Step 7 : Recursion [ PatternWise]

1.1 [**8. String to Integer (atoi)**](https://leetcode.com/problems/string-to-integer-atoi/)

Implement the myAtoi(string s) function, which converts a string to a 32-bit signed integer.

The algorithm for myAtoi(string s) is as follows:

1. **Whitespace**: Ignore any leading whitespace (" ").
2. **Signedness**: Determine the sign by checking if the next character is '-' or '+', assuming positivity if neither present.
3. **Conversion**: Read the integer by skipping leading zeros until a non-digit character is encountered or the end of the string is reached. If no digits were read, then the result is 0.
4. **Rounding**: If the integer is out of the 32-bit signed integer range [-231, 231 - 1], then round the integer to remain in the range. Specifically, integers less than -231 should be rounded to -231, and integers greater than 231 - 1 should be rounded to 231 - 1.

Return the integer as the final result.

**Example 1:**

**Input:** s = "42"

**Output:** 42

**Explanation:**

The underlined characters are what is read in and the caret is the current reader position.

Step 1: "42" (no characters read because there is no leading whitespace)

^

Step 2: "42" (no characters read because there is neither a '-' nor '+')

^

Step 3: "42" ("42" is read in)

^

**Example 2:**

**Input:** s = " -042"

**Output:** -42

**Explanation:**

Step 1: " -042" (leading whitespace is read and ignored)

^

Step 2: " -042" ('-' is read, so the result should be negative)

^

Step 3: " -042" ("042" is read in, leading zeros ignored in the result)

^

**Example 3:**

**Input:** s = "1337c0d3"

**Output:** 1337

**Explanation:**

Step 1: "1337c0d3" (no characters read because there is no leading whitespace)

^

Step 2: "1337c0d3" (no characters read because there is neither a '-' nor '+')

^

Step 3: "1337c0d3" ("1337" is read in; reading stops because the next character is a non-digit)

^

**Example 4:**

**Input:** s = "0-1"

**Output:** 0

**Explanation:**

Step 1: "0-1" (no characters read because there is no leading whitespace)

^

Step 2: "0-1" (no characters read because there is neither a '-' nor '+')

^

Step 3: "0-1" ("0" is read in; reading stops because the next character is a non-digit)

^

**Example 5:**

**Input:** s = "words and 987"

**Output:** 0

**Explanation:**

Reading stops at the first non-digit character 'w'.

**Constraints:**

* 0 <= s.length <= 200
* s consists of English letters (lower-case and upper-case), digits (0-9), ' ', '+', '-', and '.'.

1.2 [**50. Pow(x, n)**](https://leetcode.com/problems/powx-n/)

Implement [pow(x, n)](http://www.cplusplus.com/reference/valarray/pow/), which calculates x raised to the power n (i.e., xn).

**Example 1:**

**Input:** x = 2.00000, n = 10

**Output:** 1024.00000

**Example 2:**

**Input:** x = 2.10000, n = 3

**Output:** 9.26100

**Example 3:**

**Input:** x = 2.00000, n = -2

**Output:** 0.25000

**Explanation:** 2-2 = 1/22 = 1/4 = 0.25

**Constraints:**

* -100.0 < x < 100.0
* -231 <= n <= 231-1
* n is an integer.
* Either x is not zero or n > 0.
* -104 <= xn <= 104

1.3 [**1922. Count Good Numbers**](https://leetcode.com/problems/count-good-numbers/)

A digit string is **good** if the digits **(0-indexed)** at **even** indices are **even** and the digits at **odd** indices are **prime** (2, 3, 5, or 7).

* For example, "2582" is good because the digits (2 and 8) at even positions are even and the digits (5 and 2) at odd positions are prime. However, "3245" is **not** good because 3 is at an even index but is not even.

Given an integer n, return *the****total****number of good digit strings of length*n. Since the answer may be large, **return it modulo**109 + 7.

A **digit string** is a string consisting of digits 0 through 9 that may contain leading zeros.

