UNIVERSIDADE FEDERAL DE SÃO JOÃO DELREI

TP 3 - AEDS II

Análise dos Algorítimos de Ordenação

Alunos: Davi Medeiros Gonçalves (212050085)
Rafael Carvalho Avidago Geraldo (212050057)

Professor: Rafael Sachetto Oliveira

Introdução

O terceiro trabalho prático de AEDS II, consistiu em utilizar árvores binárias na construção de um código capaz de traduzir:

Texto para código Morse
Morse para texto

De forma que conforme fossemos descendo pelas raízes da árvore os resultados fossem sendo encontrados pelo código assim como seus respectivos símbolos.

Implementação

A seguir vamos explicar e resumir as funções e o funcionamento geral do código

createNode e createNode2:

```c
struct TreeNode* createNode(char data) {
    struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

TreeNode2* createNode2(char value) {
    TreeNode2* newNode = (TreeNode2*)malloc(sizeof(TreeNode2));
    newNode->value = value;
    newNode->ponto = NULL;
    newNode->barra = NULL;
    return newNode;
}
```

A função **createNode** cria nós em uma árvore binária de pesquisa simples. Cada nó contém um campo data para armazenar um valor do tipo char, bem como ponteiros para os nós filhos esquerdo e direito. O nó é alocado dinamicamente na memória usando malloc , em seguida, os campos são inicializados e o nó é retornado.

Já a função **createNode2** cria nós em uma árvore que contém campos adicionais. Além do campo value para armazenar um valor do tipo char, cada nó possui ponteiros adicionais ponto e barra, que podem apontar para outros nós na árvore. Assim como na função anterior, o nó é alocado dinamicamente na memória, os campos são inicializados e o nó é retornado.

Ambas as funções são utilizadas para criar nós em diferentes tipos de árvores, com campos estruturas de dados específicos para cada caso.

```c
void insertNode(struct TreeNode** root, char data, const char* code) {
    if (*root == NULL) {
        *root = createNode('\0');
    }
    struct TreeNode* currentNode = *root;
    int i = 0;
    while (code[i] != '\0') {
        if (code[i] == '.') {
            if (currentNode->left == NULL) {
                currentNode->left = createNode('\0');
            }
            currentNode = currentNode->left;
        } else if (code[i] == '-') {
            if (currentNode->right == NULL) {
                currentNode->right = createNode('\0');
            }
            currentNode = currentNode->right;
        }
        i++;
    }
    currentNode->data = data;
}
```

A função **insertNode** recebe um ponteiro para ponteiro para o nó raiz de uma árvore binária de pesquisa, um caractere para ser armazenado no novo nó e um código que determina a posição do novo nó na árvore.

O código verifica se a árvore está vazia e, se estiver, cria um nó com valor vazio e atribui-o como o nó raiz. Em seguida, percorre o código caracter por caracter e realiza as seguintes ações: se encontrar um ponto, cria um nó filho esquerdo se ele não existir, e atualiza o nó atual para ser o novo nó filho esquerdo; se encontrar um hífen, cria um novo nó filho direito se ele não existir, e atualiza o nó atual para ser o novo nó filho direito. No final, atribui o caractere fornecido ao nó atual.

Em resumo, a função insere um novo nó na árvore binária de pesquisa com base em um código, criando novos nós e percorrendo a árvore conforme necessário.

```c
void freeTree(TreeNode2* root2) {
    if (root2 == NULL)
        return;
    freeTree(root2->ponto);
    freeTree(root2->barra);
    free(root2);
}
```

A função **freeTree** é responsável por liberar a memória alocada para todos os nós de uma árvore representada pela estrutura TreeNode2.

A função segue uma abordagem de recursão para percorrer a árvore. Inicialmente, verifica se o ponteiro para o nó atual é nulo. Se for, não há mais nós para liberar, então a função retorna.

Caso contrário, a função chama recursivamente a si mesma para liberar os nós filhos esquerdo e direito do nó atual, passando os ponteiros root2->ponto e root2->barra, respectivamente. Isso garante que todos os nós da árvore sejam percorridos e liberados.

Após liberar os nós filhos, a função usa free(root2) para liberar a memória do nó atual. Isso é feito de forma recursiva para percorrer toda a árvore e liberar a memória de todos os nós.

```c
void translateToText(struct TreeNode* root, const char* morseCode) {
    int i = 0;
    while (morseCode[i] != '\0') {
        if (morseCode[i] == '/') {
            printf(" ");
        } else {
            struct TreeNode* temp = root;
            while (morseCode[i] != ' ' && morseCode[i] != '\0') {
                if (morseCode[i] == '.') {
                    temp = temp->left;
                } else if (morseCode[i] == '-') {
                    temp = temp->right;
                }
                i++;
            }
            if (temp == NULL) {
                printf("Invalid Morse code: %s\n", morseCode);
                return;
            }
            printf("%c", temp->data);
        }
        i++;
    }
}
```

