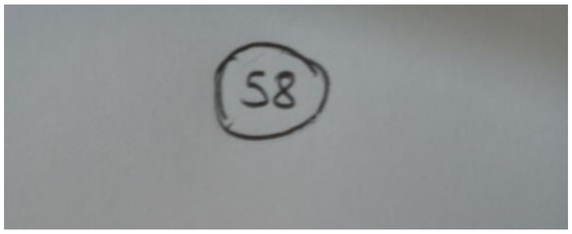
2019-2020 SPRING CS202 HW2

Rüzgar Ayan 21801984 Section 2

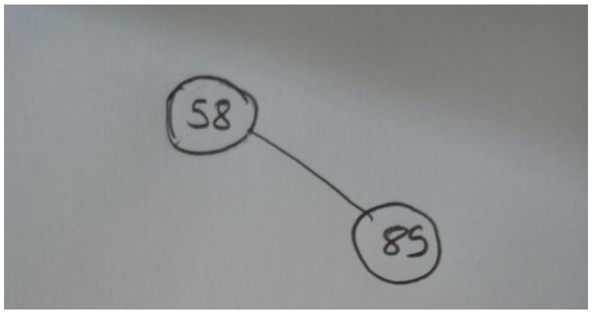
Question 1 – Explanation of Implementations

(a) [5 points] Insert 58, 85, 93, 24, 19, 44, 13, 57, 37, 94 into an empty binary search tree (BST) in the given order. Show the resulting BSTs after every insertion.