**Example 1:**

**Input:** n = 1

**Output:** 5

**Explanation:** The good numbers of length 1 are "0", "2", "4", "6", "8".

**Example 2:**

**Input:** n = 4

**Output:** 400

**Example 3:**

**Input:** n = 50

**Output:** 564908303

**Constraints:**

* 1 <= n <= 1015

1.4 **Sort a stack**

Difficulty: **Medium**Accuracy: **69.19%**Submissions: **147K+**Points: **4**Average Time: **20m**

Given a stack, the task is to sort it such that the top of the stack has the greatest element.

**Example 1:**

**Input:**

Stack: 3 2 1

**Output:** 3 2 1

**Example 2:**

**Input:**

Stack: 11 2 32 3 41

**Output:** 41 32 11 3 2

**Your Task:**  
You don't have to read input or print anything. Your task is to complete the function **sort()**which sorts the elements present in the given stack. (The sorted stack is printed by the driver's code by popping the elements of the stack.)

**Expected Time Complexity**: O(N\*N)  
**Expected Auxilliary Space**: O(N) recursive.

**Constraints:**  
1<=N<=100

1.5 **Reverse a Stack**

Difficulty: **Medium**Accuracy: **80.5%**Submissions: **107K+**Points: **4**Average Time: **20m**

You are given a stack **St**. You have to reverse the stack using recursion.

**Example 1:**

**Input:**

St = {3,2,1,7,6}

**Output:**

{6,7,1,2,3}  
**Explanation:**  
Input stack after reversing will look like the stack in the output.

**Example 2:**

**Input:**

St = {4,3,9,6}

**Output:**

{6,9,3,4}  
**Explanation:**Input stack after reversing will look like the stack in the output.

**Your Task:**

You don't need to read input or print anything. Your task is to complete the function **Reverse()** which takes the stack **St**as input and reverses the given stack.

**Expected Time Complexity:** O(N2)  
**Expected Auxiliary Space:** O(1)

**Constraints:**  
1 <= size of the stack <= 104  
-109 <= Each element of the stack <= 109  
Sum of N over all test cases doesn't exceeds 106  
Array may contain duplicate elements.

2.1 **Generate all binary strings**

Difficulty: **Medium**Accuracy: **63.9%**Submissions: **23K+**Points: **4**Average Time: **20m**

Given an integer **N** , Print all binary strings of size N which do not contain consecutive 1s.

A binary string is that string which contains only 0 and 1.

**Example 1:**

**Input:**

N = 3

**Output:**

000 , 001 , 010 , 100 , 101

**Explanation:**

None of the above strings contain consecutive 1s. "110" is not an answer as it has '1's occuring consecutively.

**Your Task:**

You don't need to read input or print anything. Your task is to complete the function**generateBinaryStrings()** which takes an integer **N**as input and returns a list of all valid binary strings in lexicographically increasing order.

**Expected Time Complexity:** O(2N)  
**Expected Auxiliary Space:** O(N)

**Constraints:**  
1 <= N <= 20

2.2 [**22. Generate Parentheses**](https://leetcode.com/problems/generate-parentheses/)

Solved

Medium

Topics

Companies

Given n pairs of parentheses, write a function to *generate all combinations of well-formed parentheses*.

**Example 1:**

**Input:** n = 3

**Output:** ["((()))","(()())","(())()","()(())","()()()"]

**Example 2:**

**Input:** n = 1

**Output:** ["()"]

**Constraints:**

* 1 <= n <= 8

2.3 [**78. Subsets**](https://leetcode.com/problems/subsets/)

Medium

Topics

Companies

Given an integer array nums of **unique** elements, return *all possible* *subsets* *(the power set)*.

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

**Example 1:**

**Input:** nums = [1,2,3]

**Output:** [[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]

**Example 2:**

**Input:** nums = [0]

**Output:** [[],[0]]

**Constraints:**

* 1 <= nums.length <= 10
* -10 <= nums[i] <= 10
* All the numbers of nums are **unique**.

2.4 Learn All Patterns of Subsequences (Theory) :-

**🔑 What is a Subsequence?**

A **subsequence** is a sequence derived from another sequence by deleting **some or no elements** **without changing the order** of the remaining elements.

📌 Example:  
Original: abc  
Subsequences: ["", "a", "b", "c", "ab", "ac", "bc", "abc"]

**🧠 Master List of All Subsequences Patterns (with Theory)**

**✅ 1. Generate All Subsequences (Backtracking)**

**Pattern:** Use recursion to either include or exclude each character.

**Theory:**  
At each step, you have two choices:

* Include current character.
* Exclude current character.

