

# The Application of Binary Tree in a Simple Spell Checker

Kevin Sendjaja, 13517023  
Program Studi Teknik Informatika  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia  
13517023@std.stei.itb.ac.id  
kevin.sendjaja@gmail.com

**Abstract** — A Spell Checker is an application or program that is often implemented within word processors. Spell checker is used to compare a word with its correct spelling that is recorded within the dictionary. This paper will cover about the concept of binary trees and its implementation in creating a simple spell checker.

**Keywords**—Binary Tree, Program, Spell Checker, Spelling

## I. INTRODUCTION

There are various languages that are being used in the world. Languages are used as a method to communicate between two or more parties, may it be in verbal or written forms. In writing documents, especially formal documents, spelling is an important factor and sometimes neglected. It's not rare to find a document with a spelling error or often referred as a typo, whether it is caused by a typing a wrong letter by mistake, or simply because the writer didn't know about the correct spelling which has been recorded within a dictionary.

Spell checkers were created to reduce the possibility of misspell. Usually, spell checkers will point out the words that have a spelling error, and in more advanced versions, Spell checkers will offer some word possibilities that may or may not be the word that the document writer wanted to write, in its correct spelling, or better yet, automatically change the words with spelling mistakes into its correct spelling.

Through this paper, the author would like to explain and implement the concept of binary trees to create a simple spell checker. The spell checker that will be explained is a simple version, which function is limited to simply showing whether the spelling is correct or incorrect

## II. TREE

### A. Definition

A Tree is a specialized form of a graph, which is a connected, undirected graph, which consists of no simple circuits. As such, a tree may not have multiple edges or loops in any of its vertices.

Let  $G = (V, E)$  is a simple, undirected graph, with the number of vertices  $n$ . The following statements are considered to be valid:

1.  $G$  is a tree.
2. Every vertex that is a member of  $G$  is connected to its root by a single path.
3.  $G$  is connected and has  $n-1$  edges.
4. Adding a single edge to the graph  $G$  will cause a circuit to be formed.

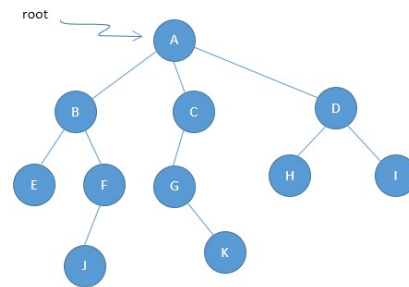


Figure 1: An example of a tree  
(Source: <http://www.studyalgorithms.com/tree/the-tree-data-structure/>)

### B. Terminology

Below is a list of important terms which are connected to trees:

1. **Root (*Akar*)**  
A root or root node is the uppermost node of a tree, which has no parent node. A tree may or may not have a root. A tree with a root node is called a Rooted Tree. Figure 1 is an example of a rooted tree with node A as its root.
2. **Parent (*Orangtua*) and Child (*Anak*)**  
Let vertex  $v$  is a vertex within a tree. Then for  $v$ , there is a unique vertex  $u$ , such that there is a directed edge from  $u$  to  $v$ .  $u$  is called the parent of  $v$ , and at the same time,  $v$  is called the child of  $u$ . A tree consists of minimal one parent and may or may not have any children. In Figure 1, node G is the child of node C, while node C is the parent of node G.
3. **Siblings (*Saudara Kandung*)**  
Two or more nodes which have the same parent are called siblings. In Figure 1, node B, C, and

D are siblings, for they have the same parent, which is node A.

4. Path (*Lintasan*)

A path is a series of vertices between a vertex and another vertex, one being the origin and the other being the destination. The length of a path is the number of edges between a vertex and another vertex. In Figure 1, the path from node A to node E is A-B-E, which has the length 2.

5. Ancestor (*Leluhur*) and Descendant (*Keturunan*)

Apart from the root, the ancestor of a vertex  $v$  within a tree is all vertices within a path from the root to  $v$ . At the same time, all vertices which have  $v$  as its ancestor are called descendants of  $v$ . In Figure 1, node A is the ancestor of node C, and node G and K are descendants of node C.

