UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
ESTRUTURAS DE DADOS BÁSICAS II
PROFa SÍLVIA MARIA DINIZ MONTEIRO MAIA

TRABALHO – 2a UNIDADE

Frequentemente, uma estrutura de dados suporta quase todas as operações que precisamos. Quando não é caso, podemos aumentar a estrutura pela adição de informações e/ou operações. Neste projeto, vocês devem implementar uma árvore binária de busca (ABB) aumentada para suportar, além das operações convencionais de busca, inserção e remoção, as operações elencadas a seguir:

1. **QualquerTipo enesimoElemento (int n)**: retorna o n-ésimo elemento (contando a partir de 1) do percurso em ordem (ordem simétrica) da ABB.
2. **int posicao (QualquerTipo x):** retorna a posição ocupada pelo elemento x em um percurso em ordem simétrica na ABB (contando a partir de 1).
3. **QualquerTipo mediana ()**: retorna o elemento que contém a mediana da ABB. Se a ABB possuir um número par de elementos, retorne o menor dentre os dois elementos medianos.
4. **boolean ehCheia ()**: retorna verdadeiro se a ABB for uma árvore binária cheia e falso, caso contrário.
5. **boolean ehCompleta ()**: retorna verdadeiro se a ABB for uma árvore binária completa.
6. **String toString ()**: retorna uma String que contém a sequência de visitação (percorrimento) da ABB por nível.

Assuma que **QualquerTipo** representa o tipo do dado que está sendo armazenado na árvore, podendo ser, por exemplo, int, de forma que não precisa fazer a implementação genérica.

A maioria das operações descritas poderia ser facilmente implementada utilizando um percurso em ordem simétrica e, talvez, armazenando resultados em um vetor. Entretanto, esse procedimento **é ineficiente**. Ao invés disso, vamos **melhorar o desempenho dessas operações** aumentando os nós da ABB, isto é, armazenando informações extras em cada nó da árvore que irão simplificar as operações. **Uma informação pode ser a quantidade de nós nas subárvores à direita e à esquerda**. **Outras informações necessárias, se for o caso, devem ser identificadas por vocês.** Seu algoritmo deve receber dois arquivos como parâmetros.

O primeiro, contém uma descrição da ABB que será utilizada e é denominado arquivo de entrada da ABB. O arquivo de entrada da ABB deve conter uma sequência de valores inteiros separados por um espaço, os valores a serem armazenados na árvore. Pressupõe-se que a árvore equivalente será gerada pela chamada ao método de inserção para cada um dos elementos listados, na sequência em que são fornecidos no arquivo.

O segundo arquivo, denominado arquivo de comandos, contém uma sequência de operações (uma operação por linha) a serem realizadas pelo seu algoritmo. O arquivo de comando poderá utilizar as operações a seguir:

*Formato:
ENESIMO N
POSICAO N
MEDIANA
CHEIA
COMPLETA
IMPRIMA
REMOVA N
INSIRA N

*Exemplo:
IMPRIMA
MEDIANA
ENESIMO 10

Observações importantes:
1. QualquerTipo pode ser entendido como inteiro.
2. Valores duplicados não serão permitidos na árvore. Tentativas de inserção de um valor duplicado devem ser previstas e devidamente ignoradas.
3. Tentativas de remoção de um elemento que não existe na árvore também devem ser previstas e devidamente ignoradas.
4. A maneira mais prática de atender aos itens 2 e 3 é realizar uma chamada prévia à função BUSCAR. Entretanto, isso **não será permitido**. Vocês devem implementar a inserção e a remoção de forma independente do algoritmo de busca, para que o seu algoritmo seja o mais eficiente possível.
5. A propósito, **não será permitido o uso de estruturas de dados prontas de uma biblioteca** qualquer, especificamente no que tange às estruturas de dados em árvores. Ou seja, vocês terão que implementar a ABB. Quase todas as operações descritas são implementadas mais facilmente de maneira recursiva. Entretanto, vocês são livres para adotar a abordagem recursiva ou não.
6. Não serão aceitos trabalhos que não compilam ou não executam.
7. A linguagem de programação é livre. Vocês devem submeter um README com instruções para compilação e execução do seu programa, incluindo instalação de software necessário para execução em máquina com Ubuntu.
8. A submissão dos trabalhos será via Sigaa, até as 23:59h do dia 30 de setembro de 2019, contendo:
    - código fonte
    - README (ver item 7)
    - Breve relatório contendo os nomes dos integrantes do grupo, descrevendo sucintamente sua abordagem de solução e contendo análise de complexidade assintótica dos métodos implementados.
9. A implementação terá peso 2 para a segunda unidade (valerá 2,0 pontos para a segunda unidade) e a análise de complexidade 1,0 ponto extra para a primeira unidade. Trabalhos nos quais forem identificados plágio, não receberão pontuação e a nota da prova da segunda unidade será, no máximo, 8,0.
10. O trabalho será realizado em grupos de, no máximo, 3 pessoas.