Trabalho de Estrutura de Dados II

Integrantes

- **Thaís Faustino Calixto**
  - Número de matrícula: **2019111746**
- **Heitor Galdino Borges Costa**
  - Número de matrícula: **2019110327**

Objetivos

- **Realizar um programa de árvores Binárias em C.**

Implementações

Estrutura de árvore binária

```c
typedef struct Node
{
    int value;
    struct Node *left;
    struct Node *right;
} Node;
```

Criação de árvore binária

```c
void startTree(Node **node)
{
    *node = NULL;
}

void addValue(int value, Node** node)
{
    if (*node == NULL)
        *node = newNode(value);
    else
    {
        if (value > (*node)->value)
            addValue(value, &(*node)->right);
        else
            addValue(value, &(*node)->left);
    }
}

Node* newNode(int value)
{
    Node* output = (Node*) malloc(sizeof(Node));
    output->value = value;
    output->left = NULL;
```

```
    output->right = NULL;
    return output;
}
```

Recepção de árvores já existentes

```
Node* createTree()
{
    Node* tree;
    startTree(&tree);

    printf("\nPrimeiro crie o nó inicial:");

    createHelper(&tree, 1);

    return tree;
}

void createHelper(Node **node, int doNewNode)
{
    int value, option;
    if (doNewNode != 1)
        return;

    printf("\nPor favor, insira o valor do nó: ");
    scanf("%d", &value);
    *node = newNode(value);

    printf("\nVocê deseja criar um filho à esquerda do nó atual?\n");
    printf("(Valor do nó atual: %d)\n1 - sim || 0 - não\n", (*node)->value);
    scanf("%d", &option);
    createHelper(&(*node)->left, option);

    printf("\nVocê deseja criar um filho à direita do nó atual?\n");
    printf("(Valor do nó atual: %d)\n1 - sim || 0 - não\n", (*node)->value);
    scanf("%d", &option);
    createHelper(&(*node)->right, option);
}

Node* newNode(int value)
{
    Node* output = (Node*) malloc(sizeof(Node));
    output->value = value;
    output->left = NULL;
    output->right = NULL;
    return output;
}
```

Função profundidade que lê a árvore e retorna sua profundidade

```c
int depth(Node *node)
{
    if (node == NULL)
        return -1;
    int leftDepth = depth(node->left);
    int rightDepth = depth(node->right);

    if (leftDepth > rightDepth)
        return leftDepth + 1;
    else
        return rightDepth + 1;
}
```

Funções de percurso "*in-order*", "*pre-order*" e "*pos-order*" para a árvore

```c
void inOrder(Node *node)
{
    if (node != NULL)
    {
        inOrder(node->left);
        printf("%d ", node->value);
        inOrder(node->right);
    }
}

void preOrder(Node *node)
{
    if (node != NULL)
    {
        printf("%d ", node->value);
        preOrder(node->left);
        preOrder(node->right);
    }
}

void posOrder(Node *node)
{
    if (node != NULL)
    {
        posOrder(node->left);
        posOrder(node->right);
        printf("%d ", node->value);
    }
}
```

Função caminho, que lê a árvore e exibe uma lista com todos os caminhos até as folhas das árvores

```c
void pathFinder(Node *node, int path[], int depth)
{
    if (node == NULL)
        return;
```

```
    path[depth] = node->value;
    depth++;

    if (node->left == NULL & node->right == NULL)
        pathPrinter(path, depth);
    else
    {
        pathFinder(node->left, path, depth);
        pathFinder(node->right, path, depth);
    }
}

void pathPrinter(int path[], int depth)
{
    int currentNode = 0;
    for (; currentNode < depth - 1; currentNode++)
        printf("%d --> ", path[currentNode]);
    printf("%d\n", path[currentNode]);
}
```

Função para testar se dada árvore é uma "*árvore binária de procura*"

```
int isBinarySearchTree(Node *node, int min, int max)
{
    if (node == NULL)
        return 1;
    if (node->value <= min || node->value > max)
        return 0;
    return isBinarySearchTree(node->left, min, node->value) &&
isBinarySearchTree(node->right, node->value, max);
}
```

