INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
CEARÁ

4,9
PARABÉNS!

Engenharia de Computação
Pesquisa e Ordenação – Prof. Glauber Cintra

Herculano Gonçalves Santos
Lucas Diego Rebouças Rocha
Thiago Duarte Medeiros

Fortaleza, 19 de novembro de 2012.

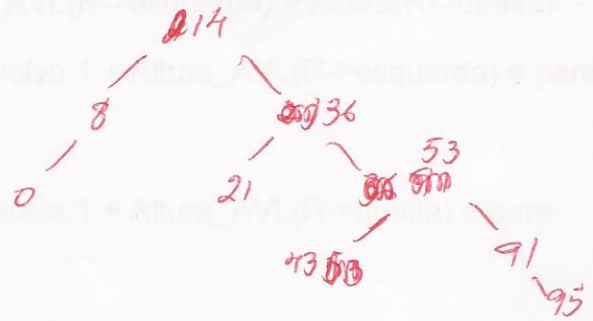
2ª Lista de Exercícios de Pesquisa e Ordenação

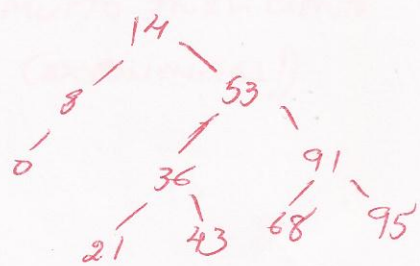
Escreva na tabela abaixo os seis últimos dígitos do número de matrícula e o ano de nascimento de um dos integrantes do seu grupo. Em seguida, calcule **cuidadosamente** os valores v_1, v_2, ..., v_{11} usando as fórmulas indicadas. Todas as questões valem **0,5 pontos**.

	d_1	d_2	d_3	d_4	d_5	d_6
Número de matrícula	0	2	0	0	9	7

	a_1	a_2	a_3	a_4
Ano de nascimento	1	9	8	3

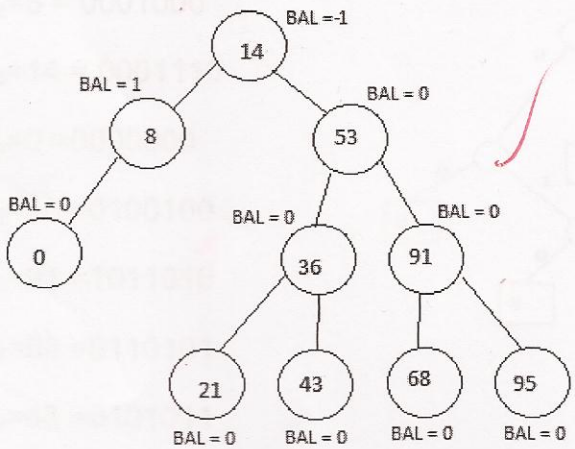
$v_1=4d_1+7a_4$	$v_2=4d_2+7d_1$	$v_3=4d_3+7d_2$	$v_4=4d_4+7d_3$	$v_5=4d_5+7d_4$	$v_6=4d_6+7d_5$	$v_7=4a_1+7d_6$	$v_8=4a_2+7a_1$	$v_9=4a_3+7a_2$	$v_{10}=4a_4+7a_3$
21	8	14	0	36	91	53	43	95	68



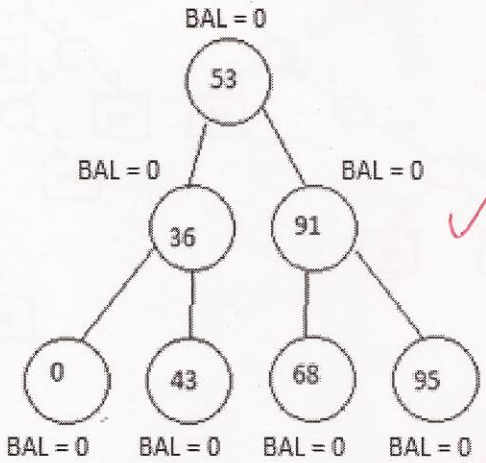


1) **Insira as chaves v_1, v_2, ..., v_{10}, nessa ordem, numa *árvore AVL*. Em seguida, remova v_1, v_2, e v_3, nessa ordem, da árvore. Desenhe como ficou a árvore, incluindo o *bal* de cada nó.**

