AGENDA

Recursion + Backtracking

- Tail Recursion
- Tower of Hanoi
- Possible Words From Phone Digits
- Combination Sum
- Rat in a Maze Problem

Tail Recursion

When the recursive call is the last thing done by the recursive function.

Some compilers are capable of optimizing the tail recursive functions because since the recursive call is the last thing done by the function, there is no need to save the current stack trace while making the recursive call.

Discuss the following examples:

1. Print numbers from N to 1.
2. Calculate N!

Tower of Hanoi

In the Tower of Hanoi puzzle, you are given **n** disks stacked in ascending order (smallest at the top) on the first of three rods. The goal is to move all disks to the third rod following two rules: only one disk can be moved at a time, and a disk can only be placed on top of a larger disk. Given the number of disks n and three rods labeled as from, to, and aux (starting rod, target rod, and auxiliary rod, respectively), returns the total **number of moves** needed to transfer all disks from the starting rod to the target rod.

Examples:

Input: n = 2
Output: 3
Explanation: For n =2 , steps will be as follows in the example and total 3 steps will be taken.
move disk 1 from rod 1 to rod 2
move disk 2 from rod 1 to rod 3
move disk 1 from rod 2 to rod 3

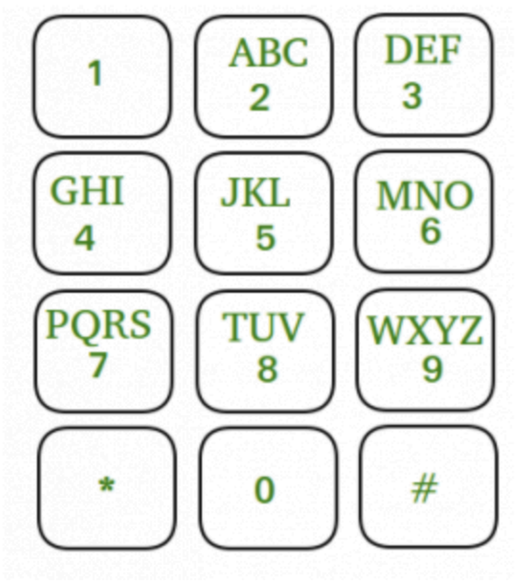
Input: n = 3
Output: 7
Explanation: For N=3 , steps will be as follows in the example and total 7 steps will be taken.
move disk 1 from rod 1 to rod 3
move disk 2 from rod 1 to rod 2
move disk 1 from rod 3 to rod 2
move disk 3 from rod 1 to rod 3
move disk 1 from rod 2 to rod 1
move disk 2 from rod 2 to rod 3
move disk 1 from rod 1 to rod 3

Input: n = 0
Output: 0
Explanation: Total 0 steps will be taken.

Constraints:
0 <= n <= 16

Possible Words From Phone Digits

Given a keypad as shown in the diagram, and an array **arr[ ]**, your task is to **list all possible words** in **any order** which can be generated by pressing numbers from array.



Examples :

```
Input: arr[] = [2, 3]
Output: ad ae af bd be bf cd ce cf
Explanation: When we press 2, 3 total possible words are 3 x 3 = 9.
```

```
Input: arr[] =[3, 4, 5]
Output: dgj dgk dgl dhj dhk dhl dij dik dil egj egk egl ehj ehk ehl eij eik eil
fgj fgk fgl fhj fhk fhl fij fik fil
Explanation: When we press 3, 4, 5 total possible words are 3 x 3 x 3 = 27.
```

```
Input: arr[] =[2]
Output: a b c
Explanation: When we press 2 total possible words are 3.
```

Constraints:

1 ≤ arr.size() ≤ 10

2 ≤ arr[i] ≤ 9

Combination Sum

Given an array **arr[]** and a **target**, your task is to find all **unique** combinations in the array where the sum is equal to target. The same number may be chosen from the array **any** number of times to make target.

You can return your answer in **any** order.

Examples:

Input: arr[] = [2, 4, 6, 8], target = 8
Output: [[2 2 2 2] [2 2 4] [2 6] [4 4] [8]]
Explanation: Total number of possible combinations are 5.

Input: arr[] = [2, 7, 6, 5], target = 16
Output: [[2 2 2 2 2 2 2 2] [2 2 2 2 2 6] [2 2 2 5 5] [2 2 5 7] [2 2 6 6] [2 7 7] [5 5 6]]
Explanation: Total number of possible combinations are 7.

Input: arr[] = [6, 5, 7], target = 8
Output: []
Explanation: There are no possible combinantions such that target sum is 8.

Constraints:
1 <= arr.size() <= 30
2 <= arr[i] <= 40
2 <= target <= 40
All arr[i] are **distinct**.

Rat in a Maze Problem

Consider a rat placed at position (0, 0) in an **n x n** square matrix mat. The rat's goal is to reach the destination at position (n-1, n-1). The rat can move in four possible directions: **'U'(up)**, **'D'(down)**, **'L' (left)**, **'R' (right)**.

The matrix contains only two possible values:

- 0: A blocked cell through which the rat **cannot** travel.
- 1: A free cell that the rat **can** pass through.

Note: In a path, no cell can be visited more than one time. If the source cell is 0, the rat cannot move to any other cell. In case of no path, return an empty list.+

The task is to find all possible paths the rat can take to reach the destination, starting from (0, 0) and ending at (n-1, n-1), under the condition that the rat cannot revisit any cell along the same path. Furthermore, the rat can only move to adjacent cells that are within the bounds of the matrix and not blocked.

Return the final result vector in lexicographically smallest order.

Examples:

Input: mat[][] = [[1, 0, 0, 0], [1, 1, 0, 1], [1, 1, 0, 0], [0, 1, 1, 1]]
Output: ["DDRDRR", "DRDDRR"]
Explanation: The rat can reach the destination at (3, 3) from (0, 0) by two paths - DRDDRR and DDRDRR, when printed in sorted order we get DDRDRR DRDDRR.

Input: mat[][] = [[1, 0], [1, 0]]
Output: []
Explanation: No path exists and the destination cell is blocked.

Input: mat = [[1, 1, 1], [1, 0, 1], [1, 1, 1]]
Output: ["DDRR", "RRDD"]
Explanation: The rat has two possible paths to reach the destination: 1. "DDRR" 2. "RRDD", These are returned in lexicographically sorted order.

Constraints:
2 ≤ mat.size() ≤ 5
0 ≤ mat[i][j] ≤ 1