LISTA DE EXERCÍCIOS – ALGORITMOS E ESTRUTURA DE DADOS III

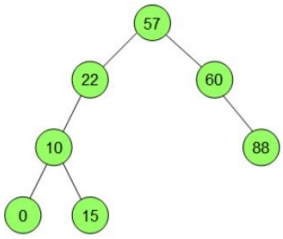
Lista de Exercícios 9

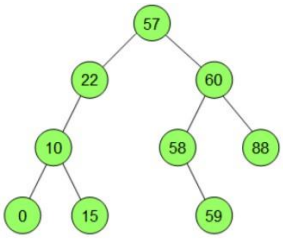
DOCENTE: Thiago França Naves **DATA:21/11/2025**

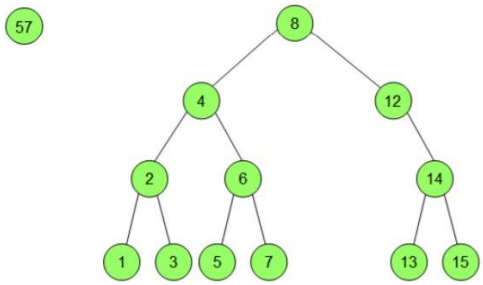
ALUNO: Edryck Freitas Nascimento

OBS: Todos os exercidos devem ser feitos utilizando o conceito de Árvores Binárias, com o projeto "ProjArvoreAVL" disponibilizado junto com o material da aula.

1) Examine cada árvore abaixo e responda se é ou não uma AVL e justifique a resposta.







Da esquerda para a direita:
Não é AVL, o nó 22 tem um fator de balanceamento de 2
Não é AVL, o nó 22 tem um fator de balaceamento de 2
É AVL
Não é AVL, o nó 12 tem um fator de balanceamento de -2

2) Monte a árvore AVL (passo-a-passo) para as seguintes inserções de chaves, indicando a cada passo qual elemento foi inserido ou qual rotação foi realizada, Ilustrar graficamente/textualmente como é feito o balanceamento:

a) 50, 30, 20, 70, 40, 35, 37, 38, 10, 32, 45, 42, 25, 47, 36.

1. Insere 50 e o 50 é a raiz
2. Insere 30 a esq do 50
3. Insere 20 a esq do 30

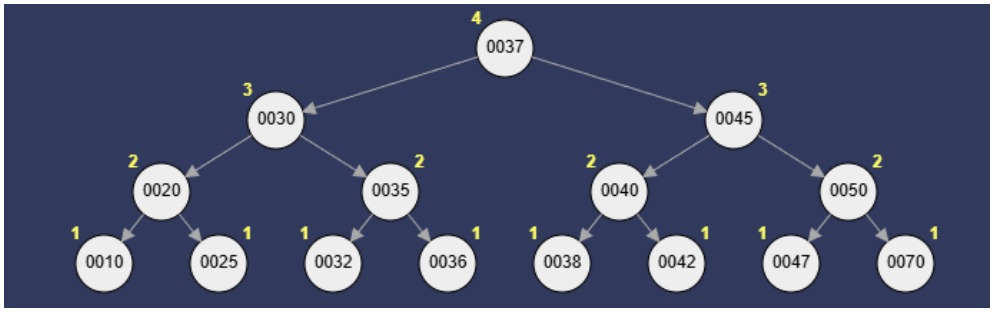
Rotaciona LL no 50 e o 30 vira a raiz

4. Insere 70 a dir do 50
5. Insere 40 a esq do 50
6. Insere 35 a esq do 35

Rotaciona RL no 30 e o 40 vira a raiz

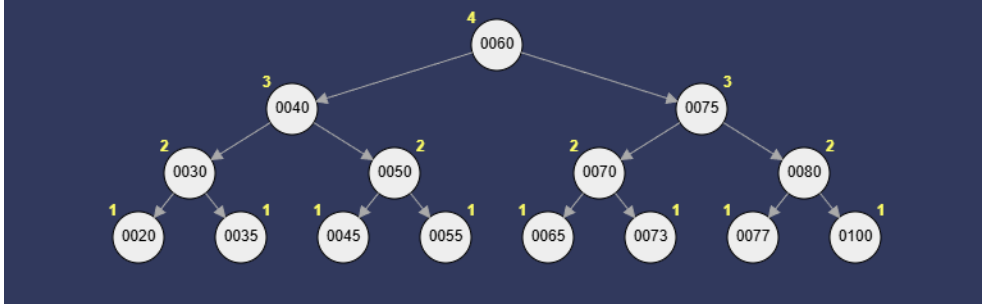
7. Insere 37 a dir do 35
8. Insere 38 a dir do 37

