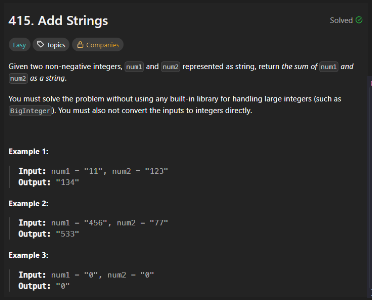
Weekly Contest - 8



415. Add Strings

Solved ✓

Easy · Topics · Companies

Given two non-negative integers, `num1` and `num2` represented as string, return *the sum of* `num1` *and* `num2` *as a string*.

You must solve the problem without using any built-in library for handling large integers (such as `BigInteger`). You must also not convert the inputs to integers directly.

Example 1:

> **Input:** num1 = "11", num2 = "123"
> **Output:** "134"

Example 2:

> **Input:** num1 = "456", num2 = "77"
> **Output:** "533"

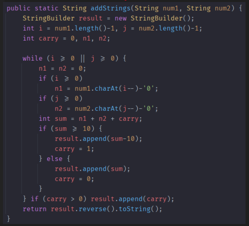
Example 3:

> **Input:** num1 = "0", num2 = "0"
> **Output:** "0"

For both numbers

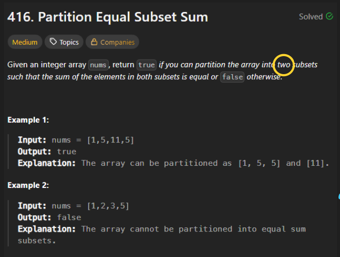
→ Go Right to left

→ Maintain Carry



```java
public static String addStrings(String num1, String num2) {
    StringBuilder result = new StringBuilder();
    int i = num1.length()-1, j = num2.length()-1;
    int carry = 0, n1, n2;

    while (i ≥ 0 || j ≥ 0) {
        n1 = n2 = 0;
        if (i ≥ 0)
            n1 = num1.charAt(i--)-'0';
        if (j ≥ 0)
            n2 = num2.charAt(j--)-'0';
        int sum = n1 + n2 + carry;
        if (sum ≥ 10) {
            result.append(sum-10);
            carry = 1;
        } else {
            result.append(sum);
            carry = 0;
        }
    } if (carry > 0) result.append(carry);
    return result.reverse().toString();
}
```



416. Partition Equal Subset Sum

Solved ✓

Medium · Topics · Companies

Given an integer array `nums`, return `true` *if you can partition the array into two subsets such that the sum of the elements in both subsets is equal or* `false` *otherwise.*

Example 1:

> **Input:** nums = [1,5,11,5]
> **Output:** true
> **Explanation:** The array can be partitioned as [1, 5, 5] and [11].

Example 2:

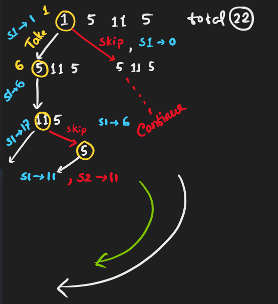
> **Input:** nums = [1,2,3,5]
> **Output:** false
> **Explanation:** The array cannot be partitioned into equal sum subsets.

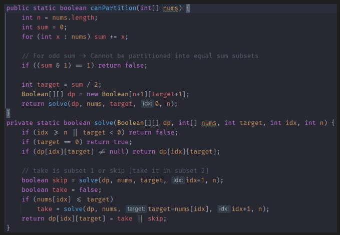
Dynamic Programming Approach

→ Calculate total

* Add current number to subset I

* Skip the number (Add to subset 2)



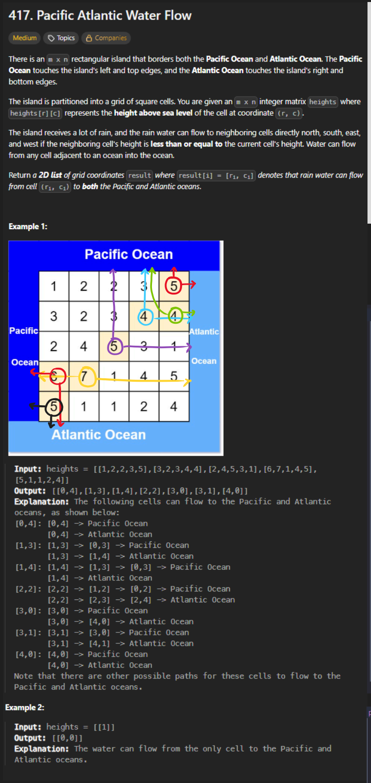


```java
public static boolean canPartition(int[] nums) {
    int n = nums.length;
    int sum = 0;
    for (int x : nums) sum += x;

    // For odd sum → Cannot be partitioned into equal sum subsets
    if ((sum & 1) == 1) return false;

    int target = sum / 2;
    Boolean[][] dp = new Boolean[n+1][target+1];
    return solve(dp, nums, target, idx: 0, n);
}
private static boolean solve(Boolean[][] dp, int[] nums, int target, int idx, int n) {
    if (idx ≥ n || target < 0) return false;
    if (target == 0) return true;
    if (dp[idx][target] ≠ null) return dp[idx][target];

    // take is subset 1 or skip [take it in subset 2]
    boolean skip = solve(dp, nums, target, idx: idx+1, n);
    boolean take = false;
    if (nums[idx] ≤ target)
        take = solve(dp, nums, target: target-nums[idx], idx: idx+1, n);
    return dp[idx][target] = take || skip;
}
```



417. Pacific Atlantic Water Flow

Medium · Topics · Companies

There is an `m x n` rectangular island that borders both the **Pacific Ocean** and **Atlantic Ocean**. The **Pacific Ocean** touches the island's left and top edges, and the **Atlantic Ocean** touches the island's right and bottom edges.

