FATEC SÃO CAETANO DO SUL – ANTONIO RUSSO

Bruno Carvalho Martins

Carolina Moreira Pavan

ESTRUTURAS DE DADOS

INSTRUMENTO AVALIATIVO DO TERCEIRO BIMESTRE

27 DE SETEMBRO DE 2023

PROFESSOR CARLOS HENRIQUE VERÍSSIMO PEREIRA

São Paulo

2023

SUMÁRIO

1. RESUMO

Este documento tem como objetivo apresentar o ocorrido da recursividade na Árvore Binaria, recentemente, foi ensinado em aula, como fazer uma Árvore Binaria, e nada mais é, do que uma lista encadeada sobre um modelo de uma hierarquia não linear, construída com **nós** utilizando a recursividade, e cada nó contém um número inteiro e duas referências para os filhos esquerdo e direito. A recursividade é usada para criar e percorrer os nós da árvore, sendo os números em ordem e o nó faz relação de parentesco (pai e filho), verificando se o número raiz contém zero ou mais sub-árvores, sendo as filhas da raiz, caso o nó tivesse filhos, autenticava como nós internos. Se por acaso, o nó não contém filhos, autenticava como nós externos ou folhas. Ou seja, entra um número na árvore, se a raiz tiver um número menor que o filho, ele iria para a direita da árvore, se o número for menor, iria para a esquerda e, até ela não tiver mais filhos, chegando nos nós externos. O final do resultado, irá mostrar a ordem dos números, os números maiores sempre vão para a direita, e os menos, para a esquerda.

2. Argumentação Teórica

2.1. Modelo de pesquisa

A busca em ordem é usada no algoritmo principalmente por duas razões fundamentais:

A busca da Ordenação dos Elementos garante que os elementos de busca sejam visitados e impressos em ordem crescente. Isso é uma característica essencial da busca, onde os valores menores estão sempre à esquerda e os valores maiores estão sempre à direita. A Recursão Natural busca aproveita essa propriedade, permitindo que os valores sejam processados em ordem.

A Recursão Natural se alinha naturalmente com a recursão. A recursão é uma técnica eficaz para percorrer estruturas de dados hierárquicas como árvores. Quando você está percorrendo uma árvore binária de busca, a busca em ordem permite que você processe os nós da árvore em uma sequência lógica: primeiro o nó mais à esquerda, depois o nó atual e, por fim, o nó mais à direita. Essa sequência é facilmente expressa de forma recursiva, pois você pode aplicar a mesma lógica a cada subárvore.

Portanto, usar a busca em ordem é uma escolha natural quando você deseja percorrer uma árvore binária de busca para imprimir seus elementos em ordem crescente. Ela garante que os elementos sejam processados de acordo com a estrutura da árvore e permite que você obtenha uma lista ordenada dos valores contidos na árvore de maneira eficiente, sem a necessidade de ordenação adicional. Isso é especialmente valioso quando se lida com grandes conjuntos de dados ou quando se deseja manter a ordenação em tempo real à medida que novos elementos são inseridos na árvore.

2.2. Recursividade

A estrutura do código e seus conceitos consiste em:

1.A Classe `no` contém um número inteiro e duas referências para os filhos esquerdo e direito. E é usada para criar e percorrer os nós da árvore.

2. A Classe `Arvore Binaria` é a classe principal. Ela possui um campo privado chamado `raiz`, que é a referência para o nó raiz da árvore.
3. O Método `inserir` é um método privado usado para inserir um novo valor. A recursividade é usada aqui para percorrer a árvore e encontrar o local adequado para inserir o novo valor. O método verifica se o valor a ser inserido é menor ou maior do que o valor no nó atual e, com base nessa comparação, move-se para a subárvore esquerda ou direita, chamando recursivamente o método `inserir`.
4. A Método `mostrar` é um método privado que realiza uma travessia em ordem da árvore binária, ou seja, imprime os valores dos nós em ordem crescente. A recursividade é usada aqui para percorrer a árvore, primeiro movendo-se para o filho esquerdo, depois imprimindo o valor do nó atual e finalmente movendo-se para o filho direito.
5. O Método `main` é o principal para iniciar a execução do programa. Ele cria uma instância da classe e lê valores inteiros do usuário e os insere na árvore binária usando o método `inserir`. A recursividade entra em ação durante a inserção e também durante a exibição dos valores da árvore usando o método `mostrar`.

2.3. Árvore binária

A Árvore Binaria é uma lista ordenada de uma hierarquia, usando nós, e cada nó contém um número inteiro e duas referências para o número ir ou para a esquerda ou para a direita, e isso só é possível pois usamos a recursividade que cria e percorre nos nós fazendo relação de parentesco. O primeiro número é chamado de raiz e os outros número que vem em ordem são as filhas da raiz, caso o nó tiver filhos, ou seja, a raiz não ser zero, entra um número na árvore, se a raiz tiver um número menor que o filho, ele iria para a direita da árvore, se o número for menor, iria para a esquerda e, até ela não tiver mais filhos chamando de folha, e no final do resultado, irá mostrar a ordem dos

números, o maior que sempre vão para a direita, e os menores sempre vão para a esquerda.

