Project 2 **Aditya Anil Raut**
012648884

Tries and Ternary Search Trees (TSTs) are data structures used for efficient storage and retrieval of strings.

Code Structure

Project3part1.cpp
- Class Trie
  - Insert function
  - Print function
  - Delete function

Project3part2.cpp
- Class TST
  - Insert function
  - Print function
  - Delete function

Time and Space Complexity Analysis

Trie Implementation

A trie is a tree-based data structure in computer science that is used to store and retrieve words or strings from a collection.

Insert Function

Recursion Equation

$T(L)=T(L-1)+O(1)$
$T(L)= T(L-2)+O(1)+O(1)=\ldots$
$T(L)= T(L-3)+O(1)+O(1)+O(1)$
$T(L)= T(L-4)+O(1)+O(1)+O(1)+O(1)$
$T(L)= T(0)+L*O(1)$
$T(L)= =T(0)+O(L)$
$T(L)= O(L)$

Time Complexity:

- Time complexity of insert is proportional to the length of the word: O(L) where L is the length of the word being inserted.

Space Complexity:

- For a word of length L, Space Complexity will be O(L) as L is the number of stack that will be opened

Print Function

Recursion Equation for printing a word of length L

$T(L)=T(L-1)+O(1)$
$T(L)= T(L-2)+O(1)+O(1)=\ldots$
$T(L)= T(L-3)+O(1)+O(1)+O(1)$
$T(L)= T(L-4)+O(1)+O(1)+O(1)+O(1)$
$T(L)= T(0)+L*O(1)$
$T(L)= T(0)+O(L)$

Time Complexity:
The time complexity of printing all the nodes in a trie is O(N), as the code performs Traversal on every node N

Space Complexity:

- The space complexity of the Trie while printing is O(L) where L is the height of tree.
- For a word of length L, Space Complexity will be O(L) as L is also the number of stack that will be opened. Maximum Space ever would be O(height of tree)

Delete Function
T(n)= 26T(n/26) +C
Giving us the time complexity as O(n)

Space Complexity:

- For a word of length L, Space Complexity will be O(L) as L is the number of stack that will be opened. Maximum Space ever would be O(height of tree)

Trie

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
cc1plus: fatal error: main.cpp: No such file or directory
compilation terminated.
aditya@Aditya:/mnt/c/Users/adity/Desktop/project3$ g++ project3part1.cpp
aditya@Aditya:/mnt/c/Users/adity/Desktop/project3$ valgrind ./a.out < t01.in > t01.out --leak-check=full
==415== Memcheck, a memory error detector
==415== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==415== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==415== Command: ./a.out --leak-check=full
==415==
==415==
==415== HEAP SUMMARY:
==415==     in use at exit: 0 bytes in 0 blocks
==415==   total heap usage: 70 allocs, 70 frees, 87,167 bytes allocated
==415==
==415== All heap blocks were freed -- no leaks are possible
==415==
==415== For lists of detected and suppressed errors, rerun with: -s
==415== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
aditya@Aditya:/mnt/c/Users/adity/Desktop/project3$
```

TST Implementation

A ternary search tree is a type of trie data structure where the child nodes are ordered as a binary search tree with 3 pointers.

Insert Function

Recursion Equation

$T(L)=T(L-1)+O(1)$
$T(L)= T(L-2)+O(1)+O(1)=\ldots$
$T(L)= T(L-3)+O(1)+O(1)+O(1)$
$T(L)= T(L-4)+O(1)+O(1)+O(1)+O(1)$
$T(L)= T(0)+L*O(1)$
$T(L)= =T(0)+O(L)$
$T(L)= O(L)$

Time Complexity:

- **Height of TST**: The height of the TST depends on the structure of the tree, which can range from O(logn) for a balanced tree O(n) for a degenerate (skewed) tree.
- **Per Word**: For a word of length L, the function traverses L levels in the tree.
- **Overall Complexity**:
  - Best case (balanced tree): O(L*log n)
  - Worst case (skewed tree): O(L*n)

Space Complexity:

- For a word of length L, Space Complexity will be O(L) as L is the number of stack that will be opened

Print Function

Time Complexity:

$T(L)=T(L-1)+O(1)$
$T(L)= T(L-2)+O(1)+O(1)=\ldots$
$T(L)= T(L-3)+O(1)+O(1)+O(1)$
$T(L)= T(L-4)+O(1)+O(1)+O(1)+O(1)$
$T(L)= T(0)+L*O(1)$
$T(L)= T(0)+O(L)$
$T(L)= O(L)$

Traversal:

- The function visits each node in the TST exactly once. For N nodes, the traversal complexity is O(N)

- **Line Number Printing**:

- For each word, printing the line numbers takes O(k) where k is the number of lines associated with the word.
- Let M be the total number of line numbers across all nodes. The total time complexity becomes: O(N+M)

Delete Function

Time Complexity:

T(n)= 3t(n/3) +C
T(n)= 9T(n/9)+4C
T(n)= 27T(n/27)+7C
T(n)=3^k T(n/3^k)+((3^k−1)/2)*C
Therefore for the base case
n/(3**k)=1. Therefore k=log3n
Substitute k=log3n
Giving us the time complexity as O(n)

Space Complexity:

- For a word of length L, Space Complexity will be O(L) as L is the number of stack that will be opened. Maximum Space ever would be O(height of tree)

TST