# Words List: An Application of Binary Search Trees

A major course output
for the course on
Data Structures and Algorithms
(CCDSALG)

Submitted by

Villarica, Matthew James D.

Joanna Pauline Rivera
Teacher

September 16, 2020

## I. Introduction

The task assigned is to design and implement an algorithm for a program that takes a text file as input and creates a text file as output that contains the word count of each word. The program must have the non-linear data structure binary search tree as a major part of its implementation. The required binary search tree operations must be implemented correctly, and at least 1 binary search tree has to be used to store the words from the text file. Exhaustive testing must be done. The purpose of this task is to practice and gain a greater understanding of non-linear data structures, specifically binary search trees and implement them practically.

## II. Design and Implementation

The Java programming language was used because of the ease of the string manipulation required to read from the text file and the built in API, such as the buffered writer and buffered reader which was used for reading text input from the text file as strings. 2 binary search trees were used to store the valid words that were scanned from the text file. One binary search tree named *allWords* contains every valid word including its duplicates. That is, if there are 14 instances of the word "the" in the text file, the binary search tree contains 14 copies of that string. The convention is to insert a string that is equal to the string in the current node to the right child during an insert operation. This binary search tree is used to determine the actual count or the number of instances of that word in the text file, with a recursive method that returns the integer count of a given string. Another binary search tree called *uniqueWords* also contains a copy of each scanned word in the text file, but it only contains 1 instance of each word without the duplicates. This binary search tree is used to do the in-order walk to print the words into the output text file and their respective counts in *allWords.*

The binary search tree and it's operations was implemented through a single class file called *BinarySearchTree.java*. An instance of this class can perform the binary search tree operations and store the words as strings in each node. The binary search tree is ordered in a lexicographic manner. Instances of a binary search tree are instantiated and used in the driver file *WordList.java* which is the program that will be run to perform the algorithm that completes the assigned task. Built in classes in the Java API like the buffered reader and buffered writer are used for text file reading and writing.

**PSUEDO CODE:**

```
• While there are still lines of text to be read from INPUT.txt
    o Read the next line as a string with the buffered reader.
    o Make a string array that splits each word if a space (“ “) Is present between them. It is assumed that all words are separated by a single space.
    o Remove the unnecessary characters like “!”, “.”, “?” from every word in the string array.
    o Remove the small words that have a length that is less than 3, all valid words are now in an array list
    o For every word in the array list of valid words:
        ▪ Insert the word into the binary search tree allWords
        ▪ If the word does not already exist in the binary search tree uniqueWords, insert it into the binary search tree uniqueWords
    o Reset the array list by removing all the entries so that it can be reused.
• Traverse the binary search tree uniqueWords with the in-order walk
    o For each word that is encountered, print the string with the buffered writer into OUTPUT.txt with it’s corresponding count, and insert a newline after. The count is determined by the amount of instances of the word in the binary search tree allWords.
```

**Java Version:**

java version "16.0.1" 2021-04-20

Java(TM) SE Runtime Environment (build 16.0.1+9-24)

```
Java HotSpot(TM) 64-Bit Server VM (build 16.0.1+9-24, mixed mode, sharing)
```

**Java Packages/API:**

- java.io.BufferedWriter
- java.io.BufferedReader
- java.lang.String

**To compile from the command line (for compiled language only):**

Compile the BST class file and driver file.

```
C:\CCDSALG>javac BinarySearchTree.java

C:\CCDSALG>javac WordList.java
```

**To run from the command line:**

```
C:\CCDSALG>java WordList.java
```

## III. Results and Analysis

**Test Input 1**

***Contents of INPUT.txt:***

*In Oathbringer, the third volume of the New York Times bestselling Stormlight Archive,*

*humanity faces a new Desolation with the return of the Voidbringers, a foe with numbers*

*as great as their thirst for vengeance!*

*Dalinar Kholin's Alethi armies won a fleeting victory at a terrible cost: The enemy Parshendi*

*summoned the violent Everstorm, which now sweeps the world with destruction, and in its* *passing awakens the once peaceful and subservient parshmen to the horror of their*

*millennia-long enslavement by humans. While on a desperate flight to warn his family of*

*the threat, Kaladin Stormblessed must come to grips with the fact that the newly kindled*

*anger of the parshmen may be wholly justified. How will Kaladin come to terms with the grayness*

*of it all?*

*Nestled in the mountains high above the storms, in the tower city of Urithiru, Shallan Davar*

*investigates the wonders of the ancient stronghold of the Knights Radiant and unearths dark*

*secrets lurking in its depths. And Dalinar realizes that his holy mission to unite his homeland*

*of Alethkar was too narrow in scope. Unless all the nations of Roshar can put aside Dalinar's*

*blood-soaked past and stand together and unless Dalinar himself can confront that past even*

*the restoration of the Knights Radiant will not prevent the end of civilization.*

*The threat of Odium proves to be quite a challenge for the Knights Radiant. What will*

