COMP2521 20T2 Data Structures and Algorithms

Week 04 Tutorial

(Balanced) Search Trees

For all tree types, we assume the following definitions:

```
typedef struct Node *Link;  // Links are pointers to Nodes
typedef struct Node *Tree;  // a tree is a pointer to its root Node
```

Assume a basic binary search tree node structure as:

```
struct Node { int data; Link left; Link right; };
```

For an AVL tree, the basic Node structure is augmented by

```
struct Node { int data; int height; Link left; Link right; };
```

For a red-black tree, the basic Node structure is augmented by

```
typedef enum {RED, BLACK} Colour;
struct Node { int data; Colour colour; Link left; Link right; };
```

For a 2-3-4 tree, a different Node structure is used:

```
struct Node { int order; int data[3]; Link child[4]; };
```

Use the above type definitions in answer the questions below. The types of Nodes in the tree should be clear from the context of the question.

Try to answer all questions without resorting to using the `tlab` program from the online sessions, or any of the algorithm animation pages that you can find on the web. Use these only to check your answers after you attempt the question.

1. What characteristic distinguishes a *search* tree form other kinds of tree structures (e.g. a taxonomy)?

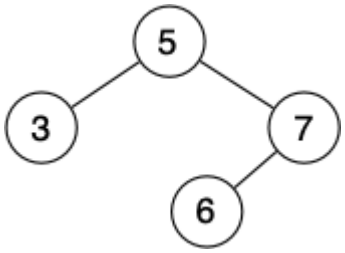
2. Implement the following function that **prints the height difference** between the left subtree and the right subtree **of every node** in a given basic binary tree, (`Tree t`). The function returns the height of the given binary tree. Use the following function interface:

   ```
   int printHeightDiff (Tree t) { ... }
   ```

   The function should print the value in the root node before printing the height difference, e.g.

   ```
   data: 3, diff: 0
   ```

```
data: 6, diff: 0
data: 7, diff: 1
data: 5, diff: -1
```



The nodes should be printed in LRN (postfix) order.

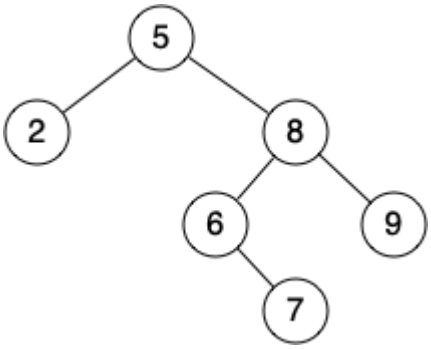
3. Implement the following function that returns the height of a given basic binary tree, if the given binary tree is a height balanced tree. Otherwise, if the given binary tree is not an a height-balanced tree, the function returns `NOT_HEIGHT_BALANCED`, which is `#define`'d as -99.

   Height-Balanced Tree: we say that a basic binary tree is a height balanced tree if for every node, the absolute difference between the height of the left subtree and the height of the right subtree is one or less. In other words, every node needs to satisfy the following criteria:
   `abs(height(left) - height(right)) ≤ 1`

   Use the following function interface:

   ```
   int isHeightBalanced (Tree t) { ... }
   ```

4. Show the sequence of rotations that would be required to move the 7 node to the root of this binary tree:



   Show the state of the tree after each rotation.

5. (Splay trees)

   a. Show how a Splay tree would be constructed if the following values were inserted into an initially empty tree in the order given:

   ```
   5 3 8 7 4
   ```

   b. Let `t` be your answer to part (a), and consider the following sequence of operations:

   ```
   SearchSplay (t, 7);
   SearchSplay (t, 8);
   SearchSplay (t, 6);
   ```

Show the tree after each operation.

6. (AVL trees)

   Answer the following question without the help of the `treeLab` program from the lecture.

   Show how an AVL tree would be constructed if the following values were inserted into an initially empty tree in the order given:

   ```
   12 10 8 6 4 2
   ```

7. Splay tree search moves the found node (or a near neighbour) to the root. AVL trees do not do this. What are the advantages/disadvantages of each strategyt?

8. (2-3-4 trees)

   Show how a 2-3-4 tree would be constructed if the following values were inserted into an initially empty tree in the order given:

   ```
   1 2 3 4 5 8 6 7 9 10
   ```

   Once you have built the tree, count the number of comparisons needed to search for each of the following values in the tree:

   ```
   1  7  9  13
   ```

9. Repeat the previous question, but this time use a red-black tree.

   Hint: translate the 2-3-4 trees from the previous question to red-black trees. Alternative hint: show the intermediate states during the insertion of one item and use the algorithm from the slides to see how the tree is transformed after the new item is added as a leaf.
