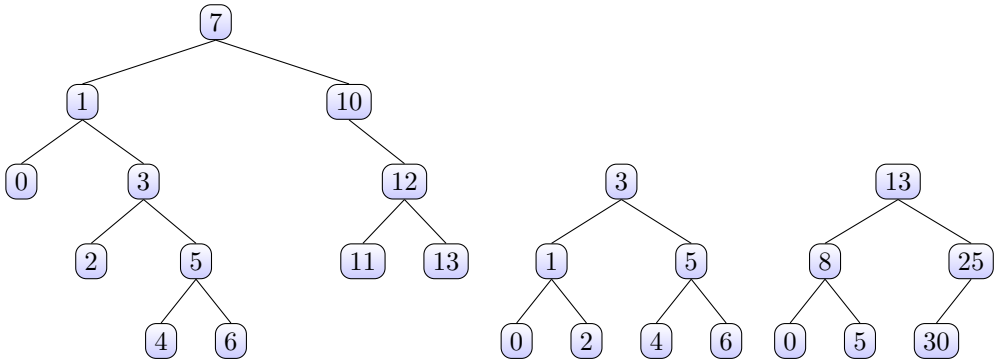
CS 284: Exercise Booklet - Trees

Solutions to selected exercises ($\diamond$) are provided at the end of this document. Important: You should first try solving them before looking at the solutions. You will otherwise learn **nothing**.

1 Binary Tree Expressions

Exercise 1.



(a) (b) (c)

1. Indicate which are full, perfect and/or complete.
2. Indicate their height.
3. List the ancestors of the node with label 11 in tree (a).
4. Indicate, for each, how many levels it has.
5. Indicate which are Binary Search Trees.
6. Every binary tree of height h has $2^h - 1$ nodes. True or false?

Exercise 2. Write the Tree Expression for each of the three trees of the previous exercise.

Exercise 3. Prove, by induction on the height h, that a perfect binary tree has $2^h - 1$ nodes.

Exercise 4. Given a perfect binary tree of height h, derive an expression $f(n)$ for the height as a function of the number of nodes n.

Exercise 5. Prove that the number of leaves in a perfect tree of height h is 2^{h-1}.

Exercise 6. From the two previous exercises, derive the number of leaves l in a perfect binary tree as a function of the number of nodes n.

Exercise 7. Prove that the number of nodes in a full binary tree of height h is between 2^{h-1} and $2^h - 1$.

2 Binary Trees

Exercise 8. Implement the following operations:

- `public int height()`
- `public int no_of_nodes()`
- `public boolean is_leaf()`
- `public int no_of_leaves()`
- `public boolean is_full()`
- `public boolean is_balanced()`. A binary tree is said to be *balanced* if both of its subtrees are balanced and the height of its left subtree differs from the height of its right subtree by at most 1.
- `public boolean is_perfect()`

Exercise 9. (◇) Implement `public BinaryTree<E> cloneAt(Node<E> current)`, that returns a new copy of the tree whose root is the current node.

Exercise 10. Implement the following operations:

- `public BinaryTree<E> mirror()`
- `public void prune(int level)`. For example, `prune(0)` should yield the empty tree. Also, `prune(h)`, for `h` the height of the tree, should not modify the tree.

Exercise 11. (◇) A path in a binary tree is a sequence of 0s (left) and 1s (right) that leads to a node in the tree. Implement the following operations:

- `public ArrayList<ArrayList<Integer>> paths()`. Eg. for the empty tree it should return `[[]]`. For a tree such as (preorder):

```
7
  1
    2
      null
      null
    3
      null
      null
  null
```

  it should produce `[[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1], [1]]`.

- `public ArrayList<ArrayList<Integer>> paths2()`. Similar to previous exercise only that paths should stop at the leaves. Eg. for the example above it should return it should produce `[[0, 0], [0, 1]]`.

Exercise 12. Define an operation `public ArrayList<E> ancestors(E item)` that returns the list of all the ancestors of `item`. For example, in the tree (a) of Exercise 1, given 5, it should return `[7,1,3]`. If the item is not in the tree, then an exception `NoSuchElementException` should be thrown.

Exercise 13. Define an operation `public ArrayList<E> bfs()` that returns the list of all the items in the tree, by levels. For example, in the tree (b) of Exercise 1, it should return `[3,1,5,0,2,4,6]`.

Exercise 14. Define an operation `public ArrayList<ArrayList<E>> bfs2()` that returns the list of all the items in the tree, by levels, where each level is a list itself. For example, in the tree (b) of Exercise 1, it should return `[[3],[1,5],[0,2,4,6]]`.

3 Binary Search Trees

Exercise 15. Draw what a binary search tree would look like if the following numbers were inserted: 50 20 75 98 80 31 150 39 23 11 77.

