JOÁO SOARES JÚNIOR

RONALDO DELFINO DE OLIVEIRA FILHO

PESQUISA ESTRUTURA DE DADOS ÁRVORE

ESTRUTURA DE DADOS

ANÁLISE E DESENVOLVIMENTOS DE SISTEMAS

Ituiutaba
Julho/2022

1- Enunciado

A pesquisa deve abordar a estrutura de dados Árvore e apresentar os seguintes tópicos:

- **Definição de Árvore**
- **Conceitos básicos**
- **Tipo de árvore que a pesquisa deve abordar :**
  - **Árvore binária:**
    - **Exemplo de funcionamento da árvore binária;**
    - **Apresentar a implementação da árvore em java com exemplo de funcionamento.**

2. Definição de Árvore

Uma estrutura de dados tipo Árvore é um conjunto finito de elementos em que cada elemento é chamado nó e o primeiro elemento é chamado de raiz da árvore. Esta estrutura tem seus elementos organizados de forma hierárquica, onde existe um elemento que fica no topo da árvore, chamado de raiz e existem os elementos subordinados a ele, que são chamados de nós filhos. Cada nó filho pode conter zero, um ou mais de um nós filhos. Os nós filhos que não contém outros nós são chamados de nós folha. Para representar os nós filhos são utilizados ponteiros, por isso é considerada uma estrutura dinâmica. São portanto estruturas adequadas para representação de hierarquias.

A estruturas de dados tipo Árvore não é linear, ou seja, os elementos que as compõem não estão armazenados de forma sequencial e também não estão todos encadeados como em uma lista. Cada elemento armazena um tipo de dado e ponteiros para o elemento à esquerda e à direita, o que permite a inserção dos valores na árvore de forma recursiva.

3. Conceitos Básicos

Características de uma Árvore

1. Raiz: Toda arvore possui o nó raiz que é o nó inicial da árvore;
2. Grau: o número de filhos que um nó possui;
3. Nível (ou profundidade): a distância de um nó até a raiz;
4. Altura: o maior nível encontrado na árvore (altura do nó raiz);
5. Folha: o nó que não possui filho.

3. Árvore Binária

Uma árvore binária é uma árvore em que, abaixo de cada nó, existem no máximo duas subárvores. Considerando uma estrutura de dados Árvore com um conjunto de nós, temos que considerar que esse conjunto pode ter outros subconjuntos chamados de sub-arvores, que contem cada uma seu próprio conjunto de nós. O conjunto de nós de uma árvore pode estar vazio ou ser dividido em três subconjuntos distintos, sendo eles: 1º subconjunto (nó raiz), 2º subconjunto (sub-árvore à direita) e 3º subconjunto (sub-árvore à esquerda). Em uma árvore binária os nós podem assumir grau 0, 1 ou 2. Em uma árvore binária completa, todos os nós possuem grau igual a 2. O número máximo de elementos em uma árvore de altura n é 2n.

Como representamos computacionalmente uma árvore binária? Unindo nós. E como representamos os nós? Com 2 ponteiros: um para a subárvore da esquerda e um para a subárvore da direita. Além de um campo para a chave e dados chave.

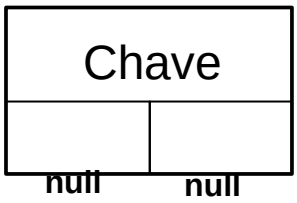

Figura 1: Representação de um nó da árvore.

Árvores binárias são largamente utilizadas em diversas aplicações. Entre as aplicações pode-se citar as árvores de decisão usadas na Inteligência Artificial. Outra aplicação é na representação de expressões aritméticas. No caso da representação das expressões aritméticas pode-se utilizar um caminhamento pós-fixado para resolver o problema, onde, por exemplo, para uma árvore binária de expressões aritméticas ter-se-ia para cada nó externo um valor associado e para cada nó interno um operador aritmético associado, esse algoritmo calcularia facilmente o resultado da expressão.

Uma árvore binária de busca possui elementos menores que a raiz armazenados na sub-árvore da esquerda e elementos maiores que a raiz na sub-árvores da direita.

