Data Structures and Algorithms

Directory Management

Mini Project

Team 31
Aryamaan Basu Roy (*2020101128*)
Tirth Motka (*2020101036*)
Aravapalli Akhilesh (*2019114016*)
Abhijith A (*2020101030*)
Chinmay Deshpande *(2020102069)*

Description

This project is a Directory Manager, implemented using the Tree data structure. The directories are represented by a single string for their names. We opted to use trees since it would be the easiest to visualize. Each node of the tree (except for the leaves) has a linked list of sibling nodes and children associated with them. Although the pointers for sibling and child nodes are present in the struct definition, they point to NULL in case of the leaf nodes. Each node struct has a name and type associated with it, corresponding to the name of the directory/file.

Functions

The core interface consists of the following functions.

Add

- This function is used to add a new directory/file within the present directory. It goes to the linked list of children and appends a new node at the start of the list.
- Its time complexity is typically O(1).
- It also takes care of error handling by not allowing you to add anything into files.

Move

- This function is used to change the current directory to any other directory. It is implemented using the function: ptr_from_path().
- The function takes a character array (path : root/dir1/dir2/) as an argument. It then iterates over each directory in the path, compares it to nodenames and hence traces to the destination directory/file. It uses a breadth-search approach. It returns the destination directory with which our current directory is updated.
- Time Complexity: O(F*C) where C is the average number of children in the directory being searched and F is the number of directories mentioned in path string.

Find

- This function is used to print all the child directories inside a given input directory, having a given prefix. For instance, if the prefix is “abc”, then it will print all child directories inside the input directory having prefix “abc” in their names.
- This consists of a checker function and a call function. The checker function checks if the prefix entered by the user matches the prefix (first few letters) of the directory/file name and the call function uses it on all the child directories.

- The time complexity of the checker function is O(L) in the worst case (Worst case is where all the prefix elements have to be checked), where L is the length of the prefix string. It is being called N times, if N is the number of child directories inside the input directory. Hence, the worst case time complexity of the find function is O(N*L). The usual time complexity is O(N*C) where C is the average number of character comparisons made across all nodes.
- In case the string has no characters then the message “No matches found” is printed.

Alias

- This function is used to give an alias to the directory or file whose complete path is given as an input and the name as a string.
- For implementing Alias to a directory we used a hash function and hash table whose nodes contain the Alias' name and the pointer to the directory which is given the Alias.
    - This function checks first if the alias is already taken using the Aliastaken function if the given name is already given to some node , if yes it handles this error and returns.
    - For implementing the hash table we used hash function as folding using the string each character's ASCII value's sum.
- Time Complexity :- finding the hash key for the alias string from the hash function takes O(n) time where n= string length of alias name. Now searching for a free space in the hashtable takes O(1) time but.
    - In case of collision we used quadratic probing so for a successful search avg time depends on (- (ln(1- c))/c) ,where c= loading factor=size of hashtables / nodes filled.
    - And in case of avg. unsuccessful search avg time depends on (1/(1-c)) , where c= loading factor = size of hashtables / nodes filled.

Teleport

- This function is used to change the current pointer to point towards a directory or file which has been given an alias.
- This uses the same hashtable as in the alias function. The teleport function takes in a string and the pointer to the alias hash-table,and firstly, checks if the alias is already present in the hashtable.
    - If it isn't already in the hashtable, prints the proper error handling, and returns NULL.

  - If the given alias is present in the hashtable, the function returns the pointer to the file/directory which has been given the alias.
- As this function is based on searching in the hashtable (which works on quadratic probing), the time complexity of this function is O(1) [ie, constant time function] . (In the worst case, if the table is full, *which is highly improbable as the function takes a dynamic hash table size*, the complexity could reach to O(N), where N is the size of the hashtable, as we *could* have to search the entire table) .

Quit
- This function basically quits from the program and also deletes the entire tree which was created from the start by implementing the free_tree() function.
- It also implements the free_aliastable() function that frees the hashtable used to store aliases.

Findall
- This function is similar to the find function, in that it can print all the files in the filesystem whose prefix matches the input prefix string.
- This makes use of preorder tree traversal and uses the checker function for all the nodes. Once the input directory is sent into the findall, the checker function is called recursively on each sibling directory of the input directory, then the same is done for the child nodes. Running time complexity is O(n), where n is the number of nodes in the filesystem.

Cut & Paste
- This function, as the name suggests, acts as a tool to cut and paste files/directories from one folder to another.
- It is implemented by the cut_and_paste() function which takes in the pointers to the root directory, current directory and path as string to destination directory.
- It then facilitates the deletion of current directory from its parent directory, then through the path, it adds the current directory to the destination directory derived from that path.
- Time Complexity: O(F*C) where C is the average number of children in the directory being searched and F is number of directories mentioned in path string.

ls

- This function is similar to that of ls in terminal, it shows the list of children in a current directory(non-recursive).
- It is implemented by taking the pointers to the current directory and then printing the children until the siblings' linked list ends
- The time complexity is O(l), where 'l' is the number of siblings present in a directory.

Back

- This function when called goes back one level of files/directories. This is similar to "cd .." in terminal.
- It is implemented by taking the CurrentPtr as input which then returns its parent. Error handling is done by considering root as the source folder, trying to call the back() from root results in a warning stating root is the source and you cannot go back from here.
- The time complexity is O(1).

PathFromPtr

- This is called at the start of the while loop.
- This function as the name suggests returns the current global path relative to root.
- This is implemented using recursion and is called in every iteration in the main function using path_from_ptr().
- Time Complexity of this function is O(h), where h is the height of the current node from root.

Delete

- This function accepts the argument of the current directory and name of file/directory to be deleted in the current directory.
- It then searches for the directory to be deleted using a breadth search on all children of the current directory. If such a directory doesn't exist, error handling is done. Once, directory is found, it is removed from the list and freed.
- Time Complexity: O(C) : C is no of children in the current directory.

Contributions

Main Functions -

Alias - Tirth Motka
Teleport - Abhijith A
Move, quit - Aryamaan Basu Roy
Find - Chinmay Deshpande
Add, main (interface) - Akhilesh Aravapalli

Additional Functions -

ls - Akhilesh Aravapalli
Cut & Paste - Aryamaan Basu Roy
Back - Akhilesh Aravapalli
Delete - Aryamaan Basu Roy
Findall - Chinmay Deshpande
PathFromPtr - Akhilesh Aravapalli
Hashtable implementation: Tirth Motka & Abhijith A

Link to GitHub Repo - https://github.com/aforakhilesh/DSA-Project-Team-31