Implementação de Pesquisa em Árvore Binária em Java usando Recursão

- **Resumo:**

Contexto: Programa Java que implementa uma árvore binária e fornece funcionalidades para a inserção de valores na árvore e a realização de buscas em árvores binárias. A estrutura central do código é a árvore binária. Uma árvore binária consiste em nós, onde cada nó pode ter até dois filhos: um à esquerda e um à direita. Os nós estão organizados hierarquicamente, com um nó raiz no topo.

O código permite a inserção de valores na árvore binária; valores são inseridos de acordo com as regras da árvore binária, onde valores menores são colocados à esquerda do nó pai e valores maiores são colocados à direita, também implementa a funcionalidade de busca em uma árvore binária. A busca é realizada de acordo com as propriedades da árvore binária, onde valores menores são procurados à esquerda do nó atual e valores maiores são procurados à direita;

Usa a recursão para percorrer a árvore binária e realizar operações de busca. Isso significa que as funções se chamam a si mesmas para processar subárvores menores, o que é uma abordagem eficiente para lidar com estruturas hierárquicas como árvores binárias;

Interage com o usuário por meio da entrada e saída padrão (console) para inserir valores na árvore e exibir resultados.

Propósito: Apresentar uma implementação em Java de pesquisa em árvore binária com foco na recursão.

Metodologia: Definição da Estrutura da Árvore Binária, Implementação de Inserção, Implementação de Busca, Recursão, Entrada e Saída de Dados, Organização do Código, Comentários e Documentação.

Resultados: A implementação permite que você insira valores na árvore binária. Os resultados obtidos incluem a construção da própria árvore com base nas regras da árvore binária. A árvore deve estar organizada de forma que cada valor menor seja colocado à esquerda do nó pai e cada valor maior à direita.

A implementação permite buscar valores na árvore binária. Os resultados obtidos incluem a capacidade de encontrar valores específicos na árvore ou identificar quando um valor não está presente.

Durante a execução do programa, os resultados incluem a capacidade de inserir valores na árvore binária a partir da entrada padrão (console). Os valores inseridos são incorporados na estrutura da árvore.

Os resultados também incluem a capacidade de realizar buscas na árvore binária para encontrar valores específicos. Isso é feito interagindo com o usuário para inserir um valor a ser pesquisado, e o programa indica se o valor foi encontrado ou não.

O programa exibe os resultados da busca, informando se o valor foi encontrado ou não. Além disso, após a inserção e busca de valores, o programa pode exibir a árvore binária em ordem (in-order), mostrando os valores na sequência correta.

O programa é interativo, permitindo que o usuário insira valores e busque por eles na árvore binária.

Os resultados obtidos dependem das entradas específicas fornecidas pelo usuário durante a execução do programa. O programa em si fornece funcionalidades para a construção da árvore binária e a realização de buscas nela, mas os resultados específicos variam de acordo com os valores inseridos e pesquisados pelo usuário.

Conclusão: No contexto geral, esse código demonstra como criar, inserir dados em uma árvore binária e realizar pesquisas nessa estrutura.

- **Argumentação Teórica:**

Modelo de pesquisa utilizado: Em ordem

Recursividade: A recursão é um conceito em programação em que uma função chama a si mesma para resolver um problema. Em outras palavras, é um processo em que um problema é dividido em casos menores e mais simples do mesmo tipo, até que os casos base sejam alcançados e a solução seja obtida pela combinação dos resultados dos casos menores.

A recursão é uma abordagem útil para resolver problemas em árvores binárias por várias razões:

Estrutura Hierárquica: As árvores binárias são estruturas hierárquicas por natureza, com um nó pai que possui dois filhos (subárvores). Isso se encaixa perfeitamente no paradigma de recursão, pois você pode tratar cada subárvore como uma árvore binária em si mesma.

Divisão de Problemas: A recursão permite dividir o problema maior de percorrer ou manipular a árvore binária em subproblemas menores. Isso torna a solução mais clara e fácil de entender, pois você pode se concentrar em resolver um subproblema de cada vez.

Manutenção do Estado: Ao chamar a função recursiva, você pode manter o estado atual do processo e rastrear a posição na árvore binária. Isso é feito através dos parâmetros passados para a função recursiva.

Economia de Memória: A recursão permite economizar memória, pois cada chamada da função recursiva cria um novo contexto de execução que armazena

variáveis locais e parâmetros. Isso é mais eficiente do que criar estruturas de dados adicionais para rastrear o estado em iterações.