Rotaciona RR no 35 e o 37 sobe
9. Insere 10 a esq do 20
10. Insere 32 a esq do 35
Rotaciona LR no 40 e o 37 vira a raiz
11. Insere 45 a esq do 50
12. Insere 42 a eqs do 42
Rotaciona RL no 40 e o 45 sobe
13. Insere 25 a dir do 20
14. Insere 47 a esq do 50
15. Insere 36 a dir do 35
Estado final:



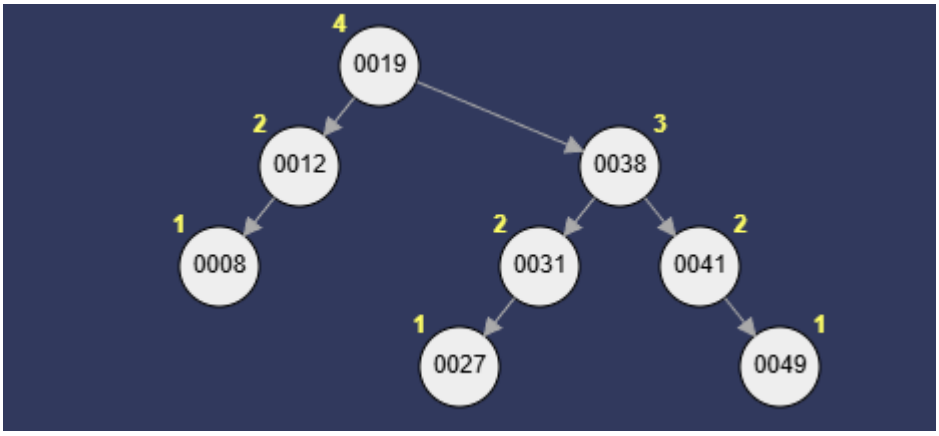
b) 100, 80, 60, 40, 20, 70, 30, 50, 35, 45, 55, 75, 65, 73, 77.

1. Insere 100 e o 100 é a raiz
2. Insere 80 a esq do 100
3. Insere 60 a esq do 80
Rotaciona LL no 100 e o 80 vira raiz
4. Insere 40 a esq do 60
5. Insere o 20 a esq do 40
Rotaciona LL 60 e o 40 sobe
6. Insere 70 a dir do 60
Rotaciona LR no 80 e o 60 sobe
Rotaciona RR no 40 e o 60 vira raiz
7. Insere 30 a dir do 40
Rotaciona LR no 40
Rotaciona RR no 20 e o 30 sobe
Rotaciona LL no 40 e o 30 sobe
8. Insere 50 na dir do 40
9. Insere 35 na esq do 40
10. Insere 45 na dir do 50
Rotaciona RR no 30 e o 40 sobe
11. Insere 55 na dir do 50
12. Insere 75 na dir do 70
13. Insere 65 na esq do 70
14. Insere 73 na esq do 75
Rotaciona LR no 80
Rotaciona RR no 70 e o 75 sobe
Rotaciona LL no 80 e o 75 sobe
15. Insere 77 na esq do 80
Estado Final:



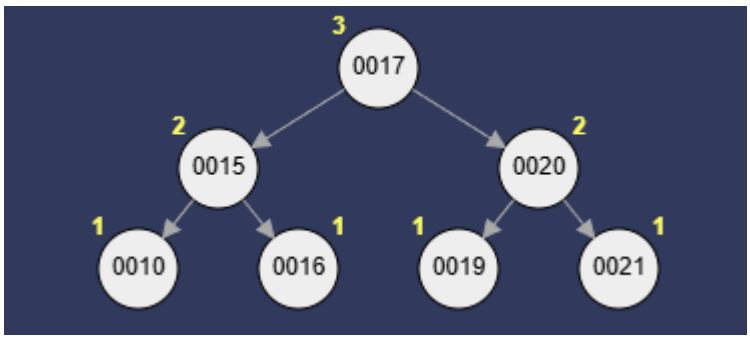
c) 41,38, 31, 12, 19, 8, 27, 49.

