

Assignment 1

Before moving on to the questions and the approaches taken to solve them, I want to briefly discuss about the structure of the tree that we are using. We are maintaining a threaded binary search tree with the count of the number of right subtree elements present.

Threaded Binary Search tree uses the concept that the extra $n+1$ nodes that remain unused can also be used for maintaining the location of the predecessor and successor of a node. To know whether we are using the left and right pointer for the inorder successor or inorder predecessor of the node we also maintain a flag (isleft and isright) which is set if the left pointer points to the inorder predecessor and not set if it points to the left child (similar for right).

Count of the number of right subtree nodes is required to print the k^{th} largest element. Detail of the approach taken is mentioned below.

N.B. In all the diagrams blue links are the threads and black links are the tree nodes.

1. [10 points] insert (x) -- insert an element x (if not present) into the BST. If x is present, throw an exception.

As mentioned in the class we are maintaining a threaded binary search tree throughout the program so that the calculation of successor and predecessor becomes faster.

While insertion of a node in the threaded binary search tree there can be three cases that may arise –

Case 1: Insertion in an empty tree

In this case both the left and right pointer of the new node will become NULL, and the root should point to the new node.

Case 2: If the inserted node is left child of the parent node

In this case we have to make the left and right pointers of the newly inserted node point to the predecessor and successor respectively.

As the node inserted is the left child of the parent node, we can conclude the following:

- Parent node is the inorder successor of the newly inserted node.
- Predecessor of the parent node will become the predecessor of the newly inserted node because the newly inserted node lies between the inorder predecessor and the parent node.

Example:

Inorder of tree – 2 3 4 6 8 ; 5 is being inserted

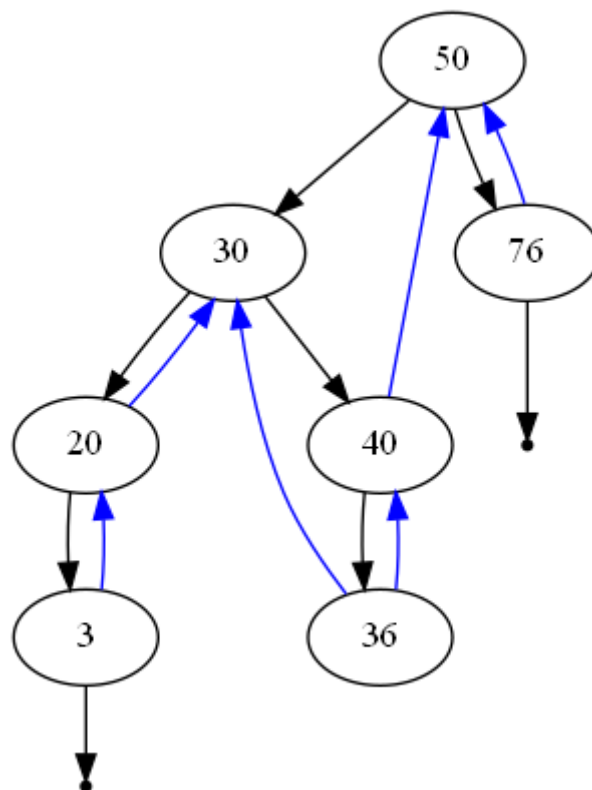
5 will come between 4 and 6 so the predecessor of 5 will become predecessor of 6 and successor of 5 will become 6.

Case 3: If the inserted node is the right child of the parent node

The operations will be almost same as that of the previous case, except that the parent node will now become the inorder predecessor of the newly inserted node and the inorder successor of the parent node will become the inorder successor of the newly inserted node.

Also, while inserting in the binary search tree, we should keep track of the number of elements present in the right subtree with respect to each node. To do that whenever we are inserting any element in the right subtree of the node then we are incrementing the right subtree element count of that node by one.

Test Case: Insert 50, 30, 40, 20, 36, 76, 3 in that order.



