OOP IA2

Div 4 and 6 Sem III CSE

<table>
<tr><td>Group 1</td><td>Given an input string s and a pattern p, implement regular expression matching with support
for '.' and '*' where:

• '.' Matches any single character.
• '*' Matches zero or more of the preceding element.

The matching should cover the entire input string (not partial).

Example 1:

Input: s = "aa", p = "a"
Output: false
Explanation: "a" does not match the entire string "aa".

Example 2:

Input: s = "aa", p = "a*"
Output: true
Explanation: '*' means zero or more of the preceding element, 'a'. Therefore, by repeating 'a' once, it becomes "aa".

Example 3:

Input: s = "ab", p = ".*"
Output: true
Explanation: ".*" means "zero or more (*) of any character (.)".</td></tr>
<tr><td>Group 2</td><td>You are given a string s and an array of strings words. All the strings of words are of the same length.

A concatenated string is a string that exactly contains all the strings of any permutation of words concatenated.

• For example, if words = ["ab","cd","ef"],
then "abcdef", "abefcd", "cdabef", "cdefab", "efabcd", and "efcdab" are all concatenated strings. "acdbef" is not a concatenated string because it is not the concatenation of any permutation of words.

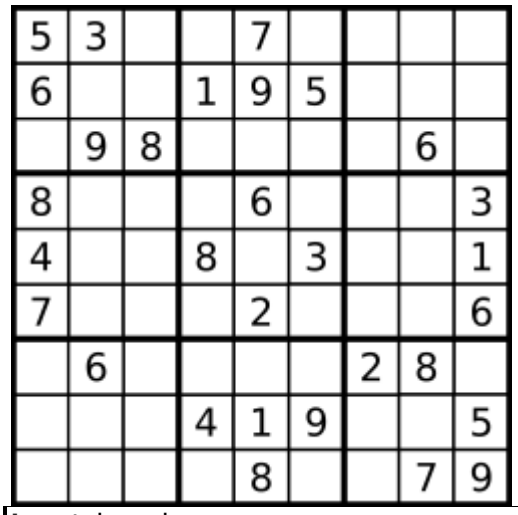
Return an array of the starting indices of all the concatenated substrings in s. You can return the answer in any order.

Example 1:

Input: s = "barfoothefoobarman", words = ["foo","bar"]</td></tr>
</table>

	Output: [0,9] **Explanation:** The substring starting at 0 is "barfoo". It is the concatenation of ["bar","foo"] which is a permutation of words. The substring starting at 9 is "foobar". It is the concatenation of ["foo","bar"] which is a permutation of words. **Example 2:** **Input:** s = "wordgoodgoodgoodbestword", words = ["word","good","best","word"] **Output:** [] **Explanation:** There is no concatenated substring. **Example 3:** **Input:** s = "barfoofoobarthefoobarman", words = ["bar","foo","the"] **Output:** [6,9,12] **Explanation:** The substring starting at 6 is "foobarthe". It is the concatenation of ["foo","bar","the"]. The substring starting at 9 is "barthefoo". It is the concatenation of ["bar","the","foo"]. The substring starting at 12 is "thefoobar". It is the concatenation of ["the","foo","bar"]. **Constraints:** • 1 <= s.length <= 10^4 • 1 <= words.length <= 5000 • 1 <= words[i].length <= 30 • s and words[i] consist of lowercase English letters.
Group 3	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.

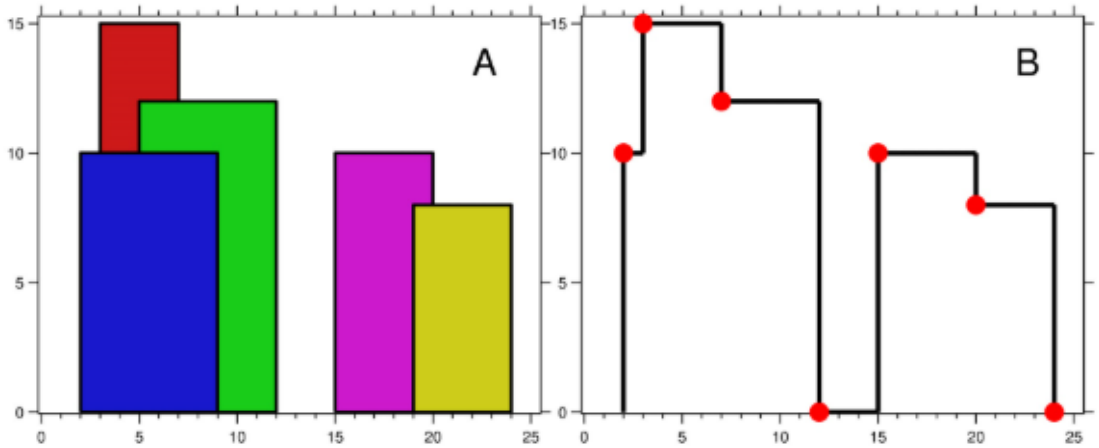
Example 1:



