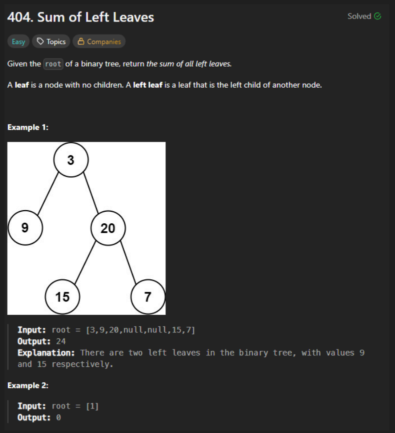
Weekly Contest - 6



404. Sum of Left Leaves

Solved ✓

Easy · Topics · Companies

Given the `root` of a binary tree, return *the sum of all left leaves.*

A **leaf** is a node with no children. A **left leaf** is a leaf that is the left child of another node.

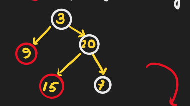
Example 1:

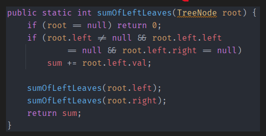
```
Input: root = [3,9,20,null,null,15,7]
Output: 24
Explanation: There are two left leaves in the binary tree, with values 9 and 15 respectively.
```

Example 2:

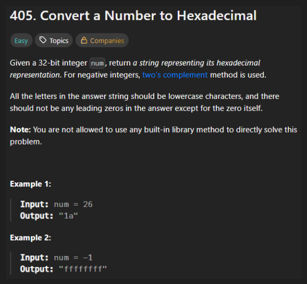
```
Input: root = [1]
Output: 0
```

Leaf → No left, No Right





```java
public static int sumOfLeftLeaves(TreeNode root) {
    if (root == null) return 0;
    if (root.left != null && root.left.left
            == null && root.left.right == null)
        sum += root.left.val;

    sumOfLeftLeaves(root.left);
    sumOfLeftLeaves(root.right);
    return sum;
}
```



405. Convert a Number to Hexadecimal

Easy · Topics · Companies

Given a 32-bit integer `num`, return *a string representing its hexadecimal representation*. For negative integers, two's complement method is used.

All the letters in the answer string should be lowercase characters, and there should not be any leading zeros in the answer except for the zero itself.

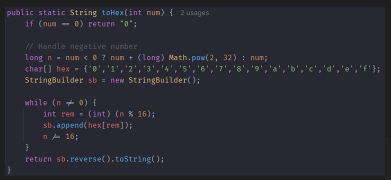
Note: You are not allowed to use any built-in library method to directly solve this problem.

Example 1:

```
Input: num = 26
Output: "1a"
```

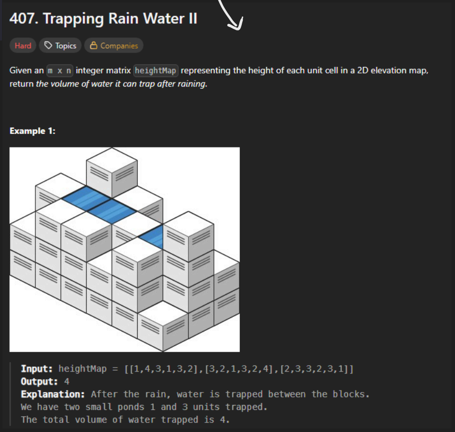
Example 2:

```
Input: num = -1
Output: "ffffffff"
```



```java
public static String toHex(int num) {  2 usages
    if (num == 0) return "0";

    // Handle negative number
    long n = num < 0 ? num + (long) Math.pow(2, 32) : num;
    char[] hex = {'0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f'};
    StringBuilder sb = new StringBuilder();

    while (n != 0) {
        int rem = (int) (n % 16);
        sb.append(hex[rem]);
        n /= 16;
    }
    return sb.reverse().toString();
}
```

