**DATA STRUCTURE**

**PROJECT**

**Members:**

**Name:** Faizan Tariq

**Roll No:** 22F-3858

**Name:** Hassaan Ch

**Roll No:** 22F-3060

I have implemented Dictionary project using “trie” data structure. I found this way more better for displaying suggestions, searching and deleting etc.

Here is some information about time and space complexity of functions in my code:  
**Time Complexity:**

1. Insert Function : Time complexity for inserting word is equal to length of word. Order of ‘k’.
2. Search Function : Time complexity for searching word is is equal to length of word. Order of ‘k’.
3. Delete Function : Time complexity equal to order of ‘k’.
4. Auto-Suggestion: Time complexity in worst case is equal to number of nodes

**CODE:**

#include <iostream>

#include <fstream>

#include <string>

#include <Windows.h>

#include <conio.h>

#include <sstream>

using namespace std;

#define KEY\_UP 72

#define KEY\_DOWN 80

#define ESC\_KEY 27

const int ROWS = 10;

int activeSuggestionIndex = 0; // for managing suggestion to store in correct index of array

int arrowDisplayIndex = 0;

struct Suggestion {

string suggestionString;

string isActive = " ";

} SuggestionArray[ROWS];

void setConsoleTextColor(int color) {

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), color);

}

void displaySuggestionList() {

for (int i = 0; i < ROWS; i++)

{

if (SuggestionArray[i].suggestionString == "") continue;

if (SuggestionArray[i].isActive == "->") {

setConsoleTextColor(9);

//setConsoleTextColor(11);

cout << SuggestionArray[i].isActive << SuggestionArray[i].suggestionString << endl;

setConsoleTextColor(7);

}

else {

cout << SuggestionArray[i].isActive << SuggestionArray[i].suggestionString << endl;

}

}

}

void resetSuggestionList(int index) { // this index tell from where to start reseting SuggestionArray

arrowDisplayIndex = 0;

for (int i = index; i < ROWS; i++)

{

SuggestionArray[i].suggestionString = "";

SuggestionArray[i].isActive = " ";

}

SuggestionArray[arrowDisplayIndex].isActive = "->";

}

struct TrieNode {

TrieNode\* children[26];

bool isWordEnd; // tells if word in completed

string meaning; // contains meaning of word

TrieNode() : isWordEnd(false), meaning("") {

for (int i = 0; i < 26; i++) {

children[i] = NULL;

}

}

};

class Trie// main class for trie data structures

{

TrieNode\* root;

public:

Trie() : root(new TrieNode()) {}

TrieNode\* getRoot() {

return root;

}

void insert(string word, string meaning) //insert function for adding words to dictionary

{

TrieNode\* curNode = root;

for (int i = 0; i < word.length(); i++) {

char cur = word[i];

if (curNode->children[cur - 'a'] == NULL) {

TrieNode\* newNode = new TrieNode();

curNode->children[cur - 'a'] = newNode;

}

curNode = curNode->children[cur - 'a'];

}

curNode->meaning = meaning;

curNode->isWordEnd = true;

}

bool search(string word) {//search word from dictionary

TrieNode\* curNode = root;

for (int i = 0; i < word.length(); i++) {

char cur = word[i];

if (!curNode->children[cur - 'a']) {

return false;

}

curNode = curNode->children[cur - 'a'];

}

return curNode->isWordEnd;

}

void findMeaning(string word) {//search word and meaning from dictionary

TrieNode\* curNode = root;

for (int i = 0; i < word.length(); i++) {

char cur = word[i];

curNode = curNode->children[cur - 'a'];

}

cout << "\nWord: " << word;

cout << "\nMeaning: " << curNode->meaning;

}

void editWord(string word, string meaning) {//edit word and meaning from dictionary

TrieNode\* curNode = root;

for (int i = 0; i < word.length(); i++) {

char cur = word[i];

curNode = curNode->children[cur - 'a'];