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:

Constraints:

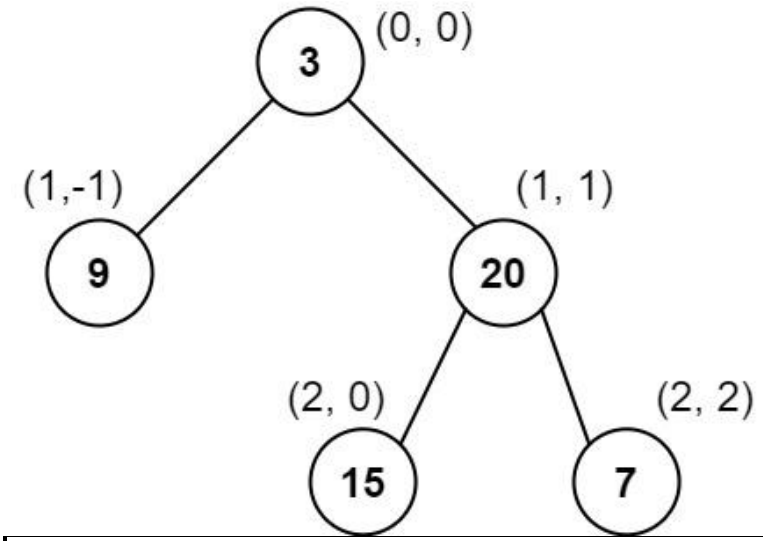
- board.length == 9
- board[i].length == 9
- board[i][j] is a digit or '.'.
- It is **guaranteed** that the input board has only one solution.

Group 4	A city's **skyline** is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Given the locations and heights of all the buildings, return *the **skyline** formed by these buildings collectively*. The geometric information of each building is given in the array buildings where buildings[i] = [$left_i$, $right_i$, $height_i$]: • $left_i$ is the x coordinate of the left edge of the i^{th} building. • $right_i$ is the x coordinate of the right edge of the i^{th} building. • $height_i$ is the height of the i^{th} building. You may assume all buildings are perfect rectangles grounded on an absolutely flat surface at height 0. The **skyline** should be represented as a list of "key points" **sorted by their x-coordinate** in the form [[x_1,y_1],[x_2,y_2],...]. Each key point is the left endpoint of some horizontal segment in the skyline except the last point in the list, which always has a y-coordinate 0 and is used to mark the skyline's termination where the rightmost building ends. Any ground between the leftmost and rightmost buildings should be part of the skyline's contour. **Note:** There must be no consecutive horizontal lines of equal height in the output skyline. For instance, [...,[2 3],[4 5],[7 5],[11 5],[12 7],...] is not acceptable; the three lines of height 5 should be merged into one in the final output as such: [...,[2 3],[4 5],[12 7],...] **Example 1:**  **Input:** buildings = [[2,9,10],[3,7,15],[5,12,12],[15,20,10],[19,24,8]] **Output:** [[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]] **Explanation:** Figure A shows the buildings of the input. Figure B shows the skyline formed by those buildings. The red points in figure B represent the key points in the output list.

