UFJF / DCC Estrutura de Dados e Lab. de Programação II – TVC 3 03/07/2017

Nome: _____ Matrícula: _____

Leia atentamente as instruções abaixo:

- Fazer o download do arquivo `Avaliacao3EDLab2-2017-1.zip` do site do curso. Descompactá-lo em um diretório de sua máquina. Este arquivo contém todos os códigos para o desenvolvimento da prova.
- A resposta de cada questão deve, **obrigatoriamente**, estar entre cada par de marcador (//Qi, //-Qi). Assim, a questão 1 está entre //Q1 e //-Q1, a questão 2 entre //Q2 e //-Q2 e assim por diante. Não remover, em hipótese alguma, tais marcadores de questões da sua prova. Caso sua solução tenha mais de uma função ou operação, elas devem estar entre esses marcadores, obrigatoriamente.
- Colocar no arquivo `main.cpp` seu nome completo e número de matrícula.
- A prova é **individual** e **sem qualquer tipo de consulta**.
- Existe apenas um projeto do Code::Blocks que será usado na prova.
- Antes de sair do laboratório, enviar ao servidor – usando a janela de upload – cada arquivo de código que contém as respostas das questões da sua prova. Aguarde um momento e verá as suas respostas de cada questão da prova.
- **O desenvolvimento e envio do código são de inteira responsabilidade do aluno!**
- Endereço do servidor: `http://172.18.40.97:8080/edlab2ufjf/`

---

1. (35 Pontos) Implementar uma operação para uma árvore binária de busca que retorne a altura da subárvore que tem como raiz o primeiro nó encontrado com valor igual a `val`. Esta operação deve passar por cada nó da árvore no máximo uma vez e usar a propriedade de árvore binária de busca, quando necessário. Retornar -1 caso não exista nenhum nó com o valor igual a `val`.

```
int ArvBinBusca::alturaNo(int val);
```

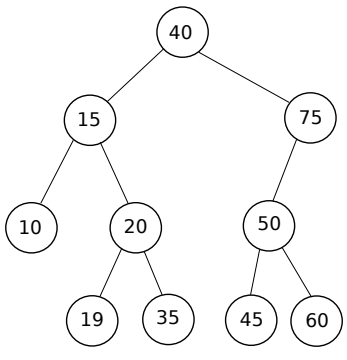
2. (35 Pontos) Implementar uma operação para uma árvore binária de busca que, dado um nível k, aloque (com um número de elementos apropriado), preencha e retorne um vetor com os valores armazenados nos nós até o nível k em ordem crescente. Visitar os nós no máximo uma vez e não visitar nós de níveis desnecessários. Caso a árvore não possua todos os elementos possíveis até o nível k, completar as posições não utilizadas do vetor com o valor -1. Se árvore for vazia, retornar `NULL`. A operação deve armazenar o tamanho do vetor que foi alocado em `n`.

```
int* ArvBinBusca::criaVetAteNivel(int k, int *n);
```

3. (30 Pontos) Implementar uma operação para uma árvore binária que, dado um nível k, remove todas as subárvores do nível $k+1$ da árvore. Portanto, após a remoção das subárvores, a altura resultante será k. Observe que a árvore original nunca se torna vazia (pois o menor nível é 0), a menos que ela já seja vazia. É necessário liberar a memória de todas as subárvores removidas.

```
void ArvBinBusca::removeSubNiveis(int k);
```

Exemplo:



Saída:

```
Questão 1: arv.alturaNo(15)
 Saída: 2
Questão 2: v = arv.criaVetAteNivel(2, &n)
 Saída: v = [10 15 20 40 50 75 -1]
Questão 3: arv.removeSubNiveis(1)
 Saída (impressão da árvore): (0)40
                              (1)--15
                              (1)--75
```