Problem 1

Intuition

Table: Movies

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| movie_id      | int     |
| title         | varchar |
+---------------+---------+
```

movie_id is the primary key (column with unique values) for this table.
title is the name of the movie.

Table: Users

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| user_id       | int     |
| name          | varchar |
+---------------+---------+
```

user_id is the primary key (column with unique values) for this table.
The column 'name' has unique values.

Table: MovieRating

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| movie_id      | int     |
| user_id       | int     |
| rating        | int     |
| created_at    | date    |
+---------------+---------+
```

(movie_id, user_id) is the primary key (column with unique values) for this table.
This table contains the rating of a movie by a user in their review.
created_at is the user's review date.

Write a solution to:

- Find the name of the user who has rated the greatest number of movies. In case of a tie, return the lexicographically smaller user name.
- Find the movie name with the **highest average** rating in February 2020. In case of a tie, return the lexicographically smaller movie name.

The result format is in the following example.

Approach

First extract the name of person who rated maximum number of movies , then movie having maximum rating separately , then union them

Solution

```mysql
# Write your MySQL query statement below
select name as results from
(select count(m.user_id) as num, m.user_id, name
from MovieRating m left join Users u on m.user_id = u.user_id group by user_id
order by num desc, name limit 1) as temp
union all
select title as results from
(select m.user_id, avg(rating) as ave, title from MovieRating m left join Movies u on
u.movie_id = m.movie_id where year(created_at) = 2020
and month(created_at) = 2 group by m.movie_id order by ave desc, title limit 1) as temp
```

Problem 2

Intuition

Create a base class "Shape" with methods to calculate the area and perimeter (pure virtual). Implement derived classes "Rectangle" and "Circle" that inherit from "Shape" and provide their own area and perimeter calculations.

Solution

```
abstract class Shape{
    public abstract double area();
    public abstract double perimeter();
}
class Rectangle extends Shape{
    private int length;
    private int breadth;

    public Rectangle(int l, int b){
        length = l;
        breadth = b;
    }

    @Override
    public double area(){
        return length*breadth;
    }

    @Override
    public double perimeter(){
        return 2*(length + breadth);
    }
}
class Circle extends Shape{
```

```
    private int radius;

    public Circle(int r){
        radius = r;
    }

    @Override
    public double area(){
        return Math.PI * radius * radius;
    }

    @Override
    public double perimeter(){
        return 2*Math.PI*radius;
    }
}
class Main {
    public static void main(String[] args) {
        Rectangle rec = new Rectangle(2, 3);
        Circle cir = new Circle(5);

        System.out.println("Rectangle Area: "+rec.area());
        System.out.println("Rectangle Perimeter: "+rec.perimeter());
        System.out.println("Circle Area: "+cir.area());
        System.out.println("Circle Perimeter: "+cir.perimeter());
    }
}
```

Problem 3

Intuition

Given two strings s and t, return *the number of distinct* ***subsequences*** *of* s *which equals* t.

The test cases are generated so that the answer fits on a 32-bit signed integer.

Example 1:

Input: s = "rabbbit", t = "rabbit"
Output: 3
Explanation:
As shown below, there are 3 ways you can generate "rabbit" from s.
rabbb**it**
rab**bbit**
rabb**bit**

Example 2:

Input: s = "babgbag", t = "bag"
Output: 5
Explanation:
As shown below, there are 5 ways you can generate "bag" from s.
bab**g**bag
babgba**g**
babgba**g**
ba**b**gba**g**
babg**bag**

Constraints:

- 1 <= s.length, t.length <= 1000
- s and t consist of English letters.

Approach

For every character we have two choices in s, either we choose it or leave it, just follow this approach, and you will get a solution.

Solution

```
class Solution {
    public int numDistinct(String s, String t) {
        int sLen = s.length();
        int tLen = t.length();

        int[][] matrix = new int[tLen][sLen];
        matrix[tLen - 1][sLen - 1] = (s.charAt(sLen - 1) == t.charAt(tLen - 1)?1:0);
        for(int col = sLen - 2; col >= 0; col--){
            if(s.charAt(col) == t.charAt(tLen - 1))
```