Inserção:



Remoção de 21, 8 e 14



✓ 0,3

2) **Escreva uma função que receba um ponteiro para a raiz de uma *árvore AVL* e devolva a altura da árvore.**

Agoritmo Altura_AVL

Entrada: um ponteiro R para a raiz de uma árvore AVL

Saída: a altura dessa árvore

Se R = nulo

devolva 0 e pare

Senão

se Altura_AVL(R->esquerda) > Altura(R->direita)

devolva 1 + Altura_AVL(R->esquerda) e pare

senão

devolva 1 + Altura_AVL(R->direita) e pare

✓ 0,4

SOLUÇÃO CORRETA MAS MUITO INEFICIENTE (EXPONENCIAL!)

3) Mostre como ficaria uma *árvore trie de ordem 2* após a inserção da representação binária (com 7 bits) das chaves v_1, v_2, ..., v_{10}. Em seguida, remova a representação binária de v_2, v_3 e v_4 e mostre como ficaria a árvore.

Inserção:

V_1=21 = 0010101

V_2=8 = 0001000

V_3=14 = 0001110

V_4=0 =0000000

V_5=36 =0100100

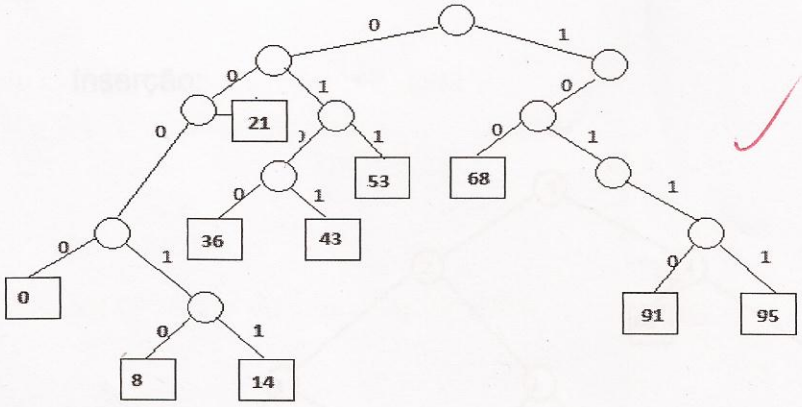
V_6=91 =1011010

V_7=53 =0110101

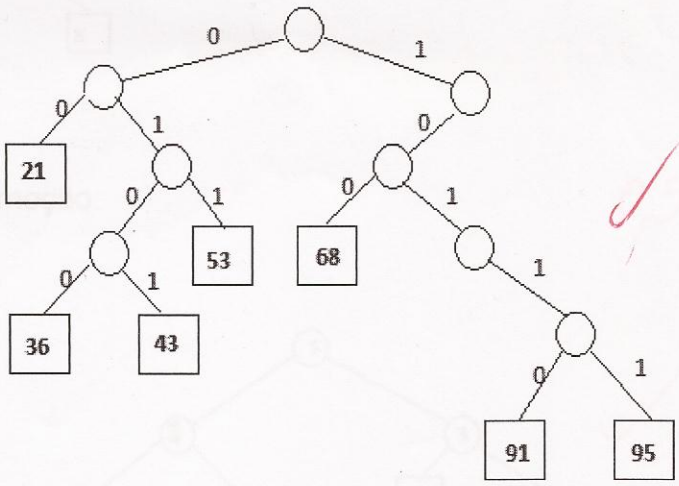
V_8=43 =0101011

V_9=95 =1011111

V_{10}=68 =1000100



Remoção: 8, 14 e 0



4) Mostre como ficaria uma *árvore patricia de ordem 2* após a inserção da representação binária (com 7 bits) das v_1, v_2, ..., v_{10}. Em seguida, remova a representação binária de v_3, v_4 e v_5 e mostre como ficaria a árvore.

