**School of Electronics and Communication Engineering**

**Second Year B. Tech.**

**Project Report on:**

**Spelling Checker using Trie**

**Group Members:**

1. Gaurav Prajapati – 1032221120
2. Sourabh Bhosale –
3. Vivaswan Sapre –

**Subject Teacher:** Ms. Alka Barhatte

**Subject:** Data Structures and Algorithms

**Div.:** B

**Academic Year:** 2023-24 (Semester-I)

**Problem Statement:**

While typing a word on our phones or computers, there can exist mistyped words. How do we check whether the word is correct quickly.

**Aim:**

To check if a word is spelled correct using the Trie Data Structure

**Objective:**

**Introduction:**

The ‘Trie’ is a special type of Tree Data Structure whose node values are only letters of the alphabet [‘a’ to ‘z’ and ‘A’ to ‘Z’]. This type of data structure can be efficiently used to store multiple words efficiently and is perfect for storing words because some words have overlaps [eg. ‘hell’ and ‘hello’]. When words have similar prefixes, the trie just appends to the end of prefix instead of starting a new list for each word.

**Algorithm/Flowchart:**

**Code:**

#include *<stdio.h>*

#include *<stdlib.h>*

#include *<string.h>*

#include *<stdbool.h>*

#include *<assert.h>*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

|  \* .-. .-. .-. .   .   .-. . . .-. .-. . .   |

|  \* `-. |-' |-  |   |   |   |-| |-  |   |<    |

|  \* `-' '   `-' `-' `-' `-' ' ` `-' `-' ' `   |

|  \* Spelling Checker Application              |

|----------------------------------------------|

|  \* @author Gaurav Prajapati   11             |

|  \* @author Sourabh Bhosale    48             |

|  \* @author Vivaswan Sapre     57             |

|----------------------------------------------|

|  \* SY BTech ECE                              |

|  \* Div. B - B3                               |

|----------------------------------------------|

|  \* This is a C program to check correctness  |

|  \* of an input spelling.                     |

|                                              |

|  \* It uses a trie data structure having      |

|  \* 27 subnodes for storing words.            |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// macro for freeing pointer

#define freeptr(\_ptr) if ((\_ptr) != **NULL**) free((\_ptr));

int conv2index(char letter) {

        switch (letter)

        {

        // if case range labels are OK:

        // case 'A' ... 'Z':

        //     letter += 32;

        // case 'a' ... 'z':

        //     index = letter - 97;

        //     break;

        // default:

        //     index = 26;

        //     break;

        case *'***\'***'*:

            return **26**;

        default:

            if(letter <= *'Z'*) letter += **32**;

            return (letter - **97**);

        }

        return -**1**;

}

// structure definition for trie

typedef struct trie\_node

{

    bool end;

    struct trie\_node \*\*letters;

} tnode;

/\*\*

 \* Node Creation Function for trie

 \* @retval tnode\*

 \*/

tnode \*trie\_init()

{

    tnode \*new = (tnode \*)calloc(**1**, sizeof(tnode));

    assert(new != **NULL**);

    // Set struct Values

    new->end = **false**;

    new->letters = (tnode \*\*)malloc(**27** \* sizeof(tnode \*));

    // initialize garbage values to NULL

    for(int i = **0**; i < **27**; i++) new->letters[i] = **NULL**;

    return new;

}

/\*\*

 \* insert a word inside the dictionary

 \* @param \*Word pointer to Word to be added

 \* @param \*Dictionary Pointer to dictionary to add to

 \*/

void str\_to\_tree(char \*Word, tnode \*Dictionary)

{

    tnode \*current = Dictionary;

    int index = **26**;

    int len = strlen(Word);

    char letter;

    for (int i = **0**; i < len && ((letter = Word[i]) != *'***\0***'*); i++)

    {

        index = conv2index(letter);

        if (current->letters[index] == **NULL**)

        {

            current->letters[index] = trie\_init();

        }

        current = current->letters[index];

    }

    current->end ^= **true**;

}

/\*\* check spelling using trie

 \* @param \*Word pointer to Word to be checked

 \* @param \*Dictionary pointer to dictionary to check with

 \*/

bool checkspell(char \*Word, tnode \*Dictionary)

{

    tnode \*current;

    current = Dictionary;

    int index = **26**;

    int len = strlen(Word);

    // for (char \*letter = Word; \*letter != '\0' && \*letter != ' ' && \*letter != '\n'; letter++)

    for(int i = **0**; i < len; i++)

    {

        char letter = Word[i];

        if(letter == *'***\0***'* || letter == *' '* || letter == *'***\n***'*) break;

        index = conv2index(letter);

        current = current->letters[index];

        if (current == **NULL**)

            return **false**;

    }

    return current->end;

}

bool checksentence(char \*Sentence, tnode \*Dictionary)

{

    bool res = **true**;

    for (char \*lptr = Sentence; res && \*lptr != *'***\0***'*; lptr++)

    {

        if (lptr == Sentence)

            res = checkspell(lptr, Dictionary);

        else if (\*lptr == *' '*)

            res = checkspell(++lptr, Dictionary);

    }

    return res;

}

void free\_trie(tnode \*\*head)

{

    if (\*head == **NULL**)

        return;

    for (int i = **0**; i < **27**; i++)

    {

        free\_trie(&((\*head)->letters[i]));

    }

    freeptr(\*head);

}

/\*\*

 \* Dictionary to be preloaded is stored

 \* in Plaintext[.txt] format.

