2020-03-28 - Handout – Binary Tree / BST Algorithms

# Q1. Path Sum (and variations)

Link: <https://leetcode.com/problems/path-sum/>

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.

**Note:** A leaf is a node with no children.

**Example:**

Given the below binary tree and sum = 22, return true, as there exist a root-to-leaf path 5->4->11->2 which sum is 22.

**5**

**/** \

**4** 8

**/** / \

**11** 13 4

/ **\** \

7 **2** 1

**Follow-up (Path sum II):** what if you have to return all root to leaf paths which sum to the target sum? Return type should be List<List<Integer>>

**Follow-up (Path sum III):** Now, suppose that the path need not start at the root or end with a leaf (it should still flow down from the top of the tree towards child nodes). Return the number of such paths which sum to the target sum.

# Q2. Binary Search Tree Iterator

Link: <https://leetcode.com/problems/binary-search-tree-iterator/>

Implement an iterator over a binary search tree (BST). Your iterator will be initialized with the root node of a BST.

Calling next() will return the next smallest number in the BST.

**Example:**

****

BSTIterator iterator = new BSTIterator(root);

iterator.next(); // return 3

iterator.next(); // return 7

iterator.hasNext(); // return true

iterator.next(); // return 9

iterator.hasNext(); // return true

iterator.next(); // return 15

iterator.hasNext(); // return true

iterator.next(); // return 20

iterator.hasNext(); // return false

**Note:**

* next() and hasNext() should run in average O(1) time and uses O(*h*) memory, where *h* is the height of the tree.
* You may assume that next() call will always be valid, that is, there will be at least a next smallest number in the BST when next() is called.

# Q3.Binary Tree Right Side View

Link: <https://leetcode.com/problems/binary-tree-right-side-view/>

Given a binary tree, imagine yourself standing on the *right* side of it, return the values of the nodes you can see ordered from top to bottom.

**Example:**

**Input:** [1,2,3,null,5,null,4]

**Output:** [1, 3, 4]

**Explanation:**

1 <---

/ \

2 3 <---

\ \

5 4 <---

|  |  |
| --- | --- |
|  |  |

**Follow-up:** How would you find the left-side view?