Universidade Federal do Espírito Santo – Centro Tecnológico
Departamento de Informática
Prof. Vinícius Fernandes Soares Mota

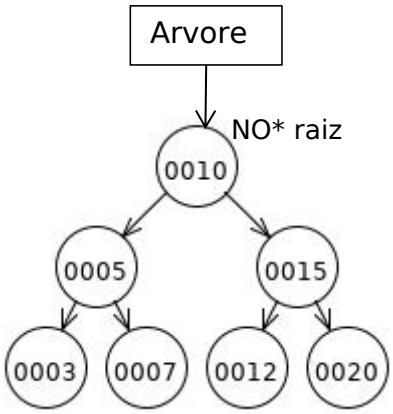
UFES

Lista de exercícios

Árvores Binárias de Busca (ABB)

Dica: Use o simulador abaixo para conferir suas respostas:

https://www.cs.usfca.edu/~galles/visualization/BST.html




```
struct No{
  struct TItem;
  struct No* esq;
  struct No* dir;
};
typedef struct NO* Arvore;
```

1. Indique uma ordem de inserção que gera uma árvore igual da figura acima.

2. Quais as propriedades de uma árvore binária de busca?

3. O que é uma árvore binária balanceada?

4. Quantos elementos (nós), no mínimo e no máximo, podem ter uma árvore binária de altura 5? (obs: uma árvore vazia tem altura 0)

5. Uma árvore binária completa e balanceada tem 31 nós até o seu nível N. Sabe-se que ela tem 2N-1 níveis. Quantos nós tem a árvore?

6. Sabendo que uma árvore binária está totalmente balanceada e completa e tem 1023 nos até o seu nível N, quantos nos tem o nível N-1?

7. Dada uma ABB inicialmente vazia, insira (E DESENHE) os seguintes elementos (nessa ordem): M, F, S, D, J, P, U, A, E, H, Q, T, W, K.

8. Dada uma ABB inicialmente vazia, insira (E DESENHE) os seguintes elementos (nessa ordem): A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z.

9. Incremente o TAD ABB escrevendo funções que:

   a) Conta o número de nós de uma árvore binária.

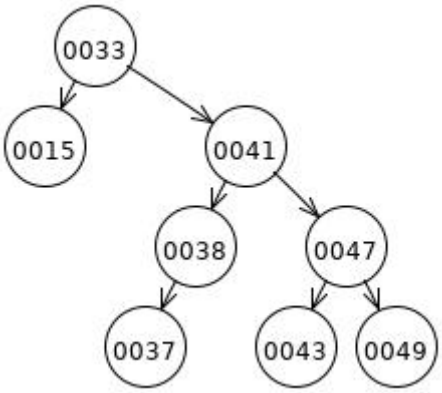
   b) Conta o número de nós não-folha de uma árvore binária.

   c) Conta o número de folhas de uma árvore binária.

d) Calcula a altura de uma árvore binária.

e) Encontra o valor máximo em uma árvore de busca binária.

10. Descreva a ordem de visita para um percurso em pré-ordem, em-ordem e pós-ordem na árvore abaixo:



11. Considere o algoritmo abaixo para a remoção em uma ABB:

```
1  int remove_ArvBin(Arvore*  raiz, int valor){
2     if(raiz == NULL)
3        return 0;
4     struct NO* ant = NULL;
5     struct NO* atual = *raiz;
6     while(atual != NULL){
7        if(valor == atual->info){
8           if(atual == *raiz)
9              *raiz = remove_atual(atual);
10          else{
11             if(ant->dir == atual)
12                ant->dir = remove_atual(atual);
13             else
14                ant->esq = remove_atual(atual);
15          }
16          return 1;
17       }
18       ant = atual;
19       if(valor > atual->info)
20          atual = atual->dir;
21       else
22          atual = atual->esq;
23    }
24    return 0;
25 }
```

```
1  struct NO* remove_atual(struct NO* atual) {
2     struct NO *no1, *no2;
3     if(atual->esq == NULL){
4        no2 = atual->dir;
5        free(atual);
6        return no2;
7     }
8     no1 = atual;
9     no2 = atual->esq;
10    while(no2->dir != NULL){
11       no1 = no2;
12       no2 = no2->dir;
13    }
14    // no2 é o nó anterior a r na ordem e-r-d
15    // no1 é o pai de no2
16    if(no1 != atual){
17       no1->dir = no2->esq;
18       no2->esq = atual->esq;
19    }
20    no2->dir = atual->dir;
21    free(atual);
22    return no2;
23 }
```

