Aula Prática 9

1 - Apresentar um critério de balanceamento para as árvores binárias de pesquisa

Para balancear nossa árvore, faremos uma pequena modificação e uma adição ao critério básico de uma árvore AVL. Dessa maneira:

- A diferença entre a **altura das subárvores** da direita e da esquerda devem ser no máximo 3.
- A diferença entre a **quantidade de nós** nas sub-árvores da direita e da esquerda é no máximo 3

Para uma árvore ser considerada balanceada, ela deverá atender a esses dois critérios simultaneamente.

2 - Explicar porque o critério escolhido realmente balanceia a árvore.

A primeira condição do nosso critério estabelece que a diferença de altura entre as subárvores da direita e da esquerda deve ser no máximo 3. Isso significa que a árvore não pode se tornar excessivamente desbalanceada em termos de altura. Ao limitar essa diferença, evitamos que uma das subárvores cresça de forma desproporcional em relação à outra, o que poderia comprometer o desempenho das operações de busca e outras operações realizadas na árvore.

A segunda condição estabelece que a diferença na quantidade de nós entre as subárvores direita e esquerda deve ser no máximo 3. Isso garante uma distribuição equilibrada dos nós na árvore, considerando não apenas a altura, mas também a quantidade de nós em cada subárvore. Essa condição é especialmente útil em cenários em que a árvore sofre atualizações frequentes, como inserções e exclusões de nós. Ao manter a distribuição balanceada dos nós, evitamos que uma das subárvores cresça desproporcionalmente mais rápido que a outra, o que poderia comprometer o balanceamento global da árvore.

Ao exigir que a árvore atenda simultaneamente a ambos os critérios para ser considerada balanceada, estamos buscando um equilíbrio refinado, que leva em conta tanto a altura quanto a distribuição dos nós. Esse critério é eficiente porque promove um balanceamento mais preciso e adaptável, permitindo uma melhor utilização dos recursos da árvore e um desempenho mais consistente em suas operações.

3 - Indicar os segmentos de código (linhas de código) que devem ser re-avaliadas tendo em vista o critério escolhido.

Para implementar esse critério, devemos reavaliar os seguintes segmentos:

- Linhas 95 – 103 e 107 – 115: Condições do BF no **método de inserção.**
- Linha 120 – 138 e 142 – 151: Condições do BF no **método de remoção**.
- Além disso, precisaremos criar uma nova função para medir a diferença entre a quantidade de nós nas sub-árvores da direita e da esquerda.

4 - Descrever a modificação necessária para o algoritmo de inclusão para manter a árvore satisfazendo o critério escolhido.

Criar essas duas funções:

```
84
85  int countNodes(node* T) {
86    if (T == NULL) {
87      return 0;
88    }
89    return 1 + countNodes(T->left) + countNodes(T->right);
90  }
91
92  int getNodesDifference(node *T) {
93    if (T == NULL)
94      return 0;
95
96    int leftNodes = countNodes(T->left); // Função auxiliar para contar o número de nós na subárvore esquerda
97    int rightNodes = countNodes(T->right); // Função auxiliar para contar o número de nós na subárvore direita
98
99    return rightNodes - leftNodes; // Retorna a diferença entre a quantidade de nós na subárvore da direita e na subárvore da esquerda
100 }
101
```

Aplicar as funções criadas e a nova diferença de altura permitida nas condições da função de inserção:

```
{
  T -> right = insert(T -> right, x, st);
  if (BF(T) == -3 || getNodesDifference(T) > 3){
    if (x > T -> right -> data){
      T = RR(T);
st->RR++;
    } else{
      T = RL(T);
st->RL++;
    }
  }
} else
if (x < T -> data) {
  T -> left = insert(T -> left, x, st);
  if (BF(T) == 3 || getNodesDifference(T) > 3){
    if (x < T -> left -> data){
      T = LL(T);
```

5 - Descrever a modificação necessária para o algoritmo de remoção para manter a árvore satisfazendo o critério escolhido.

