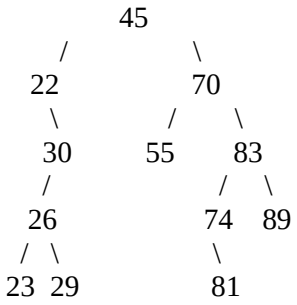
COP 3502 Quiz #4 Version B (Binary Search Trees, AVL Trees, Tries) Solutions

1) (6 pts) Provide the Preorder, Inorder and Postorder traversals of the following binary tree:



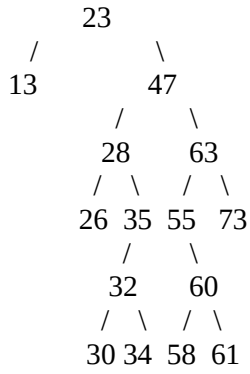
Pre-Order: **45, 22, 30, 26, 23, 29, 70, 55, 83, 74, 81, 89**

In-Order: **22, 23, 26, 29, 30, 45, 55, 70, 74, 81, 83, 89**

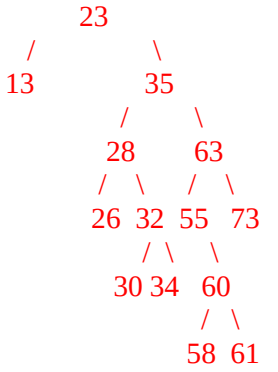
Post-Order: **23, 29, 26, 30, 22, 55, 81, 74, 89, 83, 70, 45**

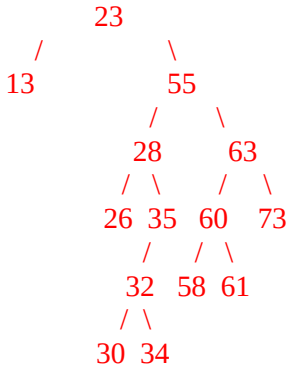
Grading: 2 pts for each correct traversal, 1 pt if more than 1/2 the values are right, if answers are all traversals but named incorrectly (they put pre-order for post-order), then -2 total.

2) (5 pts) Show the result of deleting 47 from the binary search tree shown below. (Note: There are two possible right answers.)



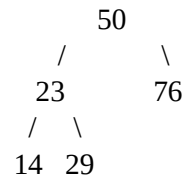
We can either replace 47 with the max on its right (35), or the min on its left (55). Here is the result of both (so both correct answers):





Grading: 2 pts valid replacement, 2 pts for patch below, 1 pt for copying rest

3) (9 pts) Write a **recursive** function which takes in a pointer to a binary search tree node and returns the number of leaf nodes stored in the tree. For example, if root was pointing to the node storing the 50 in the binary search tree below, your function should return 3, since the nodes storing 14, 29 and 76 are all leaf nodes. Please use the struct and function prototype given below:

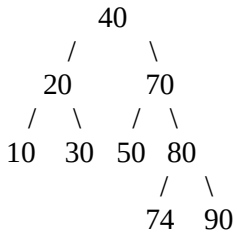



```
typedef struct bintreenode {
    int data;
    struct bintreenode* left;
    struct bintreenode* right;
} bintreenode;
```

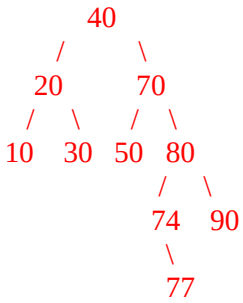
```
int countLeafNodes(bintreenode* root) {

    if (root == NULL) return 0;
    if (root->left == NULL && root->right == NULL) return 1;
    return countLeafNodes(root->left) + countLeafNodes(root->right);

}
```

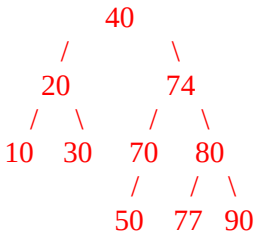
Grading: 2 pts NULL case, 3 pts single node case, 1 pt return, 1 pt each rec call, 1 pt adding

4) (5 pts) Show the result of inserting the value 77 into the AVL tree below. Put a box around your final answer.



When we do this, we get an imbalance at 70, the rebalancing (and final answer) is on the right:





Grading: 5 pts correct answer, 2 pts if shows insertion without any rebalance, partial if rebalance is attempted but incorrect (so give either 3 or 4 based on your discretion)

5) (10 pts) We define the number of differences between two equal length strings to be the number of corresponding characters that are different. (More formally, given two strings s[0..n-1] and t[0..n-1], the number of differences between the strings are the number of values of i in the range [0,n-1] such that s[i] ≠ t[i].) For example, the number of differences between "house" and "horse" is 1, because the strings are only different at their third character (i=2).

Write a function, which takes in a pointer to a trie node, **`root`**, a string, **`word`**, an integer, **`k`,** and another integer, **`numerrors`**, and does the following:

Returns the number of strings in the trie rooted at **`root`** that have **exactly `numerrors`** differences with the substring **`word[k..strlen(word)-1]`** and is exactly of the same length as **`word[k..strlen(word)-1]`.** (Note: This length is strlen(word)-k.)

Hint: As you are traversing down the trie, you can try each of 26 letters "to be the next letter." One of these choices will not reduce the number of errors you can make in the future, but the other 25 choices will reduce the number of errors you can make by 1.

```
typedef struct trie {
    int isWord;
    struct trie* next[26];
} trie;

int matchesWithErrors(trie* root, char* word, int k, int numerrors) {

  if (numerrors < 0) return 0;
  if (root == NULL) return 0;
  if (k == strlen(word)) return numerrors == 0 && root->isWord;

  int res = 0;
  for (int i=0; i<26; i++) {
    if ((word[k]-'a') == i)
      res += matchesWithErrors(root->next[i], word, k+1, numerrors);
    else
      res += matchesWithErrors(root->next[i], word, k+1, numerrors-1);
  }

  return res;
}
```

Grading: 1 pt for root NULL case,
2 pts for k == strlen case,
1 pt loop
1 pt accumulator,
1 pt check if letter is equal in loop,
1 pt rec call match,
2 pts rec call with an error
1 pt return