Principles of Computer Science II

Abstract Data Types

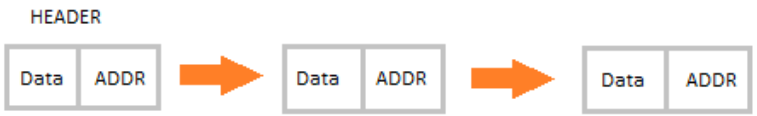
Ioannis Chatzigiannakis

Sapienza University of Rome

Lecture 14

Introduction to Linked Lists

Linked List is a linear data structure and it is very common data structure which consists of group of nodes in a sequence which is divided in two parts. Each node consists of its own data and the address of the next node and forms a chain. Linked Lists are used to create trees and graphs.

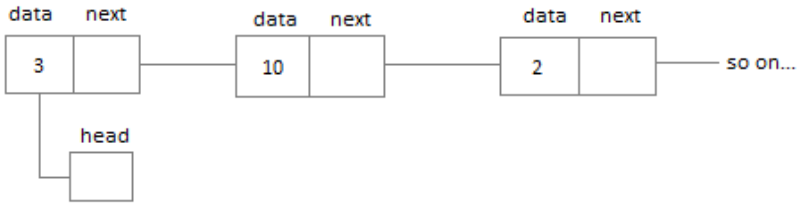


Basic features of Linked Lists

1. They are a dynamic in nature which allocates the memory when required.
2. Insertion and deletion operations can be easily implemented.
3. Stacks and queues can be easily executed.
4. Linked List reduces the access time.
5. Pointers require extra memory for storage.
6. No element can be accessed randomly; it has to access each node sequentially.
7. Reverse Traversing is difficult in linked list.
8. Linked lists let you insert elements at the beginning and end of the list.

Linear Linked Lists

Singly linked lists contain nodes which have a data part as well as an address part i.e. next, which points to the next node in sequence of nodes. The operations we can perform on singly linked lists are insertion, deletion and traversal.



Linked List Nodes as Pointers

```
class Node(object):
    def __init__(self, value=None, pointer=None):
        self.value = value
        self.pointer = pointer

    def getData(self):
        return self.value

    def getNext(self):
        return self.pointer

    def setData(self, newdata):
        self.value = newdata

    def setNext(self, newpointer):
        self.pointer = newpointer
```

Nodes: Testing

```
if __name__ == '__main__':
    L = Node("a", Node("b", Node("c", Node("d"))))
    assert(L.pointer.pointer.value=='c')

    print(L.getData())
    print(L.getNext().getData())
    L.setData('aa')
    L.setNext(Node('e'))
    print(L.getData())
    print(L.getNext().getData())
```

Linked List Code: Initialization

```
from node import Node

class LinkedList(object):

    def __init__(self):
        self.head = None
        self.length = 0
```

Linked List: Insertion at the Beginning

1. The first Node is the Head for any Linked List.
2. When a new Linked List is instantiated, it just has the Head, which is Null.
3. Else, the Head holds the pointer to the first Node of the List.
4. When we want to add any Node at the front, we must make the head point to it.
5. And the Next pointer of the newly added Node, must point to the previous Head, whether it be NULL(in case of new List) or the pointer to the first Node of the List.
6. The previous Head Node is now the second Node of Linked List, because the new Node is added at the front.

Linked List: Insertion at the Beginning

```
1     def addFirst(self, value):
2         self.length = 1
3         node = Node(value)
4         self.head = node
```

Linked List: Insertion at the End

1. If the Linked List is empty then we simply, add the new Node as the Head of the Linked List.
2. If the Linked List is not empty then we find the last node, and make it' next to the new Node, hence making the new node the last Node.

Linked List: Insertion at the Beginning

```
1     def add(self, value):
2         node = self.head
3         prev = node
4         while node:
5             prev, node = node, node.pointer
6
7         self.length += 1
8         node = Node(value)
9         prev.pointer = node
10
11    def addNode(self, value):
12        if not self.head:
13            self.addFirst(value)
14        else:
15            self.add(value)
```

Linked List: Printing the elements

```
1     def printList(self):
2         node = self.head
3         while node:
4             print(node.value)
5             node = node.pointer
```

Linked List: Finding an elements

```
1     def find(self, index):
2         prev = None
3         node = self.head
4         i = 0
5         while node and i < index:
6             prev = node
7             node = node.pointer
8             i += 1
9         return node, prev, i
```

Linked List: Deleting a Node

1. We first search the Node with data which we want to delete.
2. If the Node to be deleted is the first node, then simply set the Next pointer of the Head to point to the next element from the Node to be deleted.
3. If the Node is in the middle somewhere, then find the Node before it, and make the Node before it point to the Node next to it.

Linked List: Deleting a Node

```
1     def deleteNode(self, index):
2         if not self.head or not self.head.pointer:
3             self.deleteFirst()
4         else:
5             node, prev, i = self.find(index)
6             if i == index and node:
7                 self.length -= 1
8                 if i == 0 or not prev :
9                     self.head = node.pointer
10                else:
11                    prev.pointer = node.pointer
12
13            else:
14                print('Node with index {} not found'.format
                      (index))
```

Linked Lists: Testing

```
1 if __name__ == '__main__':
2     ll = LinkedList()
3     for i in range(1, 5):
4         ll.addNode(i)
5     print('The list is:')
6     ll.printList()
7     print('The list after deleting node with index 2:')
8     ll.deleteNode(2)
9     ll.printList()
10    print('The list after adding node with value 15')
11    ll.add(15)
12    ll.printList()
13    print("The list after deleting everything...")
14    for i in range(ll.length-1, -1, -1):
15        ll.deleteNode(i)
16    ll.printList()
```

4^{th} Assignment

- https://www.rosalind.info/
  - Complete the following challenges:
    dna, rna, revc, gc, subs, cons, lcsm, revp, splc, lexf, fib, lgis
  - http://rosalind.info/problems/{challenge}
- Create a GitHub repository and upload the code for each exercise.
- Email `ichatz@diag.uniroma1.it`
  Subject: [PCS2] Homework 4
  A .zip or a .tar.gz file with your python solutions, for all challenges.
  Also send your account user account link:
  http://rosalind.info/users/{username}
- **Deadline: 26/November/2019, 23:59**