Lista - Árvore Binária

Rodolfo Oliveira Miranda

1. Múltiplas respostas

Nós folhas: C, J, P e T
Grau da árvore: 2
Altura da árvore: 2
Descendente do nó R: P e T

---

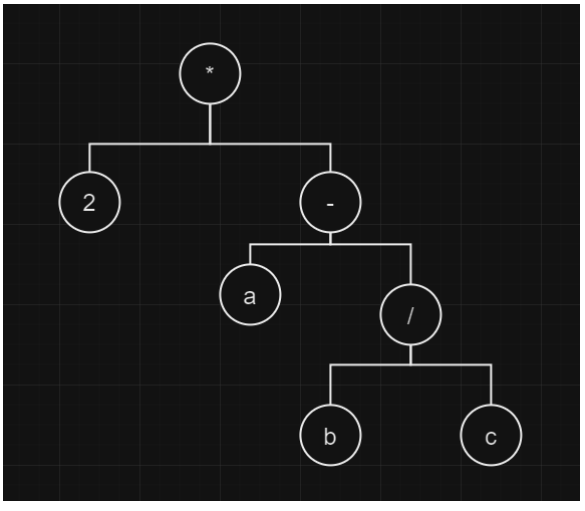
2. Usando a seguinte fórmula é possível determinar que podem existir 5 árvores binárias com 3 nós.

$$(1)C_n = \frac{1}{n+1}\binom{2n}{n} \quad (2)C_3 = \frac{1}{3+1}\binom{6}{3} \quad (3)C_3 = \frac{1}{4} \times \frac{6!}{3! \times 3!}$$

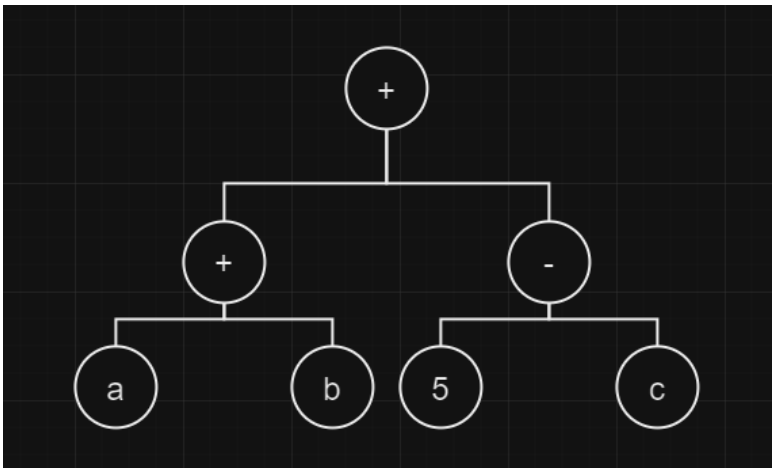
$$(4)C_3 = \frac{1}{4} \times \frac{720}{36} \quad (5)C_3 = \frac{1}{4} \times 20 \quad (6)C_3 = 5$$

---

3. Árvores de expressões aritméticas
    a. 2 * (a - b/c)

