COMPETITIVE PROGRAMMING HANDBOOK
SKILL-UP / SKILL-BRIDGE EXERCISES

CONTENTS

Introduction to Competitive Programming

Competitive programming is a sport where contestants solve algorithmic problems within a time limit using a programming language of their choice. It tests problem-solving skills, knowledge of algorithms, and ability to write efficient code. This is popular among students, computer science enthusiasts, and professionals looking to improve their skills. It is also used as a means of recruitment by many companies with many hosting their own coding competitions. The most popular competitive programming contests are CodeForces, Google Code Jam, Facebook Hacker Cup, and TopCoder Open. It can help develop skills such as problem-solving, critical thinking, and efficient coding. These can be valuable in a variety of careers such as software development, data science, and research. Many companies also use these programming problems as a way to assess job applicants. As a result, having a strong background in this can increase your chances of getting hired. However, it's important to note that competitive programming alone doesn't guarantee a successful career; you need to combine it with other relevant skills and experiences.

Competitive programming combines two topics: **the design of algorithms and the implementation of algorithms**. Design of Algorithms: the core of competitive programming is about inventing efficient algorithms that solve well-defined computational problems. The design of algorithms requires problem-solving and mathematical skills. Often a solution to a problem is a combination of well-known methods and new insights. Mathematics plays an important role in competitive programming.

In competitive programming, the solutions to problems are evaluated by testing an implemented algorithm using a set of test cases. Thus, after coming up with an algorithm that solves the problem, the next step is to correctly implement it, which requires good programming skills. Competitive programming greatly differs from traditional software engineering: Programs are short (usually at most some hundreds of lines), they should be written quickly, and it is not needed to maintain them after the contest.

When solving problems, one should keep in mind that the number of solved problems is not as important as the quality of the problems. It is tempting to select problems that look nice and easy and solve them, and skip problems that look hard and tedious. However, the way to really improve one's skills is to focus on the latter type of problems.

Another important observation is that most programming contest problems can be solved using simple and short algorithms, but the difficult part is to invent the algorithm. Competitive programming is not about learning complex and obscure algorithms by heart, but rather about learning problem solving and ways to approach difficult problems using simple tools. Finally, some people despise the implementation of algorithms: It is fun to design algorithms but boring to implement them. However, the ability to quickly and correctly implement algorithms is an important asset, and this skill can be practiced. It is a bad idea to spend most of the contest time for writing code and finding bugs, instead of thinking of how to solve problems.

Benefits of Competitive Programming

Problem-Solving and Programming Skills
This challenges contestants to solve algorithmic problems within a time limit. This helps improve problem-solving skills and the ability to write efficient and optimized code. It also nurtures a deep understanding of algorithms and data structures.

Technical Interview Preparation

Many companies use competitive programming as a means of recruitment. Also, the skills and knowledge gained through competitive programming can be valuable in technical interviews. It also gives participants an idea of the type of questions that are likely to be asked during these interviews.

Prizes
Many competitive programming contests offer prizes for the top performers. This can include cash, scholarships, or even job offers.

Enhances Your Thinking Ability
Competitive programming requires contestants to think critically and creatively, and this can help improve overall thinking ability. It also helps to develop the ability to analyze and solve problems under pressure.

Unleashes the Ability to Write Better and Optimized Code
It helps improve coding skills and the ability to write efficient and optimized code. It also helps to develop the ability to identify and fix bugs quickly.

Great Way to Start Your Programming Journey
This is a great way to start your programming journey. It helps develop a strong foundation in algorithms and data structures, and also provides a sense of accomplishment and personal satisfaction.

Competitive Programming Problems
Competitive programming problems can be found on online platforms such as CodeForces, HackerRank, LeetCode, SPOJ, and CodeChef, among others.

1. Searching

1.1 Search Insert Position

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with O(log n) runtime complexity.

Input: nums = [1, 3, 5, 6], target = 5

Output: 2

Input: nums = [1, 3, 5, 6], target = 2

Output: 1

Input: nums = [1, 3, 5, 6], target = 7

Output: 4

1.2 Search a 2D Matrix

You are given an m x n integer matrix with the following two properties:

- Each row is sorted in non-decreasing order.
- The first integer of each row is greater than the last integer of the previous row.

Given an integer target, return true if target is in matrix or false otherwise.
You must write a solution in O(log(m * n)) time complexity.

1	3	5	7
10	11	16	20
23	30	34	60

Input: matrix = [[1, 3, 5, 7], [10, 11, 16, 20], [23, 30, 34, 60]], target = 3

Output: true

1	3	5	7
10	11	16	20
23	30	34	60

Input: matrix = [[1, 3, 5, 7], [10, 11, 16, 20], [23, 30, 34, 60]], target = 13

Output: false

1.3 Search in Rotated Sorted Array

There is an integer array nums sorted in ascending order (with **distinct** values).

Prior to being passed to your function, nums is **possibly rotated** at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (**0-indexed**). For example, [0, 1, 2, 4, 5, 6, 7] might be rotated at pivot index 3 and become [4, 5, 6, 7, 0, 1, 2].

Given the array nums **after** the possible rotation and an integer target, return the index of target if it is in nums, or -1 if it is not in nums.

You must write an algorithm with O(log n) runtime complexity.

Input: nums = [4, 5, 6, 7, 0, 1, 2], target = 0

Output: 4

Input: nums = [4, 5, 6, 7, 0, 1, 2], target = 3

Output: -1

Input: nums = [1], target = 0

Output: -1

1.4 Find First and Last Position of Element in Sorted Array

Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of a given target value.

If target is not found in the array, return [-1, -1].

You must write an algorithm with O(log n) runtime complexity.

Input: nums = [5, 7, 7, 8, 8, 10], target = 8

Output: [3, 4]

Input: nums = [5, 7, 7, 8, 8, 10], target = 6

Output: [-1, -1]

Input: nums = [], target = 0

Output: [-1, -1]

1.5 Missing Number

Given an array nums containing n distinct numbers in the range [0, n], return the only number in the range that is missing from the array.

Input: nums = [3, 0, 1]

Output: 2

Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.

Input: nums = [0, 1]

Output: 2

Explanation: n = 2 since there are 2 numbers, so all numbers are in the range [0,2]. 2 is the missing number in the range since it does not appear in nums.

Input: nums = [9, 6, 4, 2, 3, 5, 7, 0, 1]

Output: 8

Explanation: n = 9 since there are 9 numbers, so all numbers are in the range [0,9]. 8 is the missing number in the range since it does not appear in nums.

2. Sorting

2.1 Relative Ranks

You are given an integer array score of size n, where score[i] is the score of the i^{th} athlete in a competition. All the scores are guaranteed to be **unique**.

The athletes are **placed** based on their scores, where the 1^{st} place athlete has the highest score, the 2^{nd} place athlete has the 2^{nd} highest score, and so on. The placement of each athlete determines their rank:

- The 1^{st} place athlete's rank is "Gold Medal".
- The 2^{nd} place athlete's rank is "Silver Medal".
- The 3^{rd} place athlete's rank is "Bronze Medal".
- For the 4^{th} place to the n^{th} place athlete, their rank is their placement number (i.e., the x^{th} place athlete's rank is "x").

Return an array answer of size n where answer[i] is the **rank** of the i^{th} athlete.

Input: score = [5, 4, 3, 2, 1]
Output: ["Gold Medal", "Silver Medal", "Bronze Medal","4","5"]
Explanation: The placements are [1^{st}, 2^{nd}, 3^{rd}, 4^{th}, 5^{th}].

Input: score = [10, 3, 8, 9, 4]
Output: ["Gold Medal", "5", "Bronze Medal", "Silver Medal", "4"]
Explanation: The placements are [1^{st}, 5^{th}, 3^{rd}, 2^{nd}, 4^{th}].

2.2 Array Partition

Given an integer array nums of 2n integers, group these integers into n pairs (a_1, b_1), (a_2, b_2), ..., (a_n, b_n) such that the sum of $\min(a_i, b_i)$ for all i is **maximized**. Return the maximized sum.

Input: nums = [1, 4, 3, 2]
Output: 4
Explanation: All possible pairings (ignoring the ordering of elements) are:
1. (1, 4), (2, 3) -> min(1, 4) + min(2, 3) = 1 + 2 = 3
2. (1, 3), (2, 4) -> min(1, 3) + min(2, 4) = 1 + 2 = 3
3. (1, 2), (3, 4) -> min(1, 2) + min(3, 4) = 1 + 3 = 4
So the maximum possible sum is 4.

Input: nums = [6, 2, 6, 5, 1, 2]
Output: 9

Explanation: The optimal pairing is (2, 1), (2, 5), (6, 6). min(2, 1) + min(2, 5) + min(6, 6) = 1 + 2 + 6 = 9.

2.3 Longest Harmonious Subsequence

We define a harmonious array as an array where the difference between its maximum value and its minimum value is **exactly** 1.

Given an integer array nums, return the length of its longest harmonious subsequence among all its possible subsequences.

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

Input: nums = [1, 3, 2, 2, 5, 2, 3, 7]
Output: 5
Explanation: The longest harmonious subsequence is [3, 2, 2, 2, 3].

Input: nums = [1, 2, 3, 4]
Output: 2

Input: nums = [1, 1, 1, 1]
Output: 0

2.4 Fair Candy Swap

Alice and Bob have a different total number of candies. You are given two integer arrays aliceSizes and bobSizes where aliceSizes[i] is the number of candies of the i^{th} box of candy that Alice has and bobSizes[j] is the number of candies of the j^{th} box of candy that Bob has.

Since they are friends, they would like to exchange one candy box each so that after the exchange, they both have the same total amount of candy. The total amount of candy a person has is the sum of the number of candies in each box they have.

Return an integer array answer where answer[0] is the number of candies in the box that Alice must exchange, and answer[1] is the number of candies in the box that Bob must exchange. If there are multiple answers, you may **return any** one of them. It is guaranteed that at least one answer exists.

Input: aliceSizes = [1, 1], bobSizes = [2, 2]
Output: [1, 2]
Input: aliceSizes = [1, 2], bobSizes = [2, 3]
Output: [1, 2]
Input: aliceSizes = [2], bobSizes = [1, 3]
Output: [2, 3]

2.5 Sort Array by Parity

Given an integer array nums, move all the even integers at the beginning of the array followed by all the odd integers.
Return **any array** that satisfies this condition.

Input: nums = [3, 1, 2, 4]

Output: [2, 4, 3, 1]

Explanation: The outputs [4, 2, 3, 1], [2, 4, 1, 3], and [4, 2, 1, 3] would also be accepted.

Input: nums = [0]

Output: [0]

2.6 Matrix Cells in Distance Order

You are given four integers row, cols, rCenter, and cCenter. There is a rows x cols matrix and you are on the cell with the coordinates (rCenter, cCenter).

Return the coordinates of all cells in the matrix, sorted by their **distance** from (rCenter, cCenter) from the smallest distance to the largest distance. You may return the answer in **any order** that satisfies this condition.

The **distance** between two cells (r_1, c_1) and (r_2, c_2) is $|r_1 - r_2| + |c_1 - c_2|$.

Input: rows = 1, cols = 2, rCenter = 0, cCenter = 0

Output: [[0, 0], [0, 1]]

Explanation: The distances from (0, 0) to other cells are: [0, 1]

Input: rows = 2, cols = 2, rCenter = 0, cCenter = 1

Output: [[0, 1], [0, 0], [1, 1], [1, 0]]

Explanation: The distances from (0, 1) to other cells are: [0, 1, 1, 2]

The answer [[0, 1], [1, 1], [0, 0], [1, 0]] would also be accepted as correct.

Input: rows = 2, cols = 3, rCenter = 1, cCenter = 2

Output: [[1, 2], [0, 2], [1, 1], [0, 1], [1, 0], [0, 0]]

Explanation: The distances from (1, 2) to other cells are: [0,1,1,2,2,3]

There are other answers that would also be accepted as correct, such as [[1,2],[1,1],[0,2],[1,0],[0,1],[0,0]].

2.7 Two City Scheduling

A company is planning to interview 2n people. Given the array costs where costs[i] = [$aCost_i$, $bCost_i$], the cost of flying the i^{th} person to city a is $aCost_i$, and the cost of flying the i^{th} person to city b is $bCost_i$. Return the minimum cost to fly every person to a city such that exactly n people arrive in each city.

Input: costs = [[10, 20], [30, 200], [400, 50], [30, 20]]

Output: 110

Explanation:

The first person goes to city A for a cost of 10.

The second person goes to city A for a cost of 30.

The third person goes to city B for a cost of 50.

The fourth person goes to city B for a cost of 20.

The total minimum cost is 10 + 30 + 50 + 20 = 110 to have half the people interviewing in each city.

Input: costs = [[259, 770], [448, 54], [926, 667], [184, 139], [840, 118], [577, 469]]

Output: 1859

Input: costs = [[515, 563], [451, 713], [537, 709], [343, 819], [855, 779], [457, 60], [650, 359], [631, 42]]

Output: 3086

2.8 Merge Sorted Array

You are given two integer arrays nums1 and nums2, sorted in **non-decreasing order**, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

Merge nums1 and nums2 into a single array sorted in **non-decreasing order**.

The final sorted array should not be returned by the function, but instead be *stored inside the array* nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n.

Input: nums1 = [1, 2, 3, 0, 0, 0], m = 3, nums2 = [2, 5, 6], n = 3

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

Explanation: The arrays we are merging are [1, 2, 3] and [2, 5, 6].

The result of the merge is [1, 2, 2, 3, 5, 6] with the underlined elements coming from nums1.

Input: nums1 = [1], m = 1, nums2 = [], n = 0

Output: [1]

Explanation: The arrays we are merging are [1] and [].

The result of the merge is [1].

Input: nums1 = [0], m = 0, nums2 = [1], n = 1

Output: [1]

Explanation: The arrays we are merging are [] and [1].

The result of the merge is [1].

Note that because m = 0, there are no elements in nums1. The 0 is only there to ensure the merge result can fit in nums1.

2.9 Insertion Sort List

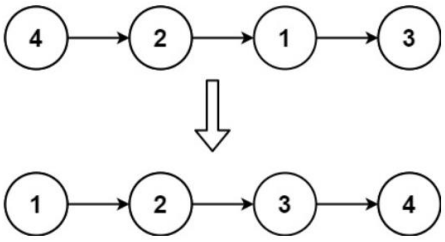
Given the head of a singly linked list, sort the list using **insertion sort**, and return *the sorted list's head*.

The steps of the **insertion sort** algorithm:

1. Insertion sort iterates, consuming one input element each repetition and growing a sorted output list.
2. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list and inserts it there.

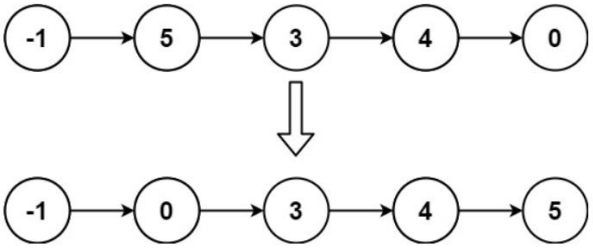
3. It repeats until no input elements remain.

The following is a graphical example of the insertion sort algorithm. The partially sorted list (black) initially contains only the first element in the list. One element (red) is removed from the input data and inserted in-place into the sorted list with each iteration.



Input: head = [4, 2, 1, 3]

Output: [1, 2, 3, 4]



Input: head = [-1, 5, 3, 4, 0]

Output: [-1, 0, 3, 4, 5]

2.10 Kth Smallest Element in a Sorted Matrix

Given an n x n matrix where each of the rows and columns is sorted in ascending order, return the k^{th} smallest element in the matrix.

Note that it is the k^{th} smallest element **in the sorted order**, not the k^{th} **distinct** element.
You must find a solution with a memory complexity better than $O(n^2)$.

Input: matrix = [[1, 5, 9], [10, 11, 13], [12, 13, 15]], k = 8

Output: 13

Explanation: The elements in the matrix are [1, 5, 9, 10, 11, 12, 13, **13**, 15], and the 8^{th} smallest number is 13

Input: matrix = [[-5]], k = 1

Output: -5

3. Cyclic Sort

3.1 Find All Duplicates in an Array

Given an integer array nums of length n where all the integers of nums are in the range [1, n] and each integer appears **once** or **twice**, return an array of all the integers that appears **twice**.

You must write an algorithm that runs in O(n) time and uses only constant extra space.

Input: nums = [4, 3, 2, 7, 8, 2, 3, 1]

Output: [2, 3]

Input: nums = [1, 1, 2]

Output: [1]

Input: nums = [1]

Output: []

3.2 Find All Numbers Disappeared in an Array

Given an array nums of n integers where nums[i] is in the range [1, n], return an array of all the integers in the range [1, n] that do not appear in nums.

Input: nums = [4, 3, 2, 7, 8, 2, 3, 1]

Output: [5, 6]

Input: nums = [1, 1]

Output: [2]

3.3 Missing Number

Given an array nums containing n distinct numbers in the range [0, n], return the only number in the range that is missing from the array.

Input: nums = [3, 0, 1]

Output: 2

Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.

Input: nums = [0, 1]

Output: 2

Explanation: n = 2 since there are 2 numbers, so all numbers are in the range [0,2]. 2 is the missing number in the range since it does not appear in nums.

Input: nums = [9, 6, 4, 2, 3, 5, 7, 0, 1]

Output: 8

Explanation: n = 9 since there are 9 numbers, so all numbers are in the range [0,9]. 8 is the missing number in the range since it does not appear in nums.

3.4 First Missing Positive

Given an unsorted integer array nums, return the smallest missing positive integer.

You must implement an algorithm that runs in O(n) time and uses O(1) auxiliary space.

Input: nums = [1, 2, 0]

Output: 3

Explanation: The numbers in the range [1, 2] are all in the array.

Input: nums = [3, 4,-1, 1]

Output: 2

Explanation: 1 is in the array but 2 is missing.

Input: nums = [7, 8, 9, 11, 12]

Output: 1

Explanation: The smallest positive integer 1 is missing.

4. Topological Sort

4.1 Find All Possible Recipes from Given Supplies

You have information about n different recipes. You are given a string array recipes and a 2D string array ingredients. The i^{th} recipe has the name recipes[i], and you can **create** it if you have **all** the needed ingredients from ingredients[i]. Ingredients to a recipe may need to be created from **other** recipes, i.e., ingredients[i] may contain a string that is in recipes.

You are also given a string array supplies containing all the ingredients that you initially have, and you have an infinite supply of all of them.

Return a list of all the recipes that you can create. You may return the answer in **any order**.

Note that two recipes may contain each other in their ingredients.

Input: recipes = ["bread"], ingredients = [["yeast", "flour"]], supplies = ["yeast", "flour", "corn"]

Output: ["bread"]

Explanation:

We can create "bread" since we have the ingredients "yeast" and "flour".

Input: recipes = ["bread", "sandwich"], ingredients = [["yeast", "flour"], ["bread", "meat"]], supplies = ["yeast", "flour", "meat"]

Output: ["bread", "sandwich"]

Explanation:

We can create "bread" since we have the ingredients "yeast" and "flour".

We can create "sandwich" since we have the ingredient "meat" and can create the ingredient "bread".

Input: recipes = ["bread", "sandwich", "burger"], ingredients = [["yeast", "flour"], ["bread", "meat"], ["sandwich", "meat", "bread"]], supplies = ["yeast", "flour", "meat"]

Output: ["bread", "sandwich", "burger"]

Explanation:

We can create "bread" since we have the ingredients "yeast" and "flour".

We can create "sandwich" since we have the ingredient "meat" and can create the ingredient "bread".

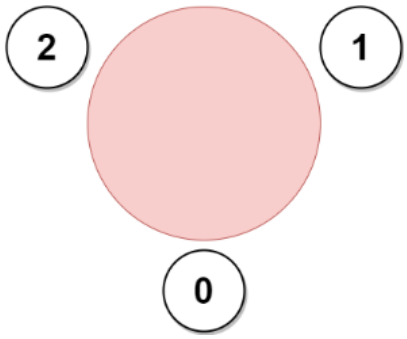
We can create "burger" since we have the ingredient "meat" and can create the ingredients "bread" and "sandwich".

4.2 Maximum Employees to Be Invited to a Meeting

A company is organizing a meeting and has a list of n employees, waiting to be invited. They have arranged for a large **circular** table, capable of seating **any number** of employees.

The employees are numbered from 0 to n - 1. Each employee has a **favorite** person and they will attend the meeting **only if** they can sit next to their favorite person at the table. The favorite person of an employee is **not** themself.

Given a **0-indexed** integer array favorite, where favorite[i] denotes the favorite person of the i^{th} employee, return the **maximum number of employees** that can be invited to the meeting.



Input: favorite = [2, 2, 1, 2]

Output: 3

Explanation:

The above figure shows how the company can invite employees 0, 1, and 2, and seat them at the round table.

All employees cannot be invited because employee 2 cannot sit beside employees 0, 1, and 3, simultaneously.

Note that the company can also invite employees 1, 2, and 3, and give them their desired seats.

The maximum number of employees that can be invited to the meeting is 3.

Input: favorite = [1, 2, 0]

Output: 3

Explanation:

Each employee is the favorite person of at least one other employee, and the only way the company can invite them is if they invite every employee.

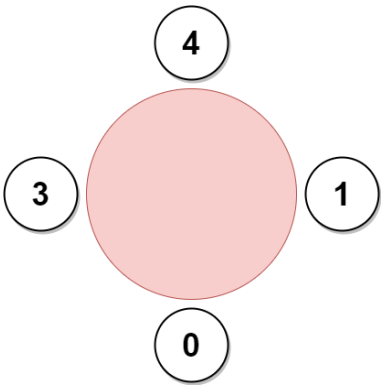
The seating arrangement will be the same as that in the figure given in example 1:

- Employee 0 will sit between employees 2 and 1.

- Employee 1 will sit between employees 0 and 2.

- Employee 2 will sit between employees 1 and 0.

The maximum number of employees that can be invited to the meeting is 3.



Input: favorite = [3, 0, 1, 4, 1]

Output: 4

Explanation:

The above figure shows how the company will invite employees 0, 1, 3, and 4, and seat them at the round table.

Employee 2 cannot be invited because the two spots next to their favorite employee 1 are taken.

So the company leaves them out of the meeting.

The maximum number of employees that can be invited to the meeting is 4.

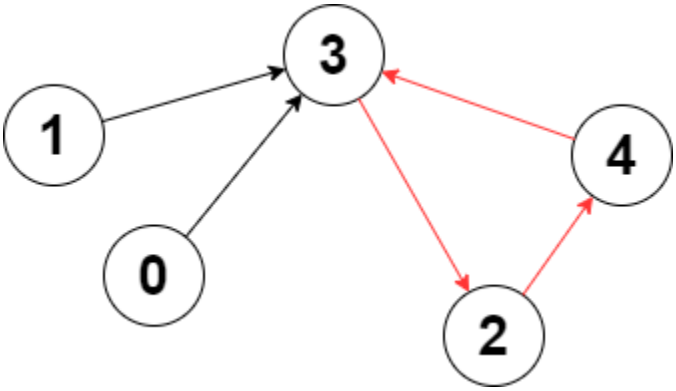
4.3 Longest Cycle in a Graph

You are given a **directed** graph of n nodes numbered from 0 to n - 1, where each node has **at most one** outgoing edge.

The graph is represented with a given **0-indexed** array edges of size n, indicating that there is a directed edge from node i to node edges[i]. If there is no outgoing edge from node i, then edges[i] == -1.

Return the length of the **longest** cycle in the graph. If no cycle exists, return -1.

A cycle is a path that starts and ends at the **same** node.

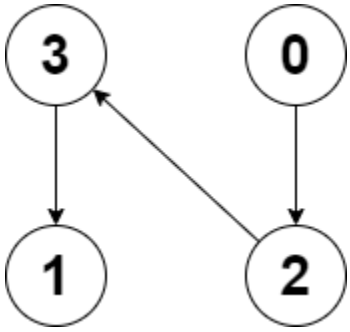


Input: edges = [3, 3, 4, 2, 3]

Output: 3

Explanation: The longest cycle in the graph is the cycle: 2 -> 4 -> 3 -> 2.

The length of this cycle is 3, so 3 is returned.



Input: edges = [2, -1, 3, 1]

Output: -1

Explanation: There are no cycles in this graph.

4.4 Build a Matrix with Conditions

You are given a **positive** integer k. You are also given:

- a 2D integer array rowConditions of size n where rowConditions[i] = [above$_i$, below$_i$], and
- a 2D integer array colConditions of size m where colConditions[i] = [left$_i$, right$_i$].

The two arrays contain integers from 1 to k.

You have to build a k x k matrix that contains each of the numbers from 1 to k **exactly once**. The remaining cells should have the value 0.

The matrix should also satisfy the following conditions:

- The number above$_i$ should appear in a **row** that is strictly **above** the row at which the number below$_i$ appears for all i from 0 to n - 1.
- The number left$_i$ should appear in a **column** that is strictly **left** of the column at which the number right$_i$ appears for all i from 0 to m - 1.

Return **any** matrix that satisfies the conditions. If no answer exists, return an empty matrix.

3	0	0
0	0	1
0	2	0

Input: k = 3, rowConditions = [[1, 2], [3, 2]], colConditions = [[2, 1], [3, 2]]

Output: [[3, 0, 0], [0, 0, 1], [0, 2, 0]]

Explanation: The diagram above shows a valid example of a matrix that satisfies all the conditions.

The row conditions are the following:

- Number 1 is in row 1, and number 2 is in row 2, so 1 is above 2 in the matrix.

- Number 3 is in row 0, and number 2 is in row 2, so 3 is above 2 in the matrix.

The column conditions are the following:

- Number 2 is in column 1, and number 1 is in column 2, so 2 is left of 1 in the matrix.

- Number 3 is in column 0, and number 2 is in column 1, so 3 is left of 2 in the matrix.

Note that there may be multiple correct answers.

Input: k = 3, rowConditions = [[1, 2], [2, 3], [3, 1], [2, 3]], colConditions = [[2, 1]]

Output: []

Explanation: From the first two conditions, 3 has to be below 1 but the third conditions needs 3 to be above 1 to be satisfied.

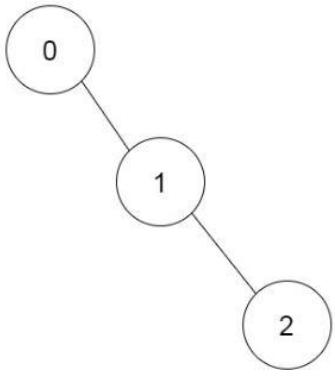
No matrix can satisfy all the conditions, so we return the empty matrix.

4.5 Count Ways to Build Rooms in an Ant Colony

You are an ant tasked with adding n new rooms numbered 0 to n-1 to your colony. You are given the expansion plan as a **0-indexed** integer array of length n, prevRoom, where prevRoom[i] indicates that you must build room prevRoom[i] before building room i, and these two rooms must be connected **directly**. Room 0 is already built, so prevRoom[0] = -1. The expansion plan is given such that once all the rooms are built, every room will be reachable from room 0.

You can only build **one room** at a time, and you can travel freely between rooms you have **already built** only if they are **connected**. You can choose to build **any room** as long as its **previous room** is already built.

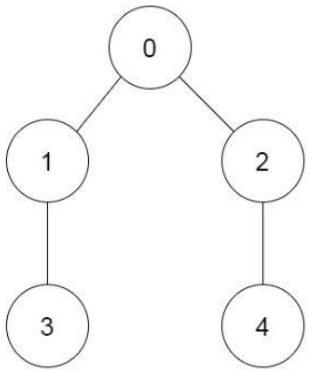
Return the **number of different orders** you can build all the rooms in. Since the answer may be large, return it **modulo** $10^9 + 7$.



Input: prevRoom = [-1, 0, 1]

Output: 1

Explanation: There is only one way to build the additional rooms: 0 → 1 → 2



Input: prevRoom = [-1, 0, 0, 1, 2]

Output: 6

Explanation:

The 6 ways are:

0 → 1 → 3 → 2 → 4

0 → 2 → 4 → 1 → 3

0 → 1 → 2 → 3 → 4

0 → 1 → 2 → 4 → 3

0 → 2 → 1 → 3 → 4

0 → 2 → 1 → 4 → 3

5. Strings

5.1 Longest Common Prefix

Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string "".

Input: strs = ["flower", "flow", "flight"]

Output: "fl"

Input: strs = ["dog", "racecar", "car"]

Output: ""

Explanation: There is no common prefix among the input strings.

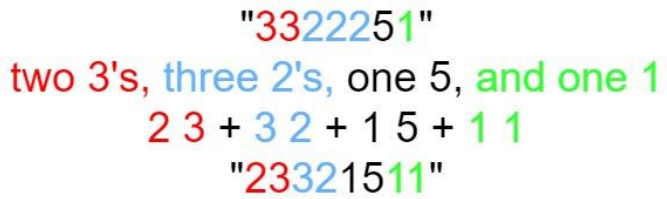
5.2 Count and Say

The **count-and-say** sequence is a sequence of digit strings defined by the recursive formula:

- countAndSay(1) = "1"
- countAndSay(n) is the way you would "say" the digit string from countAndSay(n-1), which is then converted into a different digit string.

To determine how you "say" a digit string, split it into the **minimal** number of substrings such that each substring contains exactly **one** unique digit. Then for each substring, say the number of digits, then say the digit. Finally, concatenate every said digit.

For example, the saying and conversion for digit string "3322251":



Given a positive integer n, return *the* n^{th} *term of the* ***count-and-say*** *sequence*.

Input: n = 1

Output: "1"

Explanation: This is the base case.

Input: n = 4

Output: "1211"

Explanation:

countAndSay(1) = "1"

countAndSay(2) = say "1" = one 1 = "11"

countAndSay(3) = say "11" = two 1's = "21"

countAndSay(4) = say "21" = one 2 + one 1 = "12" + "11" = "1211"

5.3 Group Anagrams

Given an array of strings strs, group **the anagrams** together. You can return the answer in **any order**.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Input: strs = ["eat", "tea", "tan", "ate", "nat", "bat"]

Output: [["bat"], ["nat", "tan"], ["ate", "eat", "tea"]]

Input: strs = [""]

Output: [[""]]

Input: strs = ["a"]

Output: [["a"]]

5.4 Restore IP Addresses

A **valid IP address** consists of exactly four integers separated by single dots. Each integer is between 0 and 255 (**inclusive**) and cannot have leading zeros.

- For example, "0.1.2.201" and "192.168.1.1" are **valid** IP addresses, but "0.011.255.245", "192.168.1.312" and "192.168@1.1" are **invalid** IP addresses.

Given a string s containing only digits, return all possible valid IP addresses that can be formed by inserting dots into s. You are **not** allowed to reorder or remove any digits in s. You may return the valid IP addresses in **any** order.

Input: s = "25525511135"

Output: ["255.255.11.135", "255.255.111.35"]

Input: s = "0000"

Output: ["0.0.0.0"]

Input: s = "101023"

Output: ["1.0.10.23", "1.0.102.3", "10.1.0.23", "10.10.2.3", "101.0.2.3"]

5.5 Interleaving String

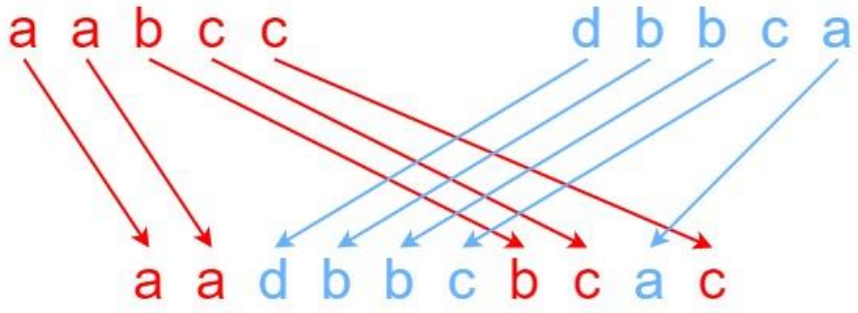
Given strings s1, s2, and s3, find whether s3 is formed by an **interleaving** of s1 and s2.

An **interleaving** of two strings s and t is a configuration where s and t are divided into n and m

Substrings respectively, such that:

- $s = s_1 + s_2 + ... + s_n$
- $t = t_1 + t_2 + ... + t_m$
- $|n - m| <= 1$
- The **interleaving** is $s_1 + t_1 + s_2 + t_2 + s_3 + t_3 + ...$ or $t_1 + s_1 + t_2 + s_2 + t_3 + s_3 + ...$

Note: a + b is the concatenation of strings a and b.



Input: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbcbcac"

Output: true

Explanation: One way to obtain s3 is:

Split s1 into s1 = "aa" + "bc" + "c", and s2 into s2 = "dbbc" + "a".

Interleaving the two splits, we get "aa" + "dbbc" + "bc" + "a" + "c" = "aadbbcbcac".

Since s3 can be obtained by interleaving s1 and s2, we return true.

Input: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbbaccc"

Output: false

Explanation: Notice how it is impossible to interleave s2 with any other string to obtain s3.

Input: s1 = "", s2 = "", s3 = ""

Output: true

5.6 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.

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

Output: true

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

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.

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

Output: false

5.7 Isomorphic Strings

Given two strings s and t, determine if they are isomorphic.

Two strings s and t are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character, but a character may map to itself.

Input: s = "egg", t = "add"

Output: true

Input: s = "foo", t = "bar"

Output: false

Input: s = "paper", t = "title"

Output: true

5.8 Bulls and Cows

You are playing the **Bulls and Cows** game with your friend.

You write down a secret number and ask your friend to guess what the number is. When your friend makes a guess, you provide a hint with the following info:

- The number of "bulls", which are digits in the guess that are in the correct position.
- The number of "cows", which are digits in the guess that are in your secret number but are located in the wrong position. Specifically, the non-bull digits in the guess that could be rearranged such that they become bulls.

Given the secret number secret and your friend's guess guess, return *the hint for your friend's guess*.

The hint should be formatted as "xAyB", where x is the number of bulls and y is the number of cows. Note that both secret and guess may contain duplicate digits.

Input: secret = "1807", guess = "7810"

Output: "1A3B"

Explanation: Bulls are connected with a '|' and cows are underlined:

"1807"

|

"7810"

Input: secret = "1123", guess = "0111"

Output: "1A1B"

Explanation: Bulls are connected with a '|' and cows are underlined:

"1123" "1123"

| or |

"0111" "0111"

Note that only one of the two unmatched 1s is counted as a cow since the non-bull digits can only be rearranged to allow one 1 to be a bull.

5.9 Remove Duplicate Letters

Given a string s, remove duplicate letters so that every letter appears once and only once. You must make sure your result is **the smallest in lexicographical order** among all possible results.

Input: s = "bcabc"

Output: "abc"

Input: s = "cbacdcbc"

Output: "acdb"

5.10 Maximum Product of Word Lengths

Given a string array words, return the maximum value of length(word[i]) * length(word[j]) where the two words do not share common letters. If no such two words exist, return 0.

Input: words = ["abcw","baz","foo","bar","xtfn","abcdef"]

Output: 16

Explanation: The two words can be "abcw", "xtfn".

Input: words = ["a","ab","abc","d","cd","bcd","abcd"]

Output: 4

Explanation: The two words can be "ab", "cd".

Input: words = ["a","aa","aaa","aaaa"]

Output: 0

Explanation: No such pair of words.

5.11 Zigzag Conversion

The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

P A H N

A P L S I I G

Y I R

And then read line by line: "PAHNAPLSIIGYIR"

Write the code that will take a string and make this conversion given a number of rows:

string convert(string s, int numRows);

Input: s = "PAYPALISHIRING", numRows = 3

Output: "PAHNAPLSIIGYIR"

Input: s = "PAYPALISHIRING", numRows = 4

Output: "PINALSIGYAHRPI"

Explanation:

P I N

A L S I G

Y A H R

P I

Input: s = "A", numRows = 1

Output: "A"

6. Stacks and Queues

6.1 Evaluate Reverse Polish Notation

You are given an array of strings tokens that represents an arithmetic expression in a Reverse Polish Notation.

Evaluate the expression. Return an integer that represents the value of the expression.

Note that:

- The valid operators are '+', '-', '*', and '/'.
- Each operand may be an integer or another expression.
- The division between two integers always **truncates toward zero**.
- There will not be any division by zero.
- The input represents a valid arithmetic expression in a reverse polish notation.
- The answer and all the intermediate calculations can be represented in a **32-bit** integer.

Input: tokens = ["2","1","+","3","*"]

Output: 9

Explanation: ((2 + 1) * 3) = 9

Input: tokens = ["4","13","5","/","+"]

Output: 6

Explanation: (4 + (13 / 5)) = 6

Input: tokens = ["10","6","9","3","+","-11","*","/","*","17","+","5","+"]

Output: 22

Explanation: ((10 * (6 / ((9 + 3) * -11))) + 17) + 5

= ((10 * (6 / (12 * -11))) + 17) + 5

= ((10 * (6 / -132)) + 17) + 5

= ((10 * 0) + 17) + 5

= (0 + 17) + 5

= 17 + 5

= 22

6.2 Min Stack

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the MinStack class:

- MinStack() initializes the stack object.
- void push(int val) pushes the element val onto the stack.
- void pop() removes the element on the top of the stack.
- int top() gets the top element of the stack.
- int getMin() retrieves the minimum element in the stack.

You must implement a solution with O(1) time complexity for each function.

Input: ["MinStack", "push", "push", "push", "getMin", "pop", "top", "getMin"]

[[], [-2], [0], [-3], [], [], [], []]

Output: [null, null, null, null, -3, null, 0, -2]

Explanation

MinStack minStack = new MinStack();

minStack.push(-2);

minStack.push(0);

minStack.push(-3);

minStack.getMin(); // return -3

minStack.pop();

minStack.top(); // return 0

minStack.getMin(); // return -2

6.3 Implement Queue using Stacks

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).

Implement the MyQueue class:

- void push(int x) Pushes element x to the back of the queue.
- int pop() Removes the element from the front of the queue and returns it.
- int peek() Returns the element at the front of the queue.
- boolean empty() Returns true if the queue is empty, false otherwise.

Notes:

- You must use **only** standard operations of a stack, which means only push to top, peek/pop from top, size, and is empty operations are valid.
- Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations.

Input: ["MyQueue", "push", "push", "peek", "pop", "empty"]

[[], [1], [2], [], [], []]

Output: [null, null, null, 1, 1, false]

Explanation

MyQueue myQueue = new MyQueue();

myQueue.push(1); // queue is: [1]

myQueue.push(2); // queue is: [1, 2] (leftmost is front of the queue)

myQueue.peek(); // return 1

myQueue.pop(); // return 1, queue is [2]

myQueue.empty(); // return false

6.4 Remove Duplicate Letters

Given a string s, remove duplicate letters so that every letter appears once and only once. You must make sure your result is **the smallest in lexicographical order** among all possible results.

Input: s = "bcabc"

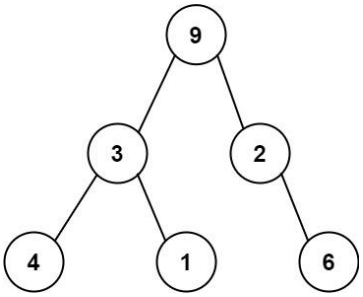
Output: "abc"

Input: s = "cbacdcbc"

Output: "acdb"

6.5 Verify Preorder Serialization of a Binary Tree

One way to serialize a binary tree is to use **preorder traversal**. When we encounter a non-null node, we record the node's value. If it is a null node, we record using a sentinel value such as '#'.



For example, the above binary tree can be serialized to the string "9,3,4,#,#,1,#,#,2,#,6,#,#", where '#' represents a null node.

Given a string of comma-separated values preorder, return true if it is a correct preorder traversal serialization of a binary tree.

It is **guaranteed** that each comma-separated value in the string must be either an integer or a character '#' representing null pointer.

You may assume that the input format is always valid.

- For example, it could never contain two consecutive commas, such as "1,,3".

Note: You are not allowed to reconstruct the tree.

Input: preorder = "9, 3, 4, #, #, 1, #, #, 2, #, 6, #, #"

Output: true

Input: preorder = "1, #"

Output: false

Input: preorder = "9, #, #, 1"

Output: false

6.6 Mini Parser

Given a string s represents the serialization of a nested list, implement a parser to deserialize it and return the deserialized NestedInteger.

Each element is either an integer or a list whose elements may also be integers or other lists.

Input: s = "324"

Output: 324

Explanation: You should return a NestedInteger object which contains a single integer 324.

Input: s = "[123, [456, [789]]]"

Output: [123, [456, [789]]]

Explanation: Return a NestedInteger object containing a nested list with 2 elements:

1. An integer containing value 123.
2. A nested list containing two elements:
   i. An integer containing value 456.
   ii. A nested list with one element:
      a. An integer containing value 789

6.7 Decode String

Given an encoded string, return its decoded string.

The encoding rule is: k[encoded_string], where the encoded_string inside the square brackets is being repeated exactly k times. Note that k is guaranteed to be a positive integer.

You may assume that the input string is always valid; there are no extra white spaces, square brackets are well-formed, etc. Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers, k. For example, there will not be input like 3a or 2[4].

