



UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO

COMPONENTE CURRICULAR: LABORATÓRIO DE ALGORITMOS E ESTRUTURA DE  
DADOS II

DOCENTE: KENNEDY REURISON LOPES

DISCENTE(S): ISABEL DE PAIVA FREIRE - 2024010417

ATIVIDADE SOBRE ÁRVORES BINÁRIAS

PAU DOS FERROS

MAIO DE 2025

## 1. ESTRUTURA DE DADOS

Construa uma estrutura de dados para representar uma árvore binária. A estrutura deve conter os seguintes campos:

- Ponteiro para o nó esquerdo
- Ponteiro para o nó direito
- Dado armazenado no nó

- Código:

```
#include <stdio.h>
#include <stdlib.h>

//definição do nó da árvore binária
typedef struct Node {
    int valor;
    struct Node *esquerda;
    struct Node *direita;
} Node;

//função para criar um novo nó
Node* criar_no(int valor) {
    Node* novo_no = (Node*) malloc(sizeof(Node));
    if (novo_no == NULL) {
        printf("Erro na alocação de memória.\n");
        exit(1);
    }

    novo_no->valor = valor;
    novo_no->esquerda = NULL;
    novo_no->direita = NULL;
    return novo_no;
}

//função principal
int main() {
    // Cria a raiz
    Node *raiz = criar_no(10);
```

```

// Adiciona filho à direita
raiz->direita = criar_no(20);

// Adiciona filho à esquerda
raiz->esquerda = criar_no(5);

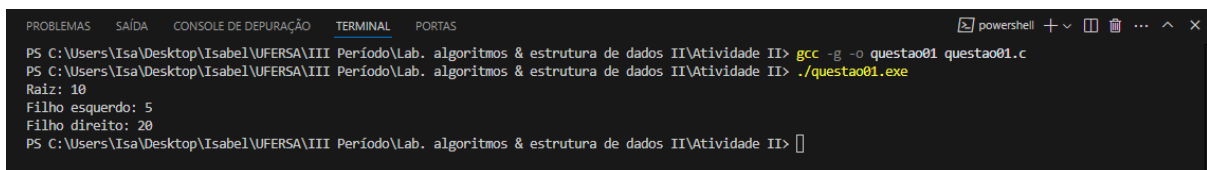
printf("Raiz: %d\n", raiz->valor);
printf("Filho esquerdo: %d\n", raiz->esquerda->valor);
printf("Filho direito: %d\n", raiz->direita->valor);

// Libera a memória
free(raiz->esquerda);
free(raiz->direita);
free(raiz);

return 0;
}

```

- Saída do código:



```

PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO  TERMINAL  PORTAS
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> gcc -g -o questao01 questao01.c
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> ./questao01.exe
Raiz: 10
Filho esquerdo: 5
Filho direito: 20
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> 

```

## 2. CRIAÇÃO DE UMA ÁRVORE BINÁRIA

- Código:

```

#include <stdio.h>
#include <stdlib.h>

//definição do nó da árvore binária
typedef struct Node {
    int valor;
    struct Node *esquerda;
    struct Node *direita;
} Node;

//função para criar um novo nó
Node* criar_no(int valor) {

```

```

Node* novo_no = (Node*) malloc(sizeof(Node));
if (novo_no == NULL) {
    printf("Erro ao alocar memoria.\n");
    exit(1);
}
novo_no->valor = valor;
novo_no->esquerda = NULL;
novo_no->direita = NULL;
return novo_no;
}

//função para imprimir a árvore (em pré-ordem)
void imprimir(Node* raiz) {
    if (raiz != NULL) {
        printf("%d ", raiz->valor);
        imprimir(raiz->esquerda);
        imprimir(raiz->direita);
    }
}

//função para liberar a memória da árvore
void liberar(Node* raiz) {
    if (raiz != NULL) {
        liberar(raiz->esquerda);
        liberar(raiz->direita);
        free(raiz);
    }
}

//função principal
int main() {
    //criação dos nós
    Node *raiz = criar_no(10);
    raiz->direita = criar_no(20);
    raiz->direita->esquerda = criar_no(15);

    //impressão da árvore
    printf("Percorrendo a arvore (pre-ordem): ");

```

```

    imprimir(raiz);
    printf("\n");

    //liberação da memória
    liberar(raiz);

    return 0;
}

```

- Saída do código:

```

C:\> Se você confia nesse comando, digite: "questao02.exe" - consulte "get help about Command Precedence" para obter mais detalhes.
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> ./questao02.exe
Percorrendo a árvore (pre-ordem): 10 20 15
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> 

```

### 3. REMOÇÃO DE UMA ÁRVORE BINÁRIA

- Código:

```

#include <stdio.h>
#include <stdlib.h>

// definição do nó da árvore binária
typedef struct Node {
    int valor;
    struct Node *esquerda;
    struct Node *direita;
} Node;

// função para criar o nó
Node* criar_no(int valor){
    Node* novo = malloc(sizeof(Node));
    if(novo) {
        novo->direita = NULL;
        novo->esquerda = NULL;
        novo->valor = valor;
    }
    return novo;
}

// função para remover o nó (com impressão)
void remover_arvore(Node* T) {
    if (T != NULL) {
        remover_arvore(T->esquerda);
        remover_arvore(T->direita);
    }
}

```

```

        printf("Removendo nó com valor: %d\n", T->valor);
        free(T);
    }
}

// função principal
int main() {
    Node *raiz = criar_no(10);
    raiz->direita = criar_no(20);
    raiz->direita->esquerda = criar_no(15);

    // libera memória e imprime os nós removidos
    remover_arvore(raiz);

    return 0;
}

```

- Saída do código:

```

PROBLEMAS  SAÍDA  CONSOLE DE DEPUÇÃO  TERMINAL  PORTAS
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> gcc questao03.c -o questao03.exe
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> ./questao03.exe
Removendo nó com valor: 15
Removendo nó com valor: 20
Removendo nó com valor: 10
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II>