Output 1

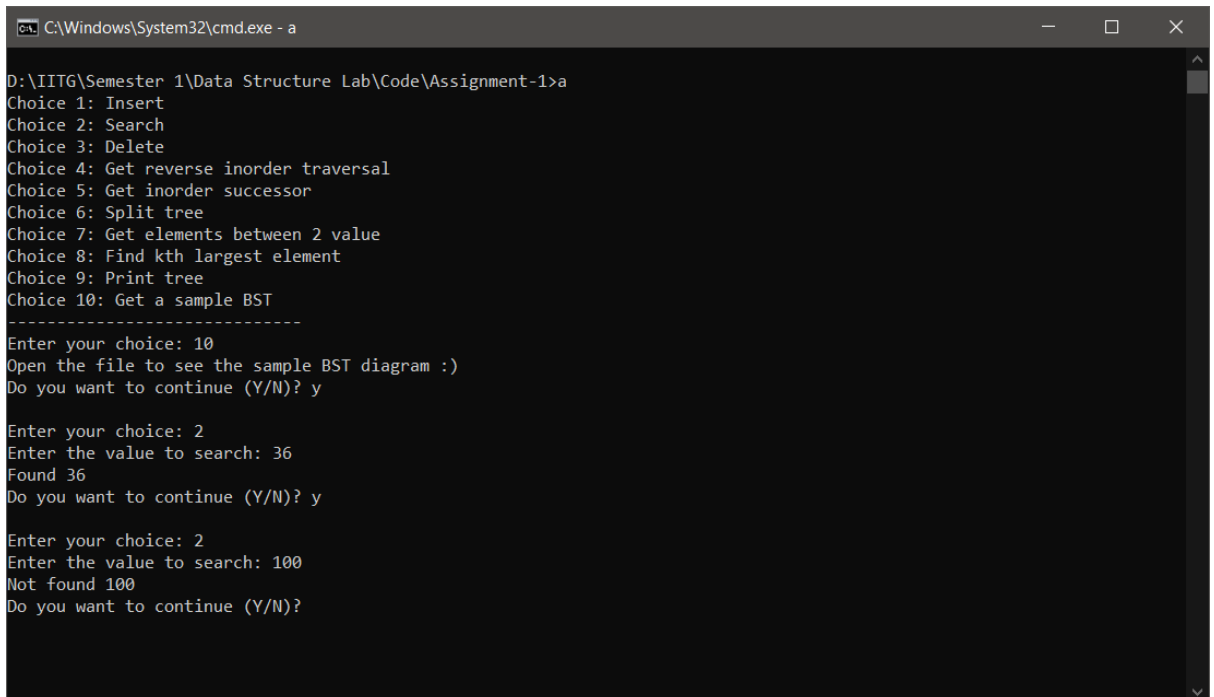
2. [10 points] search(x) -- search an element x, if found return its reference, otherwise return NULL.

Here the logic is quite simple, we have to check if the current node is less than or greater than the value to be searched. If it is less, then move to the right child else move to the left child.

We just have to take care of the threading, i.e., if we want to move left, we first have to check whether the left pointer is a tree edge or a thread. If it is thread then we don't move there and return null because it signifies that we have reached the leftmost node and we can't traverse further to left of that node.

Test Case 1: search(36)

Test Case 2: search(100)



```
C:\Windows\System32\cmd.exe - a
D:\IITG\Semester 1\Data Structure Lab\Code\Assignment-1>a
Choice 1: Insert
Choice 2: Search
Choice 3: Delete
Choice 4: Get reverse inorder traversal
Choice 5: Get inorder successor
Choice 6: Split tree
Choice 7: Get elements between 2 value
Choice 8: Find kth largest element
Choice 9: Print tree
Choice 10: Get a sample BST
-----
Enter your choice: 10
Open the file to see the sample BST diagram :)
Do you want to continue (Y/N)? y

Enter your choice: 2
Enter the value to search: 36
Found 36
Do you want to continue (Y/N)? y

Enter your choice: 2
Enter the value to search: 100
Not found 100
Do you want to continue (Y/N)?
```

Output 2

3. [20 points] delete(x) -- delete an element x, if the element x is not present, throw an exception.

Like insertion here also 3 cases can come:

Case 1: Leaf Node needs to be deleted

In binary search tree for the deletion of leaf node the left or right pointer of the parent node was set as NULL. In this case as we are using a threaded BST so instead of making it as NULL we use it as a thread.