3. Resultados Obtidos

3.1. CRIANDO O NÓ

Está é a classe “No”, que cria variáveis para o armazenamento dos números, ou seja, ele cria uma variável “número” do tipo int para guardar o número escolhido, cria outras duas variáveis para o número guardar ou na direita ou na esquerda do no. Além disso, cria um construtor “No” para criar um nó a partir do número (valor do nó) e após isso, declara o esquerdo e o direito nulo.

```
4   class No {
5           int numero;
6           No esquerda;
7           No direita;
8
9           public No(Integer numero) {
10                  this.numero = numero;
11                  this.esquerda = null;
12                  this.direita = null;
13          }
14
15  }
```

3.2. EXECUÇÃO DO NÓ

Neste trecho, é o método para adicionar um novo “nó” na raiz, visualizando, primeiramente, se a raiz tem alguma coisa dentro dela, caso não tenha, ela apenas coloca o número adicionado, se tiver, ele verifica se o número é maior que a raiz, caso for, ele vai faz o método inserir e o número vai para a direita da raiz, caso o número for menos, ele vai para a esquerda, retornando a raiz.

```java
public class ArvoreBinaria {
        private No raiz;

        private No inserir(No raiz, Integer numero) {
                if (raiz == null) {
                        raiz = new No(numero);

                } else {
                        if (numero < raiz.numero) {
                                raiz.esquerda = inserir(raiz.esquerda, numero);
                        } else if (numero > raiz.numero) {
                                raiz.direita = inserir(raiz.direita, numero);
                        }
                }
                return raiz;
        }

```

3.3. IMPRIME OS VALORES

Esse método “mostrar ()” é uma função recursiva que imprime os valores dos nós da árvore em ordem.

O “private void mostrar(No raiz)” é o início da definição do método "mostrar". Ele recebe um argumento "raiz", que é o nó que começa a impressão dos valores.

O “if” verifica de base para garantir que o nó "raiz" não seja nulo. Se for nulo, significa que não há mais nós para imprimir, e a recursão será interrompida.

O “mostrar(raiz.esquerda)” é chamado recursivamente com o nó à esquerda da raiz atual. Isso significa que ele irá imprimir todos os valores dos nós à esquerda do nó atual antes de imprimir o valor do próprio nó atual. E depois de imprimir o valor do nó atual e todos os valores à esquerda, o “mostrar(raiz.direita)” é chamado recursivamente com o nó à direita da raiz atual. Isso imprimirá todos os valores dos nós à direita do nó atual.

E o “System.out.print(raiz.numero + " ")” imprime o valor do nó atual (representado por "raiz.numero") seguido de um espaço em branco.

Principalmente, esse método percorre a árvore binária em ordem, imprimindo os valores dos nós em uma sequência ordenada.

```
21
22          private void mostrar(No raiz) {
23                  if (raiz != null) {
24                          mostrar(raiz.esquerda);
25                          System.out.print(raiz.numero + " ");
26                          mostrar(raiz.direita);
27                  }
28          }
29
```

3.4. PREENCHE A ÁRVORE

O “método ‘main’” cria e preenche uma árvore binária com valores fornecidos pelo usuário e, em seguida, imprime os valores da árvore em ordem.

A instancia “ArvoreBinaria arvore = new ArvoreBinaria()” que representa uma árvore binária e contém métodos para inserir valores nela.

A instancia “Scanner scanner = new Scanner(System.in)” serve para ler a entrada do usuário.

O “System.out.println(" Insira os valores na árvore (digite 'sair' para encerrar):")” é uma mensagem solicitando ao usuário que insira valores na árvore. A instrução "digite 'sair' para encerrar" indica que a inserção de valores pode ser interrompida digitando "sair".

O loop “while (true)” é usado para ler os valores de entrada do usuário até que o usuário digite "sair" para encerrar a inserção.

O “String entrada = scanner.nextLine()” lê a próxima linha de entrada do usuário e a armazena na variável entrada.

O “if (entrada.equalsIgnoreCase("sair"))” verifica se a entrada é igual a "sair" e, se for, o “ break;” interrompe o loop, encerrando a inserção de valores na árvore.

A conversão da entrada é feita pelo “Integer valor = Integer.parseInt(entrada)” por um valor inteiro. Isso presume que o usuário insira apenas valores inteiros válidos.

O método “arvore.raiz = arvore.inserir(arvore.raiz, valor)” chama o método **inserir** da classe ArvoreBinaria para inserir o valor na árvore.

A raiz é atualizada com o retorno desse método. Isso permite que a árvore cresça à medida que novos valores são inseridos. E Após, o loop de inserção, é exibida a mensagem "Sequência:" no console.
O método “arvore.mostrar(arvore.raiz)” serve para imprimir os valores da árvore em ordem correta.
O “scanner.close()” fecha o objeto Scanner para liberar os recursos de entrada.

```
30          public static void main(String[] args) {
31                  ArvoreBinaria arvore = new ArvoreBinaria();
32                  Scanner scanner = new Scanner(System.in);
33
34                  System.out.println("Insira os valores na árvore (digite 'sair' para encerrar):");
35
36                  while (true) {
37              String entrada = scanner.nextLine();
38              if (entrada.equalsIgnoreCase("sair")) {
39                  break;
40              }
41
42              Integer valor = Integer.parseInt(entrada);
43              arvore.raiz = arvore.inserir(arvore.raiz, valor);
44          }
45
46          System.out.println("Sequência:");
47          arvore.mostrar(arvore.raiz);
48
49          scanner.close();
50      }
51  }
```

3.5. CONCLUSÃO

A implementação da recursividade na Árvore Binaria no projeto utilizando a linguagem Java, trouxe benefícios significativos para o aprendizado, resultando em uma hierarquia não linear mais organizado, contendo números inteiro e duas referências para os filhos esquerdo e direito e uma melhor visualização da Árvore. Isso demonstra o potencial da recursividade usando nós da árvore em relação ao parentesco.