Função para inserir um elemento em uma árvore binária de procura

```
void addValue(int value, Node** node)
{
    if (*node == NULL)
        *node = newNode(value);
    else
    {
        if (value > (*node)->value)
            addValue(value, &(*node)->right);
        else
            addValue(value, &(*node)->left);
    }
}

Node* newNode(int value)
{
    Node* output = (Node*) malloc(sizeof(Node));
    output->value = value;
    output->left = NULL;
    output->right = NULL;
```

```
    return output;
}
```

Código completo

```
// Libs needed
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <string.h>

// Struct needed
typedef struct Node
{
    int value;
    struct Node *left;
    struct Node *right;
} Node;

// Functions needed
void startTree(Node **node);
void addValue(int value, Node** node);
Node* createTree();
void createHelper(Node **node, int doNewNode);
int isBinarySearchTree(Node *node, int min, int max);
int searchValue(int value, Node* node);
int depth(Node *node);
void inOrder(Node *node);
void preOrder(Node *node);
void posOrder(Node *node);
void pathFinder(Node *node, int path[], int depth);
void pathPrinter(int path[], int depth);
Node* newNode(int value);

// Main function
int main()
{
    Node* tree;
    int creationChoice, menuOption, keepAdding;

    printf("Bem vindo ao programa de árvores binárias.\n\n");
    printf("1 - Criar árvore de busca do zero.\n");
    printf("2 - Criar árvore manualmente\n");
    printf("3 - Carregar árvore de exemplo\n");
    printf("Por favor, selecione uma das opções acima: ");
    scanf("%d", &creationChoice);

    switch (creationChoice)
    {
        case 1:
            startTree(&tree);
            do
            {
                int value;

                printf("\nPor favor, insira o valor a ser adicionado: ");
```

```
                scanf("%d", &value);
                addValue(value, &tree);

                printf("\nVocê deseja inserir outro valor?\n");
                printf("Digite 1 para Sim ou 0 para Não: ");
                scanf("%d", &keepAdding);
            } while (keepAdding == 1);
            break;
        case 2:
            tree = createTree();
            break;
        default:
            startTree(&tree);
            addValue(8, &tree);
            addValue(6, &tree);
            addValue(10, &tree);
            addValue(5, &tree);
            printf("\nA árvore de exemplo a seguir foi carregada.\n");
            printf("\t    8\n");
            printf("\t   / \\\n");
            printf("\t  6   10\n");
            printf("\t /\n");
            printf("\t5\n");
            break;
    }

    do
    {
        int path[999];

        printf("\n[MENU DE AÇÕES]\n");
        printf("1 - Exibir a profundidade da árvore\n");
        printf("2 - Exibir o percurso \"in-order\"\n");
        printf("3 - Exibir o percurso \"pre-order\"\n");
        printf("4 - Exibir o percurso \"pos-order\"\n");
        printf("5 - Exibir o caminho até todas as folhas\n");
        printf("6 - Testar árvore\n");
        printf("7 - Inserir elemento\n");
        printf("0 - Encerrar o programa\n");
        printf("Por favor, digite uma das opções acima: ");
        scanf("%d", &menuOption);
        printf("\n");

        switch (menuOption)
        {
            case 0:
                printf("O programa será encerrado.\n");
                break;
            case 1:
                printf("A árvore atualmente tem profundidade %d\n",
depth(tree));
                break;
            case 2:
                printf("Percurso \"in-order\":\n");
                printf("\t");
                inOrder(tree);
                printf("\n");
                break;
```

```
            case 3:
                printf("Percurso \"pre-order\":\n");
                printf("\t");
                preOrder(tree);
                printf("\n");
                break;
            case 4:
                printf("Percurso \"pos-order\":\n");
                printf("\t");
                posOrder(tree);
                printf("\n");
                break;
            case 5:
                printf("Estes são os caminhos existentes:\n");
                pathFinder(tree, path, 0);
                break;
            case 6:
                printf(isBinarySearchTree(tree, INT_MIN, INT_MAX) ? "A árvore
atual é uma árvore de busca.\n" : "A árvore atual não é uma árvore de
busca.\n");
                break;
            case 7:
                if (isBinarySearchTree(tree, INT_MIN, INT_MAX))
                {
                    int value;

                    printf("Qual o valor a ser adicionado?\n");
                    scanf("%d", &value);
                    addValue(value, &tree);
                }
                else
                    printf("A árvore atual não é de busca, por isso é impossível
inserir.\n");
                break;
            default:
                printf("Não foi inserida uma opção válida.\n");
                break;
        }
    } while (menuOption != 0);

    return 0;
}

// Functions
void startTree(Node **node)
{
    *node = NULL;
}

void addValue(int value, Node **node)
{
    if (*node == NULL)
        *node = newNode(value);
    else
    {
        if (value > (*node)->value)
            addValue(value, &(*node)->right);
        else
```

```
            addValue(value, &(*node)->left);
    }
}

Node* createTree()
{
    Node* tree;
    startTree(&tree);

    printf("\nPrimeiro crie o nó inicial:");

    createHelper(&tree, 1);

    return tree;
}

void createHelper(Node **node, int doNewNode)
{
    int value, option;
    if (doNewNode != 1)
        return;

    printf("\nPor favor, insira o valor do nó: ");
    scanf("%d", &value);
    *node = newNode(value);

    printf("\nVocê deseja criar um filho à esquerda do nó atual?\n");
    printf("(Valor do nó atual: %d)\n1 - sim || 0 - não\n", (*node)->value);
    scanf("%d", &option);
    createHelper(&(*node)->left, option);

    printf("\nVocê deseja criar um filho à direita do nó atual?\n");
    printf("(Valor do nó atual: %d)\n1 - sim || 0 - não\n", (*node)->value);
    scanf("%d", &option);
    createHelper(&(*node)->right, option);
}

int isBinarySearchTree(Node *node, int min, int max)
{
    if (node == NULL)
        return 1;
    if (node->value <= min || node->value > max)
        return 0;
    return isBinarySearchTree(node->left, min, node->value) &&
isBinarySearchTree(node->right, node->value, max);
}

int searchValue(int search, Node *node)
{
    if (node == NULL)
        return 0;
    else
    {
        if (node->value == search)
            return 1;
        if (node->value > search)
            return searchValue(search, node->left);
        return searchValue(search, node->right);
```

```
    }
}

int depth(Node *node)
{
    if (node == NULL)
        return -1;
    int leftDepth = depth(node->left);
    int rightDepth = depth(node->right);

    if (leftDepth > rightDepth)
        return leftDepth + 1;
    else
        return rightDepth + 1;
}

void inOrder(Node *node)
{
    if (node != NULL)
    {
        inOrder(node->left);
        printf("%d ", node->value);
        inOrder(node->right);
    }
}

void preOrder(Node *node)
{
    if (node != NULL)
    {
        printf("%d ", node->value);
        preOrder(node->left);
        preOrder(node->right);
    }
}

void posOrder(Node *node)
{
    if (node != NULL)
    {
        posOrder(node->left);
        posOrder(node->right);
        printf("%d ", node->value);
    }
}

void pathFinder(Node *node, int path[], int depth)
{
    if (node == NULL)
        return;

    path[depth] = node->value;
    depth++;

    if (node->left == NULL & node->right == NULL)
        pathPrinter(path, depth);
    else
    {
```

```
        pathFinder(node->left, path, depth);
        pathFinder(node->right, path, depth);
    }
}

void pathPrinter(int path[], int depth)
{
    int currentNode = 0;
    for (; currentNode < depth - 1; currentNode++)
        printf("%d --> ", path[currentNode]);
    printf("%d\n", path[currentNode]);
}

Node* newNode(int value)
{
    Node* output = (Node*) malloc(sizeof(Node));
    output->value = value;
    output->left = NULL;
    output->right = NULL;
    return output;
}
```