**Time Complexity:** O(2^n)

➡️ Used in: Power Set, Subset Sum, String Subsequences.

**✅ 2. Subset Sum / Target Sum (Backtracking + DP)**

**Problem:** Given an array, check if a subset sums to target k.

**Theory:**  
Explore all possible sums using subsequences of the array.

**Optimized:** Use memoization/tabulation to avoid recomputation.

➡️ Used in:

* Partition Equal Subset Sum
* Count Subsets with Sum K
* Target Sum (Leetcode)

**✅ 3. Longest Common Subsequence (LCS)**

**Problem:** Find length of longest subsequence common to two strings.

**Theory:**

* If last characters match: 1 + LCS(i-1, j-1)
* Else: max(LCS(i-1, j), LCS(i, j-1))

**Tabulation/Memoization DP**

➡️ Used in:

* Edit Distance
* Shortest Common Supersequence
* Print LCS

**✅ 4. Palindrome Subsequences**

**Pattern:** Variants of LCS with reversed string.

**Examples:**

* **Longest Palindromic Subsequence**  
  LCS of string and its reverse.
* **Minimum Insertions to Make Palindrome**  
  len(s) - LCS(s, reverse(s))

**✅ 5. Distinct Subsequences (DP Counting)**

**Problem:** Count the number of ways string s can be transformed into t by deleting characters.

**Theory:**

* If characters match: count(i-1, j-1) + count(i-1, j)
* Else: count(i-1, j)

➡️ Classic: Leetcode 115 — Distinct Subsequences

**✅ 6. Partition-Based Subsequences**

**Used in:**

* Palindrome Partitioning
* Subsets with equal average
* K-Subset Partitioning

**✅ 7. Bitmasking for Subsequences**

**Theory:** Each bit in n-bit number represents include/exclude.

**Used in:**

* Power set generation
* Subset with max bitwise AND/OR
* Efficient subset filtering

**✅ 8. K-th Lexicographical Subsequence**

Generate k-th subsequence in sorted order using DFS or greedy + DP count pruning.

**✅ 9. Backtracking with Constraints**

Used in:

* Subsets with no adjacent elements
* Subsequences of length k
* Increasing/Decreasing subsequences

**🔁 Mnemonic to Remember the Patterns:**

**"G-L-O-W P-A-R-T B-K"**

* **G**enerate all
* **L**ongest common subsequence
* **O**ptimal target sum
* **W**ays to match subsequence
* **P**alindromic variations
* **A**rrays with partition
* **R**everse to match
* **T**arget subsequence count
* **B**itmask tricks
* **K**-th lexicographical subsequence

2.5 [**560. Subarray Sum Equals K**](https://leetcode.com/problems/subarray-sum-equals-k/)

Given an array of integers nums and an integer k, return *the total number of subarrays whose sum equals to* k.

A subarray is a contiguous **non-empty** sequence of elements within an array.

**Example 1:**

**Input:** nums = [1,1,1], k = 2

**Output:** 2

**Example 2:**

**Input:** nums = [1,2,3], k = 3

**Output:** 2

**Constraints:**

* 1 <= nums.length <= 2 \* 104
* -1000 <= nums[i] <= 1000
* -107 <= k <= 107

2.6 **Check if there exists a subsequence with sum K**

Difficulty: **Medium**Accuracy: **40.26%**Submissions: **23K+**Points: **4**Average Time: **30m**

Given an array **arr**and target sum **k**, check whether there exists a subsequence such that the sum of all elements in the subsequence equals the given target sum(k).

**Example:**

**Input:**  arr = [10,1,2,7,6,1,5], k = 8.

**Output:**  Yes

**Explanation:**  Subsequences like [2, 6], [1, 7] sum upto 8

**Input:**  arr = [2,3,5,7,9], k = 100.

**Output:**  No

**Explanation:**  No subsequence can sum upto 100

**Your Task:**  
You don't need to read or print anything. Your task is to complete the boolean function **checkSubsequenceSum()** which takes N, arr and K as input parameter and returns true/false based on the whether any subsequence with sum K could be found.

