CS 202 Fundamental Structures of Computer Science II

Assignment 2 – Trees

Assigned on: 19 October 2015 (Monday)
Due Date: 3 November 2015 (Tuesday)

Name: Gülsüm
Surname: Güdükbay
ID: 21401148
Section: 1

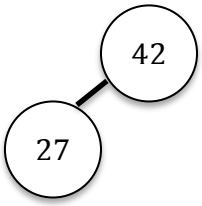
1) Giving the prefix, infix, and postfix expressions, for an expression tree

Prefix: – x x A B – + C D E / F + G H
Infix: ((A x B) x (C + D – E)) – (F / (G + H))
Postfix: A B x C D + E – x F G H + / –

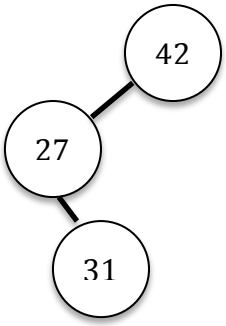
2) Showing Operations on a Binary Search Tree

Inserting 42: No comparisons are made since the root is null and 42 becomes the root.

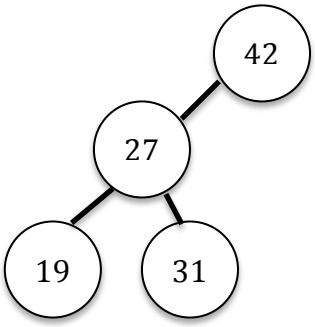
Inserting 27: Compared with 42, smaller, so inserted to the left child position of 42.



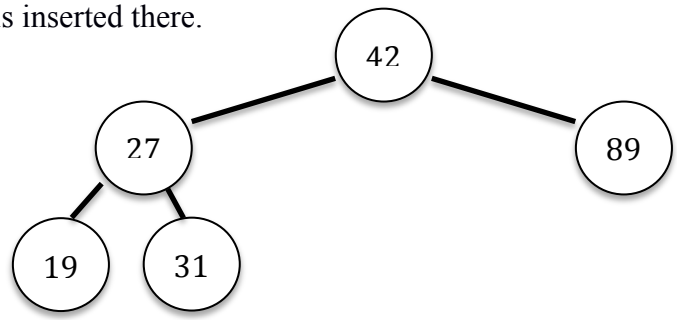
Inserting 31: Root is not null, since 31 is smaller than the root and since the root has a left child, insertion is called for 27's left child, and 31 is inserted to the right child position of 27.



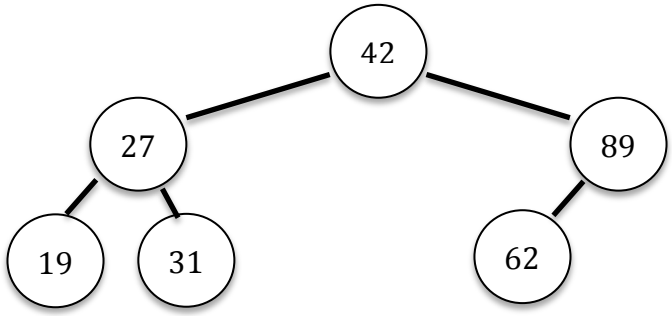
Inserting 19: Root is not null and 19 is smaller than the root 42, which has a left child, so the insertion is called for 27. 27 is larger than 19 and since 27 doesn't have a left child 19 is inserted there.



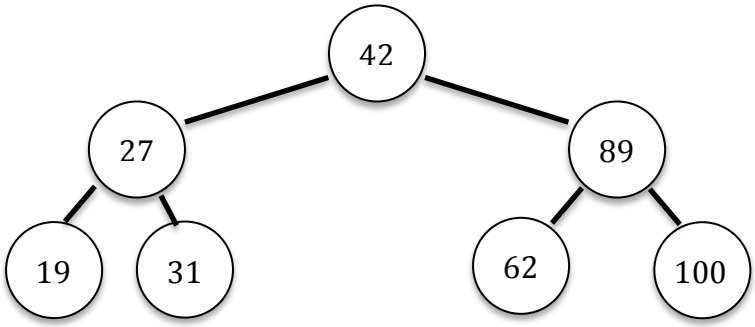
Inserting 89: Root is not null, and it is smaller than 89, so root's right child is checked. It is null, so 89 is inserted there.