The test cases are generated so that the length of the output will never exceed 10^5.

Input: s = "3[a]2[bc]"

Output: "aaabcbc"

Input: s = "3[a2[c]]"

Output: "accaccacc"

Input: s = "2[abc]3[cd]ef"

Output: "abcabccdcdcdef"

6.8 Next Greater Element I

The **next greater element** of some element x in an array is the **first greater** element that is **to the right** of x in the same array.

You are given two **distinct 0-indexed** integer arrays nums1 and nums2, where nums1 is a subset of nums2.

For each 0 <= i < nums1.length, find the index j such that nums1[i] == nums2[j] and determine the **next greater element** of nums2[j] in nums2. If there is no next greater element, then the answer for this query is -1.

Return an array ans of length nums1.length such that ans[i] is the **next greater element** as described above.

Input: nums1 = [4, 1, 2], nums2 = [1, 3, 4, 2]

Output: [-1, 3, -1]

Explanation: The next greater element for each value of nums1 is as follows:

- 4 is underlined in nums2 = [1, 3, 4, 2]. There is no next greater element, so the answer is -1.

- 1 is underlined in nums2 = [1, 3, 4, 2]. The next greater element is 3.

- 2 is underlined in nums2 = [1, 3, 4, 2]. There is no next greater element, so the answer is -1.

Input: nums1 = [2, 4], nums2 = [1, 2, 3, 4]

Output: [3, -1]

Explanation: The next greater element for each value of nums1 is as follows:

- 2 is underlined in nums2 = [1, 2, 3, 4]. The next greater element is 3.

- 4 is underlined in nums2 = [1, 2, 3, 4]. There is no next greater element, so the answer is -1.

6.9 Baseball Game

You are keeping the scores for a baseball game with strange rules. At the beginning of the game, you start with an empty record.

You are given a list of strings operations, where operations[i] is the i^{th} operation you must apply to the record and is one of the following:

- An integer x.
    - Record a new score of x.
- '+'.
    - Record a new score that is the sum of the previous two scores.
- 'D'.
    - Record a new score that is the double of the previous score.
- 'C'.
    - Invalidate the previous score, removing it from the record.

Return *the sum of all the scores on the record after applying all the operations*.

The test cases are generated such that the answer and all intermediate calculations fit in a **32-bit** integer and that all operations are valid.

Input: ops = ["5","2","C","D","+"]

Output: 30

Explanation:

"5" - Add 5 to the record, record is now [5].

"2" - Add 2 to the record, record is now [5, 2].

"C" - Invalidate and remove the previous score, record is now [5].

"D" - Add 2 * 5 = 10 to the record, record is now [5, 10].

"+" - Add 5 + 10 = 15 to the record, record is now [5, 10, 15].

The total sum is 5 + 10 + 15 = 30.

Input: ops = ["5","-2","4","C","D","9","+","+"]

Output: 27

Explanation:

"5" - Add 5 to the record, record is now [5].

"-2" - Add -2 to the record, record is now [5, -2].

"4" - Add 4 to the record, record is now [5, -2, 4].

"C" - Invalidate and remove the previous score, record is now [5, -2].

"D" - Add 2 * -2 = -4 to the record, record is now [5, -2, -4].

"9" - Add 9 to the record, record is now [5, -2, -4, 9].

"+" - Add -4 + 9 = 5 to the record, record is now [5, -2, -4, 9, 5].

"+" - Add 9 + 5 = 14 to the record, record is now [5, -2, -4, 9, 5, 14].

The total sum is 5 + -2 + -4 + 9 + 5 + 14 = 27.

Input: ops = ["1","C"]

Output: 0

Explanation:

"1" - Add 1 to the record, record is now [1].

"C" - Invalidate and remove the previous score, record is now [].

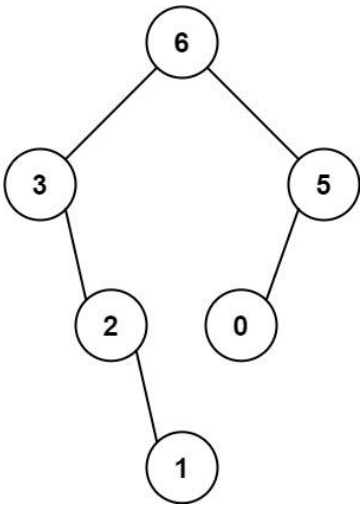
Since the record is empty, the total sum is 0.

6.10 Maximum Binary Tree

You are given an integer array nums with no duplicates. A **maximum binary tree** can be built recursively from nums using the following algorithm:

1. Create a root node whose value is the maximum value in nums.
2. Recursively build the left subtree on the **subarray prefix** to the **left** of the maximum value.
3. Recursively build the right subtree on the **subarray suffix** to the **right** of the maximum value.

Return the **maximum binary tree** built from nums.

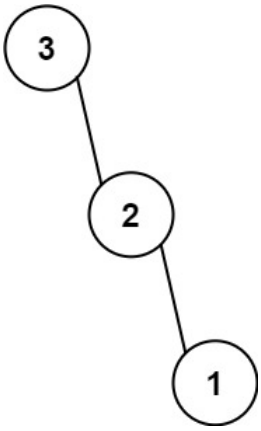


Input: nums = [3, 2, 1, 6, 0, 5]

Output: [6, 3, 5, null, 2, 0, null, null, 1]

Explanation: The recursive calls are as follow:

- The largest value in [3, 2, 1, 6, 0, 5] is 6. Left prefix is [3, 2, 1] and right suffix is [0,5].
  - The largest value in [3, 2, 1] is 3. Left prefix is [] and right suffix is [2, 1].
    - Empty array, so no child.
    - The largest value in [2, 1] is 2. Left prefix is [] and right suffix is [1].
      - Empty array, so no child.
      - Only one element, so child is a node with value 1.
  - The largest value in [0, 5] is 5. Left prefix is [0] and right suffix is [].
    - Only one element, so child is a node with value 0.
    - Empty array, so no child.



Input: nums = [3, 2, 1]

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

6.11 Daily Temperatures

Given an array of integers temperatures represents the daily temperatures, return *an array* answer *such that* answer[i] *is the number of days you have to wait after the* i^{th} *day to get a warmer temperature*. If there is no future day for which this is possible, keep answer[i] == 0 instead.

Input: temperatures = [73, 74, 75, 71, 69, 72, 76, 73]

Output: [1, 1, 4, 2, 1, 1, 0, 0]

Input: temperatures = [30, 40, 50, 60]

Output: [1, 1, 1, 0]

Input: temperatures = [30, 60, 90]

Output: [1, 1, 0]

6.12 Validate Stack Sequences

Given two integer arrays pushed and popped each with distinct values, return true if this could have been the result of a sequence of push and pop operations on an initially empty stack, or false otherwise.

Input: pushed = [1, 2, 3, 4, 5], popped = [4, 5, 3, 2, 1]

Output: true

Explanation: We might do the following sequence:

push(1), push(2), push(3), push(4),

pop() -> 4,

push(5),

pop() -> 5, pop() -> 3, pop() -> 2, pop() -> 1

Input: pushed = [1, 2, 3, 4, 5], popped = [4, 3, 5, 1, 2]

Output: false

Explanation: 1 cannot be popped before 2.

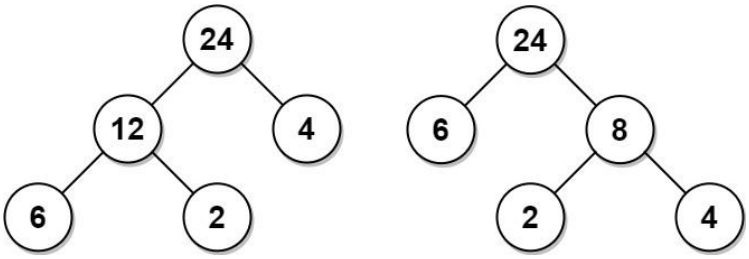
6.13 Minimum Cost Tree from Leaf Values

Given an array arr of positive integers, consider all binary trees such that:

- Each node has either 0 or 2 children;
- The values of arr correspond to the values of each **leaf** in an in-order traversal of the tree.
- The value of each non-leaf node is equal to the product of the largest leaf value in its left and right subtree, respectively.

Among all possible binary trees considered, return *the smallest possible sum of the values of each non-leaf node*. It is guaranteed this sum fits into a **32-bit** integer.

A node is a **leaf** if and only if it has zero children.

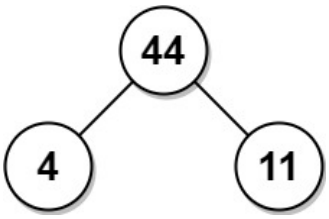


Input: arr = [6, 2, 4]

Output: 32

Explanation: There are two possible trees shown.

The first has a non-leaf node sum 36, and the second has non-leaf node sum 32.



Input: arr = [4, 11]

Output: 44

6.14 Build an Array with Stack Operations

You are given an integer array target and an integer n.

You have an empty stack with the two following operations:

- **"Push"**: pushes an integer to the top of the stack.
- **"Pop"**: removes the integer on the top of the stack.

You also have a stream of the integers in the range [1, n].

Use the two stack operations to make the numbers in the stack (from the bottom to the top) equal to target. You should follow the following rules:

- If the stream of the integers is not empty, pick the next integer from the stream and push it to the top of the stack.
- If the stack is not empty, pop the integer at the top of the stack.
- If, at any moment, the elements in the stack (from the bottom to the top) are equal to target, do not read new integers from the stream and do not do more operations on the stack.

Return the stack operations needed to build target following the mentioned rules. If there are multiple valid answers, return **any of them**.

Input: target = [1, 3], n = 3

Output: ["Push", "Push", "Pop", "Push"]

Explanation: Initially the stack s is empty. The last element is the top of the stack.

Read 1 from the stream and push it to the stack. s = [1].

Read 2 from the stream and push it to the stack. s = [1, 2].

Pop the integer on the top of the stack. s = [1].

Read 3 from the stream and push it to the stack. s = [1, 3].

Input: target = [1, 2, 3], n = 3

Output: ["Push", "Push", "Push"]

Explanation: Initially the stack s is empty. The last element is the top of the stack.

Read 1 from the stream and push it to the stack. s = [1].

Read 2 from the stream and push it to the stack. s = [1, 2].

Read 3 from the stream and push it to the stack. s = [1, 2, 3].

Input: target = [1, 2], n = 4

Output: ["Push", "Push"]

Explanation: Initially the stack s is empty. The last element is the top of the stack.

Read 1 from the stream and push it to the stack. s = [1].

Read 2 from the stream and push it to the stack. s = [1, 2].

Since the stack (from the bottom to the top) is equal to target, we stop the stack operations.

The answers that read integer 3 from the stream are not accepted.

6.15 The Number of Weak Characters in the Game

You are playing a game that contains multiple characters, and each of the characters has **two** main properties: **attack** and **defense**. You are given a 2D integer array properties where properties[i] = [$attack_i$, $defense_i$] represents the properties of the i^{th} character in the game.

A character is said to be **weak** if any other character has **both** attack and defense levels **strictly greater** than this character's attack and defense levels. More formally, a character i is said to be **weak** if there exists another character j where $attack_j > attack_i$ and $defense_j > defense_i$.

Return the number of **weak** characters.

Input: properties = [[5, 5], [6, 3], [3, 6]]

Output: 0

Explanation: No character has strictly greater attack and defense than the other.

Input: properties = [[2, 2], [3, 3]]

Output: 1

Explanation: The first character is weak because the second character has a strictly greater attack and defense.

Input: properties = [[1, 5], [10, 4], [4, 3]]

Output: 1

Explanation: The third character is weak because the second character has a strictly greater attack and defense.

6.16 Number of Visible People in a Queue

There are n people standing in a queue, and they numbered from 0 to n - 1 in **left to right** order. You are given an array heights of **distinct** integers where heights[i] represents the height of the i^{th} person.

A person can **see** another person to their right in the queue if everybody in between is **shorter** than both of them. More formally, the i^{th} person can see the j^{th} person if i < j and min(heights[i], heights[j]) > max(heights[i+1], heights[i+2], ..., heights[j-1]).

Return an array answer of length n where answer[i] is the **number of people** the i^{th} person can **see** to their right in the queue.

Input: heights = [10, 6, 8, 5, 11, 9]

Output: [3, 1, 2, 1, 1, 0]

Explanation:

Person 0 can see person 1, 2, and 4.

Person 1 can see person 2.

Person 2 can see person 3 and 4.

Person 3 can see person 4.

Person 4 can see person 5.

Person 5 can see no one since nobody is to the right of them.

Input: heights = [5, 1, 2, 3, 10]

Output: [4, 1, 1, 1, 0]

6.17 Number of People Aware of a Secret

On day 1, one person discovers a secret.

You are given an integer delay, which means that each person will share the secret with a new person every day, starting from delay days after discovering the secret. You are also given an integer forget, which means that each person will forget the secret forget days after discovering it. A person cannot share the secret on the same day they forgot it, or on any day afterwards.

Given an integer n, return the number of people who know the secret at the end of day n. Since the answer may be very large, return it modulo 109 + 7.

Input: n = 6, delay = 2, forget = 4

Output: 5

Explanation:

Day 1: Suppose the first person is named A. (1 person)

Day 2: A is the only person who knows the secret. (1 person)

Day 3: A shares the secret with a new person, B. (2 people)

Day 4: A shares the secret with a new person, C. (3 people)

Day 5: A forgets the secret, and B shares the secret with a new person, D. (3 people)

Day 6: B shares the secret with E, and C shares the secret with F. (5 people)

Input: n = 4, delay = 1, forget = 3

Output: 6

Explanation:

Day 1: The first person is named A. (1 person)

Day 2: A shares the secret with B. (2 people)

Day 3: A and B share the secret with 2 new people, C and D. (4 people)

Day 4: A forgets the secret. B, C, and D share the secret with 3 new people. (6 people)

6.18 Reveal Cards in Increasing Order

You are given an integer array deck. There is a deck of cards where every card has a unique integer. The integer on the i^{th} card is deck[i].

You can order the deck in any order you want. Initially, all the cards start face down (unrevealed) in one deck.

You will do the following steps repeatedly until all cards are revealed:

1. Take the top card of the deck, reveal it, and take it out of the deck.
2. If there are still cards in the deck then put the next top card of the deck at the bottom of the deck.
3. If there are still unrevealed cards, go back to step 1. Otherwise, stop.

Return an ordering of the deck that would reveal the cards in increasing order.

Note that the first entry in the answer is considered to be the top of the deck.

Input: deck = [17, 13, 11, 2, 3, 5, 7]

Output: [2, 13, 3, 11, 5, 17, 7]

Explanation:

We get the deck in the order [17, 13, 11, 2, 3, 5, 7] (this order does not matter), and reorder it.

After reordering, the deck starts as [2, 13, 3, 11, 5, 17, 7], where 2 is the top of the deck.

We reveal 2, and move 13 to the bottom. The deck is now [3, 11, 5, 17, 7, 13].

We reveal 3, and move 11 to the bottom. The deck is now [5, 17, 7, 13, 11].

We reveal 5, and move 17 to the bottom. The deck is now [7, 13, 11, 17].

We reveal 7, and move 13 to the bottom. The deck is now [11, 17, 13].

We reveal 11, and move 17 to the bottom. The deck is now [13, 17].

We reveal 13, and move 17 to the bottom. The deck is now [17].

We reveal 17.

Since all the cards revealed are in increasing order, the answer is correct.

Input: deck = [1, 1000]

Output: [1, 1000]

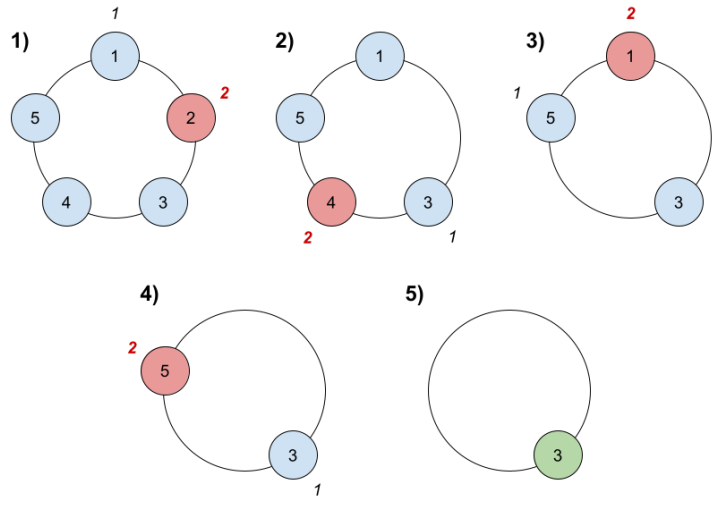
6.19 Find the Winner of the Circular Game

There are n friends that are playing a game. The friends are sitting in a circle and are numbered from 1 to n in **clockwise order**. More formally, moving clockwise from the i^{th} friend brings you to the $(i+1)^{th}$ friend for 1 <= i < n, and moving clockwise from the n^{th} friend brings you to the 1^{st} friend.

The rules of the game are as follows:

1. **Start** at the 1^{st} friend.
2. Count the next k friends in the clockwise direction **including** the friend you started at. The counting wraps around the circle and may count some friends more than once.
3. The last friend you counted leaves the circle and loses the game.
4. If there is still more than one friend in the circle, go back to step 2 **starting** from the friend **immediately clockwise** of the friend who just lost and repeat.
5. Else, the last friend in the circle wins the game.

Given the number of friends, n, and an integer k, return the winner of the game.



Input: n = 5, k = 2

Output: 3

Explanation: Here are the steps of the game:

1) Start at friend 1.

2) Count 2 friends clockwise, which are friends 1 and 2.

3) Friend 2 leaves the circle. Next start is friend 3.

4) Count 2 friends clockwise, which are friends 3 and 4.

5) Friend 4 leaves the circle. Next start is friend 5.

6) Count 2 friends clockwise, which are friends 5 and 1.

7) Friend 1 leaves the circle. Next start is friend 3.

8) Count 2 friends clockwise, which are friends 3 and 5.

9) Friend 5 leaves the circle. Only friend 3 is left, so they are the winner.

Input: n = 6, k = 5

Output: 1

Explanation: The friends leave in this order: 5, 4, 6, 2, 3. The winner is friend 1.

6.20 Design Circular Deque

Design your implementation of the circular double-ended queue (deque).

Implement the MyCircularDeque class:

- MyCircularDeque(int k) Initializes the deque with a maximum size of k.
- boolean insertFront() Adds an item at the front of Deque. Returns true if the operation is successful, or false otherwise.
- boolean insertLast() Adds an item at the rear of Deque. Returns true if the operation is successful, or false otherwise.
- boolean deleteFront() Deletes an item from the front of Deque. Returns true if the operation is successful, or false otherwise.
- boolean deleteLast() Deletes an item from the rear of Deque. Returns true if the operation is successful, or false otherwise.
- int getFront() Returns the front item from the Deque. Returns -1 if the deque is empty.
- int getRear() Returns the last item from Deque. Returns -1 if the deque is empty.
- boolean isEmpty() Returns true if the deque is empty, or false otherwise.
- boolean isFull() Returns true if the deque is full, or false otherwise.

Input: ["MyCircularDeque", "insertLast", "insertLast", "insertFront", "insertFront", "getRear", "isFull", "deleteLast", "insertFront", "getFront"]

[[3], [1], [2], [3], [4], [], [], [], [4], []]

Output: [null, true, true, true, false, 2, true, true, true, 4]

Explanation:

MyCircularDeque myCircularDeque = new MyCircularDeque(3);

myCircularDeque.insertLast(1); // return True

myCircularDeque.insertLast(2); // return True

myCircularDeque.insertFront(3); // return True

myCircularDeque.insertFront(4); // return False, the queue is full.

myCircularDeque.getRear(); // return 2

myCircularDeque.isFull(); // return True

myCircularDeque.deleteLast(); // return True

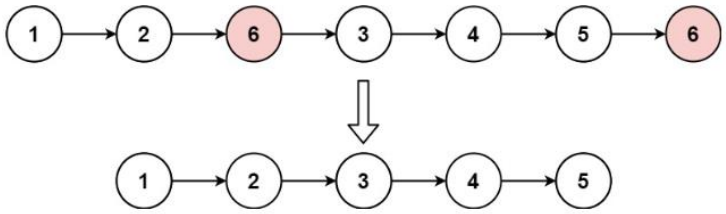
myCircularDeque.insertFront(4); // return True

myCircularDeque.getFront(); // return 4

7. Node Based Data Structures

7.1 Remove Linked List Elements

Given the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return the new head.

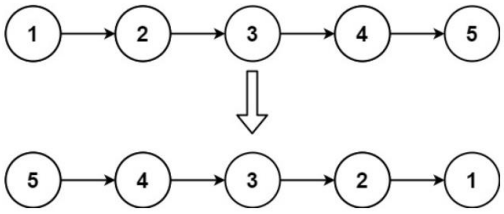


Input: head = [1, 2, 6, 3, 4, 5, 6], val = 6
Output: [1, 2, 3, 4, 5]
Input: head = [ ], val = 1
Output: [ ]
Input: head = [7, 7, 7, 7], val = 7
Output: [ ]

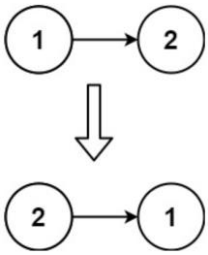
7.2 Reverse Linked List

Given the head of a singly linked list, reverse the list, and return the reversed list.

Input: head = [1, 2, 3, 4, 5]
Output: [5, 4, 3, 2, 1]

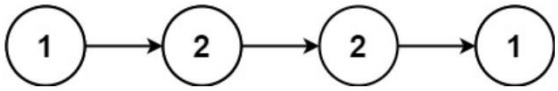


Input: head = [1, 2]
Output: [2, 1]

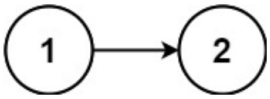


7.3 Palindrome Linked List

Given the head of a singly linked list, return true if it is a palindrome or false otherwise.



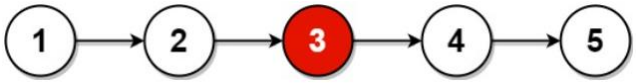
Input: head = [1, 2, 2, 1]
Output: true



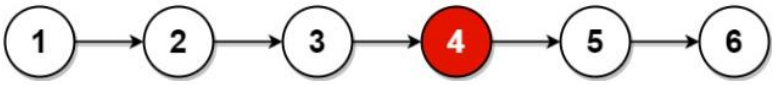
Input: head = [1, 2]
Output: false

7.4 Middle of the Linked List

Given the head of a singly linked list, return the middle node of the linked list. If there are two middle nodes, return the second middle node.



Input: head = [1, 2, 3, 4, 5]
Output: [3, 4, 5]
Explanation: The middle node of the list is node 3.

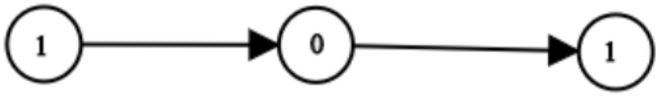


Input: head = [1, 2, 3, 4, 5, 6]
Output: [4, 5, 6]
Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.

7.5 Convert Binary Number in a Linked List to Integer

Given head which is a reference node to a singly-linked list. The value of each node in the linked list is either 0 or 1. The linked list holds the binary representation of a number.

Return the decimal value of the number in the linked list. The most significant bit is at the head of the linked list.

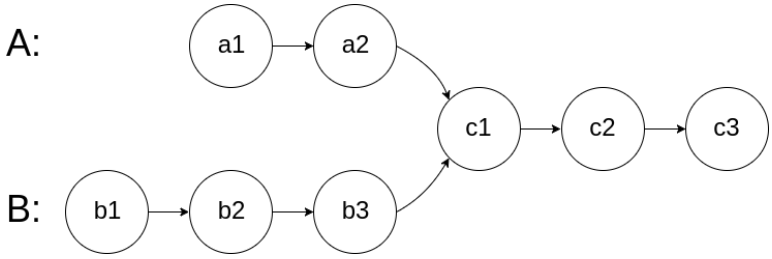


Input: head = [1, 0, 1]
Output: 5
Explanation: (101) in base 2 = (5) in base 10
Input: head = [0]
Output: 0

7.6 Intersection of Two Linked Lists

Given the heads of two singly linked-lists headA and headB, return *the node at which the two lists intersect*. If the two linked lists have no intersection at all, return null.

For example, the following two linked lists begin to intersect at node c1:



The test cases are generated such that there are no cycles anywhere in the entire linked structure.

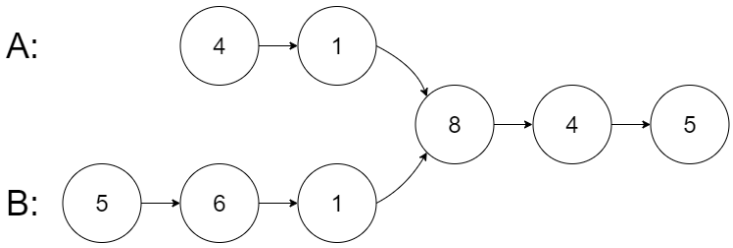
Note that the linked lists must **retain their original structure** after the function returns.

Custom Judge:

The inputs to the **judge** are given as follows (your program is **not** given these inputs):

- intersectVal - The value of the node where the intersection occurs. This is 0 if there is no intersected node.
- listA - The first linked list.
- listB - The second linked list.
- skipA - The number of nodes to skip ahead in listA (starting from the head) to get to the intersected node.
- skipB - The number of nodes to skip ahead in listB (starting from the head) to get to the intersected node.

The judge will then create the linked structure based on these inputs and pass the two heads, headA and headB to your program. If you correctly return the intersected node, then your solution will be **accepted**.



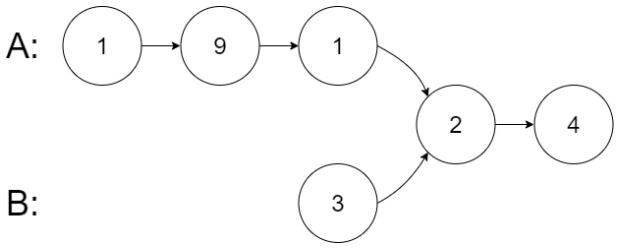
Input: intersectVal = 8, listA = [4, 1, 8, 4, 5], listB = [5, 6, 1, 8, 4, 5], skipA = 2, skipB = 3

Output: Intersected at '8'

Explanation: The intersected node's value is 8 (note that this must not be 0 if the two lists intersect).

From the head of A, it reads as [4, 1, 8, 4, 5]. From the head of B, it reads as [5,6,1,8,4,5]. There are 2 nodes before the intersected node in A; There are 3 nodes before the intersected node in B.

Note that the intersected node's value is not 1 because the nodes with value 1 in A and B (2nd node in A and 3rd node in B) are different node references. In other words, they point to two different locations in memory, while the nodes with value 8 in A and B (3rd node in A and 4th node in B) point to the same location in memory.

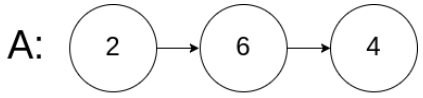


Input: intersectVal = 2, listA = [1, 9, 1, 2, 4], listB = [3, 2, 4], skipA = 3, skipB = 1

Output: Intersected at '2'

Explanation: The intersected node's value is 2 (note that this must not be 0 if the two lists intersect).

From the head of A, it reads as [1, 9, 1, 2, 4]. From the head of B, it reads as [3, 2, 4]. There are 3 nodes before the intersected node in A; There are 1 node before the intersected node in B.



Input: intersectVal = 0, listA = [2, 6, 4], listB = [1,5], skipA = 3, skipB = 2

Output: No intersection

Explanation: From the head of A, it reads as [2, 6, 4]. From the head of B, it reads as [1,5]. Since the two lists do not intersect, intersectVal must be 0, while skipA and skipB can be arbitrary values.

Explanation: The two lists do not intersect, so return null.

7.7 Delete Node in a Linked List

There is a singly-linked list head and we want to delete a node node in it.

You are given the node to be deleted node. You will **not be given access** to the first node of head.

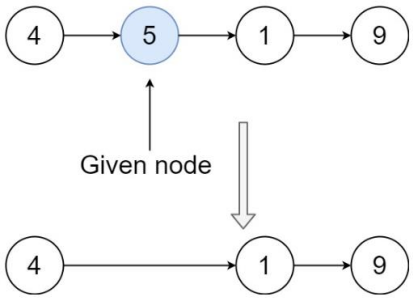
All the values of the linked list are **unique**, and it is guaranteed that the given node node is not the last node in the linked list.

Delete the given node. Note that by deleting the node, we do not mean removing it from memory. We mean:

- The value of the given node should not exist in the linked list.
- The number of nodes in the linked list should decrease by one.
- All the values before node should be in the same order.
- All the values after node should be in the same order.

Custom testing:

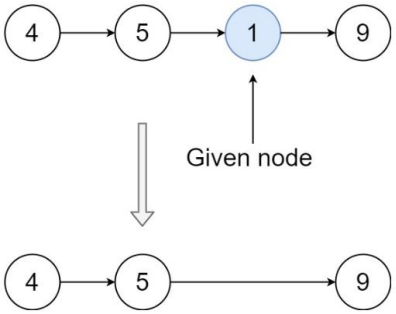
- For the input, you should provide the entire linked list head and the node to be given node. node should not be the last node of the list and should be an actual node in the list.
- We will build the linked list and pass the node to your function.
- The output will be the entire list after calling your function.



Input: head = [4, 5, 1, 9], node = 5

Output: [4, 1, 9]

Explanation: You are given the second node with value 5, the linked list should become 4 -> 1 -> 9 after calling your function.



Input: head = [4, 5, 1, 9], node = 1

Output: [4, 5, 9]

Explanation: You are given the third node with value 1, the linked list should become 4 -> 5 -> 9 after calling your function.

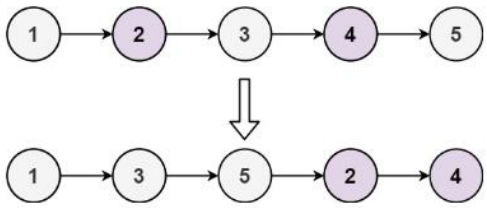
7.8 Odd Even Linked List

Given the head of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return the reordered list.

The **first** node is considered **odd**, and the **second** node is **even**, and so on.

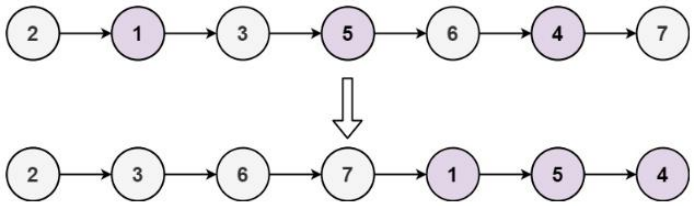
Note that the relative order inside both the even and odd groups should remain as it was in the input.

You must solve the problem in O(1) extra space complexity and O(n) time complexity.



Input: head = [1, 2, 3, 4, 5]

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



Input: head = [2, 1, 3, 5, 6, 4, 7]

Output: [2, 3, 6, 7, 1, 5, 4]

7.9 Design Twitter

Design a simplified version of Twitter where users can post tweets, follow/unfollow another user, and is able to see the 10 most recent tweets in the user's news feed.

Implement the Twitter class:

- Twitter() Initializes your twitter object.
- void postTweet(int userId, int tweetId) Composes a new tweet with ID tweetId by the user userId. Each call to this function will be made with a unique tweetId.
- List<Integer> getNewsFeed(int userId) Retrieves the 10 most recent tweet IDs in the user's news feed. Each item in the news feed must be posted by users who the user followed or by the user themself. Tweets must be **ordered from most recent to least recent**.

- void follow(int followerId, int followeeId) The user with ID followerId started following the user with ID followeeId.
- void unfollow(int followerId, int followeeId) The user with ID followerId started unfollowing the user with ID followeeId.

Input

["Twitter", "postTweet", "getNewsFeed", "follow", "postTweet", "getNewsFeed", "unfollow", "getNewsFeed"]

[[], [1, 5], [1], [1, 2], [2, 6], [1], [1, 2], [1]]

Output

[null, null, [5], null, null, [6, 5], null, [5]]

Explanation

Twitter twitter = new Twitter();

twitter.postTweet(1, 5); // User 1 posts a new tweet (id = 5).

twitter.getNewsFeed(1); // User 1's news feed should return a list with 1 tweet id -> [5]. return [5]

twitter.follow(1, 2); // User 1 follows user 2.

twitter.postTweet(2, 6); // User 2 posts a new tweet (id = 6).

twitter.getNewsFeed(1); // User 1's news feed should return a list with 2 tweet ids -> [6, 5]. Tweet id 6 should precede tweet id 5 because it is posted after tweet id 5.

twitter.unfollow(1, 2); // User 1 unfollows user 2.

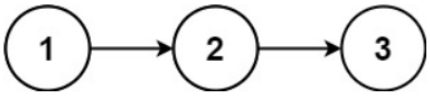
twitter.getNewsFeed(1); // User 1's news feed should return a list with 1 tweet id -> [5], since user 1 is no longer following user 2.

7.10 Linked List Random Node

Given a singly linked list, return a random node's value from the linked list. Each node must have the **same probability** of being chosen.

Implement the Solution class:

- Solution(ListNode head) Initializes the object with the head of the singly-linked list head.
- int getRandom() Chooses a node randomly from the list and returns its value. All the nodes of the list should be equally likely to be chosen.



Input

["Solution", "getRandom", "getRandom", "getRandom", "getRandom", "getRandom"]

[[[1, 2, 3]], [], [], [], [], []]

Output

[null, 1, 3, 2, 2, 3]

Explanation

Solution solution = new Solution([1, 2, 3]);

solution.getRandom(); // return 1

solution.getRandom(); // return 3

solution.getRandom(); // return 2

solution.getRandom(); // return 2

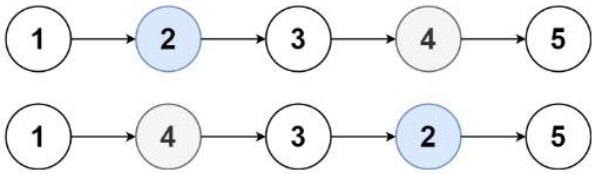
solution.getRandom(); // return 3

// getRandom() should return either 1, 2, or 3 randomly. Each element should have equal probability of returning.

7.11 Swapping Nodes in a Linked List

You are given the head of a linked list, and an integer k.

Return the head of the linked list after **swapping** the values of the k^{th} node from the beginning and the k^{th} node from the end (the list is **1-indexed**).



Input: head = [1, 2, 3, 4, 5], k = 2

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

Input: head = [7, 9, 6, 6, 7, 8, 3, 0, 9, 5], k = 5

Output: [7, 9, 6, 6, 8, 7, 3, 0, 9, 5]

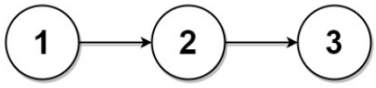
7.12 Split Linked List in Parts

Given the head of a singly linked list and an integer k, split the linked list into k consecutive linked list parts.

The length of each part should be as equal as possible: no two parts should have a size differing by more than one. This may lead to some parts being null.

The parts should be in the order of occurrence in the input list, and parts occurring earlier should always have a size greater than or equal to parts occurring later.

Return an array of the k parts.



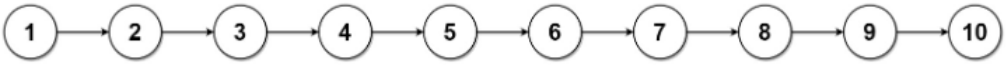
Input: head = [1, 2, 3], k = 5

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

Explanation:

The first element output[0] has output[0].val = 1, output[0].next = null.

The last element output[4] is null, but its string representation as a ListNode is [].



Input: head = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], k = 3

Output: [[1, 2, 3, 4], [5, 6, 7], [8, 9, 10]]

Explanation:

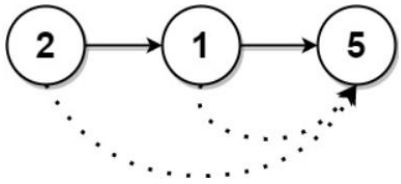
The input has been split into consecutive parts with size difference at most 1, and earlier parts are a larger size than the later parts.

7.13 Next Greater Node in Linked List

You are given the head of a linked list with n nodes.

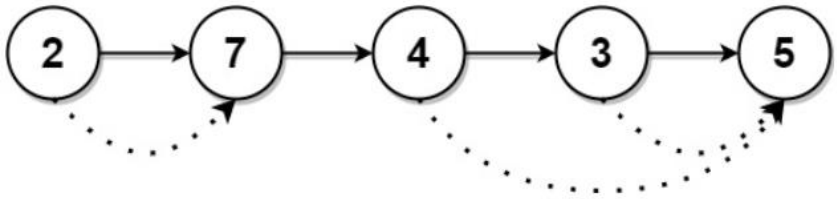
For each node in the list, find the value of the **next greater node**. That is, for each node, find the value of the first node that is next to it and has a **strictly larger** value than it.

Return an integer array answer where answer[i] is the value of the next greater node of the i^{th} node (**1-indexed**). If the i^{th} node does not have a next greater node, set answer[i] = 0.



Input: head = [2, 1, 5]

Output: [5, 5, 0]



Input: head = [2, 7, 4, 3, 5]

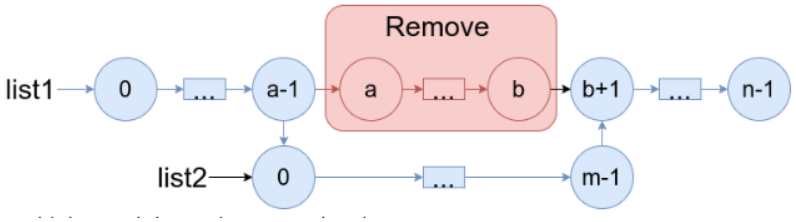
Output: [7, 0, 5, 5, 0]

7.14 Merge in Between Linked Lists

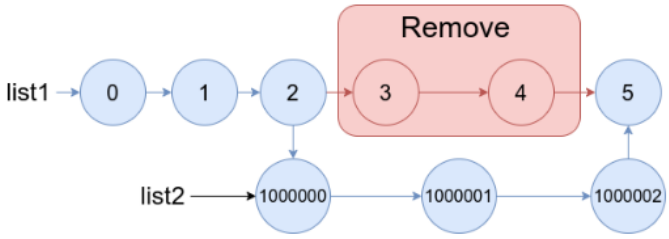
You are given two linked lists: list1 and list2 of sizes n and m respectively.

Remove list1's nodes from the a^{th} node to the b^{th} node, and put list2 in their place.

The blue edges and nodes in the following figure indicate the result:



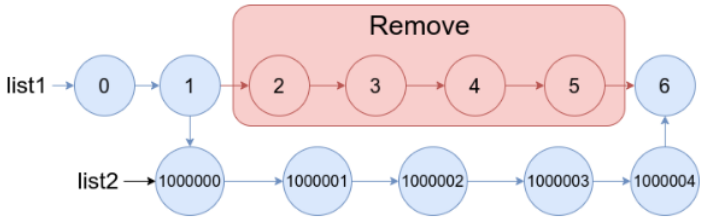
Build the result list and return its head.



Input: list1 = [0, 1, 2, 3, 4, 5], a = 3, b = 4, list2 = [1000000, 1000001, 1000002]

Output: [0, 1, 2, 1000000, 1000001, 1000002, 5]

Explanation: We remove the nodes 3 and 4 and put the entire list2 in their place. The blue edges and nodes in the above figure indicate the result.



Input: list1 = [0,1,2,3,4,5,6], a = 2, b = 5, list2 = [1000000,1000001,1000002,1000003,1000004]

Output: [0, 1, 1000000, 1000001, 1000002, 1000003, 1000004, 6]

Explanation: The blue edges and nodes in the above figure indicate the result.

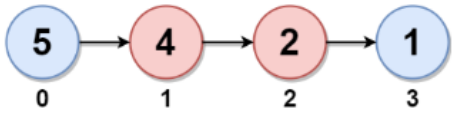
7.15 Maximum Twin Sum of a Linked List

In a linked list of size n, where n is even, the i^{th} node (0-indexed) of the linked list is known as the twin of the $(n-1-i)^{th}$ node, if 0 <= i <= (n / 2) - 1.

- For example, if n = 4, then node 0 is the twin of node 3, and node 1 is the twin of node 2. These are the only nodes with twins for n = 4.

The twin sum is defined as the sum of a node and its twin.

Given the head of a linked list with even length, return the maximum twin sum of the linked list.



Input: head = [5, 4, 2, 1]

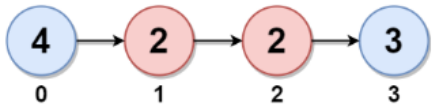
Output: 6

Explanation:

Nodes 0 and 1 are the twins of nodes 3 and 2, respectively. All have twin sum = 6.

There are no other nodes with twins in the linked list.

Thus, the maximum twin sum of the linked list is 6.