Not Solved



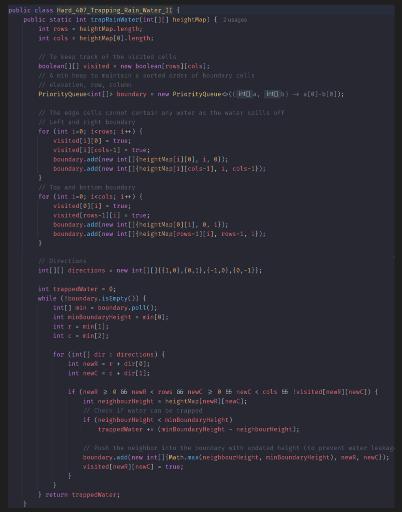
407. Trapping Rain Water II

Hard · Topics · Companies

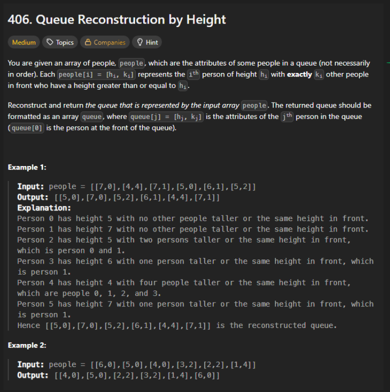
Given an `m x n` integer matrix `heightMap` representing the height of each unit cell in a 2D elevation map, return *the volume of water it can trap after raining.*

Example 1:

```
Input: heightMap = [[1,4,3,1,3,2],[3,2,1,3,2,4],[2,3,3,2,3,1]]
Output: 4
Explanation: After the rain, water is trapped between the blocks.
We have two small ponds 1 and 3 units trapped.
The total volume of water trapped is 4.
```



```java
public class Hard_407_Trapping_Rain_Water_II {
    public static int trapRainWater(int[][] heightMap) {  2 usages
        int rows = heightMap.length;
        int cols = heightMap[0].length;

        // To keep track of the visited cells
        boolean[][] visited = new boolean[rows][cols];
        // A min heap to maintain a sorted order of boundary cells
        // elevation, row, column
        PriorityQueue<int[]> boundary = new PriorityQueue<>((int[] a, int[] b) -> a[0]-b[0]);

        // The edge cells cannot contain any water as the water spills off
        // Left and right boundary
        for (int i=0; i<rows; i++) {
            visited[i][0] = true;
            visited[i][cols-1] = true;
            boundary.add(new int[]{heightMap[i][0], i, 0});
            boundary.add(new int[]{heightMap[i][cols-1], i, cols-1});
        }
        // Top and bottom boundary
        for (int i=0; i<cols; i++) {
            visited[0][i] = true;
            visited[rows-1][i] = true;
            boundary.add(new int[]{heightMap[0][i], 0, i});
            boundary.add(new int[]{heightMap[rows-1][i], rows-1, i});
        }

        // Directions
        int[][] directions = new int[][]{{1,0},{0,1},{-1,0},{0,-1}};

        int trappedWater = 0;
        while (!boundary.isEmpty()) {
            int[] min = boundary.poll();
            int minBoundaryHeight = min[0];
            int r = min[1];
            int c = min[2];

            for (int[] dir : directions) {
                int newR = r + dir[0];
                int newC = c + dir[1];

                if (newR >= 0 && newR < rows && newC >= 0 && newC < cols && !visited[newR][newC]) {
                    int neighbourHeight = heightMap[newR][newC];
                    // Check if water can be trapped
                    if (neighbourHeight < minBoundaryHeight)
                        trappedWater += (minBoundaryHeight - neighbourHeight);

                    // Push the neighbor into the boundary with updated height (to prevent water leakage
                    boundary.add(new int[]{Math.max(neighbourHeight, minBoundaryHeight), newR, newC});
                    visited[newR][newC] = true;
                }
            }
        } return trappedWater;
    }
}
```



406. Queue Reconstruction by Height

Medium · Topics · Companies · Hint

You are given an array of people, `people`, which are the attributes of some people in a queue (not necessarily in order). Each `people[i] = [hi, ki]` represents the i^{th} person of height h_i with **exactly** k_i other people in front who have a height greater than or equal to h_i.