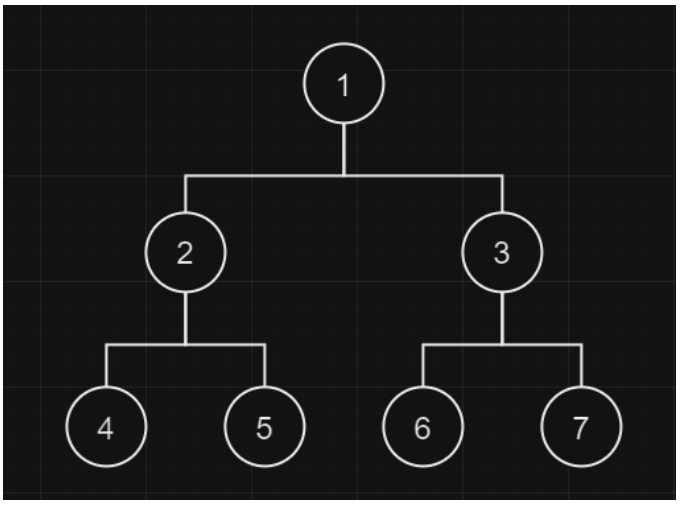


b. a + b + 5 * c



---

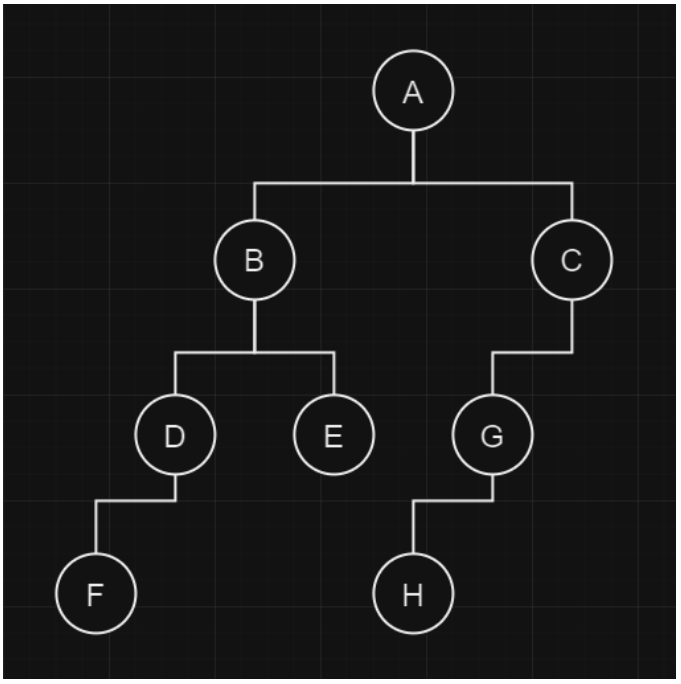
4. Representação das seguintes árvores
   a. (1 (2 (4) (5)) (3 (6) (7)))



Esta árvore está perfeitamente balanceada, pois todas as folhas estão no mesmo nível e todos os nós pais destas têm dois filhos.

i. Pre-Order: 1, 2, 4, 5, 3, 6, 7
ii. In-Order: 4, 2, 5, 1, 3, 6, 7
iii. Post-Order: 4, 5, 2, 6, 7, 3, 1

b. (A (B (D (F)) (E)) (C (G (H))))



Esta árvore não está balanceada, pois uma de suas subárvores (C) tem diferença de altura igual a 2.

i. Pre-Order: A, B, D, F, E, C, G, H

ii. In-Order: F, D, B, E, A, H, G, C

iii. Post-Order: F, D, E, B, H, G, C, A

---

5. Comparação entre códigos de Pre-Order

   a. Usando Pilha

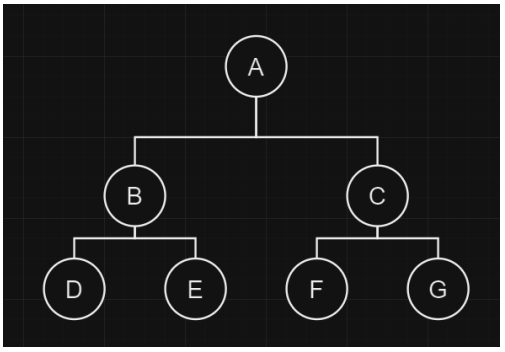
```
private void showPreOrderWithStack(BinNode<T> root) {
    Stack<BinNode<T>> stack = new Stack<>();
    stack.push(new StackNode<BinNode<T>>(root));
    StackNode<BinNode<T>> current;
    while (!stack.isEmpty()) {
        current = stack.pop();
        System.out.print(current.getNodeData().getBinNodeData() + ", ");
        if (current.getNodeData().getRightNode() != null)
            stack.push(new StackNode<BinNode<T>>(current.getNodeData().getRightNode()));
        if (current.getNodeData().getLeftNode() != null)
            stack.push(new StackNode<BinNode<T>>(current.getNodeData().getLeftNode()));
    }
    System.out.println("\n");
}
```

   b. Usando Lista Encadeada

```java
private void showPreOrderWithLinkedList(BinNode<T> root) {
    LinkedList<BinNode<T>> linkedList = new LinkedList<>();
    linkedList.add(root);

    LinkedListNode<BinNode<T>> current;

    while (!linkedList.isEmpty()) {
        current = linkedList.removeHead();
        System.out.print(current.getData().getBinNodeData() + ", ");
        if (current.getData().getLeftNode() != null)
            linkedList.add(current.getData().getLeftNode());
        if (current.getData().getRightNode() != null)
            linkedList.add(current.getData().getRightNode());

    }
    System.out.println("\n");
}
```

Apesar dos ajustes feitos, no geral, não notei nenhuma vantagem ou desvantagem no uso da lista encadeada ao invés da pilha.

