INF 213 - Roteiro da Aula Pratica 11

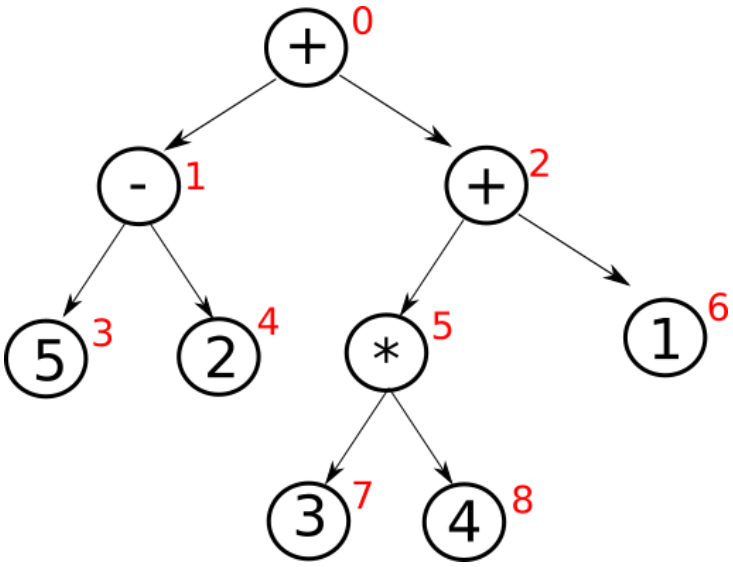
Objetivo: praticar operacoes com arvores.

→ LEMBREM-SE DE USAR PAPEL E CANETA COMO RASCUNHO ANTES DE IMPLEMENTAR <<--

Arquivos fonte e diagramas utilizados nesta aula:
https://drive.google.com/file/d/1HpTgFE5sHbVqFGoaz-TG4MmMU5Z19bXl/view?usp=sharing

Etapa 1

Considere uma expressão matemática formada apenas por operadores binários (por simplicidade, apenas +, - e *). Tal expressão poderia ser representada por uma árvore binária onde os nodos internos são os operadores e cada subárvore é uma sub-expressão. Os "operadores" nas folhas são expressões "finais" (i.e., números). Por exemplo, considere a árvore abaixo (os números em vermelho servem apenas para identificar os nodos):



(Note que tal árvore NÃO é uma árvore de pesquisa: por que?)

O valor da expressão representada pela árvore seria o valor da expressão que começa em sua raiz (nodo com número 0). Por sua vez, tal valor seria a soma do valor da subárvore esquerda (1) com o da direita (2) e assim por diante. A árvore acima poderia representar a expressão: (5-2)+((3*4)+1) (que vale 16).

O programa arvoreExpressao.cpp cria uma árvore como a acima a partir de um arquivo de texto descrevendo uma árvore. Esse arquivo começa com o número de vértices V da árvore. A seguir, há V linhas cada uma descrevendo um vértice (cada linha contém 4 valores, separados por um espaço em branco: 3 números e um caractere). Em tal arquivo, cada número (entre 0 e |v|-1) identifica um nodo. Por exemplo, a linha "0 1 2 +" indica que o filho esquerdo do nodo 0 é o 1, o da direita é o 2 e o nodo 0 contém o operador "+". Para simplificar, os números contidos

nas folhas terão sempre um caractere e seus filhos serão representados pelo número -1 (obs: as árvores de entrada serão sempre cheias).

```
9
0 1 2 +
1 3 4 -
2 5 6 +
3 -1 -1 5
4 -1 -1 2
5 7 8 *
6 -1 -1 1
7 -1 -1 3
8 -1 -1 4
```

A classe ArvoreExpressao representa uma árvore de expressões (conforme descrito acima) utilizando vetores do tipo MyVec (note que, como a árvore terá sempre um tamanho pré-determinado e estático, não usaremos ponteiros, alocação dinâmica de memória, etc). Essa implementação será mais simples do que a da árvore binária de pesquisa vista em sala.

Os vetores *filhoEsquerdo* e *filhoDireito* armazenam, respectivamente, os identificadores do filho esquerdo e direito de cada nodo (a posição “i” dos vetores armazena os dados do vértice “i”). O vetor *operador* armazena o operador contido no nodo (ou o número, no caso de folhas).

Por exemplo, a árvore acima seria representada da seguinte forma:

Posição:	0	1	2	3	4	5	6	7	8
filhoEsquerdo	1	3	5	-1	-1	7	-1	-1	-1
filhoDireito	2	4	6	-1	-1	8	-1	-1	-1
operador	+	-	+	5	2	*	1	3	4

Na etapa 1 você deverá implementar o método leArvore(), que le um arquivo no formato especificado acima e preenche os vetores (conforme descrito acima).

Etapa 2

Escreva em um arquivo README.txt os vértices da árvore apresentada na etapa 1 supondo um caminhamento pré-ordem, in-ordem e pós-ordem.

Etapa 3

Implemente a função “geraExpressao()” que imprime (na saída padrão) a expressão matemática representada pela árvore. Por exemplo, no caso acima sua função deveria imprimir uma linha contendo (5-2)+((3*4)+1) . Note que, para facilitar, você deverá colocar parênteses

em cada operando (exceto se tal operando for um número). Assim, o resultado poderia ser avaliado corretamente sem considerar a associatividade ou precedência dos operadores. Você deverá seguir o formato exato descrito acima. Assuma que a árvore terá pelo menos um nodo.

Etapa 4

Implemente a função “avaliaValor()”, que avalia a expressão definida por uma árvore e, então, retorna seu valor inteiro. Obs: nesta etapa não use o resultado da etapa 3 -- trabalhe diretamente na árvore. Assuma que a árvore terá pelo menos um nodo.

Etapa 5

Implemente as funções “altura()” e “nivelMaisNodos()” que retornam, respectivamente, qual a altura da árvore e qual nível possui mais nodos (no caso de empate retorne o menor deles). No exemplo acima, a árvore possui altura 3 e o nível com mais nodos é o nível 3 (ele tem 4 nodos). Dica: use recursividade! (ambas funções devem possuir complexidade linear no número de vértices)

Obs: para a função nivelMaisNodos, se houver empate retorne o primeiro nível encontrado (com mais nodos). Assuma que a árvore terá pelo menos um nodo.

Submissao da aula pratica:

A solucao deve ser submetida ate as 18 horas da proxima Segunda-Feira utilizando o sistema submitty (submitty.dpi.ufv.br). Envie todos os arquivos fonte (tanto os arquivos .h e .cpp fornecidos neste laboratorio quanto os que você implementou). Atualmente a submissao so pode ser realizada dentro da rede da UFV.