Input: head = [4, 2, 2, 3]

Output: 7

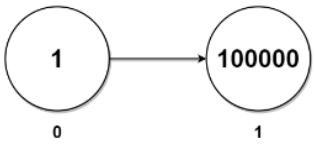
Explanation:

The nodes with twins present in this linked list are:

- Node 0 is the twin of node 3 having a twin sum of 4 + 3 = 7.

- Node 1 is the twin of node 2 having a twin sum of 2 + 2 = 4.

Thus, the maximum twin sum of the linked list is max(7, 4) = 7.



Input: head = [1, 100000]

Output: 100001

Explanation:

There is only one node with a twin in the linked list having twin sum of 1 + 100000 = 100001.

7.16 Merge Nodes in Between Zeros

You are given the head of a linked list, which contains a series of integers **separated** by 0's. The **beginning** and **end** of the linked list will have Node.val == 0.

For **every** two consecutive 0's, **merge** all the nodes lying in between them into a single node whose value is the **sum** of all the merged nodes. The modified list should not contain any 0's.

Return the head of the modified linked list.

Input: head = [0, 3, 1, 0, 4, 5, 2, 0]

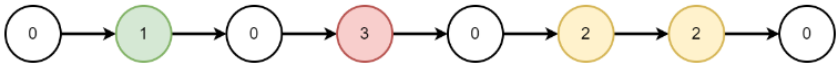
Output: [4, 11]

Explanation:

The above figure represents the given linked list. The modified list contains

- The sum of the nodes marked in green: 3 + 1 = 4.

- The sum of the nodes marked in red: 4 + 5 + 2 = 11.



Input: head = [0, 1, 0, 3, 0, 2, 2, 0]

Output: [1, 3, 4]

Explanation:

The above figure represents the given linked list. The modified list contains

- The sum of the nodes marked in green: 1 = 1.

- The sum of the nodes marked in red: 3 = 3.

- The sum of the nodes marked in yellow: 2 + 2 = 4.

7.17 Remove Nodes from Linked List

You are given the head of a linked list.

Remove every node which has a node with a greater value anywhere to the right side of it.

Return the head of the modified linked list.

Input: head = [5, 2, 13, 3, 8]

Output: [13, 8]

Explanation: The nodes that should be removed are 5, 2 and 3.

- Node 13 is to the right of node 5.

- Node 13 is to the right of node 2.

- Node 8 is to the right of node 3.

Input: head = [1, 1, 1, 1]

Output: [1, 1, 1, 1]

Explanation: Every node has value 1, so no nodes are removed.

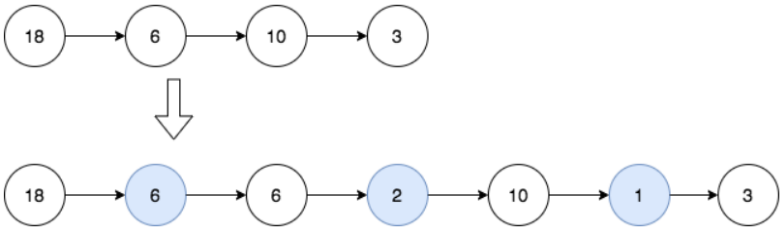
7.18 Insert Greatest Common Divisors in Linked List

Given the head of a linked list head, in which each node contains an integer value.

Between every pair of adjacent nodes, insert a new node with a value equal to the greatest common divisor of them.

Return the linked list after insertion.

The greatest common divisor of two numbers is the largest positive integer that evenly divides both numbers.



Input: head = [18, 6, 10, 3]

Output: [18, 6, 6, 2, 10, 1, 3]

Explanation: The 1st diagram denotes the initial linked list and the 2nd diagram denotes the linked list after inserting the new nodes (nodes in blue are the inserted nodes).

- We insert the greatest common divisor of 18 and 6 = 6 between the 1st and the 2nd nodes.

- We insert the greatest common divisor of 6 and 10 = 2 between the 2nd and the 3rd nodes.

- We insert the greatest common divisor of 10 and 3 = 1 between the 3rd and the 4th nodes.

There are no more adjacent nodes, so we return the linked list.

Input: head = [7]

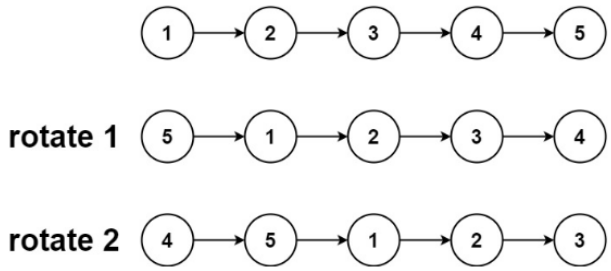
Output: [7]

Explanation: The 1st diagram denotes the initial linked list and the 2nd diagram denotes the linked list after inserting the new nodes.

There are no pairs of adjacent nodes, so we return the initial linked list.

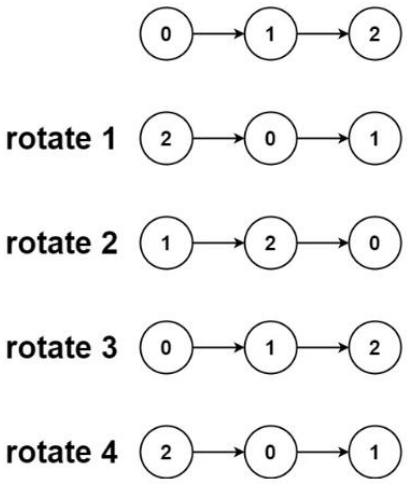
7.19 Rotate List

Given the head of a linked list, rotate the list to the right by k places.



Input: head = [1, 2, 3, 4, 5], k = 2

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



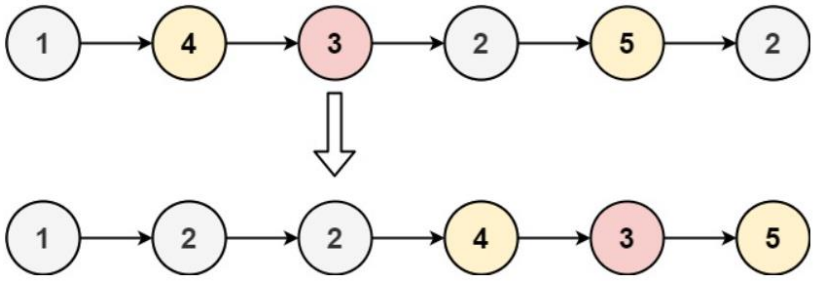
Input: head = [0, 1, 2], k = 4

Output: [2, 0, 1]

7.20 Partition List

Given the head of a linked list and a value x, partition it such that all nodes **less than** x come before nodes **greater than or equal** to x.

You should **preserve** the original relative order of the nodes in each of the two partitions.



Input: head = [1, 4, 3, 2, 5, 2], x = 3

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

Input: head = [2, 1], x = 2

Output: [1, 2]

8. Bit Manipulation

8.1 Raising Bacteria

You are a lover of bacteria. You want to raise some bacteria in a box. Initially, the box is empty. Each morning, you can put any number of bacteria into the box. And each night, every bacterium in the box will split into two bacteria. You hope to see exactly x bacteria in the box at some moment. What is the minimum number of bacteria you need to put into the box across those days?

Input: 5

Output: 2

Explanation: we can add one bacterium in the box in the first day morning and at the third morning there will be 4 bacteria in the box. Now we put one more resulting 5 in the box. We added 2 bacteria in the process so the answer is 2.

8.2 And Then There Were K

Given an integer n, find the maximum value of integer k such that the following condition holds:

n (n−1) & (n−2) & (n−3) & ... (k) = 0 where & denotes the bitwise AND operation.

Input: The first line contains a single integer t (1 ≤ t ≤ 3·104). Then t test cases follow.

The first line of each test case contains a single integer n (1 ≤ n ≤ 109).

3

2

5

17

Output: For each test case, output a single integer — the required integer k

1

3

15

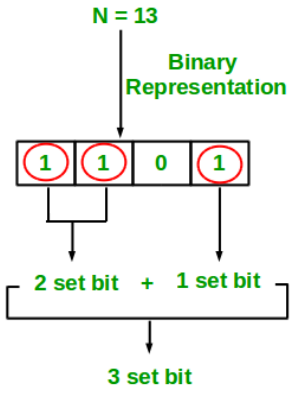
8.3 Count Set Bits in an Integer

Given an integer n, count the number of 1s in the binary representation of an integer.

Input: n = 13

Output: 3

Explanation: Binary representation of 13 is 1101 and has 3 set bits.



8.4 Rotate Bits of a Number

A rotation (or circular shift) is an operation similar to shift except that the bits that fall off at one end are put back to the other end. In left rotation, the bits that fall off at left end are put back at right end. In right rotation, the bits that fall off at right end are put back at left end.

Let n is stored using 8 bits. Left rotation of n = 11100101 by 3 makes n = 00101111 (Left shifted by 3 and first 3 bits are put back in last). If n is stored using 16 bits or 32 bits then left rotation of n (000…11100101) becomes 00…0011100101000.

Right rotation of n = 11100101 by 3 makes n = 10111100 (Right shifted by 3 and last 3 bits are put back in first) if n is stored using 8 bits. If n is stored using 16 bits or 32 bits then right rotation of n (000…11100101) by 3 becomes 101000…0011100.

Input: n = 16

Output:

Left rotation of 16 by 2 is 64

Right rotation of 16 by 2 is 4

8.5 Find the Integer Occurring Odd Number of Times

Given a list of integers where all integers occur even times, expect one which occur odd times. Find out that integer.

Input: lst = [1,2,2,1,1,4,4,4,5,4,5]

Output: 1

A simple brute force approach would say to count the number of times each integer is present in the list, and print the integer corresponding to odd count. In the worst case it will lead to a time complexity of O(n*k) where k is the number of distinct elements.

Let's try out this using bit-masking. What happens if we take XOR of all the elements?

8.6 Generating all the Subsets of an Integer Set

Given a set S of size N consisting of N items numbered from 1 to N. Print all the subsets of S. Set S can be represented as S= {1,2,3...,N}.

For a set of size equal to 2. All the possible subsets are {0}, {1}, {2}, {1,2}. Where {0} represents an empty set.

Constraints-N can range from any integer between 1 to 20 inclusive. i.e. 1<=N<=20

Input: n = 3

Output: {}, {1}, {2}, {1, 2}, {3}, {1, 3}, {2, 3}, {1, 2, 3}

8.7 Number of Steps to Reduce a Number to Zero

Given an integer num, return the number of steps to reduce it to zero.

In one step, if the current number is even, you have to divide it by 2, otherwise, you have to subtract 1 from it.

Input: num = 14

Output: 6

Explanation:

- 14 is even; divide by 2 and obtain 7.
- 7 is odd; subtract 1 and obtain 6.
- 6 is even; divide by 2 and obtain 3.
- 3 is odd; subtract 1 and obtain 2.

- 2 is even; divide by 2 and obtain 1.
- 1 is odd; subtract 1 and obtain 0.

Input: num = 8

Output: 4

Explanation:

- 8 is even; divide by 2 and obtain 4.
- 4 is even; divide by 2 and obtain 2.
- 2 is even; divide by 2 and obtain 1.
- 1 is odd; subtract 1 and obtain 0.

Input: num = 123

Output: 12

8.8 Minimum Bit Flips to Convert Number

A **bit flip** of a number x is choosing a bit in the binary representation of x and **flipping** it from either 0 to 1 or 1 to 0.

Example: for x = 7, the binary representation is 111 and we may choose any bit (including any leading zeros not shown) and flip it. We can flip the first bit from the right to get 110, flip the second bit from the right to get 101, flip the fifth bit from the right (a leading zero) to get 10111, etc.

Given two integers start and goal, return *the* ***minimum*** *number of* ***bit flips*** *to convert* start *to* goal.

Input: start = 10, goal = 7

Output: 3

Explanation:

The binary representation of 10 and 7 are 1010 and 0111 respectively. We can convert 10 to 7 in 3 steps:

- Flip the first bit from the right: 101<u>0</u> -> 101<u>1</u>.

- Flip the third bit from the right: 1<u>0</u>11 -> 1<u>1</u>11.

- Flip the fourth bit from the right: <u>1</u>111 -> <u>0</u>111.

It can be shown we cannot convert 10 to 7 in less than 3 steps. Hence, we return 3.

Input: start = 3, goal = 4

Output: 3

Explanation:

- The binary representation of 3 and 4 are 011 and 100 respectively. We can convert 3 to 4 in 3 steps:
- - Flip the first bit from the right: 01<u>1</u> -> 01<u>0</u>.
- - Flip the second bit from the right: 0<u>1</u>0 -> 0<u>0</u>0.
- - Flip the third bit from the right: <u>0</u>00 -> <u>1</u>00.

It can be shown we cannot convert 3 to 4 in less than 3 steps. Hence, we return 3.

9. Heaps

9.1 Kth Largest Element in an Array

Given an integer array nums and an integer k, return *the* k^{th} *largest element in the array*.
Note that it is the k^{th} largest element in the sorted order, not the k^{th} distinct element.
Can you solve it without sorting?

Input: nums = [3, 2, 1, 5, 6, 4], k = 2

Output: 5

Input: nums = [3, 2, 3, 1, 2, 4, 5, 5, 6], k = 4

Output: 4

9.2 Sort Characters by Frequency

Given a string s, sort it in **decreasing order** based on the **frequency** of the characters. The **frequency** of a character is the number of times it appears in the string.
Return *the sorted string*. If there are multiple answers, return *any of them*.

Input: s = "tree"

Output: "eert"

Explanation: 'e' appears twice while 'r' and 't' both appear once.

So 'e' must appear before both 'r' and 't'. Therefore "eetr" is also a valid answer.

Input: s = "cccaaa"

Output: "aaaccc"

Explanation: Both 'c' and 'a' appear three times, so both "cccaaa" and "aaaccc" are valid answers.

Note that "cacaca" is incorrect, as the same characters must be together.

Input: s = "Aabb"

Output: "bbAa"

Explanation: "bbaA" is also a valid answer, but "Aabb" is incorrect.

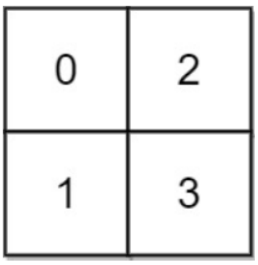
Note that 'A' and 'a' are treated as two different characters.

9.3 Swim in Rising Water

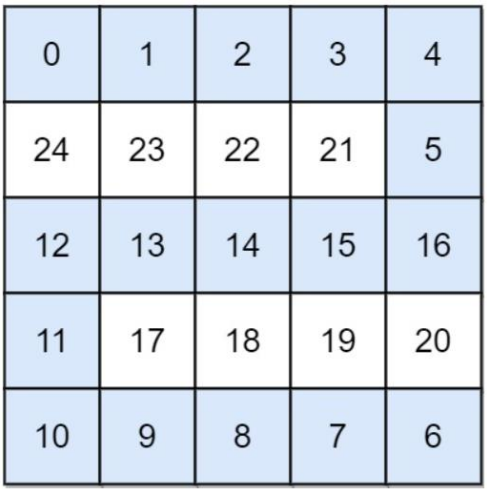
You are given an n x n integer matrix grid where each value grid[i][j] represents the elevation at that point (i, j).

The rain starts to fall. At time t, the depth of the water everywhere is t. You can swim from a square to another 4-directionally adjacent square if and only if the elevation of both squares individually are at most t. You can swim infinite distances in zero time. Of course, you must stay within the boundaries of the grid during your swim.

Return the least time until you can reach the bottom right square (n - 1, n - 1) if you start at the top left square (0, 0).



Input: grid = [[0, 2], [1,3]]
Output: 3
Explanation:
At time 0, you are in grid location (0, 0).
You cannot go anywhere else because 4-directionally adjacent neighbors have a higher elevation than t = 0.
You cannot reach point (1, 1) until time 3.
When the depth of water is 3, we can swim anywhere inside the grid.



Input: grid = [[0, 1, 2, 3, 4], [24, 23, 22, 21, 5], [12, 13, 14, 15, 16], [11, 17, 18, 19, 20], [10, 9, 8, 7, 6]]
Output: 16
Explanation: The final route is shown.
We need to wait until time 16 so that (0, 0) and (4, 4) are connected.

9.4 Car Pooling

There is a car with capacity empty seats. The vehicle only drives east (i.e., it cannot turn around and drive west).

You are given the integer capacity and an array trips where trips[i] = [numPassengers$_i$, from$_i$, to$_i$] indicates that the ith trip has numPassengers$_i$ passengers and the locations to pick them up and drop them off are from$_i$ and to$_i$ respectively. The locations are given as the number of kilometers due east from the car's initial location.

Return true *if it is possible to pick up and drop off all passengers for all the given trips, or* false *otherwise*.

Input: trips = [[2,1,5],[3,3,7]], capacity = 4

Output: false

Input: trips = [[2,1,5],[3,3,7]], capacity = 5

Output: true

9.5 Avoid Flood in the City

Your country has an infinite number of lakes. Initially, all the lakes are empty, but when it rains over the nth lake, the nth lake becomes full of water. If it rains over a lake that is **full of water**, there will be a **flood**. Your goal is to avoid floods in any lake.
Given an integer array rains where:

- rains[i] > 0 means there will be rains over the rains[i] lake.
- rains[i] == 0 means there are no rains this day and you can choose **one lake** this day and **dry it**.

Return *an array ans* where:

- ans.length == rains.length
- ans[i] == -1 if rains[i] > 0.
- ans[i] is the lake you choose to dry in the ith day if rains[i] == 0.

If there are multiple valid answers return **any** of them. If it is impossible to avoid flood return **an empty array**.
Notice that if you chose to dry a full lake, it becomes empty, but if you chose to dry an empty lake, nothing changes.

Input: rains = [1, 2, 3, 4]

Output: [-1,-1,-1,-1]

Explanation: After the first day full lakes are [1]

After the second day full lakes are [1, 2]

After the third day full lakes are [1, 2, 3]

After the fourth day full lakes are [1, 2, 3, 4]

There's no day to dry any lake and there is no flood in any lake.

Input: rains = [1, 2, 0, 0, 2, 1]

Output: [-1, -1, 2, 1, -1, -1]

Explanation: After the first day full lakes are [1]

After the second day full lakes are [1, 2]

After the third day, we dry lake 2. Full lakes are [1]

After the fourth day, we dry lake 1. There is no full lakes.

After the fifth day, full lakes are [2].

After the sixth day, full lakes are [1, 2].

It is easy that this scenario is flood-free. [-1, -1, 1, 2, -1, -1] is another acceptable scenario.

Input: rains = [1, 2, 0, 1, 2]

Output: []

Explanation: After the second day, full lakes are [1, 2]. We have to dry one lake in the third day.

After that, it will rain over lakes [1, 2]. It's easy to prove that no matter which lake you choose to dry in the 3rd day, the other one will flood.

10. HashMap

10.1 The Skyline Problem

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.

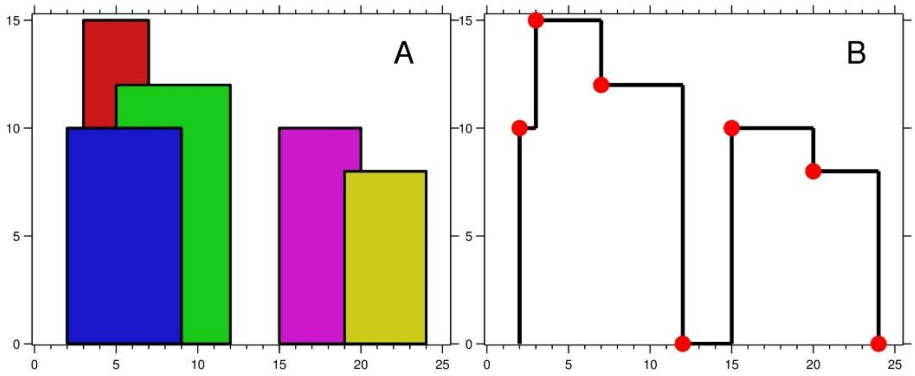
The geometric information of each building is given in the array buildings where buildings[i] = [$left_i$, $right_i$, $height_i$]:

- $left_i$ 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],...]



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.

Input: buildings = [[0, 2, 3], [2, 5, 3]]

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

10.2 Sliding Window Median

The **median** is the middle value in an ordered integer list. If the size of the list is even, there is no middle value. So the median is the mean of the two middle values.

- For examples, if arr = [2, 3, 4], the median is 3.
- For examples, if arr = [1, 2, 3, 4], the median is (2 + 3) / 2 = 2.5.

You are given an integer array nums and an integer k. There is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position.

Return the median array for each window in the original array. Answers within 10^{-5} of the actual value will be accepted.

Input: nums = [1, 3, -1, -3, 5, 3, 6, 7], k = 3

Output: [1.00000, -1.00000, -1.00000, 3.00000, 5.00000, 6.00000]

Explanation:

Window position	Median
[**1 3 -1**] -3 5 3 6 7	1
1 [**3 -1 -3**] 5 3 6 7	-1
1 3 [**-1 -3 5**] 3 6 7	-1
1 3 -1 [**-3 5 3**] 6 7	3
1 3 -1 -3 [**5 3 6**] 7	5
1 3 -1 -3 5 [**3 6 7**]	6

Input: nums = [1, 2, 3, 4, 2, 3, 1, 4, 2], k = 3

Output: [2.00000, 3.00000, 3.00000, 3.00000, 2.00000, 3.00000, 2.00000]

10.3 Smallest Range Covering Elements from K Lists

You have k lists of sorted integers in **non-decreasing order**. Find the **smallest** range that includes at least one number from each of the k lists.

We define the range [a, b] is smaller than range [c, d] if b - a < d - c **or** a < c if b - a == d - c.

Input: nums = [[4, 10, 15, 24, 26], [0, 9, 12, 20], [5, 18, 22, 30]]

Output: [20, 24]

Explanation:

List 1: [4, 10, 15, 24, 26], 24 is in range [20, 24].

List 2: [0, 9, 12, 20], 20 is in range [20, 24].

List 3: [5, 18, 22, 30], 22 is in range [20, 24].

Example 2:

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

Output: [1, 1]

10.4 Cut off Trees for Golf Event

You are asked to cut off all the trees in a forest for a golf event. The forest is represented as an m x n matrix. In this matrix:

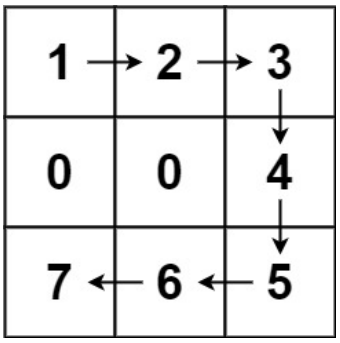
- 0 means the cell cannot be walked through.
- 1 represents an empty cell that can be walked through.
- A number greater than 1 represents a tree in a cell that can be walked through, and this number is the tree's height.

In one step, you can walk in any of the four directions: north, east, south, and west. If you are standing in a cell with a tree, you can choose whether to cut it off.

You must cut off the trees in order from shortest to tallest. When you cut off a tree, the value at its cell becomes 1 (an empty cell).

Starting from the point (0, 0), return *the minimum steps you need to walk to cut off all the trees*. If you cannot cut off all the trees, return -1.

Note: The input is generated such that no two trees have the same height, and there is at least one tree needs to be cut off.

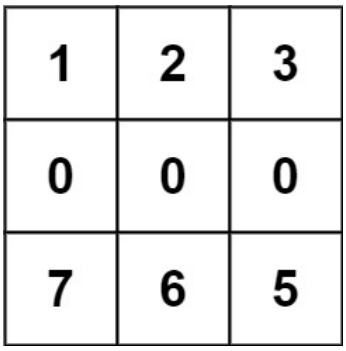


Input: forest = [[1, 2, 3], [0, 0, 4], [7, 6, 5]]

Output: 6

Explanation: Following the path above allows you to cut off the trees from shortest to tallest in 6 steps.

Example 2:



Input: forest = [[1, 2, 3], [0, 0, 0], [7, 6, 5]]

Output: -1

Explanation: The trees in the bottom row cannot be accessed as the middle row is blocked.

Input: forest = [[2, 3, 4], [0, 0, 5], [8, 7, 6]]

Output: 6

Explanation: You can follow the same path as Example 1 to cut off all the trees.

Note that you can cut off the first tree at (0, 0) before making any steps.

10.5 Find K Closest Elements

Given a **sorted** integer array arr, two integers k and x, return the k closest integers to x in the array. The result should also be sorted in ascending order.

An integer a is closer to x than an integer b if:

- |a - x| < |b - x|, or
- |a - x| == |b - x| and a < b

Input: arr = [1, 2, 3, 4, 5], k = 4, x = 3

Output: [1, 2, 3, 4]

Input: arr = [1, 2, 3, 4, 5], k = 4, x = -1

Output: [1, 2, 3, 4]

10.6 Minimum Cost to Hire K Workers

There are n workers. You are given two integer arrays quality and wage where quality[i] is the quality of the i^{th} worker and wage[i] is the minimum wage expectation for the i^{th} worker.

We want to hire exactly k workers to form a paid group. To hire a group of k workers, we must pay them according to the following rules:

1. Every worker in the paid group should be paid in the ratio of their quality compared to other workers in the paid group.
2. Every worker in the paid group must be paid at least their minimum wage expectation.

Given the integer k, return the least amount of money needed to form a paid group satisfying the above conditions. Answers within 10^{-5} of the actual answer will be accepted.

Input: quality = [10, 20, 5], wage = [70, 50, 30], k = 2

Output: 105.00000

Explanation: We pay 70 to 0^{th} worker and 35 to 2^{nd} worker.

Input: quality = [3, 1, 10, 10, 1], wage = [4, 8, 2, 2, 7], k = 3

Output: 30.66667

Explanation: We pay 4 to 0^{th} worker, 13.33333 to 2^{nd} and 3^{rd} workers separately.

10.7 Minimum Number of Refueling Stops

A car travels from a starting position to a destination which is target miles east of the starting position.

There are gas stations along the way. The gas stations are represented as an array stations where stations[i] = [$position_i$, $fuel_i$] indicates that the i^{th} gas station is $position_i$ miles east of the starting position and has $fuel_i$ liters of gas.

The car starts with an infinite tank of gas, which initially has startFuel liters of fuel in it. It uses one liter of gas per one mile that it drives. When the car reaches a gas station, it may stop and refuel, transferring all the gas from the station into the car.

Return the minimum number of refueling stops the car must make in order to reach its destination. If it cannot reach the destination, return -1.

Note that if the car reaches a gas station with 0 fuel left, the car can still refuel there. If the car reaches the destination with 0 fuel left, it is still considered to have arrived.

Input: target = 1, startFuel = 1, stations = []

Output: 0

Explanation: We can reach the target without refueling.

Input: target = 100, startFuel = 1, stations = [[10, 100]]

Output: -1

Explanation: We cannot reach the target (or even the first gas station).

Input: target = 100, startFuel = 10, stations = [[10, 60], [20, 30], [30, 30], [60, 40]]

Output: 2

Explanation: We start with 10 liters of fuel.

We drive to position 10, expending 10 liters of fuel. We refuel from 0 liters to 60 liters of gas.

Then, we drive from position 10 to position 60 (expending 50 liters of fuel),

and refuel from 10 liters to 50 liters of gas. We then drive to and reach the target.

We made 2 refueling stops along the way, so we return 2.

10.8 Distant Barcodes

In a warehouse, there is a row of barcodes, where the i^{th} barcode is barcodes[i].

Rearrange the barcodes so that no two adjacent barcodes are equal. You may return any answer, and it is guaranteed an answer exists.

Input: barcodes = [1, 1, 1, 2, 2, 2]

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

Input: barcodes = [1, 1, 1, 1, 2, 2, 3, 3]

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

10.9 Dinner Plate Stacks

You have an infinite number of stacks arranged in a row and numbered (left to right) from 0, each of the stacks has the same maximum capacity.

Implement the DinnerPlates class:

- DinnerPlates(int capacity) Initializes the object with the maximum capacity of the stacks capacity.
- void push(int val) Pushes the given integer val into the leftmost stack with a size less than capacity.
- int pop() Returns the value at the top of the rightmost non-empty stack and removes it from that stack, and returns -1 if all the stacks are empty.
- int popAtStack(int index) Returns the value at the top of the stack with the given index index and removes it from that stack or returns -1 if the stack with that given index is empty.

Input: ["DinnerPlates", "push", "push", "push", "push", "push", "popAtStack", "push", "push", "popAtStack", "popAtStack", "pop", "pop", "pop", "pop", "pop"]

[[2], [1], [2], [3], [4], [5], [0], [20], [21], [0], [2], [], [], [], [], []]

Output: [null, null, null, null, null, null, 2, null, null, 20, 21, 5, 4, 3, 1, -1]

Explanation:

DinnerPlates D = DinnerPlates(2); // Initialize with capacity = 2

D.push(1);

D.push(2);

D.push(3);

D.push(4);

D.push(5); // The stacks are now: 2 4

1 3 5

␣ ␣ ␣

D.popAtStack(0); // Returns 2. The stacks are now: 4

1 3 5

␣ ␣ ␣

D.push(20); // The stacks are now: 20 4

1 3 5

␣ ␣ ␣

D.push(21); // The stacks are now: 20 4 21
1 3 5

D.popAtStack(0); // Returns 20. The stacks are now: 4 21
1 3 5

D.popAtStack(2); // Returns 21. The stacks are now: 4
1 3 5

D.pop() // Returns 5. The stacks are now: 4
1 3

D.pop() // Returns 4. The stacks are now: 1 3


D.pop() // Returns 3. The stacks are now: 1


D.pop() // Returns 1. There are no stacks.

D.pop() // Returns -1. There are still no stacks.

10.10 Maximum Number of Eaten Apples

There is a special kind of apple tree that grows apples every day for n days. On the i^{th} day, the tree grows apples[i] apples that will rot after days[i] days, that is on day i + days[i] the apples will be rotten and cannot be eaten. On some days, the apple tree does not grow any apples, which are denoted by apples[i] == 0 and days[i] == 0.

You decided to eat **at most** one apple a day (to keep the doctors away). Note that you can keep eating after the first n days.

Given two integer arrays days and apples of length n, return *the maximum number of apples you can eat.*

Input: apples = [1, 2, 3, 5, 2], days = [3, 2, 1, 4, 2]

Output: 7

Explanation: You can eat 7 apples:

- On the first day, you eat an apple that grew on the first day.

- On the second day, you eat an apple that grew on the second day.

- On the third day, you eat an apple that grew on the second day. After this day, the apples that grew on the third day rot.

- On the fourth to the seventh days, you eat apples that grew on the fourth day.

Input: apples = [3, 0, 0, 0, 0, 2], days = [3, 0, 0, 0, 0, 2]

Output: 5

Explanation: You can eat 5 apples:

- On the first to the third day you eat apples that grew on the first day.

- Do nothing on the fourth and fifth days.

- On the sixth and seventh days you eat apples that grew on the sixth day.

11. Two Pointer Algorithm

11.1 Count Pairs whose Sum is less than Target

Given a **0-indexed** integer array nums of length n and an integer target, return the number of pairs (i, j) where 0 <= i < j < n and nums[i] + nums[j] < target.

Input: nums = [-1, 1, 2, 3, 1], target = 2

Output: 3

Explanation: There are 3 pairs of indices that satisfy the conditions in the statement:

- (0, 1) since 0 < 1 and nums[0] + nums[1] = 0 < target

- (0, 2) since 0 < 2 and nums[0] + nums[2] = 1 < target

- (0, 4) since 0 < 4 and nums[0] + nums[4] = 0 < target

Note that (0, 3) is not counted since nums[0] + nums[3] is not strictly less than the target.

Input: nums = [-6, 2, 5, -2, -7, -1, 3], target = -2

Output: 10

Explanation: There are 10 pairs of indices that satisfy the conditions in the statement:

- (0, 1) since 0 < 1 and nums[0] + nums[1] = -4 < target

- (0, 3) since 0 < 3 and nums[0] + nums[3] = -8 < target

- (0, 4) since 0 < 4 and nums[0] + nums[4] = -13 < target

- (0, 5) since 0 < 5 and nums[0] + nums[5] = -7 < target

- (0, 6) since 0 < 6 and nums[0] + nums[6] = -3 < target

- (1, 4) since 1 < 4 and nums[1] + nums[4] = -5 < target

- (3, 4) since 3 < 4 and nums[3] + nums[4] = -9 < target

- (3, 5) since 3 < 5 and nums[3] + nums[5] = -3 < target

- (4, 5) since 4 < 5 and nums[4] + nums[5] = -8 < target

- (4, 6) since 4 < 6 and nums[4] + nums[6] = -4 < target

11.2 Lexicographically Smallest Palindrome

You are given a string s consisting of **lowercase English letters**, and you are allowed to perform operations on it. In one operation, you can **replace** a character in s with another lowercase English letter.

Your task is to make s a **palindrome** with the **minimum number of operations** possible. If there are **multiple palindromes** that can be made using the **minimum** number of operations, make the **lexicographically smallest** one.

A string a is lexicographically smaller than a string b (of the same length) if in the first position where a and b differ, string a has a letter that appears earlier in the alphabet than the corresponding letter in b.

Return the resulting palindrome string.

Input: s = "egcfe"

Output: "efcfe"

Explanation: The minimum number of operations to make "egcfe" a palindrome is 1, and the lexicographically smallest palindrome string we can get by modifying one character is "efcfe", by changing 'g'.

Input: s = "abcd"

Output: "abba"

Explanation: The minimum number of operations to make "abcd" a palindrome is 2, and the lexicographically smallest palindrome string we can get by modifying two characters is "abba".

Input: s = "seven"

Output: "neven"

Explanation: The minimum number of operations to make "seven" a palindrome is 1, and the lexicographically smallest palindrome string we can get by modifying one character is "neven".

11.3 Merge Two 2D Arrays by Summing Values

You are given two **2D** integer arrays nums1 and nums2.

- nums1[i] = [id_i, val_i] indicate that the number with the id id_i has a value equal to val_i.
- nums2[i] = [id_i, val_i] indicate that the number with the id id_i has a value equal to val_i.

Each array contains **unique** ids and is sorted in **ascending** order by id.

Merge the two arrays into one array that is sorted in ascending order by id, respecting the following conditions:

- Only ids that appear in at least one of the two arrays should be included in the resulting array.
- Each id should be included **only once** and its value should be the sum of the values of this id in the two arrays. If the id does not exist in one of the two arrays then its value in that array is considered to be 0.

Return the resulting array. The returned array must be sorted in ascending order by id.

Input: nums1 = [[1,2],[2,3],[4,5]], nums2 = [[1,4],[3,2],[4,1]]

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

Explanation: The resulting array contains the following:

- id = 1, the value of this id is 2 + 4 = 6.

- id = 2, the value of this id is 3.

- id = 3, the value of this id is 2.

- id = 4, the value of this id is 5 + 1 = 6.

Input: nums1 = [[2, 4], [3, 6], [5, 5]], nums2 = [[1, 3], [4, 3]]

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

Explanation: There are no common ids, so we just include each id with its value in the resulting list.

11.4 Find the Array Concatenation Value

You are given a **0-indexed** integer array nums.

The **concatenation** of two numbers is the number formed by concatenating their numerals.

- For example, the concatenation of 15, 49 is 1549.

The **concatenation value** of nums is initially equal to 0. Perform this operation until nums becomes empty:

- If there exists more than one number in nums, pick the first element and last element in nums respectively and add the value of their concatenation to the **concatenation value** of nums, then delete the first and last element from nums.
- If one element exists, add its value to the **concatenation value** of nums, then delete it.

Return the concatenation value of the nums.

Input: nums = [7, 52, 2, 4]

Output: 596

Explanation: Before performing any operation, nums is [7, 52, 2, 4] and concatenation value is 0.

- In the first operation:

We pick the first element, 7, and the last element, 4.

Their concatenation is 74, and we add it to the concatenation value, so it becomes equal to 74.

Then we delete them from nums, so nums becomes equal to [52, 2].

- In the second operation:

We pick the first element, 52, and the last element, 2.

Their concatenation is 522, and we add it to the concatenation value, so it becomes equal to 596.

Then we delete them from the nums, so nums becomes empty.

Since the concatenation value is 596 so the answer is 596.

Input: nums = [5, 14, 13, 8, 12]

Output: 673

Explanation: Before performing any operation, nums is [5, 14, 13, 8, 12] and concatenation value is 0.

- In the first operation:

We pick the first element, 5, and the last element, 12.

Their concatenation is 512, and we add it to the concatenation value, so it becomes equal to 512.

Then we delete them from the nums, so nums becomes equal to [14, 13, 8].

- In the second operation:

We pick the first element, 14, and the last element, 8.

Their concatenation is 148, and we add it to the concatenation value, so it becomes equal to 660.

Then we delete them from the nums, so nums becomes equal to [13].

- In the third operation:

nums has only one element, so we pick 13 and add it to the concatenation value, so it becomes equal to 673.

Then we delete it from nums, so nums become empty.

Since the concatenation value is 673 so the answer is 673.

11.5 Largest Positive Integer that Exists with its Negative

Given an integer array nums that **does not contain** any zeros, find **the largest positive** integer k such that -k also exists in the array.

Return the positive integer k. If there is no such integer, return -1.

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

Output: 3

Explanation: 3 is the only valid k we can find in the array.

Input: nums = [-1, 10, 6, 7, -7, 1]

Output: 7

Explanation: Both 1 and 7 have their corresponding negative values in the array. 7 has a larger value.

Input: nums = [-10, 8, 6, 7, -2, -3]

Output: -1

Explanation: There is no a single valid k, we return -1.

11.6 Maximum Matching of Players with Trainers

You are given a **0-indexed** integer array players, where players[i] represents the **ability** of the i^{th} player. You are also given a **0-indexed** integer array trainers, where trainers[j] represents the **training capacity** of the j^{th} trainer.

The i^{th} player can **match** with the j^{th} trainer if the player's ability is **less than or equal to** the trainer's training capacity. Additionally, the i^{th} player can be matched with at most one trainer, and the j^{th} trainer can be matched with at most one player.

Return the **maximum** number of matchings between players and trainers that satisfy these conditions.

Input: players = [4, 7, 9], trainers = [8, 2, 5, 8]

Output: 2

Explanation:

One of the ways we can form two matchings is as follows:

- players[0] can be matched with trainers[0] since 4 <= 8.

- players[1] can be matched with trainers[3] since 7 <= 8.

It can be proven that 2 is the maximum number of matchings that can be formed.

Input: players = [1, 1, 1], trainers = [10]

Output: 1

Explanation:

The trainer can be matched with any of the 3 players.

Each player can only be matched with one trainer, so the maximum answer is 1.

11.7 Strictly Palindromic Number

An integer n is **strictly palindromic** if, for **every** base b between 2 and n - 2 (**inclusive**), the string representation of the integer n in base b is **palindromic**.

Given an integer n, return true if n is **strictly palindromic** and false otherwise.

A string is **palindromic** if it reads the same forward and backward.

Input: n = 9

Output: false

Explanation: In base 2: 9 = 1001 (base 2), which is palindromic.

In base 3: 9 = 100 (base 3), which is not palindromic.

Therefore, 9 is not strictly palindromic so we return false.

Note that in bases 4, 5, 6, and 7, n = 9 is also not palindromic.

Input: n = 4

Output: false

Explanation: We only consider base 2: 4 = 100 (base 2), which is not palindromic.

Therefore, we return false.

11.8 Number of Arithmetic Triplets

You are given a **0-indexed**, **strictly increasing** integer array nums and a positive integer diff. A triplet (i, j, k) is an **arithmetic triplet** if the following conditions are met:

- i < j < k,
- nums[j] - nums[i] == diff, and
- nums[k] - nums[j] == diff.

Return the number of unique **arithmetic triplets**.

Input: nums = [0, 1, 4, 6, 7, 10], diff = 3

Output: 2

Explanation:

(1, 2, 4) is an arithmetic triplet because both 7 - 4 == 3 and 4 - 1 == 3.

(2, 4, 5) is an arithmetic triplet because both 10 - 7 == 3 and 7 - 4 == 3.

Input: nums = [4, 5, 6, 7, 8, 9], diff = 2

Output: 2

Explanation:

(0, 2, 4) is an arithmetic triplet because both 8 - 6 == 2 and 6 - 4 == 2.

(1, 3, 5) is an arithmetic triplet because both 9 - 7 == 2 and 7 - 5 == 2.

11.9 Rearrange Array Elements by Sign

You are given a **0-indexed** integer array nums of **even** length consisting of an **equal** number of positive and negative integers.

You should **rearrange** the elements of nums such that the modified array follows the given conditions:

1. Every **consecutive pair** of integers have **opposite signs**.
2. For all integers with the same sign, the **order** in which they were present in nums is **preserved**.
3. The rearranged array begins with a positive integer.

Return the modified array after rearranging the elements to satisfy the aforementioned conditions.

Input: nums = [3, 1, -2, -5, 2, -4]

Output: [3, -2, 1, -5, 2, -4]

Explanation:

The positive integers in nums are [3, 1, 2]. The negative integers are [-2, -5, -4].

The only possible way to rearrange them such that they satisfy all conditions is [3, -2, 1, -5, 2, -4].

