TP3 - AEDS II

Nome: Diego Resende Braz **Matrícula: 212050084**

1 - Introdução

O Código Morse é um sistema de comunicação inventado por Samuel Morse e Alfred Vail na década de 1830. É um sistema capaz de codificar letras, números e sinais de pontuação usando uma combinação de pontos e traços.

O sistema foi desenvolvido para ser utilizado com o telégrafo elétrico, que permitia a transmissão de mensagens à distância através de fios. Esse método, teve um papel importante na história das comunicações, especialmente em situações em que a fala não era viável ou prática. Entretanto, com o passar do tempo o uso do código morse foi substituído por tecnologias mais avançadas, como o telefone.

No Código Morse, cada letra do alfabeto, números e alguns sinais de pontuação são representados por sequências específicas de pontos e traços.

Segue a tabela de representações do Código Morse:

A •–	J •–––	S •••	1 •———–
B –•••	K –•–	T –	2 ••–––
C –•–•	L •–••	U ••–	3 •••––
D –••	M ––	V •••–	4 ••••–
E •	N –•	W •––	5 •••••
F ••–•	O –––	X –••–	6 –••••
G ––•	P •––•	Y –•––	7 ––•••
H ••••	Q ––•–	Z ––••	8 –––••
I ••	R •–•	0 –––––	9 ––––•

Imagem 1: Código Morse

2 - Problema Proposto

A ideia deste trabalho é desenvolver um programa que consiga transcrever mensagens do Código Morse para mensagens de textos, e vice-versa. Para realizar a transcrição das mensagens em código morse, foi utilizado o conceito de árvore digital, organizada de forma, em que a medida que os pontos e traços são lidos, haja a “descida” na árvore até encontrar o símbolo correspondente do alfabeto. A árvore fica organizada da seguinte forma:

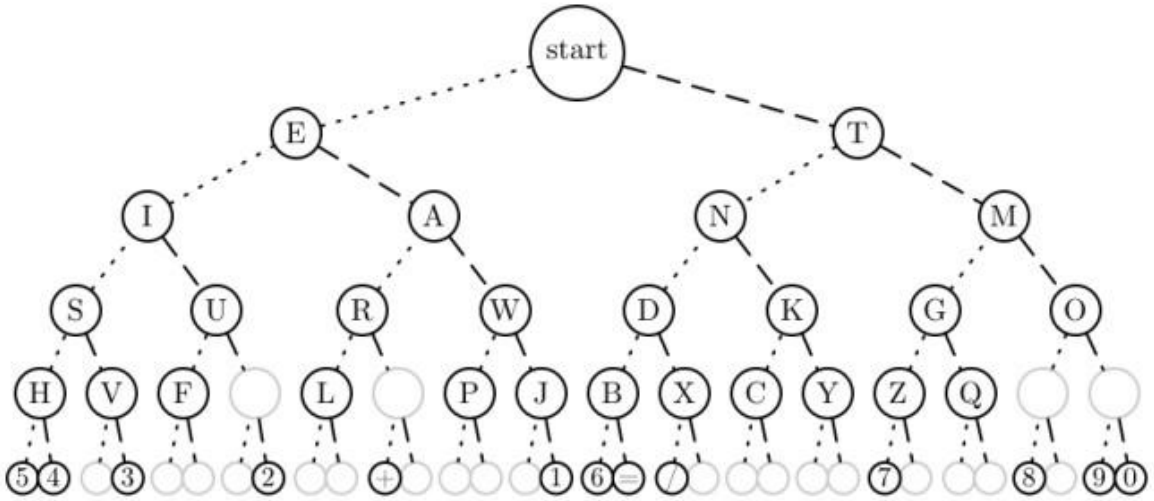

Imagem 2: Árvore Digital

No programa, realizei também a análise de tempo de execução, para poder fazer uma análise mais aprofundada dos casos. Foi executada a transcrição de 5 frases diferentes para tal análise ser realizada.

3 - Modelagem

Para realizar este problema, foi utilizado a linguagem C no Visual Studio Code, através da arquitetura Windows 10.

4 - Funções

Nessa sessão, será explicada a funcionalidade de algumas funções utilizadas no decorrer do algoritmo.

4.1 - typedef struct Node {

```
char symbol;          //armazenar os caracteres do alfabeto
struct Node* dot;   //conexão com o próximo nó quando há pontos encontrados
struct Node* dash;   //conexão com o próximo nó quando há traços encontrados
} Node;
```

Essa função tem como objetivo criar uma árvore digital responsável pela codificação do Código Morse.