The island is partitioned into a grid of square cells. You are given an `m x n` integer matrix `heights` where `heights[r][c]` represents the **height above sea level** of the cell at coordinate `(r, c)`.

The island receives a lot of rain, and the rain water can flow to neighboring cells directly north, south, east, and west if the neighboring cell's height is **less than or equal to** the current cell's height. Water can flow from any cell adjacent to an ocean into the ocean.

Return *a* ***2D list*** *of grid coordinates* `result` *where* `result[i] = [r`$_i$`, c`$_i$`]` *denotes that rain water can flow from cell* `(r`$_i$`, c`$_i$`)` *to* ***both*** *the Pacific and Atlantic oceans.*

Example 1:

Pacific Ocean

1	2	2	3	5
3	2	3	4	4
2	4	5	3	1
6	7	1	4	5
5	1	1	2	4

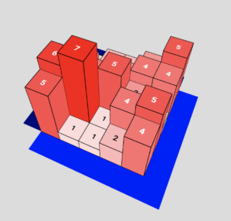
Pacific Ocean · Atlantic Ocean

Atlantic Ocean

```
Input: heights = [[1,2,2,3,5],[3,2,3,4,4],[2,4,5,3,1],[6,7,1,4,5],
[5,1,1,2,4]]
Output: [[0,4],[1,3],[1,4],[2,2],[3,0],[3,1],[4,0]]
Explanation: The following cells can flow to the Pacific and Atlantic
oceans, as shown below:
[0,4]: [0,4] -> Pacific Ocean
       [0,4] -> Atlantic Ocean
[1,3]: [1,3] -> [0,3] -> Pacific Ocean
       [1,3] -> [1,4] -> Atlantic Ocean
[1,4]: [1,4] -> [1,3] -> [0,3] -> Pacific Ocean
       [1,4] -> Atlantic Ocean
[2,2]: [2,2] -> [1,2] -> [0,2] -> Pacific Ocean
       [2,2] -> [2,3] -> [2,4] -> Atlantic Ocean
[3,0]: [3,0] -> Pacific Ocean
       [3,0] -> [4,0] -> Atlantic Ocean
[3,1]: [3,1] -> [3,0] -> Pacific Ocean
       [3,1] -> [4,1] -> Atlantic Ocean
[4,0]: [4,0] -> Pacific Ocean
       [4,0] -> Atlantic Ocean
Note that there are other possible paths for these cells to flow to the
Pacific and Atlantic oceans.
```

Example 2:

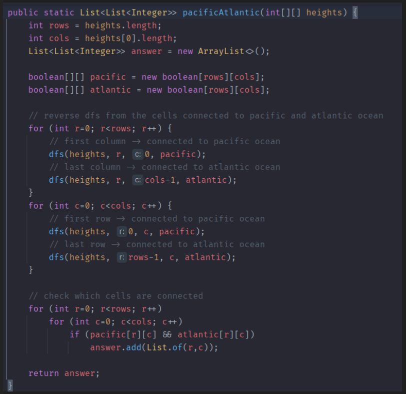
```
Input: heights = [[1]]
Output: [[0,0]]
Explanation: The water can flow from the only cell to the Pacific and
Atlantic oceans.
```



Reverse DFS → For both ocean

Start from the cells connected to ocean

DFS and check if the connected cells can be flooded



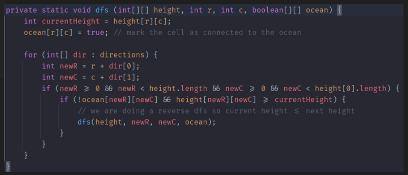
```java
public static List<List<Integer>> pacificAtlantic(int[][] heights) {
    int rows = heights.length;
    int cols = heights[0].length;
    List<List<Integer>> answer = new ArrayList<>();

    boolean[][] pacific = new boolean[rows][cols];
    boolean[][] atlantic = new boolean[rows][cols];

    // reverse dfs from the cells connected to pacific and atlantic ocean
    for (int r=0; r<rows; r++) {
        // first column → connected to pacific ocean
        dfs(heights, r, c: 0, pacific);
        // last column → connected to atlantic ocean
        dfs(heights, r, c: cols-1, atlantic);
    }
    for (int c=0; c<cols; c++) {
        // first row → connected to pacific ocean
        dfs(heights, r: 0, c, pacific);
        // last row → connected to atlantic ocean
        dfs(heights, r: rows-1, c, atlantic);
    }

    // check which cells are connected
    for (int r=0; r<rows; r++)
        for (int c=0; c<cols; c++)
            if (pacific[r][c] && atlantic[r][c])
                answer.add(List.of(r,c));

    return answer;
}
```



```java
private static void dfs (int[][] height, int r, int c, boolean[][] ocean) {
    int currentHeight = height[r][c];
    ocean[r][c] = true; // mark the cell as connected to the ocean

    for (int[] dir : directions) {
        int newR = r + dir[0];
        int newC = c + dir[1];
        if (newR ≥ 0 && newR < height.length && newC ≥ 0 && newC < height[0].length) {
            if (!ocean[newR][newC] && height[newR][newC] ≥ currentHeight) {
                // we are doing a reverse dfs so current height ≤ next height
                dfs(height, newR, newC, ocean);
            }
        }
    }
}
```