INFORMATION TECHNOLOGY UNIVERSITY

Software Engineering Department - ITU

SE200L: Data Structures & Algorithms Lab

Course Instructor: Usama Bin Shakeel	**Dated: 06/11/2024**
Teaching Assistant: Zainab Bashir & Ryan Naveed	**Semester: Fall 2024**
Lab Engineer: Sadia Ijaz	**Batch: BSSE2023B**

Lab 10. Implementation of Trie Data Structure

Name	Roll number	Report (out of 35)

Checked on: _____

Signature: _____

1.1 Objective

The objective of this lab is to practice problems related to the implementation of Trie.

1.2 Equipment and Component

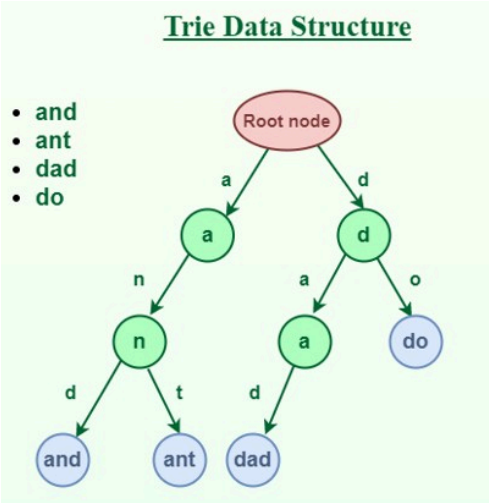
Component Description	Value	Quantity
Computer	Available in lab	1

1.3 Conduct of Lab

1. Students are required to perform this experiment individually.
2. In case the lab experiment is not understood, the students are advised to seek help from the course instructor, lab engineers, assigned teaching assistants (TA) and lab attendants.

1.4 Theory and Background

A **Trie** (pronounced as "try") is a type of search tree used to store a dynamic set of strings where keys are usually strings. Unlike a binary search tree, where keys are compared one by one, in a Trie, the position of a key is determined by the key's characters and the structure of the tree.



Structure of a Trie

The structure of a Trie consists of nodes, where each node represents a single character of a key. The root node is usually empty and serves as the starting point for all keys. Each node contains an array (or a similar structure) of child pointers, with each child corresponding to a possible character that can follow the current character in a key. For example, in a Trie that stores lowercase English letters, each node would have an array of 26 pointers, one for each letter from 'a' to 'z'.

Each path from the root to a node represents a prefix of some keys. The end of a key is typically marked by a boolean flag (often called isEndOfWord) in the node, indicating whether that particular node completes a valid key in the Trie.

INFORMATION TECHNOLOGY UNIVERSITY

Lab Tasks

Task 1:

Implement TrieNode Class

Class Name: TrieNode

Data Members:

- **bool isEndOfWord:** A flag indicating if the current node marks the end of a word.
- **TrieNode* children[52]**: An array of pointers to TrieNode representing 26 lowercase ('a' to 'z') and 26 uppercase ('A' to 'Z') English letters.

Function Prototypes:

- **TrieNode()**: Constructor that initializes isEndOfWord to false and sets all elements of children to nullptr.

Task 2:

Implement Trie Class

Class Name: Trie

Data Members:

- **TrieNode* root:** Pointer to the root node of the Trie.

Function Prototypes:

1. **Trie()**: Constructor for initializing the Trie.
2. **void insert(const string& word)**: Inserts a word into the Trie.
3. **bool search(const string& word)**: Searches for a word in the Trie and returns true if found, false otherwise.
4. **bool startsWith(const string& prefix)**: Checks if any word in the Trie starts with the given prefix.
5. **bool deleteWord(const string& word):** Deletes a word from the Trie.
6. **void displayTrie():** Prints all words stored in the Trie in lexicographical order.

Instructions:

1. **Insert Function**:
    - Convert the input word character-by-character to determine the correct index:
        - For lowercase: index = ch - 'a'

        - For uppercase: index = ch - 'A' + 26
    - Ensure that the index is valid before accessing the children array.
2. **Search Function**:
    - Traverse the Trie using the calculated index for each character.
    - Return false if any character node is missing. Return true if the end of the word is reached and isEndOfWord is true.
3. **StartsWith Function**:
    - Similar to search(), but only checks that the prefix exists without considering isEndOfWord.
4. **DeleteWord Function**:
    - Implement a recursive approach to delete a word and clean up unused nodes.
5. **DisplayTrie Function**:
    - Use a helper function to traverse and print all words in lexicographical order.

Please read the following instructions carefully:

1. ***Do Not Modify test.cpp:*** *You are strictly prohibited from making any changes to the test.cpp file. This file is designed to test your implementation and any modifications will lead to the assignment being graded as zero.*
2. ***Class Definitions:*** *All class definitions and implementations must be provided solely within the files functions.h and functions.cpp. You are not allowed to create any additional files for your class definitions or implementations.*

Any deviation from these rules, including creating additional files or modifying the test.cpp file, will result in your assignment receiving a grade of zero.

Assessment Rubric for Lab

Performance metric	CLO	Able to complete the task over 80% (4-5)	Able to complete the task 50-80% (2-3)	Able to complete the task below 50% (0-1)	Marks
1. Realization of experiment	1	Executes without errors excellent user prompts, good use of symbols, spacing in output. The testing has been completed.	Executes without errors, user prompts are understandable,minimum use of symbols or spacing in output. Some testing has been completed.	Does not execute due to syntax errors, runtime errors, user prompts are misleading or non- existent. No testing has been completed.	
2. Conducting experiment	1	Able to make changes and answer all questions.	Partially able to make changes and few incorrect answers.	Unable to make changes and answer all questions.	
3. Computer use	2	Document submission timely.	Document submission late.	Document submission not done.	
4. Teamwork	3	Actively engages and cooperates with other group member(s) in an effective manner.	Cooperates with other group member(s) in a reasonable manner but conduct can be improved.	Distracts or discourages other group members from conducting the experiment	
5. Laboratory safety and disciplinary rules	3	Code comments are added and do help the reader to understand the code.	Code comments are added and do not help the reader to understand the code.	Code comments are not added.	
6. Data collection	3	Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap.	Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables.	Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy.	
7. Data analysis	4	Solution is efficient, easy to understand, and maintain.	A logical solution that is easy to follow but it is not the most efficient.	A difficult and inefficient solution.	
Total (out of 35):					