If the leaf node to be deleted is the left child of its parent then after deletion the left pointer of the parent should become the thread pointing to its predecessor (predecessor of the leaf node deleted) else should become its successor (successor of the leaf node to be deleted).

Case 2: Node to be deleted has only once child

First, we do the deletion as we do in normal BST, i.e., connect the parent of the node to the child of the node.

Then, to maintain threading we find if the node deleted has left threading or right threading. If it is left threaded then we connect the successor of the deleted node to the predecessor of the deleted node and vice versa.

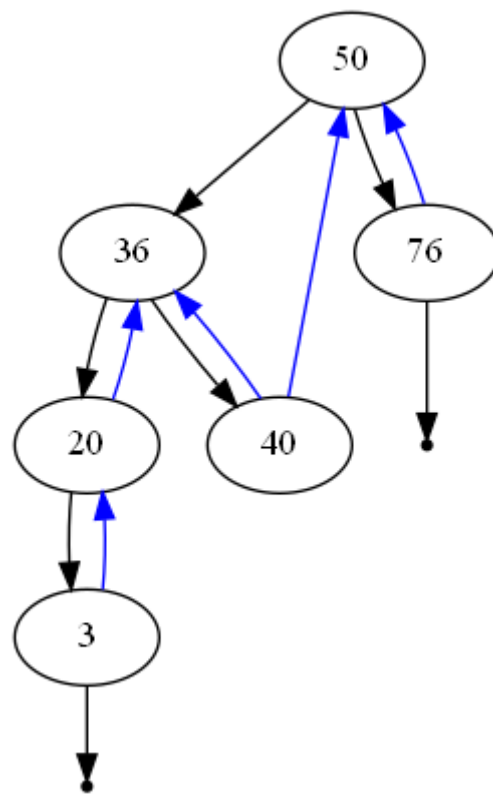
Case 3: Node to be deleted has both children

In this case we first find out the inorder successor of the node and copy the value of the inorder successor to the node. After that we delete the inorder successor of the to be deleted node using the approaches mentioned in case 1 or case 2 as it may fall into.

However, as we are also keeping track of the number of children present in the right subtree of a node, we should decrease the count whenever we traverse to right as the node to be deleted is a part of the right subtree, making us to move right.

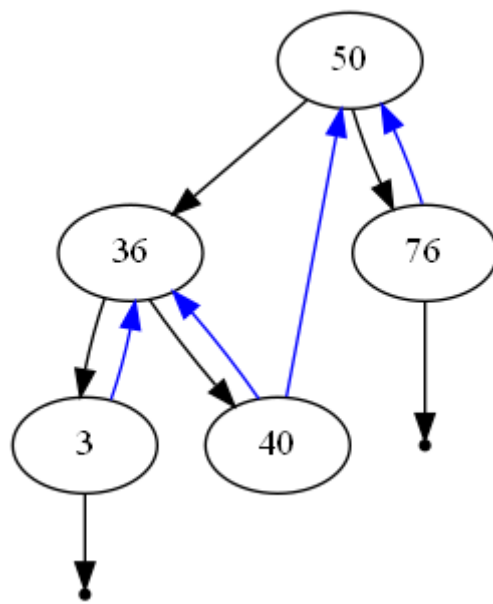
Or if the value is found and it has a right subtree (right subtree element count > 0) then also we decrease the count because we will remove an element from the right subtree of the node to be deleted.

Test Case 1: delete(30)



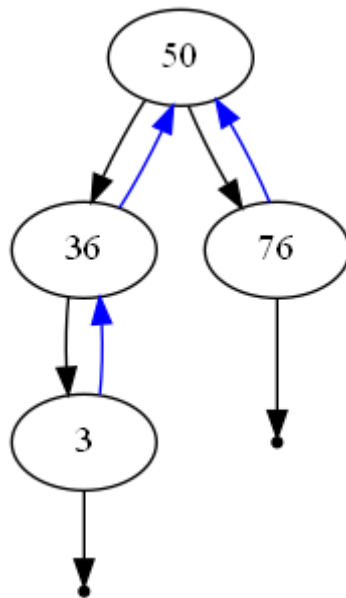
Output 3

Test Case 2: delete(20)



