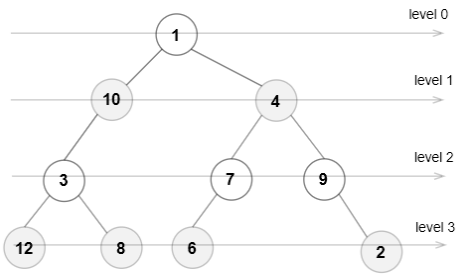
**** За решенията на задачи следва да се използват най-подходящите и близки по поведение структури от данни. Всички решенията, които не спазват добрите практики за работа със структури и ООП парадигмата се оценяват с 0 точки.***

Задача 1. (1.25 т.) Нека е дадено двоично дърво с елементи от тип T. Нека елементите на дървото се представя чрез следната структура:

```
template <typename T>
struct Node {T data; Node<T> *left, *right;};
```



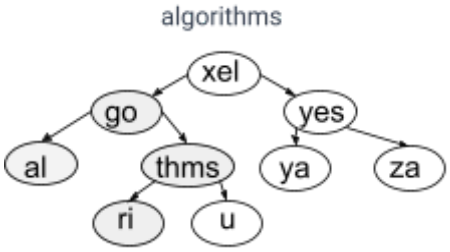
Едно двоично дърво наричаме четно-нечетно ако са изпълнени следните условия

- коренът на дървото се намира на ниво 0, а неговите наследници на ниво 1 и т.н.;
- на всяко четно ниво се намират само върхове от дървото с нечетна стойност, образуващи **строго** растяща наредба (от ляво на дясно);
- на всяко нечетно ниво се намират само върхове от дървото с четна стойност, образуващи **строго** намаляваща наредба (от ляво на дясно)

Напишете функция `isOddEven`, която по подаден указател към корен на дърво от тип `Node` определя дали то е четно-нечетно.

Задача 2. (1.25 т.) Да се напише функция, която проверява дали дадено двоично дърво е балансирано, т.е. е наредено дърво (дърво за търсене) и е балансирано (avl дърво). Дървото се подава с указател към корена от тип `Node`.

Задача 3. (1.5 т.) Дадено е двоично наредено дърво (дърво за търсене) от (под)низове. Низовете са добавени в дървото спрямо лексикографската им наредба. Казваме, че дума може да се прочете в дървото, ако е възможно да се образува при **обхождане на дървото в посока ляво - корен - дясно**. При обхождането не е позволено да има други поднизове, които прекъсват думата. Като се възползвате от свойствата и сложностите на операциите в дървото, да се напише функция `findWord`, която проверява дали дума може да се прочете в дървото и връща указател към елементът, който съдържа началото на думата.



* За решението **НЕ е** позволено използване на допълнителни структури.

***** Дървото се подава чрез указател към корен от тип `Node`