6. Subtree (*Upapohon*)

The subtree of a tree T consists of a vertex in T and all of its descendants in T. The uppermost vertex in a subtree becomes the root of that subtree. A subtree may consist of a single vertex or may form another tree.

7. Degree (*Derajat*)

The degree of a vertex is the number of subtrees that belong to that vertex. In Figure 1, node A has the degree 3, while node B has the degree 2.

8. Leaf (*Daur*)

A leaf is a vertex which has the degree 0. In other words, a leaf is a vertex without any children. In Figure 1, node E, H, I, J, and K are leaves.

9. Internal Vertices (*Simpul Dalam*)

Internal vertices are vertices that have at least a single child. The root of a tree is included as an internal vertex, unless it is the only vertex within a tree, in which case it is included as a leaf.

10. Level (*Aras*)

The root of a tree has the level 0, while the level of other vertices is the length of the path from the root to each vertex, respectively. In Figure 1, the level of node G is 2, while the level of node K is 3.

11. Height / Depth (*Tinggi / Kedalaman*)

The height or the depth of a tree is the maximum level of that tree. In Figure 1, the height of the tree is 3.

12. Skew Tree

A tree is a skew tree if and only if each vertex, with the exception of the leaf, have exactly one child. If every node has only left child, then it is

called as a left skew tree. Similarly, if every node has only right child, then it is called as a right skew tree.

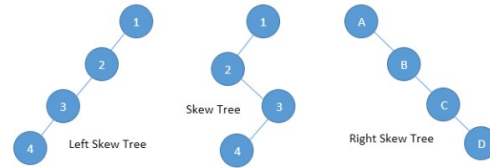


Figure 2: Examples of Skew Tree  
(Source: <http://www.studyalgorithms.com/tree/the-tree-data-structure/>)

13. n-ary Trees

n-ary tree is a tree where each vertex has a maximum number of children  $n$ . An n-ary tree is called a full n-ary tree if and only if each vertex has exactly  $n$  children.

C. Binary Tree

A Binary Tree is a type of n-ary tree which has a maximum of two children. A binary tree must have a root node, and the children of each node is classified into left child and right child respectively. A balanced binary tree is a binary tree where the difference of height between left and right subtrees is not greater than 1.

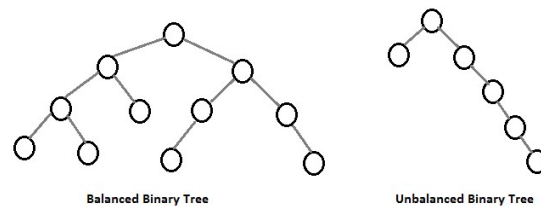


Figure 3: Comparison between Balanced and Unbalanced Binary Trees

(Source: <https://www.chegg.com/homework-help/questions-and-answers/python-3-binary-search-tree-algorithm-copy-pastable-code-pasteec-p-tfzww-problem-write-fun-q29685661>)

A full binary tree is a binary tree where each node has either 0 or 2 children. A complete binary tree is a binary tree where all levels are completely filled with the possible exception of the last level, and all the leaves on last level is placed as left as possible. In other words, in a complete binary tree with height  $n$ , for  $i$  from 0 to  $n-1$ , level  $i$  have exactly  $2^i$  nodes.

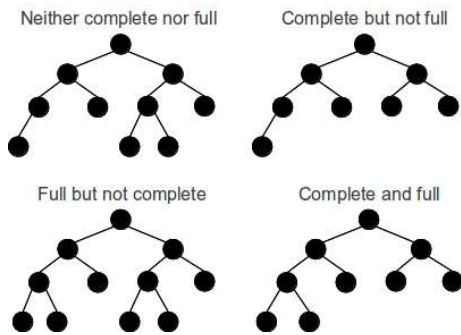


Figure 4: Comparison between Full and Complete Trees  
(Source: <http://code.cloudkaksha.org/binary-tree/types-binary-tree>)

A binary tree is called a perfect binary tree if and only if all internal nodes/ vertices have exactly two children, which in result, will cause all leaves to be on the same level. A perfect binary tree with height  $n$  will have exactly  $2^n - 1$  nodes. A full and complete binary tree may or may not be a perfect binary tree. However, a perfect binary tree will always be a full and complete binary tree.

