## День 18

Темы:

[Урок 49: Рекурсия.](https://www.youtube.com/watch?v=9Hs7DuIJ3LE)

**Задачи:**

1. Вам дан числовой массив произвольного размера. Посчитайте сумму чисел в этом массиве **не используя циклы** (необходимо использовать рекурсивные вызовы).

Использовать статическое поле для хранения суммы не рекомендуется, такое решение нельзя назвать полноценно рекурсивным.

**Пример:**

**int**[] numbers = {**1**, **10**, **1241**, **50402**, -**50**, **249**, **10215**, **665**, **2295**, **7**, **311**};

System.out.println(recursionSum(...)); // 65346

\*recursionSum(...) - это ваш рекурсивный метод, который вернет сумму чисел в массиве numbers.

2. Создайте рекурсивный метод, который принимает на вход единственный аргумент - число, и возвращает количество цифр 7 в этом числе. **Циклы использовать запрещено**, можно использовать только рекурсивные вызовы. Использовать статическое поле для хранения количества цифр не рекомендуется, такое решение нельзя назвать полноценно рекурсивным.

**Пример:**

System.out.println(count7(**717771237**)); // 5

\*В числе 717771237 ровно 5 раз используется цифра семь.

--------------------------------------------------------------------------------------------------------------------

\*Для того, чтобы тесты работали, методы recursionSum и count7 должны быть публичными.

3. Задача повышенной сложности. Эта задача потребует больше времени и больше самостоятельного поиска информации в Google. Если появляются вопросы или не получается решить: смело пишите в чат. Мы поможем!

Доп. статья:

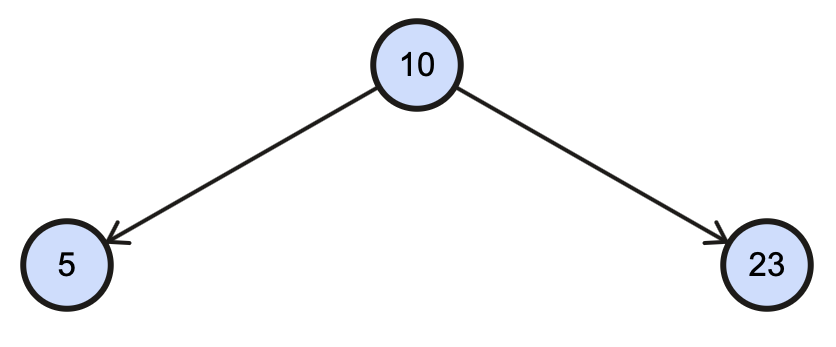
Статья про двоичные деревья:

[Теория + пример кода на kotlin](https://tproger.ru/translations/binary-search-tree-for-beginners/)

[Еще одна статья с теорией](https://javarush.ru/groups/posts/3111-strukturih-dannihkh-dvoichnoe-derevo-v-java)

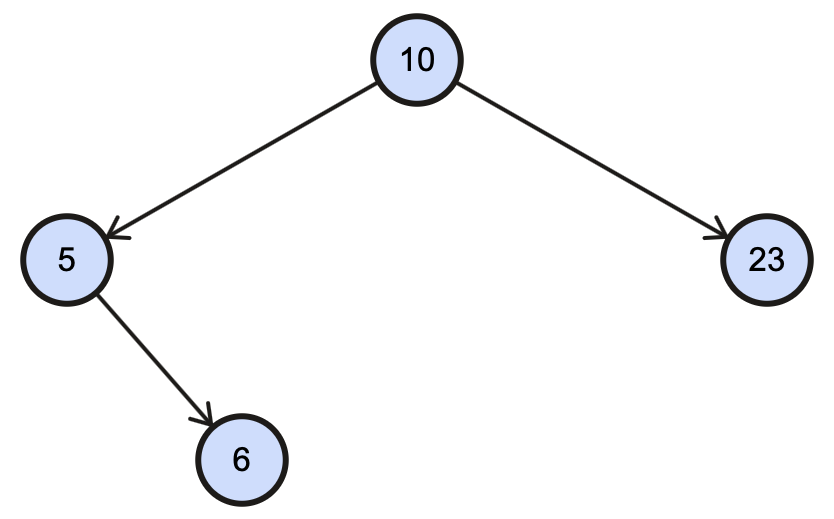
**Задача:**

Перед вами простейшее двоичное (иногда называют бинарное) дерево поиска. У этой структуры данных есть узлы (кружки с числами) и ребра (линии, соединяющие кружки). Узел с числом 10 называется родителем, а узлы 5 и 23 называются детьми (5 - левый сын, 23 - правый сын).



