Aula Prática 9 - Plano

Giovanna Naves Ribeiro
Matrícula: 2022043647

1. **Apresentar um critério de balanceamento para as árvores binárias de pesquisa;**

O critério de balanceamento escolhido para a árvore é: “A diferença de altura entre as sub-árvores direita e esquerda é no máximo 3”. Semelhante ao critério adotado no código fornecido (de AVL).

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

Ele balanceia pois faz com que não haja um nó com uma subárvore direita muito maior que a esquerda e vice e versa. Assim, a altura da árvore não ficará tão maior que o necessário/mínimo.

3. **Indicar os segmentos de código (linhas de código) que tem que ser reavaliadas tendo em vista o critério escolhido.**

Realizaremos as linhas grifadas em rosa no código a seguir, que medem a diferença de altura das subárvores nas funções de inserir e deletar elementos e checam se é igual a 2 ou a -2. Essas linhas definirão se será preciso rotacionar a árvore para balanceá-la.

```
node * insert(node * T, int x, stat_t * st) {
  if (T == NULL) {
    T = (node * ) malloc(sizeof(node));
    T -> data = x;
    T -> left = NULL;
    T -> right = NULL;
  } else
  if (x > T -> data) // insert in right subtree
  {
    T -> right = insert(T -> right, x, st);
    if (BF(T) == -2){
      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) == 2){
      if (x < T -> left -> data){
        T = LL(T);
    st->LL++;
      }else{
        T = LR(T);
    st->LR++;
      }
    }
  }
  T -> ht = height(T);
  return (T);
}
```

```
node * Delete(node * T, int x, stat_t * st) {
  node * p;
  if (T == NULL) {
    return NULL;
  } else
  if (x > T -> data) // insert in right subtree
  {
    T -> right = Delete(T -> right, x, st);
    if (BF(T) == 2){
      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) == -2){ //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) == 2){ //Rebalance during windup
        if (BF(T -> left) >= 0){
          T = LL(T);
      st->LL++;
    }
        else{
          T = LR(T);\
      st->LR++;
    }
      }
    } else
      return (T -> left);
  }
  T -> ht = height(T);
  return (T);
}
```

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

Quando a função de inserir compara BF (diferença da altura das subárvores) com 2 e -2, ela descobre se o módulo da diferença das alturas passa de 1. Como não queremos que a diferença passe de 3 em nosso método de balanceamento, devemos comparar BF com 4 e -4.

```
if (BF(T) == -4){
      if (x > T -> right -> data){
        T = RR(T);
    st->RR++;
      } else{
```

```
        T = RL(T);
    st->RL++;
      }
    }
```

```
if (BF(T) == 4){
      if (x < T -> left -> data){
        T = LL(T);
    st->LL++;
      }else{
        T = LR(T);
    st->LR++;
      }
    }
```

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

Quando a função de deletar compara BF (diferença da altura das subárvores) com 2 e -2, ela descobre se o módulo da diferença das alturas passa de 1. Como não queremos que a diferença passe de 3 em nosso método de balanceamento, devemos comparar BF com 4 e -4.

```
if (BF(T) == 4){
      if (BF(T -> left) >= 0){
        T = LL(T);
    st->LL++;
      }
      else{
        T = LR(T);
    st->LR++;
      }
    }
```

```
if (BF(T) == -4){ //Rebalance during windup
      if (BF(T -> right) <= 0){
        T = RR(T);
    st->RR++;
      }
      else{
        T = RL(T);
    st->RL++;
      }
```

```
    }
```

```
if (BF(T) == 4){ //Rebalance during windup
        if (BF(T -> left) >= 0){
          T = LL(T);
      st->LL++;
    }
        else{
          T = LR(T);\
      st->LR++;
    }
      }
```

6. **Implementar as modificações apresentadas nos pontos anteriores.**

(Ver imagens acima).