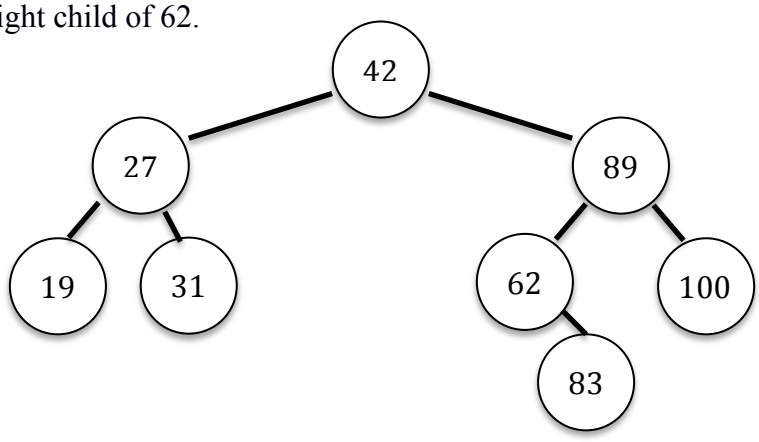
Inserting 62: Root is not null, and 62 is greater than 42, so the right child of the root is checked. It is not null, so the insertion is called for 89. 89 is a leaf node and 62 is less than 89, so 62 is inserted as the left child of 89.



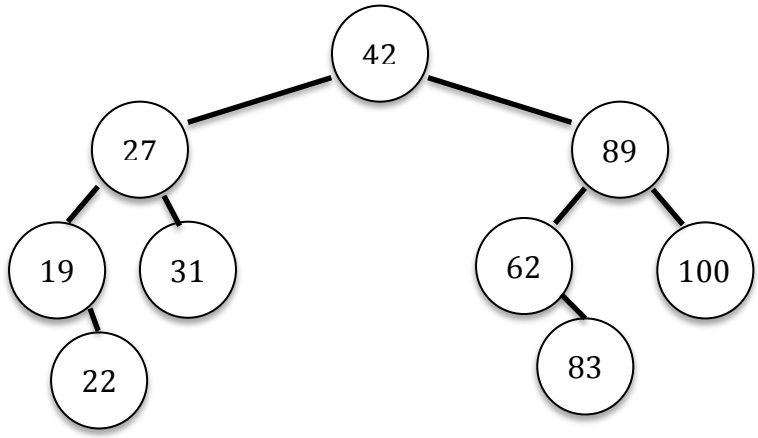
Inserting 100: Root is not null, and 100 is greater than 42, so the right child of the root is checked. It is not null, so the insertion is called for 89. 89 doesn't have a right child and 100 is greater than 89, so 100 is inserted as the right child of 89.



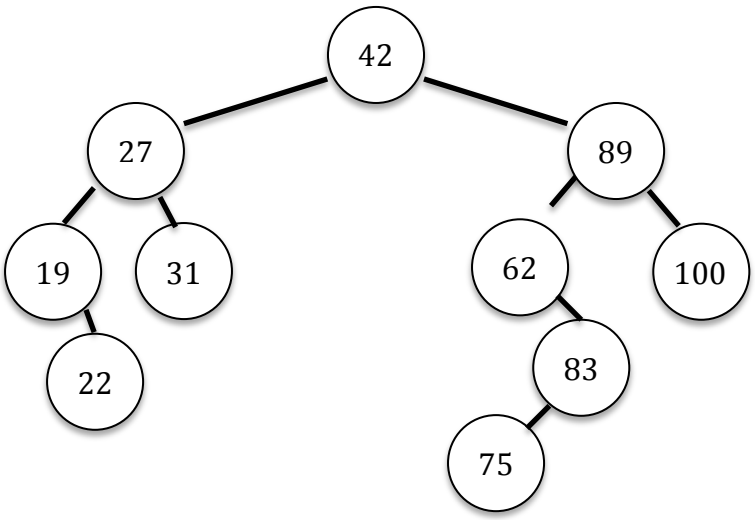
Inserting 83: 83 is greater than 42, so the right child of 42 is checked. It is not null, so the insertion is called for 89. 83 is smaller than 89, so the left child of 89 is checked. Left child of 89 is 62, so it is not null. Since 62 is a leaf node and is smaller than 83, 83 becomes the right child of 62.