```
            matrix[tLen - 1][col] = matrix[tLen - 1][col + 1] + 1;
            else
            matrix[tLen - 1][col] = matrix[tLen - 1][col + 1];
        }

        for(int row = tLen - 2; row >= 0; row--){
            for(int col = sLen - 2; col >= 0; col--){
                if(s.charAt(col) == t.charAt(row)){
                    matrix[row][col] = matrix[row][col + 1] + matrix[row + 1][col + 1];
                }else{
                    matrix[row][col] = matrix[row][col + 1];
                }
            }
        }
        return matrix[0][0];
    }
}
```

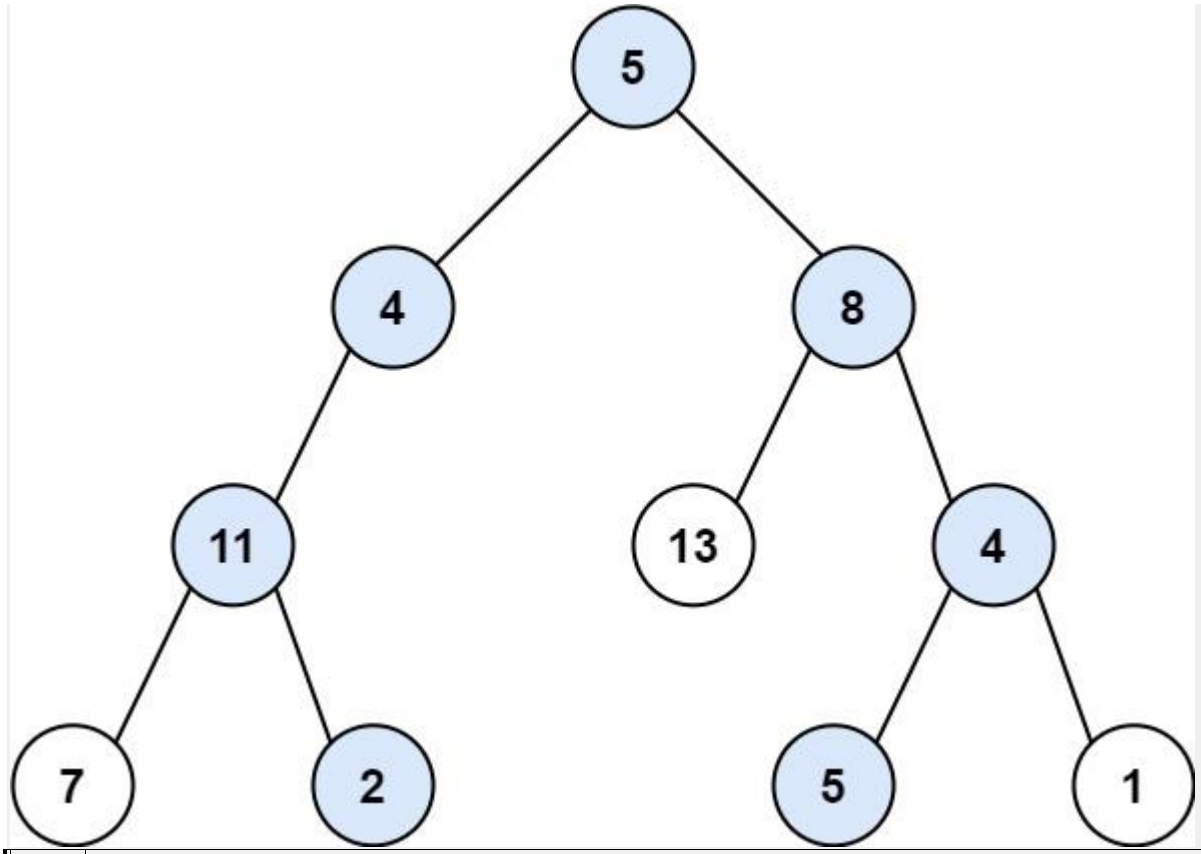
Problem 4

Intuition

Given the root of a binary tree and an integer targetSum, return *all **root-to-leaf** paths where the sum of the node values in the path equals* targetSum. *Each path should be returned as a list of the node **values**, not node references*.

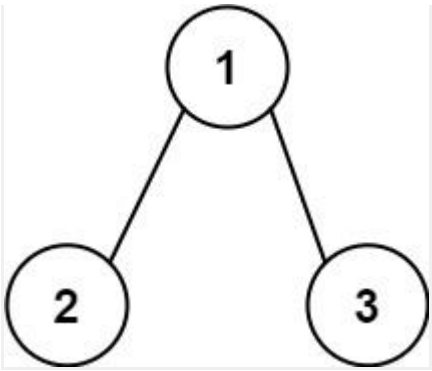
A **root-to-leaf** path is a path starting from the root and ending at any leaf node. A **leaf** is a node with no children.

Example 1:



Input: root = [5,4,8,11,null,13,4,7,2,null,null,5,1], targetSum = 22
Output: [[5,4,11,2],[5,8,4,5]]
Explanation: There are two paths whose sum equals targetSum:
5 + 4 + 11 + 2 = 22
5 + 8 + 4 + 5 = 22

Example 2:



Input: root = [1,2,3], targetSum = 5
Output: []

Example 3:

Input: root = [1,2], targetSum = 0
Output: []

Constraints:

- The number of nodes in the tree is in the range [0, 5000].
- -1000 <= Node.val <= 1000
- -1000 <= targetSum <= 1000

Approach

Perform depth first search and backtracking to find the solution, at each node reduce targetSum from node values if it is not leaf node, if it is leaf node just check whether it is equal to current target sum or not, if yes add current list to result set , else remove last added element and move to previous node and continue traversing.

Solution

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    List<List<Integer>> result = new ArrayList<>();
    public List<List<Integer>> pathSum(TreeNode root, int targetSum) {
        inorder(root, targetSum, new ArrayList<>());
        return result;
    }
    private void inorder(TreeNode root, int sum, List<Integer> temp){
        if(root == null) return;
        temp.add(root.val);

        if(root.left == null && root.right == null && sum == root.val){
            result.add(new ArrayList<>(temp));
            temp.remove(temp.size() - 1);
            return;
        }
        inorder(root.left, sum - root.val, temp);
```