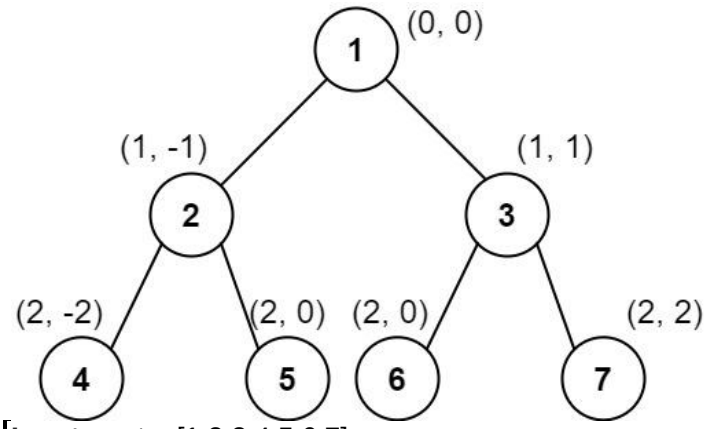
	Example 2: **Input:** buildings = [[0,2,3],[2,5,3]] **Output:** [[0,3],[5,0]] **Constraints:** • 1 <= buildings.length <= 10^4 • 0 <= $left_i$ < $right_i$ <= 2^{31} - 1 • 1 <= $height_i$ <= 2^{31} - 1 • buildings is sorted by $left_i$ in non-decreasing order.
Group 5	Given an array of integers citations where citations[i] is the number of citations a researcher received for their i^{th} paper, return *the researcher's h-index*. According to the definition of h-index on Wikipedia: The h-index is defined as the maximum value of h such that the given researcher has published at least h papers that have each been cited at least h times. **Example 1:** **Input:** citations = [3,0,6,1,5] **Output:** 3 **Explanation:** [3,0,6,1,5] means the researcher has 5 papers in total and each of them had received 3, 0, 6, 1, 5 citations respectively. Since the researcher has 3 papers with at least 3 citations each and the remaining two with no more than 3 citations each, their h-index is 3. **Example 2:** **Input:** citations = [1,3,1] **Output:** 1 **Constraints:** • n == citations.length • 1 <= n <= 5000 • 0 <= citations[i] <= 1000
Group 6	There are some spherical balloons taped onto a flat wall that represents the XY-plane. The balloons are represented as a 2D integer array points where points[i] = [x_{start}, x_{end}] denotes a balloon whose **horizontal diameter** stretches between x_{start} and x_{end}. You do not know the exact y-coordinates of the balloons. Arrows can be shot up **directly vertically** (in the positive y-direction) from different points along the x-axis. A balloon with x_{start} and x_{end} is **burst** by an arrow shot at x if x_{start} <=

	$x <= x_{end}$. There is **no limit** to the number of arrows that can be shot. A shot arrow keeps traveling up infinitely, bursting any balloons in its path. Given the array points, return *the **minimum** number of arrows that must be shot to burst all balloons*. **Example 1:** **Input:** points = [[10,16],[2,8],[1,6],[7,12]] **Output:** 2 **Explanation:** The balloons can be burst by 2 arrows: - Shoot an arrow at x = 6, bursting the balloons [2,8] and [1,6]. - Shoot an arrow at x = 11, bursting the balloons [10,16] and [7,12]. **Example 2:** **Input:** points = [[1,2],[3,4],[5,6],[7,8]] **Output:** 4 **Explanation:** One arrow needs to be shot for each balloon for a total of 4 arrows. **Example 3:** **Input:** points = [[1,2],[2,3],[3,4],[4,5]] **Output:** 2 **Explanation:** The balloons can be burst by 2 arrows: - Shoot an arrow at x = 2, bursting the balloons [1,2] and [2,3]. - Shoot an arrow at x = 4, bursting the balloons [3,4] and [4,5]. **Constraints:** • $1 <= points.length <= 10^5$ • $points[i].length == 2$ • $-2^{31} <= x_{start} < x_{end} <= 2^{31} - 1$
Group 7	Suppose Group7 will start its **IPO** soon. In order to sell a good price of its shares to Venture Capital, Group7 would like to work on some projects to increase its capital before the **IPO**. Since it has limited resources, it can only finish at most k distinct projects before the **IPO**. Help Group7 design the best way to maximize its total capital after finishing at most k distinct projects. You are given n projects where the i^{th} project has a pure profit profits[i] and a minimum capital of capital[i] is needed to start it. Initially, you have w capital. When you finish a project, you will obtain its pure profit and the profit will be added to your total capital. Pick a list of **at most** k distinct projects from given projects to **maximize your final capital**, and return *the final maximized capital*.

	The answer is guaranteed to fit in a 32-bit signed integer. **Example 1:** **Input:** k = 2, w = 0, profits = [1,2,3], capital = [0,1,1] **Output:** 4 **Explanation:** Since your initial capital is 0, you can only start the project indexed 0. After finishing it you will obtain profit 1 and your capital becomes 1. With capital 1, you can either start the project indexed 1 or the project indexed 2. Since you can choose at most 2 projects, you need to finish the project indexed 2 to get the maximum capital. Therefore, output the final maximized capital, which is 0 + 1 + 3 = 4. **Example 2:** **Input:** k = 3, w = 0, profits = [1,2,3], capital = [0,1,2] **Output:** 6 **Constraints:** • $1 <= k <= 10^5$ • $0 <= w <= 10^9$ • n == profits.length • n == capital.length • $1 <= n <= 10^5$ • $0 <= profits[i] <= 10^4$ • $0 <= capital[i] <= 10^9$
Group 8	There are n different online courses numbered from 1 to n. You are given an array courses where courses[i] = [$duration_i$, $lastDay_i$] indicate that the i^{th} course should be taken **continuously** for $duration_i$ days and must be finished before or on $lastDay_i$. You will start on the 1st day and you cannot take two or more courses simultaneously. Return *the maximum number of courses that you can take*. **Example 1:** **Input:** courses = [[100,200],[200,1300],[1000,1250],[2000,3200]] **Output:** 3 Explanation: There are totally 4 courses, but you can take 3 courses at most: First, take the 1st course, it costs 100 days so you will finish it on the 100th day, and ready to take the next course on the 101st day. Second, take the 3rd course, it costs 1000 days so you will finish it on the 1100th day, and ready to take the next course on the 1101st day. Third, take the 2nd course, it costs 200 days so you will finish it on the 1300th day. The 4th course cannot be taken now, since you will finish it on the 3300th day, which exceeds the closed date. **Example 2:**

