JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY

BACHELOR OF TECHNOLOGY, 6th SEMESTER

Non-Linear Data Structures Project

TOPIC:- DICTIONARY USING TRIE

Submitted By:

Tanmay Agrawal 17103350

Rishabh Kejariwal 17103355

Abhishek Srivastava 17103334

Akash Kumar Rai 17103177

Purpose

To provide an implementation of a dictionary, that contains words with their relevant descriptions, and allows the user to populate it with large amounts of information and still remain searchable in an acceptable and efficient timeframe.

Implementation Notes

The implementation is backed by a 'trie', commonly referred to as a 'digital tree' or a 'prefix tree'.

Unlike a binary tree that simply has two child nodes, each node has a maximum of 26 child nodes which account for each letter in the alphabet.

Each child node then has a string value assigned to it, which would be the description of the word if it has been added, and its own set of child nodes.

This approach applies itself well to a dictionary case scenario as traversing through the trie results in descending through the tree with each letter of the word.

Using this method for traversal allows us to operate in an O(m) timeframe where m is the length of the word. This leads to a linear performance given the length of the word to search for. This performance will be incredibly valuable for largely populated dictionaries as the number of words within the dictionary will have no effect on the timeframe that it can be traversed in.

How to Implement?

Hashing is one simple option for this. We can put all words in a hash table. But hashing doesn’t support operations like prefix search. Prefix search is something where a user types a prefix and your dictionary shows all words starting with that prefix. Hashing also doesn’t support the efficient printing of all words in the dictionary in alphabetical order and nearest neighbor search.

If we want both operations, look up, and prefix search, Trie is suited. With Trie, we can support all operations like insert, search, delete in O(n) time where n is the length of the word to be processed. Another advantage of Trie is, we can print all words in alphabetical order which is not possible with hashing.

Key features

Trie is the key data structure used to create the dictionary.

- Use of **file handling** to store the words with their meaning.
- **Inserting** a new word in the dictionary.
- **Searching** for the word already present in it.
- **Deletion** of a word stored in the dictionary.
- Printing the words in **alphabetical order**.
- **Prefix searching** i.e all words with the entered prefix will be displayed.
- **Nearest neighbour search** ie. Shows suggested words if the word you are looking for doesn't exist.

Running The Program:

- We have coded the following project in C++ and it's stored in a file named code.cpp
- As it is dictionary we need to have some words stored beforehand so some words are stored in a file named "word.txt" as soon you run the program all words are fetched from the file and inserted in the trie as a part of pre-processing.
- GUI of the program is self-explanatory.

Thank You