CS1102: Data Structure and Algorithms

Tutorial 8: Trees (Solutions)

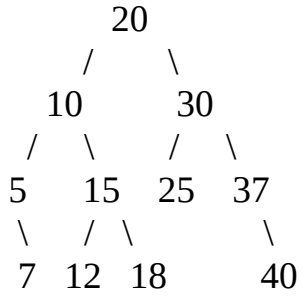
Week of 22 March 2010

1. Binary Search Tree Operations:

   (a) Draw the binary search tree after inserting the following integers in sequence: 20, 10, 30, 5, 7, 25, 15, 37, 12, 18 and 40.
   (b) What are the corresponding output using (i) pre-order, (ii) in-order, (iii) post-order, (iv) level-order traversal?
   (c) Draw the new binary search tree after the following operations: delete 30, delete 10, delete15 (You should follow the deletion algorithm in the lecture notes). Is the new tree a complete binary tree?

Answer:

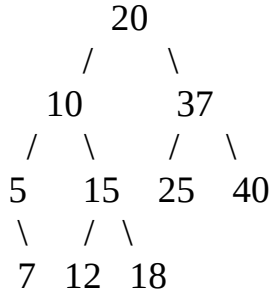
(a) The binary search tree constructed is:



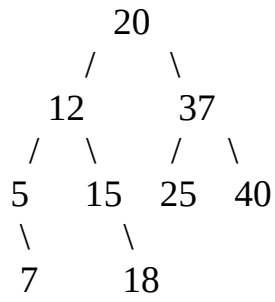
(b) Pre-order: 20, 10, 5, 7, 15, 12, 18, 30, 25, 37, 40
In-order: 5, 7, 10, 12, 15, 18, 20, 25, 30, 37, 40
Post-order: 7, 5, 12, 18, 15, 10, 25, 40, 37, 30, 20
Level-order: 20, 10, 30, 5, 15, 25, 37, 7, 12, 18, 40

(c) The evolution of the binary search tree:

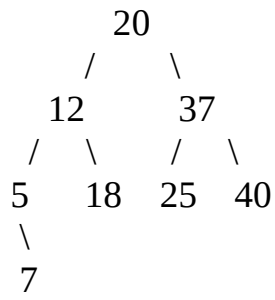
Delete 30:
According to the lecture notes, we move the smallest node (37) in the right subtree to the position of the deleted node. The move of the node 37 implies its deletion also.



Delete 10:



Delete 15:



The new tree is not a complete binary tree because 7 is the right child of 5.

CS1102: Data Structure and Algorithms

2. Suppose you have a **binary tree** T containing only distinct items. The pre-order sequence of T is 8, 7, 5, 28, 4, 9, 18, 17, 16, and the in-order sequence of T is 5, 7, 4, 28, 9, 8, 18, 16, and 17.

   (a) Can you reconstruct a unique T from these two traversal sequences? If yes, reconstruct it.
   (b) Implement the algorithm if the above construction is possible.

```
TreeNode constructTree(int[] inOrder, int[] preOrder, int n)
{
    //n is the number of distinct items
}
```

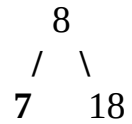
   (c) If you are given the pre-order and post-order sequences of a binary tree, can you construct a unique binary tree? Explain your answer.

Answer:

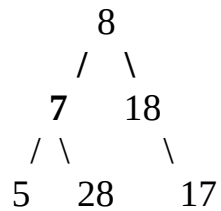
(a) Yes.

i. From the pre-order sequence, we know the root of the binary tree is 8, and from the in-order sequence, we know that the nodes of the left sub-tree of 8 are 5, 7, 4, 28, 9, and the nodes of right sub-tree of 8 are 18, 16, 17. Note that the pre-order sequence of the left sub-tree of 8 is 7, 5, 28, 4, 9, and the pre-order right sub-tree of 8 is 18, 17, 16.