Other ways such as [1,-2,2,-5,3,-4], [3,1,2,-2,-5,-4], [-2,3,-5,1,-4,2] are incorrect because they do not satisfy one or more conditions.

Input: nums = [-1, 1]

Output: [1,-1]

Explanation:

1 is the only positive integer and -1 the only negative integer in nums.

So nums is rearranged to [1,-1].

11.10 Find First Palindromic String in the Array

Given an array of strings words, return the first **palindromic** string in the array. If there is no such string, return an **empty string** "".

A string is **palindromic** if it reads the same forward and backward.

Input: words = ["abc","car","ada","racecar","cool"]

Output: "ada"

Explanation: The first string that is palindromic is "ada".

Note that "racecar" is also palindromic, but it is not the first.

Input: words = ["notapalindrome","racecar"]

Output: "racecar"

Explanation: The first and only string that is palindromic is "racecar".

Input: words = ["def","ghi"]

Output: ""

Explanation: There are no palindromic strings, so the empty string is returned.

11.11 Minimum Number of Swaps to Make the String Balanced

You are given a **0-indexed** string s of **even** length n. The string consists of **exactly** n / 2 opening brackets '[' and n / 2 closing brackets ']'.

A string is called **balanced** if and only if:

- It is the empty string, or
- It can be written as AB, where both A and B are **balanced** strings, or
- It can be written as [C], where C is a **balanced** string.

You may swap the brackets at **any** two indices **any** number of times.

Return the **minimum** number of swaps to make s **balanced**.

Input: s = "][]["

Output: 1

Explanation: You can make the string balanced by swapping index 0 with index 3.

The resulting string is "[[]]".

Input: s = "]]][[["

Output: 2

Explanation: You can do the following to make the string balanced:

- Swap index 0 with index 4. s = "[]][][".

- Swap index 1 with index 5. s = "[[][]]".

The resulting string is "[[][]]".

Input: s = "[]"

Output: 0

Explanation: The string is already balanced.

11.12 Merge Strings Alternately

You are given two strings word1 and word2. Merge the strings by adding letters in alternating order, starting with word1. If a string is longer than the other, append the additional letters onto the end of the merged string. Return the merged string.

Input: word1 = "abc", word2 = "pqr"

Output: "apbqcr"

Explanation: The merged string will be merged as so:

word1: a b c

word2: p q r

merged: a p b q c r

Input: word1 = "ab", word2 = "pqrs"

Output: "apbqrs"

Explanation: Notice that as word2 is longer, "rs" is appended to the end.

word1: a b

word2: p q r s

merged: a p b q r s

Input: word1 = "abcd", word2 = "pq"

Output: "apbqcd"

Explanation: Notice that as word1 is longer, "cd" is appended to the end.

word1: a b c d

word2: p q

merged: a p b q c d

11.13 Find the Distance Value between Two Arrays

Given two integer arrays arr1 and arr2, and the integer d, *return the distance value between the two arrays*.

The distance value is defined as the number of elements arr1[i] such that there is not any element arr2[j] where |arr1[i]-arr2[j]| <= d.

Input: arr1 = [4, 5, 8], arr2 = [10,9,1,8], d = 2

Output: 2

Explanation:

For arr1[0]=4 we have:

|4-10|=6 > d=2

|4-9|=5 > d=2

|4-1|=3 > d=2

|4-8|=4 > d=2

For arr1[1]=5 we have:

|5-10|=5 > d=2

|5-9|=4 > d=2

|5-1|=4 > d=2

|5-8|=3 > d=2

For arr1[2]=8 we have:

|8-10|=2 <= d=2

|8-9|=1 <= d=2

|8-1|=7 > d=2

|8-8|=0 <= d=2

Input: arr1 = [1, 4, 2, 3], arr2 = [-4, -3, 6, 10, 20, 30], d = 3

Output: 2

Example 3:

Input: arr1 = [2, 1, 100, 3], arr2 = [-5, -2, 10, -3, 7], d = 6

Output: 1

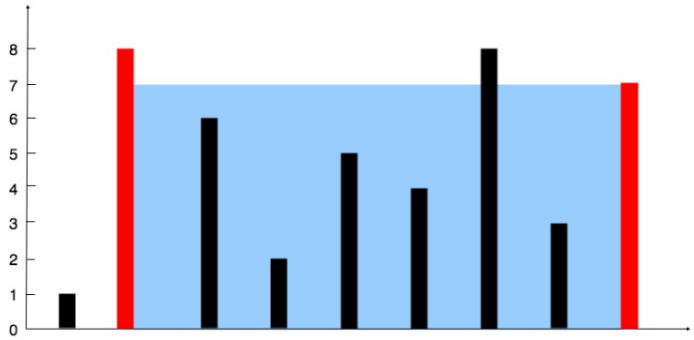
11.14 Container with Most Water

You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the i^{th} line are (i, 0) and (i, height[i]).

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return the maximum amount of water a container can store.

Notice that you may not slant the container.



Input: height = [1, 8, 6, 2, 5, 4, 8, 3, 7]

Output: 49

Explanation: The above vertical lines are represented by array [1, 8, 6, 2, 5, 4, 8, 3, 7]. In this case, the max area of water (blue section) the container can contain is 49.

Input: height = [1, 1]

Output: 1

11.15 Flipping an Image

Given an n x n binary matrix image, flip the image **horizontally**, then invert it, and return the resulting image.

To flip an image horizontally means that each row of the image is reversed.

- For example, flipping [1, 1, 0] horizontally results in [0, 1, 1].

To invert an image means that each 0 is replaced by 1, and each 1 is replaced by 0.

- For example, inverting [0, 1, 1] results in [1, 0, 0].

Input: image = [[1, 1, 0], [1, 0, 1], [0, 0, 0]]

Output: [[1, 0, 0], [0, 1, 0], [1, 1, 1]]

Explanation: First reverse each row: [[0, 1, 1], [1, 0, 1], [0, 0, 0]].

Then, invert the image: [[1, 0, 0], [0, 1, 0], [1, 1, 1]]

Input: image = [[1, 1, 0, 0], [1, 0, 0, 1], [0, 1, 1, 1], [1, 0, 1, 0]]

Output: [[1, 1, 0, 0], [0, 1, 1, 0], [0, 0, 0, 1], [1, 0, 1, 0]]

Explanation: First reverse each row: [[0, 0, 1, 1], [1, 0, 0, 1], [1, 1, 1, 0], [0, 1, 0, 1]].

Then invert the image: [[1, 1, 0, 0], [0, 1, 1, 0], [0, 0, 0, 1], [1, 0, 1, 0]]

11.16 Shortest Distance to a Character

Given a string s and a character c that occurs in s, return an array of integers answer where answer.length == s.length and answer[i] is the **distance** from index i to the **closest** occurrence of character c in s.

The **distance** between two indices i and j is abs(i - j), where abs is the absolute value function.

Input: s = "loveleetcode", c = "e"

Output: [3, 2, 1, 0, 1, 0, 0, 1, 2, 2, 1, 0]

Explanation: The character 'e' appears at indices 3, 5, 6, and 11 (0-indexed).

The closest occurrence of 'e' for index 0 is at index 3, so the distance is abs(0 - 3) = 3.

The closest occurrence of 'e' for index 1 is at index 3, so the distance is abs(1 - 3) = 2.

For index 4, there is a tie between the 'e' at index 3 and the 'e' at index 5, but the distance is still the same: abs(4 - 3) == abs(4 - 5) = 1.

The closest occurrence of 'e' for index 8 is at index 6, so the distance is abs(8 - 6) = 2.

Input: s = "aaab", c = "b"

Output: [3, 2, 1, 0]

11.17 Count Binary Substrings

Given a binary string s, return the number of non-empty substrings that have the same number of 0's and 1's, and all the 0's and all the 1's in these substrings are grouped consecutively.

Substrings that occur multiple times are counted the number of times they occur.

Input: s = "00110011"

Output: 6

Explanation: There are 6 substrings that have equal number of consecutive 1's and 0's: "0011", "01", "1100", "10", "0011", and "01".

Notice that some of these substrings repeat and are counted the number of times they occur.

Also, "00110011" is not a valid substring because all the 0's (and 1's) are not grouped together.

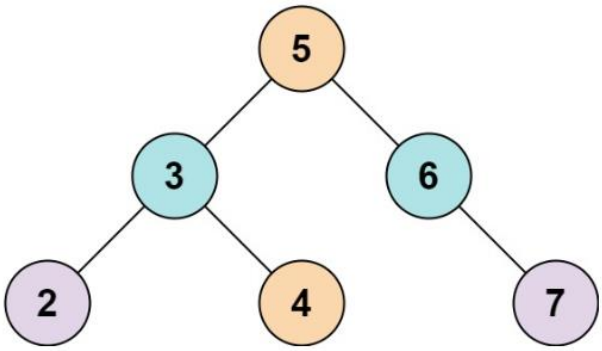
Input: s = "10101"

Output: 4

Explanation: There are 4 substrings: "10", "01", "10", "01" that have equal number of consecutive 1's and 0's.

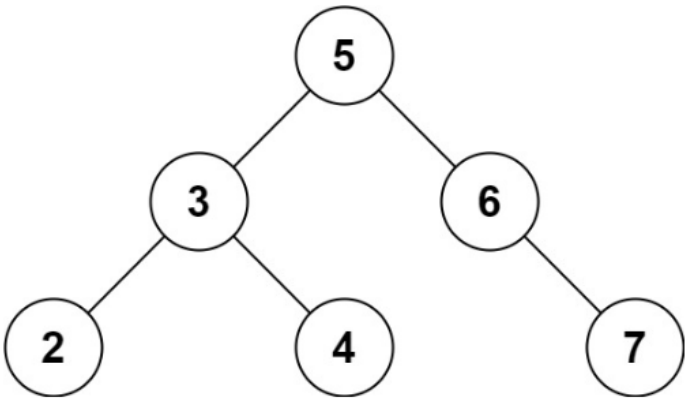
11.18 Two Sum IV - Input is a BST

Given the root of a binary search tree and an integer k, return true if there exist two elements in the BST such that their sum is equal to k, or false otherwise.



Input: root = [5, 3, 6, 2, 4, null, 7], k = 9

Output: true



Input: root = [5, 3, 6, 2, 4, null, 7], k = 28

Output: false

11.19 Find the Duplicate Number

Given an array of integers nums containing n + 1 integers where each integer is in the range [1, n] inclusive.

There is only **one repeated number** in nums, return this repeated number.

You must solve the problem **without** modifying the array nums and uses only constant extra space.

Input: nums = [1, 3, 4, 2, 2]

Output: 2

Input: nums = [3, 1, 3, 4, 2]

Output: 3

11.20 Happy Number

Write an algorithm to determine if a number n is happy.

A **happy number** is a number defined by the following process:

- Starting with any positive integer, replace the number by the sum of the squares of its digits.
- Repeat the process until the number equals 1 (where it will stay), or it **loops endlessly in a cycle** which does not include 1.
- Those numbers for which this process **ends in 1** are happy.

Return true if n is a happy number, and false if not.

Input: n = 19

Output: true

Explanation:

$1^2 + 9^2 = 82$

$8^2 + 2^2 = 68$

$6^2 + 8^2 = 100$

$1^2 + 0^2 + 0^2 = 1$

Input: n = 2

Output: false

12. Range Queries

12.1 Range Sum Query - Mutable

Given an integer array nums, handle multiple queries of the following types:

1. **Update** the value of an element in nums.
2. Calculate the **sum** of the elements of nums between indices left and right **inclusive** where left <= right.

Implement the NumArray class:

- NumArray(int[] nums) Initializes the object with the integer array nums.
- void update(int index, int val) **Updates** the value of nums[index] to be val.
- int sumRange(int left, int right) Returns the **sum** of the elements of nums between indices left and right **inclusive** (i.e. nums[left] + nums[left + 1] + ... + nums[right]).

Input: ["NumArray", "sumRange", "update", "sumRange"]

[[[1, 3, 5]], [0, 2], [1, 2], [0, 2]]

Output: [null, 9, null, 8]

Explanation

NumArray numArray = new NumArray([1, 3, 5]);

numArray.sumRange(0, 2); // return 1 + 3 + 5 = 9

numArray.update(1, 2); // nums = [1, 2, 5]

numArray.sumRange(0, 2); // return 1 + 2 + 5 = 8

12.2 Range Sum Query - Immutable

Given an integer array nums, handle multiple queries of the following type:

1. Calculate the **sum** of the elements of nums between indices left and right **inclusive** where left <= right.

Implement the NumArray class:

- NumArray(int[] nums) Initializes the object with the integer array nums.
- int sumRange(int left, int right) Returns the **sum** of the elements of nums between indices left and right **inclusive** (i.e. nums[left] + nums[left + 1] + ... + nums[right]).

Input: ["NumArray", "sumRange", "sumRange", "sumRange"]

[[[-2, 0, 3, -5, 2, -1]], [0, 2], [2, 5], [0, 5]]

Output: [null, 1, -1, -3]

Explanation:

NumArray numArray = new NumArray([-2, 0, 3, -5, 2, -1]);

numArray.sumRange(0, 2); // return (-2) + 0 + 3 = 1

numArray.sumRange(2, 5); // return 3 + (-5) + 2 + (-1) = -1

numArray.sumRange(0, 5); // return (-2) + 0 + 3 + (-5) + 2 + (-1) = -3

12.3 Range Frequency Queries

Design a data structure to find the **frequency** of a given value in a given subarray.

The **frequency** of a value in a subarray is the number of occurrences of that value in the subarray.

Implement the RangeFreqQuery class:

- RangeFreqQuery(int[] arr) Constructs an instance of the class with the given **0-indexed** integer array arr.
- int query(int left, int right, int value) Returns the **frequency** of value in the subarray arr[left...right].

A **subarray** is a contiguous sequence of elements within an array. arr[left...right] denotes the subarray that contains the elements of nums between indices left and right (**inclusive**).

Input: ["RangeFreqQuery", "query", "query"]

[[[12, 33, 4, 56, 22, 2, 34, 33, 22, 12, 34, 56]], [1, 2, 4], [0, 11, 33]]

Output: [null, 1, 2]

Explanation:

RangeFreqQuery rangeFreqQuery = new RangeFreqQuery([12, 33, 4, 56, 22, 2, 34, 33, 22, 12, 34, 56]);

rangeFreqQuery.query(1, 2, 4); // return 1. The value 4 occurs 1 time in the subarray [33, 4]

rangeFreqQuery.query(0, 11, 33); // return 2. The value 33 occurs 2 times in the whole array.

12.4 Range Product Queries of Powers

Given a positive integer n, there exists a **0-indexed** array called powers, composed of the **minimum** number of powers of 2 that sum to n. The array is sorted in **non-decreasing** order, and there is **only one** way to form the array.

You are also given a **0-indexed** 2D integer array queries, where queries[i] = [$left_i$, $right_i$]. Each queries[i] represents a query where you have to find the product of all powers[j] with $left_i$ <= j <= $right_i$.

Return an array answers, equal in length to queries, where answers[i] is the answer to the i^{th} query. Since the answer to the i^{th} query may be too large, each answers[i] should be returned **modulo** $10^9 + 7$.

Input: n = 15, queries = [[0, 1], [2, 2], [0, 3]]

Output: [2, 4, 64]

Explanation:

For n = 15, powers = [1, 2, 4, 8]. It can be shown that powers cannot be a smaller size.

Answer to 1st query: powers[0] * powers[1] = 1 * 2 = 2.

Answer to 2nd query: powers[2] = 4.

Answer to 3rd query: powers[0] * powers[1] * powers[2] * powers[3] = 1 * 2 * 4 * 8 = 64.

Each answer modulo $10^9 + 7$ yields the same answer, so [2, 4, 64] is returned.

Input: n = 2, queries = [[0, 0]]

Output: [2]

Explanation:

For n = 2, powers = [2].

The answer to the only query is powers[0] = 2. The answer modulo $10^9 + 7$ is the same, so [2] is returned.

12.5 Count of Range Sum

Given an integer array nums and two integers lower and upper, return the number of range sums that lie in [lower, upper] inclusive.

Range sum S(i, j) is defined as the sum of the elements in nums between indices i and j inclusive, where i <= j.

Input: nums = [-2, 5, -1], lower = -2, upper = 2

Output: 3

Explanation: The three ranges are: [0, 0], [2, 2], and [0, 2] and their respective sums are: -2, -1, 2.

Input: nums = [0], lower = 0, upper = 0

Output: 1

13. Trie

13.1 Implement Trie (Prefix Tree)

A **trie** (pronounced as "try") or **prefix tree** is a tree data structure used to efficiently store and retrieve keys in a dataset of strings. There are various applications of this data structure, such as autocomplete and spellchecker.

Implement the Trie class:

- Trie() Initializes the trie object.
- void insert(String word) Inserts the string word into the trie.
- boolean search(String word) Returns true if the string word is in the trie (i.e., was inserted before), and false otherwise.
- boolean startsWith(String prefix) Returns true if there is a previously inserted string word that has the prefix prefix, and false otherwise.

Input: ["Trie", "insert", "search", "search", "startsWith", "insert", "search"]

[[], ["apple"], ["apple"], ["app"], ["app"], ["app"], ["app"]]

Output: [null, null, true, false, true, null, true]

Explanation:

Trie trie = new Trie();

trie.insert("apple");

trie.search("apple"); // return True

trie.search("app"); // return False

trie.startsWith("app"); // return True

trie.insert("app");

trie.search("app"); // return True

13.2 Design Add and Search Words Data Structure

Design a data structure that supports adding new words and finding if a string matches any previously added string.

Implement the WordDictionary class:

- WordDictionary() Initializes the object.
- void addWord(word) Adds word to the data structure, it can be matched later.
- bool search(word) Returns true if there is any string in the data structure that matches word or false otherwise. word may contain dots '.' where dots can be matched with any letter.

Input:

["WordDictionary","addWord","addWord","addWord","search","search","search","search"]

[[], ["bad"], ["dad"], ["mad"], ["pad"], ["bad"], [".ad"], ["b.."]]

Output:

[null, null, null, null, false, true, true, true]

Explanation:

WordDictionary wordDictionary = new WordDictionary();

wordDictionary.addWord("bad");

wordDictionary.addWord("dad");

wordDictionary.addWord("mad");

wordDictionary.search("pad"); // return False

wordDictionary.search("bad"); // return True

wordDictionary.search(".ad"); // return True

wordDictionary.search("b.."); // return True

13.3 Lexicographical Numbers

Given an integer n, return all the numbers in the range [1, n] sorted in lexicographical order.

You must write an algorithm that runs in O(n) time and uses O(1) extra space.

Input: n = 13

Output: [1,10,11,12,13,2,3,4,5,6,7,8,9]

Input: n = 2

Output: [1,2]

13.4 Replace Words

In English, we have a concept called **root**, which can be followed by some other word to form another longer word - let's call this word **successor**. For example, when the **root** "an" is followed by the **successor** word "other", we can form a new word "another".

Given a dictionary consisting of many **roots** and a sentence consisting of words separated by spaces, replace all the **successors** in the sentence with the **root** forming it. If a **successor** can be replaced by more than one **root**, replace it with the **root** that has **the shortest length**.

Return the sentence after the replacement.

Input: dictionary = ["cat", "bat", "rat"], sentence = "the cattle was rattled by the battery"

Output: "the cat was rat by the bat"

Input: dictionary = ["a", "b", "c"], sentence = "aadsfasf absbs bbab cadsfafs"

Output: "a a b c"

13.5 Implement Magic Dictionary

Design a data structure that is initialized with a list of **different** words. Provided a string, you should determine if you can change exactly one character in this string to match any word in the data structure.

Implement the MagicDictionary class:

- MagicDictionary() Initializes the object.
- void buildDict(String[] dictionary) Sets the data structure with an array of distinct strings dictionary.
- bool search(String searchWord) Returns true if you can change **exactly one character** in searchWord to match any string in the data structure, otherwise returns false.

Input: ["MagicDictionary", "buildDict", "search", "search", "search", "search"]

[[], [["hello", "leetcode"]], ["hello"], ["hhllo"], ["hell"], ["leetcoded"]]

Output: [null, null, false, true, false, false]

Explanation:

MagicDictionary magicDictionary = new MagicDictionary();

magicDictionary.buildDict(["hello", "leetcode"]);

magicDictionary.search("hello"); // return False

magicDictionary.search("hhllo"); // We can change the second 'h' to 'e' to match "hello" so we return True

magicDictionary.search("hell"); // return False

magicDictionary.search("leetcoded"); // return False

13.6 Map Sum Pairs

Design a map that allows you to do the following:

- Maps a string key to a given value.
- Returns the sum of the values that have a key with a prefix equal to a given string.

Implement the MapSum class:

- MapSum() Initializes the MapSum object.
- void insert(String key, int val) Inserts the key-val pair into the map. If the key already existed, the original key-value pair will be overridden to the new one.
- int sum(string prefix) Returns the sum of all the pairs' value whose key starts with the prefix.

Input: ["MapSum", "insert", "sum", "insert", "sum"]

[[], ["apple", 3], ["ap"], ["app", 2], ["ap"]]

Output: [null, null, 3, null, 5]

Explanation:

MapSum mapSum = new MapSum();

mapSum.insert("apple", 3);

mapSum.sum("ap"); // return 3 (apple = 3)

mapSum.insert("app", 2);

mapSum.sum("ap"); // return 5 (apple + app = 3 + 2 = 5)

13.7 Longest Word in Dictionary

Given an array of strings words representing an English Dictionary, return the longest word in words that can be built one character at a time by other words in words.

If there is more than one possible answer, return the longest word with the smallest lexicographical order. If there is no answer, return the empty string.

Note that the word should be built from left to right with each additional character being added to the end of a previous word.

Input: words = ["w","wo","wor","worl","world"]

Output: "world"

Explanation: The word "world" can be built one character at a time by "w", "wo", "wor", and "worl".

Input: words = ["a","banana","app","appl","ap","apply","apple"]

Output: "apple"

Explanation: Both "apply" and "apple" can be built from other words in the dictionary. However, "apple" is lexicographically smaller than "apply".

13.8 Prefix and Suffix Search

Design a special dictionary that searches the words in it by a prefix and a suffix.

Implement the WordFilter class:

- WordFilter(string[] words) Initializes the object with the words in the dictionary.

- f(string pref, string suff) Returns *the index of the word in the dictionary,* which has the prefix pref and the suffix suff. If there is more than one valid index, return **the largest** of them. If there is no such word in the dictionary, return -1.

Input: ["WordFilter", "f"]

[[["apple"]], ["a", "e"]]

Output: [null, 0]

Explanation:

WordFilter wordFilter = new WordFilter(["apple"]);

wordFilter.f("a", "e"); // return 0, because the word at index 0 has prefix = "a" and suffix = "e".

13.9 Distinct Echo Substrings

Return the number of **distinct** non-empty substrings of text that can be written as the concatenation of some string with itself (i.e. it can be written as a + a where a is some string).

Input: text = "abcabcabc"

Output: 3

Explanation: The 3 substrings are "abcabc", "bcabca" and "cabcab".

Input: text = "leetcodeleetcode"

Output: 2

Explanation: The 2 substrings are "ee" and "leetcodeleetcode".

13.10 Remove Sub-Folders from the Filesystem

Given a list of folders folder, return *the folders after removing all **sub-folders** in those folders*. You may return the answer in **any order**.

If a folder[i] is located within another folder[j], it is called a **sub-folder** of it.

The format of a path is one or more concatenated strings of the form: '/' followed by one or more lowercase English letters.

- For example, "/leetcode" and "/leetcode/problems" are valid paths while an empty string and "/" are not.

Input: folder = ["/a","/a/b","/c/d","/c/d/e","/c/f"]

Output: ["/a","/c/d","/c/f"]

Explanation: Folders "/a/b" is a subfolder of "/a" and "/c/d/e" is inside of folder "/c/d" in our filesystem.

Input: folder = ["/a","/a/b/c","/a/b/d"]

Output: ["/a"]

Explanation: Folders "/a/b/c" and "/a/b/d" will be removed because they are subfolders of "/a".

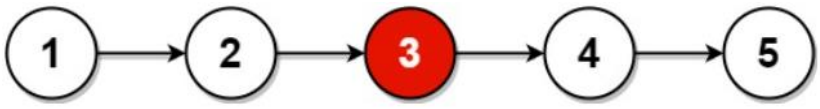
Input: folder = ["/a/b/c","/a/b/ca","/a/b/d"]

Output: ["/a/b/c","/a/b/ca","/a/b/d"]

14. Fast and Slow Pointers

14.1 Middle of the Linked List

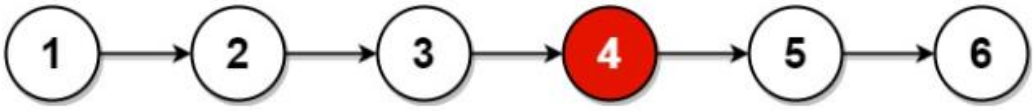
Given the head of a singly linked list, return the middle node of the linked list. If there are two middle nodes, return the second middle node.



Input: head = [1, 2, 3, 4, 5]

Output: [3, 4, 5]

Explanation: The middle node of the list is node 3.



Input: head = [1, 2, 3, 4, 5, 6]

Output: [4, 5, 6]

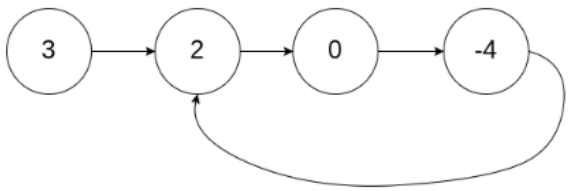
Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.

14.2 Linked List Cycle

Given head, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to. **Note that pos is not passed as a parameter**.

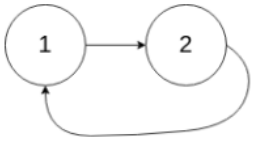
Return true if there is a cycle in the linked list. Otherwise, return false.



Input: head = [3, 2, 0, -4], pos = 1

Output: true

Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).



Input: head = [1, 2], pos = 0

Output: true

Explanation: There is a cycle in the linked list, where the tail connects to the 0th node.

Input: head = [1], pos = -1

Output: false

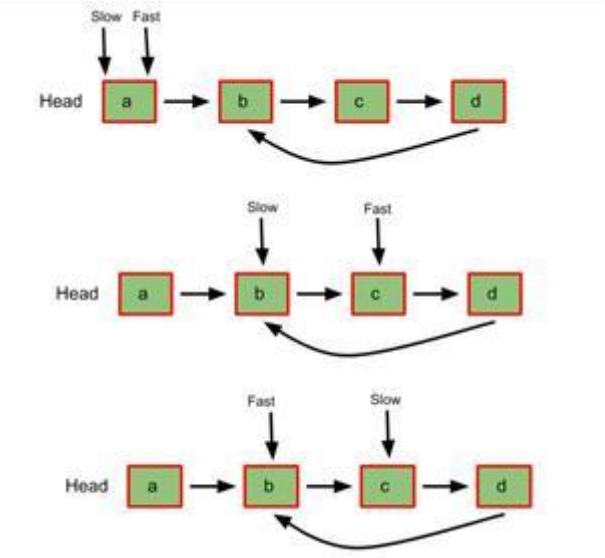
Explanation: There is no cycle in the linked list.

14.3 Floyd's Cycle Finding Algorithm

Floyd's cycle finding algorithm or Hare-Tortoise algorithm is a pointer algorithm that uses only two pointers, moving through the sequence at different speeds. This algorithm is used to find a loop in a linked list. It uses two pointers one moving twice as fast as the other one. The faster one is called the fast pointer and the other one is called the slow pointer.

While traversing the linked list one of these things will occur-

- The Fast pointer may reach the end (NULL) this shows that there is no loop in the linked list.
- The Fast pointer again catches the slow pointer at some time therefore a loop exists in the linked list.

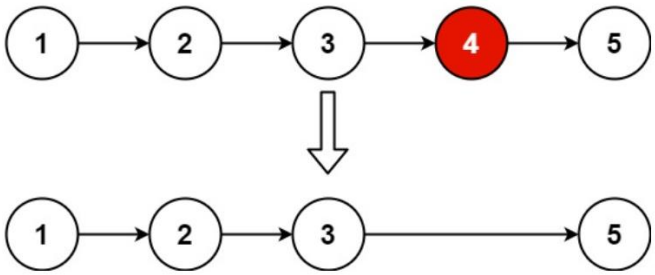


- Initialize two-pointers and start traversing the linked list.
- Move the slow pointer by one position.
- Move the fast pointer by two positions.

- If both pointers meet at some point then a loop exists and if the fast pointer meets the end position then no loop exists.

14.4 Remove Nth Node from End of List

Given the head of a linked list, remove the n^{th} node from the end of the list and return its head.



Input: head = [1, 2, 3, 4, 5], n = 2

Output: [1, 2, 3, 5]

Input: head = [1], n = 1

Output: []

Input: head = [1, 2], n = 1

Output: [1]

15. Sliding Window

15.1 Find the K-Beauty of a Number

The **k-beauty** of an integer num is defined as the number of **substrings** of num when it is read as a string that meet the following conditions:

- It has a length of k.
- It is a divisor of num.

Given integers num and k, return *the k-beauty of* num.

Note:

- **Leading zeros** are allowed.
- 0 is not a divisor of any value.

A **substring** is a contiguous sequence of characters in a string.

Input: num = 240, k = 2

Output: 2

Explanation: The following are the substrings of num of length k:

- "24" from "**24**0": 24 is a divisor of 240.

- "40" from "2**40**": 40 is a divisor of 240.

Therefore, the k-beauty is 2.

Input: num = 430043, k = 2

Output: 2

Explanation: The following are the substrings of num of length k:

- "43" from "**43**0043": 43 is a divisor of 430043.

- "30" from "4**30**043": 30 is not a divisor of 430043.

- "00" from "43**00**43": 0 is not a divisor of 430043.

- "04" from "430**04**3": 4 is not a divisor of 430043.

- "43" from "4300**43**": 43 is a divisor of 430043.

Therefore, the k-beauty is 2.

15.2 Maximum Erasure Value

You are given an array of positive integers nums and want to erase a subarray containing **unique elements**. The **score** you get by erasing the subarray is equal to the **sum** of its elements.

Return the **maximum score** you can get by erasing **exactly one** subarray.

An array b is called to be a subarray of a if it forms a contiguous subsequence of a, that is, if it is equal to a[l],a[l+1],...,a[r] for some (l,r).

Input: nums = [4, 2, 4, 5, 6]

Output: 17

Explanation: The optimal subarray here is [2, 4, 5, 6].

Example 2:

Input: nums = [5, 2, 1, 2, 5, 2, 1, 2, 5]

Output: 8

Explanation: The optimal subarray here is [5, 2, 1] or [1,2,5].

15.3 Longest Nice Substring

A string s is **nice** if, for every letter of the alphabet that s contains, it appears **both** in uppercase and lowercase. For example, "abABB" is nice because 'A' and 'a' appear, and 'B' and 'b' appear. However, "abA" is not because 'b' appears, but 'B' does not.

Given a string s, return the longest **substring** of s that is **nice**. If there are multiple, return the substring of the **earliest** occurrence. If there are none, return an empty string.

Input: s = "YazaAay"

Output: "aAa"

Explanation: "aAa" is a nice string because 'A/a' is the only letter of the alphabet in s, and both 'A' and 'a' appear.

"aAa" is the longest nice substring.

Input: s = "Bb"

Output: "Bb"

Explanation: "Bb" is a nice string because both 'B' and 'b' appear. The whole string is a substring.

Input: s = "c"

Output: ""

Explanation: There are no nice substrings.

15.4 Substrings of Size Three with Distinct Characters

A string is **good** if there are no repeated characters.

Given a string s, return the number of **good substrings** of length **three** in s.

Note that if there are multiple occurrences of the same substring, every occurrence should be counted.

A **substring** is a contiguous sequence of characters in a string.

Input: s = "xyzzaz"

Output: 1

Explanation: There are 4 substrings of size 3: "xyz", "yzz", "zza", and "zaz".

The only good substring of length 3 is "xyz".

Input: s = "aababcabc"

Output: 4

Explanation: There are 7 substrings of size 3: "aab", "aba", "bab", "abc", "bca", "cab", and "abc".

The good substrings are "abc", "bca", "cab", and "abc".

15.5 Minimum Difference between Highest and Lowest of K Scores

You are given a **0-indexed** integer array nums, where nums[i] represents the score of the i^{th} student. You are also given an integer k.

Pick the scores of any k students from the array so that the **difference** between the **highest** and the **lowest** of the k scores is **minimized**.

Return the **minimum** possible difference.

Input: nums = [90], k = 1

Output: 0

Explanation: There is one way to pick score(s) of one student:

- [**90**]. The difference between the highest and lowest score is 90 - 90 = 0.

The minimum possible difference is 0.

Input: nums = [9, 4, 1, 7], k = 2

Output: 2

Explanation: There are six ways to pick score(s) of two students:

- [**9**, **4**, 1, 7]. The difference between the highest and lowest score is 9 - 4 = 5.
- [**9**, 4, **1**, 7]. The difference between the highest and lowest score is 9 - 1 = 8.
- [**9**, 4, 1, **7**]. The difference between the highest and lowest score is 9 - 7 = 2.
- [9, **4**, **1**, 7]. The difference between the highest and lowest score is 4 - 1 = 3.
- [9, **4**, 1, **7**]. The difference between the highest and lowest score is 7 - 4 = 3.
- [9, 4, **1**, **7**]. The difference between the highest and lowest score is 7 - 1 = 6.

The minimum possible difference is 2.

15.6 Minimum Recolors to Get K Consecutive Black Blocks

You are given a **0-indexed** string blocks of length n, where blocks[i] is either 'W' or 'B', representing the color of the i^{th} block. The characters 'W' and 'B' denote the colors white and black, respectively.

You are also given an integer k, which is the desired number of **consecutive** black blocks.

In one operation, you can **recolor** a white block such that it becomes a black block.

Return the **minimum** number of operations needed such that there is at least **one** occurrence of k consecutive black blocks.

Input: blocks = "WBBWWBBWBW", k = 7

Output: 3

Explanation:

One way to achieve 7 consecutive black blocks is to recolor the 0th, 3rd, and 4th blocks

so that blocks = "BBBBBBBWBW".

It can be shown that there is no way to achieve 7 consecutive black blocks in less than 3 operations.

Therefore, we return 3.

Input: blocks = "WBWBBBW", k = 2

Output: 0

Explanation:

No changes need to be made, since 2 consecutive black blocks already exist.

Therefore, we return 0.

15.7 Minimum Consecutive Cards to Pick Up

You are given an integer array cards where cards[i] represents the **value** of the i^{th} card. A pair of cards are **matching** if the cards have the **same** value.

Return the **minimum** number of **consecutive** cards you have to pick up to have a pair of **matching** cards among the picked cards. If it is impossible to have matching cards, return -1.

Input: cards = [3, 4, 2, 3, 4, 7]

Output: 4

Explanation: We can pick up the cards [3, 4, 2, 3] which contain a matching pair of cards with value 3. Note that picking up the cards [4, 2, 3, 4] is also optimal.

Input: cards = [1, 0, 5, 3]

Output: -1

Explanation: There is no way to pick up a set of consecutive cards that contain a pair of matching cards.

15.8 Fruit into Baskets

You are visiting a farm that has a single row of fruit trees arranged from left to right. The trees are represented by an integer array fruits where fruits[i] is the **type** of fruit the i^{th} tree produces.

You want to collect as much fruit as possible. However, the owner has some strict rules that you must follow:

- You only have **two** baskets, and each basket can only hold a **single type** of fruit. There is no limit on the amount of fruit each basket can hold.
- Starting from any tree of your choice, you must pick **exactly one fruit** from **every** tree (including the start tree) while moving to the right. The picked fruits must fit in one of your baskets.
- Once you reach a tree with fruit that cannot fit in your baskets, you must stop.

Given the integer array fruits, return the **maximum** number of fruits you can pick.

Input: fruits = [1, 2, 1]

Output: 3

Explanation: We can pick from all 3 trees.

Input: fruits = [0, 1, 2, 2]

Output: 3

Explanation: We can pick from trees [1, 2, 2].

If we had started at the first tree, we would only pick from trees [0, 1].

Input: fruits = [1, 2, 3, 2, 2]

Output: 4

Explanation: We can pick from trees [2, 3, 2, 2].

If we had started at the first tree, we would only pick from trees [1, 2].

15.9 Find All Anagrams in a String

Given two strings s and p, return an array of all the start indices of p's anagrams in s. You may return the answer in **any order**.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Input: s = "cbaebabacd", p = "abc"

Output: [0, 6]

Explanation:

The substring with start index = 0 is "cba", which is an anagram of "abc".

The substring with start index = 6 is "bac", which is an anagram of "abc".

Input: s = "abab", p = "ab"

Output: [0, 1, 2]

Explanation:

The substring with start index = 0 is "ab", which is an anagram of "ab".

The substring with start index = 1 is "ba", which is an anagram of "ab".

The substring with start index = 2 is "ab", which is an anagram of "ab".

15.10 Maximum Average Subarray I

You are given an integer array nums consisting of n elements, and an integer k.

Find a contiguous subarray whose **length is equal to** k that has the maximum average value and return this value. Any answer with a calculation error less than 10^{-5} will be accepted.

Input: nums = [1, 12, -5, -6, 50, 3], k = 4

Output: 12.75000

Explanation: Maximum average is (12 - 5 - 6 + 50) / 4 = 51 / 4 = 12.75

Input: nums = [5], k = 1

Output: 5.00000

15.11 Longest Substring with At Least K Repeating Characters

Given a string s and an integer k, return the length of the longest substring of s such that the frequency of each character in this substring is greater than or equal to k.

If no such substring exists, return 0.

Input: s = "aaabb", k = 3

Output: 3

Explanation: The longest substring is "aaa", as 'a' is repeated 3 times.

Input: s = "ababbc", k = 2

Output: 5

Explanation: The longest substring is "ababb", as 'a' is repeated 2 times and 'b' is repeated 3 times.

15.12 Subarray Product Less Than K

Given an array of integer's nums and an integer k, return the number of contiguous subarrays where the product of all the elements in the subarray is strictly less than k.

Input: nums = [10, 5, 2, 6], k = 100

Output: 8

Explanation: The 8 subarrays that have product less than 100 are:

[10], [5], [2], [6], [10, 5], [5, 2], [2, 6], [5, 2, 6]

Note that [10, 5, 2] is not included as the product of 100 is not strictly less than k.

Input: nums = [1, 2, 3], k = 0

Output: 0

15.13 Repeated DNA Sequences

The **DNA sequence** is composed of a series of nucleotides abbreviated as 'A', 'C', 'G', and 'T'.

- For example, "ACGAATTCCG" is a **DNA sequence**.

When studying **DNA**, it is useful to identify repeated sequences within the DNA.

Given a string s that represents a **DNA sequence**, return all the **10-letter-long** sequences (substrings) that occur more than once in a DNA molecule. You may return the answer in **any order**.

Input: s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT"

Output: ["AAAAACCCCC","CCCCCAAAAA"]

Input: s = "AAAAAAAAAAAAA"

Output: ["AAAAAAAAAA"]

15.14 Swap for Longest Repeated Character Substring

You are given a string text. You can swap two of the characters in the text.

Return the length of the longest substring with repeated characters.

Input: text = "ababa"

Output: 3

Explanation: We can swap the first 'b' with the last 'a', or the last 'b' with the first 'a'. Then, the longest repeated character substring is "aaa" with length 3.

Input: text = "aaabaaa"

Output: 6

Explanation: Swap 'b' with the last 'a' (or the first 'a'), and we get longest repeated character substring "aaaaaa" with length 6.

Input: text = "aaaaa"

Output: 5

Explanation: No need to swap, longest repeated character substring is "aaaaa" with length is 5.

15.15 Get Equal Substrings within Budget

You are given two strings s and t of the same length and an integer maxCost.

You want to change s to t. Changing the i^{th} character of s to i^{th} character of t costs |s[i] - t[i]| (i.e., the absolute difference between the ASCII values of the characters).

Return the maximum length of a substring of s that can be changed to be the same as the corresponding substring of t with a cost less than or equal to maxCost. If there is no substring from s that can be changed to its corresponding substring from t, return 0.

Input: s = "abcd", t = "bcdf", maxCost = 3

Output: 3

Explanation: "abc" of s can change to "bcd".

That costs 3, so the maximum length is 3.

Input: s = "abcd", t = "cdef", maxCost = 3

Output: 1

Explanation: Each character in s costs 2 to change to character in t, so the maximum length is 1.

Input: s = "abcd", t = "acde", maxCost = 0

Output: 1

Explanation: You cannot make any change, so the maximum length is 1.

15.16 Replace the Substring for Balanced String

You are given a string s of length n containing only four kinds of characters: 'Q', 'W', 'E', and 'R'.

A string is said to be **balanced** if each of its characters appears n / 4 times where n is the length of the string.

Return the minimum length of the substring that can be replaced with **any** other string of the same length to make s **balanced**. If s is already **balanced**, return 0.

Input: s = "QWER"

Output: 0

Explanation: s is already balanced.

Input: s = "QQWE"

Output: 1

Explanation: We need to replace a 'Q' to 'R', so that "RQWE" (or "QRWE") is balanced.