4.2 - const char* searchMorse(Node* raiz, char caracteres) {

```
if (raiz == NULL) {     //verifica se a raiz árvore é nula
return NULL;
}
 if (raiz->caracteres == caracteres) {    //verifica se o caractere da raiz é igual ao
                                                procurado
 return "";    //string vazia representando que foi encontrado
```

```
}
const char* esq = searchMorse(raiz->ponto, caracteres); //se não foi encontrado retoma
                                                                    a  busca
if (esq != NULL) {
char* result = (char*)malloc(strlen(esq) + 2);  //aloca memória para armazenar o
                                                        caminho encontrado no nó filho
strcpy(result, ".");
strcat(result, esq);
return result;   //caractere encontrado
}
const char* dir = searchMorse(raiz->traco, caracteres);     //se não foi encontrado do
                                                    lado esquerdo, inicia a busca no lado direito
if (dir != NULL) {
char* result = (char*)malloc(strlen(dir) + 2);    //aloca memória para armazenar o
                                                        caminho encontrado no nó filho
strcpy(result, "-");
strcat(result, dir);
return result;    //caractere encontrado
}
return NULL;   //se não encontrar o caractere em nenhum dos dois lados
}
```

Essa função é utilizada para pesquisar o código morse e seu respectivo caractere na árvore criada.

```
4.3 - Node* createNode(char caracteres) {
Node* newNode = (Node*)malloc(sizeof(Node));   //armazenar memória para o novo nó
newNode->caracteres = caracteres;
newNode->ponto = NULL;     //o valor NULL é atribuído para representar que não há conexões
                                inicialmente com outros nós
newNode->traco = NULL;
return newNode;
}
```

Essa função é responsável por criar um novo nó na árvore e atribuir o caractere desse novo nó. Ela permite criar conexões para os nós filhos, que serão definidos posteriormente, formando a estrutura da árvore.

```
4.4 - void insertMorse(Node* raiz, char* morse, char caracteres) {
int len = strlen(morse);    //tamanho da sequência de pontos e traços
Node* currentNode = raiz;   //percorrer a árvore na inserção de um caractere
for (int i = 0; i < len; i++) {   //percorrer cada traço e ponto da sequência Morse
if (morse[i] == '.') {
if (currentNode->ponto == NULL)    //verifica se o nó filho é nulo
currentNode->ponto = createNode('\0');    //se for nulo, cria um novo nó
currentNode = currentNode->ponto;     //apontar que o nó foi criado
} else if (morse[i] == '-') {
if (currentNode->traco == NULL)  //verifica se o nó filho é nulo
currentNode->traco = createNode('\0'); //se for nulo cria um novo nó
```

```
currentNode = currentNode->traco; //apontar que o nó foi criado
}
}
currentNode->caracteres = caracteres; //aponta para o último nó da sequência. É oo
                                        caractere que está sendo inserido
}
```

Essa função é responsável por inserir um caractere na árvore de codificação do Código Morse. A estrutura da árvore é atualizada de acordo com a inserção de novos caracteres, criando novos nós filhos necessários para representar o Código Morse.

4.5 - void printTree(Node* raiz, char* codigoMorse, int profundidade) {

```
if (raiz == NULL)   //verifica se a raiz é nula
return;  //se for nula, interrompe a função (árvore vazia)
if (raiz->caracteres != '\0') {    //verifica se é um caractere válido da árvore
codigoMorse[profundidade] = '\0';
printf("%c: %s\n", raiz->caracteres, codigoMorse);   //printa o caractere e seu Código Morse
}
codigoMorse[profundidade] = '.';  //o nó atual representa um ponto no Morse
printTree(raiz->ponto, codigoMorse, profundidade + 1);
codigoMorse[profundidade] = '-';   //o nó atual representa um traço no Morse
printTree(raiz->traco, codigoMorse, profundidade + 1);
codigoMorse[profundidade] = '\0';  //atualizar corretamente a representação do Morse
}
```

Essa função é responsável por percorrer toda a árvore e imprimir todos os caracteres válidos juntamente com a sua representação em Código Morse. A medida da profundidade é utilizada para rastrear a posição atual na sequência de pontos e traços, permitindo a representação correta do Código Morse para cada caractere.

