Пусть задан тип данных для представления двоичных деревьев:

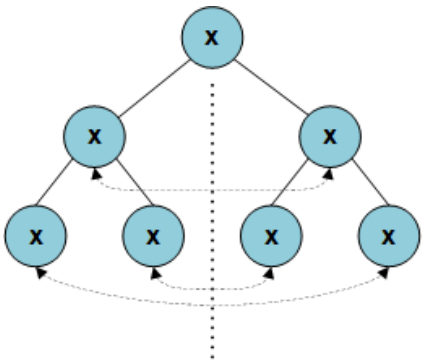
```
data Tree a = Empty | Node a (Tree a) (Tree a)
                                  deriving (Show, Read, Eq)
```

Двоичными деревьями поиска называются деревья, у которых для *каждого* элемента:

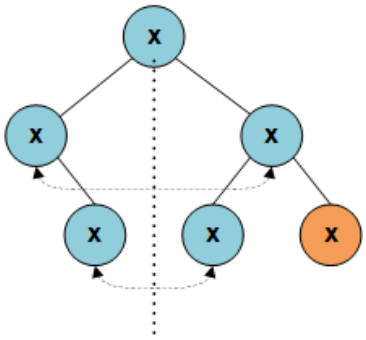
- в левом поддереве находятся меньшие элементы
- в правом поддереве находятся большие элементы

Пример реализации функций:
`singleton :: a -> Tree a`
`treeInsert :: (Ord a) => a -> Tree a -> Tree a`
`treeElem :: (Ord a) => a -> Tree a -> Bool`
можете посмотреть здесь: http://learnyouahaskell.com/making-our-own-types-and-typeclasses#recursive-data-structures (Обратите внимание, что тип данных `Tree` в книжке немного отличается)

1. `[1 балл]` Напишите функцию, которая проверяет является ли дерево симметричным. Симметричным считается дерево, у которого левое и правое поддеревья являются зеркальным отражением друг друга. При этом проверяется только структура дерева, значения в узлах не учитываются.



симметричное дерево



не симметричное дерево

2. `[1 балл]` Напишите функцию, которая строит двоичное дерево поиска из списка значений.
3. `[1 балл]` Напишите функцию, которая удалит минимальное значение из двоичного дерева поиска. Функция должна вернуть пару: новое дерево (после удаления) и минимум (значение в узле дерева, который был удален). Функция вызывает ошибку при применении к пустому дереву.

```
deleteMin :: (Ord a) => Tree a -> (Tree a, a)
deleteMin = undefined
```

Например:

```
ghci> deleteMin (Node 2 (Node 1 Empty Empty) (Node 3 Empty Empty))
```

```
(Node 2 Empty (Node 3 Empty Empty), 1)
```

4. `[2 балла]` Напишите функцию, которая удалит указанное значение из двоичного дерева поиска. Полученное в результате дерево должно быть двоичным деревом поиска. Если указанного значения в дереве нет, то оно не изменяется.
   Подсказка: возможно вам пригодится предыдущая функция. (Балансировать дерево НЕ нужно)

```
deleteValue :: (Ord a) => a -> Tree a -> Tree a
deleteValue = undefined
```

Например:

```
ghci> deleteValue 1 (Node 2 (Node 1 Empty Empty) (Node 3 Empty Empty))
Node 2 Empty (Node 3 Empty Empty)
ghci> deleteValue 2 (Node 2 (Node 1 Empty Empty) (Node 3 Empty Empty))
Node 3 (Node 1 Empty Empty) Empty
ghci> deleteValue 3 (Node 2 (Node 1 Empty Empty) (Node 3 Empty Empty))
Node 2 (Node 1 Empty Empty) Empty
ghci> deleteValue 4 (Node 2 (Node 1 Empty Empty) (Node 3 Empty Empty))
Node 2 (Node 1 Empty Empty) (Node 3 Empty Empty)
```

Дополнительное задание `[1-2 бонусных балла]`
Придумайте, как красиво и понятно отобразить двоичное дерево:

```
instance Show a => Show (Tree a) where
        show = undefined
```

За это задание вы можете получить до 2 бонусных баллов: количество зависит от "красоты и понятности" (определяется субъективным восприятием преподавателя)