1. The reason why we thought this algorithm, is suitable for this case are:
   1. Compared to the other algorithm we believe that Tries, the algorithm we used is the most efficient, stable, and the quickest, when compared to the other algorithm that’s available for choosing.
   2. Another reasoning is, we believe that Trie is more efficient in memory management, because when there’s a lot of data, and a lot of string, which using the same prefix, the other algorithm will save those prefix as each individual prefix, while Trie only save one prefix.
   3. Also Tries, is more suitable for saving a dynamic prefix, which is a consistently changing according to the user.
2. The Literature Review that we choose are:
   1. Fredkin, Edward. "Trie memory." Communications of the ACM 3.9 (1960): 490-499
   2. Witten, Ian H., Alistair Moffat, and Timothy C. Bell. "Managing gigabytes: compressing and indexing documents and images." Morgan Kaufmann, 1999.
   3. Navarro, Gonzalo, and Mathieu Raffinot. "Flexible pattern matching in strings: practical on-line search algorithms for texts and biological sequences." Cambridge University Press, 2002.
3. The advantages of the algorithm we chooses are:
   1. The complexity of Tries can be categorized as high, which is O(n)
   2. Trie is more memory efficient compared to the other algorithm
   3. Trie suitable to be applied to a dynamic prefix
4. Features

A screen shot of a computer

Description automatically generated

There are 5 menus that can chose by user

A screen shot of a black screen

Description automatically generated

Features 1: When chose number 1 user can enter new slang word with the description for that word

A black screen with white text

Description automatically generated

Features 2: When chose number 2 user can search slang word inputted by name, then if found will display the word and its description

A black screen with white text

Description automatically generated

Features 3: When chose number 3 user can search slang word that has been inputted by prefix. Example: if user type prefix ‘s’, will show words that started with ‘s’

A black screen with white text

Description automatically generated

Features 4: When chose number 4 user can view all words inputted

A black screen with white text

Description automatically generated

Features 5: User can exit program by chose number 5

FULL CODE:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

#include <ctype.h>

#define ALPHABET\_SIZE 26 //A macro to change every "ALPHABET\_SIZE" to 26 in this code

// Trie node structure

struct TrieNode {

struct TrieNode\* children[ALPHABET\_SIZE]; // Array of pointers to child nodes, one for each letter of the alphabet (26 letters)

bool isEndOfWord; //True or False value, will return true if the node represent the last character of valid word

char\* description; // Pointer to a string containing a description or additional information about the word

};

// Function to create a new Trie node

struct TrieNode\* createNode() {

// Allocate memory for a new Trie node

struct TrieNode\* node = (struct TrieNode\*)malloc(sizeof(struct TrieNode));

// Check if memory allocation was successful

if (node) {

// Initialize the isEndOfWord flag to false

node->isEndOfWord = false;

// Initialize the description pointer to NULL

node->description = NULL;

// Initialize all children pointers to NULL

for (int i = 0; i < ALPHABET\_SIZE; i++) {

node->children[i] = NULL;

}

}

// Return the newly created node

return node;

}

// Function to insert a word into the Trie

void insert(struct TrieNode\* root, const char\* word, const char\* description) {

// Start at the root node

struct TrieNode\* node = root;

// Traverse the Trie based on each character in the word

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

// Calculate the index of the character

int index = tolower(word[i]) - 'a';

// If the child node for this character doesn't exist, create it

if (!node->children[index]) {

node->children[index] = createNode();

}

// Move to the child node

node = node->children[index];

}

// Mark the end of the word

node->isEndOfWord = true;

// Free existing description if any, then allocate and copy the new description

if (node->description) {

free(node->description);

}

// Duplicate the new description and assign it to the node's description field

node->description = strdup(description);

}

// Function to search for a word in the Trie

char\* search(struct TrieNode\* root, const char\* word) {

// Start at the root node

struct TrieNode\* node = root;

// Traverse the Trie based on each character in the word

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

// Calculate the index of the character

int index = tolower(word[i]) - 'a';

// If the child node for this character doesn't exist, return NULL

if (!node->children[index]) {

return NULL;

}

// Move to the child node

node = node->children[index];

}

// After traversing all characters, check if the current node marks the end of a word

if (node != NULL && node->isEndOfWord) {

// Return the description if the word is found

return node->description;

}

// Return NULL if the word is not found or if it's a prefix but not a complete word kayae

return NULL;

}

// Function to print words recursively

void printWords(struct TrieNode\* node, const char\* prefix) {

// If the current node marks the end of a word, print the prefix

if (node->isEndOfWord) {

printf("%s\n", prefix);

}

// Traverse all possible children nodes

for (int i = 0; i < ALPHABET\_SIZE; i++) {

// If a child node exists at index i

if (node->children[i]) {

// Calculate the corresponding character

char c = 'a' + i;

// Allocate memory for the new prefix

char\* newPrefix = (char\*)malloc(strlen(prefix) + 2); // +2 for the new character and null terminator

// Copy the existing prefix to the new prefix

strcpy(newPrefix, prefix);

// Append the new character to the new prefix

strncat(newPrefix, &c, 1);

// (Recursive) will call printWords on the child node with the new prefix

printWords(node->children[i], newPrefix);

// Free the allocated memory for the new prefix

free(newPrefix);

}

}

}

// Function to search for words starting with a certain prefix

void searchByPrefix(struct TrieNode\* root, const char\* prefix) {

// Start at the root node

struct TrieNode\* node = root;