```

#### 4. BUSCA EM UMA ÁRVORE BINÁRIA

- Código:

```

#include <stdio.h>
#include <stdlib.h>

//definição do nó da árvore binária
typedef struct Node {
    int valor;
    struct Node *esquerda;
    struct Node *direita;
} Node;

//criar um novo nó
Node* criar_no(int valor){
    Node* novo = malloc(sizeof(Node));
    if(novo) {
        novo->direita = NULL;
    }
}

```

```

        novo->esquerda = NULL;
        novo->valor = valor;
    }
    return novo;
}

//inserir um valor na árvore
Node* inserir(Node* raiz, int valor){
    if(raiz == NULL){
        return criar_no(valor);
    }
    if(valor < raiz->valor){
        raiz->esquerda = inserir(raiz->esquerda, valor);
    } else if (valor > raiz->valor){
        raiz->direita = inserir(raiz->direita, valor);
    }
    return raiz;
}

//função de busca
Node* buscar(Node* raiz, int valor) {
    if (raiz == NULL) {
        return NULL;
    }
    if (raiz->valor == valor) {
        return raiz;
    }
    if (valor < raiz->valor) {
        return buscar(raiz->esquerda, valor);
    } else {
        return buscar(raiz->direita, valor);
    }
}

//função principal
int main(){
    Node *raiz = NULL;
    raiz = inserir(raiz, 20);
    raiz = inserir(raiz, 10);
    raiz = inserir(raiz, 18);
    raiz = inserir(raiz, 19);
    raiz = inserir(raiz, 17);
    raiz = inserir(raiz, 14);

```

```

    raiz = inserir(raiz, 41);
    raiz = inserir(raiz, 23);
    raiz = inserir(raiz, 15);
    raiz = inserir(raiz, 31);

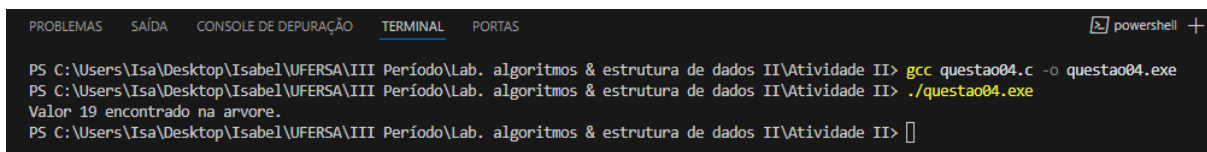
    int chave = 19;
    Node* resultado = buscar(raiz, chave);

    if(resultado != NULL){
        printf("Valor %d encontrado na arvore.\n", resultado->valor);
    } else {
        printf("Valor %d não encontrado na arvore.\n", chave);
    }

    return 0;
}

```

- Saída:



```

PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO  TERMINAL  PORTAS  powershell +
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> gcc questao04.c -o questao04.exe
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> ./questao04.exe
Valor 19 encontrado na arvore.
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> 

```

## 5. INSERÇÃO EM UMA ÁRVORE BINÁRIA

- Código:

```

#include <stdio.h>
#include <stdlib.h>

//definição do nó da árvore binária
typedef struct Node {
    int valor;
    struct Node *esquerda;
    struct Node *direita;
} Node;

// Função para criar um novo nó
Node* criar_no(int valor){
    Node* novo = malloc(sizeof(Node));
    if(novo){
        novo->direita = NULL;
        novo->esquerda = NULL;
        novo->valor = valor;
    }
}

```



```

    }
    return novo;
}

//função para inserir na árvore binária
Node* inserir(Node* raiz, int valor){
    if(raiz == NULL){
        return criar_no(valor);
    }
    if(valor < raiz->valor){
        raiz->esquerda = inserir(raiz->esquerda, valor);
    }
    else if(valor > raiz->valor){
        raiz->direita = inserir(raiz->direita, valor);
    }
    //se o valor já existe, não faz nada
    return raiz;
}

//percurso em pré-ordem
void pre_ordem(Node* raiz){
    if(raiz != NULL){
        printf("%d ", raiz->valor);
        pre_ordem(raiz->esquerda);
        pre_ordem(raiz->direita);
    }
}

//função principal
int main(){
    Node *raiz = NULL;

    //inserindo nós
    raiz = inserir(raiz, 20);
    raiz = inserir(raiz, 10);
    raiz = inserir(raiz, 18);
    raiz = inserir(raiz, 19);
    raiz = inserir(raiz, 17);
    raiz = inserir(raiz, 14);
    raiz = inserir(raiz, 41);
    raiz = inserir(raiz, 23);
    raiz = inserir(raiz, 15);

```

```

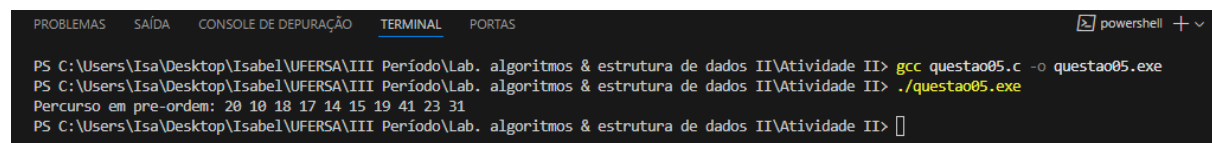
    raiz = inserir(raiz, 31);

    //impressão em pré-ordem
    printf("Percurso em pre-ordem: ");
    pre_ordem(raiz);
    printf("\n");

    return 0;
}

```

- Saída:



```

PROBLEMAS  SAÍDA  CONSOLE DE DEPUÇÃO  TERMINAL  PORTAS
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> gcc questao05.c -o questao05.exe
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> ./questao05.exe
Percurso em pre-ordem: 20 10 18 17 14 15 19 41 23 31
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> 

```

## 6. EXERCÍCIOS

a) Implemente a estrutura de dados para a árvore binária de busca

```

#include <stdio.h>
#include <stdlib.h>

//definição da estrutura do nó da árvore binária de busca
typedef struct No {
    int dado;
    struct No* esquerdo;
    struct No* direito;
} No;

```

b) Implemente as funções de criação, remo, busca e inserção.

```

#include <stdio.h>
#include <stdlib.h>

//definição da estrutura do nó
typedef struct Node {
    int valor;
    struct Node* esquerdo;
    struct Node* direito;
} Node;

//função para criar um nó

