DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJS22ITL502 **DATE:26/08/2024**

COURSE NAME: Advanced Data Structures Laboratory **CLASS:** TY B. TECH

NAME: Anish Sharma **SAP**: 60003220045

EXPERIMENT NO. 2

CO/LO: Choose appropriate data structure and use it to design algorithm for solving a specific problem

AIM / OBJECTIVE: To implement various operations on standard trie

DESCRIPTION OF EXPERIMENT:

The word "Trie" is an excerpt from the word "retrieval". Trie is a sorted tree-based data-structure that stores the set of strings. It has the number of pointers equal to the number of characters of the alphabet in each node. It can search a word in the dictionary with the help of the word's prefix. For example, if we assume that all strings are formed from the letters 'a' to 'z' in the English alphabet, each trie node can have a maximum of 26 points. Trie is also known as the digital tree or prefix tree. The position of a node in the Trie determines the key with which that node is connected.

Properties of the Trie for a set of the string:

- The root node of the trie always represents the null node.
- Each child of nodes is sorted alphabetically.
- Each node can have a maximum of 26 children (A to Z).
- Each node (except the root) can store one letter of the alphabet.

SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

#define ALPHA 26

struct Trienode {
    struct Trienode *child[ALPHA];
    bool isEnd;
};

struct Trienode *getNode() {
    struct Trienode *p = (struct Trienode *)malloc(sizeof(struct Trienode));
    p->isEnd = false;

    for (int i = 0; i < ALPHA; i++) {
        p->child[i] = NULL;
    }
    return p;
}

void insert(struct Trienode *root, const char *key) {
    struct Trienode *pcrawl = root;
    for (int level = 0; level < strlen(key); level++) {
        int index = key[level] - 'a';
        if (!pcrawl->child[index]) {
            pcrawl->child[index] = getNode();
        }
        pcrawl = pcrawl->child[index];
    }
    pcrawl->isEnd = true;
}

bool search(struct Trienode *root, const char *key) {
```

```
    struct Trienode *pcrawl = root;
    for (int level = 0; level < strlen(key); level++) {
        int index = key[level] - 'a';
        if (!pcrawl->child[index]) {
            return false;
        }
        pcrawl = pcrawl->child[index];
    }
    return (pcrawl != NULL && pcrawl->isEnd);
}

bool isEmpty(struct Trienode* root) {
    for (int i = 0; i < ALPHA; i++) {
        if (root->child[i]) {
            return false;
        }
    }
    return true;
}

struct Trienode *delete(struct Trienode *root, const char *key, int depth) {
    if (!root) {
        return NULL;
    }
    if (depth == strlen(key)) {
        if (root->isEnd) {
            root->isEnd = false;
        }
        if (isEmpty(root)) {
            free(root);
            root = NULL;
        }
        return root;
    }
    int index = key[depth] - 'a';
    root->child[index] = delete(root->child[index], key, depth + 1);
```

```
    if (isEmpty(root) && !root->isEnd) {
        free(root);
        root = NULL;
    }
    return root;
}

int main() {
    char key[][5] = {"an", "ant", "and", "bat", "bats"};
    struct Trienode *root = getNode();

    for (int i = 0; i < sizeof(key) / sizeof(key[0]); i++) {
        insert(root, key[i]);
    }

    printf("%s --- %s\n", "an", search(root, "an") ? "Present in trie" : "Not present in trie");
    printf("%s --- %s\n", "ants", search(root, "ants") ? "Present in trie" : "Not present in
trie");
    printf("%s --- %s\n", "ant", search(root, "ant") ? "Present in trie" : "Not present in trie");
    printf("%s --- %s\n", "bat", search(root, "bat") ? "Present in trie" : "Not present in trie");
    printf("%s --- %s\n", "bats", search(root, "bats") ? "Present in trie" : "Not present in
trie");

    delete(root, "bat", 0);

    printf("\nbat deleted\n");
    printf("%s --- %s\n", "bat", search(root, "bat") ? "Present in trie" : "Not present in trie");
    printf("%s --- %s\n", "bats", search(root, "bats") ? "Present in trie" : "Not present in
trie");

    return 0;
}
```

CONCLUSION: I understood the concept of Tries.

REFERENCES:

1. Peter Brass, “Advanced Data Structures”, Cambridge University Press, 2008
2. Robert Sedgewick & Kevin Wayne, “Algorithms”, 4th Edition, Addison-Wesley Professional, 2011.

3. https://www.javatpoint.com/trie-data-structure