Exercise 16. Consider the assertion: "Adding an element to a BST always increments its height". True or false? Justify your answer

Exercise 17. What's wrong with the following code for adding an element to a BST?

```
public boolean add(E item) {
    root = add(root, item);
    return addReturn;
}

private Node<E> add(Node<E> localRoot, E item) {
   if (localRoot == null) {
       // item is not in the tree, insert it.
       addReturn = true;
       return new Node<E>(item);
   } else if (item.compareTo(localRoot.data) == 0) {
       // item is equal to localRoot.data
       addReturn = false;
       return localRoot;
   } else if (item.compareTo(localRoot.data) < 0) {
       // item is less than localRoot.data
       add(localRoot.left, item);
       return localRoot;
   } else {
       // item is greater than localRoot.data
       add(localRoot.right, item);
       return localRoot;
   }
}
```

Exercise 18. (◇) Define a method in the class `BinaryTree<E>` that returns a list with all the subtrees located at a given depth. Here are the two methods you should add to the above mentioned class, one of which you have to complete.

```
private ArrayList<BinaryTree<E>> projectLevel(Node<E> localRoot, int l) {
  // Complete here
}
```

```
4
public ArrayList<BinaryTree<E>> projectLevel(int l) {
6   return projectLevel(this.root,l);
}
```

For example, if we use the binary tree from the file `Fig_6_12.txt` then the output of

```
FileReader fin = new FileReader("Fig_6_12.txt");
2 Scanner src = new Scanner(fin);
BinaryTree<String> tree = BinaryTree.readBinaryTree(src);
4 System.out.println(tree);
System.out.println(tree.projectLevel(0));
6 System.out.println(tree.projectLevel(1));
System.out.println(tree.projectLevel(2));
8 System.out.println(tree.projectLevel(3));
System.out.println(tree.projectLevel(4));
```

would be

```
1 3
    7
3     null
      null
5   8
      6
7       null
        0
9         null
          null
11    null

13 [3
    8
15    null
      6
17      0
          null
19        null
        null
21  7
      null
23    null
]
25 [8
    null
27  6
      0
29      null
        null
31    null
, 7
33  null
    null
35 ]
   [6
37  0
      null
39    null
    null
41 ]
   [0
```

```
  null
  null
]
[]
```

Exercise 19. (◇) Use the previous exercise and the height of a tree to implement the following operation, to be included in `BinaryTree<E>`, that returns a list with all the subtrees of a binary tree:

```
public ArrayList<BinaryTree<E>> st() {
  // Complete here
}
```

Sample output for `System.out.println(tree.st());` from the tree of the previous exercise would be:

```
[3
  8
    null
    6
      0
        null
        null
      null
  7
    null
    null
, 8
  null
  6
    0
      null
      null
    null
, 7
  null
  null
, 6
  0
    null
    null
  null
, 0
  null
  null
]
```

Exercise 20. Suppose we have int values between 1 and 1000 in a BST and search for 363. Which of the following cannot be the sequence of keys examined.

1. 2 252 401 398 330 363
2. 399 387 219 266 382 381 278 363
3. 3 923 220 911 244 898 258 362 363
4. 4 924 278 347 621 299 392 358 363
5. 5 925 202 910 245 363

Exercise 21. Draw the BST resulting from removing the following elements from the tree given in the slides (slide 19, third set of slides on trees):

1. kept
2. cow

4 Answers to Selected Exercises

Answer to exercise 9

```
private BTree<E> cloneAt(Node<E> current) {
  if (current==null) {
          return new BTree<E>();
  }
  if (current.isLeaf()) {
          return new BTree<E>(current.data);
  }
  return new BTree<E>(current.data,cloneAt(current.left),cloneAt(current.right));
}
```

Answer to exercise 11

```
private ArrayList<ArrayList<Integer>> paths(Node<E> lr) {
  if (lr==null) {
          return new ArrayList<ArrayList<Integer>>();  // []
  } else if (lr.isLeaf()) {
          ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>>();
          result.add(new ArrayList<Integer>());
          return result; // [[]]
  } else  {
          ArrayList<ArrayList<Integer>> resultRight = paths(lr.right);
          for (ArrayList<Integer> l :resultRight) {
                  l.add(0,1);
          }
          ArrayList<ArrayList<Integer>> resultLeft = paths(lr.left);
          for (ArrayList<Integer> l :resultLeft) {
                  l.add(0,0);
          }
          resultLeft.addAll(resultRight);
          return resultLeft;

  }
}

public ArrayList<ArrayList<Integer>> paths() {
  return paths(root);

}
```

Answer to exercise 18

```
private ArrayList<BTree<E>> projectLevel(int i, Node<E> current) {
        ArrayList<BTree<E>> r = new ArrayList<BTree<E>>();

        if (i==0 && current!=null) {  // found the designated level
```

```
            r.add(cloneAt(current));
        } else {
            if (current!=null)
            {
                r.addAll(projectLevel(i-1,current.left));
                r.addAll(projectLevel(i-1,current.right));
            }
        }
        return r;
    }

    public ArrayList<BTree<E>> projectLevel(int i) {
        return projectLevel(i,root);
    }
```

Answer to exercise 19

```
public ArrayList<BTree<E>> st() {
    ArrayList<BTree<E>> r = new ArrayList<BTree<E>>();
    for (int i = 0; i<height(); i++) {
            r.addAll(projectLevel(i));
    }
    return r;
}
```

References

- http://www.cs.washington.edu/143/
- http://www.cs.princeton.edu/courses/archive/fall13/cos126/precepts/BSTex.pd