V_1=21 = 0010101

V_2=8 = 0001000

V_3=14 = 0001110

V_4=0 =0000000

V_5=36 =0100100

V_6=91 =1011010

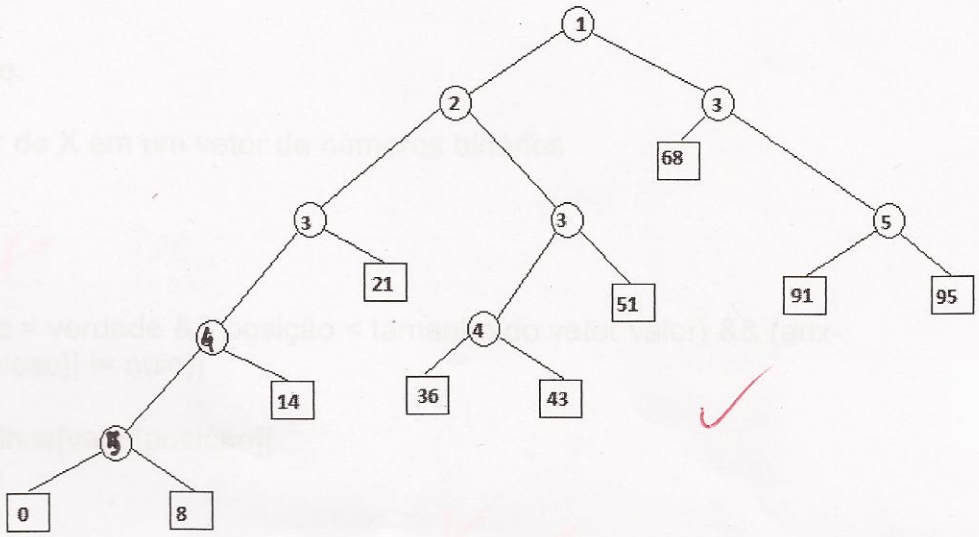
V_7=53 =0110101

V_8=43 =0101011

V_9=95 =1011111

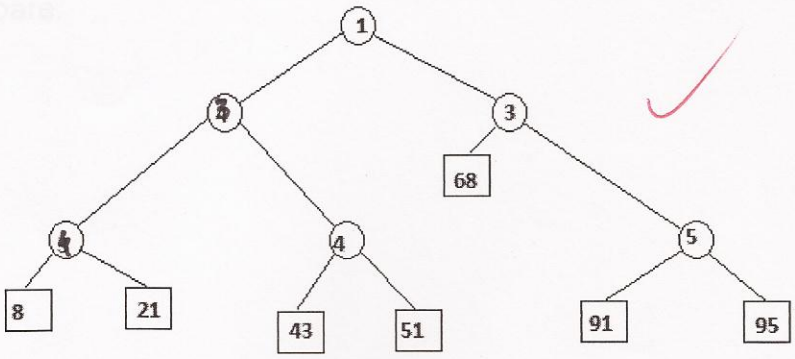
V_{10}=68 =1000100

Inserção:



Remoção:

0,5



5) **Escreva um algoritmo que receba um ponteiro para a raiz de uma *árvore patricia de ordem 2* (baseada no alfabeto binário) e um valor *x* e devolva *Sim* se *x* ocorre na árvore; *Não*, caso contrário.**

Algortimo Busca_Patricia

Entrada: um ponteiro R para a raiz de uma árvore e um valor X.
Saída: SIM se X ocorrer na árvore, NÃO caso contrario.

Se R = Nulo
    Devolva NÃO e pare.

valor = conversão do valor de X em um vetor de números binários
proximo= R
aux = proximo
posicao = ~~0~~ aux->info

Enquanto ((aux->noInterno = verdade && posição < tamanho do vetor valor) && (aux->ponteirosFilhos[valor[posicao]] != nulo))

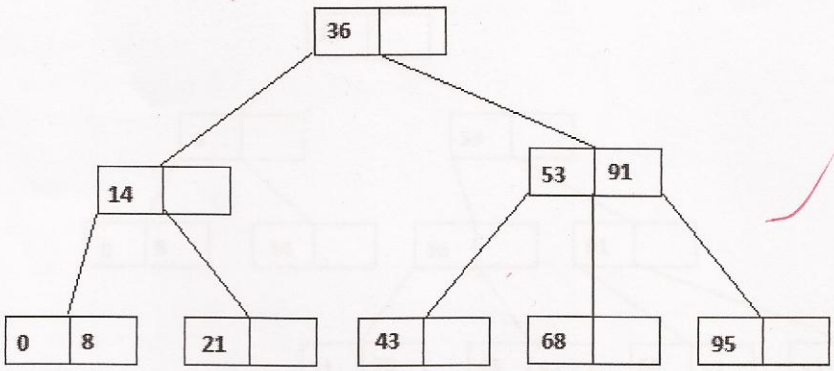
    próximo = aux->pFilhos[valor[posicao]]
    posição = aux->info
    aux = próximo

Se próximo = nulo
    Devolva NÃO e pare
Senão
    Se próximo->info == X
        Devolva SIM e pare.
    Senão
        Devolva NÃO e pare.

