Aula Prática: Árvores Binárias

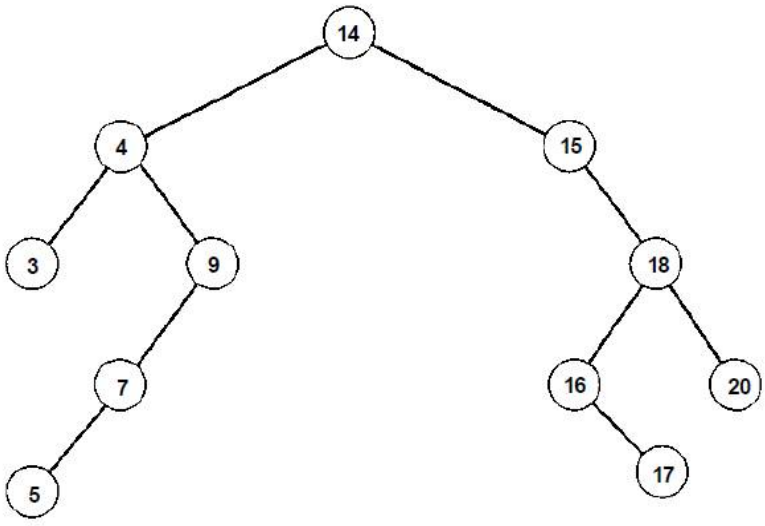
Árvores Binárias

As árvores são estruturas de dados baseadas em listas encadeadas que possuem um nó superior (raiz) que aponta para outros nós (filhos) que podem ser pais de outros nós. Embora existam muitos tipos de árvores, as árvores binárias são especiais pois, quando ordenadas, conduzem a pesquisas, inserçoes e exclusões de conteúdo com eficiência e velocidade.

Uma **árvore de busca binária** tem as propriedades:

- todos os elementos na subárvore esquerda de um nó *n* são menores que *n*
- todos os elementos na subárvore direita de *n* são maiores ou iguais a *n*.

Veja o caso da da árvore de busca binária abaixo, construída a partir dos números: 14,15, 4, 9, 7, 18, 3, 5, 16, 20, 17.



Se uma árvore de busca binária for percorrida em ordem simétrica (esquerda, raiz, direita) e o conteúdo de cada nó for impresso à medida que o nó for visitado, os números serão impressos em ordem ascendente. Para a árvore acima o resultado seria:

3 4 5 7 9 14 15 16 17 18 20

Podemos definir um tipo algébrico **ArvBinInt** em Haskell para armazenar números inteiros numa árvore binária:

```
data ArvBinInt = Nil |
                 NoInt Int ArvBinInt ArvBinInt
                                  deriving Show
```

A definição de **ArvBinInt** é recursiva. Podemos observar que uma árvore é composta de um nó, identificado por uma etiqueta (NoInt), um elemento inteiro e duas sub-árvores (direita e esquerda). Ainda, a árvore pode ser nula.

A constante **arvDados** representa a árvore binária da figura anterior:

```
arvDados::ArvBinInt
arvDados = NoInt 14
              (NoInt 4 (NoInt 3 Nil Nil)
                       (NoInt 9 (NoInt 7 (NoInt 5 Nil Nil)
                                         Nil)
                                Nil)
               )
               (NoInt 15 Nil (NoInt 18
                                (NoInt 16 Nil (NoInt 17 Nil Nil))
                                (NoInt 20 Nil Nil))
               )
```

Funções para Manipulação de árvores:

As **folhas** de uma árvore binária são aqueles nós que não contém filhos nem à esquerda, nem à direita. A função abaixo lista os nós folhas de uma árvore binária de inteiros:

```
folhas::ArvBinInt -> [Int]
folhas Nil = []
folhas (NoInt n Nil Nil) = [n]
folhas (NoInt _ esq dir) = folhas esq ++ folhas dir
```

Para **somar** os elementos registrados numa árvore binária podemos utilizar a função abaixo:

```
soma::ArvBinInt -> Int
soma Nil = 0
soma (NoInt n Nil Nil) = n
soma (NoInt n esq dir) = n + soma esq + soma dir
```

Para verificar se um elemento **pertence** à uma árvore binária temos a seguinte função:

```
pertenceArv::Int -> ArvBinInt -> Bool
pertenceArv x Nil = False
pertenceArv x (NoInt v esq dir) = x==v || if x<v
                                  then (pertenceArv x esq)
                                  else (pertenceArv x dir)
```

A função **emOrdem** possibilita percorrer todos os elementos armazenados na árvore binária, segundo o critério : *esquerda - nó - direita*

```
emOrdem::ArvBinInt -> [Int]
emOrdem Nil = []
emOrdem (NoInt n esq dir) = emOrdem esq ++ [n] ++ emOrdem dir
```

Exercícios:

1) Defina um tipo algébrico para uma árvore binária de caracteres, e faça em seguida uma função que retorna uma lista de caracteres com todos os nós da árvore.

2) Defina funções para percorrer uma árvore binária de caracteres usando os critérios pré-ordem (raiz – esquerda – direita) e pós-ordem (esquerda – direita - raiz)

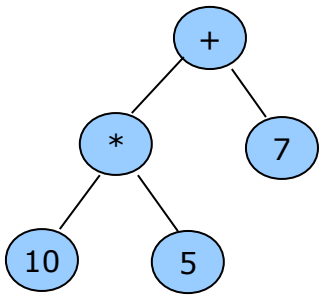
3) Duas árvores binárias são *semelhantes* se ambas estiverem vazias, ou se forem não-vazias e suas subárvores esquerdas e subárvores direitas forem semelhantes. Escreva um algoritmo para determinar se duas árvores binárias são semelhantes.

4) Uma árvore binária pode ser utilizada para armazenar expressões aritméticas. Para isso definimos o tipo ArvBinEA em que uma árvore pode ser vazia, conter um valor numérico ou conter uma expressão com um operador e outras duas expressões:

```
data ArvBinEA a = Vazia |
                  Folha a |
                  NoEA (Char, ArvBinEA a, ArvBinEA a)
                     deriving (Show)
```

A partir deste novo tipo podemos definir a expressão:

```
ea::ArvBinEA Float
ea = NoEA ('+', NoEA ('*', Folha 10, Folha 5), Folha 7)
```



Faça uma função que receba uma árvore binária de expressão aritmética e retorne o resultado da expressão. Por exemplo, dada a árvore `ea` desenhada acima, a função deve devolver o valor 57.