3.1- Operações em uma árvore binária

1. Inserir novo elemento
2. Buscar um elemento
3. Mostrar todos os elementos
4. Remover um elemento
5. Contar elementos
6. Calcular nível de um elemento

4- Exemplo de funcionamento da árvore binária: Árvore binária de busca

Uma árvore binária é uma estrutura de dados útil quando precisam ser tomadas decisões bidirecionais em cada ponto de um processo. Por exemplo, suponha que precisemos encontrar todas as repetições numa lista de números. Uma maneira de fazer isto é comparar cada número com todos que o precedem. Entretanto, isso envolve um grande número de comparações. O número de comparações pode ser reduzido usando-se uma árvore binária. O primeiro número na lista é colocado num nó estabelecido como a raiz de uma árvore binária com as subárvores esquerda e direita vazias.

Cada número sucessivo na lista é, então, comparado ao número na raiz. Se coincidirem, teremos uma repetição. Se for menor, examinaremos a subárvore esquerda; se for maior, examinaremos a subárvore direita. Se a subárvore estiver vazia, o número não será repetido e será colocado num novo nó nesta posição na árvore. Se a subárvore não estiver vazia, compararemos o número ao conteúdo da raiz da subárvore e o processo inteiro será repetido com a subárvore.

4.1- Inserção de elementos

- O primeiro elemento inserido assumirá o papel de raiz da árvore;
- Todo novo elemento entrará na árvore como uma folha;
- Se o elemento for menor ou igual à raiz será inserido no ramo da esquerda. Caso contrário, no ramo da direita (para árvores decrescentes inverte-se a a regra).

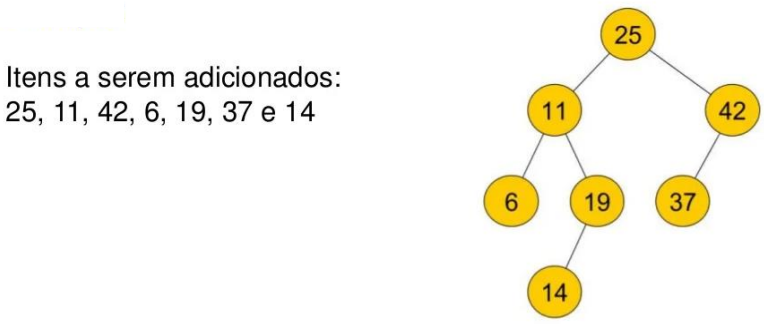

Figura 2: Exemplo de inserção

4.2- Remoção de elementos

Considerando que podemos remover qualquer elemento de uma árvore, podem ocorrer as seguintes situações:

1. O Elemento a ser removido é um nó folha (sem filhos à esquerda e à direita);

A exclusão de um nó que se encontra no fim da árvore, isto é, na folha, é o caso mais simples de exclusão. Basta remover o nó da árvore.

2. O Elemento a ser removido possui apenas um filho (à direita ou à esquerda);

Caso o nó seja excluído, o “avô” herda o “filho”.

3. O Elemento a ser removido possui dois filhos.

Neste caso o processo poderá ocorrer de duas formas:

- Substituir o nó a ser retirado pelo valor sucessor (o nó mais à esquerda da sub-árvore direita);

- Substituir o nó a ser retirado pelo valor antecessor (o nó mais à direita da sub-árvore esquerda);

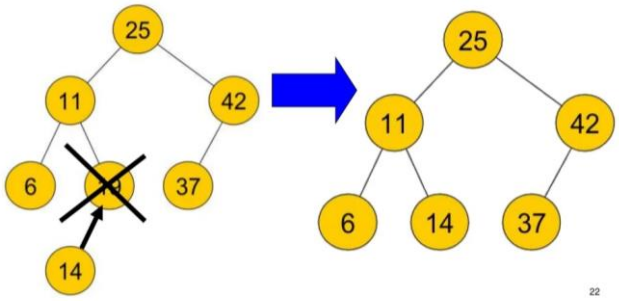

Figura 3: Exemplo de remoção de nó com filho.