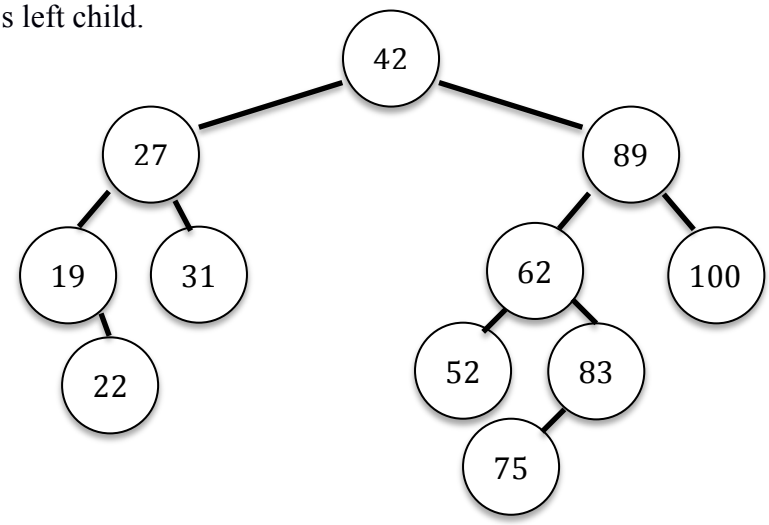
Inserting 22: 22 is smaller than the root 42, so the left child of 42, 27 is checked. Since it has two children, 22 is smaller than 27 and 19, which is the left child of 27 is a leaf node, 22 and 19 is compared. 22 is larger than 19, so 22 becomes 19's right child.



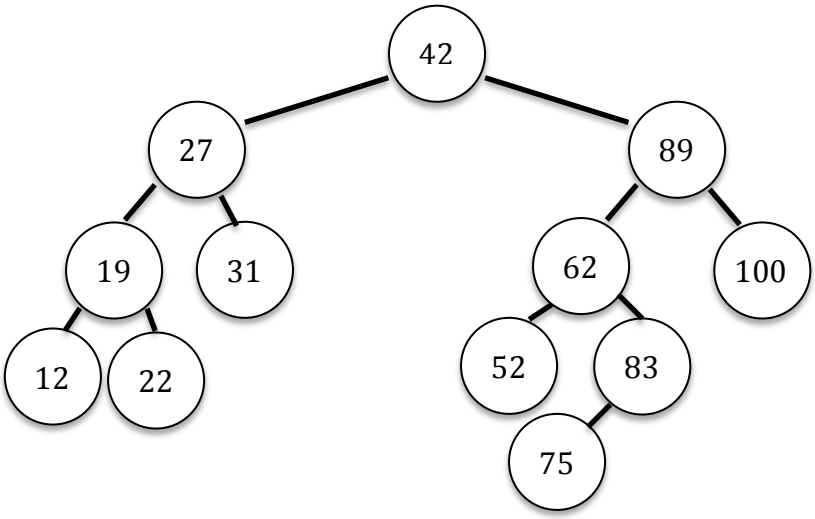
Inserting 75: 75 is larger than 42, so the left child of 42 is checked. 89 is larger than 75, so the left child of 89 is checked. 62 is smaller than 75, so right child of 62 is checked. 83 is larger than 75, so 75 becomes 83's left child.