Input: s = "QQQW"

Output: 2

Explanation: We can replace the first "QQ" to "ER".

15.17 Count Number of Nice Subarrays

Given an array of integer's nums and an integer k. A continuous subarray is called **nice** if there are k odd numbers on it.

Return the number of **nice** sub-arrays.

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

Output: 2

Explanation: The only sub-arrays with 3 odd numbers are [1, 1, 2, 1] and [1, 2, 1, 1].

Input: nums = [2, 4, 6], k = 1

Output: 0

Explanation: There is no odd numbers in the array.

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

Output: 16

15.18 Maximum Number of Occurrences of a Substring

Given a string s, return the maximum number of occurrences of **any** substring under the following rules:

- The number of unique characters in the substring must be less than or equal to maxLetters.
- The substring size must be between minSize and maxSize inclusive.

Input: s = "aababcaab", maxLetters = 2, minSize = 3, maxSize = 4

Output: 2

Explanation: Substring "aab" has 2 occurrences in the original string.

It satisfies the conditions, 2 unique letters and size 3 (between minSize and maxSize).

Input: s = "aaaa", maxLetters = 1, minSize = 3, maxSize = 3

Output: 2

Explanation: Substring "aaa" occur 2 times in the string. It can overlap.

15.19 Maximum Points You Can Obtain from Cards

There are several cards **arranged in a row**, and each card has an associated number of points. The points are given in the integer array cardPoints.

In one step, you can take one card from the beginning or from the end of the row. You have to take exactly k cards.

Your score is the sum of the points of the cards you have taken.

Given the integer array cardPoints and the integer k, return the maximum score you can obtain.

Input: cardPoints = [1, 2, 3, 4, 5, 6, 1], k = 3

Output: 12

Explanation: After the first step, your score will always be 1. However, choosing the rightmost card first will maximize your total score. The optimal strategy is to take the three cards on the right, giving a final score of 1 + 6 + 5 = 12.

Input: cardPoints = [2, 2, 2], k = 2

Output: 4

Explanation: Regardless of which two cards you take, your score will always be 4.

Input: cardPoints = [9, 7, 7, 9, 7, 7, 9], k = 7

Output: 55

Explanation: You have to take all the cards. Your score is the sum of points of all cards.

15.20 Find Two Non-overlapping Sub-arrays Each With Target Sum

You are given an array of integer's arr and an integer target.

You have to find **two non-overlapping sub-arrays** of arr each with a sum equal target. There can be multiple answers so you have to find an answer where the sum of the lengths of the two sub-arrays is **minimum**.

Return the minimum sum of the lengths of the two required sub-arrays, or return -1 if you cannot find such two sub-arrays.

Input: arr = [3, 2, 2, 4, 3], target = 3

Output: 2

Explanation: Only two sub-arrays have sum = 3 ([3] and [3]). The sum of their lengths is 2.

Input: arr = [7, 3, 4, 7], target = 7

Output: 2

Explanation: Although we have three non-overlapping sub-arrays of sum = 7 ([7], [3,4] and [7]), but we will choose the first and third sub-arrays as the sum of their lengths is 2.

Input: arr = [4, 3, 2, 6, 2, 3, 4], target = 6

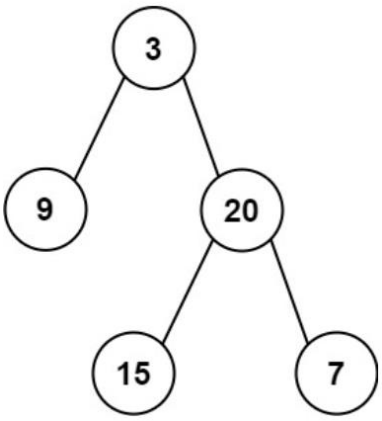
Output: -1

Explanation: We have only one sub-array of sum = 6.

16. Divide and Conquer

16.1 Construct Binary Tree from Preorder and Inorder Traversal

Given two integer arrays preorder and inorder where preorder is the preorder traversal of a binary tree and inorder is the inorder traversal of the same tree, construct and return the binary tree.



Input: preorder = [3, 9, 20, 15, 7], inorder = [9, 3, 15, 20, 7]

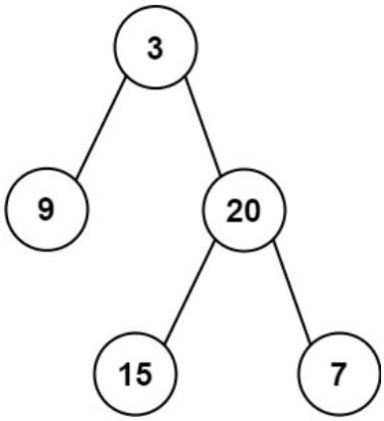
Output: [3, 9, 20, null, null, 15, 7]

Input: preorder = [-1], inorder = [-1]

Output: [-1]

16.2 Construct Binary Tree from Inorder and Postorder Traversal

Given two integer arrays inorder and postorder where inorder is the inorder traversal of a binary tree and postorder is the postorder traversal of the same tree, construct and return the binary tree.



Input: inorder = [9, 3, 15, 20, 7], postorder = [9, 15, 7, 20, 3]

Output: [3, 9, 20, null, null, 15, 7]

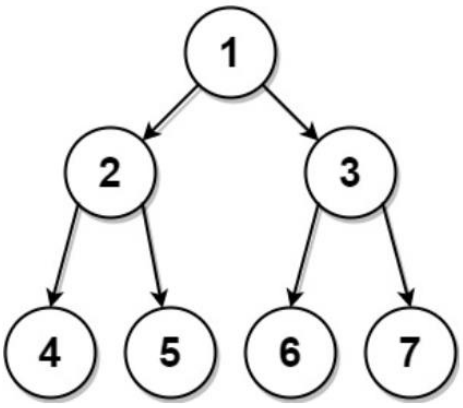
Input: inorder = [-1], postorder = [-1]

Output: [-1]

16.3 Construct Binary Tree from Preorder and Postorder Traversal

Given two integer arrays, preorder and postorder where preorder is the preorder traversal of a binary tree of **distinct** values and postorder is the postorder traversal of the same tree, reconstruct and return the binary tree.

If there exist multiple answers, you can **return any** of them.



Input: preorder = [1, 2, 4, 5, 3, 6, 7], postorder = [4, 5, 2, 6, 7, 3, 1]

Output: [1, 2, 3, 4, 5, 6, 7]

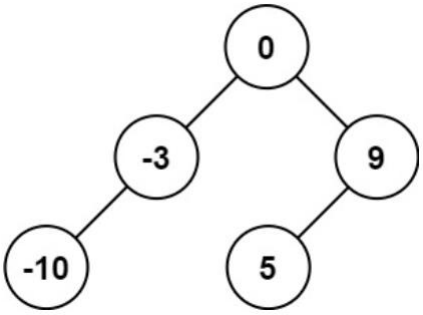
Input: preorder = [1], postorder = [1]

Output: [1]

16.4 Convert Sorted Array to Binary Search Tree

Given an integer array nums where the elements are sorted in **ascending order**, convert it to a

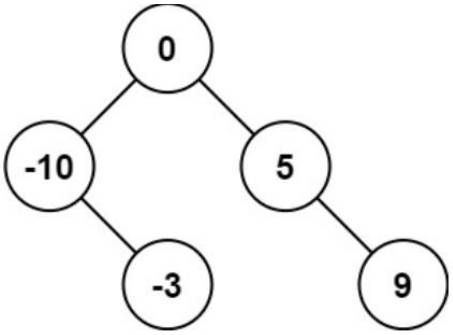
height-balanced binary search tree.

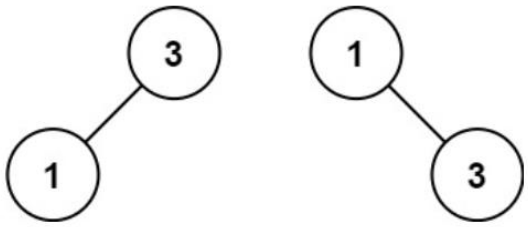


Input: nums = [-10, -3, 0, 5, 9]

Output: [0, -3, 9, -10, null, 5]

Explanation: [0, -10, 5, null, -3, null, 9] is also accepted:





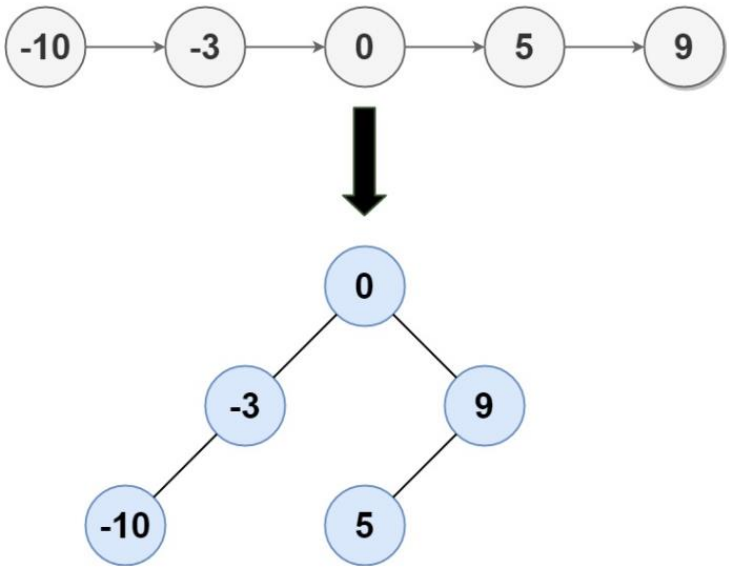
Input: nums = [1, 3]

Output: [3, 1]

Explanation: [1, null, 3] and [3, 1] are both height-balanced BSTs.

16.5 Convert Sorted List to Binary Search Tree

Given the head of a singly linked list where elements are sorted in **ascending order**, convert it to a **height-balanced** binary search tree.



Input: head = [-10, -3, 0, 5, 9]

Output: [0, -3, 9, -10, null, 5]

Explanation: One possible answer is [0, -3, 9, -10, null, 5], which represents the shown height balanced BST.

Input: head = []

Output: []

16.6 Maximum Sum Circular Subarray

Given a **circular integer array** nums of length n, return the maximum possible sum of a non-empty **subarray** of nums.

A **circular array** means the end of the array connects to the beginning of the array. Formally, the next element of nums[i] is nums[(i + 1) % n] and the previous element of nums[i] is nums[(i - 1 + n) % n].

A **subarray** may only include each element of the fixed buffer nums at most once. Formally, for a subarray nums[i], nums[i + 1], ..., nums[j], there does not exist i <= k1, k2 <= j with k1 % n == k2 % n.

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

Output: 3

Explanation: Subarray [3] has maximum sum 3.

Input: nums = [5, -3, 5]

Output: 10

Explanation: Subarray [5, 5] has maximum sum 5 + 5 = 10.

Input: nums = [-3, -2, -3]

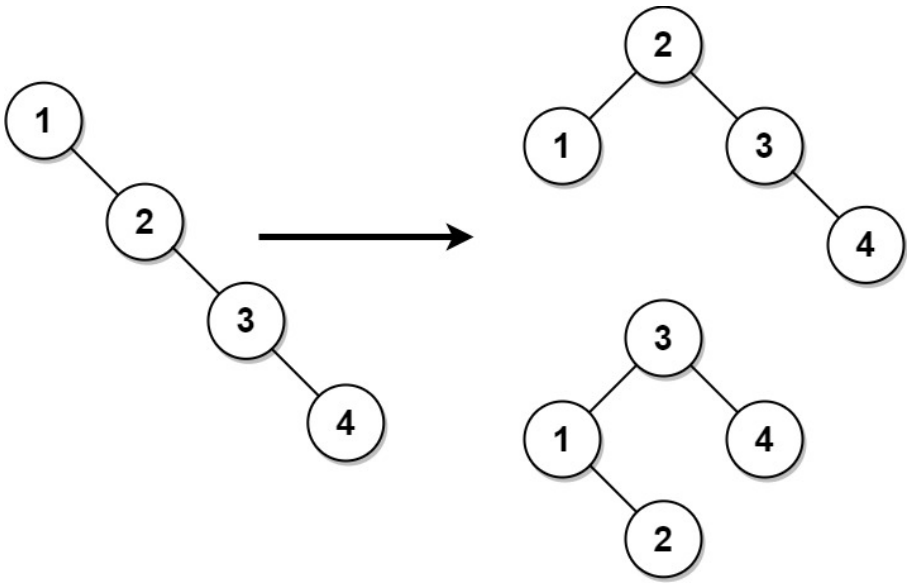
Output: -2

Explanation: Subarray [-2] has maximum sum -2.

16.7 Balance a Binary Search Tree

Given the root of a binary search tree, return a **balanced** binary search tree with the same node values. If there is more than one answer, return **any of them**.

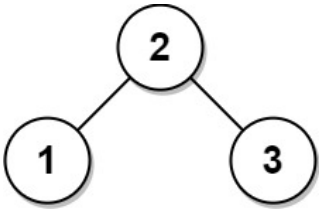
A binary search tree is **balanced** if the depth of the two subtrees of every node never differs by more than 1.



Input: root = [1, null, 2, null, 3, null, 4, null, null]

Output: [2, 1, 3, null, null, null, 4]

Explanation: This is not the only correct answer, [3, 1, 4, null, 2] is also correct.



Input: root = [2, 1, 3]

Output: [2, 1, 3]

16.8 Number of Pairs Satisfying Inequality

You are given two **0-indexed** integer arrays nums1 and nums2, each of size n, and an integer diff. Find the number of **pairs** (i, j) such that:

- 0 <= i < j <= n - 1 **and**
- nums1[i] - nums1[j] <= nums2[i] - nums2[j] + diff.

Return the **number of pairs** that satisfy the conditions.

Input: nums1 = [3, 2, 5], nums2 = [2, 2, 1], diff = 1

Output: 3

Explanation:

There are 3 pairs that satisfy the conditions:

1. i = 0, j = 1: 3 - 2 <= 2 - 2 + 1. Since i < j and 1 <= 1, this pair satisfies the conditions.

2. i = 0, j = 2: 3 - 5 <= 2 - 1 + 1. Since i < j and -2 <= 2, this pair satisfies the conditions.

3. i = 1, j = 2: 2 - 5 <= 2 - 1 + 1. Since i < j and -3 <= 2, this pair satisfies the conditions.

Therefore, we return 3.

Input: nums1 = [3, -1], nums2 = [-2, 2], diff = -1

Output: 0

Explanation:
Since there does not exist any pair that satisfies the conditions, we return 0.

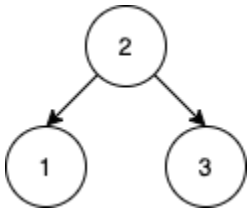
16.9 Number of Ways to Reorder Array to Get Same BST

Given an array nums that represents a permutation of integers from 1 to n. We are going to construct a binary search tree (BST) by inserting the elements of nums in order into an initially empty BST. Find the number of different ways to reorder nums so that the constructed BST is identical to that formed from the original array nums.

- For example, given nums = [2, 1, 3], we will have 2 as the root, 1 as a left child, and 3 as a right child. The array [2, 3, 1] also yields the same BST but [3,2,1] yields a different BST.

Return the number of ways to reorder nums such that the BST formed is identical to the original BST formed from nums.

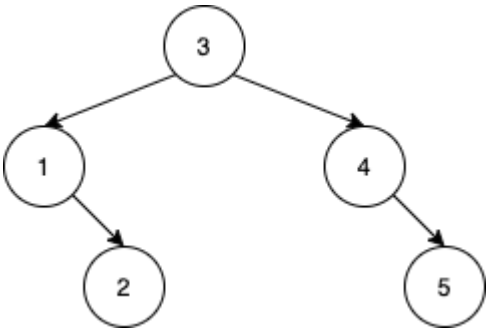
Since the answer may be very large, **return it modulo** $10^9 + 7$.



Input: nums = [2, 1, 3]

Output: 1

Explanation: We can reorder nums to be [2, 3, 1] which will yield the same BST. There are no other ways to reorder nums which will yield the same BST.



Input: nums = [3, 4, 5, 1, 2]

Output: 5

Explanation: The following 5 arrays will yield the same BST:

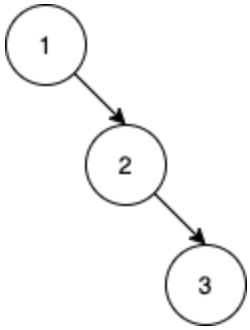
[3, 1, 2, 4, 5]

[3, 1, 4, 2, 5]

[3, 1, 4, 5, 2]

[3, 4, 1, 2, 5]

[3, 4, 1, 5, 2]



Input: nums = [1, 2, 3]

Output: 0

Explanation: There are no other orderings of nums that will yield the same BST.

16.10 Count of Smaller Numbers after Self

Given an integer array nums, return an integer array counts where counts[i] is the number of smaller elements to the right of nums[i].

Input: nums = [5, 2, 6, 1]

Output: [2, 1, 1, 0]

Explanation:

To the right of 5 there are **2** smaller elements (2 and 1).

To the right of 2 there is only **1** smaller element (1).

To the right of 6 there is **1** smaller element (1).

To the right of 1 there is **0** smaller element.

Input: nums = [-1]

Output: [0]

Input: nums = [-1, -1]

Output: [0, 0]

17. Greedy Algorithms

17.1 Task Scheduler

Given a characters array tasks, representing the tasks a CPU needs to do, where each letter represents a different task. Tasks could be done in any order. Each task is done in one unit of time. For each unit of time, the CPU could complete either one task or just be idle.

However, there is a non-negative integer n that represents the cooldown period between two **same tasks** (the same letter in the array), that is that there must be at least n units of time between any two same tasks.
Return the least number of units of times that the CPU will take to finish all the given tasks.

Input: tasks = ["A","A","A","B","B","B"], n = 2

Output: 8

Explanation:

A -> B -> idle -> A -> B -> idle -> A -> B

There is at least 2 units of time between any two same tasks.

Input: tasks = ["A","A","A","B","B","B"], n = 0

Output: 6

Explanation: On this case any permutation of size 6 would work since n = 0.

["A","A","A","B","B","B"]

["A","B","A","B","A","B"]

["B","B","B","A","A","A"]

...

And so on.

Input: tasks = ["A","A","A","A","A","A","B","C","D","E","F","G"], n = 2

Output: 16

Explanation:

One possible solution is

A -> B -> C -> A -> D -> E -> A -> F -> G -> A -> idle -> idle -> A -> idle -> idle -> A

17.2 Maximum Length of Pair Chain

You are given an array of n pairs pairs where pairs[i] = [$left_i$, $right_i$] and $left_i$ < $right_i$.
A pair p2 = [c, d] **follows** a pair p1 = [a, b] if b < c. A **chain** of pairs can be formed in this fashion.
Return the length longest chain which can be formed.
You do not need to use up all the given intervals. You can select pairs in any order.

Input: pairs = [[1, 2], [2, 3], [3, 4]]

Output: 2

Explanation: The longest chain is [1, 2] -> [3, 4].

Example 2:

Input: pairs = [[1, 2], [7, 8], [4, 5]]

Output: 3

Explanation: The longest chain is [1, 2] -> [4, 5] -> [7, 8].

17.3 Split Array into Consecutive Subsequences

You are given an integer array nums that is **sorted in non-decreasing order**.
Determine if it is possible to split nums into **one or more subsequences** such that **both** of the following conditions are true:

- Each subsequence is a **consecutive increasing sequence** (i.e. each integer is **exactly one** more than the previous integer).
- All subsequences have a length of 3 **or more**.

Return true if you can split nums according to the above conditions, or false otherwise.
A **subsequence** of an array is a new array that is formed from the original array by deleting some (can be none) of the elements without disturbing the relative positions of the remaining elements. (i.e., [1,3,5] is a subsequence of [1,2,3,4,5] while [1,3,2] is not).

Input: nums = [1, 2, 3, 3, 4, 5]

Output: true

Explanation: nums can be split into the following subsequences:

[**1**, **2**, **3**, 3, 4, 5] --> 1, 2, 3

[1, 2, 3, **3**, **4**, **5**] --> 3, 4, 5

Input: nums = [1, 2, 3, 3, 4, 4, 5, 5]

Output: true

Explanation: nums can be split into the following subsequences:

[**1**, **2**, **3**, 3, **4**, 4, **5**, 5] --> 1, 2, 3, 4, 5

[1, 2, 3, **3**, 4, **4**, 5, **5**] --> 3, 4, 5

Input: nums = [1, 2, 3, 4, 4, 5]

Output: false

Explanation: It is impossible to split nums into consecutive increasing subsequences of length 3 or more.

17.4 Maximum Swap

You are given an integer num. You can swap two digits at most once to get the maximum valued number.

Return the maximum valued number you can get.

Input: num = 2736

Output: 7236

Explanation: Swap the number 2 and the number 7.

Example 2:

Input: num = 9973

Output: 9973

Explanation: No swap.

17.5 Valid Parenthesis String

Given a string s containing only three types of characters: '(', ')' and '*', return true *if* s *is* ***valid***.

The following rules define a **valid** string:

- Any left parenthesis '(' must have a corresponding right parenthesis ')'.
- Any right parenthesis ')' must have a corresponding left parenthesis '('.
- Left parenthesis '(' must go before the corresponding right parenthesis ')'.
- '*' could be treated as a single right parenthesis ')' or a single left parenthesis '(' or an empty string "".

Input: s = "()"

Output: true

Input: s = "(*)"

Output: true

Input: s = "(*))"

Output: true

17.6 Best Time to Buy and Sell Stock with Transaction Fee

You are given an array prices where prices[i] is the price of a given stock on the i^{th} day, and an integer fee representing a transaction fee.

Find the maximum profit you can achieve. You may complete as many transactions as you like, but you need to pay the transaction fee for each transaction.

Note:

- You may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again).
- The transaction fee is only charged once for each stock purchase and sale.

Input: prices = [1, 3, 2, 8, 4, 9], fee = 2

Output: 8

Explanation: The maximum profit can be achieved by:

- Buying at prices[0] = 1

- Selling at prices[3] = 8

- Buying at prices[4] = 4

- Selling at prices[5] = 9

The total profit is ((8 - 1) - 2) + ((9 - 4) - 2) = 8.

Example 2:

Input: prices = [1, 3, 7, 5, 10, 3], fee = 3

Output: 6

17.7 Monotone Increasing Digits

An integer has **monotone increasing digits** if and only if each pair of adjacent digits x and y satisfy x <= y.

Given an integer n, return the largest number that is less than or equal to n with **monotone increasing digits**.

Input: n = 10

Output: 9

Input: n = 1234

Output: 1234

Input: n = 332

Output: 299

17.8 Reorganize String

Given a string s, rearrange the characters of s so that any two adjacent characters are not the same.

Return any possible rearrangement of s or return "" if not possible.

Input: s = "aab"

Output: "aba"

Input: s = "aaab"

Output: ""

17.9 Rabbits in Forest

There is a forest with an unknown number of rabbits. We asked n rabbits **"How many rabbits have the same color as you?"** and collected the answers in an integer array answers where answers[i] is the answer of the i^{th} rabbit.

Given the array answers, return the minimum number of rabbits that could be in the forest.

Input: answers = [1, 1, 2]

Output: 5

Explanation:

The two rabbits that answered "1" could both be the same color, say red.

The rabbit that answered "2" can't be red or the answers would be inconsistent.

Say the rabbit that answered "2" was blue.

Then there should be 2 other blue rabbits in the forest that didn't answer into the array.

The smallest possible number of rabbits in the forest is therefore 5: 3 that answered plus 2 that didn't.

Input: answers = [10, 10, 10]

Output: 11

17.10 Most Profit Assigning Work

You have n jobs and m workers. You are given three arrays: difficulty, profit, and worker where:

- difficulty[i] and profit[i] are the difficulty and the profit of the i^{th} job, and
- worker[j] is the ability of j^{th} worker (i.e., the j^{th} worker can only complete a job with difficulty at most worker[j]).

Every worker can be assigned **at most one job**, but one job can be **completed multiple times**.

- For example, if three workers attempt the same job that pays $1, then the total profit will be $3. If a worker cannot complete any job, their profit is $0.

Return the maximum profit we can achieve after assigning the workers to the jobs.

Input: difficulty = [2, 4, 6, 8, 10], profit = [10, 20, 30, 40, 50], worker = [4, 5, 6, 7]

Output: 100

Explanation: Workers are assigned jobs of difficulty [4, 4, 6, 6] and they get a profit of [20,20,30,30] separately.

Input: difficulty = [85, 47, 57], profit = [24, 66, 99], worker = [40, 25, 25]

Output: 0

17.11 Hand of Straights

Alice has some number of cards and she wants to rearrange the cards into groups so that each group is of size groupSize, and consists of groupSize consecutive cards.

Given an integer array hand where hand[i] is the value written on the i^{th} card and an integer groupSize, return true if she can rearrange the cards, or false otherwise.

Input: hand = [1, 2, 3, 6, 2, 3, 4, 7, 8], groupSize = 3

Output: true

Explanation: Alice's hand can be rearranged as [1,2,3],[2,3,4],[6,7,8]

Example 2:

Input: hand = [1, 2, 3, 4, 5], groupSize = 4

Output: false

Explanation: Alice's hand cannot be rearranged into groups of 4.

17.12 Lemonade Change

At a lemonade stand, each lemonade costs $5. Customers are standing in a queue to buy from you and order one at a time (in the order specified by bills). Each customer will only buy one lemonade and pay with either a $5, $10, or $20 bill. You must provide the correct change to each customer so that the net transaction is that the customer pays $5.

Note that you do not have any change in hand at first.

Given an integer array bills where bills[i] is the bill the i^{th} customer pays, return true if you can provide every customer with the correct change, or false otherwise.

Input: bills = [5, 5, 5, 10, 20]

Output: true

Explanation:

From the first 3 customers, we collect three $5 bills in order.

From the fourth customer, we collect a $10 bill and give back a $5.

From the fifth customer, we give a $10 bill and a $5 bill.

Since all customers got correct change, we output true.

Example 2:

Input: bills = [5, 5, 10, 10, 20]

Output: false

Explanation:

From the first two customers in order, we collect two $5 bills.

For the next two customers in order, we collect a $10 bill and give back a $5 bill.

For the last customer, we cannot give the change of $15 back because we only have two $10 bills.

Since not every customer received the correct change, the answer is false.

17.13 Advantage Shuffle

You are given two integer arrays nums1 and nums2 both of the same length.
The **advantage** of nums1 with respect to nums2 is the number of indices i for which nums1[i] > nums2[i].

Return any permutation of nums1 that maximizes its **advantage** with respect to nums2.

Input: nums1 = [2, 7, 11, 15], nums2 = [1, 10, 4, 11]

Output: [2, 11, 7, 15]

Input: nums1 = [12, 24, 8, 32], nums2 = [13, 25, 32, 11]

Output: [24, 32, 8, 12]

17.14 Boats to Save People

You are given an array people where people[i] is the weight of the i^{th} person, and an **infinite number of boats** where each boat can carry a maximum weight of limit. Each boat carries at most two people at the same time, provided the sum of the weight of those people is at most limit.

Return the minimum number of boats to carry every given person.

Input: people = [1, 2], limit = 3

Output: 1

Explanation: 1 boat (1, 2)

Input: people = [3, 2, 2, 1], limit = 3

Output: 3

Explanation: 3 boats (1, 2), (2) and (3)

Input: people = [3, 5, 3, 4], limit = 5

Output: 4

Explanation: 4 boats (3), (3), (4), (5)

17.15 DI String Match

A permutation perm of n + 1 integers of all the integers in the range [0, n] can be represented as a string s of length n where:

- s[i] == 'I' if perm[i] < perm[i + 1], and
- s[i] == 'D' if perm[i] > perm[i + 1].

Given a string s, reconstruct the permutation perm and return it. If there are multiple valid permutations perm, return **any of them**.

Input: s = "IDID"

Output: [0, 4, 1, 3, 2]

Input: s = "III"

Output: [0, 1, 2, 3]

Input: s = "DDI"

Output: [3, 2, 0, 1]

17.16 Minimum Increment to Make Array Unique

You are given an integer array nums. In one move, you can pick an index i where 0 <= i < nums.length and increment nums[i] by 1.

Return the minimum number of moves to make every value in nums **unique**.

The test cases are generated so that the answer fits in a 32-bit integer.

Input: nums = [1, 2, 2]

Output: 1

Explanation: After 1 move, the array could be [1, 2, 3].

Input: nums = [3, 2, 1, 2, 1, 7]

Output: 6

Explanation: After 6 moves, the array could be [3, 4, 1, 2, 5, 7].

It can be shown with 5 or less moves that it is impossible for the array to have all unique values.

17.17 Bag of Tokens

You have an initial **power** of power, an initial **score** of 0, and a bag of tokens where tokens[i] is the value of the i^{th} token (0-indexed).

Your goal is to maximize your total **score** by potentially playing each token in one of two ways:

- If your current **power** is at least tokens[i], you may play the i^{th} token face up, losing tokens[i] **power** and gaining 1 **score**.
- If your current **score** is at least 1, you may play the i^{th} token face down, gaining tokens[i] **power** and losing 1 **score**.

Each token may be played **at most** once and **in any order**. You do **not** have to play all the tokens.

Return the largest possible **score** you can achieve after playing any number of tokens.

Input: tokens = [100], power = 50

Output: 0

Explanation: Playing the only token in the bag is impossible because you either have too little power or too little score.

Input: tokens = [100,200], power = 150

Output: 1

Explanation: Play the 0th token (100) face up, your power becomes 50 and score becomes 1.

There is no need to play the 1st token since you cannot play it face up to add to your score.

Input: tokens = [100,200,300,400], power = 200

Output: 2

Explanation: Play the tokens in this order to get a score of 2:

1. Play the 0th token (100) face up, your power becomes 100 and score becomes 1.

2. Play the 3rd token (400) face down, your power becomes 500 and score becomes 0.

3. Play the 1st token (200) face up, your power becomes 300 and score becomes 1.

4. Play the 2nd token (300) face up, your power becomes 0 and score becomes 2.

17.18 Largest Perimeter Triangle

Given an integer array nums, return the largest perimeter of a triangle with a non-zero area, formed from three of these lengths. If it is impossible to form any triangle of a non-zero area, return 0.

Input: nums = [2, 1, 2]

Output: 5

Explanation: You can form a triangle with three side lengths: 1, 2, and 2.

Input: nums = [1, 2, 1, 10]

Output: 0

Explanation:

You cannot use the side lengths 1, 1, and 2 to form a triangle.

You cannot use the side lengths 1, 1, and 10 to form a triangle.

You cannot use the side lengths 1, 2, and 10 to form a triangle.

As we cannot use any three side lengths to form a triangle of non-zero area, we return 0.

17.19 String without AAA or BBB

Given two integers a and b, return **any** string s such that:

- s has length a + b and contains exactly a 'a' letters, and exactly b 'b' letters,
- The substring 'aaa' does not occur in s, and
- The substring 'bbb' does not occur in s.

Input: a = 1, b = 2

Output: "abb"

Explanation: "abb", "bab" and "bba" are all correct answers.

Input: a = 4, b = 1

Output: "aabaa"

17.20 Maximum Sum of Array after K Negations

Given an integer array nums and an integer k, modify the array in the following way:

- Choose an index i and replace nums[i] with -nums[i].

You should apply this process exactly k times. You may choose the same index i multiple times.

Return the largest possible sum of the array after modifying it in this way.

Input: nums = [4, 2, 3], k = 1

Output: 5

Explanation: Choose index 1 and nums becomes [4, -2, 3].

Input: nums = [3, -1, 0, 2], k = 3

Output: 6

Explanation: Choose indices (1, 2, 2) and nums becomes [3, 1, 0, 2].

Input: nums = [2, -3, -1, 5, -4], k = 2

Output: 13

Explanation: Choose indices (1, 4) and nums becomes [2, 3, -1, 5, 4].

17.21 Broken Calculator

There is a broken calculator that has the integer startValue on its display initially. In one operation, you can:

- multiply the number on display by 2, or
- subtract 1 from the number on display.

Given two integers startValue and target, return the minimum number of operations needed to display target on the calculator.

Input: startValue = 2, target = 3

Output: 2

Explanation: Use double operation and then decrement operation {2 -> 4 -> 3}.

Input: startValue = 5, target = 8

Output: 2

Explanation: Use decrement and then double {5 -> 4 -> 8}.

Input: startValue = 3, target = 10

Output: 3

Explanation: Use double, decrement and double {3 -> 6 -> 5 -> 10}.

17.22 Two City Scheduling

A company is planning to interview 2n people. Given the array costs where costs[i] = [$aCost_i$, $bCost_i$], the cost of flying the i^{th} person to city a is $aCost_i$, and the cost of flying the i^{th} person to city b is $bCost_i$.

Return the minimum cost to fly every person to a city such that exactly n people arrive in each city.

Input: costs = [[10, 20], [30, 200], [400, 50], [30, 20]]

Output: 110

Explanation:

The first person goes to city A for a cost of 10.

The second person goes to city A for a cost of 30.

The third person goes to city B for a cost of 50.

The fourth person goes to city B for a cost of 20.

The total minimum cost is 10 + 30 + 50 + 20 = 110 to have half the people interviewing in each city.

Input: costs = [[259, 770], [448, 54], [926, 667], [184, 139], [840, 118], [577, 469]]

Output: 1859

Input: costs = [[515, 563], [451, 713], [537, 709], [343, 819], [855, 779], [457, 60], [650, 359], [631, 42]]

Output: 3086

17.23 Largest Values from Labels

There is a set of n items. You are given two integer arrays values and labels where the value and the label of the i^{th} element are values[i] and labels[i] respectively. You are also given two integers numWanted and useLimit.

Choose a subset s of the n elements such that:

- The size of the subset s is **less than or equal to** numWanted.
- There are **at most** useLimit items with the same label in s.

The **score** of a subset is the sum of the values in the subset.

Return the maximum **score** of a subset s.

Input: values = [5, 4, 3, 2, 1], labels = [1, 1, 2, 2, 3], numWanted = 3, useLimit = 1

Output: 9

Explanation: The subset chosen is the first, third, and fifth items.

Input: values = [5, 4, 3, 2, 1], labels = [1, 3, 3, 3, 2], numWanted = 3, useLimit = 2

Output: 12

Explanation: The subset chosen is the first, second, and third items.

Input: values = [9, 8, 8, 7, 6], labels = [0, 0, 0, 1, 1], numWanted = 3, useLimit = 1

Output: 16

Explanation: The subset chosen is the first and fourth items.

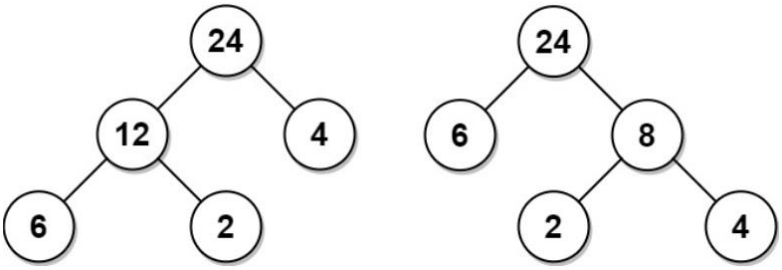
17.24 Minimum Cost Tree from Leaf Values

Given an array arr of positive integers, consider all binary trees such that:

- Each node has either 0 or 2 children;
- The values of arr correspond to the values of each **leaf** in an in-order traversal of the tree.
- The value of each non-leaf node is equal to the product of the largest leaf value in its left and right subtree, respectively.

Among all possible binary trees considered, return the smallest possible sum of the values of each non-leaf node. It is guaranteed this sum fits into a **32-bit** integer.

A node is a **leaf** if and only if it has zero children.

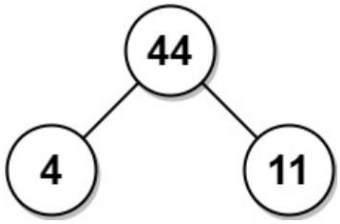


Input: arr = [6, 2, 4]

Output: 32

Explanation: There are two possible trees shown.

The first has a non-leaf node sum 36, and the second has non-leaf node sum 32.



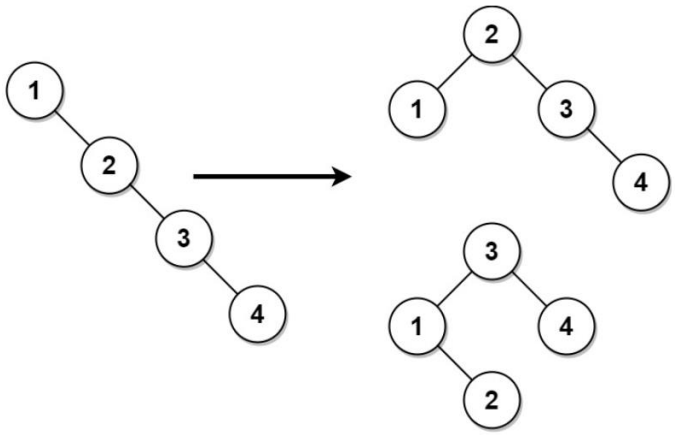
Input: arr = [4, 11]

Output: 44

17.25 Balance a Binary Search Tree

Given the root of a binary search tree, return a **balanced** binary search tree with the same node values. If there is more than one answer, return **any of them**.

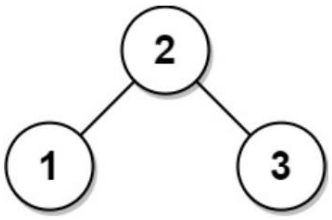
A binary search tree is **balanced** if the depth of the two subtrees of every node never differs by more than 1.



Input: root = [1, null, 2, null, 3, null, 4, null, null]

Output: [2, 1, 3, null, null, null, 4]

Explanation: This is not the only correct answer, [3, 1, 4, null, 2] is also correct.



Input: root = [2, 1, 3]

Output: [2, 1, 3]

17.26 Longest Happy String

A string s is called **happy** if it satisfies the following conditions:

- s only contains the letters 'a', 'b', and 'c'.
- s does not contain any of "aaa", "bbb", or "ccc" as a substring.
- s contains **at most** a occurrences of the letter 'a'.
- s contains **at most** b occurrences of the letter 'b'.
- s contains **at most** c occurrences of the letter 'c'.

Given three integers a, b, and c, return the **longest possible happy** string. If there are multiple longest happy strings, return any of them. If there is no such string, return the empty string "".

A **substring** is a contiguous sequence of characters within a string.

Input: a = 1, b = 1, c = 7

Output: "ccaccbcc"

Explanation: "ccbccacc" would also be a correct answer.

Input: a = 7, b = 1, c = 0

Output: "aabaa"

Explanation: It is the only correct answer in this case.

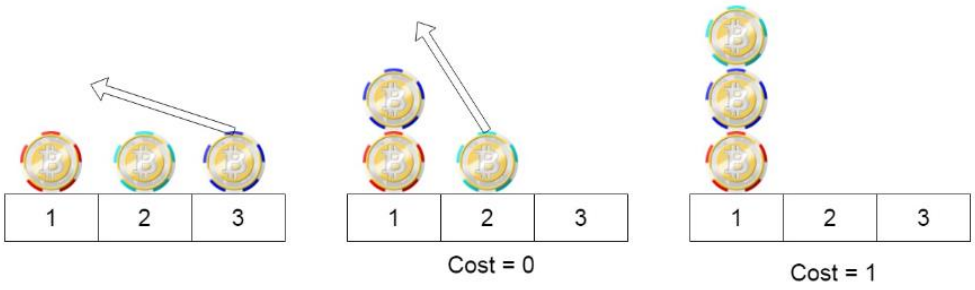
17.27 Minimum Cost to Move Chips to the Same Position

We have n chips, where the position of the i^{th} chip is position[i].

We need to move all the chips to **the same position**. In one step, we can change the position of the i^{th} chip from position[i] to:

- position[i] + 2 or position[i] - 2 with cost = 0.
- position[i] + 1 or position[i] - 1 with cost = 1.

Return the minimum cost needed to move all the chips to the same position.



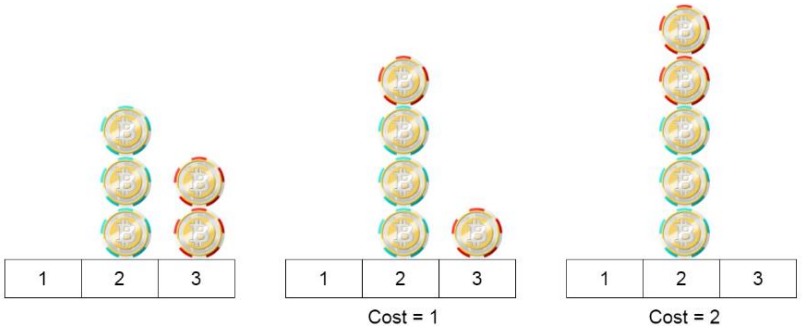
Input: position = [1, 2, 3]

Output: 1

Explanation: First step: Move the chip at position 3 to position 1 with cost = 0.

Second step: Move the chip at position 2 to position 1 with cost = 1.

Total cost is 1.