12. Considere o código anterior e a árvore do exercício 10 e explique.

a) Ao remover o 37, a função remove_atual() é chamada em qual linha da função remove_ArvBin?

b) Qual trecho do código foi executado durante a remoção do 37?

c) Explique quais são os trechos de códigos executados ao remover 33. Considere a árvore original da figura.

d) Explique quais são os trechos de códigos executados ao remover 38. Considere a árvore original da figura.

e) Explique quais são os trechos de códigos executados ao remover 41. Considere a árvore original da figura.

13. Utilizando a árvore do exercício 10 e o algoritmo do exercício 11, mostre a árvore resultante das remoções consecutivas dos seguinte elementos: 41, 38, 47, 33

14. Escreva funções não recursivas para realizar os 3 tipos de percurso na árvore binária (dica: use uma pilha):

(a) pré-ordem

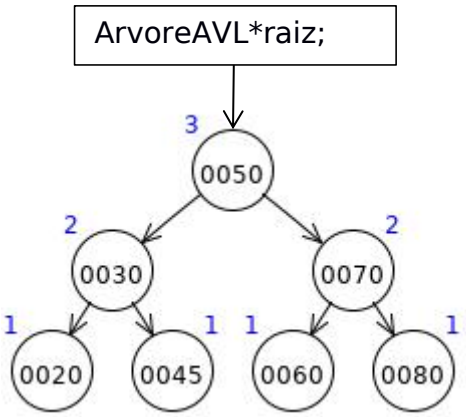
(b) em-ordem

(c) pós-ordem

15. Faça uma função que retorna a lista de caminhos da raiz até cada folha.

UFES

Árvore AVL




```
struct No{
  struct TItem;
  int  altura;
  struct No* esq;
  struct No* dir;
};
typedef struct NO* ArvoreAVL;
```

16. Defina com suas palavras o que é uma árvore AVL e como ela funciona.

17. Explique as vantagens e desvantagens de usar árvores binárias balanceadas?

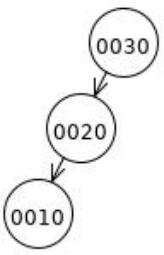
18. A inserção árvores AVL pode exigir a manutenção e atualização dos ponteiros das subárvores, as chamadas rotações. Considere as rotações simples a direita (SD) e simples a esquerda (SE) abaixo.

//A função Altura retorna a altura o a variável altura da subárvore e a função maior retorna o maior entre dois elementos.

```
void RotacaoSD(ArvAVL *A){//LL
    printf("RotacaoLL\n");
    struct NO *B;
    B = (*A)->esq;
    (*A)->esq = B->dir;
    B->dir = *A;
    (*A)->altura = maior(altura_NO((*A)-
>esq),altura_NO((*A)->dir)) + 1;
    B->altura = maior(altura_NO(B->esq),(*A)-
>altura) + 1;
//raiz passa a ser B
    *A = B;
}
```

```
void RotacaoSE(ArvAVL *A){//RR
    printf("RotacaoRR\n");
    struct NO *B;
    B = (*A)->dir;
    (*A)->dir = B->esq;
    B->esq = (*A);
    (*A)->altura = maior(altura_NO((*A)-
>esq),altura_NO((*A)->dir)) + 1;
    B->altura = maior(altura_NO(B->dir),(*A)-
>altura) + 1;
    (*A) = B;
}
```