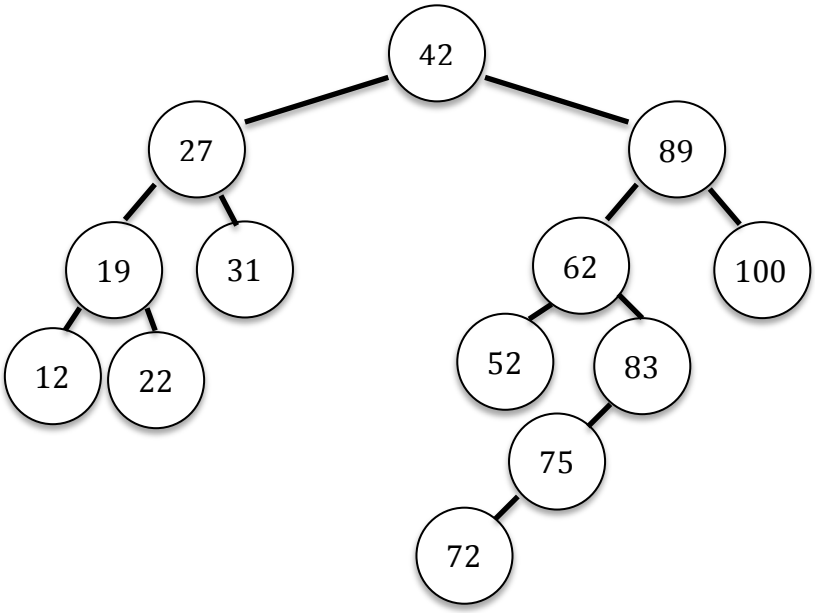
Inserting 52: 52 is larger than 42, so the right child of 42 is checked. 89 is larger than 52, so left child of 89 is checked. 62 is larger than 42 and it doesn't have a left child, so 52 becomes 62's left child.



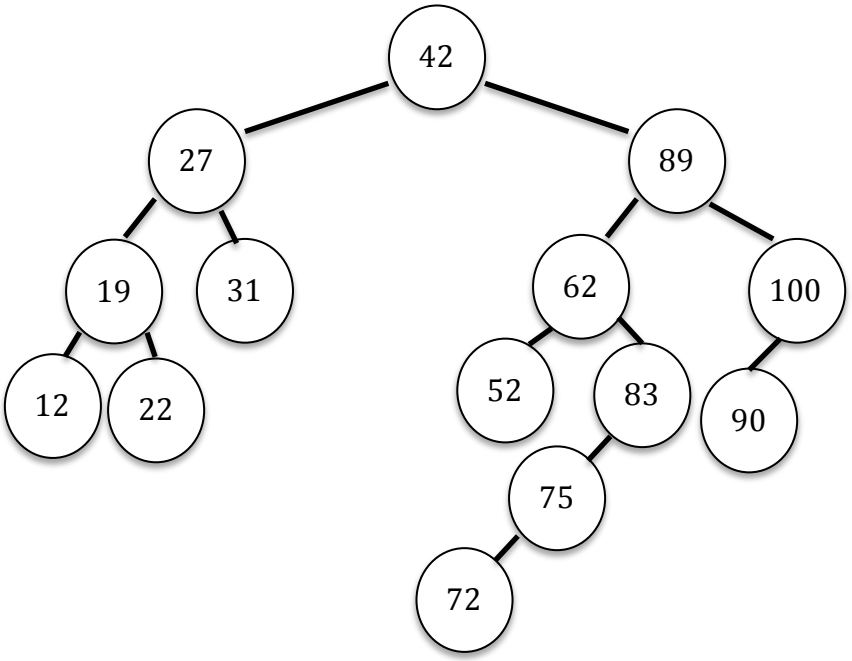
Inserting 12: 12 is smaller than 42, so the left child of 42 is checked. 27 is larger than 12, so the left child of 27 is checked. 19 is larger than 12 and it doesn't have a left child, so the left child of 19 becomes 12.



