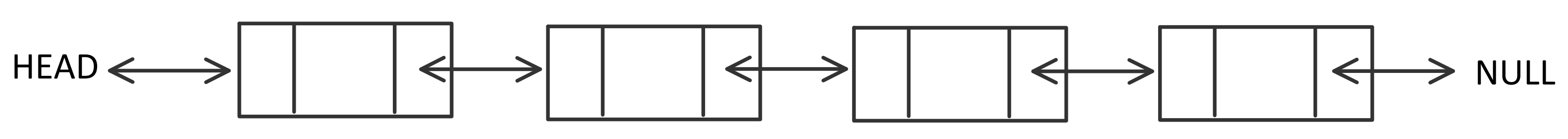
Doubly linkedlist

Tuesday, May 3, 2022 7:12 PM

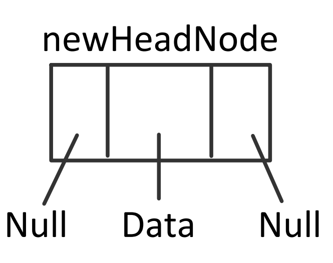
Doubly linkedlist - Data structure & algorithms



Initial state: headPointer = null
tailPointer = null

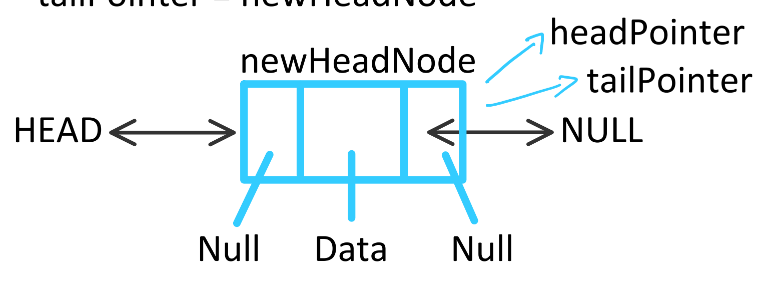
InsertAtHead(int data)

- Check if the doubly linkedlist is uninitialized/null. We can check this by verifying head == null.
- If the headPointer is null, we will need to create a new doubly linkedlist node and set it as the new head node.



Set newHeadNode.data = data
Set newHeadNode.next = null
SetNewHeadNode.prev = null

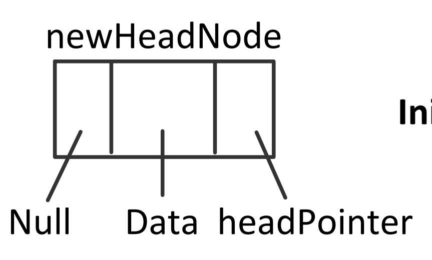
Initial state: headPointer = null,
tailPointer = null,
HEAD <—> NULL
Final state: headPointer = newHeadNode,
tailPointer = newHeadNode



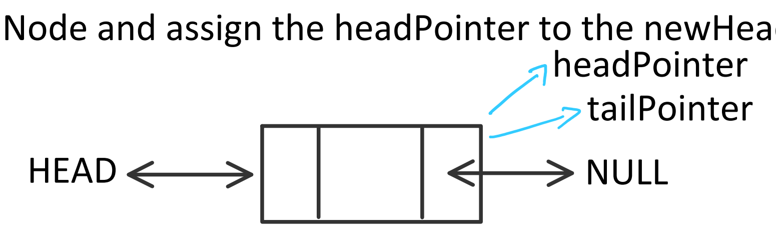
Time complexity:

- **InsertAtHead() -** O(1) constant-time
- **RemoveFromHead() -** O(1) constant-time
- **InsertAtTail() -** O(1) constant-time
- **RemoveFromTail() -** O(1) constant-time

- If the headPointer is not null, we still need to create a new doubly linkedlist node and set its next pointer equal to the current head node and set its previous pointer to null. Then we will set headPointer's previous reference to the newHeadNode and assign the headPointer to the newHeadNode.