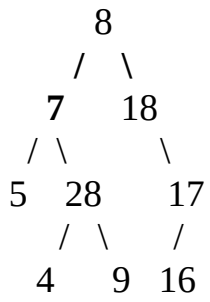
ii. Apply the similar method to the left and right sub-trees of 8, so the roots of the left and right sub-trees of 8 are 7 and 18 respectively. So, the first two levels of the binary tree so far constructed are:



iii. Continue to construct the sub-trees of 7 and 18, the first three levels of the binary tree constructed are:



iv. Continue to construct the sub-trees of 5, 28, and 7. The final binary tree constructed is:

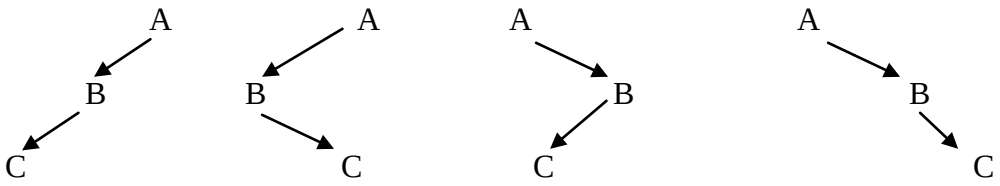


(b)

```
TreeNode constructTree(int[] inOrder, int[] preOrder, int n)
{
      //n is the number of distinct items
      //Base case, when there is only 1 item, it's trivial
      if(n == 1)
            return new TreeNode(inOrder[0]);

      //Recursive case
      //Root is always the first element of the preOrder array
      int element = preOrder[0];
      TreeNode root = new TreeNode(element);

      //Determine the location of the element in the inOrder
      //array
      for (int i=0; i<n; i++)
            if (inOrder[i] == preOrder[0]) {
                inorderIndex = i;
                break;
            }

      //We know that all elements before the inorderIndex belong
      //to the left subtree and all elements after the
      //inorderindex belong to the right subtree. With that we
      //can determine the corresponding elements that are
      //in the left and right subtrees (since number of elements
      //in both inorder and preorder arrays should be the same)
      //Determine the left and right halves
      int[] preorderLeft = copyArray(preOrder, 1, inorderIndex);
      int[] preorderRight = copyArray(preOrder, inorderIndex+1, n-1);

      int[] inorderLeft = copyArray(inOrder, 0, inorderIndex-1);
      int[] inorderRight = copyArray(inOrder, inorderIndex+1, n-1);

      root.left = constructTree(inorderLeft, preorderLeft,
                        inorderIndex);
      root.right = constructTree(inorderRight, preorderRight,
                        n - (inorderIndex+1));
      return root;
}
int [] copyArray(int[] arr, int start, int end)
{
      int[] result = new int[end-start+1];
      for(int i=0; i<end-start+1; i++)
          result[i] = arr[start++];
      return result;
}
```

Question: Can we solve the problem without creating any new arrays? If yes, how do you modify the program /algorithm?