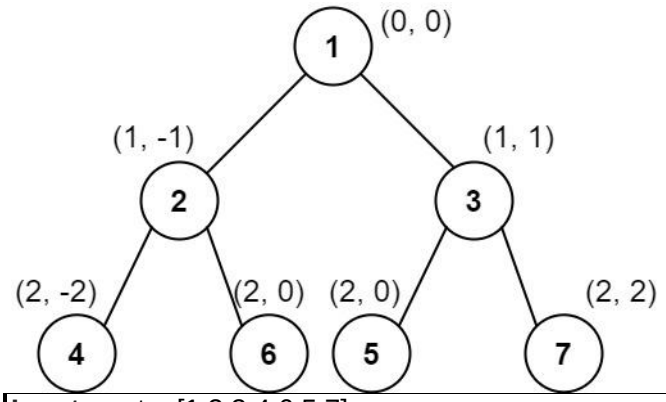
	Input: courses = [[1,2]] **Output:** 1 **Example 3:** **Input:** courses = [[3,2],[4,3]] **Output:** 0 **Constraints:** • 1 <= courses.length <= 10^4 • 1 <= $duration_i$, $lastDay_i$ <= 10^4
Group 9	Given the root of a binary tree, calculate the **vertical order traversal** of the binary tree. For each node at position (row, col), its left and right children will be at positions (row + 1, col - 1) and (row + 1, col + 1) respectively. The root of the tree is at (0, 0). The **vertical order traversal** of a binary tree is a list of top-to-bottom orderings for each column index starting from the leftmost column and ending on the rightmost column. There may be multiple nodes in the same row and same column. In such a case, sort these nodes by their values. Return *the **vertical order traversal** of the binary tree.* **Example 1:**  **Input:** root = [3,9,20,null,null,15,7] **Output:** [[9],[3,15],[20],[7]] **Explanation:** Column -1: Only node 9 is in this column. Column 0: Nodes 3 and 15 are in this column in that order from top to bottom. Column 1: Only node 20 is in this column. Column 2: Only node 7 is in this column.

Example 2:



Input: root = [1,2,3,4,5,6,7]
Output: [[4],[2],[1,5,6],[3],[7]]
Explanation:
Column -2: Only node 4 is in this column.
Column -1: Only node 2 is in this column.
Column 0: Nodes 1, 5, and 6 are in this column.
1 is at the top, so it comes first.
5 and 6 are at the same position (2, 0), so we order them by their value, 5 before 6.
Column 1: Only node 3 is in this column.
Column 2: Only node 7 is in this column.

Example 3:



Input: root = [1,2,3,4,6,5,7]
Output: [[4],[2],[1,5,6],[3],[7]]
Explanation:
This case is the exact same as example 2, but with nodes 5 and 6 swapped.
Note that the solution remains the same since 5 and 6 are in the same location and should be ordered by their values.

Constraints:

- The number of nodes in the tree is in the range [1, 1000].
- 0 <= Node.val <= 1000

Group 10	A chef has collected data on the satisfaction level of his n dishes. Chef can cook any dish in 1 unit of time. **Like-time coefficient** of a dish is defined as the time taken to cook that dish including previous dishes multiplied by its satisfaction level i.e. time[i] * satisfaction[i]. Return the maximum sum of **like-time coefficient** that the chef can obtain after preparing some amount of dishes. Dishes can be prepared in **any** order and the chef can discard some dishes to get this maximum value. **Example 1:** **Input:** satisfaction = [-1,-8,0,5,-9] **Output:** 14 **Explanation:** After Removing the second and last dish, the maximum total **like-time coefficient** will be equal to (-1*1 + 0*2 + 5*3 = 14). Each dish is prepared in one unit of time. **Example 2:** **Input:** satisfaction = [4,3,2] **Output:** 20 **Explanation:** Dishes can be prepared in any order, (2*1 + 3*2 + 4*3 = 20) **Example 3:** **Input:** satisfaction = [-1,-4,-5] **Output:** 0 **Explanation:** People do not like the dishes. No dish is prepared. **Constraints:** • n == satisfaction.length • 1 <= n <= 500 • -1000 <= satisfaction[i] <= 1000