}

curNode->meaning = meaning;

cout << "\nUpdated info :\n";

cout << "\nWord: " << word;

cout << "\nMeaning: " << curNode->meaning << endl;

}

bool isEmptyNode(TrieNode\* node) {

for (int i = 0; i < 26; i++) {

if (node->children[i] != NULL) {

return false;

}

}

return true;

}

void deleteWord(string word) {//calls deletion function

performDeletion(root, word, 0);// zero is depth here

}

void performDeletion(TrieNode\* curNode, string word, int dept) {//delete word from dictionary

if (curNode == NULL) {

return;

}

if (dept == word.length()) {

curNode->isWordEnd = false;

curNode->meaning = "";

return;

}

int index = word[dept] - 'a';

TrieNode\* nextNode = curNode->children[index];

if (nextNode != NULL) {

performDeletion(nextNode, word, dept + 1);

if (nextNode->isWordEnd == false && isEmptyNode(nextNode)) {

delete nextNode;

curNode->children[index] = NULL;

}

}

}

void loadDataFromFile() {//insert words from txt file to the trie data structure

ifstream fin("Dictionary.txt");

if (!fin.is\_open()) {

cout << "\nFile not found!\n";

return;

}

string line;

string word, meaning;

while (getline(fin, line)) {

istringstream wordAndMeaning(line);

wordAndMeaning >> word >> meaning;

insert(word, meaning);

}

}

void storeWords(TrieNode\* curNode, ofstream& fout, string curWord) {//store words to txt file

if (curNode == NULL) {

return;

}

if (curNode->isWordEnd) {

fout << curWord << "\t" << curNode->meaning << "\n";

}

for (int i = 0; i < 26; i++) {

if (curNode->children[i] != NULL) {

char nextCh = 'a' + i;

storeWords(curNode->children[i], fout, curWord + nextCh);

}

}

}

void safeToFile() {

ofstream fout("Dictionary.txt", ios::out);

if (!fout.is\_open()) {

cout << "\nFile not found.\n";

return;

}

storeWords(root, fout, "");

fout.close();

}

// Recursive function to print auto-suggestions for given node.

void suggestionsRec(TrieNode\* cur, string& currPrefix, int& count) {

if (count >= 10) { // limit displaying number of maximum suggestions

return;

}

if (cur->isWordEnd) {

//cout << currPrefix;

//cout << endl;

SuggestionArray[activeSuggestionIndex].suggestionString = currPrefix;

activeSuggestionIndex++;

resetSuggestionList(activeSuggestionIndex);// for reseting suggestion that i don't want to display

count++;

}

if (isEmptyNode(cur)) { // if cur Node is empty then remove last character and return this will go to one level up in tree

currPrefix.pop\_back();

return;

}

for (int i = 0; i < 26; i++) { // search in children array if it is not null then add that character to end of curPrefix and recursive call

if (cur->children[i]) {

currPrefix.push\_back('a' + i);

suggestionsRec(cur->children[i], currPrefix, count);

}

}

currPrefix.pop\_back();

}

int getSuggestions(string wordTosuggest) {//calls prit suggestion func

return printAutoSuggestions(root, wordTosuggest);

}

int printAutoSuggestions(TrieNode\* Root, string wordTosuggest) {

TrieNode\* pCrawl = Root;

int level; //current depth

int n = wordTosuggest.length(); // n is length of word

// cout << "HI"<<endl;

for (level = 0; level < n; level++) {

int index = wordTosuggest[level] - 'a';

if (!pCrawl->children[index]) // children[index] == NULL

{

return 0;

}

pCrawl = pCrawl->children[index]; // else traverse to next children[index] node

}

bool isWord = (pCrawl->isWordEnd == true); // this mean when pCrawl reach down length of word then check if isWordEnd is true?

bool isLast = isEmptyNode(pCrawl); // isEmptyNode Checks next children node to pCrawl that it is completly NULL or not

if (isWord && isLast) { // if it is both word and last word then print it

// cout << wordTosuggest << endl;