**Expected Time Complexity:** O(N \* K).  
**Expected Auxiliary Space:** O(N \* K

2.7 [**39. Combination Sum**](https://leetcode.com/problems/combination-sum/)

Given an array of **distinct** integers candidates and a target integer target, return *a list of all****unique combinations****of*candidates*where the chosen numbers sum to*target*.* You may return the combinations in **any order**.

The **same** number may be chosen from candidates an **unlimited number of times**. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

The test cases are generated such that the number of unique combinations that sum up to target is less than 150 combinations for the given input.

**Example 1:**

**Input:** candidates = [2,3,6,7], target = 7

**Output:** [[2,2,3],[7]]

**Explanation:**

2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.

7 is a candidate, and 7 = 7.

These are the only two combinations.

**Example 2:**

**Input:** candidates = [2,3,5], target = 8

**Output:** [[2,2,2,2],[2,3,3],[3,5]]

**Example 3:**

**Input:** candidates = [2], target = 1

**Output:** []

**Constraints:**

* 1 <= candidates.length <= 30
* 2 <= candidates[i] <= 40
* All elements of candidates are **distinct**.
* 1 <= target <= 40

2.8 [**40. Combination Sum II**](https://leetcode.com/problems/combination-sum-ii/)

Given a collection of candidate numbers (candidates) and a target number (target), find all unique combinations in candidates where the candidate numbers sum to target.

Each number in candidates may only be used **once** in the combination.

**Note:** The solution set must not contain duplicate combinations.

**Example 1:**

**Input:** candidates = [10,1,2,7,6,1,5], target = 8

**Output:**

[

[1,1,6],

[1,2,5],

[1,7],

[2,6]

]

**Example 2:**

**Input:** candidates = [2,5,2,1,2], target = 5

**Output:**

[

[1,2,2],

[5]

]

**Constraints:**

* 1 <= candidates.length <= 100
* 1 <= candidates[i] <= 50
* 1 <= target <= 30

2.9 **Subset Sums**

Difficulty: **Medium**Accuracy: **72.55%**Submissions: **162K+**Points: **4**

Given a array **arr**of integers, return the sums of all subsets in the list.  Return the sums in any order.

**Examples:**

**Input:** arr[] = [2, 3]

**Output:** [0, 2, 3, 5]

**Explanation:** When no elements are taken then Sum = 0. When only 2 is taken then Sum = 2. When only 3 is taken then Sum = 3. When elements 2 and 3 are taken then Sum = 2+3 = 5.

**Input:** arr[] = [1, 2, 1]

**Output:** [0, 1, 1, 2, 2, 3, 3, 4]  
**Explanation:** The possible subset sums are 0 (no elements), 1 (either of the 1's), 2 (the element 2), and their combinations.

**Input:** arr[] = [5, 6, 7]

**Output:** [0, 5, 6, 7, 11, 12, 13, 18]

**Explanation:** The possible subset sums are 0 (no elements), 5, 6, 7, and their combinations.

**Constraints:**  
1 ≤ arr.size() ≤ 15  
0 ≤ arr[i] ≤ 104

2.10 [**90. Subsets II**](https://leetcode.com/problems/subsets-ii/)

Given an integer array nums that may contain duplicates, return *all possible* *subsets (the power set)*.

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

**Example 1:**

**Input:** nums = [1,2,2]

**Output:** [[],[1],[1,2],[1,2,2],[2],[2,2]]

**Example 2:**

**Input:** nums = [0]

**Output:** [[],[0]]

**Constraints:**

* 1 <= nums.length <= 10
* -10 <= nums[i] <= 10

2.11 [**216. Combination Sum III**](https://leetcode.com/problems/combination-sum-iii/)

Find all valid combinations of k numbers that sum up to n such that the following conditions are true:

* Only numbers 1 through 9 are used.
* Each number is used **at most once**.

Return *a list of all possible valid combinations*. The list must not contain the same combination twice, and the combinations may be returned in any order.

**Example 1:**

**Input:** k = 3, n = 7

**Output:** [[1,2,4]]

**Explanation:**

1 + 2 + 4 = 7

There are no other valid combinations.

**Example 2:**

**Input:** k = 3, n = 9

**Output:** [[1,2,6],[1,3,5],[2,3,4]]

**Explanation:**

1 + 2 + 6 = 9

1 + 3 + 5 = 9

2 + 3 + 4 = 9

There are no other valid combinations.