6.



Pre-Order: A, B, D, E, C, F, G
In-Order: D, B, E, A, F, C, G

O primeiro elemento do pre-order é a raiz, logo (A).

No in-order, todos os elementos a esquerda de (A) são da subárvore a esquerda (D, B e E), e consequentemente, os da direita são da subárvore a direita (F, C e G).

Pode-se repetir estes passos com as subárvores.

Pre-Order: B, D, E

In-Order: D, B, E
Raiz: B
Esquerda: D
Direita: E

Pre-Order: C, F, G
In-Order: F, C, G
Raiz: C
Esquerda: F
Direita: G

7.
   Pre-Order: 8, 9, 11, 15, 19, 20, 21, 7, 3, 2, 1, 5, 6, 4, 13, 14, 10, 12, 16, 12, 18
   In-Order: 19, 15, 11, 20, 21, 9, 2, 3, 7, 1, 8, 13, 4, 6, 14, 10, 5, 16, 17, 12, 18
   Post-Order: 19, 15, 21, 20, 11, 2, 3, 1, 7, 9, 13, 4, 10, 14, 6, 16, 17, 18, 12, 5, 8

---

8. É possível perceber que a raiz da árvore binária sempre ficara no início da pré-ordem e no final da pós-ordem.

---

9. Remover nós com datas anteriores
   a. Nó do Arquivo

```java
public class FileNode<T> implements Comparable<FileNode<T>> {

    3 usages
    private String name;
    4 usages
    private T lastAccessDate;
    3 usages
    private FileNode<T> leftNode;
    3 usages
    private FileNode<T> rightNode;
    3 usages
    private FileNode<T> parent;
```

b. Árvore do Catálogo

```java
public class CatalogTree<T extends Comparable<T>> {

    8 usages
    private FileNode<T> root;

    1 usage   new *
    public CatalogTree() { this.root = null; }

    no usages   new *
    public FileNode<T> getRoot() { return this.root; }
```

c. Método para remover arquivos anteriores

```java
public void removePreviousFiles(T date) {
    removePreviousFiles(this.root, date);
}

3 usages   new *
private FileNode<T> removePreviousFiles(FileNode<T> current, T date) {
    if (current.getLastAccessDate().compareTo(date) > 0) {
        return removePreviousFiles(current.getLeftNode(), date);
    }
    if (current.getLastAccessDate().compareTo(date) < 0) {
        this.root = current.getRightNode();
        current = this.root;
        return removePreviousFiles(current, date);
    }
    current.setLeftNode(null);
    return current;
}
```

10. Reorganizar árvore de acordo com frequência de acesso dos nós
    a. Método para transformar a árvore em lista

```java
private BinNode<T> buildTreeFromList(List<BinNode<T>> nodesList, int start, int end) {
    if (start > end) {
        return null;
    }

    int mid = (start + end) / 2;

    BinNode<T> node = nodesList.get(mid);

    node.setLeftNode(buildTreeFromList(nodesList, start, end: mid - 1));
    node.setRightNode(buildTreeFromList(nodesList, start: mid + 1, end));

    return node;
}
```

b. Método para ordenar lista de acordo com a frequência

```java
public void orderNodesByFrequency(List<BinNode<T>> nodesList) {
    for (int i = 0; i < nodesList.size() - 1; i++) {
        for (int j = i + 1; j < nodesList.size(); j++) {
            if (nodesList.get(i).getFrequency().compareTo(nodesList.get(j).getFrequency()) < 0) {
                BinNode<T> temp = nodesList.get(i);
                nodesList.set(i, nodesList.get(j));
                nodesList.set(j, temp);
            }
        }
    }
}
```

c. Método para reorganizar árvore

```java
public void reorganize() {
    List<BinNode<T>> nodesList = new ArrayList<>();
    this.treeToList(this.root, nodesList);
    orderNodesByFrequency(nodesList);

    this.root = null;
    for (BinNode<T> node : nodesList) {
        insert(node.getKey(), node.getFrequency());
    }
}
```