Curso de bacharelado em Ciência da Computação
Estrtura de dados
Prof. Otávio Alcântra

Nome: Henrique Fernandes Tavares da Cunha, Emannuel Levi de Assis Bezerra e Raimundo Rafael
Data: 08/12/2024

Lista de exercícios

1. **O que é uma árvore binária própria?**
   É uma árvore aonde cada nó que não é folha tem 2 filhos.

2. **O que é uma árvore binária cheia?**
   É uma árvore aonde todos os nós folha estão no último nível.

3. **O que é uma árvore binária completa?**
   É uma árvore aonde os nós folhas ficam no último ou penúltimo nível.

4. **Qual é a altura máxima de uma árvore binária com n nós?**
   n - 1, pois caso todos os nós estejam em sequência, tal qual uma lista encadeada, a altura da árvore chegará em n - 1.

5. **Qual é a altura mínima de uma árvore binária com n nós? Dica: procure sobre árvore perfeitamente balanceada.**
   log 2, pois a altura só vai aumentar quando a árvore alcançar um valor pertencente a 2^n, onde n é inteiro. Nos outros casos ela irá permanecer na última altura alcançada.

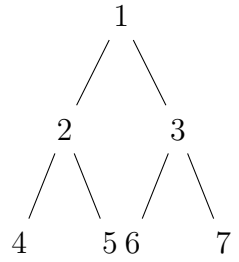
6. **Qual é o número máximo e mínimo de nós internos e externos de uma árvore binária imprópria?**

Tipo	Máximo	Mínimo
Interno	n/2	1
Externo	n - 1	n/2

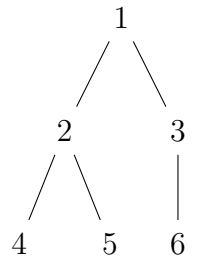
7. **Qual é o número mínimo de nós externos de uma árvore binária própria de altura h?**
   h + 1

8. **Qual é o número máximo de nós externos de uma árvore binária própria de altura h?**
   2^h

9. **O que é uma árvore binária balanceada? Desenhe um exemplo.**
   É uma árvore onde a altura dos nós filhos não pode diferir num valor tal que $-1 < valor < 1$.



10. **O que é uma árvore binária perfeitamente balanceada? Desenhe um exemplo.**
    É uma árvore que em todos os nós a diferença entre o número de nós da árvore esquerda e direita é ≤ 1.



11. **Escreva algoritmos recursivos e não -recursivos para determinar:**

    a. **O número de nós de uma árvore binária**
    Recursivo:

```
def num_nodes(r):
    if r is None:
        return 0
    return 1 + num_nodes(r.left) +
       num_nodes(r.right)
```

    Não recursivo:

```
def num_nodes(r):
    count = 0
    node = r
    pilha = Pilha()
    while node:
        count += 1
        if node.right is not None:
            pilha.push(node)
        if node.left is None:
            if not pilha.is_empty():
                node = pilha.pop()
            node = node.right
            continue
        node = node.left
    return count
```

    b. **a soma do conteúdo de todos os nós de uma árvore binária**
    Recursivo:

```
def sum_nodes(r):
    if r is None:
        return 0
    return r.data + sum_nodes(r.left) +
       sum_nodes(r.right)
```

Não recursivo:

```
def sum_nodes(r):
    count = 0
    node = r
    pilha = Pilha()
    while node:
        count += node.data
        if node.right is not None:
            pilha.push(node)
        if node.left is None:
            if not pilha.is_empty():
                node = pilha.pop()
            node = node.right
            continue
        node = node.left
    return count
```

c. **O nível com maior soma de uma árvore binária**
Recursivo:

```
def sum_level(r):
    def dfs(node, sums_levels, level=1):
        if node is None:
            return

        if not sums_levels.get(level):
            sums_levels[level] = 0

        sums_levels[level] += node.data

        dfs(node.left, level + 1,
           sums_levels)
        dfs(node.right, level + 1,
           sums_levels)

    sums_levels = {}
    dfs(r, sums_levels)

