Explicação deleta Binaria

domingo, 4 de maio de 2025 17:06

REMOVER FOLHA

```
private void removeFolha(Nodo aux, Nodo pai) {
        if (pai == null) {
            setRaiz(null);
            // o pai.getEsquerda é a mesma coisa que o aux, visto que o pai é
sempre um antes
        } else if (pai.getEsquerda() == aux) {
            pai.setEsquerda(null);
        } else if (pai.getDireita() == aux) {
            pai.setDireita(null);
        }
    }
```

O que acontece aqui?

1° se caso o pai é null então é uma raiz única.
2° se o pai.getEsquerda == aux, então está assim:

```
Pensando na esquerda
    8 <- pai
    /
4 -> aux
```

Quero remover o 4, ele é só uma folha, então fazemos:

A gente desvincula o 4 com set null de forma simples, já que não tem problema com filhos e herdeiros.

```
Pensando na direita
    8 <- pai
    / \
  4    9
```

Se quero remover o 9?
Então vamos fazer pai.setDireita(NULL);
O pai corta a \ do 9 e ele se desvincula, sem problemas, pois ele não tem herdeiros e nem filhos.

REMOVER esquerda

```
private void removeComFilhoEsquerda(Nodo aux, Nodo pai) {
        if (pai == null) {
            setRaiz(aux.getEsquerda());
        } else if (pai.getEsquerda() == aux) {
         //ideia "padrao" se filho esquerda é igual aux pai tem que cortar
esquerda
```

```
                pai.setEsquerda(aux.getEsquerda());
            }
            else if (pai.getDireita() == aux) {
             // o filho da esquerda do no vira a direita
                pai.setDireita(aux.getEsquerda());
            }
        }
```

Agora vamos pensar no sentido em que queremos em especifico remover o filho da esquerda.

1° coisa sempre tem que ver se é null.

```
   8
  /  \
4     1
```

Se eu quero tirar o 8 e o filho vira o 4.
setRaiz(aux.getEsquerda);

2° pai.setEsquerda(aux.getEsquerda)
O que está acontecendo?

```
        10 pai
        /
      2   aux
     /
   1
```

Então vou remover o 2 pai.setEsquerda(aux.getEsquerda()); e colocar o 1 no lugar dele.

```
        10 pai
        /
      1
```

3° (pai.getDireita() == aux) o que vamos fazer?

```
          8 -> pai
           \
           10 -> aux
           /
         5
```

Vamos traduzir o código
pai.setDireita(aux.getEsquerda());

RemoveEsq(aux = 10, pai = 8)
paiSetDireita(5)

Estou realocando o filho para a esquerda.

```
      8
     /  \
        5
```

Portanto o filho da esquerda do Nodo virou a Direita.

REMOVER direita

```
private void removeComFilhoDireita(Nodo aux, Nodo pai) {
        if (pai == null) {
            setRaiz(aux.getDireita());
        } else if (pai.getEsquerda() == aux) {
            pai.setEsquerda(aux.getDireita());
        } else if (pai.getDireita() == aux) {
            pai.setDireita(aux.getDireita());
        }
    }
```

Seria o mesmo esquema?
- Sim, mas vamos revisar para não ter problemas

Se quero remover na condição
`if (pai == null)`
Significa que:

```
Pai = null
10 aux
  \
    6
```

Agora se `(pai.getEsquerda() == aux)`

Vamos pensar o seguinte:

```
    10 -> pai
   /
 5   -> aux
  \
   6
```

Vamos fazer: `pai.setEsquerda(aux.getDireita())` por que?
Ele vai puxar o 6 para o local do 5

```
10 -> pai
    /
   6
```

```
else if (pai.getDireita() == aux) {
            pai.setDireita(aux.getDireita());
        }
```

E agora?

```
  10   pai
    \
    20  aux
      \
       21
```

```
pai.setDireita(aux.getDireita());
```

```
            10 pai
                  \
                   21
```

E assim tiramos o 20 e o 21 herda seu lugar devido ao setDireita(aux.getDireita.

REMOVER Dois filhos

```
private void removeComDoisFilhos(Nodo aux) {
        // Encontrar o sucessor (menor valor da subárvore direita)
        Nodo sucessor = aux.getDireita();
        Nodo paiSucessor = aux;

        while (sucessor.getEsquerda() != null) {
            paiSucessor = sucessor;
            sucessor = sucessor.getEsquerda();
        }

        aux.setValor(sucessor.getValor());

        // Agora removemos o sucessor (ele tem no máximo um filho à direita)
        if (sucessor.getDireita() != null) {
            removeComFilhoDireita(sucessor, paiSucessor);
        } else {
            removeFolha(sucessor, paiSucessor);
        }
    }
```

°1 vamos desmembrar os códigos

```
 while (sucessor.getEsquerda() != null) {
          1   paiSucessor = sucessor;
          2   sucessor = sucessor.getEsquerda();
         }
```

O que esse while faz? Ele procura na direita o menor valor da arvore para ser o sucessor e tem que ser na esquerda.

```
                    10    -> aux
                  /    \
                5       15
                      /
                    12 -> sucessor e paisucessor
```

1 paiSucessor = sucessor = 15
2 sucessor = sucessor.getEsquerda = 12

Vamos traduzir acima:

PaiSucesspr = sucessor
Sucessor = sucessor.getEsquerda

paiSucessor guarda o valor de 12 antigo para removermos ele, se caso não fizermos isso, iriamos trocar o 10 pelo 12 e o 12 ficaria ali onde está.
Eu quero percorrer o loop até o menor valor, que seria o 12.

Resultado:

Paisucessor = 15
Sucessor = 12

E depois disso?

```
aux.setValor(sucessor.getValor());
```

Traduzindo:

10 sai para o 12 substituir

Agora o 12 é raiz

```
      12
    /   \
  5      15
        /
      12 (por que ainda está aqui?)
```

Aguarde logo abaixo:

Chamamos o método REMOVEFILHOCOMDIREITA para tratar uma condição do if. Ta mas por que só o direita?
- É a questão lógica que o 12 tem que ser o último da esquerda, mas não quer dizer que é o último da direita.

```
if (sucessor.getDireita() != null) {
            removeComFilhoDireita(sucessor, paiSucessor);
        }
```

E se não tiver nada? Então vamos remover oq? Somente o 12!!

```
else {
            removeFolha(sucessor, paiSucessor);
        }
```

E assim o 12 se vai!!

```
          12
        /   \
      5      15
```

Agora o método principal

```
public void removerRecursivo(Nodo aux, Nodo pai, int valor) {
     // parte principal para o funcionamento
        if (aux == null) {
            System.out.println("Valor não encontrado!");
            return;
        }
```

```
        if (valor < aux.getValor()) {
            removerRecursivo(aux.getEsquerda(), aux, valor);
        } else if (valor > aux.getValor()) {
            removerRecursivo(aux.getDireita(), aux, valor);
        } else {

            if (aux.getEsquerda() == null && aux.getDireita() == null) {
                removeFolha(aux, pai);
            } else if (aux.getEsquerda() != null && aux.getDireita() == null) {
                removeComFilhoEsquerda(aux, pai);
            } else if (aux.getEsquerda() == null && aux.getDireita() != null) {
                removeComFilhoDireita(aux, pai);
            } else {
                removeComDoisFilhos(aux);
            }
            System.out.println("Valor " + valor + " removido.");
        }
    }
```