```

```

Node* criar_no(int valor) {
    Node* novo = (Node*) malloc(sizeof(Node));
    if (novo) {
        novo->valor = valor;
        novo->esquerdo = NULL;
        novo->direito = NULL;
    }
    return novo;
}

//função para inserir na árvore
Node* inserir(Node* raiz, int valor) {
    if (raiz == NULL) {
        return criar_no(valor);
    }
    if (valor < raiz->valor) {
        raiz->esquerdo = inserir(raiz->esquerdo, valor);
    } else if (valor > raiz->valor) {
        raiz->direito = inserir(raiz->direito, valor);
    }
    return raiz;
}

//função para buscar um valor na árvore
Node* buscar(Node* raiz, int valor) {
    if (raiz == NULL) {
        return NULL;
    }
    if (raiz->valor == valor) {
        return raiz;
    }
    if (valor < raiz->valor) {
        return buscar(raiz->esquerdo, valor);
    } else {
        return buscar(raiz->direito, valor);
    }
}

//função para remover todos os nós
void remover_arvore(Node* raiz) {
    if (raiz != NULL) {
        remover_arvore(raiz->esquerdo);
        remover_arvore(raiz->direito);
    }
}

```

```

        free(raiz);
    }
}

//função para exibir a árvore
void em_ordem(Node* raiz) {
    if (raiz != NULL) {
        em_ordem(raiz->esquerdo);
        printf("%d ", raiz->valor);
        em_ordem(raiz->direito);
    }
}

//função principal
int main() {
    Node* raiz = NULL;

    raiz = inserir(raiz, 50);
    raiz = inserir(raiz, 30);
    raiz = inserir(raiz, 70);
    raiz = inserir(raiz, 20);
    raiz = inserir(raiz, 40);
    raiz = inserir(raiz, 60);
    raiz = inserir(raiz, 80);

    printf("arvore em ordem: ");
    em_ordem(raiz);
    printf("\n");

    // busca
    int valor_buscado = 40;
    Node* resultado = buscar(raiz, valor_buscado);
    if (resultado != NULL) {
        printf("Valor %d encontrado na arvore.\n", valor_buscado);
    } else {
        printf("Valor %d nao encontrado na arvore.\n", valor_buscado);
    }

    //remoção de toda a árvore
    remover_arvore(raiz);
    printf("arvore removida.\n");

    return 0;
}

```

```
}
```

c) Teste as funções com diferentes casos de teste.

```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURACÃO  TERMINAL  PORTAS
PS C:\Users\Isa\Desktop\Isabel\UFERSA\VIII Período\Lab. algoritmos & estrutura de dados II\Atividade II> gcc questao06b.c -o questao06b.exe
PS C:\Users\Isa\Desktop\Isabel\UFERSA\VIII Período\Lab. algoritmos & estrutura de dados II\Atividade II> .\questao06b.exe
arvore em ordem: 20 30 40 50 60 70 80
Valor 40 encontrado na arvore.
arvore removida.
PS C:\Users\Isa\Desktop\Isabel\UFERSA\VIII Período\Lab. algoritmos & estrutura de dados II\Atividade II> █
```

d) Crie um algoritmo para percorrer a árvore em ordem (in-ordem).

```
#include <stdio.h>
#include <stdlib.h>

//definição da estrutura do nó
typedef struct Node {
    int valor;
    struct Node* esquerda;
    struct Node* direita;
} Node;

//função para criar um nó
Node* criar_no(int valor) {
    Node* novo = (Node*) malloc(sizeof(Node));
    if (novo) {
        novo->valor = valor;
        novo->esquerda = NULL;
        novo->direita = NULL;
    }
    return novo;
}

//função para inserir na árvore
Node* inserir(Node* raiz, int valor) {
    if (raiz == NULL) {
        return criar_no(valor);
    }
    if (valor < raiz->valor) {
        raiz->esquerda = inserir(raiz->esquerda, valor);
    } else if (valor > raiz->valor) {
        raiz->direita = inserir(raiz->direita, valor);
    }
    return raiz;
}
```

```

//função para buscar um valor na árvore
Node* buscar(Node* raiz, int valor) {
    if (raiz == NULL) {
        return NULL;
    }
    if (raiz->valor == valor) {
        return raiz;
    }
    if (valor < raiz->valor) {
        return buscar(raiz->esquerda, valor);
    } else {
        return buscar(raiz->direita, valor);
    }
}

//função para remover todos os nós
void remover_arvore(Node* raiz) {
    if (raiz != NULL) {
        remover_arvore(raiz->esquerda);
        remover_arvore(raiz->direita);
        free(raiz);
    }
}

//função para exibir a árvore em ordem - in-ordem
void em_ordem(Node* raiz) {
    if (raiz != NULL) {
        em_ordem(raiz->esquerda);
        printf("%d ", raiz->valor);
        em_ordem(raiz->direita);
    }
}

//função principal
int main() {
    Node* raiz = NULL;

    //inserção de valores na árvore
    raiz = inserir(raiz, 1);
    raiz = inserir(raiz, 2);
    raiz = inserir(raiz, 4);
    raiz = inserir(raiz, 8);
    raiz = inserir(raiz, 16);
}

```

```

    raiz = inserir(raiz, 32);
    raiz = inserir(raiz, 64);

    //exibir a árvore em ordem
    printf("arvore em ordem: ");
    em_ordem(raiz);
    printf("\n");

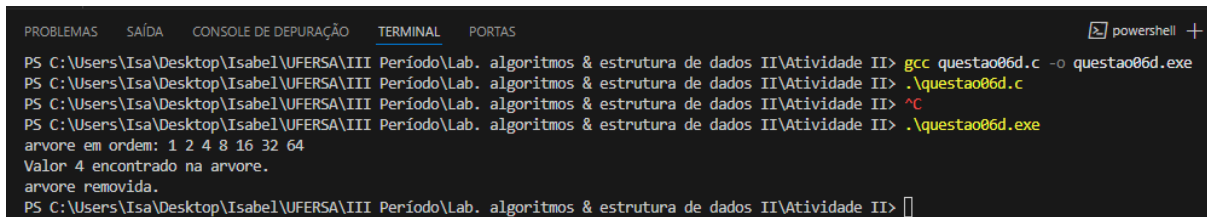
    //buscar um valor na árvore
    int valor_buscado = 4;
    Node* resultado = buscar(raiz, valor_buscado);
    if (resultado != NULL) {
        printf("Valor %d encontrado na arvore.\n", valor_buscado);
    } else {
        printf("Valor %d nao encontrado na arvore.\n", valor_buscado);
    }

    //remover toda a árvore
    remover_arvore(raiz);
    printf("arvore removida.\n");

    return 0;
}