Output 4

Test Case 3: delete(40)

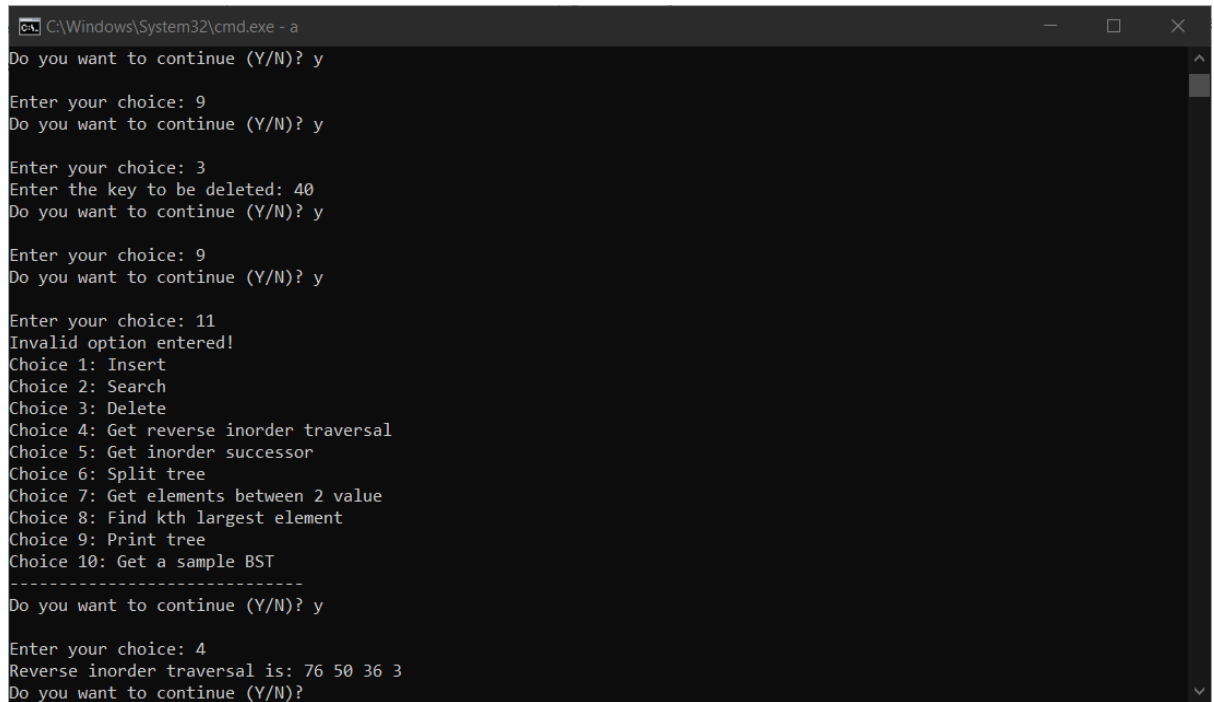


Output 5

4. [20 points] reverseInorder() -- returns a singly linked list containing the elements of the BST in max to min order. You should not use any extra stack and use threading for non-recursive implementation.

As we are already maintaining a threaded BST the implementation is very straight forward. We can just move to the rightmost part of the tree and keep printing the predecessors till we reach NULL.

Test Case 1: reverseInorder() on Output 5



```
C:\Windows\System32\cmd.exe - a
Do you want to continue (Y/N)? y

Enter your choice: 9
Do you want to continue (Y/N)? y

Enter your choice: 3
Enter the key to be deleted: 40
Do you want to continue (Y/N)? y

Enter your choice: 9
Do you want to continue (Y/N)? y

Enter your choice: 11
Invalid option entered!
Choice 1: Insert
Choice 2: Search
Choice 3: Delete
Choice 4: Get reverse inorder traversal
Choice 5: Get inorder successor
Choice 6: Split tree
Choice 7: Get elements between 2 value
Choice 8: Find kth largest element
Choice 9: Print tree
Choice 10: Get a sample BST
-----
Do you want to continue (Y/N)? y

Enter your choice: 4
Reverse inorder traversal is: 76 50 36 3
Do you want to continue (Y/N)?
```