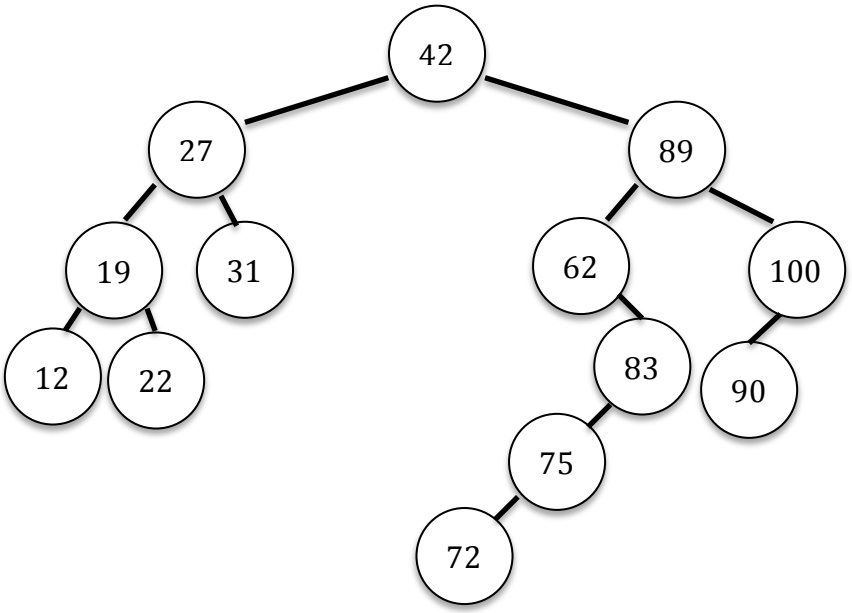
Inserting 72: 72 is larger than 42 and smaller than 42's right child 89, so 62 is checked. 62 is smaller than 72, so the right child of 62, 83 is checked. 83 is larger than 72, so left child of 83, 75 is checked, 75 is larger than 72, so the left child of 75 becomes 72.



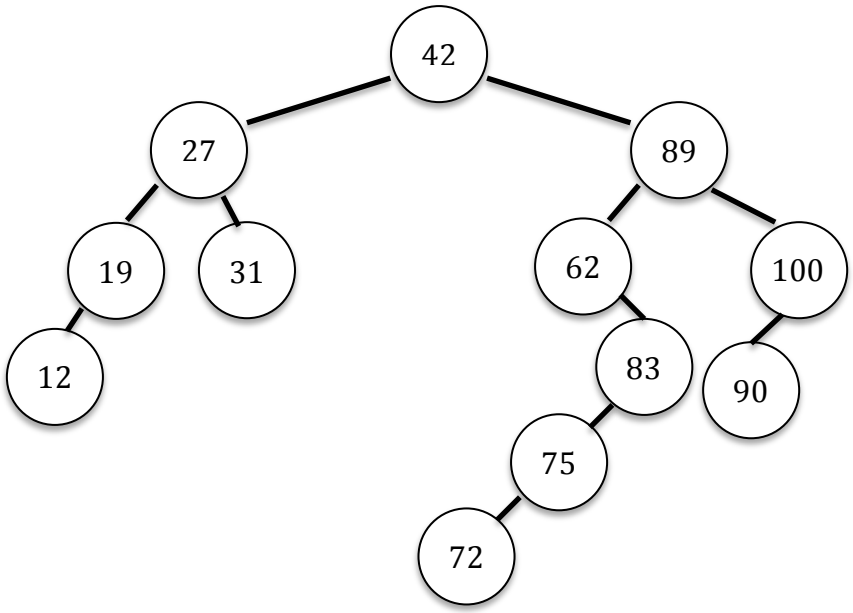
Inserting 90: 90 is larger than 42, also larger than right child of 42, 89, and smaller than 89's right child 100, which is a leaf node, so 90 becomes the left child of 100.



