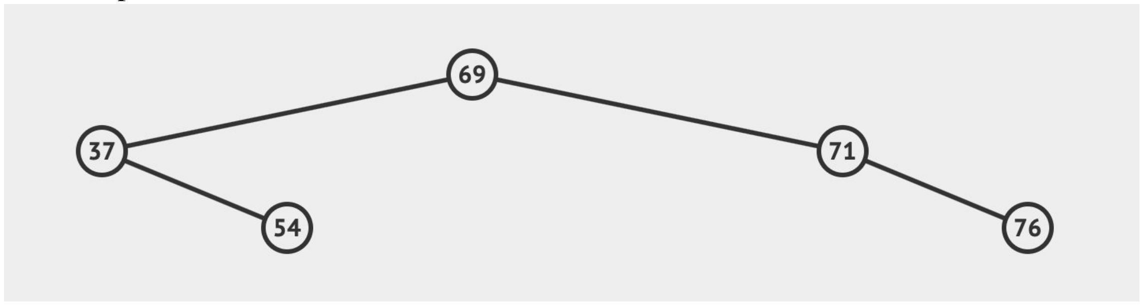
Problem 1 (1 points) – Suppose that BSTree is a class that implements a binary search tree whose elements are integers. Write a method that returns a doubly linked list containing the elements of the tree's nodes that are not leaves. The list should be sorted in **descending order**.

Some help:

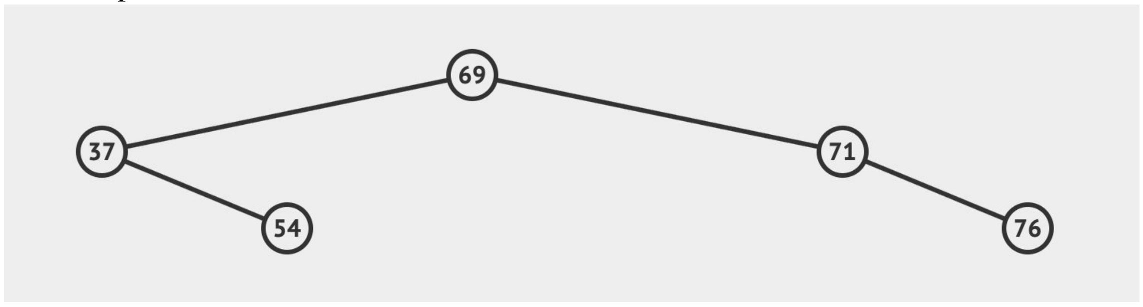
- It is not allowed to use any sorting algorithm to sort the list.
- It is not allowed to use the Python List class. You must use the Doubly Linked List class studied during the course. You must also implement those methods of this class that you use in your solution.

For example:



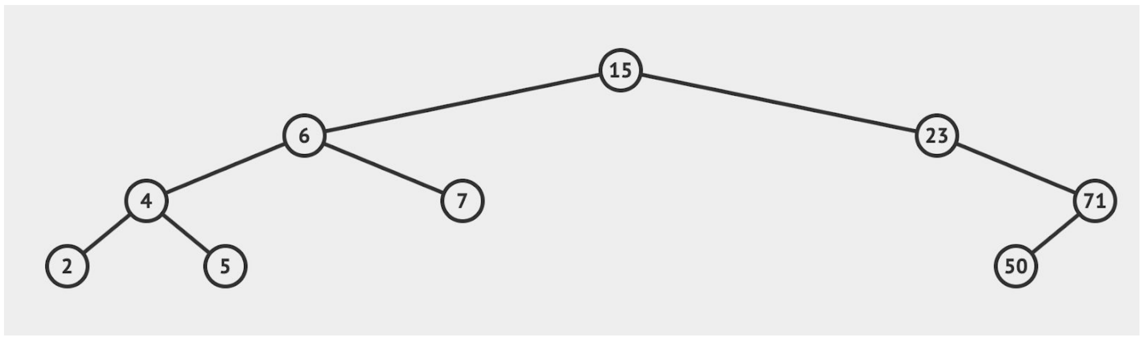
The output will be the doubly linked list containing the following elements: 71,69,37

Problem 2 (1 points) – In the BSTree class, write a method that returns the kth smallest element of the binary tree. What is the time complexity of the method?.
For example:



The 1st smallest element is 37
The 2nd smallest element is 54
The 3th smallest element is 69
The 4th smallest element is 71
The 5th smallest element is 76

Problem 3 (1 point):
A) (0.5) Draw its height-balanced tree.
B) (0.5) Draw its size-balanced tree.



Problem 4 (2 points). Given the following class that implements a social network for UC3M students:

```
class FriendsUC3M:
    def __init__(self):
        self.users = {}

    def addUser(self,email):
        if email in self.users:
            print(email, 'already exists!!!')
            return
        self.users[email] = []

    def getFriends(self,email):
        if email  not in self.users:
            print(email, 'does not exist!!!')
            return
        return self.users[email]


    def areFriends(self,email1,email2):
        if email1 not in self.users:
            return False
        if email2 not in self.users:
            return False
        if email2 in self.users[email1]:
            return True
        if email1 in self.users[email2]:
            return True
        return False

    def addFriends(self,email1,email2):
        if email1 not in self.users:
            self.addUser(email1)
        if email2 not in self.users:
            self.addUser(email2)
        if not self.areFriends(email1,email2):
            self.users[email1].append(email2)
            self.users[email2].append(email1)
```

Write a method, **friendsAtdistance**, which takes a user (an email) and a number, k, and returns a Python lists containing those users with a distance ‘k’ from the input user.

For example, given the social network:

> *pmf:['isa', 'lourdes']*
> *isa:['pmf']*
> *lourdes:['pmf', 'ana']*
> *ana:['lourdes', 'ines', 'mateo']*
> *ines:['ana']*
> *mateo:['ana']*
>
> *friendsAtdistance(‘pmf’,1) returns ['isa', 'lourdes']*
> *friendsAtdistance(‘pmf’,2) returns ['ana']*
> *friendsAtdistance(‘pmf’,3) returns [ines, 'mateo']*

It is allowed to use data structures such as Python dictionary or Python List to implement your final solution.

Problem 5 (1 point). Implement a method based on **divide and conquer** strategy that takes a sorted Python List of numbers, *A*, and a number, x. The method must return the index of x in the list. If x is not found, the method returns -1.