Universidade Federal do Piauí – UFPI
Campus Senador Helvídio Nunes de Barros – CSHNB
Curso de Sistemas de Informação
Disciplina: Estruturas de Dados II
Professora: Juliana Oliveira de Carvalho

Trabalho Ponto Extra na 2ª Avaliação de ED-II
DATA DA POSTAGEM NO SIGAA: 09/01/2024

Obs.: o trabalho é individual e deve ser feito a mão, tire foto da resposta gere um pdf e post no Sigaa

Questão 01: Explique com exemplo o funcionamento da função insereTrie.

Questão 02: Explique com exemplo o funcionamento da função buscar.

Questão 03: Modifique o código do ANEXO I permitindo que o usuário insira quantas palavras desejar, e busque quantas palavras desejar.

Questão 04: Modifique o código de forma que a função insereTrie devolva 1 se a palavra que se deseja inserir já havia sido inserida anteriormente, e 0 caso contrário. No main imprima uma mensagem se ele retornar 1.

ANEXO I

	Código para Inserir e remover em uma árvore Trie de palavras
1	`#include <stdio.h>`
2	`#include <stdlib.h>`
3	`#include <string.h>`
4	
5	`// Definição da estrutura do nó Trie`
6	`struct TrieNode {`
7	`    struct TrieNode* children[26];  // Para letras do alfabeto inglês`
8	`    int isEndOfWord;  // Indica se o nó representa o final de uma palavra`
9	`};`
10	
11	`// Função para inicializar um novo nó Trie`
12	`struct TrieNode* createNode() {`
13	`    struct TrieNode* node = (struct TrieNode*)malloc(sizeof(struct TrieNode));`
14	`    node->isEndOfWord = 0;`
15	
16	`    for (int i = 0; i < 26; i++) {`
17	`        node->children[i] = NULL;`
18	`    }`
19	
20	`    return node;`
21	`}`
22	
23	
24	
25	
26	
27	
28	

	Código para Inserir e remover em uma árvore Trie de palavras
29	// Função para inserir uma palavra na Trie
30	void insereTrie(struct TrieNode **root, const char *word, int i, int tam) {
31	
32	int index;
33	index = word[i] - 'a';
34	
35	if(i < tam -1) {
36	
37	if ((*root)->children[index] == NULL) {
38	(*root)->children[index] = createNode();
39	}
40	
41	insereTrie(&((*root)->children[index]), word, ++i, tam);
42	}
43	else {
44	if ((*root)->children[index] == NULL)
45	(*root)->children[index] = createNode();
46	
47	((*root)->children[index])->isEndOfWord = 1;
48	}
49	
50	}
51	
52	// Função para buscar uma palavra na Trie
53	int buscar(struct TrieNode* root, const char* word)
54	{ struct TrieNode* currentNode = root;
55	int achou = 0, tam, i, index;
56	
57	tam = strlen(word);
58	
59	for (i = 0; (i < tam && !achou); i++) {

	Código para Inserir e remover em uma árvore Trie de palavras
60	index = word[i] - 'a';
61	
62	if (currentNode->children[index] != NULL){
63	if ((currentNode->children[index])->isEndOfWord == 1 && i == tam - 1)
64	achou = 1;
65	else currentNode = currentNode->children[index];
66	}
67	
68	}
69	
70	
71	return achou;
72	}
73	
74	
75	// Função principal para teste
76	int main() {
77	struct TrieNode* root;
78	int achou = 0;
79	root = createNode();
80	// Inserindo algumas palavras na Trie
81	insereTrie(&root, "apple",0,strlen("apple"));
82	insereTrie(&root, "banana",0,strlen(" banana"));
83	insereTrie(&root, "app",0, strlen("app"));
84	
85	printf("Insercoes concluidas.\n");
86	
87	achou = buscar(root, "banana");
88	if (achou)

	Código para Inserir e remover em uma árvore Trie de palavras
89	printf("Palavra Cadastrada \n");
90	else printf("Palavra NAO Cadastrada \n");
91	
92	getchar();
93	return 0;
94	}