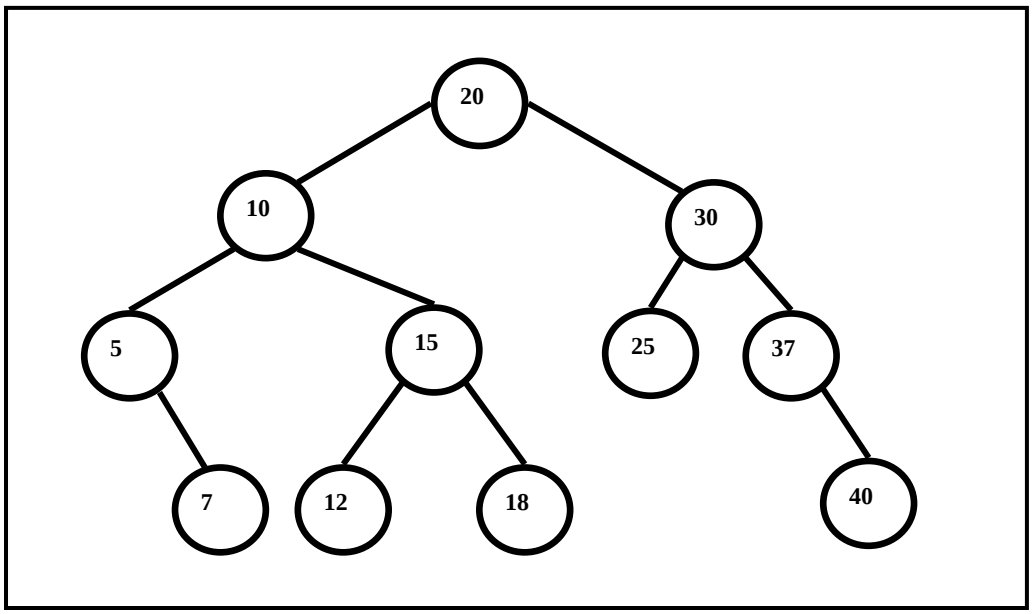
(c) No. for example, pre-order = A B C and Post-order = C B A can produce the following trees:



CS1102: Data Structure and Algorithms

3. Another way of representing a binary search tree is to use an array. The items in the tree are assigned to locations in the array in a level-order fashion. For example, the figure below shows an array representation of the binary search tree in question 1.
   We assume there is no repeating value in the tree and there is no node with value 0, so we use 0 to represent non-existing node.

Value	20	10	30	5	15	25	37	0	7	12	18	0	0	0	40
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14




```
public class BinaryTree {
	int[] tree;
	int MAXNUM=15;

	public BinaryTree(){
		tree= new int[MAXNUM];
	}
	public void insert(int i)
	public int search(int i)
	//return the index of integer i, -1 if it is not in the BST
	public int findMin()
	//return the minimum value in the tree
}
```

(a) Implement the recursive insertion method of binary search tree.
(b) Implement the recursive search method of binary search tree.
(c) Implement the finding minimum method of binary search tree.
(d) What are the advantages and disadvantages of array representation?

Answer:

(a) To implement a recursive insertion algorithm, we can use method overloading to create another recursive method insert (int i, int index).
If the index value is larger than the size of the tree, we create a new index with size=2*size+1 and copy the value from the original tree.
The size of new tree is 2*size+1 because the number of nodes in a complete tree of level n is

$$size(n) = 1 + 2 + 2^2 + 2^3 + \cdots + 2^{n-1} = 2^n - 1$$
$$= 2 * size(n-1) + 1$$

In the base case, the method changes the value of the node is set to the insertion value i if it is empty. If the insertion integer is smaller than the value of the node, the method inserts the integer to the left subtree. Otherwise, it will be inserted into the right subtree.

```
public void insert(int i){
	insert(i, 0);
}

public void insert(int i, int index){
	//when the index is larger than array size
	//enlarge the array size
	if(index>MAXNUM) enlargeArr();
	//base case
	if(tree[index]==0) tree[index]=i;
	//recursive case
	else if(i<tree[index]) insert(i, 2*index+1);
	else insert(i, 2*index+2);
}

private void enlargeArr(){
	// The new size is 2*maxnum+1 which is equal to the size
	// of the full complete binary tree with height increased
	// by 1
	MAXNUM=2*MAXNUM+1;
	//create a new array with copied value
	int[] newTree = new int[MAXNUM];
	for (int j=0; j<tree.length; j++){
		newTree[j]=tree[j];
	}
	tree=newTree;
}
```