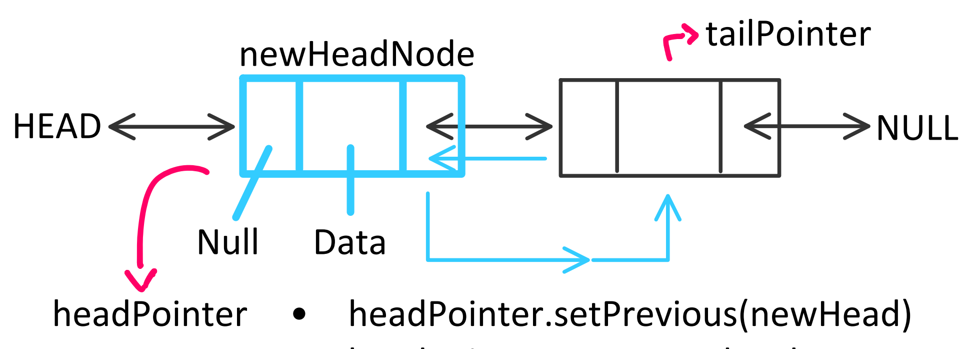


Initial state:



Set newHeadNode.data = data
Set newHeadNode.next = headPointer
Set newHeadNode.prev = null

Final state:

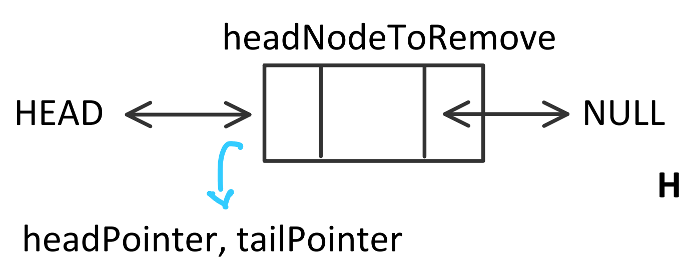


- headPointer.setPrevious(newHead)
- headPointer = newHeadNode

RemoveFromHead()

- If the doubly linkedlist is uninitialized/null, then we cannot removeFromHead() & report ERROR.
- If the doubly linkedlist is not null, that means we do have an existing head node and we need to remove it from our doubly linkedlist.

Initial state:



Final state:

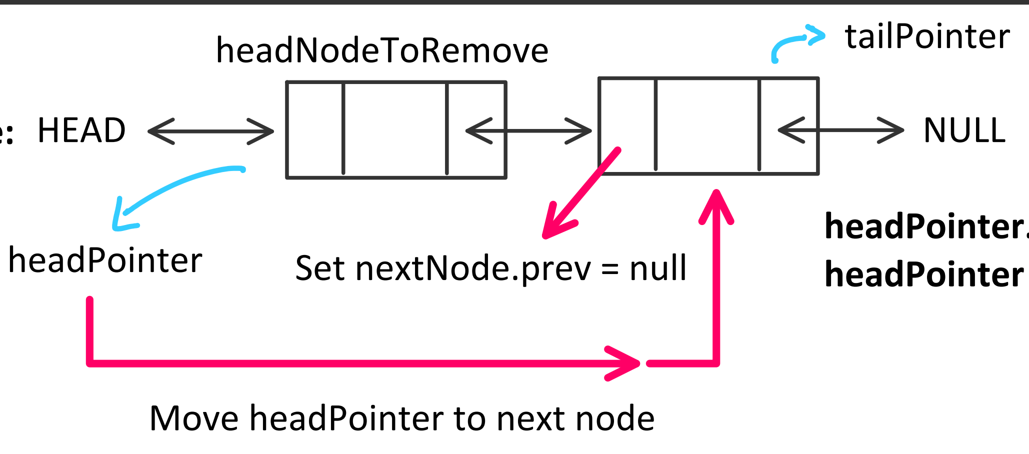
- **headPointer = null**
- **tailPointer = null**

HEAD <—> NULL

Case 1: The doubly linkedlist only contains one node (head) which means head.next == null

