HackerRank

# Data Structures

| | | |
|---|---|---|
| Author | : | Avinash Sorab |
| Date Created | : | 25th Aug 2019 |
| Language Used | : | Python 3 |
| Platform | : | HackerRank |

# Contents

## 1. Print the elements of a linked list

The task is to print all the elements of a linked list.

Only one call to this function is required and the head of the linked list is passed.

Keep printing the data of the node and keep iterating till the next node is null.

```
def printLinkedList(head):
    node = head
    while(node != None):
        print(node.data)
        node = node.next
```

## 2. Insert a node at the head of a linked list

The task is to insert a node at the head of a linked list.

A loop needs to be run fetching inputs one by one and calling the insert function.

Every time the function is called, the head of the node is passed. The head after the insert is obviously updated to the new data and this needs to be returned to the main (calling) function so that the function gets the updated head everytime it's being called.

Inside the insert function, a node is created for the data it receives.

If the head is null, which is for the first call (empty linked list), The created node is simply returned, indicating that it is the first element.

Else, the created node's next is set to the head. This way new head will be the newly created node and old head continues to link the other elements.

The created node is returned in the end.

```
def insertNodeAtHead(head, data):
    node = SinglyLinkedListNode(data)
    if(head):
        node.next = head
    return node
```

## 3. Insert a node at a specific position in a linked list

The task is to insert a node in any specified position in a linked list.

The function is given the data and the position at which the data node needs to be inserted.

The function needs to return the head node everytime for the calling function. This helps the calling function to call the insert function by sending it an updated linked list everytime.

The function does the following functionality, it creates the node out of the data given to it.

If the head is None, which means the linked list is empty, the node just created is sent as a head, since that's the first element of the linked list.

Else, The existing linked list is iterated using a new inode variable(iterable node)

We need to keep track of how many nodes are we jumping, using a count variable, when this count equals position, there we need to insert the node created earlier.

To insert, first we need to store the next node in a temporary variable, replace this node as the next node and update the next node's next as temp.

Finally return the head immediately to come out of the while loop.

```
def insertNodeAtPosition(head, data, position):
    node = SinglyLinkedListNode(data)
    if(head == None):
        return node
    else:
        count = 0
        inode = head
        while(inode != None):
            count+=1
            if(count == position):
                temp = inode.next
                inode.next = node
                node.next = temp
                return head
            inode = inode.next
```

## 4. Insert a node at the tail of a linked list

The task is to insert a node at the tail of a linked list.

The function receives head and data every time it is called, so it is important to return the head of the linked list from the function so that the next time function is called, the function is going to receive an updated linked list (head).

Create a node out of the data sent.

If the head is None, return the node created since it is the only element in the linked list.

Else, using the head node as the iterative node, loop through the linked list to reach the tail and the way to ensure the reach of tail node is to check if the next node is null from being at the current node.

If the next node is null, replace the next node with the node created earlier.

Immediately returning the head from there is very important other wise it keeps adding the same node over and over and your code will never actually terminate/come out of while loop.

```
def insertNodeAtTail(head, data):
    node = SinglyLinkedListNode(data)

    if(head == None):
        return node
    else:
        inode = head
        while(inode != None):
            if(inode.next != None):
                inode = inode.next
            else:
                inode.next = node
                return head
```