(b) The recursive search method is an overloading method of search method. The method returns -1 (no such value) if the index is larger than the maximum index of array. It returns the index of the node if the value of the node equals to the searching value.
In the recursive case, if the searching value is smaller than the value of the node, the method recursively searches it in the left subtree. Otherwise, the method will search it in the right subtree.

```
public int search(int i)
//return the index of integer i, -1 if it is not in the BST
{
      return search(i, 0);
}

public int search(int i, int index){
      //base case
      if(index>MAXNUM || tree[index]==0) return -1;
      else if(tree[index]==i) return index;
      //recursive case
      else if(tree[index]>i)  return search(i, 2*index+1);
      else return search(i, 2*index+2);
}
```

(c) In the findMin method, it keeps looking for the left child. If the left child is a leaf, it is the minimum in the tree.

```
public int findMin()
//return the minimum value in the tree
{
      int index=0;
      while(index<MAXNUM/2 && tree[2*index+1]!=0){
            index=2*index+1;
      }
      return tree[index];
}
```

(d) Doing level order transversal using array representation of BST is fast. Retrieving values in an array is also fast. The parent, children and siblings can be easily found in array representation of tree by dong simple calculation. There is no need to use extra memory space to save references to another node in an array. Array representation of tree is good for complete BST.

However, the storage utilization of this approach is very low if the BST is not close to a full binary tree. Moreover, since the size of array is fixed, if the tree grows beyond the size, the program needs to create another array with size $=$ size $\times 2 + 1$ and copy the values into the new array. In the worst case where the tree is inserted in descending order, the required size of array is $2^{\text{maxnum}} + 1$.

Note that we have not mentioned the node deletion method. This method is not easy to implement.

4. When deciphering ancient languages, historians often take advantage of the fact that some words are more commonly repeated than others, for example the word "I" or "We" are often repeated more than others in the English language. Hence, this idea can be used to decipher words in long texts. For example, the long text "CATTCAT" will produce the following possible words:
   Length 1: C (2 times), A (2 times), T (3 times)
   Length 2: CA (2 times), AT (2 times), TT (1 time), TC (1 time)
   Length 3: CAT (2 times), ATT (1 time), TTC (1 time), TCA (1 time)
   Length 4: …
   Length 5: …
   Length 6: CATTCA (1 time), ATTCAT (1 time)
   Length 7: CATTCAT (1 time).

   (a) Given a long string S, design an ADT to facilitate the query of the occurrence of all the possible words of length $l \leq$ length(S). (Just write down the required method headers, no need to write down their implementations) In the above example, the occurrence of "CAT" is equal to 2.

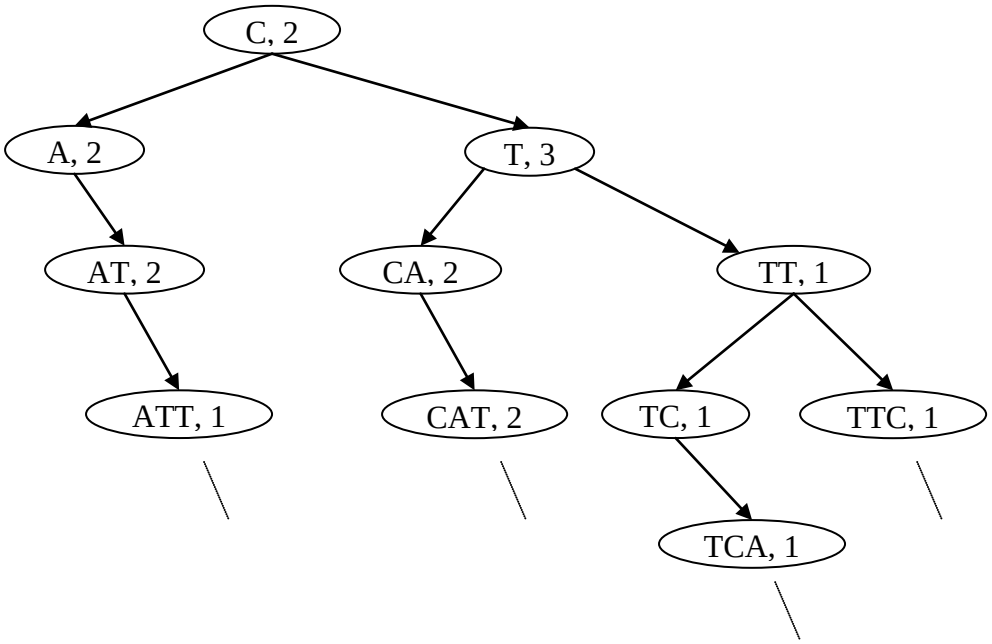
   (b) Describe the underlying data structure you would use to implement this ADT, and analyze the complexity of each method you designed in the ADT. With your data structure, is the time complexity of getting the word occurrence dependent on the number of possible words *n*? If yes, can you think of a data structure that makes the query of word occurrence independent of *n*? You may use diagrams to explain your solution. You may not use Hash tables or maps.