SuggestionArray[activeSuggestionIndex].suggestionString = wordTosuggest;

activeSuggestionIndex++;

resetSuggestionList(activeSuggestionIndex);// for reseting suggestion that i don't want to display

return -1;

}

if (!isLast) { // if it is not last then go to suggestionsRecursive function that will display in preOrder traversal

int count = 0;

suggestionsRec(pCrawl, wordTosuggest, count); // count gives number of max suggestions to display prefix is

return 1;

}

return 0;

}

};

void menu() {

setConsoleTextColor(14); // Yellow

cout << "\n-----------------------------\n";

cout << " DICTIONARY \n";

cout << "-----------------------------\n";

setConsoleTextColor(7); // White

cout << "1. Add Word\n";

cout << "2. Delete Word\n";

cout << "3. Search Word\n";

cout << "4. Suggest Word\n";

cout << "5. Edit Word\n";

cout << "6. Save and Exit\n";

setConsoleTextColor(11); // Light Cyan

cout << "-----------------------------\n";

setConsoleTextColor(7); // White

}

string toLowerCase(string S) {//converst upper case to lower case

for (int i = 0; i < S.length(); i++)

{

S[i] = tolower(S[i]);

}

return S;

}

bool isAlphabet(char key) {

return key >= 'A' && key <= 'Z' || key >= 'a' && key <= 'z';

}

void displaySuggestions(char& key, string& currentInput, Trie& dict, bool isBackspacePressed) {

if (isAlphabet(key)) {

activeSuggestionIndex = 0;

resetSuggestionList(activeSuggestionIndex);

if (!isBackspacePressed) {

currentInput += tolower(key); // if case is upper it convert that into lower

}

system("cls");

menu();

cout << "Current Input: " << currentInput;

cout << endl;

dict.printAutoSuggestions(dict.getRoot(), currentInput);

displaySuggestionList();

}

}

int main() {

Trie dict;

setConsoleTextColor(14); // Yellow

cout << "Loading";

for (int i = 0; i < 2; ++i) {

Sleep(1000 - i \* 150);

cout << ".";

}

dict.loadDataFromFile();

cout << ".";

Sleep(250);

againStart:

try {

setConsoleTextColor(7); // White

cout << "\n";

string choice = "0";

string currentInput = "";

system("CLS");

while (true) {

menu();

cout << "Enter your choice (1-6): ";

cin >> choice;

if (choice == "1") {

string word, meaning;

cout << "\nEnter Word :";

cin.ignore();// without this control ignores user to input his word..

getline(cin, word); // gets input in string with spaces..

bool isValid = true;

for (int i = 0; i < word.length(); i++)

{

if (!isAlphabet(word[i])) {

isValid = false;

}

}

if (isValid) {

word = toLowerCase(word);

if (!dict.search(word)) {

cout << "Enter Meaning :";

getline(cin, meaning);

meaning = toLowerCase(meaning);

cout << "INSERTING :" << word;

dict.insert(word, meaning);

}

else {

cout << "\nWord already exists in the dictionary.\n";

}

}

else {

cout << "\nInvalid Word\n";

}

}

else if (choice == "2") {

string word;

cout << "\nEnter Word to delete :";

cin.ignore();

getline(cin, word);

bool isValid = true;

for (int i = 0; i < word.length(); i++)

{

if (!isAlphabet(word[i])) {

isValid = false;

}

}

if (isValid) {

word = toLowerCase(word);

if (dict.search(word)) {

dict.deleteWord(word);

}

else {

cout << "\nWord does not exist.\n";

}

}

else{

cout << "\nInvalid Word\n";

}

}

else if (choice == "3") {

string word;

cout << "\nEnter Word to find meaning :";

cin.ignore();

getline(cin, word);

bool isValid = true;

for (int i = 0; i < word.length(); i++)

{

if (!isAlphabet(word[i])) {

isValid = false;

}

}

if (isValid) {

word = toLowerCase(word);

if (dict.search(word)) {

dict.findMeaning(word);

