**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:**

The project aims to check the grammatical correctness of a singular word or an array of words on the basis of English spelling, and display the same in a easy to use Text UI.

**Objective:**

To efficiently store a dictionary of English words

**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 *<string.h>*

#include *<assert.h>*

#include *<stdbool.h>*

#include *<stdlib.h>*

#include *<stdio.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.            |

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

// structure definition for trie

*struct* trie\_node {

*bool* end;

*struct* trie\_node \*\*letters;

}

tnode;

*int* tn\_index(*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**;

}

/\*\*

 \* Node Creation Function for trie

 \* *@retval* tnode\*

 \*/

tnode \*tn\_new()

{

    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* tn\_insert(*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++)

    {

        if(letter == *' '* || letter == *'.'* || letter == *','*) break;

        index = tn\_index(letter);

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

        {

            current->letters[index] = tn\_new();

        }

        current = current->letters[index];

    }

    current->end = **true**;

}

/\*\* Delete word from dicitionary

 \* *@param* \*Word word to remove

 \* *@param* \*Dictionary trie to delete from

 \*/

*bool* tn\_delete(*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++)

    {

        if(letter == *' '* || letter == *'.'* || letter == *','*) break;

        index = tn\_index(letter);

        // word does not exist

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

        current = current->letters[index];

    }

    current->end = **false**;

    // successful deletion

    return **true**;

}

/\*\* check spelling using trie

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

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

 \*/

*bool* check\_word(*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 = tn\_index(letter);

        current = current->letters[index];

        if (current == **NULL**)

            return **false**;

    }

    return current->end;

}

/\*\*

 \* *@brief*  Check an english sentence against a dictionary

 \* *@note* Uses check\_word() which ends words at [',' '.' ' '] and so, only requires pointer to first character of word

 \* *@param*  \*Sentence: Sentence to be checked

 \* *@param*  \*Dictionary: Trie to check against

 \* *@retval*  \*true\* if no wrongly spelled words, \*false\* otherwise

 \*/

*bool* check\_sentence(*char* \**Sentence*, tnode \**Dictionary*)

{

*bool* res = **true**;

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

    {

        if (lptr == Sentence)

            res = check\_word(lptr, Dictionary);

        else if (\*lptr == *' '*)

            res = check\_word(++lptr, Dictionary);

    }

    return res;

}

/\*\*

 \* *@brief*  Free up memory used by trie

 \* *@note*   Similar to postorder traversal of BST

 \* *@param*  \*\*head: Pointer to a Pointer Variable containing address of root of tree

 \* *@retval* None

 \*/

*void* tn\_free(tnode \*\**head*)

{

    if (\*head == **NULL**)

        return;

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

        tn\_free(&((\*head)->letters[i]));

    if((\*head) != **NULL**) free((\*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)     |

 \* +-------------------+

 \*/

/\*\*

 \* *@brief*  preload words into trie from res/words.txt

 \* *@retval* Address of head node of created trie

 \*/

tnode \*file\_to\_trie()

{

    tnode \*new = tn\_new(), \*current;

    current = new;

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

    // assert(dfile != NULL);

*char* letter;

*int* index;

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

    {

        switch (letter)

        {

        // '\r' -> Carriage Return

        // This case is to fix any issues due to presence of \r

        case *'***\r***'*:

            break;

        case *'***\n***'*:

            current->end = **true**;

            current = new;

            break;

        default:

            index = tn\_index(letter);

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

            {

                current->letters[index] = tn\_new();

                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

 \*/

// Array to hold frames of UI

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

// 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"*);

}

// Set Message

// @param Key int

// @param success bool

// @param str char\*

*void* set\_message(*int* *Key*, bool *success*, *char* \**str*)

{

    // Clear str message

    strcpy(message, *"***\0***"*);

    switch (*Key*)

    {

    case **1**:

        if(*success*)

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

        else

            sprintf(message, *"***\033***[31mError in adding* **%s\033***[0;0m"*, *str*);

        break;

    case **2**:

        if(*success*)

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

        else

            sprintf(message, *"***\033***[31m***%s** *not in Dictionary***\033***[0;0m"*, *str*);

        break;

    case **3**:

        if (*success*)

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

        else

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

        break;

    case **4**:

        if (*success*)

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

        else

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

        break;

    default:

        sprintf(message, *"***\033***[31mInvalid Choice!!***\033***[0;0m"*);

        break;

    }

}

*int* main()

{

    init\_frames();

*tnode* \*DICT = file\_to\_trie();

*int* opt = **0**;

    bool res = false;

*char* buf[**32**],ch;

    // start alternative buffer

    printf(*"***\033***[?1049h"*);

    do

    {

        // display frame for menu [0]

        display\_ui(**0**);

        fflush(stdin);

        scanf(*"***%d***"*, &opt);

        // display selected option [opt]

        display\_ui(opt);

        fflush(stdin);

        switch (opt)

        {

        case **1**:

            scanf(*"***%s***"*, buf);

            tn\_insert(buf, DICT);

            if(check\_word(buf,DICT)) res = true;

            else res = false;

            break;

        case **2**:

            scanf(*"***%s***"*, buf);

            res = tn\_delete(buf, DICT);

            break;

        case **3**:

            scanf(*"***%s***"*, buf);

            res = check\_word(buf, DICT);

            break;

        case **4**:

            // get rid of leading \n in input buffer

            while((ch = fgetc(stdin)) == *'***\n***'*);

            buf[**0**] = ch;

            buf[**1**] = *'***\0***'*;

            scanf(*"*%*[^***\n***]"*, buf+**1**);

            res = check\_sentence(buf, DICT);

            break;

        default:

            break;

        }

        // set message line

        set\_message(opt,res,buf);

    } while (opt);

    // close alternate buffer

    printf(*"***\033***[?1049l"*);

    // free up memory taken by trie

    tn\_free(&DICT);

    return **0**;

}