```

- Saída do código:



```

PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> gcc questao06d.c -o questao06d.exe
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> .\questao06d.c
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> ^C
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> .\questao06d.exe
arvore em ordem: 1 2 4 8 16 32 64
Valor 4 encontrado na arvore.
arvore removida.
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> 

```

e) Crie um algoritmo para percorrer a árvore em pré-ordem (pré-ordem).

```

#include <stdio.h>
#include <stdlib.h>

//definição da estrutura do nó
typedef struct Node {
    int valor;
    struct Node* esquerda;
    struct Node* direita;
} Node;

```

```

//função para criar um nó
Node* criar_no(int valor) {
    Node* novo = (Node*) malloc(sizeof(Node));
    if (novo) {
        novo->valor = valor;
        novo->esquerda = NULL;
        novo->direita = NULL;
    }
    return novo;
}

//função para inserir na árvore
Node* inserir(Node* raiz, int valor) {
    if (raiz == NULL) {
        return criar_no(valor);
    }
    if (valor < raiz->valor) {
        raiz->esquerda = inserir(raiz->esquerda, valor);
    } else if (valor > raiz->valor) {
        raiz->direita = inserir(raiz->direita, valor);
    }
    return raiz;
}

//função para exibir a árvore em pós-ordem
void pos_ordem(Node* raiz) {
    if (raiz != NULL) {
        pos_ordem(raiz->esquerda);
        pos_ordem(raiz->direita);
        printf("%d ", raiz->valor);
    }
}

int main() {
    Node* raiz = NULL;

    raiz = inserir(raiz, 50);
    raiz = inserir(raiz, 30);
    raiz = inserir(raiz, 70);
    raiz = inserir(raiz, 20);
    raiz = inserir(raiz, 40);
    raiz = inserir(raiz, 60);
}

```



```

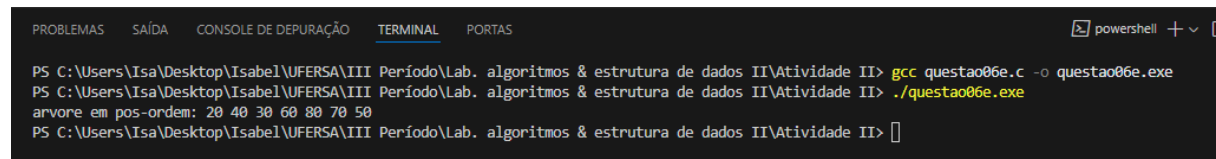
    raiz = inserir(raiz, 80);

    printf("arvore em pos-ordem: ");
    pos_ordem(raiz);
    printf("\n");

    return 0;
}

```

- Saída do código:



```

PROBLEMAS  SAÍDA  CONSOLE DE DEPURACÃO  TERMINAL  PORTAS
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> gcc questao06e.c -o questao06e.exe
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> ./questao06e.exe
arvore em pos-ordem: 20 40 30 60 80 70 50
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> 

```

f) Crie um algoritmo para percorrer a árvore em pós-ordem (pos-ordem).

```

#include <stdio.h>
#include <stdlib.h>

//definição da estrutura do nó
typedef struct Node {
    int valor;
    struct Node* esquerdo;
    struct Node* direito;
} Node;

//função para criar um nó
Node* criar_no(int valor) {
    Node* novo = (Node*) malloc(sizeof(Node));
    if (novo) {
        novo->valor = valor;
        novo->esquerdo = NULL;
        novo->direito = NULL;
    }
    return novo;
}

//função para inserir na árvore
Node* inserir(Node* raiz, int valor) {
    if (raiz == NULL) {
        return criar_no(valor);
    }
    if (valor < raiz->valor) {

```

```

        raiz->esquerdo = inserir(raiz->esquerdo, valor);
    } else if (valor > raiz->valor) {
        raiz->direito = inserir(raiz->direito, valor);
    }
    return raiz;
}

//função para buscar um valor na árvore
Node* buscar(Node* raiz, int valor) {
    if (raiz == NULL) {
        return NULL;
    }
    if (raiz->valor == valor) {
        return raiz;
    }
    if (valor < raiz->valor) {
        return buscar(raiz->esquerdo, valor);
    } else {
        return buscar(raiz->direito, valor);
    }
}

//função para remover todos os nós
void remover_arvore(Node* raiz) {
    if (raiz != NULL) {
        remover_arvore(raiz->esquerdo);
        remover_arvore(raiz->direito);
        free(raiz);
    }
}

//função para exibir a árvore em pós-ordem
void pos_ordem(Node* raiz) {
    if (raiz != NULL) {
        pos_ordem(raiz->esquerdo);
        pos_ordem(raiz->direito);
        printf("%d ", raiz->valor);
    }
}

//função principal
int main() {
    Node* raiz = NULL;