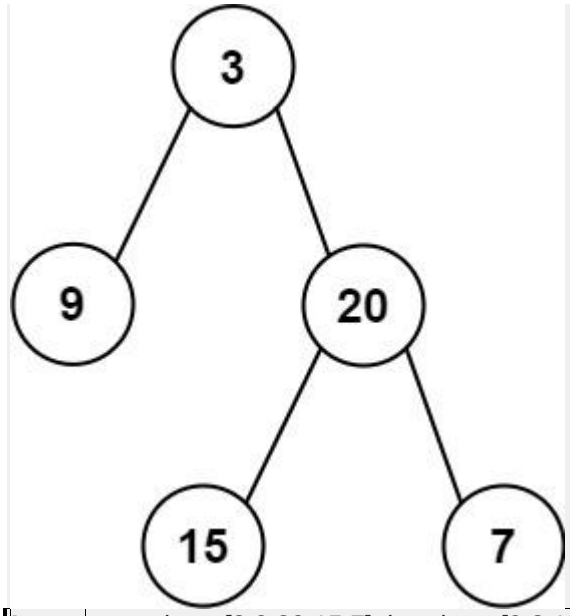
```
        inorder(root.right, sum - root.val, temp);
        temp.remove(temp.size() - 1);
    }
}
```

Problem 5

Intuition

Given two integer arrays preorder and inorder where preorder is the preorder traversal of a binary tree and inorder is the inorder traversal of the same tree, construct and return *the binary tree*.

Example 1:



Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]
Output: [3,9,20,null,null,15,7]

Example 2:

Input: preorder = [-1], inorder = [-1]
Output: [-1]

Constraints:

- 1 <= preorder.length <= 3000
- inorder.length == preorder.length
- -3000 <= preorder[i], inorder[i] <= 3000
- preorder and inorder consist of **unique** values.
- Each value of inorder also appears in preorder.
- preorder is **guaranteed** to be the preorder traversal of the tree.
- inorder is **guaranteed** to be the inorder traversal of the tree.

Approach

Preorder traversal gives the order of roots of tree, means if preorder is p1, p2, p3…, then we will first make root p1 in Inorder traversal (given) and make p1 left elements as left tree and right elements as right tree , then continue for p2, then p3

Solution

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    Map<Integer, Integer> mp = new HashMap<>();
    int index = 0;
    public TreeNode buildTree(int[] preorder, int[] inorder) {
        for(int i = 0; i < preorder.length; i++) mp.put(inorder[i], i);
        return splitTree(preorder, 0, preorder.length - 1);
    }
    private TreeNode splitTree(int[] preorder, int left, int right){
        if(left > right) return null;

        TreeNode root = new TreeNode(preorder[index]);

        int mid = mp.get(preorder[index++]);

        root.left = splitTree(preorder, left, mid - 1);
        root.right = splitTree(preorder, mid + 1, right);
        return root;
    }
}
```