4.6 - void convertToText(Node* raiz, char* morse) {

```
char* token = strtok(morse, " "); //trata a sequência morse individualmente, delimitadas por
                                    espaços
while (token != NULL) {
if (strcmp(token, "/") == 0) {  //verifica se é um espaço entre palavras
printf(" ");
} else {
int len = strlen(token);  //calcula o comprimento da sequência
Node* currentNode = raiz;  //representa um caractere
for (int i = 0; i < len; i++) {  //percorrer cada ponto e traço da sequência em loop
if (token[i] == '.') {
if (currentNode->ponto == NULL) {   //verifica se o nó filho correspondente existe na árvore
printf("Caractere inválido encontrado: %s\n", token);  //se  não existir, informa que o caractere
                                                          é inválido e retorna
return;
}
currentNode = currentNode->ponto;
} else if (token[i] == '-') {
```

```
 if (currentNode->traco == NULL) {
printf("Caractere inválido encontrado: %s\n", token);
return;
}
currentNode = currentNode->traco;
}
}
printf("%c", currentNode->caracteres);   //após percorrer todos os elementos da sequência, a função imprime o caractere correspondente
}
token = strtok(NULL, " ");
}
}
```

Essa função é responsável por converter uma sequência de Código Morse para mensagem de texto usando a árvore criada. Ela verifica a validade dos caracteres Morse e lida com espaços entre palavras, inserindo o / como separador, imprimindo o texto convertido.

4.7 - void convertTextToMorse(Node* raiz, const char* mensagem) {

```
if (mensagem == NULL || *mensagem == '\0') {  //verificar se a mensagem é nula ou vazia
return;     //retorna se não houver nada a ser convertido
 }
size_t mensagemComprimento = strlen(mensagem);   //calcular o comprimento da mensagem
for (size_t i = 0; i < mensagemComprimento; i++) {
char c = toupper((unsigned char)mensagem[i]);    //altera todos as letras para maiúsculas, para
                                                 ser feita a pesquisa corretamente na árvore
const char* morse = searchMorse(raiz, c);   //buscar a representação em Morse na árvore
if (morse != NULL) {   //se a representação em morse existir
printf("%s ", morse);  //imprime a sequência em morse
} else {
printf("/ ", c);  //imprime o barra para separar as representações
}
}
}
```

Essa função é responsável por converter uma mensagem de texto para sua representação em Morse usando a árvore criada. Ela lida com caracteres e imprime o texto convertido para morse.

5 - Análise de Complexidade

Nessa sessão, será analisada a complexidade de algumas funções utilizadas no decorrer do algoritmo.

5.1 - const char* searchMorse(Node* raiz, char caracteres)

A complexidade irá depender da quantidade de nós existentes na árvore. No pior caso, em que o elemento não é encontrado na árvore, a função percorrerá todos os nós uma vez, fazendo chamada recursiva para cada um dos filhos, portanto a complexidade será O(n), onde n é a quantidade de nós da árvore.

5.2 - Node* createNode(char caracteres)

A complexidade dessa função é constante, independentemente do tamanho da árvore. Sua complexidade é O(1).

5.3 - void insertMorse(Node* raiz, char* morse, char caracteres)

A complexidade dessa função depende do comprimento do código morse informado, comprimento calculado e representado pela variável 'len'. Portanto, a complexidade da função é O(len).

5.4 - void printTree(Node* raiz, char* codigoMorse, int profundidade)

A complexidade dessa função irá depender da quantidade existente de nós na árvore. Portanto, a complexidade da função é O(n), onde n é a quantidade de nós da árvore.

5.5 - void convertToText(Node* raiz, char* morse)

A complexidade dessa função depende do número de sequências morse separadas por espaço (tokens), e também do tamanho médio de cada token. Portanto, a complexidade da função é O(T * tamanhoT), onde 'T' é o número de tokens na sequência morse e 'tamanhoT' é o tamanho médio de cada token.

5.6 - void convertTextToMorse(Node* raiz, const char* mensagem)

A complexidade dessa função depende do comprimento da mensagem de texto fornecida, há também a necessidade de realizar uma busca ao caractere na árvore, usando a função 'searchMorse', na qual a complexidade é O(n). Portanto, a complexidade da função 'convertTextToMorse' é O(T * n).

6 - Resultados

Os testes foram realizados em meu notebook pessoal:

- 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz
- 8gb de memória RAM
- 256gb de armazenamento interno - SSD

Os resultados obtidos estão detalhados através de prints do terminal de saída do Visual Studio Code. Assim como, o documento de entrada utilizado estará descrito abaixo.

codifica.in

```
1  Trabalho pratico 3 de algoritmos e estruturas de dados II
2  Diego Resende Braz
3  Rafael Sachetto Oliveira
4  Ciencia da computacao
5  Sao Joao del Rei
```

Imagem 3: Entrada de texto para realizar conversão para morse.

Imagem 4: Entrada em Morse para realizar conversão para mensagem de texto.