Criar essas duas funções:

```
int countNodes(node* T) {
  if (T == NULL) {
    return 0;
  }
  return 1 + countNodes(T->left) + countNodes(T->right);
}

int getNodesDifference(node *T) {
  if (T == NULL)
    return 0;

  int leftNodes = countNodes(T->left); // Função auxiliar para contar o número de nós na subárvore esquerda
  int rightNodes = countNodes(T->right); // Função auxiliar para contar o número de nós na subárvore direita

  return rightNodes - leftNodes; // Retorna a diferença entre a quantidade de nós na subárvore da direita e na subárvore da esquerda
}
```

Aplicar as funções criadas e a nova diferença de altura permitida nas condições da função de remoção:

```
  if (BF(T) == 3 || getNodesDifference(T) > 3){
    if (BF(T -> left) >= 0){
      T = LL(T);
st->LL++;
    }
    else{
      T = LR(T);
st->LR++;
    }
  }
} else
if (x < T -> data) {
  T -> left = Delete(T -> left, x, st);
  if (BF(T) == -3 || getNodesDifference(T) > 3){ //Rebalance during windup
    if (BF(T -> right) <= 0){
      T = RR(T);
st->RR++;
    }
    else{
      T = RL(T);
st->RL++;
    }
  }
} else {
  //data to be deleted is found
  if (T -> right != NULL) { //delete its inorder succesor
    p = T -> right;
    while (p -> left != NULL)
      p = p -> left;
    T -> data = p -> data;
    T -> right = Delete(T -> right, p -> data, st);
    if (BF(T) == 3 || nodesDifference(T) > 3){ //Rebalance during windup
```

6 - Implementar as modificações apresentadas nos pontos anteriores.

Já feito.

7 - Comparar com a AVL original em termos do número de rotações para um conjunto de operações de inserções e retiradas.

AVL Original: Insere 1, 0, 2, 3, 8, 4. Remove: 1
Foram feitas 4 rotações: uma na inserção do 8, duas na inserção do 4 e uma na remoção do 1

```
Insert 1
   1 (Bf=0, H=0)
0

Insert 0
   1 (Bf=1, H=1)
       0 (Bf=0, H=0)
2

Insert 2
   1 (Bf=0, H=1)
       0 (Bf=0, H=0)
       2 (Bf=0, H=0)
3

Insert 3
   1 (Bf=-1, H=2)
       0 (Bf=0, H=0)
       2 (Bf=-1, H=1)
           3 (Bf=0, H=0)
8

Insert 8
   1 (Bf=-1, H=2)
       0 (Bf=0, H=0)
       3 (Bf=0, H=1)
           2 (Bf=0, H=0)
           8 (Bf=0, H=0)
4

Insert 4
   3 (Bf=0, H=2)
       1 (Bf=0, H=1)
           0 (Bf=0, H=0)
           2 (Bf=0, H=0)
       8 (Bf=1, H=1)
           4 (Bf=0, H=0)
```

```
Dump:
   3 (Bf=0, H=2)
       2 (Bf=1, H=1)
           0 (Bf=0, H=0)
       8 (Bf=1, H=1)
           4 (Bf=0, H=0)
```

AVL Modificada:

Foram feitas 2 rotações: uma na inserção do 4 e uma na remoção do 2. Porém, ela teve o 2 como eixo.

```
Insert 1
   1 (Bf=0, H=0)
0

Insert 0
   1 (Bf=1, H=1)
       0 (Bf=0, H=0)
2

Insert 2
   1 (Bf=0, H=1)
       0 (Bf=0, H=0)
       2 (Bf=0, H=0)
3

Insert 3
   1 (Bf=-1, H=2)
       0 (Bf=0, H=0)
       2 (Bf=-1, H=1)
           3 (Bf=0, H=0)
8

Insert 8
   1 (Bf=-2, H=3)
       0 (Bf=0, H=0)
       2 (Bf=-2, H=2)
           3 (Bf=-1, H=1)
               8 (Bf=0, H=0)
4

Insert 4
   1 (Bf=-2, H=3)
       0 (Bf=0, H=0)
       3 (Bf=-1, H=2)
           2 (Bf=0, H=0)
           8 (Bf=1, H=1)
               4 (Bf=0, H=0)
```

```
Dump:
   2 (Bf=-2, H=3)
       0 (Bf=0, H=0)
       3 (Bf=-2, H=2)
           8 (Bf=1, H=1)
               4 (Bf=0, H=0)
```