3.4- Percurso em uma árvore binária

1. Pré-ordem: raiz, sub-árvore esquerda, sub-árvore direita

2. Em ordem: sub-árvore esquerda, raiz, sub-árvore direita

3. Pós-ordem: sub-árvore esquerda, sub-árvore direita, raiz

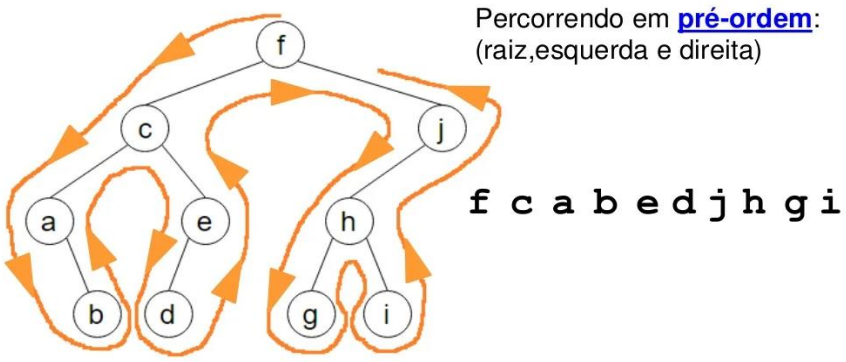

Figura 4: Exemplo de inserção

5. Implementação da árvore em java com exemplo de funcionamento

Tomando como exemplo a sequência de elementos numéricos:

[10, 8, 5, 9, 7, 18, 13, 20]

A inserção dos elementos na árvore ocorrerá de acordo com a figura abaixo.

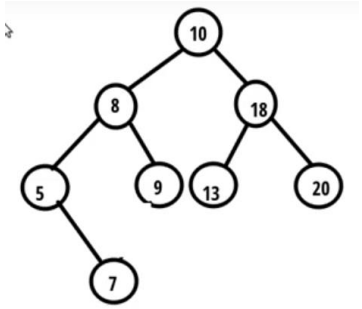

Figura 5: Exemplo de inserção de elementos numéricos.

Como exemplo de implementação foi utilizado um programa em Java com os métodos para inserir, remover, buscar e exibir a árvore em ordem, pós-ordem e pré-ordem. Ao serem inseridos os elementos da sequência acima, foi selecionado a opção de exibir, obtendo-se os resultados abaixo:

- Exibindo em ordem: 5 7 8 9 10 13 18 20
- Exibindo em pos-ordem: 7 5 9 8 13 20 18 10
- Exibindo em pre-ordem: 10 8 5 7 9 18 13 20
- Altura da arvore: 3
- Quantidade de folhas: 4
- Quantidade de Nós: 8
- Valor mínimo: 5
- Valor máximo: 20

A seguir apresenta-se o código do programa “EstruturaArvore” supracitado (adaptação do código do programa citado na referência 7).

