LUCAS BASTOS FRANCO

ÁRVORE BINÁRIA DE BUSCA – ÁRVORES E GRAFOS

03 / 2025

lucas.software.engineering@gmail.com
+55 62 8499-1644

SUMÁRIO

ÁRVORE BINÁRIA DE BUSCA....3
CONCEITOS TEORICOS....3
O QUE É....3
DEFINIÇÃO DE ABB....3
APLICAÇÃO DA ABB....3
OPERAÇÃO DE BUSCA....3
PARTES PRINCIPAIS DO CODIGO....3
DEFINIÇÃO....3
FUNÇÃO PARA CRIAR....4
FUNÇÃO PARA ADICIONAR....4
IMPRIMIR....4
PARTES SECUNDARIA DO CODIGO....5
BUSCAR....5
PRÉ-ORDEM....5
EM ORDEM....5
PÓS ORDEM....6
LIBERAR....6

lucas.software.engineering@gmail.com
+55 62 8499-1644

ÁRVORE BINÁRIA DE BUSCA

CONCEITOS TEORICOS

O QUE É

Uma Árvore Binária de Busca (ABB) é uma estrutura de dados em forma de árvore onde cada nó possui no máximo dois filhos (daí o nome "binária"), e os elementos são organizados de forma ordenada para facilitar a busca eficiente.

DEFINIÇÃO DE ABB

Uma ABB obedece à seguinte regra para qualquer nó:

- Todos os valores na subárvore à esquerda são menores que o valor do nó.
- Todos os valores na subárvore à direita são maiores que o valor do nó.

Essa propriedade permite que a árvore seja percorrida rapidamente para encontrar, inserir ou remover elementos.

APLICAÇÃO DA ABB

A ABB é usada em muitos contextos onde é necessário armazenar e consultar dados de forma eficiente, por exemplo:

- Sistemas de banco de dados (indexação de registros).
- Implementação de dicionários e conjuntos (como no C++, Java ou Python).
- Compiladores, para organizar variáveis e identificadores.
- Sistemas de arquivos, onde há necessidade de ordenar nomes ou valores.

OPERAÇÃO DE BUSCA

A busca em uma ABB segue a lógica:

1. Comece na raiz.
2. Compare o valor buscado com o valor do nó atual:
   - Se for igual, retorna.
   - Se for menor, vá para a subárvore da esquerda.
   - Se for maior, vá para a subárvore da direita.
3. Repita até encontrar o valor ou chegar a um nó NULL (nesse caso, o valor não está na árvore).

PARTES PRINCIPAIS DO CODIGO

DEFINIÇÃO

Cada nó de uma arvores vai precisar da variável para armazenar um valor e dois ponteiros, um sendo apontado para a esquerda e outra para a direita

```
typedef struct No {
    int valor;
    struct No* esquerda;
    struct No* direita;
```


lucas.software.engineering@gmail.com
+55 62 8499-1644

```
} No;
```

FUNÇÃO PARA CRIAR

Essa função tem como objetivo de sempre quando for inserir um valor, ela vai criar um nó para o valor

```
No* criarNo(int valor) {
    No* novo = (No*)malloc(sizeof(No));
    if (novo) {
        novo->valor = valor;
        novo->esquerda = NULL;
        novo->direita = NULL;
    }
    return novo;
}
```

FUNÇÃO PARA ADICIONAR

Essa função tem como objetivo inserir o novo valor, respeitando a seguinte regra:

- Menores que o nó atual → vão para a esquerda.
- Maiores que o nó atual → vão para a direita.

```
No* inserir(No* raiz, int valor) {
    if (raiz == NULL) {
        return criarNo(valor);
    }
    if (valor < raiz->valor) {
        raiz->esquerda = inserir(raiz->esquerda, valor);
    } else if (valor > raiz->valor) {
        raiz->direita = inserir(raiz->direita, valor);
    }
    return raiz;
}
```

IMPRIMIR

Função com o objetivo de exibir árvore

```
void imprimirArvore(No* raiz, int espaco) {
    if (raiz == NULL)
        return;

    int espacoEntreNiveis = 5;
    espaco += espacoEntreNiveis;

    imprimirArvore(raiz->direita, espaco);

    printf("\n");
```


lucas.software.engineering@gmail.com
+55 62 8499-1644

```
    for (int i = espacoEntreNiveis; i < espaco; i++)
        printf(" ");
    printf("%d\n", raiz->valor);

    imprimirArvore(raiz->esquerda, espaco);
}
```

PARTES SECUNDARIA DO CODIGO

BUSCAR

Função com o objetivo de achar o número solicitado

```
No* buscar(No* raiz, int valor) {
    if (raiz == NULL || raiz->valor == valor) {
        return raiz;
    }
    if (valor < raiz->valor) {
        return buscar(raiz->esquerda, valor);
    } else {
        return buscar(raiz->direita, valor);
    }
}
```

PRÉ-ORDEM

Função com o objetivo percorrer a árvore [raiz -> esquerda -> direita]

```
void preOrdem(No* raiz) {
    if (raiz != NULL) {
        printf("%d ", raiz->valor);
        preOrdem(raiz->esquerda);
        preOrdem(raiz->direita);
    }
}
```

EM ORDEM

Função com o objetivo percorrer a árvore [esquerda -> raiz -> direita]

```
void emOrdem(No* raiz) {
    if (raiz != NULL) {
        emOrdem(raiz->esquerda);
        printf("%d ", raiz->valor);
        emOrdem(raiz->direita);
    }
}
```


lucas.software.engineering@gmail.com
+55 62 8499-1644

PÓS ORDEM

Função com o objetivo percorrer a árvore [esquerda -> direita -> raiz]

```
void posOrdem(No* raiz) {
    if (raiz != NULL) {
        posOrdem(raiz->esquerda);
        posOrdem(raiz->direita);
        printf("%d ", raiz->valor);
    }
}
```

LIBERAR

Função com o objetivo de liberar/destruir a memória da árvore

```
void liberarArvore(No* raiz) {
    if (raiz != NULL) {
        liberarArvore(raiz->esquerda);
        liberarArvore(raiz->direita);
        free(raiz);
    }
}
```


lucas.software.engineering@gmail.com
+55 62 8499-1644