1. Insere 41 e o 41 é a raiz
2. Insere 38 na esq do 41
3. Insere 31 na esq do 38
Rotaciona LL no 41 e o 38 vira raiz
4. Insere 12 na esq do 31
5. Insere 19 na dir do 12
Rotaciona RR no 31 e o 19 sobe
Rotaciona LL no 31 e o 19 sobe
6. Insere 8 na esq do 12
7. Insere 27 na esq do 31
8. Insere 49 na dir do 41
Estado Final:



d) 10, 21, 15, 17, 16, 19, 20

1. Insere 10 e a raiz é o 10
2. Insere 21 a dir do 10
3. Insere 15 a esq do 21
Rotaciona RL no 10
Rotaciona LL no 21
Rotaciona RR no 10 e o 15 vira raiz
4. Insere 17 na esq do 21
5. Insere 16 na esq do 17
Rotaciona LL no 21 e o 17 fica no lugar do 21
6. Insere 19 na esq do 21
Rotaciona RR no 15 e o 17 vira a raiz
7. Insere o 20 na dir do 19
Rotaciona LR no 21
Rotaciona RR no 19 e o 20 sobe
Rotaciona LL no 21 e o 20 sobe
Estado final:



3) Um certo professor Amongus afirma que a ordem pela qual um conjunto fixo de elementos é inserido em uma árvore AVL não interessa – sempre resulta na mesma árvore. Apresente um pequeno exemplo que prove que ele está errado.

   Ele está errado, por mais que sejam iguais (nos nós) ainda vai se resultar em uma diferente da outra, por causa das rotações que vão mudar a árvore.
   Exemplos:
   Para o conjunto {1, 2, 3, 4}

   Ordem de inserção: 3, 1, 2, 4

```
      [3]
     /   \
   [2]   [4]
  /
[1]
```

   Ordem de inserção: 2, 1, 4, 3

```
    [2]
   /   \
 [1]   [4]
      /
    [3]
```

4) Construa a função de rotação simples a esquerda *void RotacaoRR(ArvAVL *raiz)*

   Código está no .c

5) Descreva passo a passo o funcionamento da função *insere_ArvAVL(ArvAVL *raiz, int valor)*, descreva também o uso de qualquer outras funções dentro dessa. Utilize trechos do código para exemplificar.

   A função de inserção funciona de forma recursiva, primeiro buscando a posição correta para o novo nó (if(*raiz == NULL)).
   Após a inserção, a função atualiza a altura do nó atual no caminho de volta (atual->altura = maior(...) + 1).
   Em seguida, ela verifica se o nó ficou desbalanceado (if(fatorBalanceamento_NO(atual) >= 2) ou o oposto no lado direito).
   Se houver desbalanceamento, o código identifica o tipo (Simples: if(valor < (*raiz)->esq->info) ou Dupla: else) e aplica a rotação que melhor se aplica para rebalancear a árvore (ex: RotacaoLL(raiz)), garantindo que a propriedade AVL seja mantida em todos os níveis antes de retornar.

6) Descreva passo a passo o funcionamento da função *remove_ArvAVL(ArvAVL *raiz, int valor)*, descreva também o uso de qualquer outras funções dentro dessa. Utilize trechos do código para exemplificar.

   A remoção na AVL é recursiva e ocorre em três fases principais: busca, remoção ou substituição e balanceamento.
   A remoção do nó alvo (if((*raiz)->info == valor)) lida com os casos de 0/1 filho (liberando o nó) ou 2 filhos (substituindo pelo menor sucessor, achado por procuraMenor).
   O balanceamento é verificado e corrigido depois do retorno de cada chamada recursiva, tanto na busca (if((res = remove_ArvAVL(...) == 1)) quanto na substituição.
   O código usa a comparação de alturas dos netos (ex: altura_NO((*raiz)->dir->esq) <= altura_NO((*raiz)->dir->dir)) para decidir se aplica uma Rotação Simples (RotacaoRR(raiz)) ou Dupla (RotacaoRL(raiz)), assim garantindo que a árvore continue sendo AVL.

7) Faça uma função que dada uma árvore verifica se a mesma é AVL.

   Código está no .c

8) Faça uma função Arv* transforma(Arv *raiz) que dada uma árvore binária de busca qualquer, retorna uma nova árvore AVL.

   Código está no .c