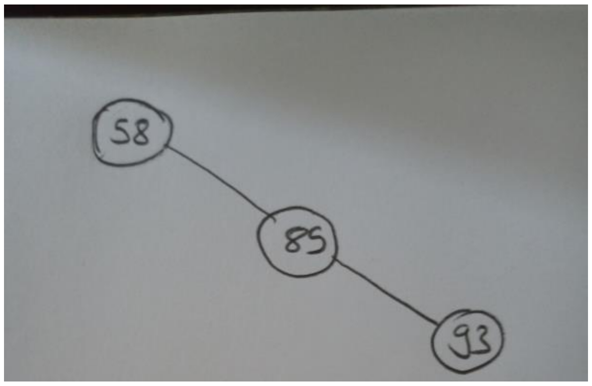
Insert 58



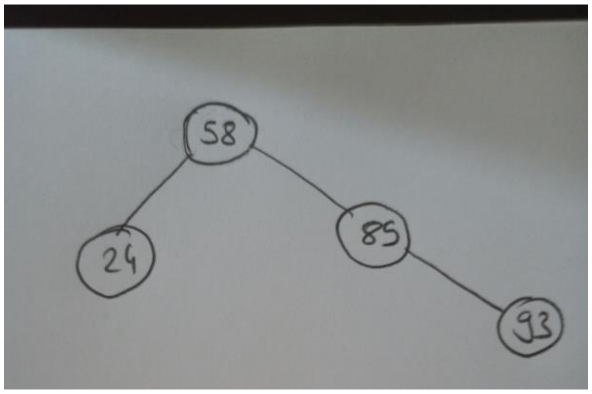
Insert 85



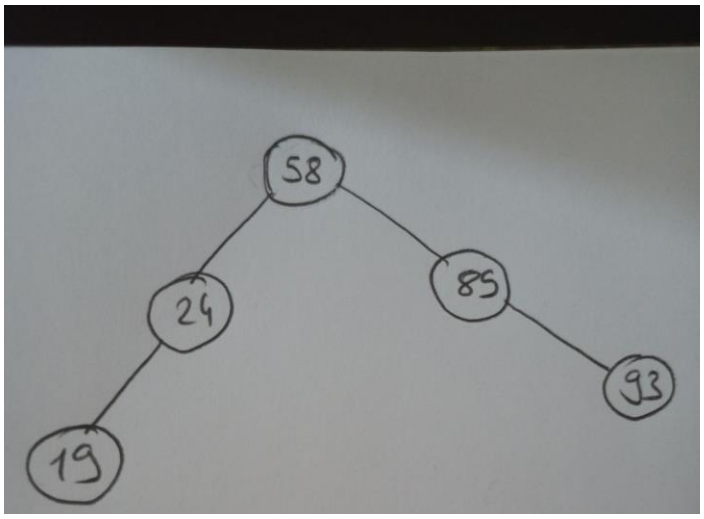
Insert 93



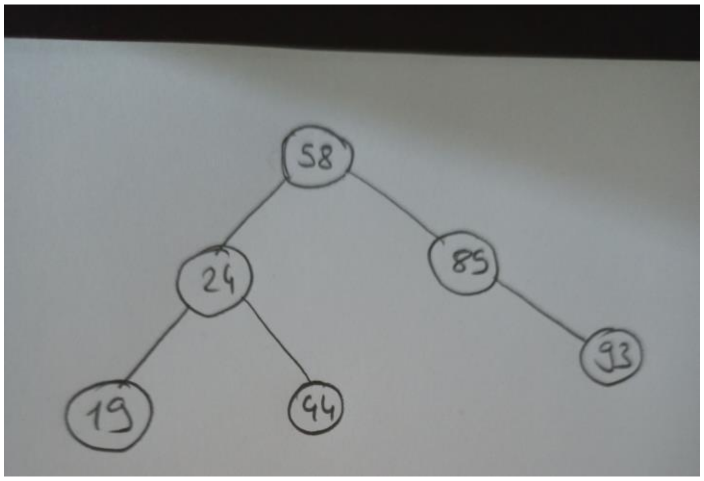
Insert 24



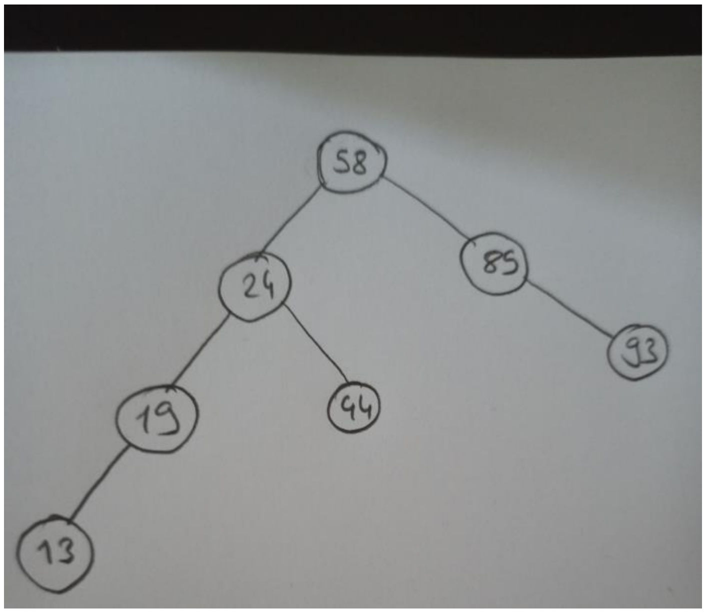
Insert 19



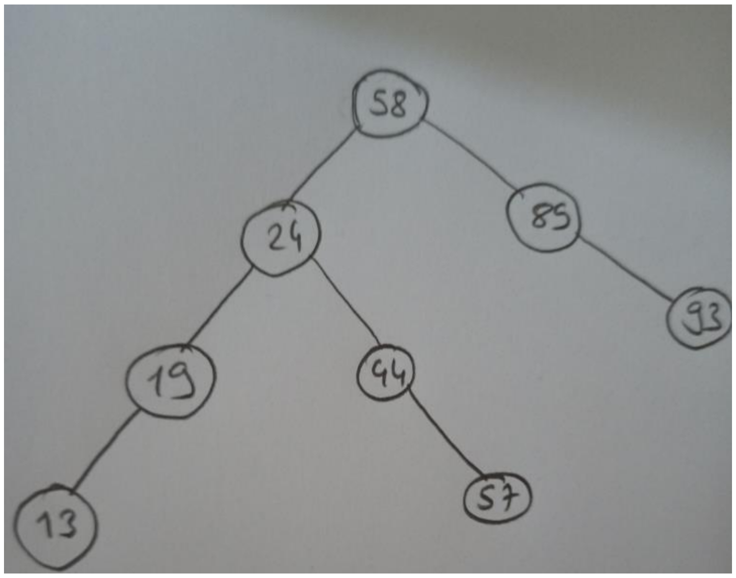
Insert 44



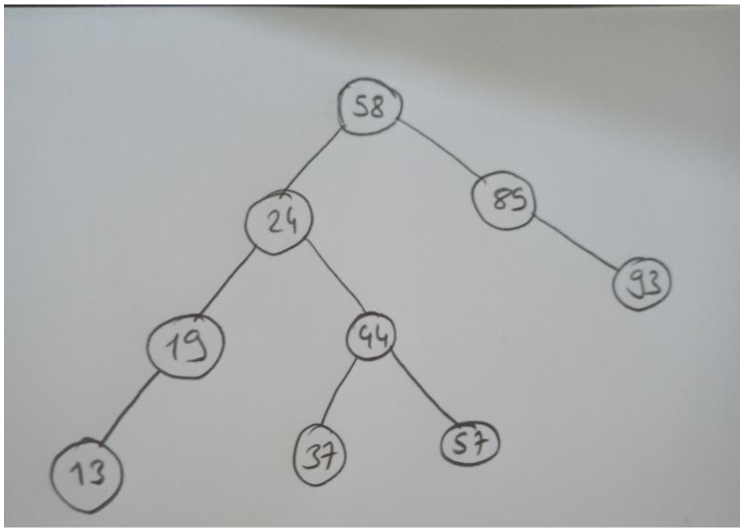
