**IMPLEMENTATION OF TEXT EDITOR USING ROPE DATA STRUCTURE**

A MAJOR PROJECT REPORT

***Submitted by***

CH.EN.U4AIE20001 ANBAZHAGAN E

CH.EN.U4AIE20053 RAMYA POLAKI

CH.EN.U4AIE20069 TRINAYA KODAVATI

CH.EN.U4AIE20031 SMITHIN REDDY K

CH.EN.U4AIE20035 ABHIRAM KUNCHAPU

***in partial fulfilment for the award of the degree***

***Of***

**BACHELOR** **OF TECHNOLOGY**

IN

DATA STRUCTURES AND ALGORITHMS



AMRITA SCHOOL OF ENGINEERING, CHENNAI

AMRITA VISHWA VIDYAPEETHAM

CHENNAI – 601103, TAMIL NADU

**AMRITA VISHWA VIDYAPEETHAM**

**AMRITA SCHOOL OF ENGINEERING, CHENNAI, 601103**



**BONAFIDE CERTIFICATE**

This is to certify that the major project report entitled “**IMPLEMENTATION OF STACK USING TEXT EDITOR”** submitted by

CH.EN.U4AIE20001 ANBAZHAGAN E

CH.EN.U4AIE20053 RAMYA POLAKI

CH.EN.U4AIE20069 TRINAYA KODAVATI

CH.EN.U4AIE20031 SMITHIN REDDY K

CH.EN.U4AIE20035 ABHIRAM KUNCHAPU

in partial fulfilment of the requirements for the award of the **bachelor** **of Master of Technology** in **COMPUTER SCIENCE ENGINERRING**  is a bonafide record of the work carried out under my guidance and supervision at Amrita School of Engineering, Chennai.

Signature

Vigneshwaran Muralidaran

CSE Dept professor

This project report was evaluated by us on ……………...

INTERNAL EXAMINER EXTERNAL EXAMINER

**ACKNOWLEDGEMENT:**

I offer my sincere pranams at the lotus feet of Universal guru, MATA AMRITANANDAMAYI DEVI who blessed me with her grace to make this a successful project.

I express my deep sense of gratitude to Dr. P Shankar, Principal, Amrita School of Engineering, Chennai for his kind support. I would like to extend my gratitude and heartfelt thanks to Mr. vigneshwaran Muralidaran CSE Department and Dr. A Manikandan, , , for his constant help, suggestions and inspiring guidance. I am grateful to my guide, Department of EEE , for his invaluable support and guidance during the course of the major project work. I am also indebted to Dr. Prasanna Kumar, Chairperson of Department of AI, ASE, Chennai for his guidance.

I specially thank our director Sri I B Manikandan who supported us in every possible manner.

**INDEX:**

**Topic Page. No**

1. Abstract----------------------------------------------------------------------------------- 5
2. Introduction------------------------------------------------------------------------------ 6
3. Methodology----------------------------------------------------------------------------
4. Discussion and Result-----------------------------------------------------------------
5. Conclusion-------------------------------------------------------------------------------
6. References-------------------------------------------------------------------------------

**1. ABSTRACT:**

A Rope data structure is a tree data structure that is used to more efficiently store and manage huge strings. In comparison to a typical String, it allows operations like insertion, deletion, search, and random access to be performed considerably faster and more efficiently. This data structure, as well as its many operations and implementation, will be examined.

To handle huge strings effectively, software such as text editors like Sublime, email services like Gmail, and text buffers use this data structure.