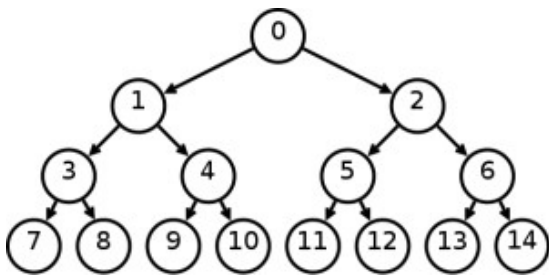


Figure 5: Example of a Perfect Binary Tree  
(Source: <http://code.cloudkaksha.org/binary-tree/types-binary-tree>)

Data contained within each node may or may not be sorted in an increasing order. Binary trees with sorted data are called Binary Search Trees (BST). Such trees, as the name imply, are often used in data sorting and searching. With the data stored in order, searching for a specific datum within the tree can be done faster by comparing the datum value with the current node value and follow the path corresponding to the comparison result.

### III. SPELL CHECKER



Figure 6: Example of a portable Spell Checker  
(Source: <https://images-na.ssl-images-amazon.com/images/I/918irSgirWL.SL1500.jpg>)

A spell checker is an application, program, or a function which determines whether the spelling of a given word based on a language set, is correct or incorrect. It can be a standalone program, which can be implemented into a device such as in Figure 6. It can also be a part of a greater program which operates on blocks of texts. The most common example is the spell corrector on word processors such as Microsoft Word. Other examples include search engines and email client.

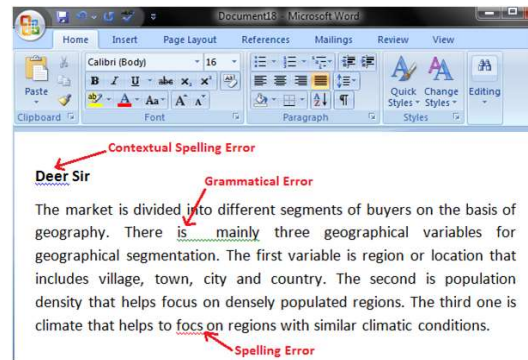


Figure 7: An example of spelling check on Microsoft Word  
(Source: <https://www.javatpoint.com/to-correct-errors-in-ms-word>)

It can be seen from Figure 7, that the spell checker in Microsoft Word is already quite advanced, as not only it underline the word with spelling error, it also underline the words with both grammatical error and contextual spelling error, each with a different color.

While spell checker is often taken for granted these days, it was considered an important research in 1957, under the branch of Artificial Intelligence (AI). The first official spell checker application was created by Ralph Gorin in the Artificial Intelligence Laboratory at Stanford University in February 1971. The application was called Spell for DEC PDP-10 and became widely available for mainframe computers during that decade. In 1980, the first spell checkers for personal computers were available for the TRS-80 and CP/M computers, followed by IBM computers in the following year.

The process of spell checking are as follows:

- The program scans a block of text and separates each individual word.
- The program compares each word with known words within a language set, which are contained within a dictionary file of correctly spelled words. The dictionary file may also include punctuation and grammatical rules.
- The program marks the word with incorrect spelling. The program may also offer the

- correct spelling or even automatically correct the word.

#### IV. IMPLEMENTATION OF BINARY TREE FOR SPELL CHECKING

##### A. Application

First, it is necessary to construct the binary tree prior to the spelling check. Each nodes contain exactly one letter, starting from any alphabet as the root. The left child and right child contain a second letter that will lead to a possible valid word. The pattern repeats until it reaches the last letter of a valid word. The last letter of a valid word serves as the leaves of the tree.

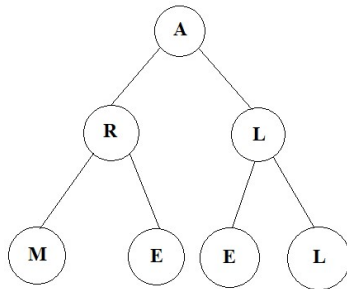


Figure 8: An example of a binary tree for spell checking (Source: Author's documentation)

Figure 8 is an example of a simple binary tree for several 3-letter words starting from the letter 'A'. The nodes passed from the root to each respective leaf shows the word that can be checked using this particular binary tree. The words that are verified to be correctly spelled are "ARM", "ARE", "ALE", and "ALL".