Input: position = [2, 2, 2, 3, 3]

Output: 2

Explanation: We can move the two chips at position 3 to position 2. Each move has cost = 1. The total cost = 2.

Input: position = [1, 1000000000]

Output: 1

17.28 Split a String in Balanced Strings

Balanced strings are those that have an equal quantity of 'L' and 'R' characters.

Given a **balanced** string s, split it into some number of substrings such that:

- Each substring is balanced.

Return the **maximum** number of balanced strings you can obtain.

Input: s = "RLRRLLRLRL"

Output: 4

Explanation: s can be split into "RL", "RRLL", "RL", "RL", each substring contains same number of 'L' and 'R'.

Input: s = "RLRRRLLRLL"

Output: 2

Explanation: s can be split into "RL", "RRRLLRLL", each substring contains same number of 'L' and 'R'.

Note that s cannot be split into "RL", "RR", "RL", "LR", "LL", because the 2nd and 5th substrings are not balanced.

Input: s = "LLLLRRRR"

Output: 1

Explanation: s can be split into "LLLLRRRR".

17.29 Maximum 69 Number

You are given a positive integer num consisting only of digits 6 and 9.

Return the maximum number you can get by changing at most one digit (6 becomes 9, and 9 becomes 6).

Input: num = 9669

Output: 9969

Explanation:

Changing the first digit results in 6669.

Changing the second digit results in 9969.

Changing the third digit results in 9699.

Changing the fourth digit results in 9666.

The maximum number is 9969.

Input: num = 9996

Output: 9999

Explanation: Changing the last digit 6 to 9 results in the maximum number.

Input: num = 9999

Output: 9999

Explanation: It is better not to apply any change.

7.30 Assign Cookies

Assume you are an awesome parent and want to give your children some cookies. But, you should give each child at most one cookie.

Each child i has a greed factor g[i], which is the minimum size of a cookie that the child will be content with; and each cookie j has a size s[j]. If s[j] >= g[i], we can assign the cookie j to the child i, and the child i will be content. Your goal is to maximize the number of your content children and output the maximum number.

Input: g = [1, 2, 3], s = [1, 1]

Output: 1

Explanation: You have 3 children and 2 cookies. The greed factors of 3 children are 1, 2, 3.

And even though you have 2 cookies, since their size is both 1, you could only make the child whose greed factor is 1 content.

You need to output 1.

Input: g = [1, 2], s = [1, 2, 3]

Output: 2

Explanation: You have 2 children and 3 cookies. The greed factors of 2 children are 1, 2.

You have 3 cookies and their sizes are big enough to gratify all of the children,

You need to output 2.

18. Dynamic Programming

18.1. Muzicari

"The Drinking Musicians", a widely known and popular folk group, are coming to your town. The musicians are known not only by their playing skills, but also their rough character. They never arrive on time, don't know which town they're in, and frequently have trouble finding the stage.

Additionally, during the concert, each of the musicians at one point takes a break. If three or more of them are on a break at the same time, they start stirring trouble in town and the rest of the group start panicking and playing the wrong chords.

The concert will be T minutes long, during which each of the N members will take a break. The length of the break is known for each member.

Help the organizer of the concert by writing a program that determines how to schedule the breaks of the members so that, at any given moment, at most two are absent from the stage. All breaks must be **entirely** during the concert.

Input
The first line of input contains the integers T and N ($1 \le T \le 5000$, $1 \le N \le 500$), the length of the concert in minutes and the number of musicians in the group.

The next line contains N integers (each between 1 and T inclusive) separated by single spaces, the length of the break in minutes for each member.

Note: The input data will be such that a solution, although not necessarily unique, will always exist.

Output
For each musician output one integer, the number of minutes the musician will spend on stage before going on the break. Output the musicians in the same order they were given in the input.

Sample Input 1:
8 3

4 4 4

Sample Output 1:
0 2 4

Sample Input 2:
10 5
7 5 1 2 3

Sample Output 2:
3 3 9 0 0

18.2 Climb the Stairs

You are climbing a staircase. It takes n steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Input: n = 2

Output: 2

Explanation: There are two ways to climb to the top.

1. 1 step + 1 step

2. 2 steps

Input: n = 3

Output: 3

Explanation: There are three ways to climb to the top.

1. 1 step + 1 step + 1 step

2. 1 step + 2 steps

3. 2 steps + 1 step

18.3 Edit Distance

The Given two strings word1 and word2, return the minimum number of operations required to convert word1 to word2.
You have the following three operations permitted on a word:

- Insert a character
- Delete a character
- Replace a character

Input: word1 = "horse", word2 = "ros"

Output: 3

Explanation:

horse -> rorse (replace 'h' with 'r')

rorse -> rose (remove 'r')

rose -> ros (remove 'e')

Input: word1 = "intention", word2 = "execution"

Output: 5

Explanation:

intention -> inention (remove 't')

inention -> enention (replace 'i' with 'e')

enention -> exention (replace 'n' with 'x')

exention -> exection (replace 'n' with 'c')

exection -> execution (insert 'u')

18.4 Decode Ways

A message containing letters from A-Z can be **encoded** into numbers using the following mapping:

'A' -> "1"

'B' -> "2"

...

'Z' -> "26"

To **decode** an encoded message, all the digits must be grouped then mapped back into letters using the reverse of the mapping above (there may be multiple ways). For example, "11106" can be mapped into:

- "AAJF" with the grouping (1 1 10 6)
- "KJF" with the grouping (11 10 6)

Note that the grouping (1 11 06) is invalid because "06" cannot be mapped into 'F' since "6" is different from "06".

Given a string s containing only digits, return *the **number** of ways to **decode** it*.

The test cases are generated so that the answer fits in a **32-bit** integer.

Input: s = "12"

Output: 2

Explanation: "12" could be decoded as "AB" (1 2) or "L" (12).

Input: s = "226"

Output: 3

Explanation: "226" could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).

Input: s = "06"

Output: 0

Explanation: "06" cannot be mapped to "F" because of the leading zero ("6" is different from "06").

18.5 Maximum Subarray

Given an integer array nums, find the subarray with the largest sum, and return *its sum*.

Input: nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4]

Output: 6

Explanation: The subarray [4, -1, 2, 1] has the largest sum 6.

Input: nums = [1]

Output: 1

Explanation: The subarray [1] has the largest sum 1.

Input: nums = [5, 4, -1, 7, 8]

Output: 23

Explanation: The subarray [5, 4,-1, 7, 8] has the largest sum 23.

18.6 Jump Game

You are given an integer array nums. You are initially positioned at the array's **first index**, and each element in the array represents your maximum jump length at that position.

Return true *if you can reach the last index, or* false *otherwise*.

Input: nums = [2, 3, 1, 1, 4]

Output: true

Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.

Input: nums = [3, 2, 1, 0, 4]

Output: false

Explanation: You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.

18.7 Jump Game II

You are given a **0-indexed** array of integer's nums of length n. You are initially positioned at nums[0].

Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at nums[i], you can jump to any nums[i + j] where:

- 0 <= j <= nums[i] and
- i + j < n

Return *the minimum number of jumps to reach* nums[n - 1]. The test cases are generated such that you can reach nums[n - 1].

Input: nums = [2, 3, 1, 1, 4]

Output: 2

Explanation: The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.

Input: nums = [2, 3, 0, 1, 4]

Output: 2

18.8 Unique Paths

There is a robot on an m x n grid. The robot is initially located at the **top-left corner** (i.e., grid[0][0]). The robot tries to move to the **bottom-right corner** (i.e., grid[m - 1][n - 1]). The robot can only move either down or right at any point in time.

Given the two integers m and n, return the number of possible unique paths that the robot can take to reach the bottom-right corner.

The test cases are generated so that the answer will be less than or equal to $2 * 10^9$.

Input: m = 3, n = 7

Output: 28

Example 2:

Input: m = 3, n = 2

Output: 3

Explanation: From the top-left corner, there are a total of 3 ways to reach the bottom-right corner:

1. Right -> Down -> Down

2. Down -> Down -> Right

3. Down -> Right -> Down

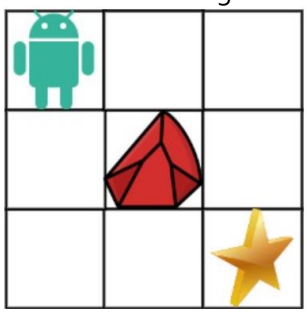
18.9 Unique Paths II

You are given an m x n integer array grid. There is a robot initially located at the **top-left corner** (i.e., grid[0][0]). The robot tries to move to the **bottom-right corner** (i.e., grid[m - 1][n - 1]). The robot can only move either down or right at any point in time.

An obstacle and space are marked as 1 or 0 respectively in grid. A path that the robot takes cannot include **any** square that is an obstacle.

Return the number of possible unique paths that the robot can take to reach the bottom-right corner.

The test cases are generated so that the answer will be less than or equal to $2 * 10^9$.



Input: obstacleGrid = [[0,0,0],[0,1,0],[0,0,0]]

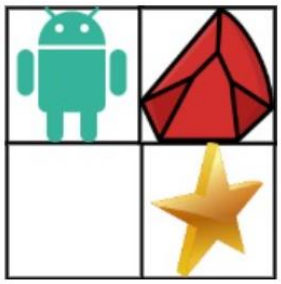
Output: 2

Explanation: There is one obstacle in the middle of the 3x3 grid above.

There are two ways to reach the bottom-right corner:

1. Right -> Right -> Down -> Down

2. Down -> Down -> Right -> Right

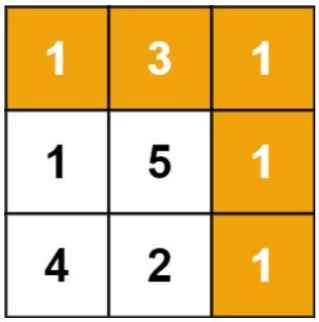


Input: obstacleGrid = [[0, 1], [0, 0]]

Output: 1

18.10 Minimum Path Sum

Given a m x n grid filled with non-negative numbers, find a path from top left to bottom right, which minimizes the sum of all numbers along its path.
Note: You can only move either down or right at any point in time.



Input: grid = [[1, 3, 1], [1, 5, 1], [4, 2, 1]]

Output: 7

Explanation: Because the path 1 → 3 → 1 → 1 → 1 minimizes the sum.

Example 2:

Input: grid = [[1, 2, 3], [4, 5, 6]]

Output: 12

18.11 Trapping Rain Water

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Input: height = [0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]

Output: 6

Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

Input: height = [4, 2, 0, 3, 2, 5]

Output: 9

18.12 Best Time to Buy and Sell Stock

You are given an array prices where prices[i] is the price of a given stock on the i^{th} day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

Input: prices = [7, 1, 5, 3, 6, 4]

Output: 5

Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.

Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.

Input: prices = [7, 6, 4, 3, 1]

Output: 0

Explanation: In this case, no transactions are done and the max profit = 0.

18.13 Palindrome Partitioning

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

Input: s = "aab"

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

Example 2:

Input: s = "a"

Output: [["a"]]

18.14 House Robber

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given an integer array nums representing the amount of money of each house, return the maximum amount of money you can rob tonight **without alerting the police**.

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

Output: 4

Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).

Total amount you can rob = 1 + 3 = 4.

Input: nums = [2, 7, 9, 3, 1]

Output: 12

Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).

Total amount you can rob = 2 + 9 + 1 = 12.

18.15 Different Ways to Add Parenthesis

Given a string expression of numbers and operators, return all possible results from computing all the different possible ways to group numbers and operators. You may return the answer in **any order**.

The test cases are generated such that the output values fit in a 32-bit integer and the number of different results does not exceed 10^4.

Input: expression = "2-1-1"

Output: [0, 2]

Explanation:

((2-1)-1) = 0

(2-(1-1)) = 2

Input: expression = "2*3-4*5"

Output: [-34, -14, -10, -10, 10]

Explanation:

(2*(3-(4*5))) = -34

((2*3)-(4*5)) = -14

((2*(3-4))*5) = -10

(2*((3-4)*5)) = -10

(((2*3)-4)*5) = 10

18.16 Ugly Number II

An **ugly number** is a positive integer whose prime factors are limited to 2, 3, and 5.

Given an integer n, return the n^{th} **ugly number**.

Input: n = 10

Output: 12

Explanation: [1, 2, 3, 4, 5, 6, 8, 9, 10, 12] is the sequence of the first 10 ugly numbers.

Input: n = 1

Output: 1

Explanation: 1 has no prime factors, therefore all of its prime factors are limited to 2, 3, and 5.

18.17 Perfect Squares

Given an integer n, return the least number of perfect square numbers that sum to n.

A **perfect square** is an integer that is the square of an integer; in other words, it is the product of some integer with itself. For example, 1, 4, 9, and 16 are perfect squares while 3 and 11 are not.

Input: n = 12

Output: 3

Explanation: 12 = 4 + 4 + 4.

Input: n = 13

Output: 2

Explanation: 13 = 4 + 9.

18.18 Arithmetic Slices

An integer array is called arithmetic if it consists of **at least three elements** and if the difference between any two consecutive elements is the same.

- For example, [1,3,5,7,9], [7,7,7,7], and [3,-1,-5,-9] are arithmetic sequences.

Given an integer array nums, return the number of arithmetic **subarrays** of nums.

A **subarray** is a contiguous subsequence of the array.

Input: nums = [1, 2, 3, 4]

Output: 3

Explanation: We have 3 arithmetic slices in nums: [1, 2, 3], [2, 3, 4] and [1,2,3,4] itself.

Input: nums = [1]

Output: 0

18.19 Longest Palindromic Subsequence

Given a string s, find the longest palindromic **subsequence**'s length in s.

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

Input: s = "bbbab"

Output: 4

Explanation: One possible longest palindromic subsequence is "bbbb".

Input: s = "cbbd"

Output: 2

Explanation: One possible longest palindromic subsequence is "bb".

18.20 Coin Change

You are given an integer array coins representing coins of different denominations and an integer amount representing a total amount of money.

Return the number of combinations that make up that amount. If that amount of money cannot be made up by any combination of the coins, return 0.

You may assume that you have an infinite number of each kind of coin.

The answer is **guaranteed** to fit into a signed **32-bit** integer.

Input: amount = 5, coins = [1,2,5]

Output: 4

Explanation: there are four ways to make up the amount:

5=5

5=2+2+1

5=2+1+1+1

5=1+1+1+1+1

Input: amount = 3, coins = [2]

Output: 0

Explanation: the amount of 3 cannot be made up just with coins of 2.

Input: amount = 10, coins = [10]

Output: 1

18.21 Beautiful Arrangement

Suppose you have n integers labeled 1 through n. A permutation of those n integers perm (**1-indexed**) is considered a **beautiful arrangement** if for every i (1 <= i <= n), **either** of the following is true:

- perm[i] is divisible by i.
- i is divisible by perm[i].

Given an integer n, return the **number** of the **beautiful arrangements** that you can construct.

Input: n = 2

Output: 2

Explanation:

The first beautiful arrangement is [1,2]:

- perm[1] = 1 is divisible by i = 1
- perm[2] = 2 is divisible by i = 2

The second beautiful arrangement is [2,1]:

- perm[1] = 2 is divisible by i = 1
- i = 2 is divisible by perm[2] = 1

Input: n = 1

Output: 1

18.22 Min Cost Climbing Stairs

You are given an integer array cost where cost[i] is the cost of i^{th} step on a staircase. Once you pay the cost, you can either climb one or two steps.

You can either start from the step with index 0, or the step with index 1.

Return the minimum cost to reach the top of the floor.

Input: cost = [10, 15, 20]

Output: 15

Explanation: You will start at index 1.

- Pay 15 and climb two steps to reach the top.

The total cost is 15.

Input: cost = [1, 100, 1, 1, 1, 100, 1, 1, 100, 1]

Output: 6

Explanation: You will start at index 0.

- Pay 1 and climb two steps to reach index 2.
- Pay 1 and climb two steps to reach index 4.
- Pay 1 and climb two steps to reach index 6.
- Pay 1 and climb one step to reach index 7.
- Pay 1 and climb two steps to reach index 9.
- Pay 1 and climb one step to reach the top.

The total cost is 6.

18.23 Partition Array for Maximum Sum

Given an integer array arr, partition the array into (contiguous) subarrays of length **at most** k. After partitioning, each subarray has their values changed to become the maximum value of that subarray.

Return the largest sum of the given array after partitioning. Test cases are generated so that the answer fits in a **32-bit** integer.

Input: arr = [1, 15, 7, 9, 2, 5, 10], k = 3

Output: 84

Explanation: arr becomes [15, 15, 15, 9, 10, 10, 10]

Example 2:

Input: arr = [1,4,1,5,7,3,6,1,9,9,3], k = 4

Output: 83

Example 3:

Input: arr = [1], k = 1

Output: 1

18.24 Airplane Seat Assignment Probability

n passengers board an airplane with exactly n seats. The first passenger has lost the ticket and picks a seat randomly. But after that, the rest of the passengers will:

- Take their own seat if it is still available, and
- Pick other seats randomly when they find their seat occupied

Return the probability that the n^{th} person gets his own seat.

Input: n = 1

Output: 1.00000

Explanation: The first person can only get the first seat.

Example 2:

Input: n = 2

Output: 0.50000

Explanation: The second person has a probability of 0.5 to get the second seat (when first person gets the first seat).

18.25 Reducing Dishes

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.

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.

Input: satisfaction = [4, 3, 2]

Output: 20

Explanation: Dishes can be prepared in any order, (2*1 + 3*2 + 4*3 = 20)

Input: satisfaction = [-1,-4,-5]

Output: 0

Explanation: People do not like the dishes. No dish is prepared.

19. Backtracking Approach

19.1 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.

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"]

19.2 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.

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.

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

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

Input: candidates = [2], target = 1

Output: []

19.3 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.

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

Output:

[

[1, 1, 6],

[1, 2, 5],

[1, 7],

[2, 6]

]

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

Output:

[

[1, 2, 2],

[5]

]

19.4 Permutations

Given an array nums of distinct integers, return all the possible permutations. You can return the answer in **any order**.

Input: nums = [1, 2, 3]

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

Input: nums = [0, 1]

Output: [[0, 1], [1, 0]]

Input: nums = [1]

Output: [[1]]

19.5 Subsets

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**.

Input: nums = [1, 2, 3]

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

Input: nums = [0]

Output: [[], [0]]

19.6 Gray Code

An **n-bit gray code sequence** is a sequence of 2^n integers where:

- Every integer is in the **inclusive** range [0, 2^n - 1],
- The first integer is 0,
- An integer appears **no more than once** in the sequence,
- The binary representation of every pair of **adjacent** integers differs by **exactly one bit**, and
- The binary representation of the **first** and **last** integers differs by **exactly one bit**.

Given an integer n, return any valid **n-bit gray code sequence**.

Input: n = 2

Output: [0, 1, 3, 2]

Explanation:

The binary representation of [0, 1, 3, 2] is [00, 01, 11, 10].

- 0<u>0</u> and 0<u>1</u> differ by one bit

- <u>0</u>1 and <u>1</u>1 differ by one bit

- 1<u>1</u> and 1<u>0</u> differ by one bit

- <u>1</u>0 and <u>0</u>0 differ by one bit

[0, 2, 3, 1] is also a valid gray code sequence, whose binary representation is [00, 10, 11, 01].

- <u>0</u>0 and <u>1</u>0 differ by one bit

- 1<u>0</u> and 1<u>1</u> differ by one bit

- <u>1</u>1 and <u>0</u>1 differ by one bit

- 0<u>1</u> and 0<u>0</u> differ by one bit

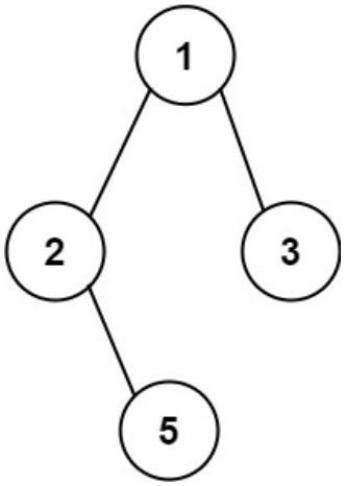
Input: n = 1

Output: [0, 1]

19.7 Binary Tree Paths

Given the root of a binary tree, return all root-to-leaf paths in **any order**.

A **leaf** is a node with no children.



Input: root = [1, 2, 3, null, 5]

Output: ["1->2->5","1->3"]

Input: root = [1]

Output: ["1"]

19.8 Binary Watch

A binary watch has 4 LEDs on the top to represent the hours (0-11), and 6 LEDs on the bottom to represent the minutes (0-59). Each LED represents a zero or one, with the least significant bit on the right.

- For example, the below binary watch reads "4:51".

Given an integer turnedOn which represents the number of LEDs that are currently on (ignoring the PM), return all possible times the watch could represent. You may return the answer in **any order**.

The hour must not contain a leading zero.

- For example, "01:00" is not valid. It should be "1:00".

The minute must consist of two digits and may contain a leading zero.

- For example, "10:2" is not valid. It should be "10:02".

Input: turnedOn = 1

Output: ["0:01","0:02","0:04","0:08","0:16","0:32","1:00","2:00","4:00","8:00"]

Input: turnedOn = 9

Output: []

19.9 Additive Number

An **additive number** is a string whose digits can form an **additive sequence**.

A valid **additive sequence** should contain **at least** three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the preceding two.

Given a string containing only digits, return true if it is an **additive number** or false otherwise.

Note: Numbers in the additive sequence **cannot** have leading zeros, so sequence 1, 2, 03 or 1, 02, 3 is invalid.

Input: "112358"

Output: true

Explanation:

The digits can form an additive sequence: 1, 1, 2, 3, 5, 8.

1 + 1 = 2, 1 + 2 = 3, 2 + 3 = 5, 3 + 5 = 8

Input: "199100199"

Output: true

Explanation:

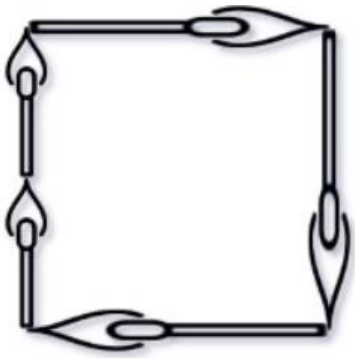
The additive sequence is: 1, 99, 100, 199.

1 + 99 = 100, 99 + 100 = 199

19.10 Matchsticks to Square

You are given an integer array matchsticks where matchsticks[i] is the length of the i^{th} matchstick. You want to use **all the matchsticks** to make one square. You **should not break** any stick, but you can link them up, and each matchstick must be used **exactly one time**.

Return true if you can make this square and false otherwise.



Input: matchsticks = [1, 1, 2, 2, 2]

Output: true

Explanation: You can form a square with length 2, one side of the square came two sticks with length 1.

Input: matchsticks = [3, 3, 3, 3, 4]

Output: false

Explanation: You cannot find a way to form a square with all the matchsticks.

19.11 Path with Maximum Gold

In a gold mine grid of size m x n, each cell in this mine has an integer representing the amount of gold in that cell, 0 if it is empty.

Return the maximum amount of gold you can collect under the conditions:

- Every time you are located in a cell you will collect all the gold in that cell.
- From your position, you can walk one step to the left, right, up, or down.
- You can't visit the same cell more than once.
- Never visit a cell with 0 gold.
- You can start and stop collecting gold from **any** position in the grid that has some gold.

Input: grid = [[0, 6, 0], [5, 8, 7], [0, 9, 0]]

Output: 24

Explanation:

[[0, 6, 0],

[5, 8, 7],

[0, 9, 0]]

Path to get the maximum gold, 9 -> 8 -> 7.

Input: grid = [[1, 0, 7], [2, 0, 6], [3, 4, 5], [0, 3, 0], [9, 0, 20]]

Output: 28

Explanation:

[[1, 0, 7],

[2, 0, 6],

[3, 4, 5],

[0, 3, 0],

[9, 0, 20]]

Path to get the maximum gold, 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7.

19.12 Fair Distribution of Cookies

You are given an integer array cookies, where cookies[i] denotes the number of cookies in the i^{th} bag. You are also given an integer k that denotes the number of children to distribute **all** the bags of cookies to. All the cookies in the same bag must go to the same child and cannot be split up.

The **unfairness** of a distribution is defined as the **maximum total** cookies obtained by a single child in the distribution.

Return the **minimum** unfairness of all distributions.

Input: cookies = [8, 15, 10, 20, 8], k = 2

Output: 31

Explanation: One optimal distribution is [8, 15, 8] and [10, 20]

- The 1^{st} child receives [8, 15, 8] which has a total of 8 + 15 + 8 = 31 cookies.

- The 2^{nd} child receives [10, 20] which has a total of 10 + 20 = 30 cookies.

The unfairness of the distribution is max(31,30) = 31.

It can be shown that there is no distribution with an unfairness less than 31.

Input: cookies = [6, 1, 3, 2, 2, 4, 1, 2], k = 3

Output: 7

Explanation: One optimal distribution is [6, 1], [3, 2, 2], and [4, 1, 2]

- The 1^{st} child receives [6, 1] which has a total of 6 + 1 = 7 cookies.

- The 2^{nd} child receives [3, 2, 2] which has a total of 3 + 2 + 2 = 7 cookies.

- The 3^{rd} child receives [4, 1, 2] which has a total of 4 + 1 + 2 = 7 cookies.

The unfairness of the distribution is max(7,7,7) = 7.

It can be shown that there is no distribution with an unfairness less than 7.

19.13 Split a String into the Max Number of Unique Substrings

Given a string s, return the maximum number of unique substrings that the given string can be split into.

You can split string s into any list of **non-empty substrings**, where the concatenation of the substrings forms the original string. However, you must split the substrings such that all of them are **unique**.

A **substring** is a contiguous sequence of characters within a string.

Input: s = "ababccc"

Output: 5

Explanation: One way to split maximally is ['a', 'b', 'ab', 'c', 'cc']. Splitting like ['a', 'b', 'a', 'b', 'c', 'cc'] is not valid as you have 'a' and 'b' multiple times.

Input: s = "aba"

Output: 2

Explanation: One way to split maximally is ['a', 'ba'].

Input: s = "aa"

Output: 1

Explanation: It is impossible to split the string any further.

19.14 Sum of All Subset XOR Totals

The **XOR total** of an array is defined as the bitwise XOR of **all its elements**, or 0 if the array is **empty**.

- For example, the **XOR total** of the array [2,5,6] is 2 XOR 5 XOR 6 = 1.

Given an array nums, return the **sum** of all **XOR totals** for every **subset** of nums.

Note: Subsets with the **same** elements should be counted **multiple** times.

An array a is a **subset** of an array b if a can be obtained from b by deleting some (possibly zero) elements of b.

Input: nums = [1, 3]

Output: 6

Explanation: The 4 subsets of [1, 3] are:

- The empty subset has an XOR total of 0.

- [1] has an XOR total of 1.

- [3] has an XOR total of 3.

- [1, 3] has an XOR total of 1 XOR 3 = 2.

0 + 1 + 3 + 2 = 6

Input: nums = [5, 1, 6]

Output: 28

Explanation: The 8 subsets of [5, 1, 6] are:

- The empty subset has an XOR total of 0.

- [5] has an XOR total of 5.

- [1] has an XOR total of 1.

- [6] has an XOR total of 6.

- [5, 1] has an XOR total of 5 XOR 1 = 4.

- [5, 6] has an XOR total of 5 XOR 6 = 3.

- [1, 6] has an XOR total of 1 XOR 6 = 7.

- [5, 1, 6] has an XOR total of 5 XOR 1 XOR 6 = 2.

0 + 5 + 1 + 6 + 4 + 3 + 7 + 2 = 28

Input: nums = [3, 4, 5, 6, 7, 8]

Output: 480

Explanation: The sum of all XOR totals for every subset is 480.

19.15 Find the Punishment Number of an Integer

Given a positive integer n, return the **punishment number** of n.

The **punishment number** of n is defined as the sum of the squares of all integers i such that:

- 1 <= i <= n
- The decimal representation of i * i can be partitioned into contiguous substrings such that the sum of the integer values of these substrings equals i.

Input: n = 10

Output: 182

Explanation: There are exactly 3 integers i that satisfy the conditions in the statement:

- 1 since 1 * 1 = 1

- 9 since 9 * 9 = 81 and 81 can be partitioned into 8 + 1.

- 10 since 10 * 10 = 100 and 100 can be partitioned into 10 + 0.

Hence, the punishment number of 10 is 1 + 81 + 100 = 182

Input: n = 37

Output: 1478

Explanation: There are exactly 4 integers i that satisfy the conditions in the statement:

- 1 since 1 * 1 = 1.

- 9 since 9 * 9 = 81 and 81 can be partitioned into 8 + 1.

- 10 since 10 * 10 = 100 and 100 can be partitioned into 10 + 0.

- 36 since 36 * 36 = 1296 and 1296 can be partitioned into 1 + 29 + 6.

Hence, the punishment number of 37 is 1 + 81 + 100 + 1296 = 1478

19.16 Maximum Split of Positive Even Integers

You are given an integer finalSum. Split it into a sum of a **maximum** number of **unique** positive even integers.

- For example, given finalSum = 12, the following splits are **valid** (unique positive even integers summing up to finalSum): (12), (2 + 10), (2 + 4 + 6), and (4 + 8). Among them, (2 + 4 + 6) contains the maximum number of integers. Note that finalSum cannot be split into (2 + 2 + 4 + 4) as all the numbers should be unique.

Return a list of integers that represent a valid split containing a **maximum** number of integers. If no valid split exists for finalSum, return an **empty** list. You may return the integers in **any** order.

Input: finalSum = 12

Output: [2, 4, 6]

Explanation: The following are valid splits: (12), (2 + 10), (2 + 4 + 6), and (4 + 8).

(2 + 4 + 6) has the maximum number of integers, which is 3. Thus, we return [2, 4, 6].

Note that [2, 6, 4], [6, 2, 4], etc. are also accepted.

Input: finalSum = 7

Output: []

Explanation: There are no valid splits for the given finalSum.

Thus, we return an empty array.

Input: finalSum = 28

Output: [6, 8, 2, 12]

Explanation: The following are valid splits: (2 + 26), (6 + 8 + 2 + 12), and (4 + 24).

(6 + 8 + 2 + 12) has the maximum number of integers, which is 4. Thus, we return [6, 8, 2,12].

Note that [10, 2, 4, 12], [6, 2, 4, 16], etc. are also accepted.

19.17 Numbers with Same Consecutive Differences

Given two integers' n and k, return an array of all the integers of length n where the difference between every two consecutive digits is k. You may return the answer in **any order**.

Note that the integers should not have leading zeros. Integers as 02 and 043 are not allowed.

Input: n = 3, k = 7

Output: [181, 292, 707, 818, 929]

Explanation: Note that 070 is not a valid number, because it has leading zeroes.

Input: n = 2, k = 1

Output: [10, 12, 21, 23, 32, 34, 43, 45, 54, 56, 65, 67, 76, 78, 87, 89, 98]

19.18 Letter Tile Possibilities

You have n tiles, where each tile has one letter tiles[i] printed on it.

Return the number of possible non-empty sequences of letters you can make using the letters printed on those tiles.

Input: tiles = "AAB"

Output: 8

Explanation: The possible sequences are "A", "B", "AA", "AB", "BA", "AAB", "ABA", "BAA".

Input: tiles = "AAABBC"

Output: 188

Input: tiles = "V"

Output: 1

19.19 What Does It Mean?

Bob has always been interested in his family history, and above all else his family name's meaning. Unfortunately for Bob, no one else in his family has ever had any similar interest whatsoever. Because of this the family name seems to have changed at random points in time without any reason that Bob can find.

You have a dictionary of words that may make up part of Bob's family name, and one or more unique meanings associated with each word. By concatenating one or more of these dictionary words to construct exactly the family name, count the number of different meanings associated with these different constructions.

Input: The first line contains an integer N and a word W, indicating the number of words in the dictionary and the family name respectively. The following N lines contain the dictionary. Each dictionary line starts with the dictionary word followed by an integer representing the number of meanings of the word.

Output: Output the number of possible meanings Bob's family name can have. As this number can be very large, output it modulo 1000000007.

Input:

5 heimark

hei 2

mark 2

heim 1

ark 2

heima 1

Output: 6

19.20 Non-decreasing Subsequences

Given an integer array nums, return all the different possible non-decreasing subsequences of the given array with at least two elements. You may return the answer in **any order**.

Input: nums = [4, 6, 7, 7]

Output: [[4, 6], [4, 6, 7], [4, 6, 7, 7], [4, 7], [4, 7, 7], [6, 7], [6, 7, 7], [7, 7]]

Input: nums = [4, 4, 3, 2, 1]

Output: [[4, 4]]

20. 0/1 Knapsack

20.1 Target Sum

You are given an integer array nums and an integer target.

You want to build an **expression** out of nums by adding one of the symbols '+' and '-' before each integer in nums and then concatenate all the integers.

- For example, if nums = [2, 1], you can add a '+' before 2 and a '-' before 1 and concatenate them to build the expression "+2-1".

Return the number of different **expressions** that you can build, which evaluates to target.

Input: nums = [1, 1, 1, 1, 1], target = 3

Output: 5

Explanation: There are 5 ways to assign symbols to make the sum of nums be target 3.

-1 + 1 + 1 + 1 + 1 = 3

+1 - 1 + 1 + 1 + 1 = 3

+1 + 1 - 1 + 1 + 1 = 3

+1 + 1 + 1 - 1 + 1 = 3

+1 + 1 + 1 + 1 - 1 = 3

Input: nums = [1], target = 1

Output: 1

20.2 Painting the Walls

You are given two **0-indexed** integer arrays, cost and time, of size n representing the costs and the time taken to paint n different walls respectively. There are two painters available:

- A **paid painter** that paints the i^{th} wall in time[i] units of time and takes cost[i] units of money.
- A **free painter** that paints **any** wall in 1 unit of time at a cost of 0. But the free painter can only be used if the paid painter is already **occupied**.

Return the minimum amount of money required to paint the n walls.

Input: cost = [1, 2, 3, 2], time = [1, 2, 3, 2]

Output: 3

Explanation: The walls at index 0 and 1 will be painted by the paid painter, and it will take 3 units of time; meanwhile, the free painter will paint the walls at index 2 and 3, free of cost in 2 units of time. Thus, the total cost is 1 + 2 = 3.

Input: cost = [2, 3, 4, 2], time = [1, 1, 1, 1]

Output: 4

Explanation: The walls at index 0 and 3 will be painted by the paid painter, and it will take 2 units of time; meanwhile, the free painter will paint the walls at index 1 and 2, free of cost in 2 units of time. Thus, the total cost is 2 + 2 = 4.

20.3 Partition Equal Subset Sum

Given an integer array nums, return true if you can partition the array into two subsets such that the sum of the elements in both subsets is equal or false otherwise.

Input: nums = [1, 5, 11, 5]

Output: true

Explanation: The array can be partitioned as [1, 5, 5] and [11].

Input: nums = [1, 2, 3, 5]

Output: false

Explanation: The array cannot be partitioned into equal sum subsets.

20.4 Ones and Zeroes

You are given an array of binary strings strs and two integers' m and n.

Return the size of the largest subset of strs such that there are **at most** m 0's and n 1's in the subset.

A set x is a **subset** of a set y if all elements of x are also elements of y.

Input: strs = ["10","0001","111001","1","0"], m = 5, n = 3

Output: 4

Explanation: The largest subset with at most 5 0's and 3 1's is {"10", "0001", "1", "0"}, so the answer is 4.

Other valid but smaller subsets include {"0001", "1"} and {"10", "1", "0"}.

{"111001"} is an invalid subset because it contains 4 1's, greater than the maximum of 3.

Input: strs = ["10","0","1"], m = 1, n = 1

Output: 2

Explanation: The largest subset is {"0", "1"}, so the answer is 2.

20.5 Number of Ways to Earn Points

There is a test that has n types of questions. You are given an integer target and a **0-indexed** 2D integer array types where types[i] = [$count_i$, $marks_i$] indicates that there are $count_i$ questions of the i^{th} type, and each one of them is worth $marks_i$ points.

Return the number of ways you can earn **exactly** target points in the exam. Since the answer may be too large, return it **modulo** $10^9 + 7$.

Note that questions of the same type are indistinguishable.

- For example, if there are 3 questions of the same type, then solving the 1^{st} and 2^{nd} questions is the same as solving the 1^{st} and 3^{rd} questions, or the 2^{nd} and 3^{rd} questions.

Input: target = 6, types = [[6, 1], [3, 2], [2, 3]]

Output: 7

Explanation: You can earn 6 points in one of the seven ways:

- Solve 6 questions of the 0^{th} type: 1 + 1 + 1 + 1 + 1 + 1 = 6
- Solve 4 questions of the 0^{th} type and 1 question of the 1^{st} type: 1 + 1 + 1 + 1 + 2 = 6
- Solve 2 questions of the 0^{th} type and 2 questions of the 1^{st} type: 1 + 1 + 2 + 2 = 6
- Solve 3 questions of the 0^{th} type and 1 question of the 2^{nd} type: 1 + 1 + 1 + 3 = 6
- Solve 1 question of the 0^{th} type, 1 question of the 1^{st} type and 1 question of the 2^{nd} type: 1 + 2 + 3 = 6
- Solve 3 questions of the 1^{st} type: 2 + 2 + 2 = 6
- Solve 2 questions of the 2^{nd} type: 3 + 3 = 6

Input: target = 5, types = [[50, 1], [50, 2], [50, 5]]

Output: 4

Explanation: You can earn 5 points in one of the four ways:

- Solve 5 questions of the 0^{th} type: 1 + 1 + 1 + 1 + 1 = 5
- Solve 3 questions of the 0^{th} type and 1 question of the 1^{st} type: 1 + 1 + 1 + 2 = 5
- Solve 1 questions of the 0^{th} type and 2 questions of the 1^{st} type: 1 + 2 + 2 = 5
- Solve 1 question of the 2^{nd} type: 5

Input: target = 18, types = [[6, 1], [3, 2], [2, 3]]

Output: 1

Explanation: You can only earn 18 points by answering all questions.

21. Graph Representation

21.1 Build a Graph

You are given an integer n. Determine if there is an unconnected graph with n vertices that contains at least two connected components and contains the number of edges that is equal to the number of vertices. Each vertex must follow one of these conditions:

- Its degree is less than or equal to 1.
- It's a cut-vertex.

Note:

- The graph must be simple.
- Loops and multiple edges are not allowed.

Input: First line: n

Output: Print Yes if it is an unconnected graph. Otherwise, print No.

Sample Input	Sample Output
3	No

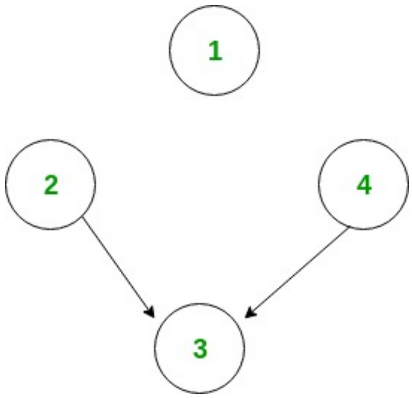
Constraints: 1 ≤ n ≤ 100

Explanation: There is only one graph with the number of edges equal to the number of vertices (triangle) which is connected.

21.2 Number of Sink Nodes in a Graph

Given a Directed Acyclic Graph of n nodes (numbered from 1 to n) and m edges. The task is to find the number of sink nodes. A sink node is a node such that no edge emerges out of it.

Input: n = 4, m = 2, edges[] = {{2, 3}, {4, 3}}



Only node 1 and node 3 are sink nodes.

Input: n = 4, m = 2, edges[] = {{3, 2}, {3, 4}}

Output: 3

The idea is to iterate through all the edges. And for each edge, mark the source node from which the edge emerged out. Now, for each node check if it is marked or not. And count the unmarked nodes.

Algorithm:

1. Make any array A[] of size equal to the number of nodes and initialize to 1.

2. Traverse all the edges one by one, say, u -> v.

   (i) Mark A[u] as 1.

3. Now traverse whole array A[] and count number of unmarked nodes.

21.3 Connected Components in a Graph

Given n, i.e. total number of nodes in an undirected graph numbered from 1 to n and an integer e, i.e. total number of edges in the graph. Calculate the total number of connected components in the graph. A connected component is a set of vertices in a graph that are linked to each other by paths.

Input: First line of input line contains two integers' n and e. Next e line will contain two integers u and v meaning that node u and node v are connected to each other in undirected fashion.

Output: For each input graph print an integer x denoting total number of connected components.

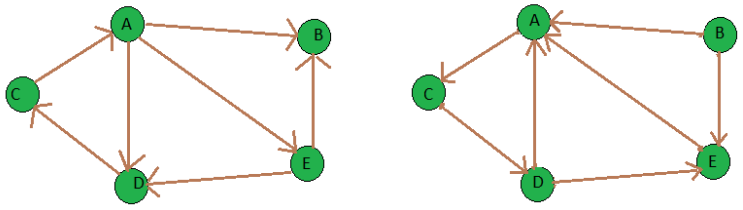
Sample Input	Sample Output
8 5	3
1 2	
2 3	

2 4 3 5 6 7	

Constraints: All the input values are well within the integer range.

