DSA Documentation – Assignment 2

# Introduction

The problem to be solved is implementing a dictionary that takes words from a text file, strips any non-alphabetical characters from them and then stores the words in an AVL tree. This involves using an input stream for inputting the text file. The words then have to be printed in alphabetical order and to do this, the tree is traversed in inorder to get the whole tree. For the search, part of a word is checked against the dictionary until it is found, then that subtree is traversed to find the rest of the words with that part in it.

# Algorithms

Algorithm for trimming words, trimWords():

istringstream iss(word);

vector<string> tokens;

copy (istream\_iterator<string>(iss),

istream\_iterator<string>(),

back\_inserter(tokens));

for ( int i = 0; i < tokens.size(); i++ )

{

//testWord = tokens[i];

int length = tokens[i].length();

for ( int j = 0; j < length; j++ )

{

if ( !isalpha(tokens[i][j]) )

{

tokens[i].erase(j, 1);

j--;

length--;

}

}

if ( tokens[i] != "" )

{

cout << tokens[i] << endl;

wordNode n(tokens[i], 1, 1 / (Dictionary.AVL\_Return\_Count() + 1) );

Dictionary.AVL\_Insert(n);

}

}

Algorithm for retrieving the words for the priority queue, AVL\_Retrieve():

NODE<TYPE> \*node;

vector<wordNode> wordList;

if (!tree)

return wordList;

\_retrieve (key, tree, wordList);

return wordList;

\_retrieve()

if (root)

{

int check = 0;

for ( int i = 0; i < key.size(); i++ )

{

if ( key[i] == root->data.key[i] )

check++;

}

if (check == key.size())

{

wordList.push\_back(root->data);

\_retrieveAll(key, root->left, wordList);

return;

}

else

{

if ( key < root->data.key )

\_retrieve(key, root->left, wordList);

else if ( key > root->data.key )

\_retrieve(key, root->right, wordList);

}

}

# Data Structures

AVL Tree: This is used to store the entire dictionary of words in alphabetical order and make it very efficient to find a word.

Priority Queue: This is used to store the words in order of frequency for the search prediction and pop them out to print them.

wordNode: This is packed as the data in the AVL Tree and contains the word, the number of times it appears in the dictionary and the frequency.

NODE: The struct that comprises each node of the AVL Tree. Contains a wordNode, a pointer to the left NODE of the tree, a pointer to the right NODE of the tree and an integer which stores the balance of the subtree.

Vector: This stores the words from the search prediction before it is pushed into the priority queue.

# Complexity Analysis

## Building a Frequency Dictionary

Equation is:

# Conclusion