   Answer:

   (a)

```
class MultiSet
{
   // construct a data structure to store all the
   //possible words
   public MultiSet() { … }

   // add words to the data structure
   void addWord(String word) {…}

   // get the occurrence of a certain word
   int getWordOccurrence(String word) {…}
}
```

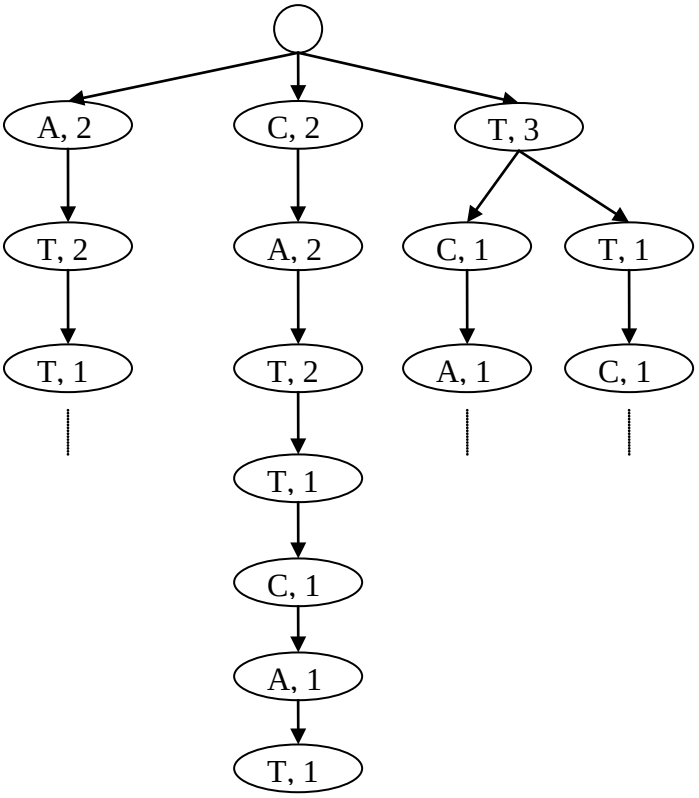
(b)
Most intuitively, a BST would be used. In particular, each treeNode will contain one possible word and its occurrence number. A partial tree constructed is shown below:



In this case, since there are *n* possible words, a balanced binary search tree (e.g. AVL tree) would require log(*n*) for each addWord operation as well as for each getWordOccurrence operation. The total complexity of initializing the tree is O(*n log n*).

To allow for complexity independent of *n*, we use an n-ary tree, with each letter representing a node. The child of each node is the letter that succeeds the previous letter in a word. For example in the word "CAT" the child of node C is A, and the child of node A is T. Each node also stores the number of occurrence of the word with this letter as the last letter.

For example: "CATTCAT" will produce the tree –



The height of the tree is at most $l + 1$ (dummy head) where l is the length of the input String. Creating the tree by calling the addWord method (for each word in the text) is O(l). The getWordOccurrence operation is also O(l), which is independent of n. The total complexity of initializing the tree is O(nl) since n words have to be inserted.

Question: Can we represent/store an n-ary tree using a binary tree?