X 0,4

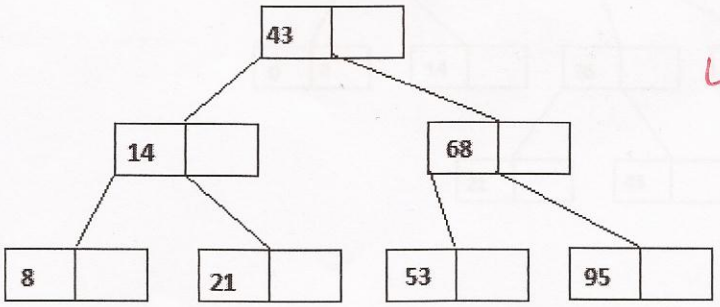
6) **Mostre como ficaria uma *árvore B de ordem 1* após a inserção das chaves v_1, v_2, ..., v_{10}, nesta ordem. Em seguida, remova v_4, v_5, e v_6 e mostre como ficaria a árvore.**

Inserção:



0,45

Remoção: 0, 36, 91



53

21 91

36

8, 14 43 68 95

14

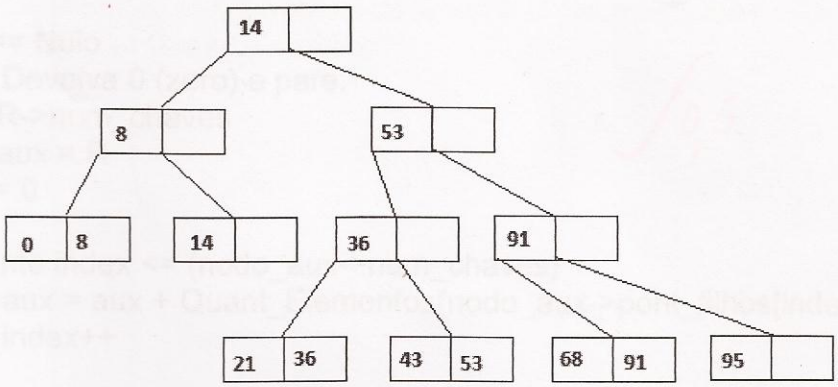
53, 91

0, 8 21 43 95

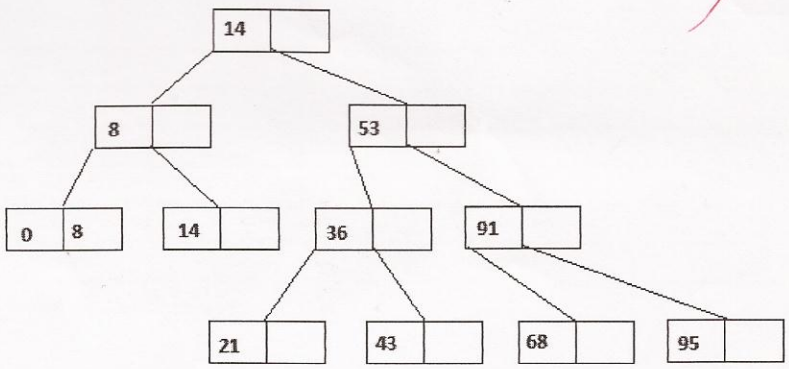
68

7) **Mostre como ficaria uma *árvore B+ de ordem 1* após a inserção das chaves v_1, v_2, ..., v_{10}, nesta ordem. Em seguida, remova v_5, v_6 e v_7 e mostre como ficaria a árvore.**

Inserção:



Remoção:



8) **Escreva uma função que receba um ponteiro para a raiz de uma *árvore B de ordem m* e devolva a quantidade de chaves contidas na árvore.**