The binary tree and the checking process uses the Binary Tree Abstract Data Type (ADT) in its process. As the ADT uses recursive processing, the checking process also use recursive to check whether the word is correctly spelled or not.

For example, let T is a binary tree containing data exactly as pictured in Figure 8, and C is the first letter of the input obtained using the `getchar()` function. The checking process is as follows:

```

function SpellCheck(T : BinTree, C :
char) -> boolean
LOCAL VARIABLE
D : char
ALGORITHM
D <- getchar()
if((Info(T) = C) and (IsDaun(T)) and
(D = '\n')) then
return true
else if((Info(T) = C) and (IsDaun(T))
and (D <> '\n')) then
return false
  
```

```

else if((Info(T) = C) and (!IsDaun(T))
and (D = '\n')) then
return false
else if(Info(T) <> C) then
return false
else
return
SpellCheck(Left(T), D)
or
SpellCheck(Right(T), D)
  
```

Figure 9: Pseudocode for the spell checker (Source: Author's documentation)

The first conditional is the condition for the word to be correctly spelled, which are as follows:

- The word reaches the end at the same time as the current node in the binary tree reaches any leaf
- Every letter within the word matches with the value of the nodes passed by the path from the root until any leaf.

The next three conditionals are the conditions for the word to be spelled incorrectly, which are as follows:

- The word reaches the end before the current node in the binary tree reaches any leaf.
- The word hasn't ended, but the current node in the binary tree is a leaf.
- The current letter in the word doesn't match the value of the current node.

The last two conditionals are meant to continue the recursive process. The functions repeat the same process using the next letter and both the left and right subtree at the same time. Even should one return a false value, if the other return a true value, then the word will be considered spelled correctly. The word will be considered spelled incorrectly if and only if both the left and right subtree return a false value, which implies that the next letter doesn't match the root of either left or right subtree.

For example, we have the input "ARM", the process begin by matching the first letter of the input, which is 'A', with the value of the root of the tree, which is also 'A'. Since it matches, then the process continues with the next letter of the input, which is 'R'. The letter matches the root of the left subtree, but since the node R is not a leaf, the process continues once again with the next letter, which is 'M'. The letter matches the root of the left subtree, and since node M is a leaf and the letter 'M' is the last letter of the word as well, the function returns a true value, considering the word to be spelled correctly.

Another example, let the input be "ARN", which has one letter difference from the previous example. The process goes through exactly as the previous example until it reaches node R. When the function checks both of the left subtree of the node R, none of them matches the letter 'N' from the input. As such, the function returns a false value, and the word is considered to be spelled incorrectly.

Yet another example, this time let the input be "AR". The process also exactly the same until the node R. Node R is not a leaf, but there's no more letter in the word (in C

program, we use the character ‘\n’ to define a new line, meaning the end of input within a line). And so, function returns a false value, and the word is considered to be spelled incorrectly as well.

One final example, let the input be “ARMA”. The process is similar to the first example until the node M. This time, the node M is a leaf, but there’s still another letter within the word, which is ‘A’. And because of that, the function returns a false value, and the word is also considered to be spelled incorrectly.

*B. Advantages and Disadvantages*

The advantage of using this method is that should the word is spelled incorrectly, the function will stop immediately as it finds a letter that didn’t match the value on the binary tree. With this, the efficiency of the program can be increased for cases that include misspelled words. By using this method, the program too, doesn’t have to search the dictionary file for the matching word. As there could be more than a million words within the file, even a word beginning with letter ‘B’ would need to pass a lot of checking before the program could check whether the word is spelled correctly or not.

The disadvantage of this method is that a binary tree could not include all words that are spelled correctly. There’s more than a million words that are recorded in the dictionary for English alone, and a binary tree which consists of only two children couldn’t possibly be used to store each and every single word. One possible solution for this problem, is to change binary tree to n-ary tree, which could contain much more data than a binary tree. That way, more data could be stored within the tree, and more words could be checked for its spelling.