}

else {

cout << "\nWord does not exist.\n";

}

}

else {

cout << "\nInvalid Word!!\n";

}

}

else if (choice == "4") {

cout << "\nEnter the word to suggest: ";

currentInput = "";

bool exit = false;

while (!exit) {

if (\_kbhit()) {

char key = \_getch();

switch (key) {

case 13: // Enter key pressed

currentInput = SuggestionArray[arrowDisplayIndex].suggestionString;

system("cls");

menu();

cout << "Current Input: " << currentInput;

cout << endl;

if (dict.search(currentInput)) {

dict.findMeaning(currentInput);

}

break;

case 8: // Backspace key pressed

if (!currentInput.empty()) {

char lastCh = currentInput.back();

currentInput.pop\_back();

system("cls");

menu();

//cout << endl;

if (currentInput.empty()) {

cout << "Current Input: \n" ;

displaySuggestionList();

}

else {

displaySuggestions(lastCh, currentInput, dict, true); // 4th paramter is true when backspace key is pressed

}

}

break;

case KEY\_UP:

SuggestionArray[arrowDisplayIndex].isActive = " ";

if (arrowDisplayIndex == 0) arrowDisplayIndex = 9;

else {

arrowDisplayIndex--;

}

while (SuggestionArray[arrowDisplayIndex].suggestionString == "") {

arrowDisplayIndex--;

}

SuggestionArray[arrowDisplayIndex].isActive = "->";

system("cls");

menu();

cout << "Current Input: " << currentInput;

cout << endl;

displaySuggestionList();

break;

case KEY\_DOWN:

SuggestionArray[arrowDisplayIndex].isActive = " ";

if (arrowDisplayIndex == 9) arrowDisplayIndex = 0;

else {

arrowDisplayIndex++;

}

while (SuggestionArray[arrowDisplayIndex].suggestionString == "") {

arrowDisplayIndex++;

if (arrowDisplayIndex >= 9) arrowDisplayIndex = 0;

}

SuggestionArray[arrowDisplayIndex].isActive = "->";

system("cls");

menu();

cout << "Current Input: " << currentInput;

cout << endl;

displaySuggestionList();

break;

case ESC\_KEY:

activeSuggestionIndex = 0;

resetSuggestionList(activeSuggestionIndex);

system("CLS");

exit = true;

break;

default:

displaySuggestions(key, currentInput, dict, false);// 4th paramter is true when backspace key is pressed

break;

}

}

}

}

else if (choice == "5") {

string word, meaning = "";

cout << "\nEnter Word to edit :";

cin.ignore();

getline(cin, word);

bool isValid = true;

for (int i = 0; i < word.length(); i++)

{

if (!isAlphabet(word[i])) {

isValid = false;

}

}

if (isValid) {

word = toLowerCase(word);

if (dict.search(word)) {

cout << "Enter new meaning : ";

getline(cin, meaning);

meaning = toLowerCase(meaning);

dict.editWord(word, meaning);

}

else {

cout << "\nWord does not exist.\n";

}

}

else {

cout << "\nInvalid Word!\n";

}

}

else if (choice == "6") {

dict.safeToFile();

break;

}

else {

throw "Invalid Input!!";

}

}

}

catch (...) {

//cerr << "Invalid Input!!" << endl;

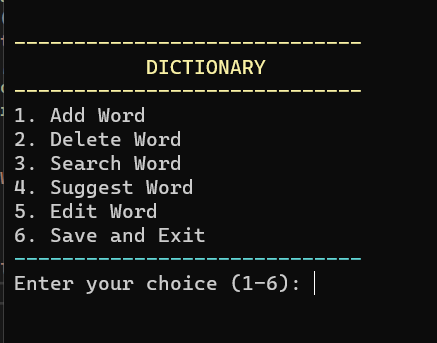
//cin.ignore(1000,'\n');

goto againStart;

}

return 0;

}

**OUTPUT:  
**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screen shot of a computer program

Description automatically generatedA screen shot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**