Reconstruct and return *the queue that is represented by the input array* `people`. The returned queue should be formatted as an array `queue`, where `queue[j] = [hj, kj]` is the attributes of the j^{th} person in the queue (`queue[0]` is the person at the front of the queue).

Example 1:

```
Input: people = [[7,0],[4,4],[7,1],[5,0],[6,1],[5,2]]
Output: [[5,0],[7,0],[5,2],[6,1],[4,4],[7,1]]
Explanation:
Person 0 has height 5 with no other people taller or the same height in front.
Person 1 has height 7 with no other people taller or the same height in front.
Person 2 has height 5 with two persons taller or the same height in front, which is person 0 and 1.
Person 3 has height 6 with one person taller or the same height in front, which is person 1.
Person 4 has height 4 with four people taller or the same height in front, which are people 0, 1, 2, and 3.
Person 5 has height 7 with one person taller or the same height in front, which is person 1.
Hence [[5,0],[7,0],[5,2],[6,1],[4,4],[7,1]] is the reconstructed queue.
```

Example 2:

```
Input: people = [[6,0],[5,0],[4,0],[3,2],[2,2],[1,4]]
Output: [[4,0],[5,0],[2,2],[3,2],[1,4],[6,0]]
```

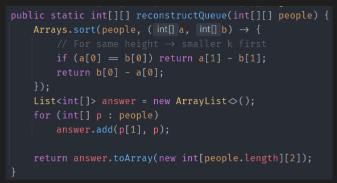
(7,0) , (4,4) , (7,1) , (5,0) , (6,1) , (5,2)

→ (7,0),(5,0) , (7,1) ,(6,1) , (5,2) , (4,4)

Sort p[1] increasing
p[0] decreasing

insert p at index p[1]

→ (5,0) ,(7,0) ,(6,1) ,(7,1) , (5,2) ,(4,4)



```java
public static int[][] reconstructQueue(int[][] people) {
    Arrays.sort(people, (int[] a, int[] b) -> {
        // For same height -> smaller k first
        if (a[0] == b[0]) return a[1] - b[1];
        return b[0] - a[0];
    });
    List<int[]> answer = new ArrayList<>();
    for (int[] p : people)
        answer.add(p[1], p);

    return answer.toArray(new int[people.length][2]);
}
```

407 →

Maximum amount of water
= Min of side - Elevation

Already visited

Min boundary

Height = H

H = 1

1	4	3	1	3	2
3	2	1	3	2	4
2	3	3	2	3	1

Boundary cell cannot contain water

min heap ① ~~1~~ ~~1~~ ~~2~~ ~~2~~ ~~2~~ 3
3 3 3 3 3 4 ,3

~~1~~	4	3	~~1~~	3	2
3	2	1	3	2	4
2	3	3	2	3	~~1~~

Check the nodes connected with min boundary height

1	4	3	1	3	~~2~~
3	2	1	3	2	4
~~2~~	3	3	~~2~~	3	1

ignore visited

H = 2

H = 3 water = 0 + (3-2) = 1

min ~~3~~ ~~3~~ ~~3~~ ~~3~~ ~~3~~ ~~3~~ ~~3~~ ,4

1	4	~~3~~	1	~~3~~	2
~~3~~ →	2	1	~~3~~	2	4
2	~~3~~	~~3~~	2	~~3~~	1

1 + (3-1) + (3-2) = 4

1	4	3	1	3	2
3	3	3	3	3	4
2	3	3	2	3	1

why water can be trapped?
neighbour < current H

why? since we are maintaining a min heap

all the next H >= current H