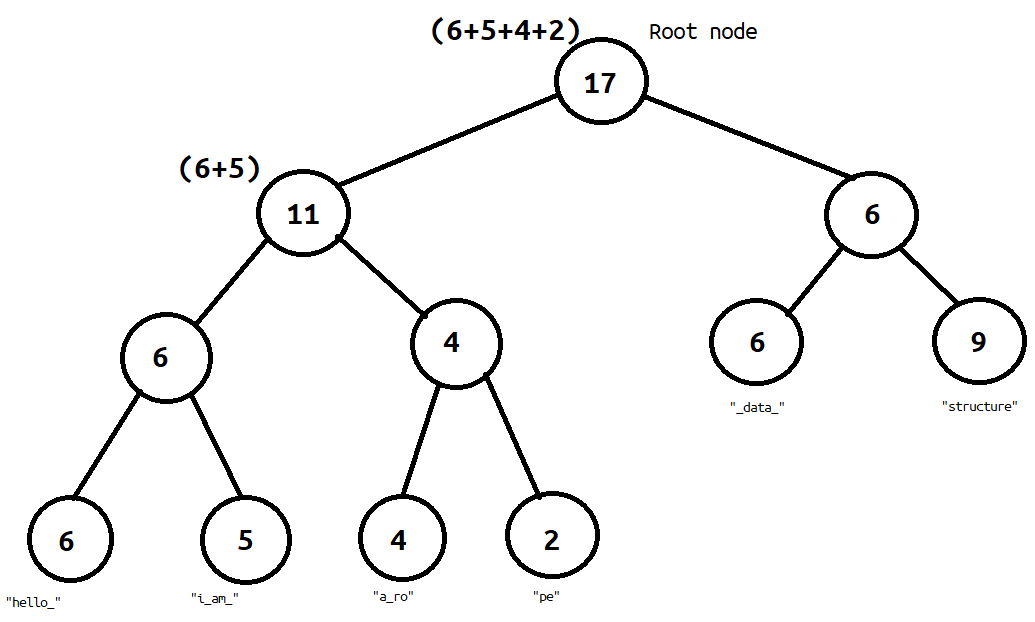
In this report, we'll look at the Rope Data Structure and compare it to a conventional string to see when it's appropriate to utilise it, as well as its benefits and drawbacks. The Rope has a wide range of uses, but the most popular is in text editors that deal with vast amounts of text.

**2. INTRODUCTION:**

A rope is a type of binary tree i.e. each node can have a maximum of 2 children at the maximum, where every leaf node holds a String or a Substring and the length of the same(Also known as the "weight" for the corresponding leaf node) and every following parent node holds the total sum of the weights of the leaf nodes in its left subtree, which in turn is described as the weight of the said node. In a simpler form, the weight of a node is the sum of all the leaf nodes that sprout from its immediate left child node.

In computer programming, a rope, also known as a cord, is a data structure made up of smaller strings that is used to store and manage a long string efficiently. A text editing tool, for example, might utilise a rope to represent the text being edited, allowing actions like insertion, deletion, and random access to be performed quickly.

The string is : Hello I am a rope data structure



The image shows how the string is stored in memory. Each leaf node contains substrings of the original string and all other nodes contain the number of characters present to the left of that node. The idea behind storing the number of characters to the left is to minimise the cost of finding the character present at i-th position.

Advantages of Rope data structure:

1. Ropes significantly reduce the cost of joining two strings.

2. Ropes do not require massive contiguous memory allocations, unlike arrays.

3. Ropes don't need O(n) extra memory for operations like insertion, deletion, and searching.

4. If a user wants to undo the previous concatenation, he can do so in O(1) time by simply removing the tree's root node.

**3. METHODOLOGY**

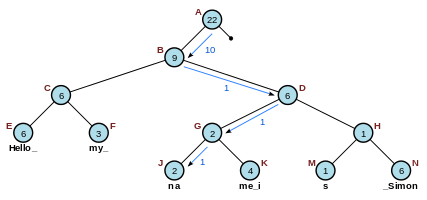
How rope data structure is used in text editor

**i)Search**

In order to find the character at i-th position, we search recursively beginning at the root node.

We follow the premise that if the weight of the current node is lower than the value of i, we subtract the weight from i & move right. If the weight is less than value of i we simply move left. We continue till the point we reach a leaf node.

Average case Time complexity:



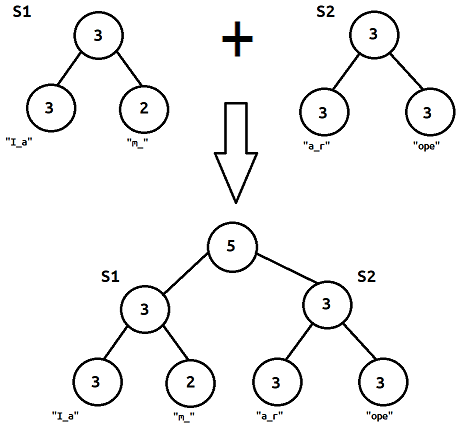
*Fig 1 searching i-th position*

For example, to find the character at i=10 in Figure 2.1 shown on the right, start at the root node (A), find that 22 is greater than 10 and there is a left child, so go to the left child (B). 9 is less than 10, so subtract 9 from 10 (leaving i=1) and go to the right child (D). Then because 6 is greater than 1 and there's a left child, go to the left child (G). 2 is greater than 1 and there's a left child, so go to the left child again (J). Finally 2 is greater than 1 but there is no left child, so the character at index 1 of the short string "na" (ie "a") is the answer.