- Classe Tree:

```
public class Tree {
    private No root; // raiz

    public Tree() { root=null; } // inicializa arvore

    public void inserir(long v) {
        No novo = new No(); // cria um novo Nó
        novo.item = v; // atribui o valor recebido ao item de dados do Nó
        novo.dir = null;
        novo.esq = null;

        if (root == null) root = novo;
        else  { // se nao for a raiz
            No atual = root;
            No anterior;
            while(true) {
                anterior = atual;
                if (v <= atual.item) { // ir para esquerda
                    atual = atual.esq;
                    if (atual == null) {
                        anterior.esq = novo;
                        return;
                    }
                }  // fim da condição ir a esquerda
                else { // ir para direita
                    atual = atual.dir;
                    if (atual == null) {
                        anterior.dir = novo;
                        return;
                    }
                } // fim da condição ir a direita
            } // fim do laço while
```

```
    } // fim do else não raiz
}

public No buscar(long chave) {
    if (root == null) return null; // se arvore vazia
    No atual = root;  // começa a procurar desde raiz
    while (atual.item != chave) { // enquanto nao encontrou
        if(chave < atual.item ) atual = atual.esq; // caminha para esquerda
        else atual = atual.dir; // caminha para direita
        if (atual == null) return null; // encontrou uma folha -> sai
    } // fim laço while
    return atual; // terminou o laço while e chegou aqui é pq encontrou item
}

public boolean remover(long v) {
    if (root == null) return false; // se arvore vazia

    No atual = root;
    No pai = root;
    boolean filho_esq = true;

    // ****** Buscando o valor **********
    while (atual.item != v) { // enquanto nao encontrou
        pai = atual;
        if(v < atual.item ) { // caminha para esquerda
            atual = atual.esq;
            filho_esq = true; // é filho a esquerda? sim
        }
        else { // caminha para direita
            atual = atual.dir;
            filho_esq = false; // é filho a esquerda? NAO
        }
        if (atual == null) return false; // encontrou uma folha -> sai
    } // fim laço while de busca do valor
```

```
// ************************************************************
// se chegou aqui quer dizer que encontrou o valor (v)
// "atual": contem a referencia ao No a ser eliminado
// "pai": contem a referencia para o pai do No a ser eliminado
// "filho_esq": é verdadeiro se atual é filho a esquerda do pai
// ************************************************************

// Se nao possui nenhum filho (é uma folha), elimine-o
if (atual.esq == null && atual.dir == null) {
    if (atual == root ) root = null; // se raiz
    else if (filho_esq) pai.esq = null; // se for filho a esquerda do pai
    else pai.dir = null; // se for filho a direita do pai
}

// Se é pai e nao possui um filho a direita, substitui pela subarvore a direita
else if (atual.dir == null) {
    if (atual == root) root = atual.esq; // se raiz
    else if (filho_esq) pai.esq = atual.esq; // se for filho a esquerda do pai
    else pai.dir = atual.esq; // se for filho a direita do pai
}

// Se é pai e nao possui um filho a esquerda, substitui pela subarvore a esquerda
else if (atual.esq == null) {
    if (atual == root) root = atual.dir; // se raiz
    else if (filho_esq) pai.esq = atual.dir; // se for filho a esquerda do pai
    else pai.dir = atual.dir; // se for  filho a direita do pai
}

// Se possui mais de um filho, se for um avô ou outro grau maior de parentesco
else {
    No sucessor = no_sucessor(atual);
    // Usando sucessor que seria o Nó mais a esquerda da subarvore a direita do No que deseja-se remover
    if (atual == root) root = sucessor; // se raiz
```

```
            else if(filho_esq) pai.esq = sucessor; // se for filho a esquerda do pai
            else pai.dir = sucessor; // se for filho a direita do pai
            sucessor.esq = atual.esq; // acertando o ponteiro a esquerda do sucessor agora que ele assumiu
            // a posição correta na arvore
        }

        return true;
    }

    // O sucessor é o Nó mais a esquerda da subarvore a direita do No que foi passado como parametro do metodo
    public No no_sucessor(No apaga) { // O parametro é a referencia para o No que deseja-se apagar
        No paidosucessor = apaga;
        No sucessor = apaga;
        No atual = apaga.dir; // vai para a subarvore a direita

        while (atual != null) { // enquanto nao chegar no Nó mais a esquerda
            paidosucessor = sucessor;
            sucessor = atual;
            atual = atual.esq; // caminha para a esquerda
        }
        // ********************************************************************************
        // quando sair do while "sucessor" será o Nó mais a esquerda da subarvore a direita
        // "paidosucessor" será o o pai de sucessor e "apaga" o Nó que deverá ser eliminado
        // ********************************************************************************
        if (sucessor != apaga.dir) { // se sucessor nao é o filho a direita do Nó que deverá ser eliminado
            paidosucessor.esq = sucessor.dir; // pai herda os filhos do sucessor que sempre serão a direita
            // lembrando que o sucessor nunca poderá ter filhos a esquerda, pois, ele sempre será o
            // Nó mais a esquerda da subarvore a direita do Nó apaga.
            // lembrando também que sucessor sempre será o filho a esquerda do pai

            sucessor.dir = apaga.dir; // guardando a referencia a direita do sucessor para
            // quando ele assumir a posição correta na arvore
        }
```

```
    return sucessor;
}

public void caminhar() {
    System.out.print("\n Exibindo em ordem: ");
    inOrder(root);
    System.out.print("\n Exibindo em pos-ordem: ");
    posOrder(root);
    System.out.print("\n Exibindo em pre-ordem: ");
    preOrder(root);
    System.out.print("\n Altura da arvore: " + altura(root));
    System.out.print("\n Quantidade de folhas: " + folhas(root));
    System.out.print("\n Quantidade de Nós: " + contarNos(root));
    if (root != null ) {  // se arvore nao esta vazia
        System.out.print("\n Valor minimo: " + min().item);
        System.out.println("\n Valor maximo: " + max().item);
    }
}

public void inOrder(No atual) {
    if (atual != null) {
        inOrder(atual.esq);
        System.out.print(atual.item + " ");
        inOrder(atual.dir);
    }
}

public void preOrder(No atual) {
    if (atual != null) {
        System.out.print(atual.item + " ");
        preOrder(atual.esq);
        preOrder(atual.dir);
    }
}
```

```
public void posOrder(No atual) {
    if (atual != null) {
        posOrder(atual.esq);
        posOrder(atual.dir);
        System.out.print(atual.item + " ");
    }
}

public int altura(No atual) {
    if(atual == null || (atual.esq == null && atual.dir == null))
        return 0;
    else {
        if (altura(atual.esq) > altura(atual.dir))
            return ( 1 + altura(atual.esq) );
        else
            return ( 1 + altura(atual.dir) );
    }
}

public int folhas(No atual) {
    if(atual == null) return 0;
    if(atual.esq == null && atual.dir == null) return 1;
    return folhas(atual.esq) + folhas(atual.dir);
}

public int contarNos(No atual) {
    if(atual == null)  return 0;
    else return ( 1 + contarNos(atual.esq) + contarNos(atual.dir));
}

public No min() {
    No atual = root;
    No anterior = null;
```

```
        while (atual != null) {
            anterior = atual;
            atual = atual.esq;
        }
        return anterior;
    }

    public No max() {
        No atual = root;
        No anterior = null;
        while (atual != null) {
            anterior = atual;
            atual = atual.dir;
        }
        return anterior;
    }

}
```