```

```

    raiz = inserir(raiz, 50);
    raiz = inserir(raiz, 30);
    raiz = inserir(raiz, 70);
    raiz = inserir(raiz, 20);
    raiz = inserir(raiz, 40);
    raiz = inserir(raiz, 60);
    raiz = inserir(raiz, 80);

    printf("arvore em pos-ordem: ");
    pos_ordem(raiz);
    printf("\n");

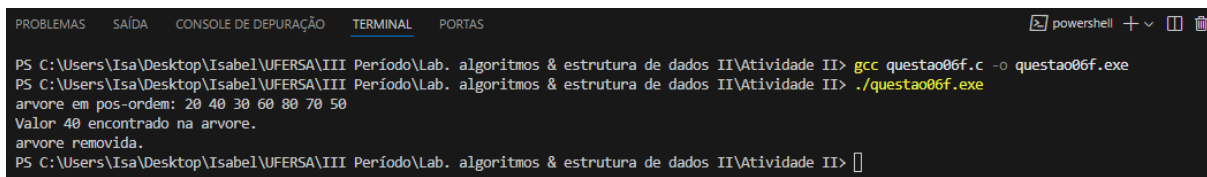
    // busca
    int valor_buscado = 40;
    Node* resultado = buscar(raiz, valor_buscado);
    if (resultado != NULL) {
        printf("Valor %d encontrado na arvore.\n", valor_buscado);
    } else {
        printf("Valor %d não encontrado na arvore.\n", valor_buscado);
    }

    //remoção de toda a árvore
    remover_arvore(raiz);
    printf("arvore removida.\n");

    return 0;
}

```

- Saída do código:



```

PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> gcc questao06f.c -o questao06f.exe
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> ./questao06f.exe
arvore em pos-ordem: 20 40 30 60 80 70 50
Valor 40 encontrado na arvore.
arvore removida.
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> 

```

g) Crie um algoritmo para calcular a altura da árvore.

```

#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int valor;
    struct Node* esquerdo;

```

```

    struct Node* direito;
} Node;

int altura(Node* raiz) {
    if (raiz == NULL) return -1;
    int e = altura(raiz->esquerdo);
    int d = altura(raiz->direito);
    return (e > d ? e : d) + 1;
}

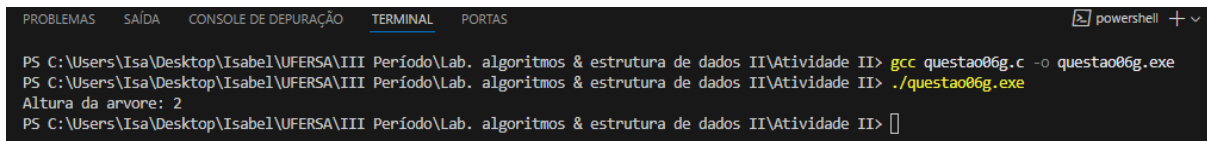
int main() {
    Node n1 = {20, NULL, NULL};
    Node n2 = {40, NULL, NULL};
    Node n3 = {30, &n1, &n2};
    Node n4 = {70, NULL, NULL};
    Node raiz = {50, &n3, &n4};

    printf("Altura da arvore: %d\n", altura(&raiz));

    return 0;
}

```

- Saída do código:



```

PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> gcc questao06g.c -o questao06g.exe
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> ./questao06g.exe
Altura da arvore: 2
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> 

```

h) Crie um algoritmo para calcular a profundidade de um nó na árvore.

```

#include <stdio.h>
#include <stdlib.h>

//definição da estrutura do nó
typedef struct Node {
    int valor;
    struct Node* esquerdo;
    struct Node* direito;
} Node;

//função para criar um nó
Node* criar_no(int valor) {
    Node* novo = (Node*) malloc(sizeof(Node));

```

```

    if (novo) {
        novo->valor = valor;
        novo->esquerdo = NULL;
        novo->direito = NULL;
    }
    return novo;
}

//função para inserir na árvore
Node* inserir(Node* raiz, int valor) {
    if (raiz == NULL) {
        return criar_no(valor);
    }
    if (valor < raiz->valor) {
        raiz->esquerdo = inserir(raiz->esquerdo, valor);
    } else if (valor > raiz->valor) {
        raiz->direito = inserir(raiz->direito, valor);
    }
    return raiz;
}

//função para calcular a profundidade de um nó
int profundidade(Node* raiz, int valor) {
    if (raiz == NULL) {
        return -1; // nó não encontrado
    }
    if (raiz->valor == valor) {
        return 0; // raiz do próprio nó
    }
    if (valor < raiz->valor) {
        int prof = profundidade(raiz->esquerdo, valor);
        return (prof >= 0) ? prof + 1 : -1;
    } else {
        int prof = profundidade(raiz->direito, valor);
        return (prof >= 0) ? prof + 1 : -1;
    }
}

//função principal
int main() {
    Node* raiz = NULL;

    //inserindo valores na árvore

```

```

    raiz = inserir(raiz, 50);
    raiz = inserir(raiz, 30);
    raiz = inserir(raiz, 70);
    raiz = inserir(raiz, 20);
    raiz = inserir(raiz, 40);
    raiz = inserir(raiz, 60);
    raiz = inserir(raiz, 80);

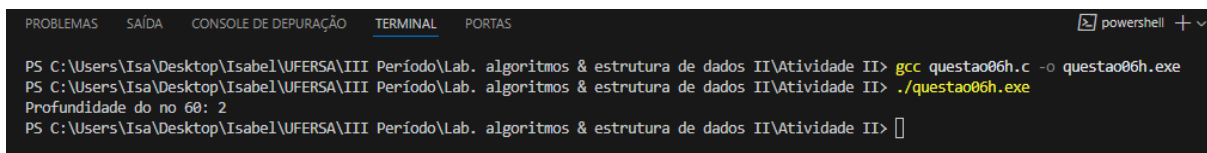
    int valor = 60;
    int prof = profundidade(raiz, valor);

    if (prof != -1) {
        printf("Profundidade do nó %d: %d\n", valor, prof);
    } else {
        printf("Nó %d não encontrado na árvore.\n", valor);
    }

    return 0;
}

```

- Saída do código:



```

PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> gcc questao06h.c -o questao06h.exe
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> ./questao06h.exe
Profundidade do nó 60: 2
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II>

```

i) Crie um algoritmo para calcular a soma dos valores armazenados na árvore.

```

#include <stdio.h>
#include <stdlib.h>

//definição do nó da árvore
typedef struct Node {
    int valor;
    struct Node* esquerdo;
    struct Node* direito;
} Node;

//função para criar um nó
Node* criar_no(int valor) {
    Node* novo = (Node*) malloc(sizeof(Node));
    if (novo) {
        novo->valor = valor;
    }
}

