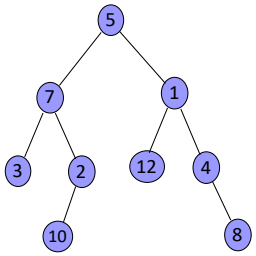
Travessias de árvores binárias

```
arv = (Node 5 (Node 7 (Node 3 Empty Empty)
                      (Node 2 (Node 10 Empty Empty) Empty)
              )
              (Node 1 (Node 12 Empty Empty)
                      (Node 4 Empty (Node 8 Empty Empty))
              )
      )
```




```
preorder arv  = [5,7,3,2,10,1,12,4,8]

inorder arv   = [3,7,10,2,5,12,1,4,8]

postorder arv = [3,10,2,7,12,8,4,1,5]
```

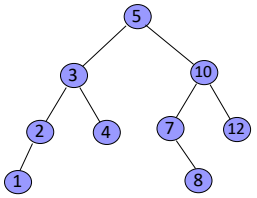

Árvores binárias de procura

Uma árvore binária em que o valor de cada nodo é maior do que os nodos à sua esquerda, e menor do que os nodos à sua direita diz-se uma árvore binária de procura (ou de pesquisa)

Uma **árvore binária de procura** é uma árvore binária que verifica as seguinte condição:

- a raiz da árvore é maior do que todos os elementos que estão na sub-árvore esquerda;
- a raiz da árvore é menor do que todos os elementos que estão na sub-árvore direita;
- ambas as sub-árvores são árvores binárias de procura.

Exemplo: Esta é uma árvore binária de procura de procura



Árvores binárias de procura

Exemplo: Testar se um elemento pertence a uma árvore binária de procura.

```
elemBT :: Ord a => a -> BTree a -> Bool
elemBT x Empty = False
elemBT x (Node y e d)
          | x < y  = elemBT x e
          | x > y  = elemBT x d
          | x == y = True
```

Exemplo: Inserir um elemento numa árvores binária de procura.

```
insertBT :: Ord a => a -> BTree a -> BTree a
insertBT x Empty = Node x Empty Empty
insertBT x (Node y e d)
          | x < y  = Node y (insertBT x e) d
          | x > y  = Node y e (insertBT x d)
          | x == y = Node y e d
```

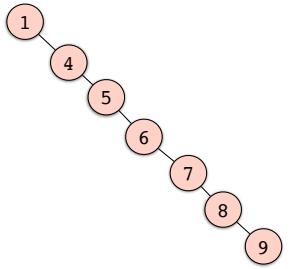

Árvores binárias de procura

O formato de uma árvore depende da ordem pela qual os elementos vão sendo inseridos.

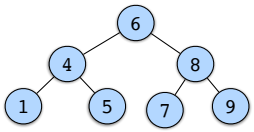
Exemplo: Considere a seguinte função que converte uma lista numa árvore, inserindo os elementos pela a ordem em que estes aparecem na lista.

```
listToBT :: Ord a => [a] -> BTree a
listToBT l = foldl (flip insertBT) Empty l
```

```
listToBT [1,4,5,6,7,8,9] =
```




```
listToBT [6,4,1,8,9,5,7] =
```

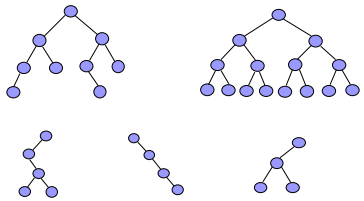


Árvores balanceadas

Uma árvore binária diz-se **balanceada** (ou **equilibrada**) se é vazia, ou se verifica as seguintes condições:

- as alturas das sub-árvores esquerda e direita diferem no máximo em uma unidade;
- ambas as sub-árvores são balanceadas.

Exemplos:



Balanceadas.

Desbalanceadas.

Exercício: Defina uma função que testa se uma árvore é balanceada.

Árvores binárias de procura

As árvores binárias de procura possibilitam pesquisas potencialmente mais eficientes do que as listas.

Exemplo:

Pesquisa numa lista não ordenada.

```
lookup :: Eq a => a -> [(a,b)] -> Maybe b
lookup x [] = Nothing
lookup x ((y,z):t) | x == y = Just z
                   | x /= y = lookup x t
```

Pesquisa numa lista ordenada.

```
lookupSL :: Ord a => a -> [(a,b)] -> Maybe b
lookupSL x [] = Nothing
lookupSL x ((y,z):t) | x < y  = Nothing
                     | x == y = Just z
                     | x > y  = lookupSL x t
```

O número de comparações de chaves é **no máximo igual ao comprimento da lista.**

Pesquisa numa árvore binária de procura.

```
lookupBT :: Ord a => a -> BTree (a,b) -> Maybe b
lookupBT x Empty = Nothing
lookupBT x (Node (y,z) e d) | x < y  = lookupBT x e
                            | x > y  = lookupBT x d
                            | x == y = Just z
```

O número de comparações de chaves é **no máximo igual à altura da árvore**.

Árvores binárias de procura

A pesquisa em árvores binárias de procura são especialmente mais eficientes se as árvores forem balanceadas.

Uma forma de balancear uma árvore de procura consiste em: primeiro gerar uma lista ordenada com os seus elementos e depois, a partir dessa lista, gerar a árvore.

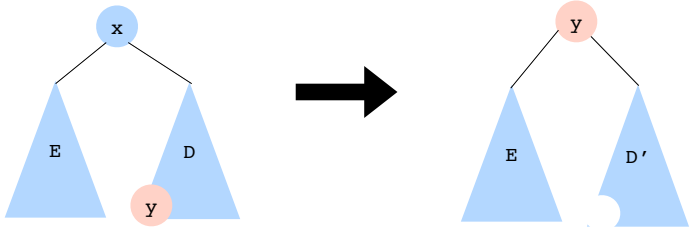
```
balance :: BTree a -> BTree a
balance t = constroi (inorder t)

constroi :: [a] -> BTree a
constroi [] = Empty
constroi l = let n = length l
                 (l1, x:l2) = splitAt (n `div` 2) l
             in Node x (constroi l1) (constroi l2)
```

Exercício: Defina uma versão mais eficiente desta função que não esteja a calcular sempre o comprimento da lista. (Sugestão: trabalhe com um par que tem a lista e o seu comprimento.)

Árvores binárias de procura

A remoção do elemento que está na raiz de uma árvore de procura pode ser feita indo buscar o menor elemento da sub-árvore direita (ou, em alternativa, o maior elemento da sub-árvore esquerda) para tomar o seu lugar.



Exercício: Com base nesta ideia, defina uma função que remove um elemento de uma árvore de procura. Comece por definir uma função que devolve um par com o mínimo de uma árvore não vazia e a árvore sem o mínimo.