Expressividade e Simplicidade: A recursão muitas vezes leva a soluções mais concisas e elegantes para problemas em árvores binárias. Ela permite que você expresse a solução de forma direta, especialmente quando se trata de percorrer a árvore em ordem, pré-ordem ou pós-ordem.

Flexibilidade: A abordagem recursiva pode ser aplicada a uma ampla variedade de problemas em árvores binárias, desde a pesquisa e travessia até a inserção e remoção de nós. Isso a torna uma técnica versátil para lidar com estruturas de dados hierárquicas.

No entanto, ao usar recursão, é importante definir casos base claros para evitar loops infinitos e garantir que o algoritmo termine. Além disso, a recursão pode ter um custo adicional de memória devido à pilha de chamadas, então é necessário considerar a eficiência em termos de espaço e tempo ao aplicar recursão em problemas específicos.

Árvore binária: Uma árvore binária é uma estrutura de dados em forma de árvore usada na ciência da computação para armazenar e organizar dados de maneira hierárquica. Ela recebe o nome "binária" porque cada nó da árvore pode ter no máximo dois filhos: um filho à esquerda e outro à direita. Os nós são conectados por meio de arestas (linhas) que representam as relações hierárquicas entre os dados.

Os principais componentes de uma árvore binária incluem:

Nó Raiz: O nó superior da árvore, que é o ponto de partida para a estrutura hierárquica.

Nós: Cada elemento da árvore é chamado de nó. Cada nó possui um valor (ou dado) e pode ter até dois filhos.

Filhos: Os nós que estão diretamente conectados a um nó pai. Um nó pode ter no máximo dois filhos: um à esquerda e outro à direita.

Nó Folha: Um nó que não tem filhos é chamado de nó folha. Os nós folha estão localizados nas extremidades da árvore.

Subárvore: Uma árvore binária pode ser subdividida em subárvores, onde cada subárvore é uma árvore binária por si só.

A estrutura de uma árvore binária permite a organização eficiente dos dados de forma hierárquica, o que é útil em muitos algoritmos e estruturas de dados.

Além disso, as árvores binárias são amplamente utilizadas em algoritmos de pesquisa, como a pesquisa binária, e em algoritmos de ordenação, como o heapsort. As árvores binárias podem ser usadas para representar uma variedade de estruturas e problemas, desde a organização de dados em bancos de dados até a construção de árvores de expressões matemáticas em análise sintática. Portanto, entender os conceitos básicos das árvores binárias é fundamental para

a compreensão de estruturas de dados mais complexas e algoritmos em ciência da computação.

- **Resultados obtidos:**

```
import javax.swing.JOptionPane;

// Classe para a estrutura dos nós da árvore
class NoArvore<T extends Comparable<T>> {
    T dado;
    NoArvore<T> esquerda;
    NoArvore<T> direita;

    public NoArvore(T dado) {
        this.dado = dado;
        this.esquerda = null;
        this.direita = null;
    }
}

// Classe para entrada de dados
class Entrada {
    public static <T extends Comparable<T>> NoArvore<T> construirArvoreBinaria() {
        String entrada = JOptionPane.showInputDialog("Digite um valor (ou 'sair' para encerrar):");
        if (entrada == null || entrada.equalsIgnoreCase("sair")) {
            return null;
        }

        T dado = (T) entrada;
        NoArvore<T> no = new NoArvore<>(dado);

        no.esquerda = construirArvoreBinaria();
        no.direita = construirArvoreBinaria();

        return no;
    }
}
```

A classe “NoArvore” representa um nó na árvore binária.

A classe “Entrada” permite ao usuário criar uma árvore binária inserindo valores. A árvore é construída recursivamente.

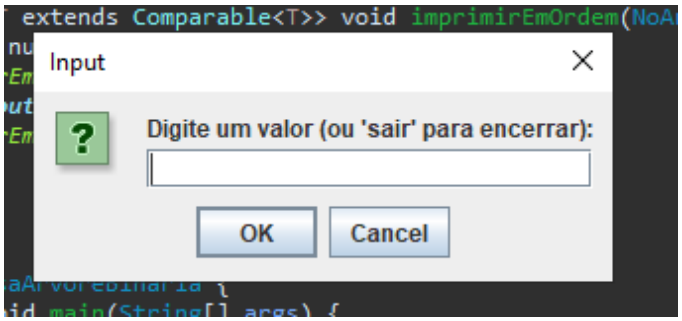
```
// Classe para saída de dados
class Saida {
    public static <T extends Comparable<T>> void imprimirEmOrdem(NoArvore<T> raiz) {
        if (raiz != null) {
            imprimirEmOrdem(raiz.esquerda);
            System.out.print(raiz.dado + " ");
            imprimirEmOrdem(raiz.direita);
        }
    }
}

public class PesquisaArvoreBinaria {
    public static void main(String[] args) {
        NoArvore<String> raiz = Entrada.construirArvoreBinaria();

        if (raiz == null) {
            System.out.println("Árvore vazia.");
        } else {
            System.out.println("Travessia em ordem (in-order):");
            Saida.imprimirEmOrdem(raiz);
        }
    }
}
```