- Classe No:

```
public class No {
    public long item;
    public No dir;
    public No esq;}
```

- Classe Principal: ArvoreBinaria-Main

```
 import java.util.*;

public class ArvoreBinaria_Main {
    public static void main(String[] args) {
        Scanner le = new Scanner(System.in);
        Tree arv = new Tree();
```

```
int opcao;
long x;
System.out.print("\n Programa Arvore binaria de long");
do {
   System.out.print("\n************************************");
   System.out.print("\nEntre com a opcao:");
   System.out.print("\n ----1: Inserir");
   System.out.print("\n ----2: Excluir");
   System.out.print("\n ----3: Pesquisar");
   System.out.print("\n ----4: Exibir");
   System.out.print("\n ----5: Sair do programa");
   System.out.print("\n************************************");
   System.out.print("\n-> ");
   opcao = le.nextInt();
   switch(opcao) {
      case 1: {
         System.out.print("\n Informe o valor (long) -> ");
         x = le.nextLong();
         arv.inserir(x);
         break;
      }
      case 2: {
         System.out.print("\n Informe o valor (long) -> ");
         x = le.nextLong();
         if ( !arv.remover(x) )
            System.out.print("\n Valor nao encontrado!");;
         break;
      }
      case 3: {
         System.out.print("\n Informe o valor (long) -> ");
         x = le.nextLong();
         if( arv.buscar(x) != null )
            System.out.print("\n Valor Encontrado");
         else
```

```
                    System.out.print("\n Valor nao encontrado!");
                    break;
                }
                case 4: {
                    arv.caminhar();
                    break;
                }
            } // fim switch
        } while(opcao != 5);

    }
}
```

6- Referências

1. Apresentação:

   Árvores: Conceitos Básicos, Aula 15 Estrutura de Dados

   Norton T. Roman & Luciano A. Digiampietri

   http://each.uspnet.usp.br/digiampietri/ACH2023/va15.pdf

2. Página Web:

   Aula 11 – Árvores, Estrutura de Dados

   Prof. Fernando De Siqueira

   https://sites.google.com/site/proffdesiqueiraed/aulas/aula-10---arvores

3. Apresentação:

   Árvores Binárias – Aplicações

   Marcelo Linder - Univasf

   http://www.univasf.edu.br/~marcelo.linder/arquivos_aed1C2/material_semestre_passado/aula24.pdf

4. Página web:

   Trabalhando com árvores binárias em Java

   https://www.devmedia.com.br/trabalhando-com-arvores-binarias-em-java/25749

5. Video-aula Youtube:

   Aula 22 - Em-ordem, Pré-ordem e Pós-ordem - Estruturas de Dados com Java

   Leandro Guarino

   https://www.youtube.com/watch?v=4Bl2Xhd8hOA

6. Apresentação:

   Estruturas de dados Árvores – Aula 13, Parte 1

   Divani Barbosa

   https://www.slideshare.net/DivaniBarbosaGavinie/aula-18-82478778?from_action=save

7. Programa:

   ArvoreBinariaApp.java

Autora: Divani Barbosa

https://gist.github.com/divanibarbosa/819e7cfcf1b9bae48c4e0f5bd74fb658