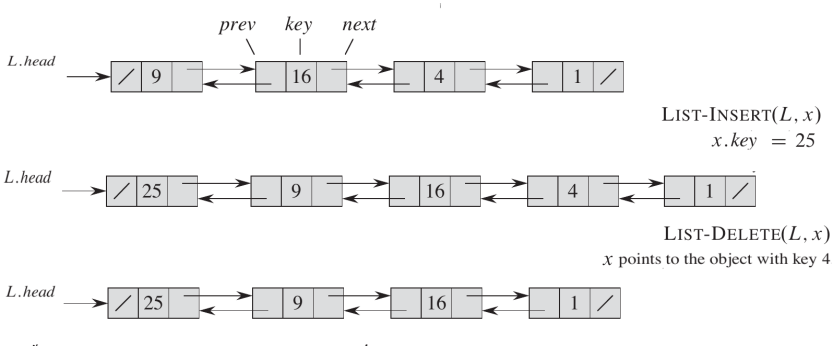
- Cada elemento en una lista L doblemente encadenada es un objeto con un atributo key y dos atributos apuntadores: next y prev.
- Dado un elemento x en la lista, x.next apunta al sucesor en la lista, y x.prev apunta a su predecesor.
- Si x.prev = NIL, el elemento x no tiene predecesor y es el primer elemento. Si x.next = NIL, el elemento x no tiene sucesor y es por tanto el último elemento.
- Un atributo L.head apunta al primer elemento de la lista. Si L.head = NIL, la lista es vacia.

Listas Encadenadas



- Dado un elemento x cuyo atributo llave ya ha sido definido, el procedimiento LIST-INSERT ubica a x al frente de la lista encadenada.
- El procedimiento LIST-DELETE remueve un elemento x de una lista encadenada L. El recibe un apuntador a x, y debe borrar a x al actualizar los punteros.

Listas Encadenadas



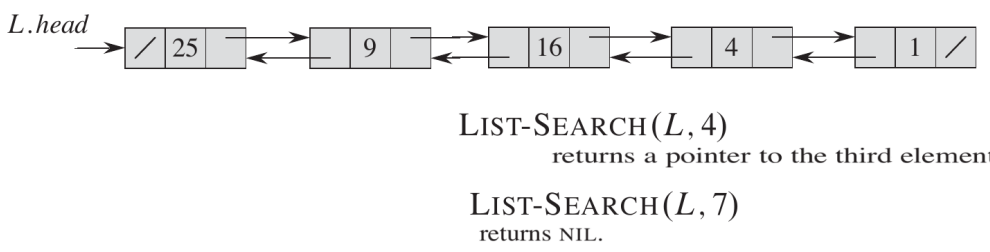
LIST-INSERT(L, x)

```
1  x.next = L.head
2  if L.head ≠ NIL
3        L.head.prev = x
4  L.head = x
5  x.prev = NIL
```

LIST-DELETE(L, x)

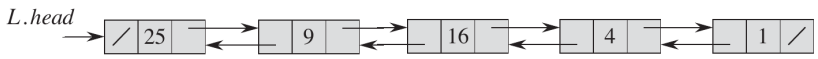
```
1  if x.prev ≠ NIL
2        x.prev.next = x.next
3  else L.head = x.next
4  if x.next ≠ NIL
5        x.next.prev = x.prev
```

Listas Encadenadas



- La función LIST-SEARCH(L, k) encuentra el primer elemento con una llave k en una lista L, usando una búsqueda lineal. Si no se encuentra un objeto con llave k en la lista, el procedimiento retorna NIL.

Listas Encadenadas



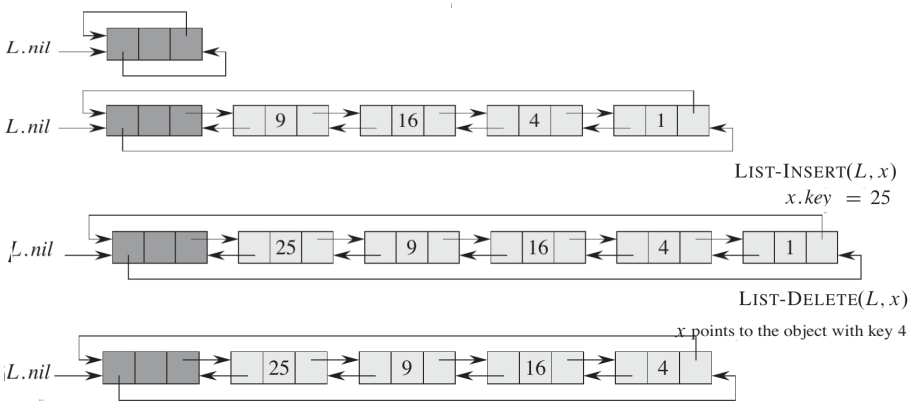
LIST-SEARCH$(L, 4)$
returns a pointer to the third element

LIST-SEARCH$(L, 7)$
returns NIL.

