Binary trees

Implement the lexicon class to represent text vocabularies in binary format search trees. The class should support the following functions, which you should implement:

```
class lexicon {
public:
  lexicon();
  ~lexicon();

  void insert(const string &s);
  int  lookup(const string &s) const;
  int  depth(const string &s);
  void replace(const string &s1, const string &s2);

  friend ostream & operator << (ostream &out, const lexicon &l);
};
```

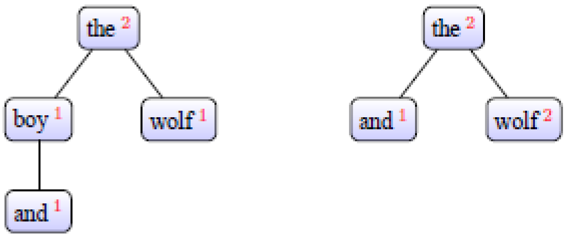
Assume that the words stored in a dictionary will not be empty and will be they contain only lowercase Latin letters. Each word will correspond to a node in the tree. The node will contain the word and its frequency in the dictionary (ie a counter that will remember how many times this word has been entered). The left child of the node will contain alphabetically (dictionary-graphically) smaller words, while the right child alphabetically (lexicographically) larger. No word should not appear in more than one node in the tree. Also, the tree will should be a simple BST: not required and should not be balanced (eg, AVL).
The insert (s) method will insert the word s into the tree. The lookup method (s) will look for the word s in the tree and return the occurrence frequency. The result will be 0 (zero) if the word does not exist in the tree. The depth (s) method will look for the word s in the tree, as will lookup. If it does not exist, it will returns 0 (zero). But if it exists, it will return the depth at which the node that holds it is contains in the tree, ie the length of the path from the root to this node. We suppose that the root of the tree is at a depth of 1 (one). The replace method (s1, s2) will replace all occurrences of the word s1 with an equal number of occurrences of the word s2. If s1 does not exist in the tree, then nothing will happen. If it exists and the number of its occurrences is k> 0, then s1 will be deleted and the word s2 will be searched. If not, it will be inserted with an occurrence frequency of k. But if there is, then the frequency of its occurrence will updated accordingly (will increase by k).
When deleting a word from the dictionary (done indirectly by the replace method), if a node with two non-empty children gets deleted, then the deleted node is replaced by the one containing the immediately smaller word. If a node with a non-empty child is deleted, then it is replaced by its child.
Dictionaries should be printed in alphabetical order, with the words followed by the frequency of their occurrence. You can test your implementation with programs like the following:

```
int main() {
  lexicon l;
  l.insert("the");
  l.insert("boy");
  l.insert("and");
  l.insert("the");
  l.insert("wolf");
  cout << "The word 'the' is found " << l.lookup("the") << " time(s)" << endl;
  cout << "The word 'and' is found at depth " << l.depth("and") << endl;
  cout << l;
  l.replace("boy", "wolf");
  cout << "After replacement:\n";
  cout << l;
  cout << "Now the word 'and' is found at depth " << l.depth("and") << endl;
}
```

Its execution should display:

```
The word 'the' is found 2 time(s)
The word 'and' is found at depth 3
and 1
boy 1
the 2
wolf 1
After replacement:
and 1
the 2
wolf 2
Now the word 'and' is found at depth 2
```

The shape of the tree before (left) and after (right) the replace call is given below:


the 2
boy 1
wolf 1
and 1
the 2
and 1
wolf 2