```

```

        novo->esquerdo = NULL;
        novo->direito = NULL;
    }
    return novo;
}

//função para inserir na árvore
Node* inserir(Node* raiz, int valor) {
    if (raiz == NULL) {
        return criar_no(valor);
    }
    if (valor < raiz->valor) {
        raiz->esquerdo = inserir(raiz->esquerdo, valor);
    } else if (valor > raiz->valor) {
        raiz->direito = inserir(raiz->direito, valor);
    }
    return raiz;
}

//função para calcular a soma dos valores da árvore
int soma(Node* raiz) {
    if (raiz == NULL) {
        return 0;
    }
    return raiz->valor + soma(raiz->esquerdo) + soma(raiz->direito);
}

//função principal
int main() {
    Node* raiz = NULL;

    //inserindo valores na árvore
    raiz = inserir(raiz, 50);
    raiz = inserir(raiz, 30);
    raiz = inserir(raiz, 70);
    raiz = inserir(raiz, 20);
    raiz = inserir(raiz, 40);
    raiz = inserir(raiz, 60);
    raiz = inserir(raiz, 80);

    //calculando a soma dos valores da árvore
    int resultado = soma(raiz);
    printf("Soma dos valores da arvore: %d\n", resultado);
}

```

```
    return 0;
}
```

- Saída do código:

```
PROBLEMAS  SAÍDA  CONSOLE DE DEPUÇÃO  TERMINAL  PORTAS  powershell +
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> gcc questao06i.c -o questao06i.exe
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> ./questao06i.exe
Soma dos valores da árvore: 350
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> []
```

j) Crie um algoritmo para calcular o nível de um nó na árvore.

```
#include <stdio.h>
#include <stdlib.h>

//definição do nó da árvore
typedef struct Node {
    int valor;
    struct Node* esquerdo;
    struct Node* direito;
} Node;

//função para criar um nó
Node* criar_no(int valor) {
    Node* novo = (Node*) malloc(sizeof(Node));
    if (novo) {
        novo->valor = valor;
        novo->esquerdo = NULL;
        novo->direito = NULL;
    }
    return novo;
}

//função para inserir na árvore
Node* inserir(Node* raiz, int valor) {
    if (raiz == NULL) {
        return criar_no(valor);
    }
    if (valor < raiz->valor) {
        raiz->esquerdo = inserir(raiz->esquerdo, valor);
    } else if (valor > raiz->valor) {
        raiz->direito = inserir(raiz->direito, valor);
    }
}
```



```

        return raiz;
    }

    //função para calcular o nível de um nó
    int nivel(Node* raiz, int valor, int nivel_atual) {
        if (raiz == NULL) {
            return -1; //valor não encontrado
        }

        if (raiz->valor == valor) {
            return nivel_atual;
        }

        if (valor < raiz->valor) {
            return nivel(raiz->esquerdo, valor, nivel_atual + 1);
        } else {
            return nivel(raiz->direito, valor, nivel_atual + 1);
        }
    }

    //função principal
    int main() {
        Node* raiz = NULL;

        //inserindo valores na árvore
        raiz = inserir(raiz, 50);
        raiz = inserir(raiz, 30);
        raiz = inserir(raiz, 70);
        raiz = inserir(raiz, 20);
        raiz = inserir(raiz, 40);
        raiz = inserir(raiz, 60);
        raiz = inserir(raiz, 80);

        int valor_procurado = 60;
        int resultado = nivel(raiz, valor_procurado, 0);

        if (resultado != -1) {
            printf("O nível do valor %d na árvore é: %d\n",
valor_procurado, resultado);
        } else {
            printf("Valor %d não encontrado na árvore.\n",
valor_procurado);
        }
    }

```

```
    return 0;
}
```

- Saída do código:

```
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> gcc questao06j.c -o questao06j.exe
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> ./questao06j.exe
O nível do valor 60 na árvore é: 2
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> █
```

k) Crie um algoritmo para calcular o número de nós na árvore.

```
#include <stdio.h>
#include <stdlib.h>

//definição do nó da árvore
typedef struct Node {
    int valor;
    struct Node* esquerdo;
    struct Node* direito;
} Node;

//função para criar um nó
Node* criar_no(int valor) {
    Node* novo = (Node*) malloc(sizeof(Node));
    if (novo) {
        novo->valor = valor;
        novo->esquerdo = NULL;
        novo->direito = NULL;
    }
    return novo;
}

//função para inserir na árvore
Node* inserir(Node* raiz, int valor) {
    if (raiz == NULL) {
        return criar_no(valor);
    }
    if (valor < raiz->valor) {
        raiz->esquerdo = inserir(raiz->esquerdo, valor);
    } else if (valor > raiz->valor) {
        raiz->direito = inserir(raiz->direito, valor);
    }
    return raiz;
}
```

```

}

//função para contar o número de nós
int contar_nos(Node* raiz) {
    if (raiz == NULL) {
        return 0;
    }
    return 1 + contar_nos(raiz->esquerdo) + contar_nos(raiz->direito);
}

//função principal
int main() {
    Node* raiz = NULL;

    //inserindo valores na árvore
    raiz = inserir(raiz, 50);
    raiz = inserir(raiz, 30);
    raiz = inserir(raiz, 70);
    raiz = inserir(raiz, 20);
    raiz = inserir(raiz, 40);
    raiz = inserir(raiz, 60);
    raiz = inserir(raiz, 80);

    int total_nos = contar_nos(raiz);
    printf("O numero total de nos na arvore e: %d\n", total_nos);

    return 0;
}

```

- Saída do código:

```

PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> gcc questao06k.c -o questao06k.exe
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> ./questao06k.exe
O numero total de nos na arvore e: 7
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> 

```

l) Crie um algoritmo para calcular o número de folhas na árvore.

```

#include <stdio.h>
#include <stdlib.h>

//definição do nó da árvore
typedef struct Node {
    int valor;
    struct Node* esquerdo;
    struct Node* direito;
}

