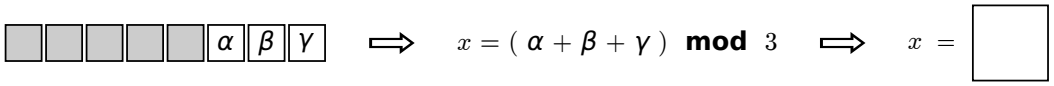
Prova 2 – Estruturas de Dados (INE5408) – 18mai2021

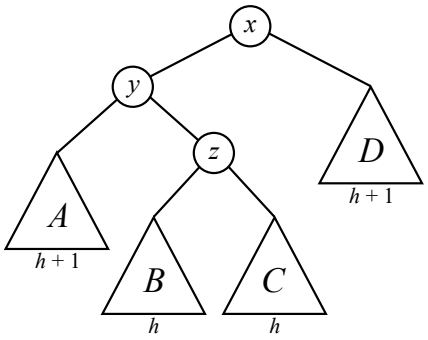
Ciências da Computação – Universidade Federal de Santa Catarina

Estudante: _____

Defina x em função dos últimos três dígitos de sua matrícula:



1. A figura abaixo é uma AVL. Cada triângulo (A, B, C e D) representa uma subárvore **completa** (ou seja, com todos os níveis preenchidos, de modo que qualquer inserção acarrete um acréscimo de sua altura). Sabe-se ainda que as alturas das subárvores A, B, C e D são, respectivamente, $h+1$, h, h e $h+1$. Pede-se:



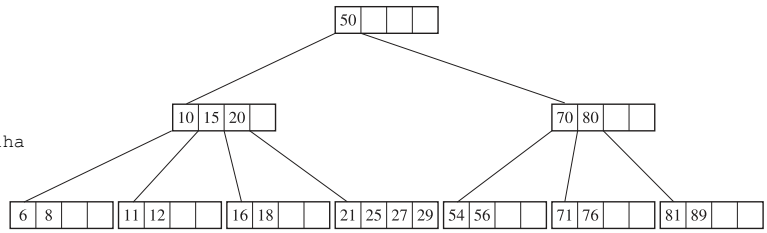
   (a) *(1,0pt)* Calcule o fator de balanceamento (diferença entre alturas de suas duas subárvores) para os nós x, y e z.

   (b) *(1,0pt)* Desenhe a AVL novamente, supondo que ocorra:

   - Para alunos com $x = 0$ $\Rightarrow$ Uma inserção na subárvore C.
   - Para alunos com $x = 1$ $\Rightarrow$ Uma inserção na subárvore A.
   - Para alunos com $x = 2$ $\Rightarrow$ Uma inserção na subárvore B.

2. Considerando a seguinte estrutura de nó de Árvore-B, sendo `M` igual ao número máximo de ponteiros para filhos (como exemplo, no desenho, `M = 5`):

```
template <class T, int M>
class BTreeNode {
  public:
    BTreeNode();
    BTreeNode(const T&);
  private:
    bool leaf;   // verdadeiro se for folha
    int size;    // nº chaves inseridas
    T keys[M-1]; // vetor de chaves
    BTreeNode *pointers[M];
};
```

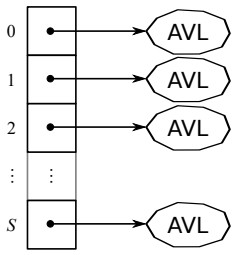


(2,5pt) De posse de um ponteiro para a raiz de uma Árvore-B, implemente um algoritmo:

- Para alunos com $x = 0$ $\Rightarrow$ Conte a quantidade de nós totalmente cheios.
- Para alunos com $x = 1$ $\Rightarrow$ Conte a quantidade de nós com o número mínimo de chaves.
- Para alunos com $x = 2$ $\Rightarrow$ Conte o número de folhas.

3. Um *hashing* aberto foi implementado por meio de uma tabela (vetor) com S ponteiros, e resolução de colisão feita por árvore binária de busca balanceada, AVL, conforme o desenho a seguir.

```
template <typename T>
class Hash {
  public:
    Hash(int S_);
  private:
    int S;
    AVLTree<T> *tabela;
    void insere(const T& chave);
    LinkedList<T> ordena();
    T maximo();
};
```



Considerando $S = 3$ e função de espalhamento igual a $f(\text{chave}) = \text{chave}$ **mod** S , pede-se:

(a) *(1,0pt)* Desenhe o *hashing* para inserções das seguintes chaves:
- Para alunos com $x = 0$ $\Rightarrow$ 10, 19, 28, 20, 16, 13, 7, 50
- Para alunos com $x = 1$ $\Rightarrow$ 60, 20, 30, 45, 39, 42, 50, 12
- Para alunos com $x = 2$ $\Rightarrow$ 47, 15, 65, 56, 83, 71, 92, 60

(b) *(1,0pt)* Escreva o método `LinkedList<T> ordena();` que cria uma lista (considere a estrutura `LinkedList` disponível) com todos os elementos do *hashing* em ordem crescente (reproduza o código necessário para o percurso em cada AVL).

(c) *(1,0pt)* Escreva o método `T maximo();` que devolve o maior elemento do *hashing* de modo mais eficiente possível (ou seja, com a menor quantidade de operações/comparações).

(d) *(1,0pt)* Discuta a complexidade computacional dos métodos implementados nos itens (b) e (c).

4. Segue uma implementação do QuickSort, considerando seu valor de x, definido no início da prova, como parâmetro de entrada, utilizado na escolha do pivô de particionamento:

```
QuickSort(A[ ], limInf, limSup, x)
1: se limInf < limSup então
2:     i ← Particione(A, limInf, limSup, x)
3:     QuickSort(A, limInf, i − 1, x)
4:     QuickSort(A, i + 1, limSup, x)
```

```
Particione(A[ ], limInf, limSup, x)
 1: se x = 0 então
 2:     i_pivo ← limInf
 3: senão se x = 1 então
 4:     i_pivo ← limSup
 5: senão se x = 2 então
 6:     i_pivo ← (limInf + limSup) div 2          ▷ (divisão inteira)
 7: pivo ← A[i_pivo]
 8: A[limSup] ↔ A[i_pivo]                        ▷ (troca de variáveis)
 9: i ← limInf − 1
10: para j ← limInf até (limSup − 1) faça
11:     se A[j] ≤ pivo então
12:         i ← i + 1
13:         A[i] ↔ A[j]                         ▷ (troca de variáveis)
14: A[i + 1] ↔ A[limSup]                        ▷ (troca de variáveis)
15: devolve i + 1
```

Considerando o seguinte vetor A de entrada:

A	30	20	50	10	70	80	90	100	60	40

Ao executar QuickSort$(A[\,],\ 0,\ 9,\ x)$, pede-se:

(a) *(0,5pt)* Liste todos os pivôs selecionados (variável *pivo*).

(b) *(1,0pt)* Escreva o conteúdo do vetor após cada particionamento.

Boa prova!