```

```
    biggest_num = max(sums_levels.values
        ())
    for k, v in sums_levels.items():
        if v == biggest_num:
            return k
    return 0
```

Não recursivo:

```
def sum_level(r):
    sums_levels = {}
    level = 0
    node = r
    pilha = Pilha()
    while node:
        level += 1
        if not sums_levels.get(level):
            sums_level[level] = 0
        sums_level[level] += node.data
        level += 1
        if node.right is not None:
            pilha.push(node)
        if node.left is None:
            if not pilha.is_empty():
                node = pilha.pop()
                level -= 1
            node = node.right
            continue
        if not pilha.is_empty():
            level -= 1
        node = node.left
    biggest_num = max(sums_levels.values
        ())
    for k, v in sums_levels.items():
        if v == biggest_num:
            return k
    return 0
```

d. **a altura de uma árvore binária.**
Recursivo:

```
def height(r):
    if r.right is None and r.left is
       None:
        return 0
    return 1 + max(height(r.left),
       height(r.right))
```

Não recursivo:

```
def height(r):
    if r is None:
        return 0

    queue = deque([(r, 1)])
    max_height = 0

    while queue:
        node, level = queue.popleft()
        max_height = max(max_height,
           level)

        if node.left:
            queue.append((node.left,
               level + 1))
        if node.right:
            queue.append((node.right,
               level + 1))

    return max_height
```

e. **a profundidade de uma árvore binária.**
Recursivo:

```
def deepness(r):
    if r.right is None and r.left is
       None:
        return 1
```

```
    return 1 + max(deepness(r.left),
       deepness(r.right))
```

Não recursivo:

```
def deepness(r):
    if r is None:
        return 0

    queue = deque([(r, 1)])
    max_deepness = 1

    while queue:
        node, level = queue.popleft()
        max_deepness = max(max_deepness,
            level)

        if node.left:
            queue.append((node.left,
               level + 1))
        if node.right:
            queue.append((node.right,
               level + 1))

    return max_height
```

12. **Escreva um algoritmo para determinar se uma árvore binária é:**

    a. **Própria**

```
def isPropria(r):
    if r is None:
        return True
    if r.right is None and r.left is not
        None:
        return false
```

```
    if r.right is not None and r.left is
        None:
        return false
    return isPropria(r.left) and
       isPropria(r.right)
```

b. **Completa**

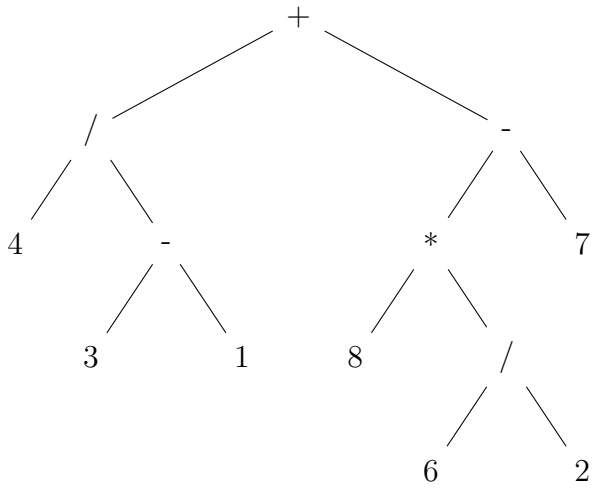
```
def isCompleta(root):
    if not root:
        return True

    queue = Queue()
    queue.enqueue(root)
    found_empty = False

    while queue:
        node = queue.dequeue()

        if not node:
            found_empty = True
        else:
            if found_empty:
                return False
            queue.append(node.left)
            queue.append(node.right)

    return True
```