Output 6

Test Case 2: reverseInorder() on Output 1

```
C:\Windows\System32\cmd.exe - a

D:\IITG\Semester 1\Data Structure Lab\Code\Assignment-1>a
Choice 1: Insert
Choice 2: Search
Choice 3: Delete
Choice 4: Get reverse inorder traversal
Choice 5: Get inorder successor
Choice 6: Split tree
Choice 7: Get elements between 2 value
Choice 8: Find kth largest element
Choice 9: Print tree
Choice 10: Get a sample BST
-----
Enter your choice: 10
Open the file to see the sample BST diagram :)
Do you want to continue (Y/N)? y

Enter your choice: 4
Reverse inorder traversal is: 76 50 40 36 30 20 3
Do you want to continue (Y/N)?
```

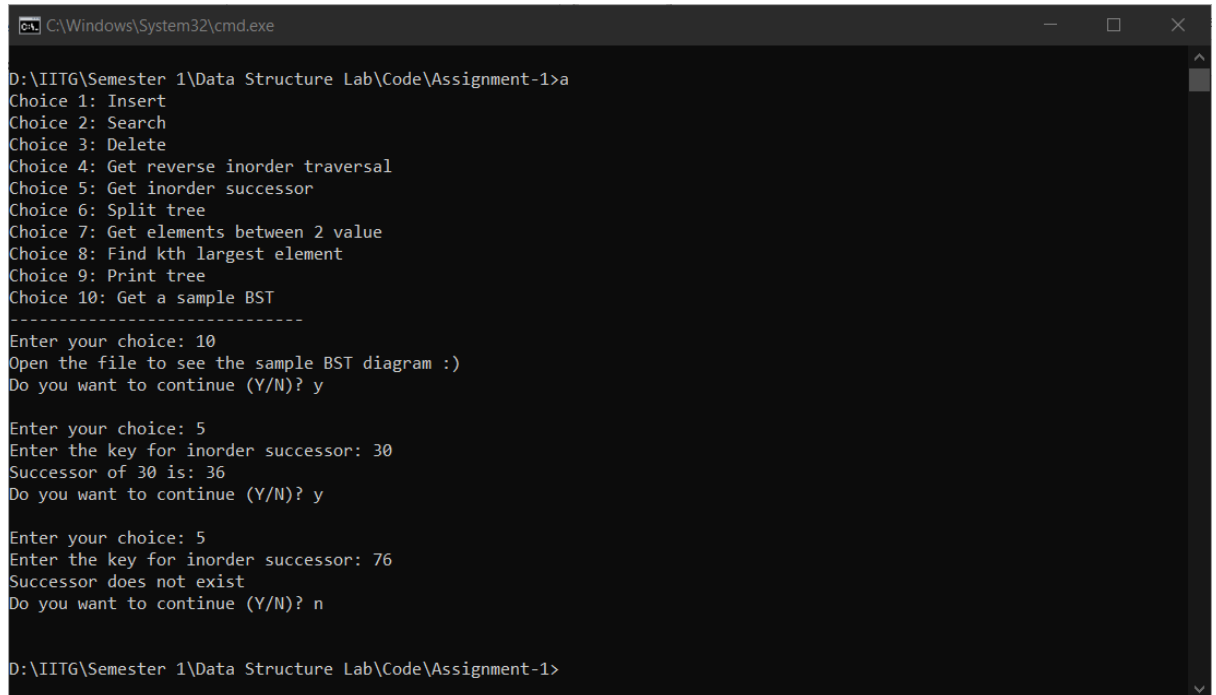
Output 7

5. [10 points] `successor(ptr)` -- returns the key value of the node which is the inorder successor of the `x`, where `x` is the key value of the node pointed by `ptr`.