Insert 13



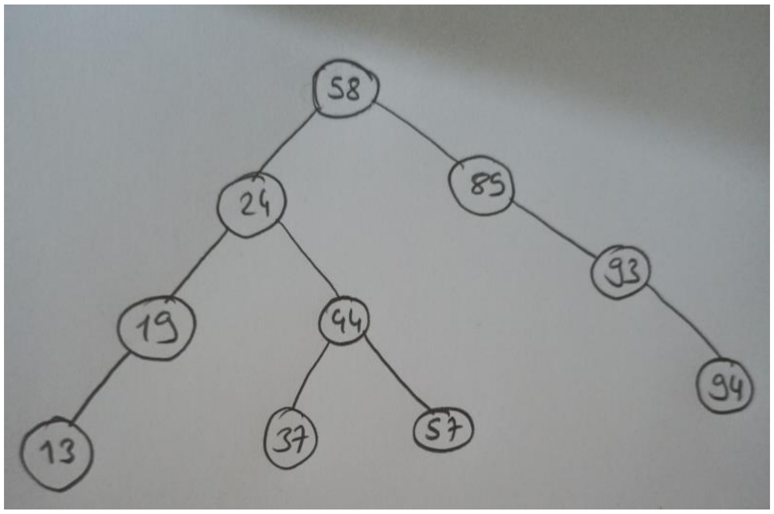
Insert 57



Insert 37



Insert 94



(b) [5 points] What are the preorder, inorder, and postorder traversals of the BST you have after (a)?

Preorder:

13 19 24 37 44 57 58 85 93 94

Inorder:

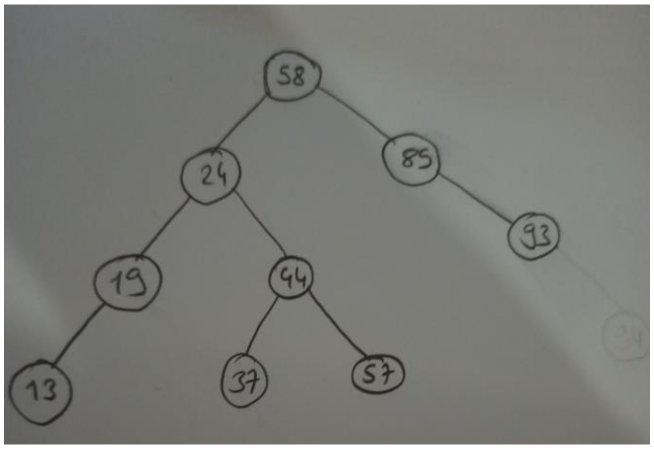
58 24 19 13 44 37 57 85 93 94

Postorder:

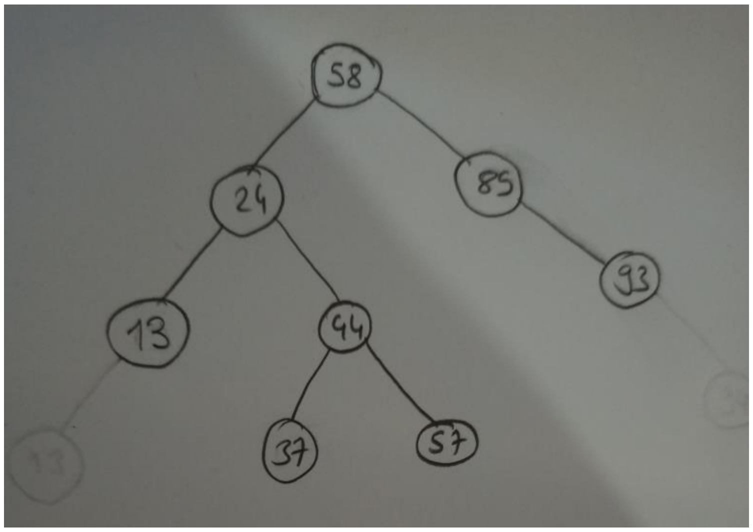
13 19 37 57 44 24 94 93 85 58

(c) [5 points] Delete 94, 19, 44, 24, 58 from the BST you have after (a) in the given order. Show the resulting BSTs after every deletion.