*happen in the next installment of the Stormlight Archive?*

*Hello how are you today?*

***Contents of WORDS.txt:***

```
above   1

alethi   1
```

| | |
|---|---|
| *alethkar* | *1* |
| *all* | *2* |
| *ancient* | *1* |
| *and* | *6* |
| *anger* | *1* |
| *archive* | *2* |
| *are* | *1* |
| *armies* | *1* |
| *aside* | *1* |
| *awakens* | *1* |
| *bestselling* | *1* |
| *blood-soaked* | *1* |
| *can* | *2* |
| *challenge* | *1* |
| *city* | *1* |
| *civilization* | *1* |
| *come* | *2* |
| *confront* | *1* |
| *cost* | *1* |
| *dalinar* | *3* |
| *dalinars* | *1* |
| *dark* | *1* |
| *davar* | *1* |
| *depths* | *1* |
| *desolation* | *1* |

| | |
|---|---|
| *desperate* | *1* |
| *destruction* | *1* |
| *end* | *1* |
| *enemy* | *1* |
| *enslavement* | *1* |
| *even* | *1* |
| *everstorm* | *1* |
| *faces* | *1* |
| *fact* | *1* |
| *family* | *1* |
| *fleeting* | *1* |
| *flight* | *1* |
| *foe* | *1* |
| *for* | *2* |
| *grayness* | *1* |
| *great* | *1* |
| *grips* | *1* |
| *happen* | *1* |
| *hello* | *1* |
| *high* | *1* |
| *himself* | *1* |
| *his* | *3* |
| *holy* | *1* |
| *homeland* | *1* |
| *horror* | *1* |

| | |
|---|---|
| *how* | *2* |
| *humanity* | *1* |
| *humans* | *1* |
| *installment* | *1* |
| *investigates* | *1* |
| *its* | *2* |
| *justified* | *1* |
| *kaladin* | *2* |
| *kholins* | *1* |
| *kindled* | *1* |
| *knights* | *3* |
| *lurking* | *1* |
| *may* | *1* |
| *millennia-long* | *1* |
| *mission* | *1* |
| *mountains* | *1* |
| *must* | *1* |
| *narrow* | *1* |
| *nations* | *1* |
| *nestled* | *1* |
| *new* | *2* |
| *newly* | *1* |
| *next* | *1* |
| *not* | *1* |
| *now* | *1* |

| | |
|---|---|
| *numbers* | *1* |
| *oathbringer* | *1* |
| *odium* | *1* |
| *once* | *1* |
| *parshendi* | *1* |
| *parshmen* | *2* |
| *passing* | *1* |
| *past* | *2* |
| *peaceful* | *1* |
| *prevent* | *1* |
| *proves* | *1* |
| *put* | *1* |
| *quite* | *1* |
| *radiant* | *3* |
| *realizes* | *1* |
| *restoration* | *1* |
| *return* | *1* |
| *roshar* | *1* |
| *scope* | *1* |
| *secrets* | *1* |
| *shallan* | *1* |
| *stand* | *1* |
| *stormblessed* | *1* |
| *stormlight* | *2* |
| *storms* | *1* |

| | |
|---|---|
| *stronghold* | *1* |
| *subservient* | *1* |
| *summoned* | *1* |
| *sweeps* | *1* |
| *terms* | *1* |
| *terrible* | *1* |
| *that* | *3* |
| *the* | *28* |
| *their* | *2* |
| *third* | *1* |
| *thirst* | *1* |
| *threat* | *2* |
| *times* | *1* |
| *today* | *1* |
| *together* | *1* |
| *too* | *1* |
| *tower* | *1* |
| *unearths* | *1* |
| *unite* | *1* |
| *unless* | *2* |
| *urithiru* | *1* |
| *vengeance* | *1* |
| *victory* | *1* |
| *violent* | *1* |
| *voidbringers* | *1* |

```
volume   1

warn   1

was   1

what   1

which   1

while   1

wholly   1

will   3

with   5

won   1

wonders   1

world   1

york   1

you   1
```

**Test Input 2**

**Contents of INPUT.txt:**

```
Linear algebra is an important course for a diverse number of students for at
least

two reasons. First, few subjects can claim have such widespread applications

in other areas of mathematics multivariable calculus, differential equations, and

probability, for example as well as in physics, biology, chemistry. economics,
finance, psychology, sociology. and all fields of engineering. Second, The
subject

presents the student at the sophomore level with an excellent opportunity to
learn

how to handle abstract concepts.
```

*This book provides an introduction to the basic ideas and computational techniques of linear algebra at the sophomore level. It also includes a wide variety*

*of carefully selected applications. These include topics of contemporary interest.*

*such as Google and Global Positioning System. The book also introduces*

*the student to working with abstract concepts. In covering the basic ideas of linear*

*algebra. the abstract ideas are carefully balanced by considerable emphasis on the*

*geometrical and computational aspects of the subject. This edition continues to*

