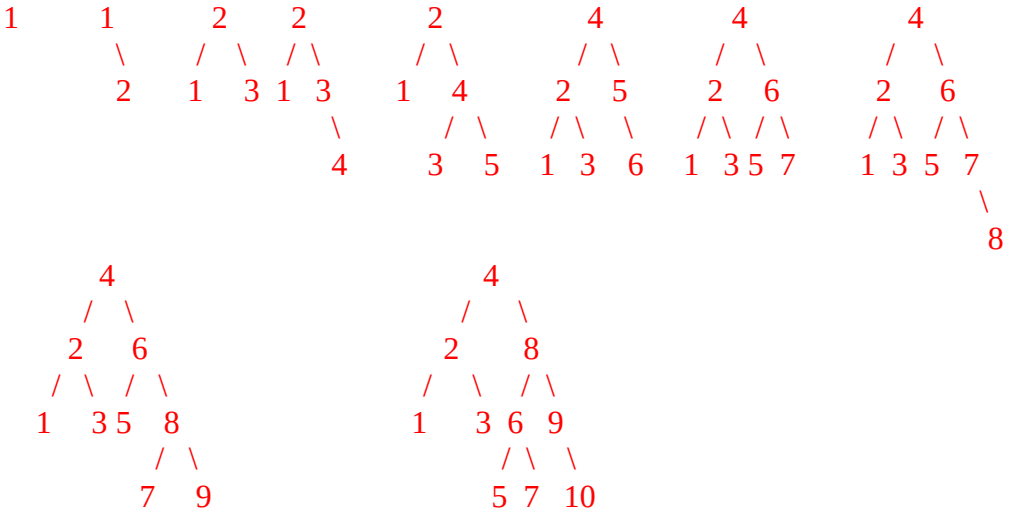
COP 3502 Study Group Sheet: AVL Trees, Tries Solution

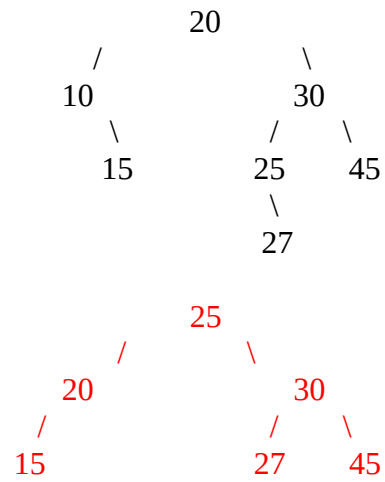
1) Show the result of inserting the following items into an initially empty AVL Tree:

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

Here is the status of the tree after each insertion



2) Show the result of deleting 10 from the AVL Tree shown below:



Deleting 10 forces the 15 to be the child of 20, which then forces a rebalance at 20, putting 25 at the root.

3) Edit the function insert in the file in this link:

www.cs.ucf.edu/~dmarino/ucf/transparency/cop3502/sampleprogs/mytrie.c

to adapt to this struct:

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

where sumWords will store the total number of valid words that are stored at or below that particular node in the trie. (For example, in a trie with two words: "cat" and "cab", the node storing the 'c' and 'a' would store 2 for sumWords while the nodes storing the 't' and 'b' will store 1 for sumWords.

```
void insert(TrieNode* tree, char word[], int k, int wordlen) {

    tree->numWords++;

    if (k == wordlen) {
        tree->flag = 1;
        return;
    }

    int nextIndex = word[k] - 'a';
    if (tree->children[nextIndex] == NULL)
        tree->children[nextIndex] = init();

    insert(tree->children[nextIndex], word, k+1, wordlen);
}
```

Note: only the line in bold has to be added to the original insert function.

4) Write a function that takes in a pointer to the root node of a trie (guaranteed not to be NULL) and prints out each word in the trie. The function should also take in a string that stores the current answer built up, as well as the current depth in the trie. Here is the function prototype:

```
void printRec(struct trie* root, char* word, int k);
```

Here is the wrapper function:

```
void print(struct trie* root) {

    char* word = malloc(sizeof(char)*1000000);
    word[0] = '\0';
    printRec(root, word, 0);
    free(word);
}
```

We assume all words in the trie are less than 999,999 characters!

```
void printRec(struct trie* root, char* word, int k) {

    if (root == NULL) return;

    if (root->isWord) {
        word[k] = '\0';
        printf("%s\n", word);
    }

    for (int i=0; i<26; i++) {
        word[k] = (char)('a'+i);
        printRec(root->next[i], word, k+1);
    }

}
```

Note: This assumes the use of the struct from the file mytrie.c.