PROJECT REPORT

TEAM NO.: 47
TOPIC: ASSIGNMENT SHELL

DATA STRUCTURES USED:

The key data structure we have implemented is the tree data structure in Assignment Shell. In this project , it was required to create a normal bash-like shell where we use commands like switch , create , update , setup , test , submit , compare and use in order to ensure the efficient submission and uploading of an assignment by a student. Each command used needs to keep track of the current working directory, subdirectories and subfolders. So, the best way to implement this is by using ‘tree’ data structure.

Tree structure :

- The root of the tree would be the user folder.
- Then it branches out into its children which would be the subject folders .
- The subject folders will be having the assignment folders (created by using the create command).
- The subject folders will also contain the final zip file submissions of the assignment folders (this is done by using submit command).

Along with implementing a tree structure we also have a linked list connecting all the children of a specific node i.e. all the subject folders can be accessed this way.
We have a current_directory variable which keeps track of the directory which the user is currently in and helps in efficient traversal within the tree.

ALGORITHMS USED:

The Algorithms we used in the Assignment Shell are mainly for tree traversal:

1) **DFS in tree (Depth First Search)**

   Time complexity : O (d+f)
   d: total number of directories in the tree
   f: total number of files in the tree

   In the Assignment Shell project, the prompt shows that we are in the user directory i.e root node at first. Our journey starts from switching to a subject followed by creating the folder for an assignment, copying the files into it, updating if any updates were made,etc. So, we are exploring the whole tree. We first travel from the root node to the subject to the required assignment folder and either use it or perform the required actions thus, it is DFS.
   We use tree structure in our project to keep track of all files and folders.
   Every node in our tree has 5 fields NAME,PATH,POINTER TO SUB DIRECTORIES,POINTER TO SUBFILES.
   The sub directories is a linked list now..with directories at each node and for each directory we go into the sub directories and so on for every directory we recursively call list directories and it continues till there are no subfolders.
   So basically we enter first subdirectory then first subdirectory of this subdirectory and so on.

2) **Linked List in tree**

   Time complexity : O (n)
   n: number of nodes in the linked list

   In our tree structure , each directory or each folder represents a node . So, It is used to traverse between the different subdirectories like subjects or assignments. All the subdirectories are linked to each other making traversal easier at a level. For sub files it's just a linked list.So the complexity for this will be O(n).
   We use only linked list search in switch, submit, create, compare, etc. to find if a certain folder or file exists in the current directory.

DIVISION OF WORK:

Miryala Narayana Reddy : Implementation of tree, switch command, Setup command and test command
Mudit Gaur : Create command
Tisha Dubey : Update command
Poorva Pisal : Submit and compare command
Merugu Nanditha : Use command

RUNNING MOSS AND CHECKING DURING EVALUATIONS:

The command that can be added could be of the form *moss <dirname>* where *dirname* refers to the directory in which the codes are stored. We will also have a folder named moss at the root which will contain the moss executable file and a text file named *mossthesecodes.txt* which will contain the string "*moss -l c* ". Note that we are assuming the codes to be written in C language.

We will first copy the contents of the moss folder into the assignment folder. We can then have a function that will first scan the names of all the files in the given directory after which it will continue adding their names to the mossthesecodes.txt file which will then be executed using *cat* command in the terminal.

This can be achieved by executing *ls -c >> codes.txt* via *system()* function to store the names of all the files in the given directory after stepping into it. Then we merge the two .txt files together with the command *cat codes.txt >> mossthesecodes.txt* . After that we can execute moss by passing the command *cat mossthesecodes.txt | bash* into the terminal.