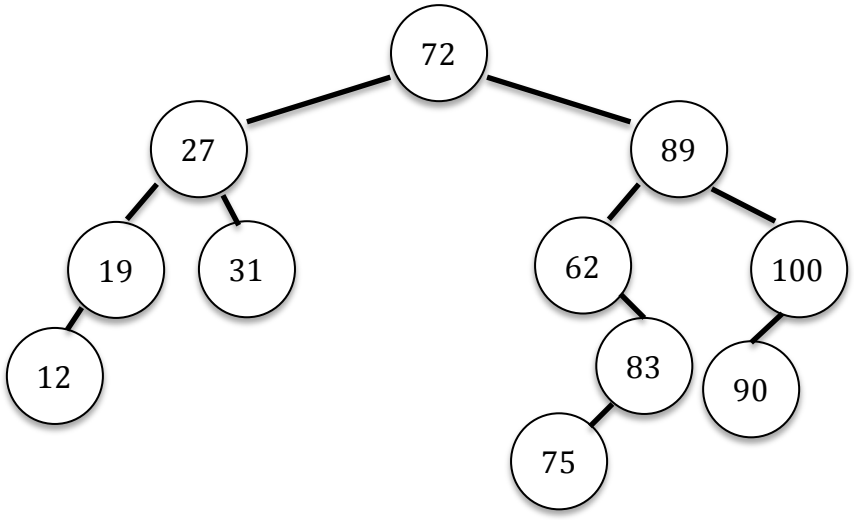
Delete 52: 52 is found and it is a leaf node, so it is directly deleted.



Deleting 22: 22 is a leaf node, so it is directly deleted.



Deleting 42: 42 is a node that has two children, so it is replaced with its inorder successor. Its inorder successor is the leftmost node in its right child, so 72. The remaining structure is a BST again.



3) Sample output of the program

```
Total 3-gram count: 15
aaa appears 1 time(s).
ccc appears 1 time(s).
ddd appears 1 time(s).
fff appears 1 time(s).
ggg appears 1 time(s).
hhh appears 1 time(s).
iii appears 1 time(s).
kkk appears 1 time(s).
lll appears 1 time(s).
mmm appears 1 time(s).
nnn appears 1 time(s).
sss appears 1 time(s).
ttt appears 1 time(s).
vvv appears 1 time(s).
zzz appears 1 time(s).
3-gram tree is complete: Yes

Total 3-gram count: 15

Total 3-gram count: 16
--- appears 1 time(s).
aaa appears 1 time(s).
ccc appears 1 time(s).
ddd appears 1 time(s).
fff appears 1 time(s).
ggg appears 1 time(s).
hhh appears 1 time(s).
iii appears 1 time(s).
kkk appears 1 time(s).
lll appears 1 time(s).
mmm appears 1 time(s).
nnn appears 1 time(s).
sss appears 1 time(s).
ttt appears 1 time(s).
vvv appears 1 time(s).
zzz appears 1 time(s).
3-gram tree is complete: Yes
3-gram tree is full: No
```

4) Time Complexity Analyses of addNgram and printNgramFrequencies Functions of the NgramTree Class

Time Complexity Analysis of addNgram method:

Considering the worst case to be a BST with only left or right children present (like a linked list), the insertion method would be linear, since the worst case would be adding the Ngram to the very end of the list.

T(n) = n + 10, where T(n) is the time complexity function and n is the number of nodes present in the BST.

Here, "10" is a constant that is obtained from all the comparisons and the assignments during the insertion, therefore we can infer that:

T(n) = n + c , where c is a constant and T(n) = O(n), where O(n) is the big Oh notation for the addNgram method of a BST made up of Ngrams.

Time Complexity Analysis of printNgramFrequencies method:

Considering the worst case to be a BST with only left or right children present (like a linked list), the printNgramFrequencies method would be linear, since the worst case would be linearly printing out the data of each node in a linear traversal.

T(n) = n + 3, where T(n) is the time complexity function and n is the number of nodes present in the BST.

Here, "3" is a constant that is obtained from the two comparisons and one "cout" statement while printing the node contents, therefore we can infer that:

T(n) = n + c, where c is a constant and T(n) = O(n), where O(n) is the big Oh notation for the printNgramFrequencies method of a BST made up of Ngrams.