A função **translateToText** é responsável por traduzir um código Morse em uma sequência de caracteres legíveis. Ela percorre o código Morse fornecido e utiliza uma árvore binária TreeNode para mapear os pontos (.) e hífens (-) para caracteres correspondentes. O código Morse é percorrido caracter por caracter, e dependendo do caractere encontrado, o ponteiro temp é atualizado para apontar para o nó filho esquerdo ou direito da árvore. Ao encontrar um espaço entre palavras, é impresso um espaço em branco. Se o código Morse não puder ser mapeado para um caractere válido na árvore, uma mensagem de erro é exibida. Caso contrário, o caractere correspondente ao código Morse é impresso. Em resumo, a função translateToText realiza a tradução de um código Morse para uma sequência de caracteres legíveis, usando uma árvore binária como referência.

```c
TreeNode2* traverseTree(TreeNode2* root2, char target, char* path, int level) {
    if (root2 == NULL)
        return NULL;

    if (root2->value == target) {
        path[level] = '\0';
        return root2;
    }

    path[level] = '.';
    TreeNode2* node = traverseTree(root2->ponto, target, path, level + 1);
    if (node != NULL)
        return node;

    path[level] = '-';
    node = traverseTree(root2->barra, target, path, level + 1);
    if (node != NULL)
        return node;

    path[level] = '\0';
    return NULL;
}
```

Em resumo, a função **traverseTree** percorre recursivamente uma árvore TreeNode2 em busca de um nó com um valor alvo específico, registrando o caminho percorrido. Ela retorna o ponteiro para o nó encontrado ou nulo se o valor alvo não for encontrado na árvore.

```c
void getMorseCode(char character, char* morseCode) {
    TreeNode2* root2 = Constroi2();
    char path[10];
    TreeNode2* node = traverseTree(root2, character, path, 0);
    if (node != NULL)
        strcpy(morseCode, path);
    else
        morseCode[0] = '\0';
    freeTree(root2);
}
```

A função **getMorseCode** é responsável por obter o código Morse correspondente a um determinado caractere. Ela utiliza a função **traverseTree** para percorrer uma árvore TreeNode2 em busca do nó que contém o caractere desejado.

```c
char* readTextFile(const char* filename) {
    FILE* file = fopen(filename, "r");
    if (file == NULL) {
        printf("Erro ao abrir o arquivo.\n");
        return NULL;
    }

    fseek(file, 0, SEEK_END);  // Move o indicador de posição para o final do arquivo
    long fileSize = ftell(file);  // Obtém o tamanho do arquivo
    fseek(file, 0, SEEK_SET);  // Move o indicador de posição de volta para o início do arquivo

    char* content = (char*)malloc((fileSize + 1) * sizeof(char));  // Aloca memória para o conteúdo do arquivo
    fread(content, sizeof(char), fileSize, file);  // Lê o conteúdo completo do arquivo
    content[fileSize] = '\0';  // Adiciona o caractere nulo ao final do conteúdo

    fclose(file);

    return content;
}
```

Em resumo, a função readTextFile abre um arquivo de texto, lê seu conteúdo e o armazena em uma string dinamicamente alocada. O conteúdo é então retornado como uma string, e o arquivo é fechado.

Função Main:

O programa começa exibindo um menu com duas opções para o usuário selecionar: traduzir de Morse para Texto ou de Texto para Morse. O usuário insere sua escolha através do scanf.

Se a opção selecionada for 1, o programa cria uma árvore TreeNode e insere os nós correspondentes aos caracteres Morse. Em seguida, solicita ao usuário que digite o código Morse que deseja traduzir. O código Morse digitado é lido usando fgets, e é removido o caractere de nova linha (\n) usando strcspn.

A função translateToText é chamada, passando a árvore root e o código Morse digitado como parâmetros. A função percorre a árvore de acordo com o código Morse e imprime o texto traduzido.

Se a opção selecionada for 2, o programa chama a função readTextFile para ler o conteúdo do arquivo "msg.txt" e armazená-lo em uma string character. Em seguida, itera sobre cada caractere de character.

Se o caractere atual não for um espaço em branco, a função getMorseCode é chamada para obter o código Morse correspondente ao caractere. O código Morse é armazenado em morseCode.

Se houver um espaço em branco anterior (previousSpace), o programa imprime "/ " para indicar uma separação entre palavras. Em seguida, imprime o código Morse do caractere atual seguido de um espaço.

Se o caractere atual for um espaço em branco, o programa marca previousSpace como 1 para indicar que há um espaço em branco anterior.

Se o caractere atual for uma nova linha (\n), o programa imprime duas novas linhas para separar as frases traduzidas.

Após iterar por todos os caracteres de character, a memória alocada para character é liberada usando free.

Se a opção selecionada não for nem 1 nem 2, o programa imprime "Opcao invalida."

O programa retorna 0, indicando que foi executado sem erros.

Em resumo, o código permite ao usuário escolher entre traduzir de Morse para Texto ou de Texto para Morse. Ele usa uma árvore para armazenar os caracteres Morse e suas traduções correspondentes. Dependendo da opção selecionada, o programa solicita entrada do usuário ou lê o conteúdo de um arquivo de texto. Em seguida, realiza a tradução desejada e imprime o resultado.

Resultados e discussões

O programa funciona como esperado e apresenta os resultados como previstos, traduzindo corretamente.
Quanto as dificuldades encontradas achamos importante ressaltar que criar a árvore de uma forma que ela reconheça pontos e traços, levando aos símbolos corretos.

Conclusão

A implementação do código discorre bem com os assuntos vistos e discutidos em sala de aula, trazendo uma situação interessante e uma utilização prática da árvore binária.