 \*

 \* Each word is on a new line

 \*

 \* FILE STRUCTURE:

 \* +-------------------+

 \* | word1(\n)         |

 \* | word2(\n)         |

 \* | ...               |

 \* | lastWord(EOF)     |

 \* +-------------------+

 \*/

tnode \*file\_to\_trie()

{

    tnode \*new = trie\_init(), \*current;

    current = new;

    FILE \*dfile = fopen(*"./res/words.txt"*, *"r"*);

    // assert(dfile != NULL);

    char letter;

    int index;

    while ((letter = fgetc(dfile)) != EOF)

    {

        switch (letter)

        {

        case *'***\n***'*:

            current->end = **true**;

            current = new;

            break;

        default:

            index = conv2index(letter);

            if(current->letters[index] == **NULL**)

            {

                current->letters[index] = trie\_init();

                current->letters[index]->end = **false**;

            }

            current = current->letters[index];

            break;

        }

    }

    current->end = **true**;

    fclose(dfile);

    current = **NULL**;

    return new;

}

/\*\*

 \* Frames for the UI are stored in ui.txt

 \* Each frame is 44x15+8 ~ 700

 \*   44 -> columns

 \*   15 -> rows

 \*   8 -> VT100 sequence terms

 \* Total of 5 frames:

 \*   1  -> STARTUP/IDLE

 \*  2-5 -> MENU CHOICE

 \*/

// Global Array to hold frames of UI

static char frames[**5**][**700**];

// Global string holding output from last executed command

static char message[**70**];

// initialize frames from res/ui.txt

void init\_frames()

{

    FILE \*ui\_file = **NULL**;

    // open res/ui.txt for reading frames

    ui\_file = fopen(*"./res/ui.txt"*, *"r+"*);

    if (ui\_file == **NULL**)

        perror(*"Error opening file"*);

    for (int i = **0**; i < **5**; i++)

    {

        char px = fgetc(ui\_file);

        int j, x\_cnt = -**2**;

        for (j = **0**; x\_cnt && px != EOF; j++)

        {

            switch (px)

            {

            case *'***\\***'*:

                fgetc(ui\_file);

                frames[i][j] = *'***\033***'*;

                break;

            default:

                frames[i][j] = px;

                break;

            }

            px = fgetc(ui\_file);

            if (px == *'X'*)

                x\_cnt++;

        }

        frames[i][j] = *'***\0***'*;

        fgetc(ui\_file);

    }

    fclose(ui\_file);

}

// Display UI Frame

// @param Frame Frame number to be displayed

void display\_ui(int Frame)

{

    if (!(Frame >= **0** && Frame < **5**))

        return;

    // reset cursor position and style

    printf(*"***\033***[2J***\033***[H***\033***[0;0m"*);

    // print frame

    printf(*"***%s***"*, frames[Frame]);

    // save input cursor position

    printf(*"***\033***[2A***\033***[6C***\033***[s"*);

    // move cursor to message area

    printf(*"***\033***[H***\033***[5B***\033***[4C"*);

    // print message

    printf(*"***%s***"*, message);

    // restore saved cursor position

    printf(*"***\033***[u"*);

}

int main()

{

    init\_frames();

    tnode \*DICT = file\_to\_trie();

    int opt = **0**;

    bool res = **false**;

    // start alternative buffer

    printf(*"***\033***[?1049h"*);

    do

    {

        char buf[**32**];

        display\_ui(**0**);

        scanf(*"***%d***"*, &opt);

        display\_ui(opt);

        switch (opt)

        {

        case **1**:

            fflush(stdin);

            scanf(*"***%s***"*, buf);

            str\_to\_tree(buf, DICT);

            strcpy(message, *"***\0***"*);

            sprintf(message, *"***\033***[32mSuccessfully added* **%s\033***[0;0m"*, buf);

            break;

        case **2**:

            fflush(stdin);

            scanf(*"***%s***"*, buf);

            str\_to\_tree(buf, DICT);

            strcpy(message, *"***\0***"*);

            sprintf(message, *"***\033***[32mSuccessfully deleted* **%s\033***[0;0m"*, buf);

            break;

        case **3**:

            fflush(stdin);

            scanf(*"***%s***"*, buf);

            res = checkspell(buf, DICT);

            strcpy(message, *"***\0***"*);

            if (res)

                sprintf(message, *"***\033***[32m***%s** *is Correct!!***\033***[0;0m"*, buf);

            else

                sprintf(message, *"***\033***[31m***%s** *is Wrong!!***\033***[0;0m"*, buf);

            break;

        case **4**:

            fflush(stdin);

            scanf(*"%[***\n***]%[^***\n***]"*, buf, buf);

            res = checksentence(buf, DICT);

            strcpy(message, *"***\0***"*);

            if (res)

                sprintf(message, *"***\033***[32mSentence is spelled Correct!!***\033***[0;0m"*);

            else

                sprintf(message, *"***\033***[31mSentence has Wrong spellings!!***\033***[0;0m"*);

            break;

        default:

            break;

        }

    } while (opt);

    // close alternate buffer

    printf(*"***\033***[?1049l"*);

    // free up memory taken by trie

    free\_trie(&DICT);

    return **0**;

}