```swift
import Foundation
/// **剑指 Offer 13. 机器人的运动范围**
/// 地上有一个 m 行 n 列的方格，从坐标 [0,0] 到坐标 [m-1,n-1] 。
/// 一个机器人从坐标 [0, 0]
    的格子开始移动，它每次可以向左、右、上、下移动一格（不能移动到方格外），
/// 也不能进入行坐标和列坐标的数位之和大于 k 的格子。
/// 例如，当 k 为18时，机器人能够进入方格 [35, 37] ，因为3+5+3+7=18。
/// 但它不能进入方格 [35, 38]，因为3+5+3+8=19。请问该机器人能够到达多少个格子?
///
/// 示例 1:
/// 输入: m = 2, n = 3, k = 1
/// 输出: 3
class Solution {
    private var m: Int = 0
    private var n: Int = 0
    private var k: Int = 0

    func movingCount(_ m: Int, _ n: Int, _ k: Int) -> Int {
        /// 存放所有可以移动到的点的索引
        var visited: [[Bool]] = Array(repeating: Array(repeating: false,
         count: n),
                                      count: m)
        (self.m, self.n, self.k) = (m, n, k)
        return dfs(0, 0, &visited)
    }
    /// 本题的思路应该是深度优先搜索算法 (DFS) 而不是回溯算法。
    /// 可以理解为暴力法模拟机器人在矩阵中的所有路径。DFS 通过递归，先朝一个方向搜到底，
    /// 再回溯至上个节点，沿另一个方向搜索，以此类推。根据可达解的结构和连通性，
    /// 易推出机器人可**仅通过向右和向下移动，访问所有可达解 **
    /// **时间复杂度 O(MN)** : 最差情况下，机器人遍历矩阵所有单元格，此时时间复杂度为 O(MN) 。
    /// **空间复杂度 O(MN)**: 所有单元格被访问，使用 O(MN) 的额外空间。
    /// - Parameters:
    ///   - row: 当前格子的行索引
    ///   - column: 当前格子的列索引
    ///   - visited: 表示机器人是否到达某一个格子的矩阵
    /// - Returns: 返回可以到达的点的数量
    private func dfs(_ row: Int, _ column: Int, _ visited: inout [[Bool]]) ->
     Int {
        /// 跳出递归的条件：行列不在范围内、已经访问过、行坐标和列坐标的数位之和大于 k
        if ((row >= self.m) || (row < 0) ||
            (column >= self.n) || (column < 0) ||
            (visited[row][column]) ||
            (sumRowColumn(row, column) > self.k)) { return 0 }
        visited[row][column] = true
        /// 继续向右和向下移动
        return  1 + dfs(row+1, column, &visited) + dfs(row, column+1, &visited)
    }
```

```
    private func sumRowColumn(_ row: Int, _ column: Int) -> Int {
        return sumOfSingleNumber(row) + sumOfSingleNumber(column)
    }

    private func sumOfSingleNumber(_ num: Int) -> Int {
        var res: Int = 0
        var _num = num
        while _num != 0 {
            res += (_num % 10)
            _num /= 10
        }
        return res
    }
}

let solution = Solution()
print(solution.movingCount(2, 3, 1))
print(solution.movingCount(3, 1, 0))
```