LIST-SEARCH(L, k)

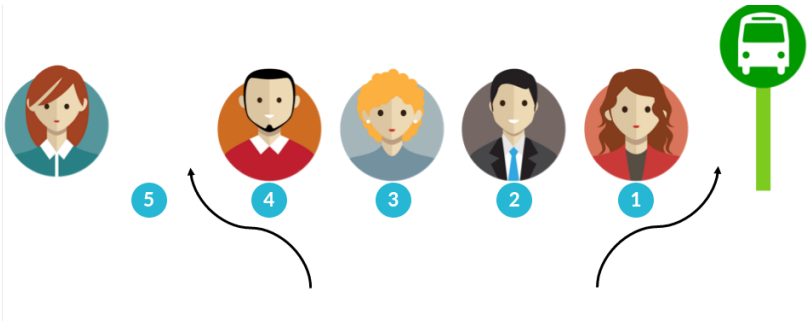
```
1  x = L.head
2  while x ≠ NIL and x.key ≠ k
3        x = x.next
4  return x
```

Listas Encadenadas-Centinela



- Un centinela es un objeto que permite simplificar las condiciones de borde.
- Supongamos, que se provee a la lista L un objeto L.nil que representa NIL pero tiene todos los atributos de los demás objetos de la lista. Cada vez que se tiene una referencia a NIL, se reemplaza por una referencia al centinela L.nil.
- El atributo L.nil.next apunta al head de la lista, y L.nil.prev apunta al tail de la lista.

Colas




```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")
>>> queue.append("Graham")
>>> queue.popleft()
'Eric'
>>> queue.popleft()
'John'
>>> queue
deque(['Michael', 'Terry', 'Graham'])
```

Agenda

Ejercicios

1. Considere una pila *S* y una cola *Q*, ambas vacías en un principio. Ilustre el paso a paso de las siguientes operaciones:
   - $S.push(2) \rightarrow S.push(8) \rightarrow S.push(11) \rightarrow S.pop() \rightarrow S.push(-3) \rightarrow S.push(7) \rightarrow S.pop() \rightarrow S.pop()$
   - $Q.enqueue(4) \rightarrow Q.enqueue(17) \rightarrow Q.enqueue(20) \rightarrow Q.enqueue(6) \rightarrow Q.dequeue(\square) \rightarrow Q.dequeue(\square) \rightarrow Q.enqueue(-5) \rightarrow Q.dequeue(\square)$
2. Desarrolle un algoritmo que identifique si una cadena de texto contiene una lista de paréntesis correctamente anidados y balanceados, por ejemplo:
   - Correcto: [{((())())()}]{[]}
   - Correcto: (({}))[]
   - Incorrecto:)([])(
   - Incorrecto: (){[]([]

   Bonus: identifique la posición del primer paréntesis mal ubicado en caso de que los paréntesis no estén correctamente anidados y balanceados.
3. Diseñe una función para invertir la dirección de una lista enlazada simple, es decir, una función que invierta todos los punteros entre los elementos de la lista. El algoritmo debe tener complejidad lineal $O(n)$
4. Modifique el código de la lista enlazada para que sea una doble lista enlazada, e implemente la siguientes funciones:
   - Insertar un nuevo elemento
   - Eliminar un elemento dado su valor (considere el caso de borrar la cabeza de la lista)
   - Eliminar los elementos duplicados
   - Unir dos listas

Ejercicios

10-1 ***Comparisons among lists***

For each of the four types of lists in the following table, what is the asymptotic worst-case running time for each dynamic-set operation listed?

	unsorted, singly linked	sorted, singly linked	unsorted, doubly linked	sorted, doubly linked
SEARCH(L, k)				
INSERT(L, x)				
DELETE(L, x)				
SUCCESSOR(L, x)				
PREDECESSOR(L, x)				
MINIMUM(L)				
MAXIMUM(L)				