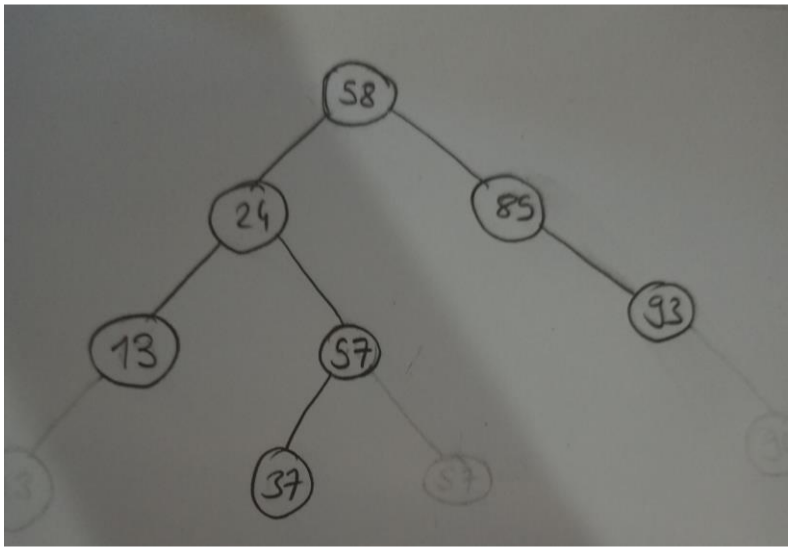
Delete 94



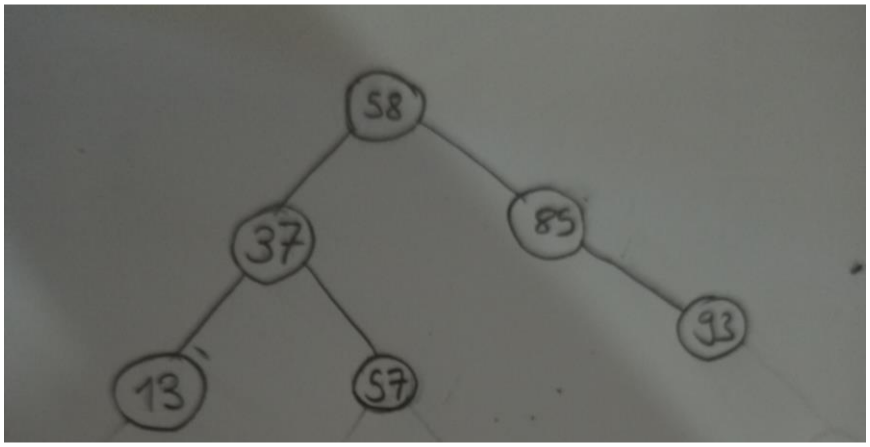
Delete 19



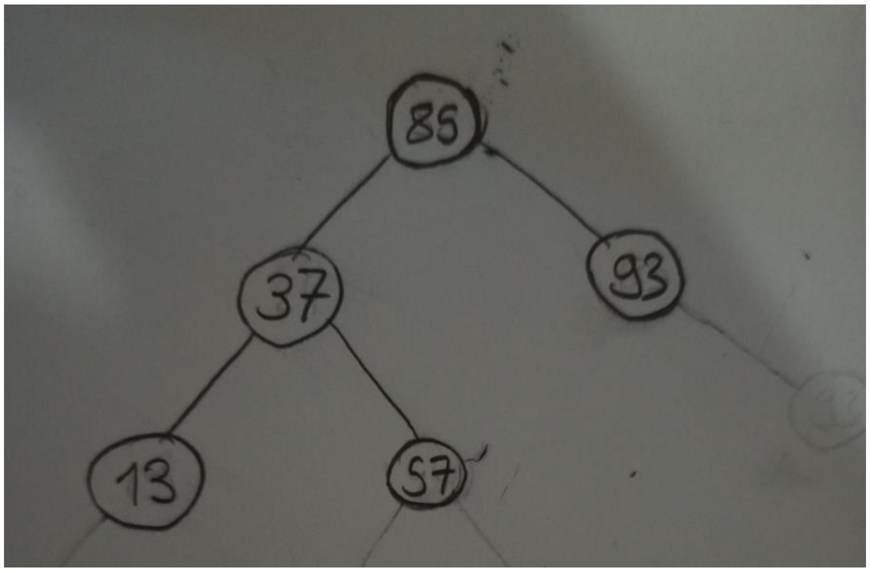
Delete 44



Delete 24



Delete 58



Question 3 – Explanation of Implementations

For the analysis of the time complexities of all the functions, let

- n denote the number of nodes in the decision tree,
- m denote the number of samples,
- k denote the number of features and
- c denote the number of classes.

1) calculateEntropy function

Implementation of this function is really simple. Using the definition of entropy given in the assignment pdf, we just sum P_i log P_i over all the classes. We are already given the number of samples in each class, so all we need is the total number of samples to find P_i's.

Time complexity is O(c) since we only have two separate for loops in the function and they both loop c times.

2) calculateInformationGain function

Unlike the above function, we are given the raw data and we need to process them before using the above function with them. We first determine the number of classes by finding the biggest value of class between the labels. Then we construct the classCounts array for the current node, left child and right child and then we use these with the above function to find the entropies of the current node, left child and right child. The final result just comes from combining these three entropies as in the definition of information gain given in the assignment pdf.

Finding the number of classes is O(m), constructing classCounts for each case is O(m) and we use the calculateEntropy function with time complexity O(c) 3 times. Then the time complexity of calculateInformationGain function is O(c+m). We can also assume that m>=c since it will be the case for almost all the given data, then O(c+m) becomes O(m).

3) DecisionTreeNode class

Only has a constructor and a destructor. Has the following data members:

- bool isLeaf => True if this is a leaf node, false if not.
- int featureIndex => Shows the feature that this node will use, meaningless if this is a leaf node.
- int classDecision => Shows the class number that this node will use, meaningless if this is not a leaf node.
- DecisionTreeNode* leftChild, rightChild => The pointers to the left and right children.