*provide The optional opportunity to use MATLAB™ or other software to enhance*

*the pedagogy of the book.*

**Contents of WORDS.txt:**

```
abstract   3

algebra   3

all   1

also   2

and   5

applications   2

are   1

areas   1

aspects   1

balanced   1

basic   2

biology   1

book   3

calculus   1
```

| | |
|---|---|
| *can* | *1* |
| *carefully* | *2* |
| *chemistry* | *1* |
| *claim* | *1* |
| *computational* | *2* |
| *concepts* | *2* |
| *considerable* | *1* |
| *contemporary* | *1* |
| *continues* | *1* |
| *course* | *1* |
| *covering* | *1* |
| *differential* | *1* |
| *diverse* | *1* |
| *economics* | *1* |
| *edition* | *1* |
| *emphasis* | *1* |
| *engineering* | *1* |
| *enhance* | *1* |
| *equations* | *1* |
| *example* | *1* |
| *excellent* | *1* |
| *few* | *1* |
| *fields* | *1* |
| *finance* | *1* |
| *first* | *1* |

| | |
|---|---|
| *for* | *3* |
| *geometrical* | *1* |
| *global* | *1* |
| *google* | *1* |
| *handle* | *1* |
| *have* | *1* |
| *how* | *1* |
| *ideas* | *3* |
| *important* | *1* |
| *include* | *1* |
| *includes* | *1* |
| *interest* | *1* |
| *introduces* | *1* |
| *introduction* | *1* |
| *learn* | *1* |
| *least* | *1* |
| *level* | *2* |
| *linear* | *3* |
| *mathematics* | *1* |
| *matlab* | *1* |
| *multivariable* | *1* |
| *number* | *1* |
| *opportunity* | *2* |
| *optional* | *1* |
| *other* | *2* |

| | |
|---|---|
| *pedagogy* | *1* |
| *physics* | *1* |
| *positioning* | *1* |
| *presents* | *1* |
| *probability* | *1* |
| *provide* | *1* |
| *provides* | *1* |
| *psychology* | *1* |
| *reasons* | *1* |
| *second* | *1* |
| *selected* | *1* |
| *sociology* | *1* |
| *software* | *1* |
| *sophomore* | *2* |
| *student* | *2* |
| *students* | *1* |
| *subject* | *2* |
| *subjects* | *1* |
| *such* | *2* |
| *system* | *1* |
| *techniques* | *1* |
| *the* | *14* |
| *these* | *1* |
| *this* | *2* |
| *topics* | *1* |

```
two   1

use   1

variety   1

well   1

wide   1

widespread   1

with   2

working   1
```

During testing, it was found that the program can successfully detect all words from the text file as long as each of them were separated with a space. Unnecessary punctuation marks such as question marks, periods, and commas are removed from the words before they are inserted into the binary search trees and all words with uppercase letters are converted to their lowercase versions. The count of each word which is derived from the *allWords* binary search tree is accurate. The method responsible for getting the count of a certain word in *allWords* works similarly to the default search operation of a binary search tree, so getting the count of a certain word is more efficient compared to using linear data structures, such as an array.

The output file correctly prints the words into OUTPUT.txt in an alphabetical manner and the corresponding count is printed next to the word, which means the in-order walk operation of the binary search tree was implemented correctly. Words with hyphen are supported, such as the word "blood-soaked" in test input 1. If there are any unnecessary punctuation marks in a word, it is removed automatically. All the implemented binary search tree operations and functions are working correctly.

One weakness of the program is that it uses the presence of a space between each string as a strict indicator of separating words. One instance is the word "New York" in test input 1. This is supposed to be treated as 1 word but since a space is used as a way to separate words, the program does not correctly output “new york” as 1 word. The program is also unable to differentiate between words that are common nouns or proper nouns. Common nouns are correctly all in lowercase, but proper nouns such as names like “Dalinar” in test input 1 and the country “New York” should be in uppercase in the output, but they are also converted to lower case. This is because differentiating between common and proper nouns and checking for cases like these in the algorithm was not implemented.

## IV. Conclusions and Recommendations

A program that made use of binary search trees to store words as strings and get the count of the instances of each word was implemented using the Java programming language. The program takes a text file as input and creates a file with the list of words in the text file and their corresponding count. Two binary search trees were used: one that contains all the words including the duplicates, and one that only contains one copy of each unique word in the text file. An in-order walk is done in the binary search tree that contains the unique words to print the words into the output text file in alphabetical order, and the binary search tree that contains all the words including the duplicates is used to get the count of the corresponding word which is printed next to the word in the text file.

It was found after testing that the program can correctly detect words and get each of their counts accurately in the text file, and that it supports words with hyphen. It was interesting that the buffered writer can be passed and used as a parameter in a recursive method, which is the method in the binary search tree responsible for printing the words in the output text file using the in-order walk. The output of the text file was not affected at all by the recursion. That is, it does not reset the contents of the text file after every recursive call, which was an interesting finding for me. It is recommended that the program can be improved by adding detection for proper nouns and keeping them capitalized. Support for single words that actually consist of multiple words like “New York” can also be implemented to improve the program.

## V. References

*Binary Search Tree*. (2021). GeeksforGeeks.
https://www.geeksforgeeks.org/binary-search-tree-data-structure/

## Appendix A: Contribution of Members

| Name | Contributions |
|---|---|
| Matthew James D. Villarica | Coding, documentation, testing |
| | |
| | |