As we are already maintaining a double threaded BST hence if the pointer whose successor is needed to be found is right threaded then we simply

Test Case 1: `successor(30)` on Output 1

Test Case 2: `successor(76)` on Output 1



```
C:\Windows\System32\cmd.exe
D:\IITG\Semester 1\Data Structure Lab\Code\Assignment-1>a
Choice 1: Insert
Choice 2: Search
Choice 3: Delete
Choice 4: Get reverse inorder traversal
Choice 5: Get inorder successor
Choice 6: Split tree
Choice 7: Get elements between 2 value
Choice 8: Find kth largest element
Choice 9: Print tree
Choice 10: Get a sample BST
-----
Enter your choice: 10
Open the file to see the sample BST diagram :)
Do you want to continue (Y/N)? y

Enter your choice: 5
Enter the key for inorder successor: 30
Successor of 30 is: 36
Do you want to continue (Y/N)? y

Enter your choice: 5
Enter the key for inorder successor: 76
Successor does not exist
Do you want to continue (Y/N)? n

D:\IITG\Semester 1\Data Structure Lab\Code\Assignment-1>
```

Output 8

6. [20 points] `split(k)` -- split the BST into two BSTs T1 and T2, where $\text{keys}(T1) \leq k$ and $\text{keys}(T2) > k$. Note that, k may / may not be an element of the tree. Your code should run in $O(h)$ time, where h is the height of the tree. Print the inorder of both the BSTs T1 and T2 using recursive/non-recursive implementation within this function.

As we know that in a binary search tree every node in the left subtree has value less than the value of the root node and every node in the right subtree has value greater than the value of the root node. Thus if the root node's value is less than or equal to the given value 'K' then after splitting the tree, this root node must be included in the 'first' tree, and also we can assure that the left subtree of this root node will also be part of 'first' tree, as all nodes of left subtree must have a value less than 'K'. Or if the root node's value is greater than the given value 'K', then this root node will be included in the 'second' tree and the right subtree of the root node will also be included in the 'second' tree as all nodes in right subtree must have a value greater than 'K'. In order to split the tree into two parts, we can iteratively move over nodes and compare the value of nodes to find the next node to check, if the value of a current node is smaller or equal to the given value, then add the left subtree to the 'first' tree and move to the right child of the current node, else add right subtree to 'second' tree and move to left child of the node.

Test Case 1: split(40) on Output 1

Test Case 2: split(100) on Output 1

Test Case 3: split(1) on Output 1

```
Choice 3: Delete
Choice 4: Get reverse inorder traversal
Choice 5: Get inorder successor
Choice 6: Split tree
Choice 7: Get elements between 2 value
Choice 8: Find kth largest element
Choice 9: Print tree
Choice 10: Get a sample BST
-----
Enter your choice: 10
Open the file to see the sample BST diagram :)
Do you want to continue (Y/N)? y

Enter your choice: 6
Enter the value to split: 40

First BST after split: 3 20 30 36 40

Second BST after split: 50 76
Do you want to continue (Y/N)? y

Enter your choice: 6
Enter the value to split: 100

First BST after split: 3 20 30 36 40 50 76

Second BST after split:
Do you want to continue (Y/N)? y

Enter your choice: 6
Enter the value to split: 1

First BST after split:

Second BST after split: 3 20 30 36 40 50 76
Do you want to continue (Y/N)? n

D:\IITG\Semester 1\Data Structure Lab\Code\Assignment-1>
```

Output 9

7. [20 points] allElementsBetween(k1, k2) -- returns a singly linked list (write your own class) that contains all the elements (k) between k1 and k2, i.e., $k_1 \leq k \leq k_2$. Your code should run in $O(h + N)$ time, where N is the number of elements that appears between k1 and k2 in the BST.

For this we are using the property of threaded BST. We are traversing to the left most child and then doing an inorder traversal by only going through the successor nodes.

While traversing through the nodes whenever we are coming across a node whose value lies between k1 and k2 we are inserting in the linked list and moving forward. If we cross the point k2 then we are breaking out of the traversal and just returning the linked list.