Constructor simply gives these data members their initial values. Destructor deletes the child nodes.

4) DecisionTree class

This class contains the most important functions of the implementation. Has the following data members:

- DecisionTreeNode* root => The pointer that points to the root node of the decision tree.
- int numFeatures => Number of features, will be given its value during the training and will be used in testing.

Constructor just initializes these data members and destructor deletes the root node. From the implementation of the destructor of DecisionTreeNode, all the other nodes under the root are deleted recursively. Therefore the time complexity of constructor is O(1) and the time complexity of the destructor is O(n).

5) train function (from data)

Set the numFeatures data member to the value given as parameter to this function. We start by constructing two arrays, usedFeatures which will indicate the features already used up until that node and usedSamples which will indicate the samples that will be used in that node. We initialize all the elements of usedFeatures to false and usedSamples to true and call a recursive version of train function on the root node with these two arrays. To simply explain, this recursive function will split into two children and call itself for these children until it reaches a leaf node. This will construct all of the decision tree starting from the root.

In the beginning of this recursive function, we check if the given node will be a leaf or not. There are two possibilities for it to be a leaf node. We first check if the node is pure and if it is we call a helper function makePureLeaf. For the second possibility, we check if there any features left to use and if there aren't any left to use we call the helper function makeNonPureLeaf that finds the most frequent class in the node and sets it as the node's classDecision. If one of these happens, then we know that the node is leaf and then we can return to stop the recursion.

Otherwise, we know that the node will make a split and we should now choose the best possible split. We call the calculateInformationGain function for each unused feature and find the split with the highest information gain. We make a split with the chosen feature and create the two child nodes for the current node. Then we have to construct two new usedSamples arrays for each child node, so we construct these arrays and also construct a new usedFeatures array with the currently chosen one excluded. With these new constructed arrays and the new nodes we call the function itself recursively for each child.

The time complexity of this function is O(n*k*m). In the function, the part which has the dominant time complexity is finding the optimal split. It uses calculateInformationGain function with all unused features. Although the number of features and the number of samples that will be used in calculateInformationGain will be decreasing as we go lower in the tree, at maximum they will be k and m respectively. As we know calculateInformationGain is O(m) and we use it k times for each of the n nodes, we can give the upper bound for the time complexity as O(n*k*m).

6) train function (from file)

Not much to say for this one, it constructs two arrays “bool** data” and “int* labels” which are used in the original train function. It then calls the original train function with these parameters. If we exclude the time required for the original train function, the time complexity of just parsing the data is O(m*k) since the total number of reads from the training file will be m*(k+1) where m is the number of samples and k is the number of features.

7) predict function

Returns -1 if the decision tree is not trained yet. Otherwise, starting from the root, goes the right or left child according to the feature index of the current node and the data given to predict. It does this in a while loop until it reaches a leaf node. The time complexity is O(k) since the height of the three is at most k.

8) test function (from data)

Returns -1 if the decision tree is not trained yet. Otherwise uses predict function with all the given test samples and then returns the ratio of correct predictions to the total number of test samples. The time complexity is O(k*t) where t is the number of test samples.

9) test function (from file)

Returns -1 if the decision tree is not trained yet. Otherwise, it constructs two arrays “bool** data” and “int* labels” which are used in the original test function. It uses the “int numFeatures” data member of the decision tree object in order to parse the right amount of features for each sample from the test text file. The time complexity of just parsing is O(t*k) where t is the number of test samples.

10)print function

Simply does a recursive preorder traversal of the decision tree. If the current node is not a leaf, it first calls itself for the left child, then prints the data in the current node by putting tabs in front of it and printing “feature=” and lastly calls itself for the right child. If the current node is a leaf, it just prints the data in the node by again putting the right amount of tabs in front of it and then printing “class=”.

The number of tabs we print for a node is between 0 and k. But since the decision tree is not likely to be a complete binary tree it is not easy to find a good upper bound. Therefore, I will give the simple upper bound O(n*k) since we go over all the nodes recursively and do 0 to k operations in each of these nodes.