# 2025-01-18 - Handout – Backtracking Vs DFS

## Q1. Path Sum II

Link:https://leetcode.com/problems/path-sum-ii

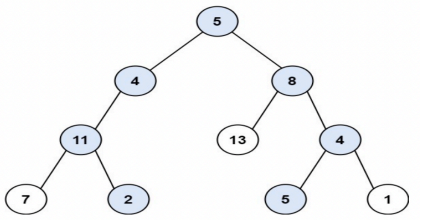
Given the root of a binary tree and an integer targetSum, return *all **root-to-leaf** paths where the sum of the node values in the path equals* targetSum*. Each path should be returned as a list of the node **values**, not node references*.

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.

**Constraints:**

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

Example 1:



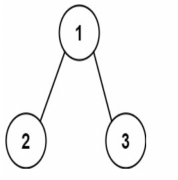

**Input:** root = [5,4,8,11,null,13,4,7,2,null,null,5,1], targetSum = 22

**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




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

**Output:** []

## Q2. Number of Islands

Link:https://leetcode.com/problems/number-of-islands

Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return *the number of islands*.

An **island** is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

**Constraints:**

- m == grid.length
- n == grid[i].length
- 1 <= m, n <= 300
- grid[i][j] is '0' or '1'.

**Example 1:**

```
Input: grid = [
  ["1","1","1","1","0"],
  ["1","1","0","1","0"],
  ["1","1","0","0","0"],
  ["0","0","0","0","0"]
]
Output: 1
```

**Example 2:**

```
Input: grid = [
  ["1","1","0","0","0"],
  ["1","1","0","0","0"],
  ["0","0","1","0","0"],
  ["0","0","0","1","1"]
]
Output: 3
```

## Q3. Word Search

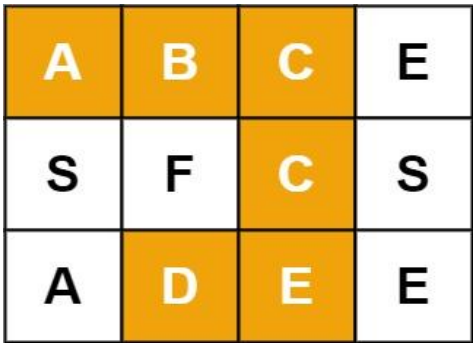
Link:https://leetcode.com/problems/word-search

Given an m x n grid of characters board and a string word, return true *if the word exists in the grid*. The word can be constructed from letters of sequentially adjacent cells, where adjacent cells are horizontally or vertically neighboring. The same letter cell may not be used more than once.

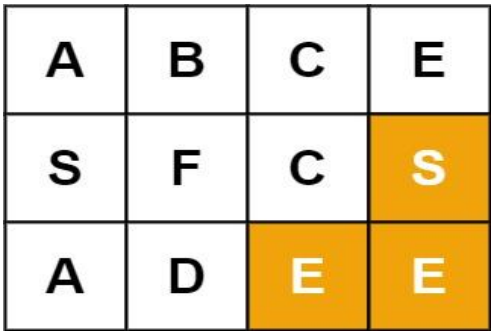
**Constraints:**

- m == board.length
- n = board[i].length
- 1 <= m, n <= 6
- 1 <= word.length <= 15
- board and word consists of only lowercase and uppercase English letters.

**Example 1:**




**Example 2:**



**Input: board =**

**[["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCCED"**

**Output: true**

**Input: board =**

**[["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "SEE"**

**Output: true**

### Q4. Robot Room Cleaner

Link:https://leetcode.com/problems/robot-room-cleaner

You are controlling a robot that is located somewhere in a room. The room is modeled as an m x n binary grid where 0 represents a wall and 1 represents an empty slot.

The robot starts at an unknown location in the room that is guaranteed to be empty, and you do not have access to the grid, but you can move the robot using the given API Robot.

You are tasked to use the robot to clean the entire room (i.e., clean every empty cell in the room). The robot with the four given APIs can move forward, turn left, or turn right. Each turn is 90 degrees.

When the robot tries to move into a wall cell, its bumper sensor detects the obstacle, and it stays on the current cell.

Design an algorithm to clean the entire room using the following APIs:

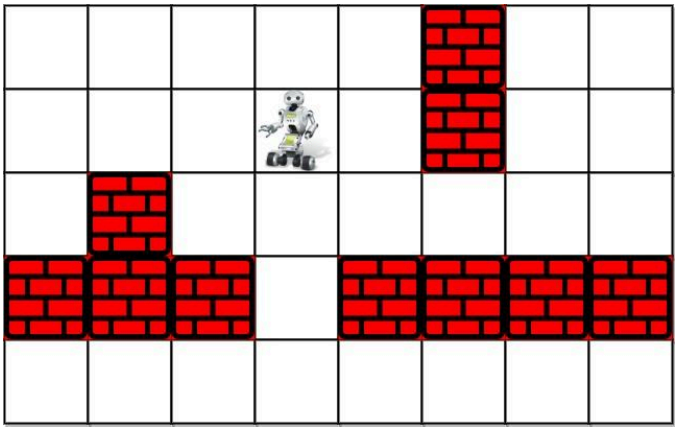
```
interface Robot {

  // returns true if next cell is open and robot moves into the cell.

  // returns false if next cell is obstacle and robot stays on the current cell.

  boolean move();

  // Robot will stay on the same cell after calling turnLeft/turnRight.

  // Each turn will be 90 degrees.

  void turnLeft();

  void turnRight();

  // Clean the current cell.

  void clean();

}
```

**Note** that the initial direction of the robot will be facing up. You can assume all four edges of the grid are all surrounded by a wall.

**Custom testing:**

The input is only given to initialize the room and the robot's position internally. You must solve this problem "blindfolded". In other words, you must control the robot using only the four mentioned APIs without knowing the room layout and the initial robot's position.

**Example 1:**



**Input: room = [[1,1,1,1,1,0,1,1],[1,1,1,1,1,1,0,1,1],[1,0,1,1,1,1,1,1,1],[0,0,0,1,0,0,0,0,0],[1,1,1,1,1,1,1,1,1]], row = 1, col = 3**

**Output: Robot cleaned all rooms.**

**Explanation: All grids in the room are marked by either 0 or 1.**

**0 means the cell is blocked, while 1 means the cell is accessible.**

**The robot initially starts at the position of row=1, col=3.**

**From the top left corner, its position is one row below and three columns right.**

**Example 2:**

**Input:** room = [[1]], row = 0, col = 0

**Output:** Robot cleaned all rooms.