Refactor

The functions that I will be refactoring is the insert function. However, the addEachkey function works with the insert function. The first thing I will refactor is the handling of root case for the insert function. Below is the initial code for insert function that handles the case without root and addEachfunction that adds node to the tree down word. From line 36 to line 45, I realized that I was trying to do the work that was or suppose to be done in addEachKey. Since when there's no root, I am supposed to add the nodes downwards. The three things I did in this block of code were point root node to a newNode, increment the index and check whether or not the word is a single letter. However, it was doing the same thing addEachKey does in each of its iteration. I figured if my addEachKey has a condition where root is checked if it exists, I can just pass this root condition to addEachKey.

```
15  void DictionaryTrie::addEachKey(string word, unsigned
      int freq, TrieNode* curr, int index)
16  {
17    for(unsigned int i = index; i<word.length();i++){
18      TrieNode* newNode = new TrieNode(word[i]);
19      curr->child = newNode;
20      curr=curr->child;
21      if(i == word.length()-1){
22        curr->freq=freq;
23        curr->lword=true;
24      }
25    }
26  }
27  /* Insert a word with its frequency into the dictionar
    .
28   * Return true if the word was inserted, and false if
     it
29   * was not (i.e. it was already in the dictionary or
     it was
30   * invalid (empty string) */
31  bool DictionaryTrie::insert(string word, unsigned int
      freq)
32  {
33    unsigned int index=0;
34    char letter = word[index];
35    if(!root){
36      root = new TrieNode(word[index]);
37      index++;
38      TrieNode* curr=root;
39      if (index == word.length()) {
40        curr->lword = true;
41        curr->freq = freq;
42      }
43      else {
44        addEachKey(word, freq, curr, index);
45      }
46      return true;
```

```
15  void DictionaryTrie::addEachKey(string word, unsigned
      int freq, TrieNode* curr, int index)
16  {
17    if(!root){
18      root = new TrieNode(word[index]);
19      curr = root;
20
21      if(index == word.length()-1){
22        curr->freq=freq;
23        curr->lword=true;
24      }
25      index++;
26    }
27    for(unsigned int i = index; i<word.length();i++){
28      TrieNode* newNode = new TrieNode(word[i]);
29      curr->child = newNode;
30      curr=curr->child;
31      if(i == word.length()-1){
32        curr->freq=freq;
33        curr->lword=true;
34      }
35    }
36  }
37  /* Insert a word with its frequency into the dictionary
    .
38   * Return true if the word was inserted, and false if
     it
39   * was not (i.e. it was already in the dictionary or
     it was
40   * invalid (empty string) */
41  bool DictionaryTrie::insert(string word, unsigned int
      freq)
42  {
43    unsigned int index=0;
44    char letter = word[index];
45    TrieNode* curr = root;
46
47    if(!root){
48      addEachKey(word, freq, root, index);
49      return true;
```

In the second figure, you can see that I edited my addEachKey to handle two new cases. First, if there's no root, I need to point that root to a new node. Second, if index is equal to word.length()-1 it means the string that we're trying to add to the tree is a single node, and the for loop below won't run. Therefore I need to give it frequency and indicator of word node.

Another change I made to insert is the cases within the while loop. Each case when curr->left or curr->right doesn't exist, it has to check whether or not the character is the last node. Because I have to repeat this process for when left child or right child doesn't exist, I figured it will be more efficient to add another case in addEachKey because it is going to be called anyway.

New case line 27 to 30

```
void DictionaryTrie::addEachKey(string word, unsigned
  int freq, TrieNode* curr, int index)
{
  if(!root){
    root = new TrieNode(word[index]);
    curr = root;

    if(index == word.length()-1){
      curr->freq=freq;
      curr->lword=true;
    }
    index++;
  }
  if(index == word.length()){
      curr->freq=freq;
      curr->lword=true;
  }
  for(unsigned int i = index; i<word.length();i++){
    TrieNode* newNode = new TrieNode(word[i]);
    curr->child = newNode;
    curr=curr->child;
    if(i == word.length()-1){
      curr->freq=freq;
      curr->lword=true;
    }
  }
}
```

Old

```
52      if(curr->label > letter){
53        if(curr->left){
54          curr=curr->left;
55        }
56        else{
57          TrieNode* newNode = new TrieNode(word[index])
58          index++;
59          curr->left = newNode;
60          curr = curr->left;
61          if(index == word.length()){
62            curr->lword=true;
63            curr->freq=freq;
64          }
65          else{
66              addEachKey(word, freq, curr, index);
67          }
68          return true;
69        }
70      }
```

New

```
57      if(curr->label > letter){
58        if(curr->left){
59          curr=curr->left;
60        }
61        else{
62          TrieNode* newNode = new TrieNode(word[index]);
63          curr->left = newNode;
64          curr=curr->left;
65          index++;
66          addEachKey(word, freq, curr, index);
67          return true;
68        }
69      }
```

Besides the insert function I removed the getFreq in the class because I realized it was only for debugging purposes.