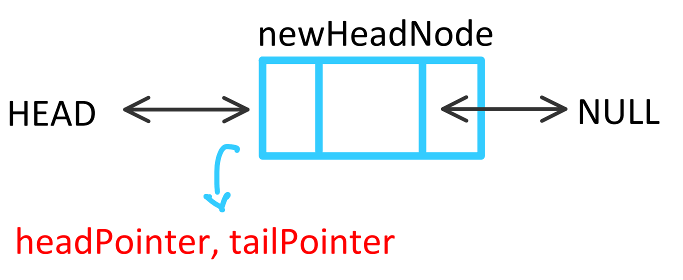
---

Initial state:



headPointer.next.prev = null
headPointer = headPointer.next

Final state:



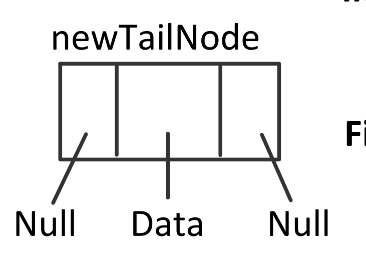
newHeadNode.prev null
newHeadNode.next = nextNode || NULL

Case 2: The doubly linkedlist contains more than one node (head) which means head.next != null

---

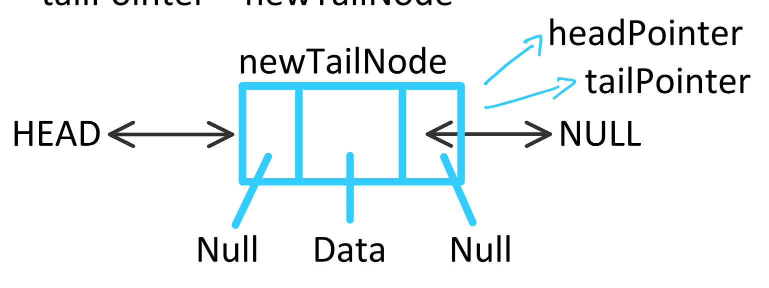
InsertAtTail(int data)

- Check if the doubly linkedlist is uninitialized/null. We can check this by verifying head == null.
- If the headPointer is null, we will need to create a new doubly linkedlist node and set it as the new tail (head) node.

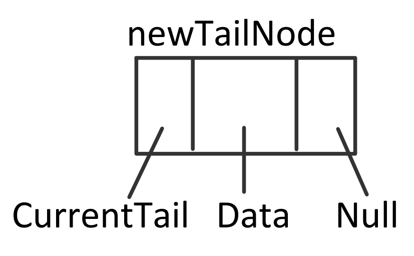


Set newTailNode.data = data
Set newTailNode.next = null
SetNewTailNode.prev = null

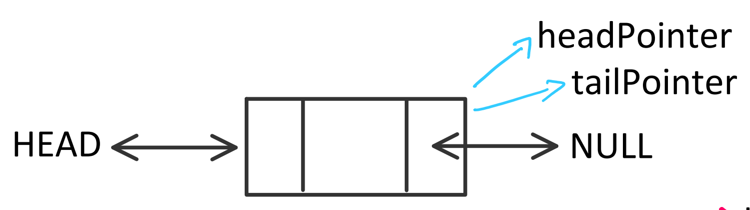
Initial state: headPointer = null,
tailPointer = null,
HEAD <—> NULL
Final state: headPointer = newTailNode,
tailPointer = newTailNode



- If the headPointer is not null, we still need to create a new doubly linkedlist node and set its previous pointer equal to the current tail node and set its next pointer to null. Then we will set tailPointer as the newTailNode.