```

```

} Node;

//função para criar um nó
Node* criar_no(int valor) {
    Node* novo = (Node*) malloc(sizeof(Node));
    if (novo) {
        novo->valor = valor;
        novo->esquerdo = NULL;
        novo->direito = NULL;
    }
    return novo;
}

//função para inserir na árvore
Node* inserir(Node* raiz, int valor) {
    if (raiz == NULL) {
        return criar_no(valor);
    }
    if (valor < raiz->valor) {
        raiz->esquerdo = inserir(raiz->esquerdo, valor);
    } else if (valor > raiz->valor) {
        raiz->direito = inserir(raiz->direito, valor);
    }
    return raiz;
}

//função para contar o número de folhas
int contar_folhas(Node* raiz) {
    if (raiz == NULL) {
        return 0;
    }
    if (raiz->esquerdo == NULL && raiz->direito == NULL) {
        return 1;
    }
    return contar_folhas(raiz->esquerdo) +
    contar_folhas(raiz->direito);
}

//função principal
int main() {
    Node* raiz = NULL;

    //inserindo elementos na árvore

```

```

    raiz = inserir(raiz, 50);
    raiz = inserir(raiz, 30);
    raiz = inserir(raiz, 70);
    raiz = inserir(raiz, 20);
    raiz = inserir(raiz, 40);
    raiz = inserir(raiz, 60);
    raiz = inserir(raiz, 80);

    int folhas = contar_folhas(raiz);
    printf("O numero de folhas na arvore e: %d\n", folhas);

    return 0;
}

```

- Saída do código:

```

PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> gcc questao061.c -o questao061.exe
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> ./questao061.exe
O numero de folhas na arvore e: 4
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> 

```

Questão 02.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//definindo a estrutura do nó da árvore
typedef struct Aluno {
    char nome[50];
    int matricula;
    float nota;
    struct Aluno* esquerdo;
    struct Aluno* direito;
} Aluno;

//função para criar um novo aluno - nó
Aluno* criar_aluno(char nome[], int matricula, float nota) {
    Aluno* novo = (Aluno*) malloc(sizeof(Aluno));
    if (novo) {
        strcpy(novo->nome, nome);
        novo->matricula = matricula;
        novo->nota = nota;
        novo->esquerdo = NULL;
        novo->direito = NULL;
    }
}

```

```

    }
    return novo;
}

//função para inserir um aluno na árvore
Aluno* inserir(Aluno* raiz, char nome[], int matricula, float nota) {
    if (raiz == NULL) {
        return criar_aluno(nome, matricula, nota);
    }
    if (matricula < raiz->matricula) {
        raiz->esquerdo = inserir(raiz->esquerdo, nome, matricula,
nota);
    } else if (matricula > raiz->matricula) {
        raiz->direito = inserir(raiz->direito, nome, matricula, nota);
    }
    return raiz;
}

//função para exibir os alunos em ordem (ordenado pela matrícula)
void ordem(Aluno* raiz) {
    if (raiz != NULL) {
        ordem(raiz->esquerdo);
        printf("Nome: %s | Matricula: %d | Nota: %.2f\n", raiz->nome,
raiz->matricula, raiz->nota);
        ordem(raiz->direito);
    }
}

//função principal
int main() {
    Aluno* raiz = NULL;

    //inserindo valores
    raiz = inserir(raiz, "Isabel", 102, 9.5);
    raiz = inserir(raiz, "Itallo", 101, 7.8);
    raiz = inserir(raiz, "Charles", 105, 8.2);
    raiz = inserir(raiz, "Iara", 103, 6.9);
    raiz = inserir(raiz, "Ikaro", 104, 9.0);

    //exibir os alunos em ordem crescente de matrícula
    printf("Alunos em ordem de matricula:\n");
    ordem(raiz);
}

```

```
    return 0;
}
```

- Saída do código:

```
PROBLEMAS  SAÍDA  CONSOLE DE DEPUÇÃO  TERMINAL  PORTAS  powershell +
compilation terminated.
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> gcc questao07.c -o questao07.exe
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> ./questao07.exe
Alunos em ordem de matricula:
Nome: Itallo | Matricula: 101 | Nota: 7.80
Nome: Isabel | Matricula: 102 | Nota: 9.50
Nome: Iara | Matricula: 103 | Nota: 6.90
Nome: Ikaro | Matricula: 104 | Nota: 9.00
Nome: Charles | Matricula: 105 | Nota: 8.20
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> |
```

Questão 03.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//definição da estrutura do nó
typedef struct Aluno {
    char nome[50];
    int matricula;
    float nota;
    struct Aluno* esquerdo;
    struct Aluno* direito;
} Aluno;

//criar um novo nó
Aluno* criar_aluno(char nome[], int matricula, float nota) {
    Aluno* novo = (Aluno*) malloc(sizeof(Aluno));
    if (novo) {
        strcpy(novo->nome, nome);
        novo->matricula = matricula;
        novo->nota = nota;
        novo->esquerdo = NULL;
        novo->direito = NULL;
    }
    return novo;
}

//inserir aluno na árvore (baseado na matrícula)
Aluno* inserir(Aluno* raiz, char nome[], int matricula, float nota) {
    if (raiz == NULL) {
```

```

        return criar_aluno(nome, matricula, nota);
    }
    if (matricula < raiz->matricula) {
        raiz->esquerdo = inserir(raiz->esquerdo, nome, matricula,
nota);
    } else if (matricula > raiz->matricula) {
        raiz->direito = inserir(raiz->direito, nome, matricula, nota);
    }
    return raiz;
}

//buscar aluno na árvore pela matrícula
Aluno* buscar(Aluno* raiz, int matricula) {
    if (raiz == NULL) {
        return NULL;
    }
    if (raiz->matricula == matricula) {
        return raiz;
    }
    if (matricula < raiz->matricula) {
        return buscar(raiz->esquerdo, matricula);
    } else {
        return buscar(raiz->direito, matricula);
    }
}

//exibir alunos em ordem de matrícula
void ordem(Aluno* raiz) {
    if (raiz != NULL) {
        ordem(raiz->esquerdo);
        printf("Nome: %s | Matricula: %d | Nota: %.2f\n", raiz->nome,
raiz->matricula, raiz->nota);
        ordem(raiz->direito);
    }
}

//função principal
int main() {
    Aluno* raiz = NULL;

    //inserindo valores
    raiz = inserir(raiz, "Isabel", 102, 9.5);
    raiz = inserir(raiz, "Itallo", 101, 7.8);

