

Title: Trees

Author: Tolga Han Arslan

ID: 22003061

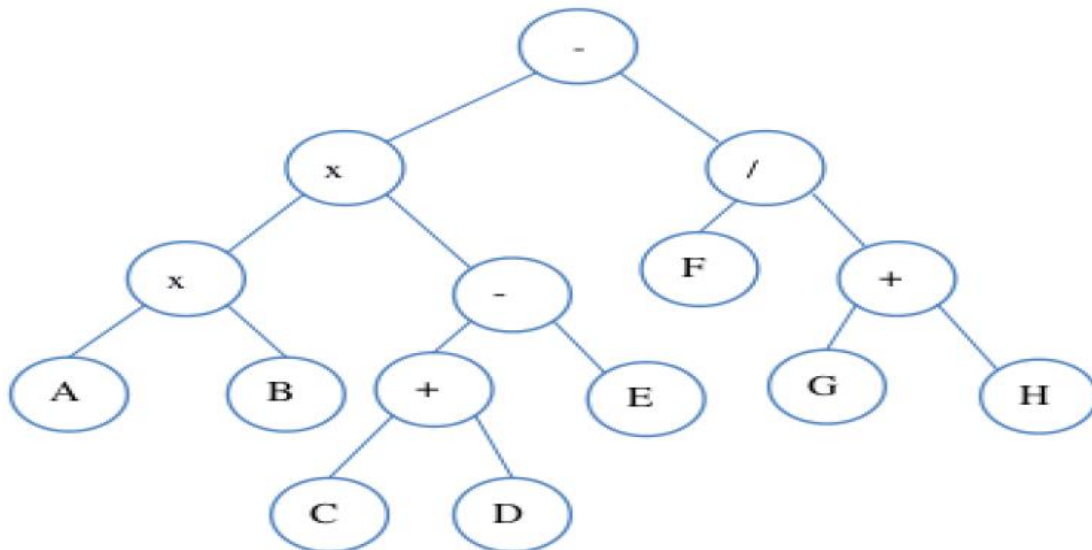
Section: 1

Assignment: 2

Description: HW2 Report

Question 1:

Give the prefix, infix, and postfix expressions obtained by preorder, inorder, and postorder traversals, respectively, for the expression tree below:



Question 1

Preorder Traversal (Prefix) = $r \rightarrow T_L \rightarrow T_R$
 $- x x A B - + C D E / F + G H$

Inorder Traversal (Infix) = $T_L \rightarrow r \rightarrow T_R$
 $A \times B \times C + D - E - F / G + H$

Postorder Traversal (Postfix) = $T_L \rightarrow T_R \rightarrow r$
 $A B \times C D + E - \times F G H + / -$

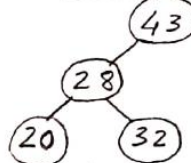
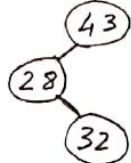
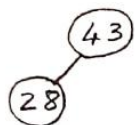
Question 2:

Draw the initially empty Binary Search Tree after operations as follows (show all intermediate steps):

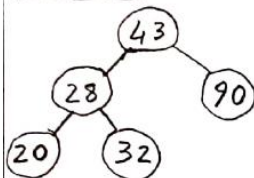
insert 43, 28, 32, 20, 90, 83, 101, 84, 23, 76, 53, 13, 73, 91; then delete 53, 23, 43.

Question 2 (Initially tree is empty)

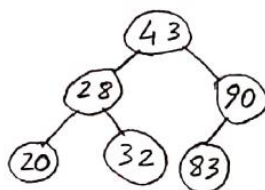
Insert 43 → Insert 28 → Insert 32 → Insert 20