**ii)Concatenation**

A concatenation operation between 2 strings(S1 & S2) is performed by creation of a new root node which has a weight equal to the sum of weights of leaf nodes in S1. This takes time if the tree is already balanced. Since, most of the rope operations need a balanced tree, it might require re-balancing after the operation.

Time complexity:  to concatenate strings and  to calculate weight of root.



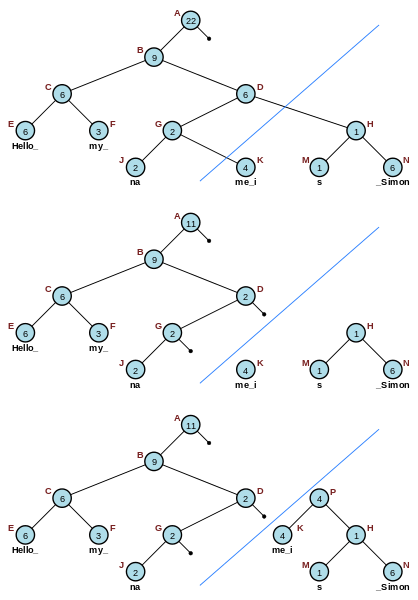
*Fig 2 concatenation*

**iii)Split**

In order to split a string at any given point i, there are 2 major cases we might encounter:

1. Split point being the last character of a leaf node.
2. Split point being a middle character of a leaf node.

With the second case, we can reduce it to the much simpler first one by splitting the string at given point into 2 leaf nodes & creating a new parent for these component strings. Follow this with subtracting the weights of the cut off parts from the parent nodes and re-balancing the tree.

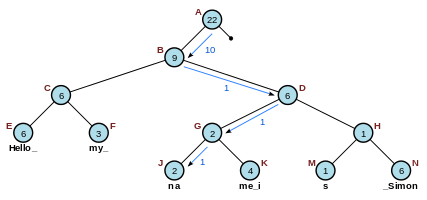


*Fig 3 Splitting*

For example, to split the 22-character rope pictured in Figure 2.3 into two equal component ropes of length 11, query the 12th character to locate the node K at the bottom level. Remove the link between K and G. Go to the parent of G and subtract the weight of K from the weight of D. Travel up the tree and remove any right links to subtrees covering characters past position 11, subtracting the weight of K from their parent nodes (only node D and A, in this case). Finally, build up the newly orphaned nodes K and H by concatenating them together and creating a new parent P with weight equal to the length of the left node K. As most rope operations require balanced trees, the tree may need to be re-balanced after splitting.

**iv)Indexing**

it means returning the character at position *i*



*Fig 4 Indexing*

For example, to find the character at i=10 in Figure 2.1 shown on the right, start at the root node (A), find that 22 is greater than 10 and there is a left child, so go to the left child (B). 9 is less than 10, so subtract 9 from 10 (leaving i=1) and go to the right child (D). Then because 6 is greater than 1 and there's a left child, go to the left child (G). 2 is greater than 1 and there's a left child, so go to the left child again (J). Finally 2 is greater than 1 but there is no left child, so the character at index 1 of the short string "na" (ie "a") is the answer**.**

**v)Insertion**

In order to insert a string S' between our string, at position i, we simply need to split it at the index from which to insert, and the concatenate the new string followed by the split off part of the original string. The cost of this operation is the sum of the three operations performed.

**vi)Deletion**

In order to delete a part of string from the middle, split the string at provided indices from ith to i+jth character and then concatenate the strings without the remaining part.

**4. DISCUSSION AND RESULT**

In comparison to string arrays, where operations have a temporal complexity of O, ropes allow for substantially faster text insertion and deletion (n). When working with ropes, unlike arrays, which require O(n) extra memory for copying operations, there is no need for O(n) extra memory. It does not require vast contiguous memory spaces.

**5. CONCLUSION**

**6. REFERENCES**

<https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.9450&rep=rep1&type=pdf>

<https://www.sgi.com/tech/stl/ropeimpl.html>

<https://www.geeksforgeeks.org/ropes-data-structure-fast-string-concatenation/>

<https://www.averylaird.com/programming/the%20text%20editor/2017/09/30/the-piece-table/>

<https://www.adoclib.com/blog/how-to-use-rope-in-text-editors.html>