21.4 Transpose Graph

Transpose of a directed graph G is another directed graph on the same set of vertices with all of the edges reversed compared to the orientation of the corresponding edges in G. That is, if G contains an edge (u, v) then the converse/transpose/reverse of G contains an edge (v, u) and vice versa. Given a graph (represented as adjacency list), we need to find another graph which is the transpose of the given graph.



Input: figure (i) is the input graph.

Output: figure (ii) is the transpose graph of the given graph.
0--> 2
1--> 0 4
2--> 3
3--> 0 4
4--> 0

Explanation: We traverse the adjacency list and as we find a vertex v in the adjacency list of vertex u which indicates an edge from u to v in main graph, we just add an edge from v to u in the transpose graph i.e. add u in the adjacency list of vertex v of the new graph. Thus traversing lists of all vertices of main graph we can get the transpose graph.

21.5 Counting Triplets

You are given an undirected, complete graph G that contains N vertices. Each edge is colored in either white or black. You are required to determine the number of triplets (i, j, k) ($1 \le i < j < k \le N$) of vertices such that the edges (i, j), (j, k), (i, k) are of the same color.

There are M white edges and (N (N-1)/2) – M black edges.

Input:

First line: Two integers – N and M ($3 \le N \le 10^5$, $1 \le M \le 3 * 10^5$)

$(i+1)^{th}$ line: Two integers – u_i and v_i ($1 \le u_i, v_i \le N$) denoting that the edge (u_i, v_i) is white in color.

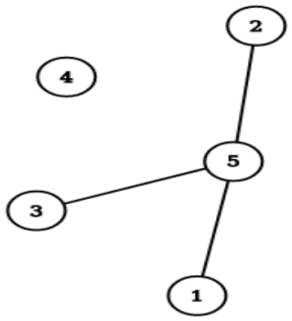
Note: The conditions $(u_i, v_i) \ne (u_j, v_j)$ and $(u_i, v_i) \ne (v_j, u_j)$ are satisfied for all $1 \le i < j \le M$.

.**Output:** Print an integer that denotes the number of triples that satisfy the mentioned condition.

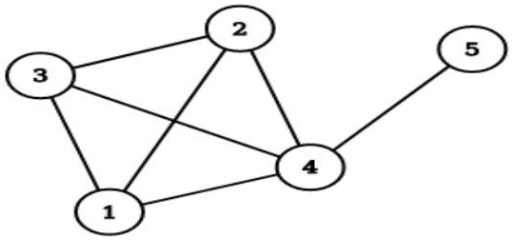
Sample Input	Sample Output
5 3	4
1 5	
2 5	
3 5	

Explanation: The triplets are: {(1, 2, 3), (1, 2, 4), (2, 3, 4), (1, 3, 4)}

The graph consisting of only white edges:

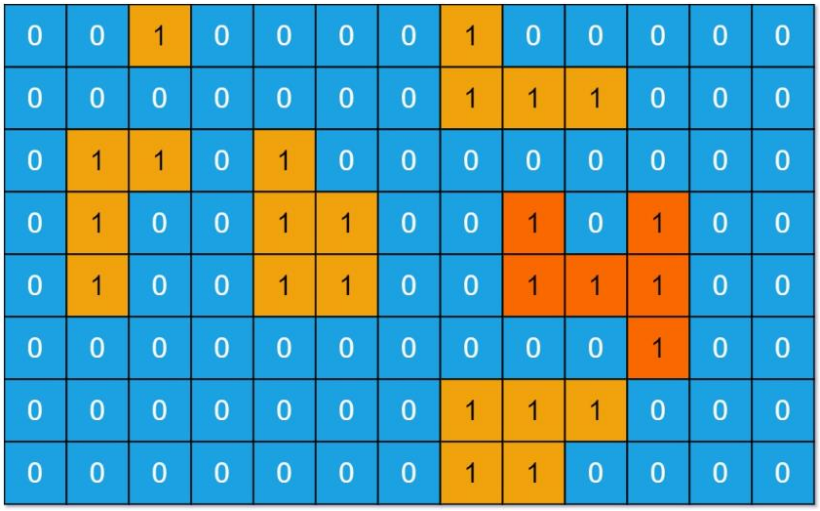


The graph consisting of only black edges:



21.6 Max Area of Island

You are given an m x n binary matrix grid. An island is a group of 1's (representing land) connected 4-directionally (horizontal or vertical.) You may assume all four edges of the grid are surrounded by water. The area of an island is the number of cells with a value 1 in the island. Return the maximum area of an island in grid. If there is no island, return 0.



Input: grid = [[0,0,1,0,0,0,0,1,0,0,0,0,0],
[0,0,0,0,0,0,0,1,1,1,0,0,0],
[0,1,1,0,1,0,0,0,0,0,0,0,0],
[0,1,0,0,1,1,0,0,1,0,1,0,0],
[0,1,0,0,1,1,0,0,1,1,1,0,0],
[0,0,0,0,0,0,0,0,0,0,1,0,0],
[0,0,0,0,0,0,0,1,1,1,0,0,0],
[0,0,0,0,0,0,0,1,1,0,0,0,0]]

Output: 6

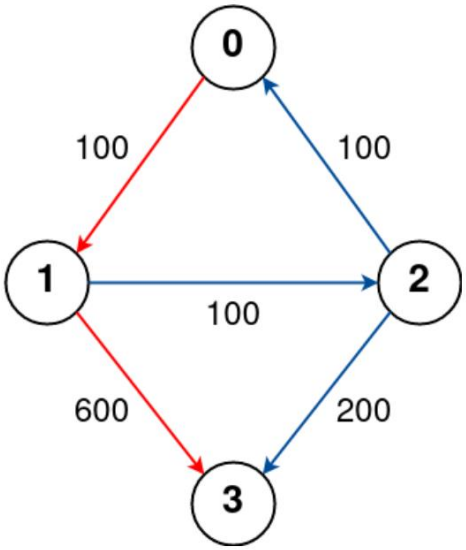
Explanation: The answer is not 11, because the island must be connected 4-directionally.

Input: grid = [[0, 0, 0, 0, 0, 0, 0, 0]]

Output: 0

21.7 Cheapest Flights within K Stops

There are n cities connected by some number of flights. You are given an array flights where flights[i] = [fromi, toi, pricei] indicates that there is a flight from city fromi to city toi with cost pricei. You are also given three integers src, dst, and k, return the cheapest price from src to dst with at most k stops. If there is no such route, return -1.

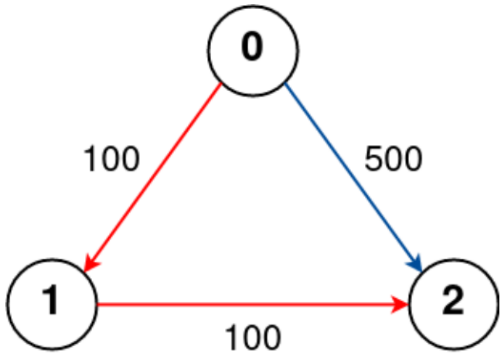


Input: n = 4, flights = [[0,1,100],[1,2,100],[2,0,100],[1,3,600],[2,3,200]], src = 0, dst = 3, k = 1

Output: 700

Explanation: The graph is shown above. The optimal path with at most 1 stop from city 0 to 3 is marked in red and has cost 100 + 600 = 700.

Note that the path through cities [0, 1, 2, 3] is cheaper but is invalid because it uses 2 stops.



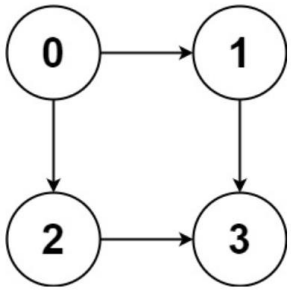
Input: n = 3, flights = [[0, 1, 100], [1, 2, 100], [0, 2, 500]], src = 0, dst = 2, k = 1

Output: 200

Explanation: The graph is shown above. The optimal path with at most 1 stop from city 0 to 2 is marked in red and has cost 100 + 100 = 200.

21.8 All Paths from Source to Target

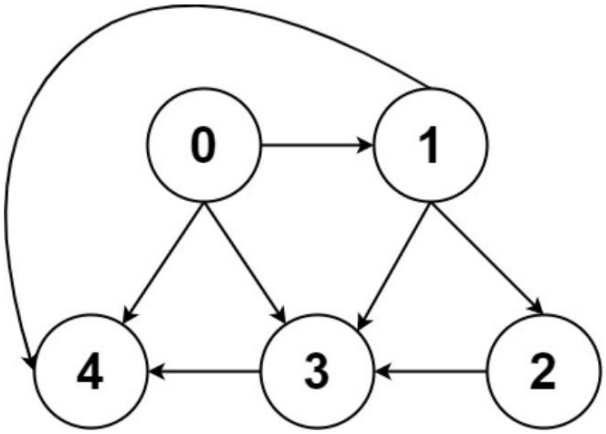
Given a directed acyclic graph (DAG) of n nodes labeled from 0 to n - 1, find all possible paths from node 0 to node n - 1 and return them in any order. The graph is given as follows: graph[i] is a list of all nodes you can visit from node i (i.e., there is a directed edge from node i to node graph[i][j]).



Input: graph = [[1, 2], [3], [3], []]

Output: [[0, 1, 3], [0, 2, 3]]

Explanation: There are two paths: 0 -> 1 -> 3 and 0 -> 2 -> 3.



Input: graph = [[4, 3, 1], [3, 2, 4], [3], [4], []]

Output: [[0, 4], [0, 3, 4], [0, 1, 3, 4], [0, 1, 2, 3, 4], [0, 1, 4]]

21.9 Shortest Path with Alternating Colors

You are given an integer n, the number of nodes in a directed graph where the nodes are labeled from 0 to n - 1. Each edge is red or blue in this graph, and there could be self-edges and parallel edges.

You are given two arrays redEdges and blueEdges where:

- redEdges[i] = [ai, bi] indicates that there is a directed red edge from node ai to node bi in the graph, and
- blueEdges[j] = [uj, vj] indicates that there is a directed blue edge from node uj to node vj in the graph.

Return an array answer of length n, where each answer[x] is the length of the shortest path from node 0 to node x such that the edge colors alternate along the path, or -1 if such a path does not exist.

Input: n = 3, redEdges = [[0, 1], [1, 2]], blueEdges = [ ]

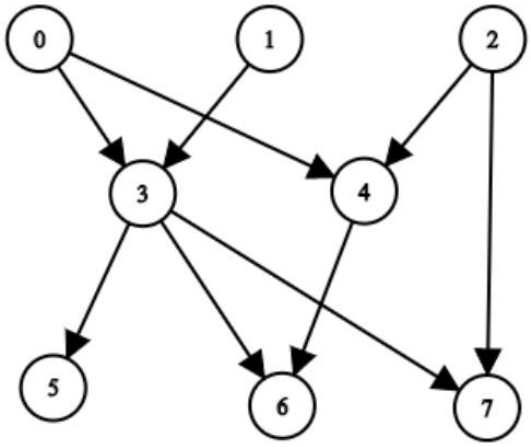
Output: [0, 1, -1]

Input: n = 3, redEdges = [[0, 1]], blueEdges = [[2, 1]]

Output: [0, 1, -1]

21.10 All Ancestors of a Node in a Directed Acyclic Graph

You are given a positive integer n representing the number of nodes of a Directed Acyclic Graph (DAG). The nodes are numbered from 0 to n - 1 (inclusive). You are also given a 2D integer array edges, where edges[i] = [$from_i$, to_i] denotes that there is a unidirectional edge from $from_i$ to to_i in the graph. Return *a list* answer, *where* answer[i] *is the list of ancestors of the* i^{th} *node, sorted in ascending order*. A node u is an ancestor of another node v if u can reach v via a set of edges.



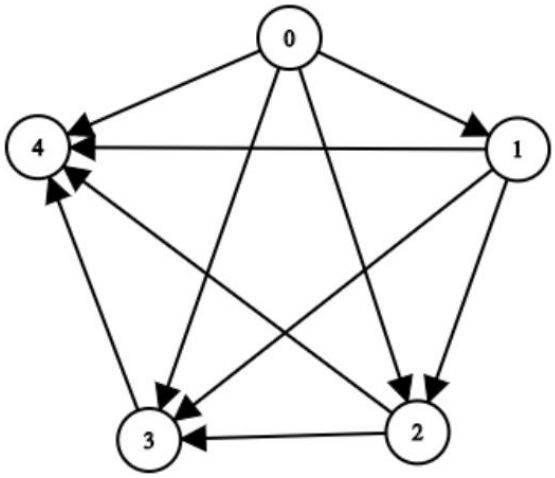
Input: n = 8, edgeList = [[0, 3], [0, 4], [1, 3], [2, 4], [2, 7], [3, 5], [3, 6], [3, 7], [4, 6]]

Output: [[], [], [], [0, 1], [0, 2], [0, 1, 3], [0, 1, 2, 3, 4], [0, 1, 2, 3]]

Explanation:

The above diagram represents the input graph.

- Nodes 0, 1, and 2 do not have any ancestors.
- Node 3 has two ancestors 0 and 1.
- Node 4 has two ancestors 0 and 2.
- Node 5 has three ancestors 0, 1, and 3.
- Node 6 has five ancestors 0, 1, 2, 3, and 4.
- Node 7 has four ancestors 0, 1, 2, and 3.



Input: n = 5, edgeList = [[0, 1], [0, 2], [0, 3], [0, 4], [1, 2], [1, 3], [1, 4], [2, 3], [2, 4], [3, 4]]

Output: [[], [0], [0, 1], [0, 1, 2], [0, 1, 2, 3]]

Explanation:

The above diagram represents the input graph.

- Node 0 does not have any ancestor.

- Node 1 has one ancestor 0.

- Node 2 has two ancestors 0 and 1.

- Node 3 has three ancestors 0, 1, and 2.

- Node 4 has four ancestors 0, 1, 2, and 3.

21.11 Reorder Routes to Make All Paths Lead to the City Zero

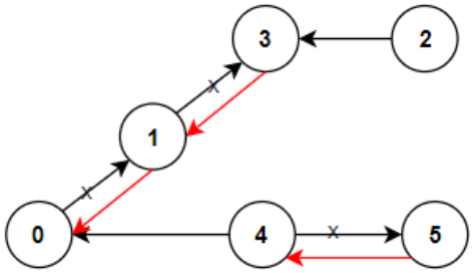
There are n cities numbered from 0 to n - 1 and n - 1 roads such that there is only one way to travel between two different cities (this network form a tree). Last year, the ministry of transport decided to orient the roads in one direction because they are too narrow.

Roads are represented by connections where connections[i] = [a_i, b_i] represents a road from city a_i to city b_i.

This year, there will be a big event in the capital (city 0), and many people want to travel to this city.

Your task consists of reorienting some roads such that each city can visit the city 0. Return the **minimum** number of edges changed.

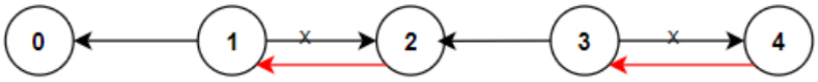
It's **guaranteed** that each city can reach city 0 after reorder.



Input: n = 6, connections = [[0, 1], [1, 3], [2, 3], [4, 0], [4, 5]]

Output: 3

Explanation: Change the direction of edges show in red such that each node can reach the node 0 (capital).



Input: n = 5, connections = [[1, 0], [1, 2], [3, 2], [3, 4]]

Output: 2

Explanation: Change the direction of edges show in red such that each node can reach the node 0 (capital).

Input: n = 3, connections = [[1, 0], [2, 0]]

Output: 0

21.12 Map of Highest Peak

You are given an integer matrix isWater of size m x n that represents a map of **land** and **water** cells.

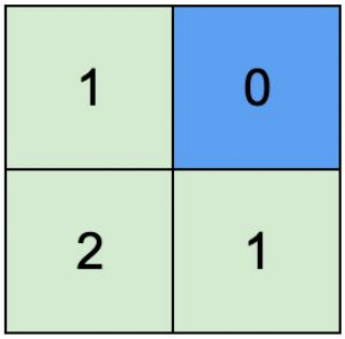
- If isWater[i][j] == 0, cell (i, j) is a **land** cell.
- If isWater[i][j] == 1, cell (i, j) is a **water** cell.

You must assign each cell a height in a way that follows these rules:

- The height of each cell must be non-negative.
- If the cell is a **water** cell, its height must be 0.
- Any two adjacent cells must have an absolute height difference of **at most** 1. A cell is adjacent to another cell if the former is directly north, east, south, or west of the latter (i.e., their sides are touching).

Find an assignment of heights such that the maximum height in the matrix is **maximized**.

Return an integer matrix height of size m x n where height[i][j] is cell (i, j)'s height. If there are multiple solutions, return **any** of them.

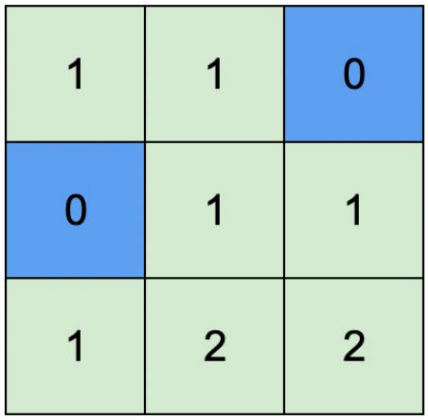


Input: isWater = [[0, 1], [0, 0]]

Output: [[1, 0], [2, 1]]

Explanation: The image shows the assigned heights of each cell.

The blue cell is the water cell, and the green cells are the land cells.



Input: isWater = [[0, 0, 1], [1, 0, 0], [0, 0, 0]]

Output: [[1, 1, 0], [0, 1, 1], [1, 2, 2]]

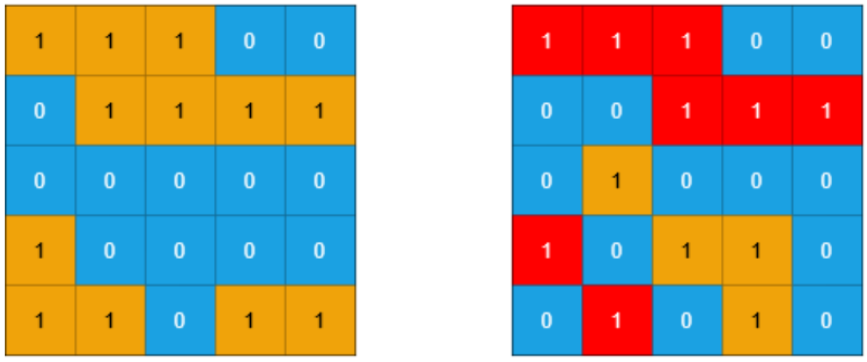
Explanation: A height of 2 is the maximum possible height of any assignment. Any height assignment that has a maximum height of 2 while still meeting the rules will also be accepted.

21.13 Count Sub Islands

You are given two m x n binary matrices grid1 and grid2 containing only 0's (representing water) and 1's (representing land). An **island** is a group of 1's connected **4-directionally** (horizontal or vertical). Any cells outside of the grid are considered water cells.

An island in grid2 is considered a **sub-island** if there is an island in grid1 that contains **all** the cells that make up **this** island in grid2.

Return the **number** of islands in grid2 that are considered **sub-islands**.



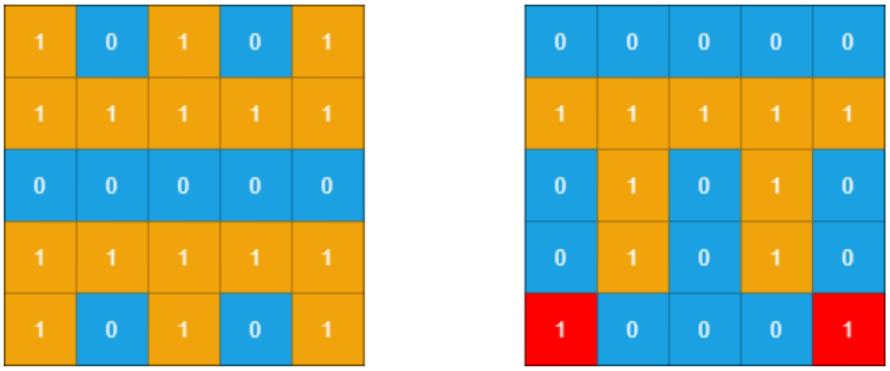
Input: grid1= [[1, 1, 1, 0, 0], [0, 1, 1, 1, 1], [0, 0, 0, 0, 0], [1, 0, 0, 0, 0], [1, 1, 0, 1, 1]],

grid2 = [[1, 1, 1, 0, 0], [0, 0, 1, 1, 1], [0, 1, 0, 0, 0], [1, 0, 1, 1, 0], [0, 1, 0, 1, 0]]

Output: 3

Explanation: In the picture above, the grid on the left is grid1 and the grid on the right is grid2.

The 1s colored red in grid2 are those considered to be part of a sub-island. There are three sub-islands.



Input: grid1 = [[1, 0, 1, 0, 1], [1, 1, 1, 1, 1], [0, 0, 0, 0, 0], [1, 1, 1, 1, 1], [1, 0, 1, 0, 1]],

grid2 = [[0, 0, 0, 0, 0], [1, 1, 1, 1, 1], [0, 1, 0, 1, 0], [0, 1, 0, 1, 0], [1, 0, 0, 0, 1]]

Output: 2

Explanation: In the picture above, the grid on the left is grid1 and the grid on the right is grid2.

The 1s colored red in grid2 are those considered to be part of a sub-island. There are two sub-islands.

21.14 Find the Town Judge

In a town, there are n people labeled from 1 to n. There is a rumor that one of these people is secretly the town judge.

If the town judge exists, then:

1. The town judge trusts nobody.

2. Everybody (except for the town judge) trusts the town judge.
3. There is exactly one person that satisfies properties **1** and **2**.

You are given an array trust where trust[i] = [a_i, b_i] representing that the person labeled a_i trusts the person labeled b_i. If a trust relationship does not exist in trust array, then such a trust relationship does not exist.

Return the label of the town judge if the town judge exists and can be identified, or return -1 otherwise.

Input: n = 2, trust = [[1, 2]]

Output: 2

Input: n = 3, trust = [[1, 3], [2, 3]]

Output: 3

Input: n = 3, trust = [[1, 3], [2, 3], [3, 1]]

Output: -1

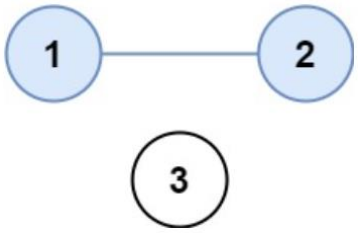
21.15 Number of Provinces

There are n cities. Some of them are connected, while some are not. If city a is connected directly with city b, and city b is connected directly with city c, then city a is connected indirectly with city c.

A **province** is a group of directly or indirectly connected cities and no other cities outside of the group.

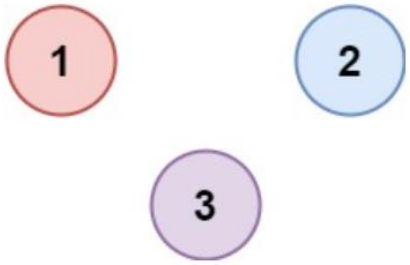
You are given an n x n matrix isConnected where isConnected[i][j] = 1 if the i^{th} city and the j^{th} city are directly connected, and isConnected[i][j] = 0 otherwise. Return the total number of **provinces**.

Input: isConnected = [[1, 1, 0], [1, 1, 0], [0, 0, 1]]



Output: 2

Input: isConnected = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]



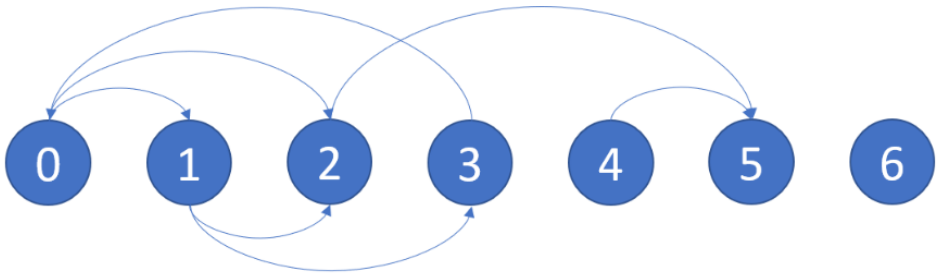
Output: 3

21.16 Find Eventual Safe States

There is a directed graph of n nodes with each node labeled from 0 to n - 1. The graph is represented by a **0-indexed** 2D integer array graph where graph[i] is an integer array of nodes adjacent to node i, meaning there is an edge from node i to each node in graph[i].

A node is a **terminal node** if there are no outgoing edges. A node is a **safe node** if every possible path starting from that node leads to a **terminal node** (or another safe node).

Return an array containing all the **safe nodes** of the graph. The answer should be sorted in **ascending** order.



Input: graph = [[1, 2], [2, 3], [5], [0], [5], [], []]

Output: [2, 4, 5, 6]

Explanation: The given graph is shown above.

Nodes 5 and 6 are terminal nodes as there are no outgoing edges from either of them.

Every path starting at nodes 2, 4, 5, and 6 all lead to either node 5 or 6.

Input: graph = [[1, 2, 3, 4], [1, 2], [3, 4], [0, 4], []]

Output: [4]

Explanation:

Only node 4 is a terminal node, and every path starting at node 4 leads to node 4.

21.17 Course Schedule

There are a total of numCourses courses you have to take, labeled from 0 to numCourses - 1. You are given an array prerequisites where prerequisites[i] = [a_i, b_i] indicates that you **must** take course b_i first if you want to take course a_i. For example, the pair [0, 1], indicates that to take course 0 you have to first take course 1.

Return true if you can finish all courses. Otherwise, return false.

Input: numCourses = 2, prerequisites = [[1, 0]]

Output: true

Explanation: There are a total of 2 courses to take.

To take course 1 you should have finished course 0. So it is possible.

Input: numCourses = 2, prerequisites = [[1, 0], [0, 1]]

Output: false

Explanation: There are a total of 2 courses to take.

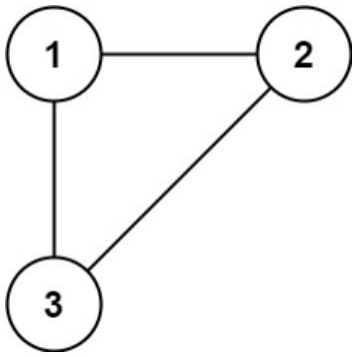
To take course 1 you should have finished course 0, and to take course 0 you should also have finished course 1. So it is impossible.

21.18 Redundant Connection

In this problem, a tree is an **undirected graph** that is connected and has no cycles.

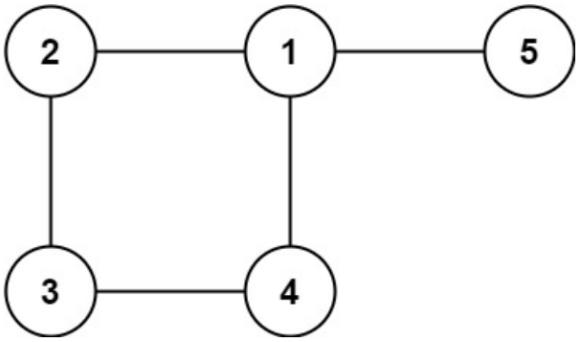
You are given a graph that started as a tree with n nodes labeled from 1 to n, with one additional edge added. The added edge has two **different** vertices chosen from 1 to n, and was not an edge that already existed. The graph is represented as an array edges of length n where edges[i] = $[a_i, b_i]$ indicates that there is an edge between nodes a_i and b_i in the graph.

Return an edge that can be removed so that the resulting graph is a tree of n nodes. If there are multiple answers, return the answer that occurs last in the input.



Input: edges = [[1, 2], [1, 3], [2, 3]]

Output: [2, 3]



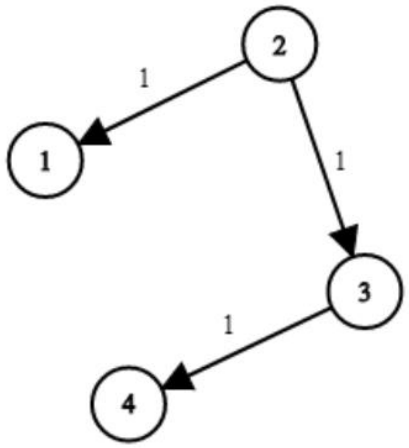
Input: edges = [[1, 2], [2, 3], [3, 4], [1, 4], [1, 5]]

Output: [1, 4]

21.19 Network Delay Time

You are given a network of n nodes, labeled from 1 to n. You are also given times, a list of travel times as directed edges times[i] = (u_i, v_i, w_i), where u_i is the source node, v_i is the target node, and w_i is the time it takes for a signal to travel from source to target.

We will send a signal from a given node k. Return the **minimum** time it takes for all the n nodes to receive the signal. If it is impossible for all the n nodes to receive the signal, return -1.



Input: times = [[2, 1, 1], [2, 3, 1], [3, 4, 1]], n = 4, k = 2

Output: 2

Input: times = [[1, 2, 1]], n = 2, k = 1

Output: 1

Input: times = [[1, 2, 1]], n = 2, k = 2

Output: -1

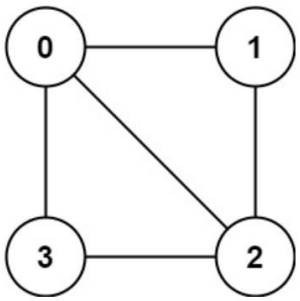
21.20 Is Graph Bipartite

There is an **undirected** graph with n nodes, where each node is numbered between 0 and n - 1. You are given a 2D array graph, where graph[u] is an array of nodes that node u is adjacent to. More formally, for each v in graph[u], there is an undirected edge between node u and node v. The graph has the following properties:

- There are no self-edges (graph[u] does not contain u).
- There are no parallel edges (graph[u] does not contain duplicate values).
- If v is in graph[u], then u is in graph[v] (the graph is undirected).
- The graph may not be connected, meaning there may be two nodes u and v such that there is no path between them.

A graph is **bipartite** if the nodes can be partitioned into two independent sets A and B such that **every** edge in the graph connects a node in set A and a node in set B.

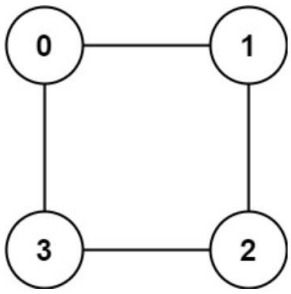
Return true if and only if it is **bipartite**.



Input: graph = [[1, 2, 3], [0, 2], [0, 1, 3], [0, 2]]

Output: false

Explanation: There is no way to partition the nodes into two independent sets such that every edge connects a node in one and a node in the other.



Input: graph = [[1, 3], [0, 2], [1, 3], [0, 2]]

Output: true

Explanation: We can partition the nodes into two sets: {0, 2} and {1, 3}.

21.21 Keys and Rooms

There are n rooms labeled from 0 to n - 1 and all the rooms are locked except for room 0. Your goal is to visit all the rooms. However, you cannot enter a locked room without having its key.

When you visit a room, you may find a set of **distinct keys** in it. Each key has a number on it, denoting which room it unlocks, and you can take all of them with you to unlock the other rooms.

Given an array rooms where rooms[i] is the set of keys that you can obtain if you visited room i, return true if you can visit **all** the rooms, or false otherwise.

Input: rooms = [[1], [2], [3], []]

Output: true

Explanation:

We visit room 0 and pick up key 1.

We then visit room 1 and pick up key 2.

We then visit room 2 and pick up key 3.

We then visit room 3.

Since we were able to visit every room, we return true.

Input: rooms = [[1, 3], [3, 0, 1], [2], [0]]

Output: false

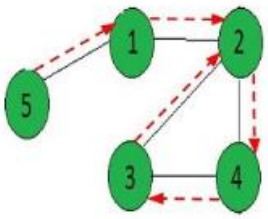
Explanation: We cannot enter room number 2 since the only key that unlocks it is in that room.

21.22 Seven Bridges of Konigsberg

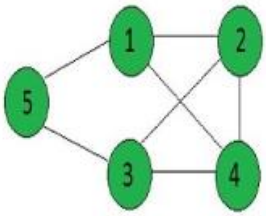
There was 7 bridges connecting 4 lands around the city of Königsberg in Prussia. Was there any way to start from any of the land and go through each of the bridges once and only once? Euler first introduced graph theory to solve this problem. He considered each of the lands as a node of a graph and each bridge in

between as an edge in between. Now he calculated if there is any Eulerian Path in that graph. If there is an Eulerian path then there is a solution otherwise not.

There are n nodes and m bridges in between these nodes. Print the possible path through each node using each edges (if possible), traveling through each edges only once.



Map 1



Map 2

No such path

Input: [[0, 1, 0, 0, 1],
[1, 0, 1, 1, 0],
[0, 1, 0, 1, 0],
[0, 1, 1, 0, 0],
[1, 0, 0, 0, 0]]

Output: 5 -> 1 -> 2 -> 4 -> 3 -> 2

Input: [[0, 1, 0, 1, 1],
[1, 0, 1, 0, 1],
[0, 1, 0, 1, 1],
[1, 1, 1, 0, 0],
[1, 0, 1, 0, 0]]

Output: "No Solution"

21.23 Hamiltonian Cycle

The Hamiltonian cycle of undirected graph G = <V, E> is the cycle containing each vertex in V. If graph contains a Hamiltonian cycle, it is called Hamiltonian graph otherwise it is non-Hamiltonian.
Hamiltonian Path in an undirected graph is a path that visits each vertex exactly once. A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian path such that there is an edge (in the graph) from the last vertex to the first vertex of the Hamiltonian path. Consider a graph G, and determine whether a given graph contains Hamiltonian cycle or not. If it contains, then prints the path. Following are the input and output of the required function.

Input: A 2D array graph [V][V] where V is the number of vertices in graph and graph [V][V] is adjacency matrix representation of the graph. A value graph[i][j] is 1 if there is a direct edge from i to j, otherwise graph[i][j] is 0.

Output: An array path [V] that should contain the Hamiltonian Path. Path [i] should represent the i^{th} vertex in the Hamiltonian Path. The code should also return false if there is no Hamiltonian Cycle in the graph.

For example, a Hamiltonian Cycle in the following graph is {0, 1, 2, 4, 3, 0}.

(0)--(1)--(2)

| / \ |

| / \ |

| / \ |

(3)-------(4)

And the following graph doesn't contain any Hamiltonian Cycle.

(0)--(1)--(2)

| / \ |

| / \ |

| / \ |

(3) (4)

Backtracking Algorithm: Create an empty path array and add vertex 0 to it. Add other vertices, starting from the vertex 1. Before adding a vertex, check for whether it is adjacent to the previously added vertex and not already added. If we find such a vertex, we add the vertex as part of the solution. If we do not find a vertex then we return false.

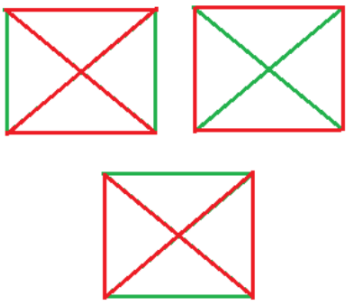
21.24 Number of Hamiltonian Cycle

Given an undirected complete graph of N vertices where N > 2. The task is to find the number of different Hamiltonian cycle of the graph.

Complete Graph: A graph is said to be complete if each possible vertices is connected through an Edge.
Hamiltonian Cycle: It is a closed walk such that each vertex is visited at most once except the initial vertex. and it is not necessary to visit all the edges.

Formula: (N – 1)! / 2

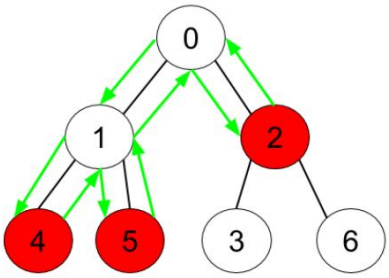
Input: N = 6
Output: Hamiltonian cycles = 60
Input: N = 4
Output: Hamiltonian cycles = 3
Explanation: Let us take the example of N = 4 complete undirected graph, The 3 different Hamiltonian cycle is as shown below:



22. Graph Traversal

22.1 Minimum Time to Collect All Apples in a Tree

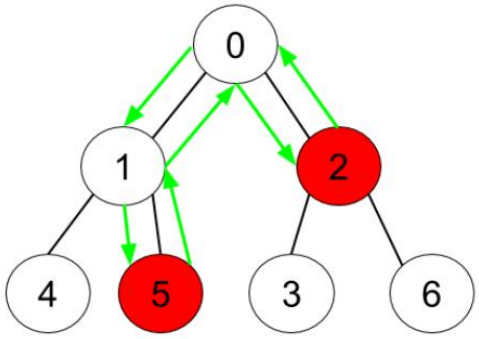
Given an undirected tree consisting of n vertices numbered from 0 to n-1, which has some apples in their vertices. You spend 1 second to walk over one edge of the tree. Return the minimum time in seconds you have to spend to collect all apples in the tree, starting at **vertex 0** and coming back to this vertex.
The edges of the undirected tree are given in the array edges, where edges[i] = [a_i, b_i] means that exists an edge connecting the vertices a_i and b_i. Additionally, there is a boolean array hasApple, where hasApple[i] = true means that vertex i has an apple; otherwise, it does not have any apple.



Input: n = 7, edges = [[0, 1], [0, 2], [1, 4], [1, 5], [2, 3], [2, 6]], hasApple = [false, false, true, false, true, true, false]

Output: 8

Explanation: The figure above represents the given tree where red vertices have an apple. One optimal path to collect all apples is shown by the green arrows.



Input: n = 7, edges = [[0, 1], [0, 2], [1, 4], [1, 5], [2, 3], [2, 6]], hasApple = [false, false, true, false, false, true, false]

Output: 6

Explanation: The figure above represents the given tree where red vertices have an apple. One optimal path to collect all apples is shown by the green arrows.

Input: n = 7, edges = [[0, 1], [0, 2], [1, 4], [1, 5], [2, 3], [2, 6]], hasApple = [false, false, false, false, false, false, false]

Output: 0

22.2 Depth First Search

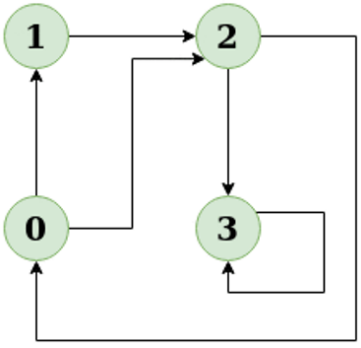
Depth First Traversal (or DFS) for a graph is similar to Depth First Traversal of a tree. The only catch here is, that, unlike trees, graphs may contain cycles (a node may be visited twice). To avoid processing a node more than once, use a boolean visited array. A graph can have more than one DFS traversal.

For a given graph G, print DFS traversal from a given source vertex.

Input: n = 4, e = 6
0 -> 1, 0 -> 2, 1 -> 2, 2 -> 0, 2 -> 3, 3 -> 3

Output: DFS from vertex 1: 1 2 0 3

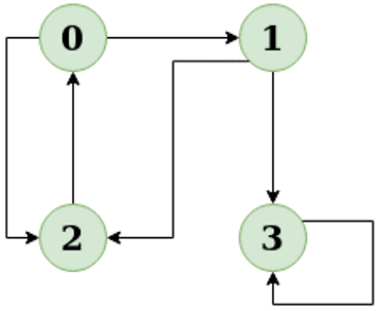
Explanation:
DFS Diagram:



Input: n = 4, e = 6
2 -> 0, 0 -> 2, 1 -> 2, 0 -> 1, 3 -> 3, 1 -> 3

Output: DFS from vertex 2: 2 0 1 3

Explanation:
DFS Diagram:



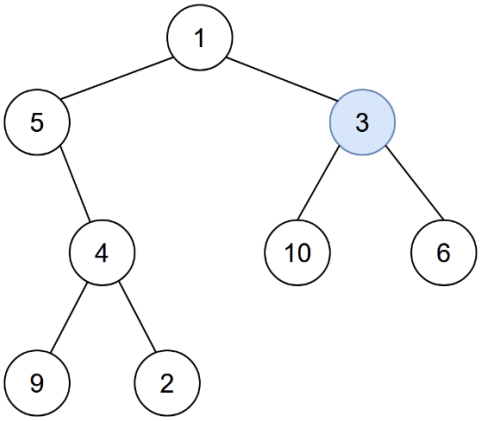
22.3 Amount of Time for Binary Tree to be Infected

You are given the root of a binary tree with **unique** values, and an integer start. At minute 0, an **infection** starts from the node with value start.

Each minute, a node becomes infected if:

- The node is currently uninfected.
- The node is adjacent to an infected node.

Return the number of minutes needed for the entire tree to be infected.



Input: root = [1, 5, 3, null, 4, 10, 6, 9, 2], start = 3

Output: 4

Explanation: The following nodes are infected during:

- Minute 0: Node 3
- Minute 1: Nodes 1, 10 and 6
- Minute 2: Node 5
- Minute 3: Node 4
- Minute 4: Nodes 9 and 2

It takes 4 minutes for the whole tree to be infected so we return 4.