Основное свойство двоичного дерева поиска заключается в том, что значение в правом сыне всегда больше или равно значения в родительском узле, а значение в левом сыне всегда меньше родительского. Действительно, мы видим на примере выше, что правый сын содержит значение 23 (больше, чем у родителя), а левый сын содержит значение 5 (меньше, чем у родителя). Поэтому, перед нами корректное двоичное дерево поиска.

Мы можем добавлять сколько угодно значений в двоичное дерево поиска. Если мы захотим добавить в наше дерево новое значение 6, дерево изменится таким образом:



Логика добавления следующая:

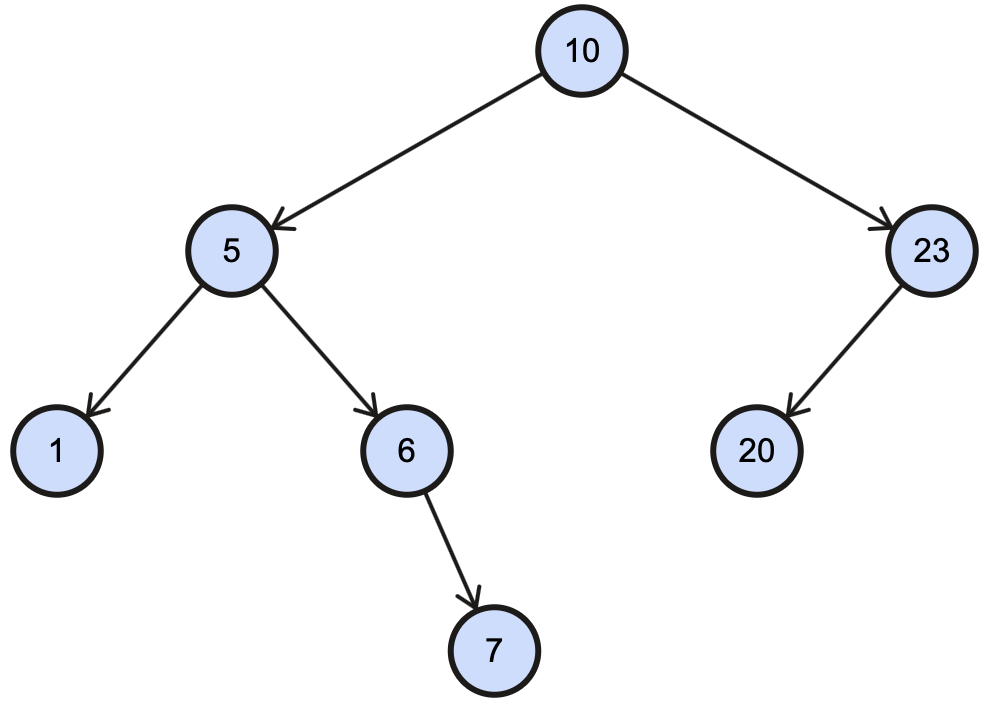
* 6 больше или меньше 10? -> меньше, поэтому должно уйти в левое поддерево
* 6 больше или меньше 5? -> больше, поэтому должно уйти в правое поддерево

В результате такого добавления, наше двоичное дерево осталось корректным. Для всех родителей и сыновей сохраняется свойство, описанное выше (левый сын всегда меньше родителя, правый сын больше или равен родителю).

Теперь в нашем дереве два родителя - узел 10 (родитель для узлов 5 и 23) и узел 5 (родитель для узла 6).

Давайте добавим еще три значения в наше двоичное дерево поиска. Добавим 1, 20 и 7.

Получим такое дерево:



Каждое из значений добавлялось по логике, описанной выше:

1 больше, чем 10? -> нет -> уходим влево

1 больше, чем 5? -> нет -> уходим влево

20 больше, чем 10? -> да -> уходим вправо

20 больше, чем 23? -> нет -> уходим влево

7 больше, чем 10? -> нет -> уходим влево

7 больше, чем 5? -> да -> уходим вправо

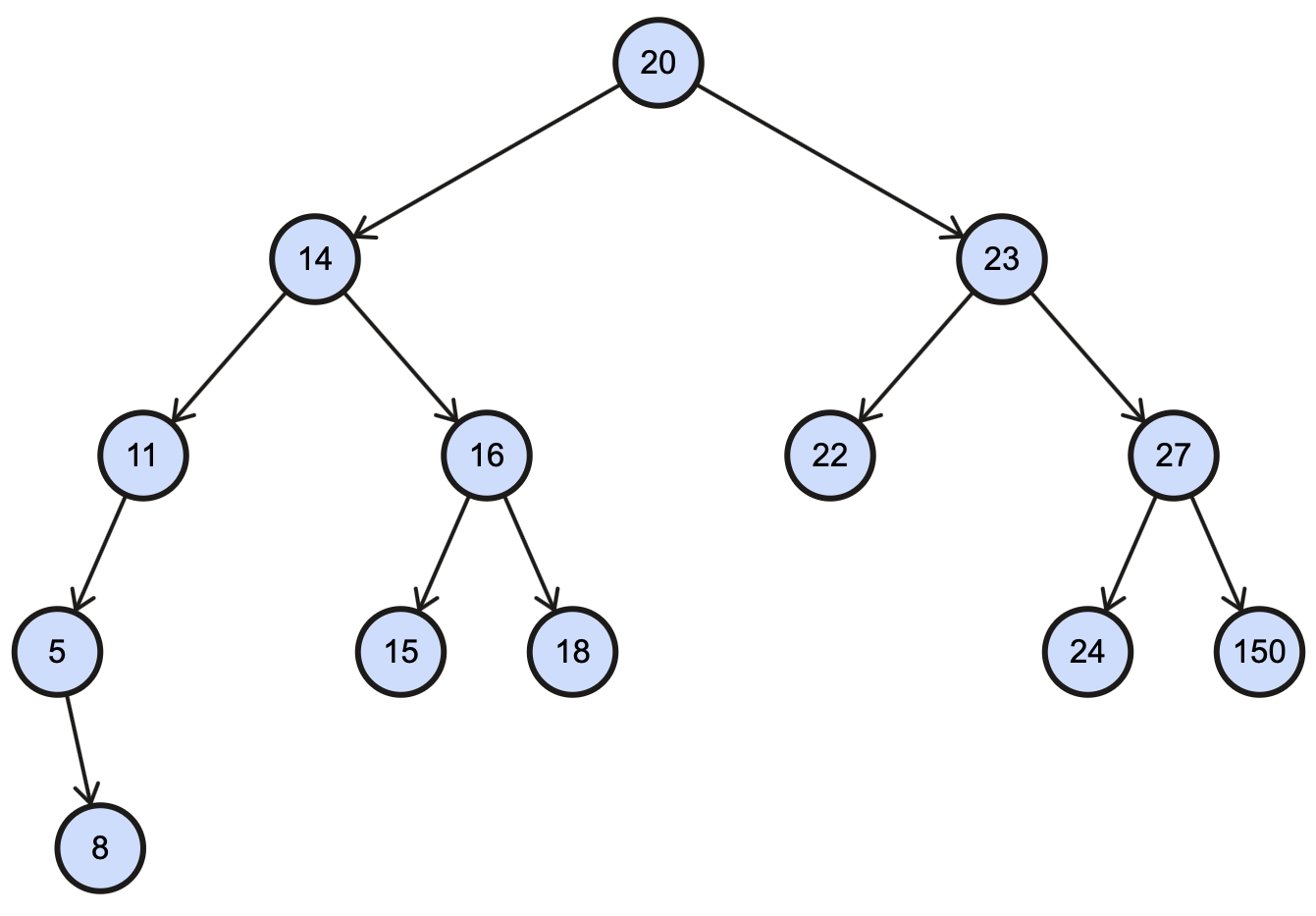
7 больше, чем 6? -> да -> уходим вправо

Вы можете потренироваться в добавлении значений в двоичное дерево поиска на этом сайте: <http://btv.melezinek.cz/binary-search-tree.html>

Теперь, в нашем дереве 4 родителя - узел 10 (для 5 и 23), узел 5 (для 1 и 6), узел 23 (для 20) и узел 6 (для 7).

Так как для всех родителей в этом дереве сохраняется основной закон двоичного дерева поиска (левый сын меньше, правый сын больше или равен), это двоичное дерево поиска корректное.

Очевидное свойство двоичного дерева поиска заключается в том, что поддеревья тоже являются двоичными деревьями поиска.



Вам необходимо создать двоичное дерево поиска, изображенное на картинке выше.

Для этого создайте класс “Узел” (англ. Node), объекты которого будут содержать само значение узла и поля-ссылки на два других узла (левый и правый сын).

Затем, создайте корневой (англ. root) узел (со значением 20).

После этого, необходимо реализовать метод, который будет добавлять новые узлы в ваше дерево.

Этот метод должен принимать в качестве аргументов добавляемое значение и ссылку на корень дерева. Проходясь по дереву, он должен вставлять новый узел в правильное место дерева.  
  
Когда двоичное дерево, изображенное выше, будет создано, необходимо используя рекурсию вывести в консоль все числа из этого двоичного дерева поиска в отсортированном виде. Ваше рекурсивное решение должно работать для любого корректного двоичного дерева поиска. Этот алгоритм называется “обход в глубину” (иногда называют “поиск в глубину”).

Ваш рекурсивный метод (пусть он будет называться dfs) должен в качестве единственного аргумента принимать ссылку на корень дерева (root).

Вызов: dfs(root);

Вывод в консоль: 5 8 11 14 15 16 18 20 22 23 24 27 150