Another disadvantage is, as the previous paragraph described, there is a lot of words that needs to be recorded into the tree before the spell checking could be initiated. If it’s only a few words, then the recording process wouldn’t take much time. But the same couldn’t be said for a million words, since it would take a very long time to finish recording every last word into the tree. Since this is a mandatory process, unfortunately there’s no other solution to increase the efficiency in recording the data beforehand.

Another disadvantage that could be pointed is that a word that ended before the current node reaches a leaf node could be a word that is correctly spelled.

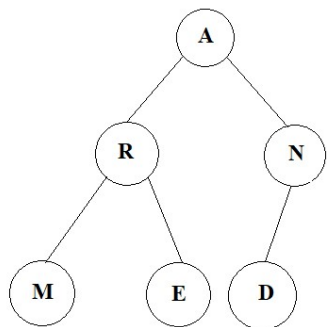


Figure 10: Another example of binary tree for spell checking

(Source: Author’s documentation)

Figure 10 is a similar to Figure 8 with a slight change on the right subtree. On Figure 10, it could be seen that input “AND” will be considered correctly spelled by the function. However, if we use the function for the input “AN”, it will return a false value and will be considered to be spelled incorrectly, when the word “AN” is recorded in the dictionary and is spelled correctly. This is caused by the Abstract Data Type itself, since it would be difficult to differentiate the comparison between a word that ends when the current node reaches a leaf and a word that ends before the current node reaches a leaf, if both case are words that is correctly spelled. A possible solution for this problem is to modify the data type BinTree from the ADT as well as the function SpellCheck. It is possible to add another element within the struct Node, for example we could add an element named Valid, which is a boolean type element. This element could be assigned with true and false value, which could be used to show whether the word is correctly spelled or not, should the node is not a leaf node and have at least one child. The function SpellCheck could be modified as well so that it includes checking the value of the element Valid from each node, to make additional checking if the word ended before the current node reaches a leaf. That way, the word “AN” from the previous example could be considered to be spelled correctly.

V. CONCLUSION

A binary tree is a data type which can be used for various purposes. One possible use of binary tree is for spell checking. Using a binary tree, simple spell checking could be done more efficiently.

A thorough spell checking would require more data to be recorded before the checking. In this case, using n-ary trees and modifying the Abstract Data Type for Binary Tree could help in making the process more accurate.

VI. ACKNOWLEDGMENT

First of all, the author would like to thank God for His blessing, because without it, the author would not be able to finish this paper. The author would also like to thank Mrs. Harlili, as the lecturer for class 02 for the course IF2120 Matematika Diskrit, for the knowledge that was taught during the class, which helped the author to finish this paper. The author also would like to thank the authors parents and friends which have given the author their support, may it be directly or indirectly. Lastly, the author would like to apologize for any mistakes that may have been made accidentally. May this paper be useful for future references and research purposes.

REFERENCES

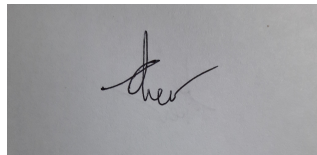
[1] Rosen, Kenneth H., *Discrete Mathematics and Its Applications*, 7<sup>th</sup> ed, New York: McGraw-Hill International, 2012.

- [2] [http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2013-2014/Pohon%20\(2013\).pdf](http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2013-2014/Pohon%20(2013).pdf) accessed on December 9<sup>th</sup>, 2018.
- [3] <https://www.techopedia.com/definition/12396/spell-checker> accessed on December 9<sup>th</sup>, 2018.
- [4] <http://code.cloudkaksha.org/binary-tree/types-binary-tree> accessed on December 9<sup>th</sup>, 2018.
- [5] <http://www.studyalgorithms.com/tree/the-tree-data-structure/> accessed on December 9<sup>th</sup>, 2018.

#### PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 9 Desember 2018

A rectangular box containing a handwritten signature in black ink. The signature is stylized and appears to read 'Kevin'.

Kevin Sendjaja / 13517023