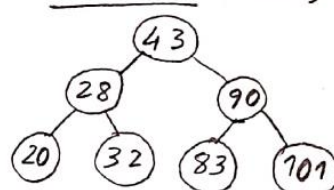
Insert 90



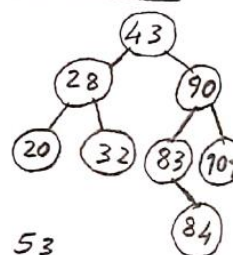
Insert 83



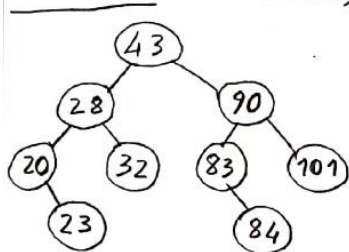
Insert 101



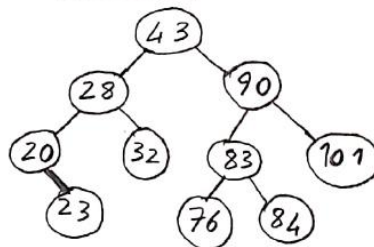
Insert 84



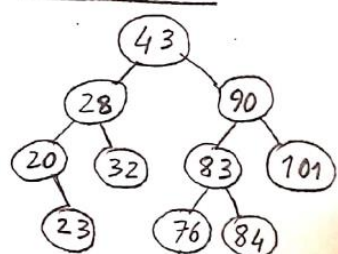
Insert 23



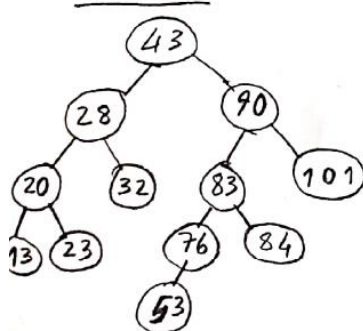
Insert 76



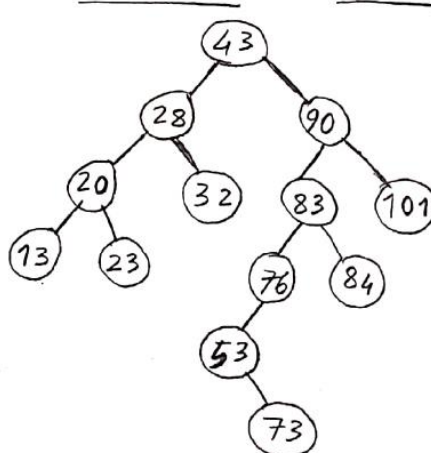
Insert 53



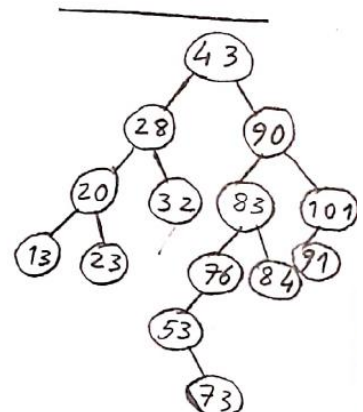
Insert 13



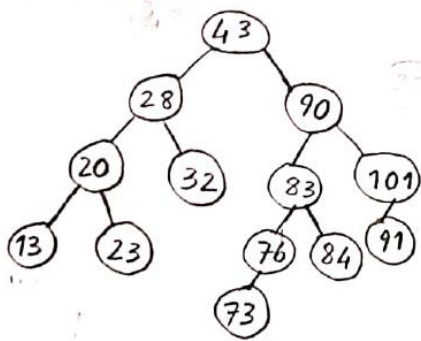
Insert 73



Insert 91

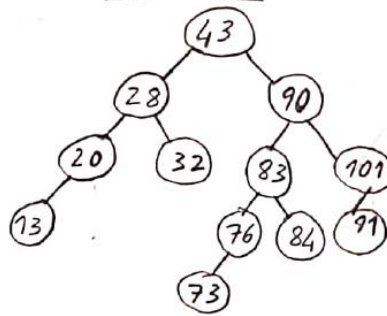


Delete 53 (one right child)



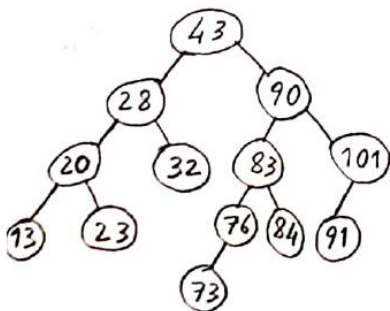
53 has only one right child (73).
73 replaces 53.

Delete 23 (No child)

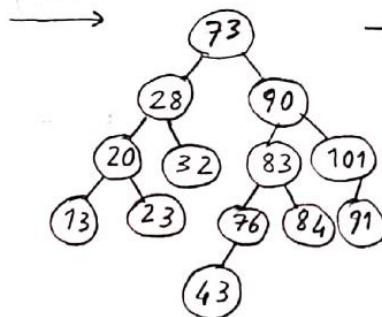


23 has no child (leaf).
No replacement needed.

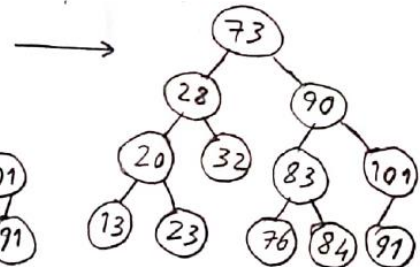
Delete 43 (Two child)



43 has two children.
Inorder successor = 73



73 replaces 43.



43 is deleted.