**Example 3:**

**Input:** k = 4, n = 1

**Output:** []

**Explanation:** There are no valid combinations.

Using 4 different numbers in the range [1,9], the smallest sum we can get is 1+2+3+4 = 10 and since 10 > 1, there are no valid combination.

**Constraints:**

* 2 <= k <= 9
* 1 <= n <= 60

2.12 [**17. Letter Combinations of a Phone Number**](https://leetcode.com/problems/letter-combinations-of-a-phone-number/)

Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in **any order**.

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.



**Example 1:**

**Input:** digits = "23"

**Output:** ["ad","ae","af","bd","be","bf","cd","ce","cf"]

**Example 2:**

**Input:** digits = ""

**Output:** []

**Example 3:**

**Input:** digits = "2"

**Output:** ["a","b","c"]

**Constraints:**

* 0 <= digits.length <= 4
* digits[i] is a digit in the range ['2', '9'].

3.1 [**131. Palindrome Partitioning**](https://leetcode.com/problems/palindrome-partitioning/)

Medium

Topics

Companies

Given a string s, partition s such that every substring of the partition is a **palindrome**. Return *all possible palindrome partitioning of*s.

**Example 1:**

**Input:** s = "aab"

**Output:** [["a","a","b"],["aa","b"]]

**Example 2:**

**Input:** s = "a"

**Output:** [["a"]]

**Constraints:**

* 1 <= s.length <= 16
* s contains only lowercase English letters.

3.2 [**79. Word Search**](https://leetcode.com/problems/word-search/)

Medium

Topics

Companies

Given an m x n grid of characters board and a string word, return true *if* 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.

**Example 1:**

A white and orange squares with letters

AI-generated content may be incorrect.

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

**Output:** true

**Example 2:**

A white and orange squares with black letters

AI-generated content may be incorrect.

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

**Output:** true

**Example 3:**

A grid of letters in black

AI-generated content may be incorrect.

**Input:** board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCB"

**Output:** false

**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.

3.3 [**51. N-Queens**](https://leetcode.com/problems/n-queens/)

The **n-queens** puzzle is the problem of placing n queens on an n x n chessboard such that no two queens attack each other.

Given an integer n, return *all distinct solutions to the****n-queens puzzle***. You may return the answer in **any order**.

Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' both indicate a queen and an empty space, respectively.

**Example 1:**

A chess pieces on a checkerboard

AI-generated content may be incorrect.

**Input:** n = 4

**Output:** [[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]

**Explanation:** There exist two distinct solutions to the 4-queens puzzle as shown above

**Example 2:**

**Input:** n = 1

**Output:** [["Q"]]

**Constraints:**

* 1 <= n <= 9

3.4 **Rat in a Maze Problem - I**

Difficulty: **Medium**Accuracy: **35.75%**Submissions: **339K+**Points: **4**Average Time: **25m**

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.

Your 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.  
If no path exists, return an **empty list.**

**Note:** 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 as 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

3.5 [**139. Word Break**](https://leetcode.com/problems/word-break/)

Given a string s and a dictionary of strings wordDict, return true if s can be segmented into a space-separated sequence of one or more dictionary words.

**Note** that the same word in the dictionary may be reused multiple times in the segmentation.

**Example 1:**

**Input:** s = "leetcode", wordDict = ["leet","code"]

**Output:** true

**Explanation:** Return true because "leetcode" can be segmented as "leet code".

**Example 2:**

**Input:** s = "applepenapple", wordDict = ["apple","pen"]

**Output:** true

**Explanation:** Return true because "applepenapple" can be segmented as "apple pen apple".

Note that you are allowed to reuse a dictionary word.

**Example 3:**

**Input:** s = "catsandog", wordDict = ["cats","dog","sand","and","cat"]

**Output:** false

**Constraints:**

* 1 <= s.length <= 300
* 1 <= wordDict.length <= 1000
* 1 <= wordDict[i].length <= 20
* s and wordDict[i] consist of only lowercase English letters.
* All the strings of wordDict are **unique**.