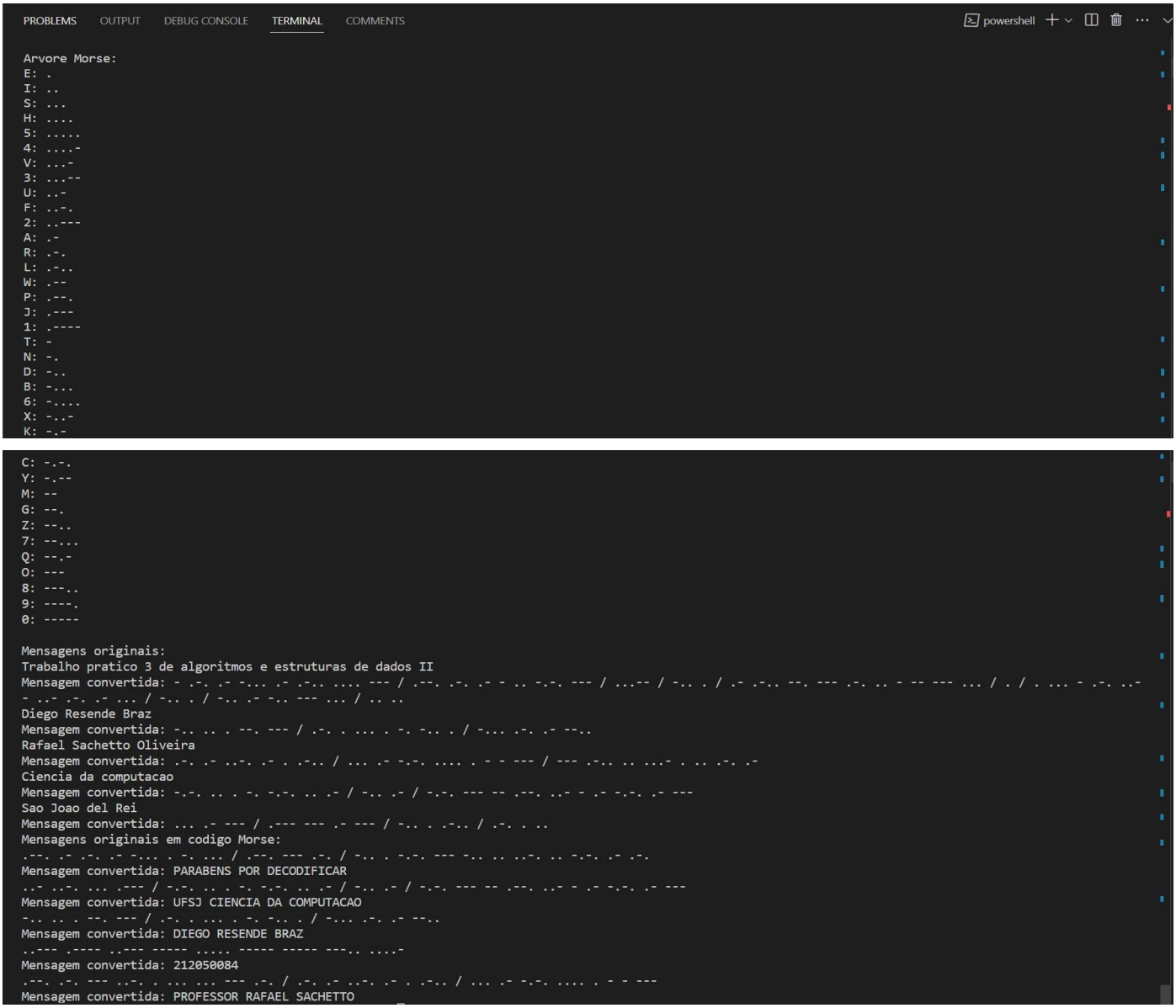

Imagem 5: Impressão da Árvore criada e das conversões pedidas.

Os resultados foram obtidos com sucesso. A impressão da árvore foi feita de maneira correta, com o código morse e seu símbolo correspondente. As conversões foram feitas de maneira clara, assim como era esperado.

7 - Conclusão

O trabalho ajudou a praticar o conteúdo ensinado na disciplina, sobre pesquisa em memória principal, aplicações de árvores, bem como aplicá-lo em um cenário real, conceito no qual é bastante utilizado em diversas áreas atualmente.

Tive dificuldades na transcrição de Código Morse para mensagem de texto, onde após alguns ajustes obtive êxito. Ademais, consegui implementar todas as outras funções, consultando os slides disponibilizados pelo Portal Didático, assim como implementando os conhecimentos adquiridos durante as aulas.

8 - Referências Bibliográficas:

- Slides disponibilizados na plataforma Campus Virtual (https://campusvirtual.ufsj.edu.br/portal/2023_1n/course/view.php?id=1184)

- Aulas ministradas pelo professor Rafael Sachetto Oliveira.