Test Case 1: allElementsBetween(1, 40) on Output 1

Test Case 2: allElementBetween(20, 100) on Output 1

Test Case 3: allElementsBetween(18, 50) on Output 1

```
D:\IITG\Semester 1\Data Structure Lab\Code\Assignment-1>a
Choice 1: Insert
Choice 2: Search
Choice 3: Delete
Choice 4: Get reverse inorder traversal
Choice 5: Get inorder successor
Choice 6: Split tree
Choice 7: Get elements between 2 value
Choice 8: Find kth largest element
Choice 9: Print tree
Choice 10: Get a sample BST
-----
Enter your choice: 10
Open the file to see the sample BST diagram :)
Do you want to continue (Y/N)? y

Enter your choice: 7
Enter first value: 1
Enter second value: 40
3 20 30 36 40
Do you want to continue (Y/N)? y

Enter your choice: 7
Enter first value: 20
Enter second value: 100
20 30 36 40 50 76
Do you want to continue (Y/N)? y

Enter your choice: 7
Enter first value: 18
Enter second value: 50
20 30 36 40 50
Do you want to continue (Y/N)? n
```

Output 10

8. [20 points] kthElement(k) -- finds the k-th largest element in the BST and prints the key value. Your code should run in $O(h)$ time.

To implement the k^{th} largest element, we are maintaining the right subtree count in each of the nodes and doing the following steps starting from the root of the tree:

- If right subtree count of the element is less than k then we are moving to the left subtree and reducing k by right subtree count of the current node + 1, as we have already passed that many larger elements.

$$K = K - (\text{right subtree count of the current node} + 1)$$

- If the right subtree count of the element is more than required then we move to the right subtree and do not change the value of k.
- If we have reached the leaf node then we check if the count of the leaf node + 1 is the value of k then we print it else we say that invalid k is given as input.

Test Case 1: kthElement(4) on Output 1

```
D:\IITG\Semester 1\Data Structure Lab\Code\Assignment-1>a
Choice 1: Insert
Choice 2: Search
Choice 3: Delete
Choice 4: Get reverse inorder traversal
Choice 5: Get inorder successor
Choice 6: Split tree
Choice 7: Get elements between 2 value
Choice 8: Find kth largest element
Choice 9: Print tree
Choice 10: Get a sample BST
-----
Enter your choice: 10
Open the file to see the sample BST diagram :)
Do you want to continue (Y/N)? y

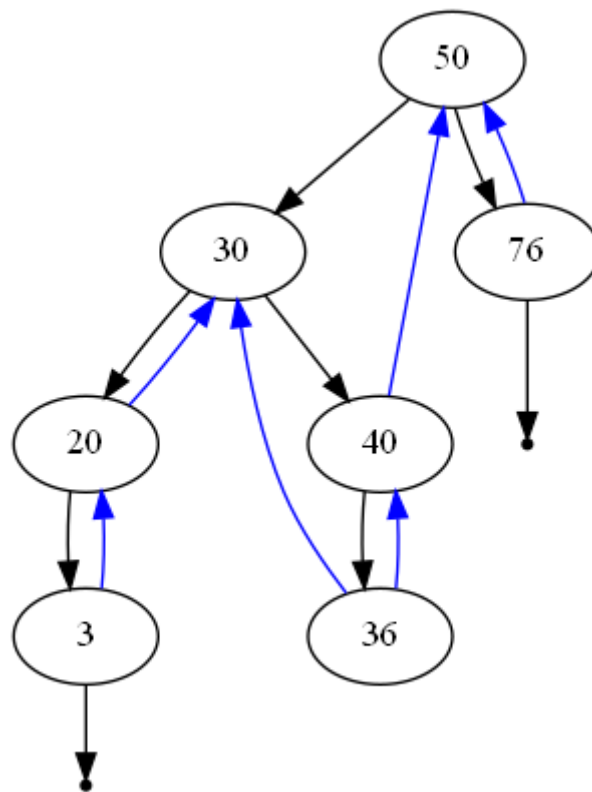
Enter your choice: 8
Enter the value of k: 100
Invalid value entered!
Do you want to continue (Y/N)? y

Enter your choice: 8
Enter the value of k: 4
4th largest element is: 36
Do you want to continue (Y/N)? y

Enter your choice: 4
Reverse inorder traversal is: 76 50 40 36 30 20 3
Do you want to continue (Y/N)? n
```

9. [20 points] printTree() -- Your program should print the BST (in a tree like structure) [use graphviz].

For implementing this function, we are using a helper function which recursively traverses all the nodes and outputs a file. This file is takes as input by the dot engine which outputs a png file showing the picture of the tree



Example 1