19. Execute a operação RotacaoSD ao receber como raiz a subárvore 30 da figura abaixo:

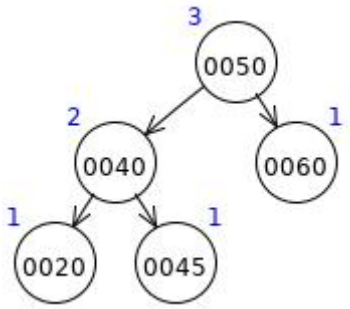


20. As rotações são aplicadas sempre que o fator de balanceamento é maior que 1. Isto é, ao inserir um elemento, atualiza-se a altura de cada sub-árvore e calcula a diferença de altura da subarvore esquerda para a direita. Considerando a sub-árvore abaixo, indique:

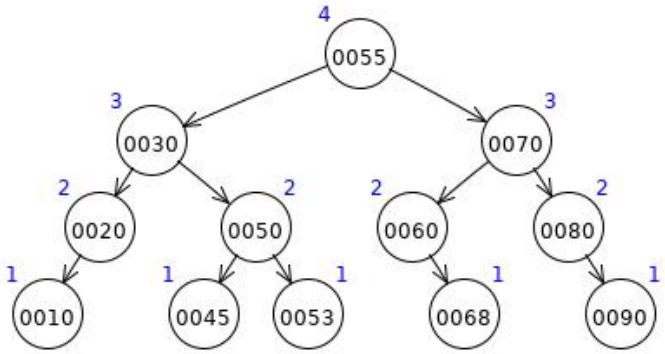
A) Qual a raiz da subárvore irá ficar desbalanceada ao inserir o 10.

B) Qual rotação será executada?

C) Execute os passos do algoritmo de rotação apropriado e desenhe a árvore AVL resultante.



21. Considerando a figura abaixo, responda:



A) Qual a raiz da sub-árvore irá ficar desbalanceada ao inserir o 100.

B) Qual rotação será executada?

C) Execute os passos do algoritmo de rotação apropriado e desenhe a árvore AVL resultante.

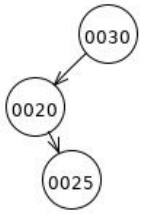
D) Insira o 110 na árvore resultante. Novamente, verifique quem é a raiz afetada e qual operação será executada.

22. A rotação dupla ocorre quando um nó é inserido a direita de uma sub-árvore esquerda (rotação dupla à direita), ou a esquerda de uma sub-árvore direita (rotação dupla à esquerda). A implementação é dada a seguir

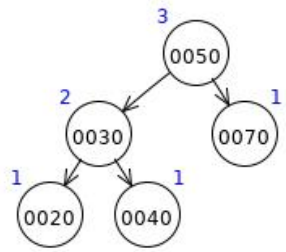
```
void RotacaoLR(ArvAVL *A){//LR
    RotacaoSE(&(*A)->esq);
    RotacaoSD(A);
}
```

```
void RotacaoRL(ArvAVL *A){//RL
    RotacaoSD(&(*A)->dir);
    RotacaoSE(A);
}
```

A) Execute a operação RotacaoLR ao receber como argumento a raiz **A* da sub-árvore 30 da figura abaixo:



B) Indique o nó desbalanceado, a operação e a árvore resultante ao inserir 35 na árvore abaixo:



23. Desenhe passo a passo a inserção de cada elemento em uma árvore AVL, indique quais operações foram necessárias:

35, 39, 51, 20, 13, 28, 22, 32, 25, 33 (nesta ordem).

Exercícios baseados em:

A. Backes. Estrutura de dados descomplicada em linguagem C. Elsevier. 2016.

N. Ziviani, F.C. Botelho, Projeto de Algoritmos com implementações em Java e C++, Editora Thomson, 2006.