3.6 **M-Coloring Problem**

Difficulty: **Medium**Accuracy: **34.42%**Submissions: **156K+**Points: **4**Average Time: **45m**

You are given an undirected graph consisting of **V** vertices and **E** edges represented by a list **edges[][]**, along with an integer **m**. Your task is to determine whether it is possible to **color the graph** using at most **m** different colors such that no two adjacent vertices share the **same color**. Return true if the graph can be colored with at most **m** colors, otherwise return false.

**Note:** The graph is indexed with 0-based indexing.

**Examples:**

**Input:** V = 4, edges[][] = [[0, 1], [1, 3], [2, 3], [3, 0], [0, 2]], m = 3

**Output:** true

**Explanation:** It is possible to color the given graph using 3 colors, for example, one of the possible ways vertices can be colored as follows:  
A diagram of a number connected to a network

AI-generated content may be incorrect.  
Vertex 0: Color 1

Vertex 1: Color 2

Vertex 2: Color 2

Vertex 3: Color 3

**Input:** V = 3, edges[][] = [[0, 1], [1, 2], [0, 2]], m = 2

**Output:** false  
**Explanation:** It is not possible to color the given graph using only 2 colors because vertices 0, 1, and 2 form a triangle.

**Constraints:**  
1 ≤ V ≤ 10  
1 ≤ E = edges.size() ≤ (V\*(V-1))/2  
0 ≤ edges[i][j] ≤ V-1  
1 ≤ m ≤ V

3.7 [**37. Sudoku Solver**](https://leetcode.com/problems/sudoku-solver/)

Hard

Topics

Companies

Write a program to solve a Sudoku puzzle by filling the empty cells.

A sudoku solution must satisfy **all of the following rules**:

1. Each of the digits 1-9 must occur exactly once in each row.
2. Each of the digits 1-9 must occur exactly once in each column.
3. Each of the digits 1-9 must occur exactly once in each of the 9 3x3 sub-boxes of the grid.

The '.' character indicates empty cells.

**Example 1:**

A square puzzle with numbers

AI-generated content may be incorrect.

**Input:** board = [["5","3",".",".","7",".",".",".","."],["6",".",".","1","9","5",".",".","."],[".","9","8",".",".",".",".","6","."],["8",".",".",".","6",".",".",".","3"],["4",".",".","8",".","3",".",".","1"],["7",".",".",".","2",".",".",".","6"],[".","6",".",".",".",".","2","8","."],[".",".",".","4","1","9",".",".","5"],[".",".",".",".","8",".",".","7","9"]]

**Output:** [["5","3","4","6","7","8","9","1","2"],["6","7","2","1","9","5","3","4","8"],["1","9","8","3","4","2","5","6","7"],["8","5","9","7","6","1","4","2","3"],["4","2","6","8","5","3","7","9","1"],["7","1","3","9","2","4","8","5","6"],["9","6","1","5","3","7","2","8","4"],["2","8","7","4","1","9","6","3","5"],["3","4","5","2","8","6","1","7","9"]]

**Explanation:** The input board is shown above and the only valid solution is shown below:

A square of numbers with red and black text

AI-generated content may be incorrect.

**Constraints:**

* board.length == 9
* board[i].length == 9
* board[i][j] is a digit or '.'.
* It is **guaranteed** that the input board has only one solution.

3.8 [**282. Expression Add Operators**](https://leetcode.com/problems/expression-add-operators/)

Given a string num that contains only digits and an integer target, return ***all possibilities****to insert the binary operators*'+'*,*'-'*, and/or*'\*'*between the digits of*num*so that the resultant expression evaluates to the*target*value*.

Note that operands in the returned expressions **should not** contain leading zeros.

**Example 1:**

**Input:** num = "123", target = 6

**Output:** ["1\*2\*3","1+2+3"]

**Explanation:** Both "1\*2\*3" and "1+2+3" evaluate to 6.

**Example 2:**

**Input:** num = "232", target = 8

**Output:** ["2\*3+2","2+3\*2"]

**Explanation:** Both "2\*3+2" and "2+3\*2" evaluate to 8.

**Example 3:**

**Input:** num = "3456237490", target = 9191

**Output:** []

**Explanation:** There are no expressions that can be created from "3456237490" to evaluate to 9191.

**Constraints:**

* 1 <= num.length <= 10
* num consists of only digits.
* -231 <= target <= 231 - 1