

1. Напишете функция, която проверява дали дадено двоично дърво е двоично дърво за търсене
2. Напишете функция, която връща най-малкия и най-големия елемент в двоично дърво за търсене
3. Конвертирайте сортиран списък в двоично дърво за търсене
4. Намерете k-тия най-малък елемент в двоично дърво за търсене
5. Намерете сумата от елементите в интервал $[a,b]$ в двоично дърво за търсене
6. Напишете шаблонен клас `TreeTransformer`, който да е аналог на `map` за списъци.

Имплементирайте следните наследници на класа:

- a. `MaxTransformer` - работи върху `BinTree<List<T>>` и трансформира до `BinTree<T>`. Резултата съдържа максималната стойност на списъците в оригиналното дърво
 - b. `AvgTransformer` - работи върху `BinTree<Query<T>>` и трансформира до `BinTree<T>`. Резултата съдържа средноаритметичната стойност на опашките в оригиналното дърво
 - c. `PathTransformer` - работи върху `BinTree<T>` и трансформира до `BinTree<PathInfo<T>>`. Резултата съдържа елемента и пътя от корена на дървото до него
7. Напишете шаблонен клас `TreeReducer`, който да е аналог на `reduce` за списъци. Имплементирайте следните наследници на класа:
 - a. `SumReducer` - работи върху `BinTree<T>` и трансформира до `T` - сумата от всички елементи на дървото
 - b. `SetReducer` - работи върху `BinTree<T>` и трансформира до `List<T>` - списък от всички уникални елементи на дървото

**Относно имплементацията на `TreeReducer` - обхождането има значение, тъй че при изпълнението на операцията `reduce` трябва да се укаже как да бъдат обходени елементите - `Left - Root - Right` | `Right - Root - Left` | `Root - Left - Right` | `Root - Right - Left`*

За `TreeTransformer`, обхождането на елементите няма значение