INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

DIRECTORY MANAGER

CS1.201

DATA STRUCTURES AND ALGORITHMS

MINI PROJECT

Team 24

Swetha Vipparla
Roll No: 2020101121

Sankalp Bhat
Roll No: 2020112018

Anurag Gupta
Roll No: 2020101019

Mitul Garg
Roll No: 2020102026

Vidhi Pareek
Roll No: 2020101102

CONTENTS

ACKNOWLEDGEMENTS

We would like to extend our sincere, boundless gratitude to the director, Dr P.J. Narayanan, our Data Structures and Algorithms professors Dr Ravi Kiran S and Dr Sujit Gujar, our teaching assistants, and all the teaching and non-teaching staff at IIIT-Hyderabad for providing us with the experience, knowledge, and skill set needed to present this project of Data Structures and Algorithms.

In particular, we would like to thank our professors and our teaching assistants for their efforts in guiding us in our times of doubt and providing us with this learning opportunity.

This has truly been a great learning experience that has broadened our point of view towards Data Structures and their implementation in practical life.

LINK TO GITHUB REPOSITORY

https://github.com/SwethaVipparla/Directory-Manager

DATA STRUCTURES USED

The primary data structure used for building the directory is **First Child Next Sibling Tree**.

The tree uses the format where every node of the tree stores the address to one of its child nodes (Data contained inside that directory) and the address to the next directory/file stored underneath the same parent node.

The project also uses **Hash Tables** for the implementation of the store Alias and the Teleport functionality where every node of the Hash-Table stores the address and the alias to that address.

The Linking methodology involves **Linked List** Data Structure implementation.

For Find and Move functionality implementation for error handling involves **2D Character Array** usage for storing and searching the added elements.

FUNCTIONALITIES

I. ADD FUNCTION

Implemented by **Swetha Vipparla** and **Vidhi Pareek**

Data Structure Used:
First Child Next Sibling Tree, Linked List

Time Complexity:
Complexity is **O(N)** where N is the number of currently present elements.

Algorithm:
The Add function calls upon the makeNode function which allocates memory to a node in the linked list corresponding to the First Child of the current directory.

Error Handling:
The function call ensures that no two directories or no two files or no file and directory can have the same name inside the same directory using search functionality.

Note:
The name entered in the Add functionality must be a single word.

FUNCTIONALITIES

II. MOVE FUNCTION

Implemented by **Vidhi Pareek** and **Swetha Vipparla**

Data Structure Used:
Character Arrays, Trees, Linked List

Time Complexity:
Complexity can be **O(K+M*N)** where K is the length of the input string, M is the number of words(expected to be names of directories) we extract from the string and N is the average number of elements present in the directory we are searching.
If (K << N) complexity will be O(M*N). In the worst case, we will be searching all the nodes of the tree.

Algorithm:
The function accepts the path where the current pointer needs to be shifted. It iterates over the address and breaks it into its components based on the '/' sign. Each component is looked for in the linked list corresponding to the previous component. If any search returns NULL, the function terminates indicating an error. If the iteration is complete and the element is found, it returns a pointer to that element and the current pointer is now replaced with the found element.

Error Handling:
The function is making sure that we do not enter a file, it makes sure that no change in the current position is performed if the address given is incorrect. Also, it prints an error message according to different errors.

Note:
The name of the initial directory is root so the address for any directory like root-->Directory1-->Directory11 should be root/Directory1/Directory11

FUNCTIONALITIES

III. ALIAS FUNCTION

Implemented by **Anurag Gupta** and **Sankalp Bhat**

Data Structure Used:
Character Arrays, Hash Tables

Time Complexity:
O(N) where N is the length of the Alias meant to be stored followed by **O(1)** for storing it in the Hash-Table.

Algorithm:
The hash function used is Horner’s Hashing rule. It converts the given alias into an integer value and stores it into a globally declared Hash-table. The hash table node stores the Alias along with the path for that alias using the separate chaining methodology. It makes sure the address is valid and the alias mentioned has not been used before. An address can have multiple aliases, but no two addresses can have the same Alias.

Error Handling:
The function ensures that no two addresses can have the same alias. An address can have multiple aliases while no two addresses can have the same alias. It ensures no change in the current directory is performed if an incorrect alias is input. When an alias is input, the function first looks for already input aliases and ensures it is not repeated.

Note:
The Alias must be a single word. Take into consideration the path constraints while entering the address to be stored.

FUNCTIONALITIES

IV. TELEPORT FUNCTION

Implemented by **Sankalp Bhat** and **Anurag Gupta**

Data Structure Used:
Character Arrays, Hash Tables

Time Complexity:
O(N*M*1) where N*M is the complexity for the Move functionality used to change the current directory to the required directory and **O(1)** is the complexity to extract the Address corresponding to the given alias.

Algorithm:
The function looks for the given Alias in the hashtable and the hash-table node returns the address corresponding to the given alias. The address is then fed to the MOVE function which returns the pointer to the directory which is being looked for.

Error Handling:
The Function gives an error if an input alias is not present. It verifies the input alias by running a search function over hashtable.

FUNCTIONALITIES

V. FIND FUNCTION

Implemented by **Mitul Garg**

Data Structure Used:
String Array (2D Character Array)

Time Complexity:
O(L*K) where N is the total number of nodes in the entire directory, K is the length of prefix needed to be looked for, M is the number of nodes in the current subtree, and L is the number of elements in the current directory.

Algorithm:
All the Elements added to the directory are also added to a string array where every node's string is compared with the prefix string and if it is present, it prints. The function outputs the elements with the given prefix in the current directory.
For searching, it performs linked list traversal.

Note:
The prefix entered must be a single word.

FUNCTIONALITIES

VI. LS FUNCTION

Implemented by **Mitul Garg**

Data Structure Used:
Linked list, First Child Next Sibling Tree

Time Complexity:
Complexity is **O(N)** where N is the number of currently present elements.

Algorithm:
The working node is taken as an input for the ls function, and a linked list traversal is performed in the directory, listing out the names of all the components in the directory, as we move through the linked list.

VII. QUIT FUNCTION

Implemented by **Swetha Vipparla**

Time Complexity:
Complexity is constant, i.e **O(1).**

Algorithm:
Quits the program.

DIVISION OF WORK

1. **ADD Function:**
   Swetha Vipparla, Vidhi Pareek

2. **MOVE Function:**
   Vidhi Pareek, Swetha Vipparla

3. **ALIAS Function:**
   Anurag Gupta, Sankalp Bhat

4. **TELEPORT Function:**
   Sankalp Bhat, Anurag Gupta

5. **FIND Function:**
   Mitul Garg

6. **LS Function:**
   Mitul Garg

7. **QUIT Function:**
   Swetha Vipparla

Testing:
Vidhi Pareek, Sankalp Bhat, Mitul Garg, Swetha Vipparla, Anurag Gupta

Command Line Interface:
Swetha Vipparla

Documentation (Readme / Report):
Sankalp Bhat, Swetha Vipparla, Anurag Gupta

Debugging of the code:
Sankalp Bhat, Swetha Vipparla, Anurag Gupta, Vidhi Pareek, Mitul Garg