c. **Quase Completa**

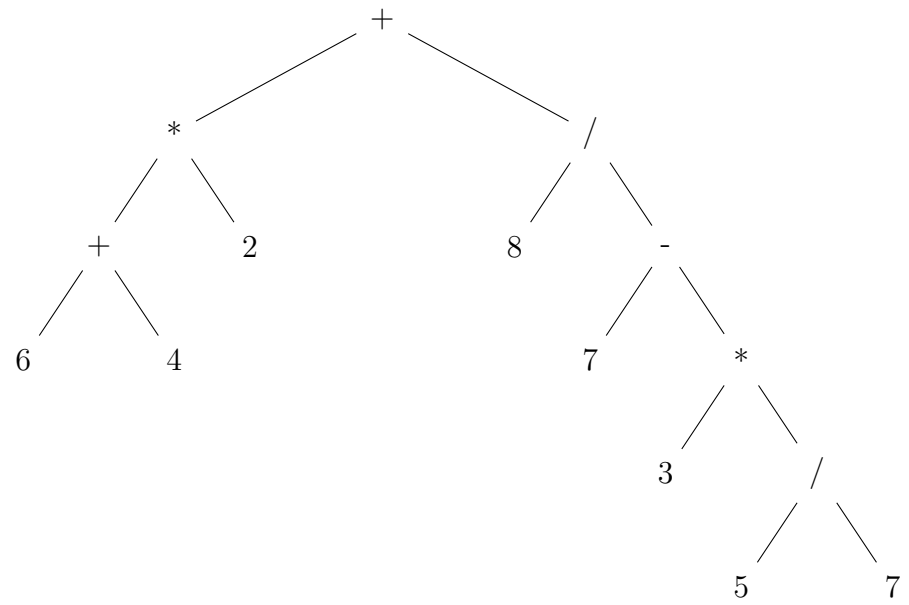
```
def isQuaseCompleta(root):
    if not root:
        return True

    queue = Queue()
    queue.enqueue(root)
```

```
    found_empty = False

    while queue:
        node = queue.dequeue()

        if not node:
            found_empty = True
        else:
            if found_empty:
                return False
            queue.append(node.left)
            queue.append(node.right)

    return True
```

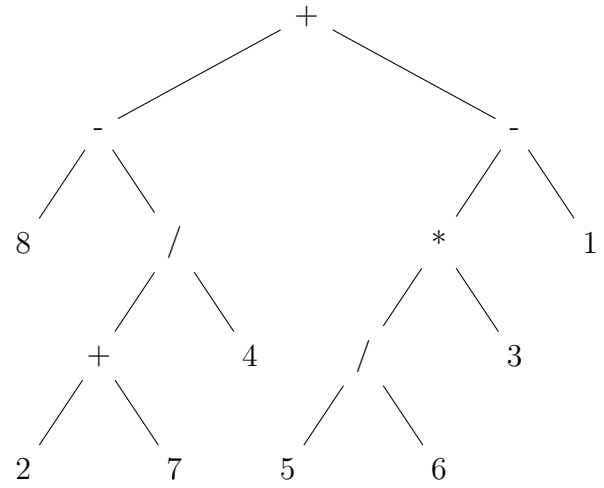
13. **Desenhe a árvore binária correspondente à expressão prefixada abaixo. Todos os operadores são binários.**
+ / 4 - 3 1 - * 8 / 6 2 7



14. **Desenhe a árvore binária correspondente à expressão posfixa abaixo. Todos os operadores são binários.**
5 7 / 3 * 1 - 8 2 4 6 + / * +



15. **Desenhe a árvore binária correspondente à expressão infixa abaixo. Todos os operadores são binários.**
(8 -((2+7)/4) +(((5 / 6)*3)-1)



16. **Seja:**
    a. **G uma árvore binária genérica não vazia, P uma árvore binária própria não vazia.**
    b.

	Árvore G	Árvore P
Número de Nós externos	e_g	e_p
Número de nós internos	i_g	i_p
Número total de nós	n_g	n_p
Altura	h_g	h_p

c. **Verdadeiro ou Falso**

i. $2h_p + 1 \leq n_p$ (V)

ii. $h_p + 1 \leq e_p$ (V)

iii. $n_g = 2e_g - 1$ (F)

iv. $1 \leq e_g \leq 2h_g$ (V)

v. $h_g \leq i_g \leq 2h_g - 1$ (V)