Algoritmo Quant_Elementos
Entrada: um ponteiro R para a raiz de uma árvore B.
Saída: o número de elementos contidos na árvore B.

```
Se R == Nulo
        Devolva 0 (zero) e pare.
aux = R->num_chaves
nodo_aux = R
index = 0

Enquanto index <= (nodo_aux->num_chaves)
        aux = aux + Quant_Elementos(nodo_aux->pont_filhos[index])
        index++

Devolva aux e pare.
```

0,5

9) Uma tabela deve ser indexada pela sua chave primária através de uma árvore B. A chave primária tem 12 bytes e ponteiros ocupam 4 bytes. Além disso, junto com cada chave, é preciso armazenar o número da linha da tabela que contém aquela chave. Tal número requer 4 bytes. Sabendo que em cada leitura do disco, são lidos 2048 bytes, responda as questões a seguir, justificando suas respostas.

a) **Qual deve ser a *ordem* da árvore B de modo que cada nó possa ser lido com apenas 1 acesso a disco?**

$2m*12 + 2m*4 + (2m+1)*4 + 5 \leq 2048$
$24m + 8m + 8m + 9 \leq 2048$
$40m + 9 \leq 2048$
$40m \leq 2039$
$m \leq 50{,}975$
$m = 50$

0,3

b) **Se precisarmos armazenar 1 bilhão de chaves na árvore, qual será sua altura máxima, caso ela tenha a ordem calculada no item anterior?**

Nível	Nº mínimo de nós	Nº mínimo de chaves
1	1	1
2	2	100
3	202	10.100
4	10.302	515.100
5	525.402	26.270.100

No máximo 5 níveis de altura

6

c) **Se pudermos dispor de até 1 MB para armazenar os primeiros níveis da árvore na memória principal, quantos acessos a disco serão necessários, no pior caso, para encontrar uma chave na árvore, supondo que ela armazena 1 bilhão de chaves ?**

Nível	Nº mínimo de chaves	Bytes
1	1	40
2	100	4.000
3	10.100	404.000
4	515.100	20.604.000
5	26.270.100	1.050.804.000

Aproximadamente 3 acessos.

4

10) **Mostre como ficaria uma tabela de *hashing fechado* com 11 posições, após a inserção das chaves v_1, v_2, ..., v_{10}, nesta ordem (nessa e na próxima questão, os valores associados às chaves devem ser ignorados). Utilize a seguinte função de *hashing*: $h(x) = x \bmod 11$. Em seguida, remova v_1, v_2 e v_3 (nesta ordem) e mostre como ficaria a tabela.**

Inserção:

0		0
1		~~43*~~
2		68
3	3	~~14~~
4		~~36*~~
5		~~91*~~
6		
7		95
8	2	~~8~~
9		53
10	1	~~21~~

Remoção:

0	0
1	
2	68
3	36
4	91*
5	
6	
7	95
8	
9	53
10	43

11) **Mostre como ficaria uma tabela de *hashing aberto* com 7 posições, após a inserção das chaves v_1, v_2, ..., v_{10}, nesta ordem. Utilize a seguinte função de *hashing*: $h(x) = x \bmod 7$. Em seguida, remova v_7, v_8 e v_9 (nesta ordem) e mostre como ficaria a tabela.**

Inserção:

0	→ 21	→ 14	→ ~~10~~	→ 91
1	→ 8	→ 36	→ 43	
2				
3	→			
4	→ 53	→ 95		
5	→ 68			
6				

Remoção:

0	→ 21	→ 14	→ 10	→ 91
1	→ 8	→ 36		
2				
3				
4				
5	→ 68			
6				

12) **Explique o que é a *carga* de uma tabela de hashing e diga quando ela é considerada *baixa*. Explique também o que é uma *boa* função de hashing.**

A carga de uma tabela de Hashing é definida como sendo a razão entre o numero de chaves pelo tamanho da tabela(quantidade de posições).

A carga de uma de Hashing é considerada baixa quando ela não excede 50% do tamanho da tabela.

Uma tabela de Hashing é considerada boa se ela tiver tempo de execução constante, e se ela "dispersar" as chaves de maneira uniforme, evitando ao máximo as chances de colisões das chaves.

10,4