A classe “Saida” fornece um método para imprimir a árvore em ordem (in-order).

A classe principal “PesquisaArvoreBinaria” controla o fluxo do programa e exibe os resultados.

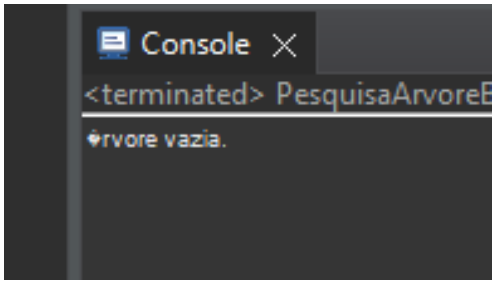


Ao rodar o projeto, aparecerá uma mensagem para inserir os valores, como no display acima.

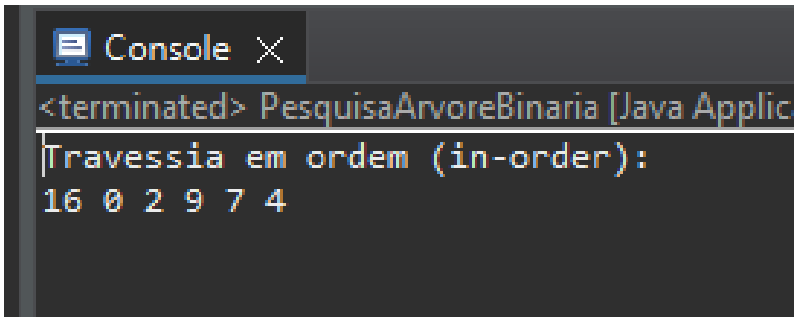
Pode ser inserido quantos valores quiser.

*Observação: Caso inseriu cinco valores, por exemplo, deverá utilizar a ação de “sair” cinco vezes.

Caso não insira nenhum valor, aparecerá “árvore vazia”.



Ao sair, aparecerá o resultado:



```java
public class PesquisanaArvoreBinaria {

      // Solicita ao usuário o número a ser pesquisado
    String valorPesquisado = JOptionPane.showInputDialog("Digite um valor para pesquisar:");

    // Executa a pesquisa na árvore com o valor fornecido pelo usuário
    <raiz> void pesquisaNaArvore(raiz valorPesquisado) {
        System.out.println("\nO valor está na árvore.");

        System.out.println("\nO valor não está na árvore.");
    }
    public static <T extends Comparable<T>> boolean pesquisaNaArvore(NoArvore<T> raiz, T valor) {
        if (raiz == null) {
            return false; // A árvore está vazia, o valor não está presente.
        }

        int comparacao = valor.compareTo(raiz.dado);

        if (comparacao == 0) {
            return true; // Encontrou o valor na raiz.
        } else if (comparacao < 0) {
            return pesquisaNaArvore(raiz.esquerda, valor); // Pesquisar na subárvore esquerda.
        } else {
            return pesquisaNaArvore(raiz.direita, valor); // Pesquisar na subárvore direita.
        }
    }

    public static void main(String[] args) {
        NoArvore<String> raiz = Entrada.construirArvoreBinaria();

        if (raiz == null) {
            System.out.println("Árvore vazia.");
        } else {
            System.out.println("Travessia em ordem (in-order):");
            Saida.imprimirEmOrdem(raiz);

            // Exemplo de pesquisa de um valor na árvore
            String valorPesquisado = JOptionPane.showInputDialog("Digite um valor para pesquisar:");
            if (pesquisaNaArvore(raiz, valorPesquisado)) {
```

```java
                    System.out.println("\nO valor está na árvore.");
                } else {
                    System.out.println("\nO valor não está na árvore.");
                }
            }
        }
    }

}
```

A classe “PesquisanaArvoreBinaria” é responsável por buscar o valor dentro da árvore.