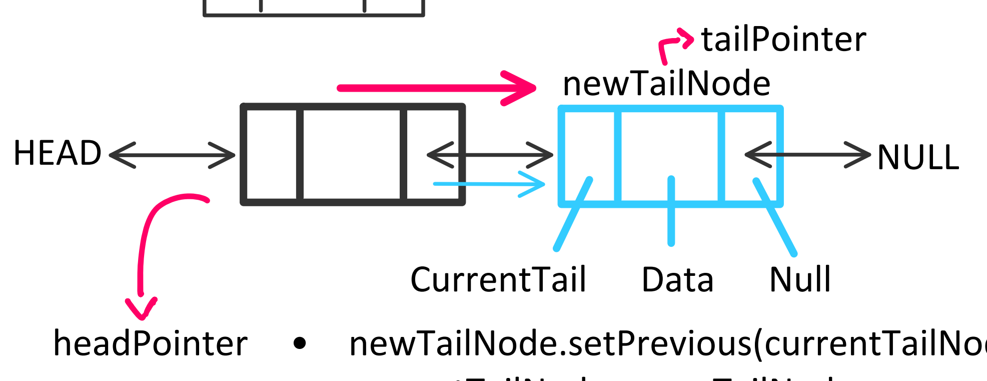


Initial state:



Set newTailNode.data = data
Set newTailNode.next = null
Set newTailNode.prev = currentTailNode
Set currentTailNode.next = newTailNode

Final state:

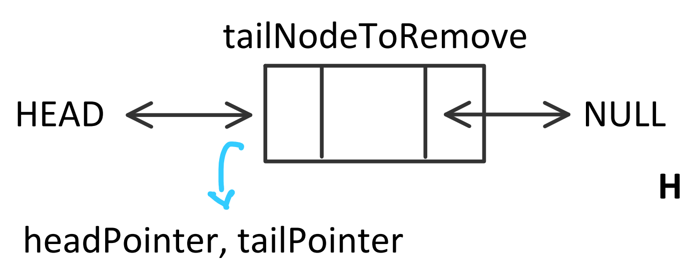


- newTailNode.setPrevious(currentTailNode)
- currentTailNode = newTailNode

RemoveFromTail()

- If the doubly linkedlist is uninitialized/null, then we cannot removeFromTail() & report ERROR.
- If the doubly linkedlist is not null, that means we do have an existing tail node and we need to remove it from our doubly linkedlist.

Initial state:



Final state:

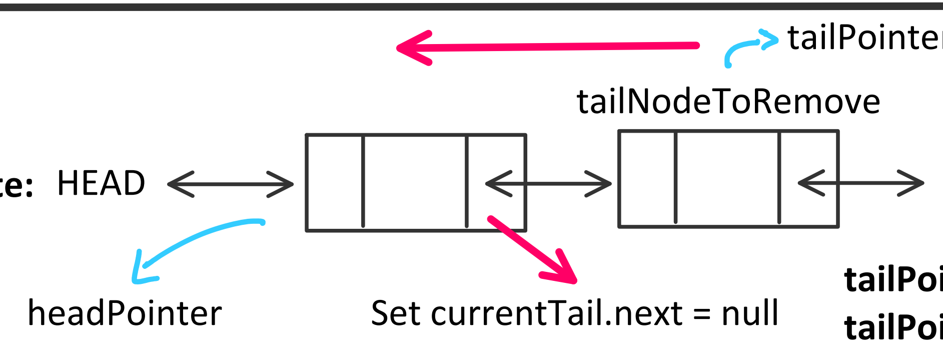
- **headPointer = null**
- **tailPointer = null**

HEAD <—> NULL

Case 1: The doubly linkedlist only contains one node (tail) which means tail.prev == null

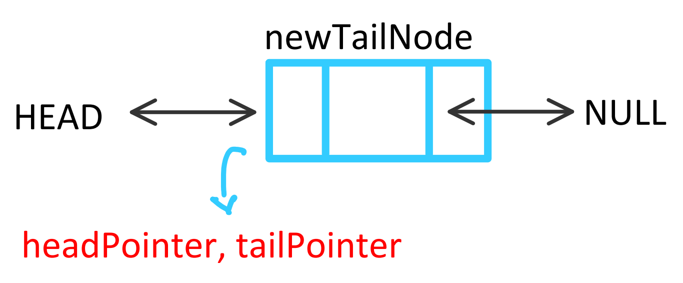
---

Initial state:



tailPointer.prev.next = null
tailPointer = tailPointer.prev

Final state:



newTailNode.prev = null
newTailNode.next = null

Case 2: The doubly linkedlist contains more than one node (head) which means tail.prev != null