Question 3 (Sample output):

```
han.arslan@dijkstra:~  
Microsoft Windows [Version 10.0.19045.3208]  
(c) Microsoft Corporation. Tüm hakları saklıdır.  
  
C:\Users\USER>ssh han.arslan@dijkstra.ug.bcc.bilkent.edu.tr  
han.arslan@dijkstra.ug.bcc.bilkent.edu.tr's password:  
Activate the web console with: systemctl enable --now cockpit.socket  
  
Last login: Wed Jul 19 16:05:04 2023 from 10.201.182.217  
[han.arslan@dijkstra ~]$ ls  
hw2.cpp input.txt Makefile NgramTree.cpp NgramTree.h  
[han.arslan@dijkstra ~]$ make  
g++ -Wall -Wextra -std=c++11 -o hw2 NgramTree.cpp hw2.cpp  
hw2.cpp: In function 'int main(int, char**)':  
hw2.cpp:14:14: warning: unused parameter 'argc' [-Wunused-parameter]  
int main(int argc, char** argv) {           //command line arguments: input.txt 4 add them on dijkstra like ./a.out input.txt 4  
      ~~~~~^~~~~~  
[han.arslan@dijkstra ~]$ ls  
hw2 hw2.cpp input.txt Makefile NgramTree.cpp NgramTree.h  
[han.arslan@dijkstra ~]$ ./hw2 input.txt 4  
  
Total 4-gram count: 6  
"ampl" appears 1 time(s)  
"hise" appears 1 time(s)  
"mple" appears 1 time(s)  
"samp" appears 1 time(s)  
"text" appears 1 time(s)  
"this" appears 2 time(s)  
  
4-gram tree is complete: No  
4-gram tree is full: No  
  
Total 4-gram count: 6  
  
Total 4-gram count: 8  
"aatt" appears 1 time(s)  
"ampl" appears 1 time(s)  
"hise" appears 1 time(s)  
"mple" appears 1 time(s)  
"samp" appears 3 time(s)  
"text" appears 1 time(s)  
"this" appears 2 time(s)  
"zinc" appears 1 time(s)  
  
4-gram tree is complete: No  
4-gram tree is full: No  
[han.arslan@dijkstra ~]$
```

Question 4:

- **addNgram:**

```
void NgramTree::addNgram(const string& ngram ){
    TreeNode* node = nullptr;

    bool exist = retrieveItem(root, ngram, node);

    if(exist){
        node->count++;
    }

    else{
        insertItem(root, ngram);
    }

}
```

```
void NgramTree::insertItem(TreeNode *& treePtr, const string& newItem){

    if(treePtr == nullptr){
        treePtr = new TreeNode(newItem, nullptr, nullptr);
    }

    if(treePtr == nullptr){
        cout << "Insertion failed" << endl;
    }

    else if(newItem.compare(treePtr->item) < 0){ //search for left subtree

        insertItem(treePtr->leftChildPtr, newItem);
    }

    else{ //search for right subtree

        insertItem(treePtr->rightChildPtr, newItem);
    }

}
```

```

bool NgramTree::retrieveItem(TreeNode * treePtr, string ngram, TreeNode *& treeltem){

    if(treePtr == nullptr){

        return false;

    }

    else if(ngram.compare(treePtr->item) == 0){

        treeltem = treePtr;

        return true;

    }

    else if(ngram.compare(treePtr->item) < 0){

        return retrieveItem(treePtr->leftChildPtr, ngram, treeltem);

    }

    else{

        return retrieveItem(treePtr->rightChildPtr, ngram, treeltem);

    }

}

```

- ❖ addNgram function uses two helper functions insertItem and retrieveItem from the binary search tree class provided in the lecture slides. It first checks if the given ngram is present in the tree with retrieveItem, and do insertion or simple increment accordingly. Since both retrieveItem and insertItem are $O(n)$ algorithms in the worst case (when the tree's height is n , and therefore they both search all the nodes of the tree), addNgram is also $O(n)$ in the worst case.

- **operator<<:**

```
//prints each Ngram alphabetical order, uses inorder traversal

ostream& operator<<(ostream& out, const NgramTree& tree) {

    tree.printInorder(tree.root, out);

    return out;

}

void NgramTree::printInorder(TreeNode * treePtr, ostream& out) const{

    if(treePtr != nullptr){

        printInorder(treePtr->leftChildPtr, out);

        out << "\"" << treePtr->item << "\" " << "appears " << treePtr->count << " time(s)" << endl;

        printInorder(treePtr->rightChildPtr, out);

    }

}
```

- ❖ Operator<< uses helper function called printInorder to print the ngrams in the tree in alphabetical order. Since NgramTree is a binary search tree, inorder traversal will visit its nodes in sorted (alphabetical, in this case) search-key order. Since the traversal must visit all the nodes of the tree, best case, average case and worst case behaviour of inorder traversal is $O(n)$, the worst case behaviour of operator<< is also $O(n)$.