// Traverse the Trie based on each character in the prefix

for (int i = 0; i < strlen(prefix); i++) {

// Calculate the index of the character

int index = tolower(prefix[i]) - 'a';

// If the child node for this character doesn't exist, print a message and return

if (!node->children[index]) {

printf("No slang words found with the given prefix.\n");

return;

}

// Move to the child node

node = node->children[index];

}

// Print all words starting from the current node

printf("Slang words starting with prefix '%s':\n", prefix);

printWords(node, prefix);

}

// Function to display all slang words

void displayAll(struct TrieNode\* root) {

// Print header message

printf("All slang words:\n");

// Initialize an empty prefix

char prefix[100] = ""; // Prefix awal kosong

// Call printWords to print all words starting from the root node

printWords(root, prefix);

}

bool validWord(char word[]){

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

if(word[i] == ' ') return false; // if there is any space in string return false

}

return true;

}

bool validDesc(char desc[]){

int counter = 0; // declare counter to count word

char temp[1001]; // declare char temp

strcpy(temp, desc); // copy desc to temp so string on desc not affected by strtok

// strtok to separate words from sentence

char\* token = strtok(temp, " ");

while(token != NULL){

counter++; // every word found, counter ++

token = strtok(NULL, " ");

}

// printf("%d\n", counter);

if(counter < 2) return false; //if word < 2 return false

return true;

}

// Function to release a new slang word

void releaseSlangWord(struct TrieNode\* root) {

char word[100]; //declaration variable char named "word"

char description[1000]; //declaration variable char named "description"

do{

printf("Enter the new slang word: ");

gets(word);

if(validWord(word) == false) printf("Slang word must be only 1 word and contain no space.\n");

//the user must input only 1 word and there hasn't to be a space or ' ' in it

}while(validWord(word) == false);

do{

printf("Enter the description of the slang word: ");

gets(description); //scanning the user input untill the user press enter on their keyboard

if(validDesc(description) == false) printf("Description must be more than 1 word.\n");

}while(validDesc(description) == false);

insert(root, word, description); //calling the function insert if the condition is met

printf("Slang word released successfully!\n");//succes message

}

//function to search slang word

void searchSlangWord(struct TrieNode\* root) {

char word[100];

printf("Enter the slang word to search: ");

scanf("%s", word);

if (strlen(word) <= 1 || strchr(word, ' ')) {//if the slang word must have more than 1 word and contain a space in it

printf("Slang word must be more than 1 character and contain no space.\n");

return;

}

char\* description = search(root, word);//calling the function search that is written beforehand

if (description) {// Checking if the description outputted a true or false

printf("Slang word: %s\nDescription: %s\n", word, description);//if the output is true, this would print

} else {

printf("Slang word not found.\n");//if it's false then this will bre printed

}

}

// check whether the tries data is empty

bool isTrieEmpty(struct TrieNode\* root) {

for (int i = 0; i < ALPHABET\_SIZE; i++) {

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

return false;

}

}

return true;

}

// Main function

int main() {

struct TrieNode\* root = createNode();

int choice;

do{//the system will print and run these program below if the condition of the "while" is correct

system("cls");//used to clear the console screen of the user

printf("Boogle - Slang Dictionary\n");//the header of the menu

printf("1. Release a new slang word\n");//the menus

printf("2. Search a slang word\n");//the menus

printf("3. View all slang words starting with a certain prefix word\n");//the menus

printf("4. View all slang words\n");//the menus

printf("5. Exit\n");//the menus

printf("Enter your choice: ");//the menus

scanf("%d", &choice);getchar();//the user will input the option they wanted, and the getchar

switch (choice) {

case 1:

{

system("cls");//used to clear the console screen of the user

releaseSlangWord(root);//calling the function in the code

system("pause");//the user must do an input to continue the program, if the condition of the previous function is met

break;//the end of he first menu

}

case 2:

{

system("cls");//used to clear the console screen of the user

if(isTrieEmpty(root)){ //if no data

printf("No data yet\n");

} else {

searchSlangWord(root);//calling the function in the code

}

system("pause");//the user must do an input to continue the program, if the condition of the previous function is met

break;//the end of he second menu

}

case 3:

{

system("cls");//used to clear the console screen of the user

if(isTrieEmpty(root)){ //if no data

printf("No data yet\n");

} else {

printf("Enter the prefix word: ");//the system will print these text out

char prefix[100];//declaration for variable prefix with the limit of 100 char

scanf("%s", prefix);//the user will input the prefix

searchByPrefix(root, prefix);//calling the function in the code

}

system("pause");//the user must do an input to continue the program, if the condition of the previous function is met

break;//the end of he third menu

}

case 4:

{

system("cls");//used to clear the console screen of the user

if(isTrieEmpty(root)){ //if no data

printf("No data yet\n");

} else {

displayAll(root);//calling the function in the code

}

system("pause");//the user must do an input to continue the program, if the condition of the previous function is met

break;//the end of he fourth menu

}

case 5:

{

printf("Exiting Boogle - Slang Dictionary. Goodbye!\n");//the system will print the text

exit(0);//exitting the case loop

}

default://if the inputted number isn't corresponding to the condition, this case will run

printf("Invalid choice. Please choose again.\n");//the system will print these text

}

}while(choice != 5);//the validation of the previous "do" function

return 0;//default escape sequence

}