```



```

    raiz = inserir(raiz, "Iara", 105, 8.2);
    raiz = inserir(raiz, "Ikaro", 103, 6.9);
    raiz = inserir(raiz, "Charles", 104, 9.0);

    //exibir alunos em ordem
    printf("Alunos em ordem de matricula:\n");
    ordem(raiz);

    // realizando uma busca
    int buscar_matricula;
    printf("\nDigite a matricula do aluno que deseja buscar: ");
    scanf("%d", &buscar_matricula);

    Aluno* resultado = buscar(raiz, buscar_matricula);
    if (resultado != NULL) {
        printf("Aluno encontrado:\n");
        printf("Nome: %s | Matricula: %d | Nota: %.2f\n",
resultado->nome, resultado->matricula, resultado->nota);
    } else {
        printf("Aluno com matricula %d nao encontrado.\n",
buscar_matricula);
    }

    return 0;
}

```

- Saída do código:

```

PROBLEMAS  SAÍDA  CONSOLE DE DEPURACÃO  TERMINAL  PORTAS
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> gcc questao08.c -o questao08.exe
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> ./questao08.exe
Alunos em ordem de matricula:
Nome: Itallo | Matricula: 101 | Nota: 7.80
Nome: Isabel | Matricula: 102 | Nota: 9.50
Nome: Ikaro | Matricula: 103 | Nota: 6.90
Nome: Charles | Matricula: 104 | Nota: 9.00
Nome: Iara | Matricula: 105 | Nota: 8.20

Digite a matricula do aluno que deseja buscar: 102
Aluno encontrado:
Nome: Isabel | Matricula: 102 | Nota: 9.50
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II> 

```

Questão 04.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// definição da estrutura do nó
typedef struct Aluno {

```

```

    char nome[50];
    int matricula;
    float nota;
    struct Aluno* esquerdo;
    struct Aluno* direito;
} Aluno;

// criar um novo nó
Aluno* criar_aluno(char nome[], int matricula, float nota) {
    Aluno* novo = (Aluno*) malloc(sizeof(Aluno));
    if (novo) {
        strcpy(novo->nome, nome);
        novo->matricula = matricula;
        novo->nota = nota;
        novo->esquerdo = NULL;
        novo->direito = NULL;
    }
    return novo;
}

// inserir aluno na árvore (baseado na matrícula)
Aluno* inserir(Aluno* raiz, char nome[], int matricula, float nota) {
    if (raiz == NULL) {
        return criar_aluno(nome, matricula, nota);
    }
    if (matricula < raiz->matricula) {
        raiz->esquerdo = inserir(raiz->esquerdo, nome, matricula,
nota);
    } else if (matricula > raiz->matricula) {
        raiz->direito = inserir(raiz->direito, nome, matricula, nota);
    }
    return raiz;
}

// buscar aluno na árvore pela matrícula
Aluno* buscar(Aluno* raiz, int matricula) {
    if (raiz == NULL) {
        return NULL;
    }
    if (raiz->matricula == matricula) {
        return raiz;
    }
    if (matricula < raiz->matricula) {

```

```

        return buscar(raiz->esquerdo, matricula);
    } else {
        return buscar(raiz->direito, matricula);
    }
}

// exibir alunos em ordem de matrícula
void ordem(Aluno* raiz) {
    if (raiz != NULL) {
        ordem(raiz->esquerdo);
        printf("Nome: %s | Matricula: %d | Nota: %.2f\n", raiz->nome,
raiz->matricula, raiz->nota);
        ordem(raiz->direito);
    }
}

// função para calcular soma das notas e total de alunos
void calcular_soma(Aluno* raiz, float* soma, int* contador) {
    if (raiz != NULL) {
        *soma += raiz->nota;
        *contador += 1;
        calcular_soma(raiz->esquerdo, soma, contador);
        calcular_soma(raiz->direito, soma, contador);
    }
}

// função para calcular a média das notas
float calcular_media(Aluno* raiz) {
    float soma = 0;
    int contador = 0;
    calcular_soma(raiz, &soma, &contador);
    if (contador == 0) return 0; // evita divisão por zero
    return soma / contador;
}

// função principal
int main() {
    Aluno* raiz = NULL;

    // inserindo valores
    raiz = inserir(raiz, "Isabel", 102, 9.5);
    raiz = inserir(raiz, "Itallo", 101, 7.8);
    raiz = inserir(raiz, "Iara", 105, 8.2);

```

```

    raiz = inserir(raiz, "Ikaro", 103, 6.9);
    raiz = inserir(raiz, "Charles", 104, 9.0);

    // exibir alunos em ordem
    printf("Alunos em ordem de matricula:\n");
    ordem(raiz);

    // buscando um aluno
    int buscar_matricula;
    printf("\nDigite a matricula do aluno que deseja buscar: ");
    scanf("%d", &buscar_matricula);

    Aluno* resultado = buscar(raiz, buscar_matricula);
    if (resultado != NULL) {
        printf("Aluno encontrado:\n");
        printf("Nome: %s | Matricula: %d | Nota: %.2f\n",
resultado->nome, resultado->matricula, resultado->nota);
    } else {
        printf("Aluno com matricula %d nao encontrado.\n",
buscar_matricula);
    }

    // calculando a média das notas
    float media = calcular_media(raiz);
    printf("\nMedia das notas dos alunos: %.2f\n", media);

    return 0;
}

```

- Saída do código:

```

PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO  TERMINAL  PORTAS

Alunos em ordem de matricula:
Nome: Itallo | Matricula: 101 | Nota: 7.80
Nome: Isabel | Matricula: 102 | Nota: 9.50
Nome: Ikaro | Matricula: 103 | Nota: 6.90
Nome: Charles | Matricula: 104 | Nota: 9.00
Nome: Iara | Matricula: 105 | Nota: 8.20

Digite a matricula do aluno que deseja buscar: 102
Aluno encontrado:
Nome: Isabel | Matricula: 102 | Nota: 9.50

Media das notas dos alunos: 8.28
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Lab. algoritmos & estrutura de dados II\Atividade II>

```