Input: root = [1], start = 1

Output: 0

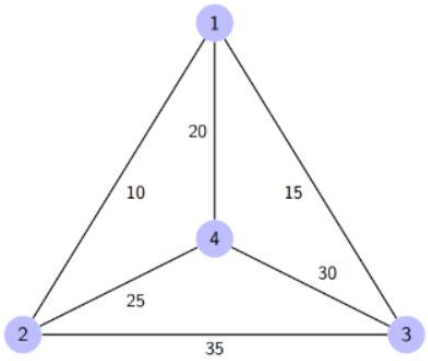
Explanation: At minute 0, the only node in the tree is infected so we return 0.

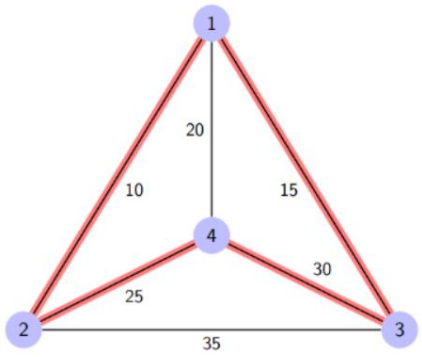
23. Shortest Paths

23.1 Travelling Salesman Problem

Given a set of cities and the distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point. The problem statement gives a list of cities along with the distances between each city.

Objective: To start from the origin city, visit other cities only once, and return to the original city again. Our target is to find the shortest possible path to complete the round-trip route.





Here a graph is given where 1, 2, 3, and 4 represent the cities, and the weight associated with every edge represents the distance between those cities. The goal is to find the shortest possible path for the tour that starts from the origin city, traverses the graph while only visiting the other cities or nodes once, and returns to the origin city.
For the above graph, the optimal route is to follow the minimum cost path: 1 – 2 – 4 – 3 - 1. And this shortest route would cost 10 + 25 + 30 + 15 =80

Algorithm for Traveling Salesman Problem: We will use the dynamic programming approach to solve the Travelling Salesman Problem (TSP).

- A graph G=(V, E), which is a set of vertices and edges.
- V is the set of vertices.
- E is the set of edges.
- Vertices are connected through edges.
- Dist(i,j) denotes the non-negative distance between two vertices, i and j.

Let's assume S is the subset of cities and belongs to {1, 2, 3, …, n} where 1, 2, 3…n are the cities and i, j are two cities in that subset. Now cost(i, S, j) is defined in such a way as the length of the shortest path visiting node in S, which is exactly once having the starting and ending point as i and j respectively.

For example, cost (1, {2, 3, 4}, 1) denotes the length of the shortest path where:

- Starting city is 1
- Cities 2, 3, and 4 are visited only once
- The ending point is 1

The dynamic programming algorithm would be:

- Set cost(i,, i) = 0, which means we start and end at i, and the cost is 0.
- When |S| > 1, we define cost(i, S, 1) = ∝ where i !=1 . Because initially, we do not know the exact cost to reach city i to city 1 through other cities.
- Now, we need to start at 1 and complete the tour. We need to select the next city in such a way-
- cost(i, S, j) = min cost (i, S−{i}, j) + dist(i, j) where i ∈ S and i ≠ j

For the given figure above, the adjacency matrix would be the following:

dist(i, j)	1	2	3	4
1	0	10	15	20

2	10	0	35	25
3	15	35	0	30
4	20	25	30	0

Now S = {1, 2, 3, 4}. There are four elements. Hence the number of subsets will be 2^4 or 16. Those subsets are-

1) |S| = Null:

{Φ}

2) |S| = 1:

{{1}, {2}, {3}, {4}}

3) |S| = 2:

{{1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 4}}

4) |S| = 3:

{{1, 2, 3}, {1, 2, 4}, {2, 3, 4}, {1, 3, 4}}

5) |S| = 4:

{{1, 2, 3, 4}}

As we are starting at 1, we could discard the subsets containing city 1. The algorithm calculation steps:

1) |S| = Φ:

cost (2, Φ, 1) = dist(2, 1) = 10
cost (3, Φ, 1) = dist(3, 1) = 15
cost (4, Φ, 1) = dist(4, 1) = 20

2) |S| = 1:

cost (2, {3}, 1) = dist(2, 3) + cost (3, Φ, 1) = 35+15 = 50
cost (2, {4}, 1) = dist(2, 4) + cost (4, Φ, 1) = 25+20 = 45
cost (3, {2}, 1) = dist(3, 2) + cost (2, Φ, 1) = 35+10 = 45
cost (3, {4}, 1) = dist(3, 4) + cost (4, Φ, 1) = 30+20 = 50
cost (4, {2}, 1) = dist(4, 2) + cost (2, Φ, 1) = 25+10 = 35
cost (4, {3}, 1) = dist(4, 3) + cost (3, Φ, 1) = 30+15 = 45

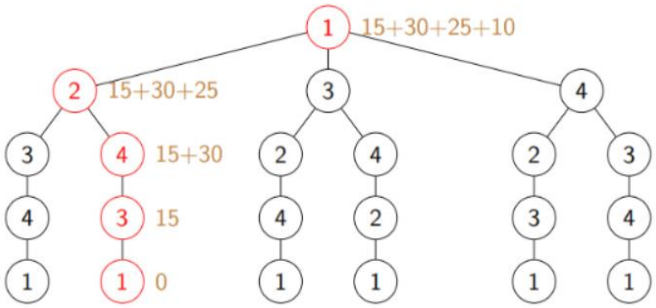
3) |S| = 2:

cost (2, {3, 4}, 1) = min [dist[2,3] + Cost(3,{4},1) = 35+50 = 85,
dist[2,4]+Cost(4,{3},1) = 25+45 = 70] = 70
cost (3, {2, 4}, 1) = min [dist[3,2] + Cost(2,{4},1) = 35+45 = 80,
dist[3,4]+Cost(4,{2},1) = 30+35 = 65] = 65
cost (4, {2, 3}, 1) = min [dist[4,2] + Cost(2,{3},1) = 25+50 = 75
dist[4,3] + Cost(3,{2},1) = 30+45 = 75] = 75

4) |S| = 3:

cost (1, {2, 3, 4}, 1) = min [dist[1,2]+Cost(2,{3,4},1) = 10+70 = 80
dist[1,3]+Cost(3,{2,4},1) = 15+65 = 80
dist[1,4]+Cost(4,{2,3},1) = 20+75 = 95] = 80

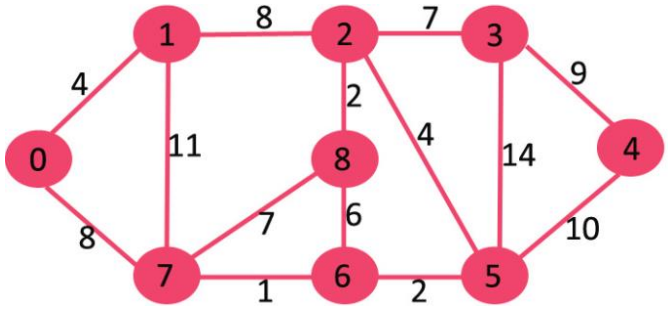
So the optimal solution would be 1-2-4-3-1



23.2 Shortest Paths from Source to all Vertices (Dijkstra's Algorithm)

Given a graph and a source vertex in the graph, find the shortest paths from the source to all vertices in the given graph.

Input: src = 0, the graph is shown below.



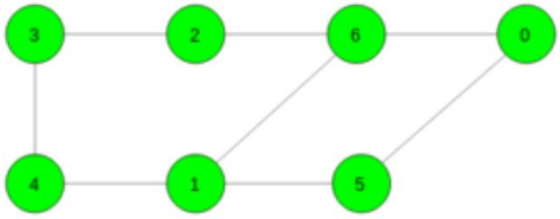
Output: 0 4 12 19 21 11 9 8 14

Explanation: The distance from 0 to 1 = 4.
The minimum distance from 0 to 2 = 12. 0->1->2
The minimum distance from 0 to 3 = 19. 0->1->2->3
The minimum distance from 0 to 4 = 21. 0->7->6->5->4
The minimum distance from 0 to 5 = 11. 0->7->6->5
The minimum distance from 0 to 6 = 9. 0->7->6
The minimum distance from 0 to 7 = 8. 0->7
The minimum distance from 0 to 8 = 14. 0->1->2->8

23.3 Shortest Cycle in an Undirected Unweighted Graph

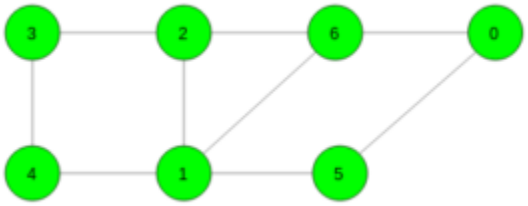
Given an undirected unweighted graph. The task is to find the length of the shortest cycle in the given graph. If no cycle exists print -1.

Input: Consider the graph given below



Output: 4
Cycle 6 -> 1 -> 5 -> 0 -> 6

Input: Consider the graph given below



Output: 3

Cycle 6 -> 1 -> 2 -> 6

23.4 Count Unique and all Possible Paths in a M x N Matrix

Count unique paths: The problem is to count all unique possible paths from the top left to the bottom right of a M X N matrix with the constraints that from each cell you can either move only to the right or down.

Input: M = 2, N = 2

Output: 2

Explanation: There are two paths
(0, 0) -> (0, 1) -> (1, 1)
(0, 0) -> (1, 0) -> (1, 1)

Input: M = 2, N = 3

Output: 3

Explanation: There are three paths
(0, 0) -> (0, 1) -> (0, 2) -> (1, 2)
(0, 0) -> (0, 1) -> (1, 1) -> (1, 2)
(0, 0) -> (1, 0) -> (1, 1) -> (1, 2)

Count all possible paths: We can recursively move to right and down from the start until we reach the destination and then add up all valid paths to get the answer.

Procedure:

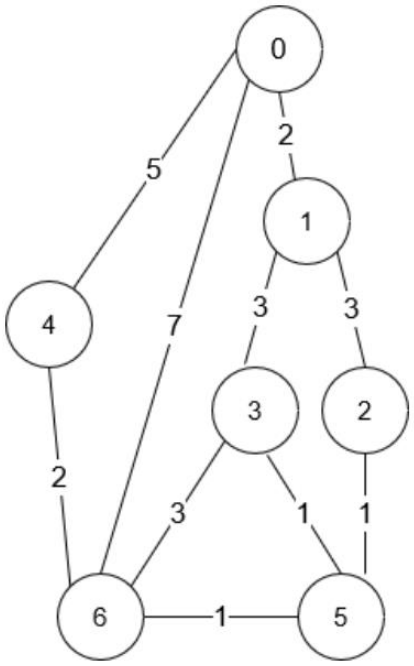
- Create a recursive function with parameters as row and column index
- Call this recursive function for N-1 and M-1
- In the recursive function
    - If N == 1 or M == 1 then return 1
    - else call the recursive function with (N-1, M) and (N, M-1) and return the sum of this
- Print the answer

23.5 Number of Ways to Arrive at Destination

You are in a city that consists of n intersections numbered from 0 to n - 1 with bi-directional roads between some intersections. The inputs are generated such that you can reach any intersection from any other intersection and that there is at most one road between any two intersections.

You are given an integer n and a 2D integer array roads where roads[i] = [u_i, v_i, $time_i$] means that there is a road between intersections u_i and v_i that takes $time_i$ minutes to travel. You want to know in how many ways you can travel from intersection 0 to intersection n - 1 in the shortest amount of time.

Return the number of ways you can arrive at your destination in the shortest amount of time. Since the answer may be large, return it modulo $10^9 + 7$.



Input: n = 7, roads = [[0,6,7],[0,1,2],[1,2,3],[1,3,3],[6,3,3],[3,5,1],[6,5,1],[2,5,1],[0,4,5],[4,6,2]]

Output: 4

Explanation: The shortest amount of time it takes to go from intersection 0 to intersection 6 is 7 minutes. The four ways to get there in 7 minutes are:

- 0 → 6
- 0 → 4 → 6
- 0 → 1 → 2 → 5 → 6
- 0 → 1 → 3 → 5 → 6

Input: n = 2, roads = [[1, 0, 10]]

Output: 1

Explanation: There is only one way to go from intersection 0 to intersection 1, and it takes 10 minutes.

23.6 Minimum Cost of a Path with Special Roads

You are given an array start where start = [startX, startY] represents your initial position (startX, startY) in a 2D space. You are also given the array target where target = [targetX, targetY] represents your target position (targetX, targetY).

The cost of going from a position (x1, y1) to any other position in the space (x2, y2) is |x2 - x1| + |y2 - y1|.

There are also some special roads. You are given a 2D array specialRoads where specialRoads[i] = [$x1_i$, $y1_i$, $x2_i$, $y2_i$, $cost_i$] indicates that the i^{th} special road can take you from ($x1_i$, $y1_i$) to ($x2_i$, $y2_i$) with a cost equal to $cost_i$. You can use each special road any number of times.

Return the minimum cost required to go from (startX, startY) to (targetX, targetY).

Input: start = [1,1], target = [4,5], specialRoads = [[1,2,3,3,2],[3,4,4,5,1]]

Output: 5

Explanation: The optimal path from (1,1) to (4,5) is the following:

- (1,1) -> (1,2). This move has a cost of |1 - 1| + |2 - 1| = 1.

- (1,2) -> (3,3). This move uses the first special edge, the cost is 2.

- (3,3) -> (3,4). This move has a cost of |3 - 3| + |4 - 3| = 1.

- (3,4) -> (4,5). This move uses the second special edge, the cost is 1.

So the total cost is 1 + 2 + 1 + 1 = 5.

It can be shown that we cannot achieve a smaller total cost than 5.

Input: start = [3,2], target = [5,7], specialRoads = [[3,2,3,4,4],[3,3,5,5,5],[3,4,5,6,6]]

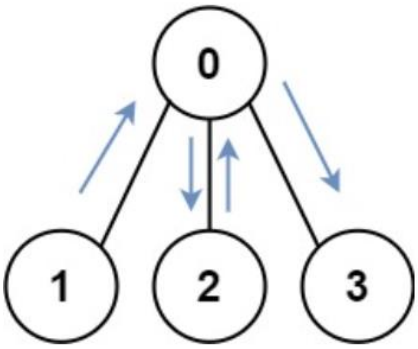
Output: 7

Explanation: It is optimal to not use any special edges and go directly from the starting to the ending position with a cost |5 - 3| + |7 - 2| = 7.

23.7 Shortest Path Visiting All Nodes

You have an undirected, connected graph of n nodes labeled from 0 to n - 1. You are given an array graph where graph[i] is a list of all the nodes connected with node i by an edge.

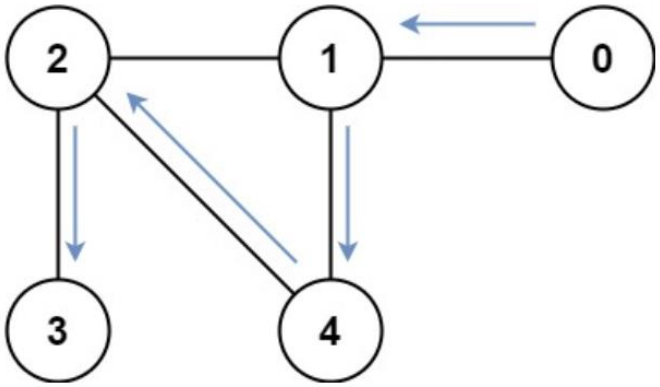
Return the length of the shortest path that visits every node. You may start and stop at any node, you may revisit nodes multiple times, and you may reuse edges.



Input: graph = [[1, 2, 3], [0], [0], [0]]

Output: 4

Explanation: One possible path is [1, 0, 2, 0, 3]



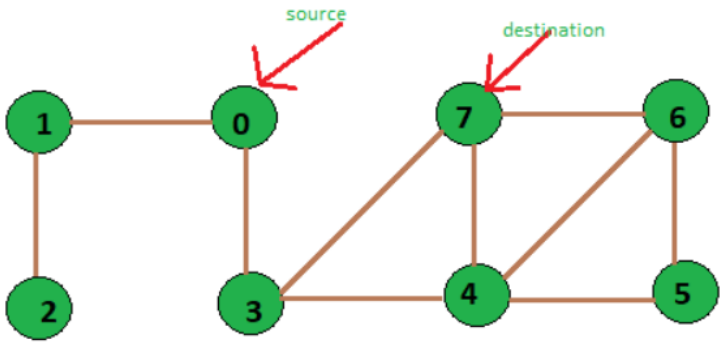
Input: graph = [[1], [0, 2, 4], [1, 3, 4], [2], [1, 2]]

Output: 4

Explanation: One possible path is [0, 1, 4, 2, 3]

23.8 Shortest Path in an Unweighted Graph

Given an unweighted graph, a source, and a destination, we need to find the shortest path from source to destination in the graph in the most optimal way.



Input: source vertex = 0 and destination vertex is = 7.

Output: Shortest path length is: 2

Path is:: 0 3 7

Input: source vertex is = 2 and destination vertex is = 6.

Output: Shortest path length is: 5

Path is:: 2 1 0 3 4 6

24. Graph Coloring

24.1 Graph Coloring using Greedy Algorithm

Greedy algorithm is used to assign colors to the vertices of a graph. It doesn't guarantee to use minimum colors, but it guarantees an upper bound on the number of colors. The basic algorithm never uses more than d+1 colors where d is the maximum degree of a vertex in the given graph.

Basic Greedy Coloring Algorithm:
1. Color first vertex with first color.
2. Do following for remaining V-1 vertices.
    a) Consider the currently picked vertex and color it with the lowest numbered color that has not been used on any previously colored vertices adjacent to it. If all previously used colors appear on vertices adjacent to v, assign a new color to it.

Output:
Coloring of graph 1
Vertex 0 ---> Color 0
Vertex 1 ---> Color 1
Vertex 2 ---> Color 2
Vertex 3 ---> Color 0
Vertex 4 ---> Color 1

Coloring of graph 2
Vertex 0 ---> Color 0
Vertex 1 ---> Color 1
Vertex 2 ---> Color 2
Vertex 3 ---> Color 0
Vertex 4 ---> Color 3

24.2 Coloring a Cycle Graph

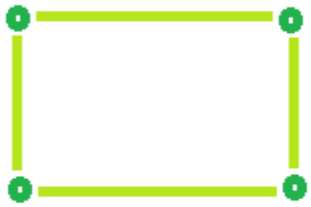
Given the number of vertices in a Cyclic Graph. The task is to determine the Number of colors required to color the graph so that no two adjacent vertices have the same color.

Approach:
- If the no. of vertices is Even then it is Even Cycle and to color such graph we require 2 colors.
- If the no. of vertices is Odd then it is Odd Cycle and to color such graph we require 3 colors.

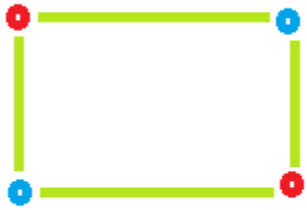
Input: Vertices = 3
Output: No. of colors require is: 3

Input: vertices = 4
Output: No. of colors require is: 2

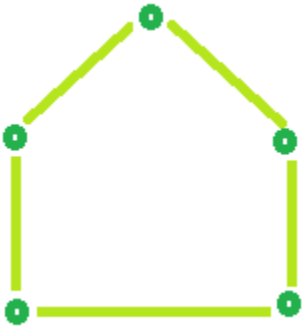
Example 1: Even Cycle: Number of vertices = 4



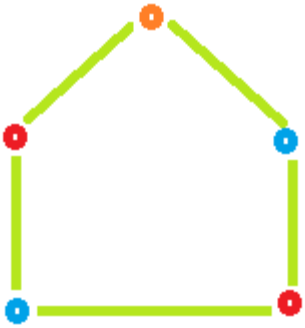
Color required = 2



Example 2: Odd Cycle: Number of vertices = 5



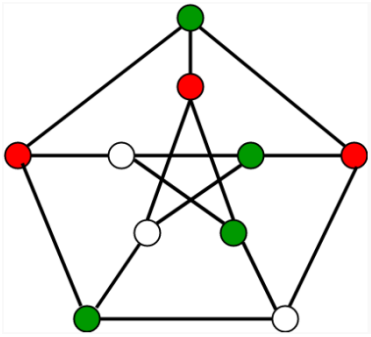
Color required = 3



24.3 m Coloring Problem

Given an undirected graph and a number m, determine if the graph can be colored with at most m colors such that no two adjacent vertices of the graph are colored with the same color.

Note: Here coloring of a graph means the assignment of colors to all vertices. Following is an example of a graph that can be colored with 3 different colors:



Input: graph = {0, 1, 1, 1},
{1, 0, 1, 0},

{1, 1, 0, 1},
{1, 0, 1, 0}

Output: Solution Exists: Following are the assigned colors: 1 2 3 2
Explanation: By coloring the vertices with following colors, adjacent vertices does not have same colors

Input: graph = {1, 1, 1, 1},
{1, 1, 1, 1},
{1, 1, 1, 1},
{1, 1, 1, 1}
Output: Solution does not exist
Explanation: No solution exits

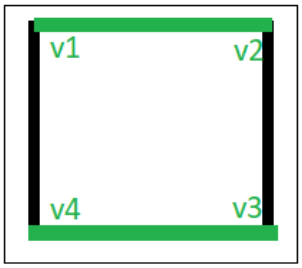
24.4 Edge Coloring of a Graph

Edge coloring of a graph is an assignment of "colors" to the edges of the graph so that no two adjacent edges have the same color with an optimal number of colors. Two edges are said to be adjacent if they are connected to the same vertex. There is no known polynomial time algorithm for edge-coloring every graph with an optimal number of colors.

Input: u1 = 1, v1 = 4
u2 = 1, v2 = 2
u3 = 2, v3 = 3
u4 = 3, v4 = 4

Output: Edge 1 is of color 1
Edge 2 is of color 2
Edge 3 is of color 1
Edge 4 is of color 2

The above input shows the pair of vertices (u_i, v_i) who have an edge between them. The output shows the color assigned to the respective edges.



Edge colorings are one of several different types of graph coloring problems. The above figure of a Graph shows an edge coloring of a graph by the colors green and black, in which no adjacent edge have the same color.

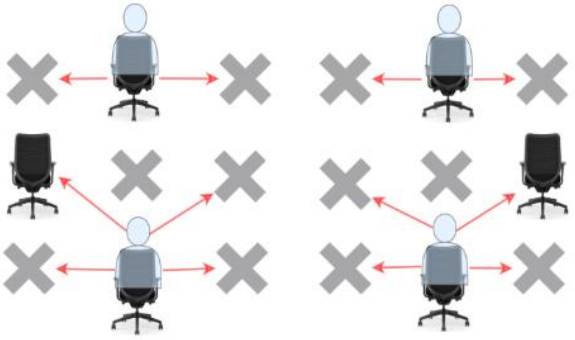
25. Maximum FLow

25.1 Maximum Students Taking Exam

Given a m * n matrix seats that represent seats distributions in a classroom. If a seat is broken, it is denoted by '#' character otherwise it is denoted by a '.' character.

Students can see the answers of those sitting next to the left, right, upper left and upper right, but he cannot see the answers of the student sitting directly in front or behind him. Return the **maximum** number of students that can take the exam together without any cheating being possible.

Students must be placed in seats in good condition.



Input: seats = [["#",".","#","#",".","#"],
["."."#","#","#","#","."],
["#",".","#","#",".","#"]]

Output: 4

Explanation: Teacher can place 4 students in available seats so they don't cheat on the exam.

Input: seats = [["."."#"],
["#","#"],
["#","."],
["#","#"],
["."."#"]]

Output: 3

Explanation: Place all students in available seats.

Input: seats = [["#",".",".",".",".","#"],
[".","#",".",".","#","."],
[".",".","#",".",".","."],
[".","#",".",".","#","."],
["#",".",".",".",".","#"]]

Output: 10

Explanation: Place students in available seats in column 1, 3 and 5.

25.2 Minimum XOR Sum of Two Arrays

You are given two integer arrays nums1 and nums2 of length n.

The **XOR sum** of the two integer arrays is (nums1[0] XOR nums2[0]) + (nums1[1] XOR nums2[1]) + ... + (nums1[n - 1] XOR nums2[n - 1]) (**0-indexed**).

- For example, the **XOR sum** of [1,2,3] and [3,2,1] is equal to (1 XOR 3) + (2 XOR 2) + (3 XOR 1) = 2 + 0 + 2 = 4.

Rearrange the elements of nums2 such that the resulting **XOR sum** is **minimized**.

Return the **XOR sum** after the rearrangement.

Input: nums1 = [1,2], nums2 = [2,3]

Output: 2

Explanation: Rearrange nums2 so that it becomes [3,2].

The XOR sum is (1 XOR 3) + (2 XOR 2) = 2 + 0 = 2.

Input: nums1 = [1,0,3], nums2 = [5,3,4]

Output: 8

Explanation: Rearrange nums2 so that it becomes [5,4,3].

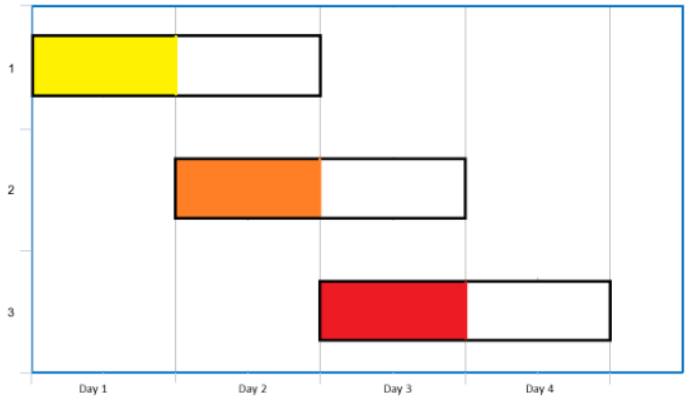
The XOR sum is (1 XOR 5) + (0 XOR 4) + (3 XOR 3) = 4 + 4 + 0 = 8.

25.3 Maximum Number of Events that can be Attended

You are given an array of events where events[i] = [$startDay_i$, $endDay_i$]. Every event i starts at $startDay_i$ and ends at $endDay_i$.

You can attend an event i at any day d where $startTime_i$ <= d <= $endTime_i$. You can only attend one event at any time d.

Return the maximum number of events you can attend.



Input: events = [[1, 2], [2, 3], [3, 4]]

Output: 3

Explanation: You can attend all the three events.

One way to attend them all is as shown.

Attend the first event on day 1.

Attend the second event on day 2.

Attend the third event on day 3.

Input: events= [[1, 2], [2, 3], [3, 4], [1, 2]]

Output: 4

25.4 Maximum AND Sum of Array

You are given an integer array nums of length n and an integer numSlots such that 2 * numSlots >= n. There are numSlots slots numbered from 1 to numSlots.

You have to place all n integers into the slots such that each slot contains at **most** two numbers. The **AND sum** of a given placement is the sum of the **bitwise** AND of every number with its respective slot number.

- For example, the **AND sum** of placing the numbers [1, 3] into slot 1 and [4, 6] into slot 2 is equal to (1 AND 1) + (3 AND 1) + (4 AND 2) + (6 AND 2) = 1 + 1 + 0 + 2 = 4.

Return the maximum possible **AND sum** of nums given numSlots slots.

Input: nums = [1, 2, 3, 4, 5, 6], numSlots = 3

Output: 9

Explanation: One possible placement is [1, 4] into slot 1, [2, 6] into slot 2, and [3, 5] into slot 3.

This gives the maximum AND sum of (1 AND 1) + (4 AND 1) + (2 AND 2) + (6 AND 2) + (3 AND 3) + (5 AND 3) = 1 + 0 + 2 + 2 + 3 + 1 = 9.

Input: nums = [1, 3, 10, 4, 7, 1], numSlots = 9

Output: 24

Explanation: One possible placement is [1, 1] into slot 1, [3] into slot 3, [4] into slot 4, [7] into slot 7, and [10] into slot 9.

This gives the maximum AND sum of (1 AND 1) + (1 AND 1) + (3 AND 3) + (4 AND 4) + (7 AND 7) + (10 AND 9) = 1 + 1 + 3 + 4 + 7 + 8 = 24.

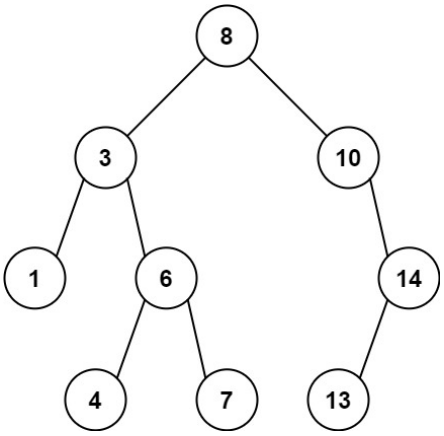
Note that slots 2, 5, 6, and 8 are empty which is permitted.

26. Trees

26.1 Maximum Difference between Node and Ancestor

Given the root of a binary tree, find the maximum value v for which there exist **different** nodes a and b where v = |a.val - b.val| and a is an ancestor of b.

A node a is an ancestor of b if either: any child of a is equal to b or any child of a is an ancestor of b.



Input: root = [8,3,10,1,6,null,14,null,null,4,7,13]

Output: 7

Explanation: We have various ancestor-node differences, some of which are given below :

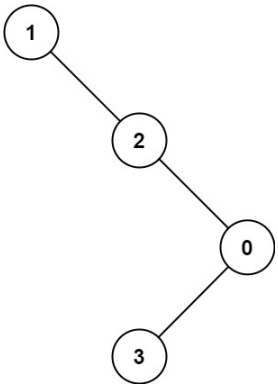
|8 - 3| = 5

|3 - 7| = 4

|8 - 1| = 7

|10 - 13| = 3

Among all possible differences, the maximum value of 7 is obtained by |8 - 1| = 7.



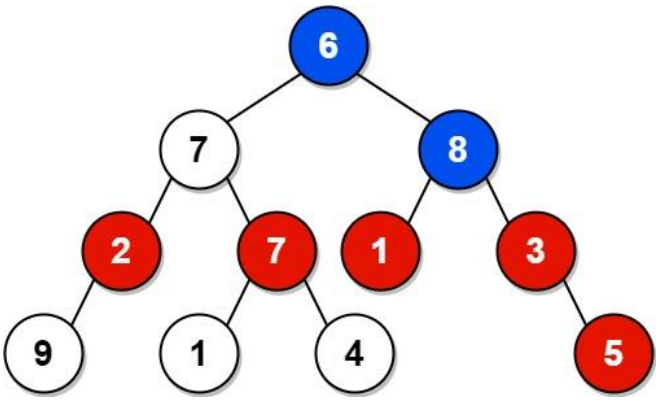
Input: root = [1, null, 2, null, 0, 3]

Output: 3

26.2 Sum of Nodes with Even-Values Grandparent

Given the root of a binary tree, return the sum of values of nodes with an **even-valued grandparent**. If there are no nodes with an **even-valued grandparent**, return 0.

A **grandparent** of a node is the parent of its parent if it exists.



Input: root = [6, 7, 8, 2, 7, 1, 3, 9, null, 1, 4, null, null, null, 5]

Output: 18

Explanation: The red nodes are the nodes with even-value grandparent while the blue nodes are the even-value grandparents.

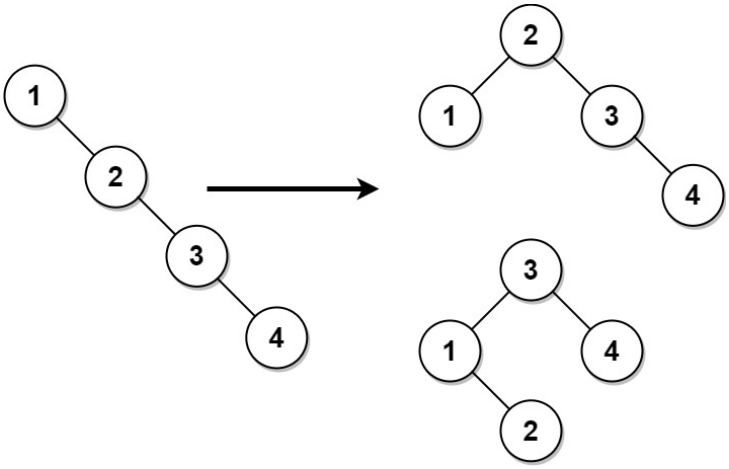
Input: root = [1]

Output: 0

26.3 Balance a Binary Search Tree

Given the root of a binary search tree, return *a **balanced** binary search tree with the same node values*. If there is more than one answer, return **any of them**.

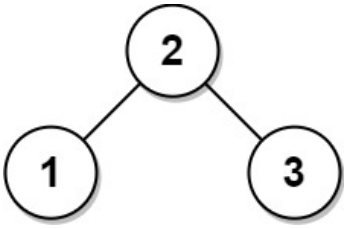
A binary search tree is **balanced** if the depth of the two subtrees of every node never differs by more than 1.



Input: root = [1, null, 2, null, 3, null, 4, null, null]

Output: [2, 1, 3, null, null, null, 4]

Explanation: This is not the only correct answer, [3, 1, 4, null, 2] is also correct.

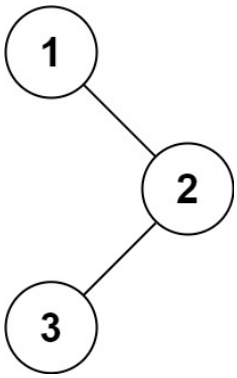


Input: root = [2, 1, 3]

Output: [2, 1, 3]

26.4 Binary Tree Inorder Traversal

Given the root of a binary tree, return the inorder traversal of its nodes' values.



Input: root = [1, null, 2, 3]

Output: [1, 3, 2]

Input: root = []

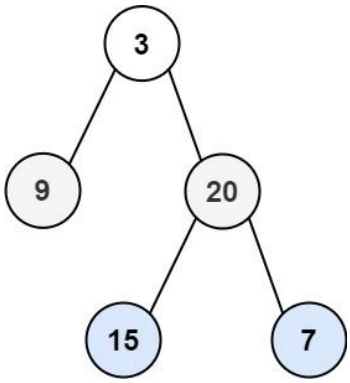
Output: []

Input: root = [1]

Output: [1]

26.5 Binary Tree Level Order Traversal

Given the root of a binary tree, return *the level order traversal of its nodes' values*. (i.e., from left to right, level by level).



Input: root = [3, 9, 20, null, null, 15, 7]

Output: [[3], [9, 20], [15, 7]]

Input: root = [1]

Output: [[1]]

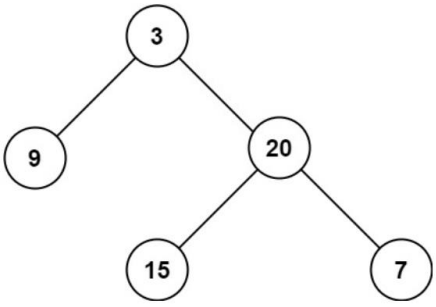
Input: root = []

Output: []

26.6 Maximum Depth of Binary Tree

Given the root of a binary tree, return *its maximum depth*.

A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.



Input: root = [3, 9, 20, null, null, 15, 7]

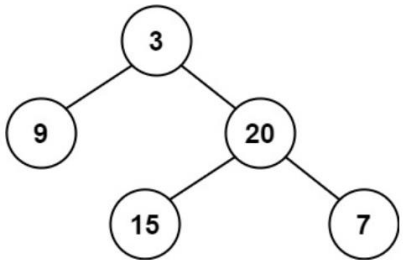
Output: 3

Input: root = [1, null, 2]

Output: 2

26.7 Balanced Binary Tree

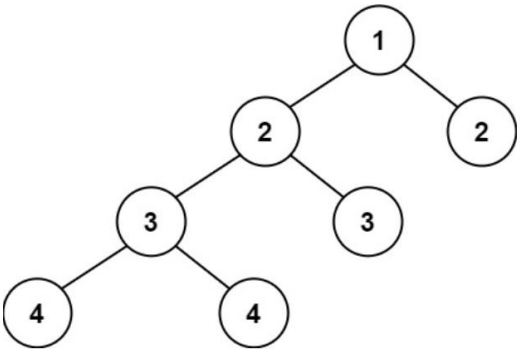
Given a binary tree, determine if it is **height-balanced**



.

Input: root = [3, 9, 20, null, null, 15, 7]

Output: true



Input: root = [1, 2, 2, 3, 3, null, null, 4, 4]

Output: false

Input: root = []

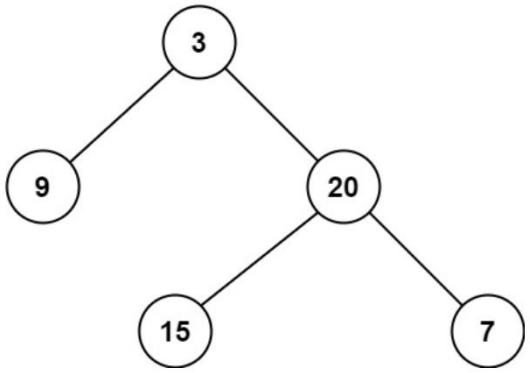
Output: true

26.8 Minimum Depth of Binary Tree

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

Note: A leaf is a node with no children.



Input: root = [3, 9, 20, null, null, 15, 7]

Output: 2

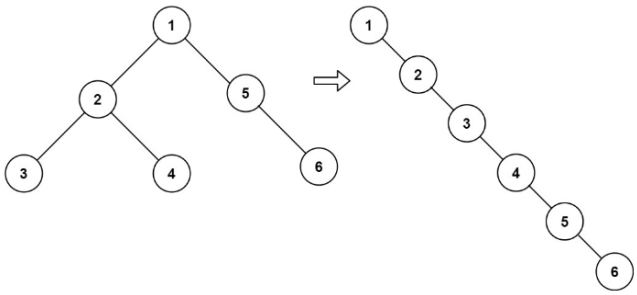
Input: root = [2, null, 3, null, 4, null, 5, null, 6]

Output: 5

26.9 Minimum Depth of Binary Tree

Given the root of a binary tree, flatten the tree into a "linked list":

- The "linked list" should use the same TreeNode class where the right child pointer points to the next node in the list and the left child pointer is always null.
- The "linked list" should be in the same order as a **pre-order traversal** of the binary tree.



Input: root = [1, 2, 5, 3, 4, null, 6]

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

Input: root = []

Output: []

Input: root = [0]

Output: [0]

26.10 Sum Root to Leaf Numbers

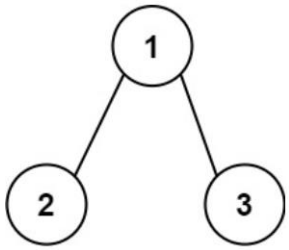
You are given the root of a binary tree containing digits from 0 to 9 only.

Each root-to-leaf path in the tree represents a number.

- For example, the root-to-leaf path 1 -> 2 -> 3 represents the number 123.

Return the total sum of all root-to-leaf numbers. Test cases are generated so that the answer will fit in a **32-bit** integer.

A **leaf** node is a node with no children.



Input: root = [1, 2, 3]

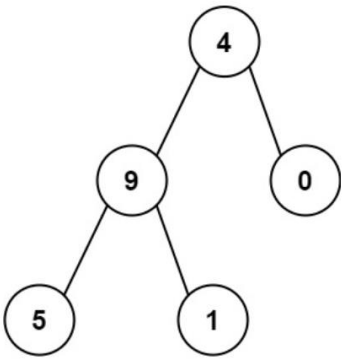
Output: 25

Explanation:

The root-to-leaf path 1->2 represents the number 12.

The root-to-leaf path 1->3 represents the number 13.

Therefore, sum = 12 + 13 = 25.



Input: root = [4, 9, 0, 5, 1]

Output: 1026

Explanation:

The root-to-leaf path 4->9->5 represents the number 495.

The root-to-leaf path 4->9->1 represents the number 491.

The root-to-leaf path 4->0 represents the number 40.

Therefore, sum = 495 + 491 + 40 = 1026.

27. Minimum Spanning Trees

27.1 Kruskal's Algorithm

In Kruskal's algorithm, sort all edges of the given graph in increasing order. Then it keeps on adding new edges and nodes in the MST if the newly added edge does not form a cycle. It picks the minimum weighted edge at first and the maximum weighted edge at last.

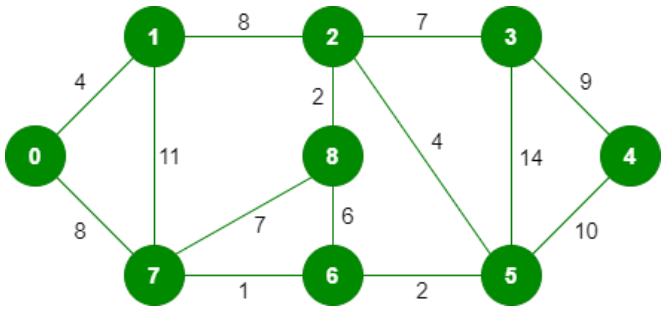
MST using Kruskal's algorithm:

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it.

3. Repeat step#2 until there are (V-1) edges in the spanning tree.

Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far.

Input: For the given graph G find the minimum cost spanning tree.



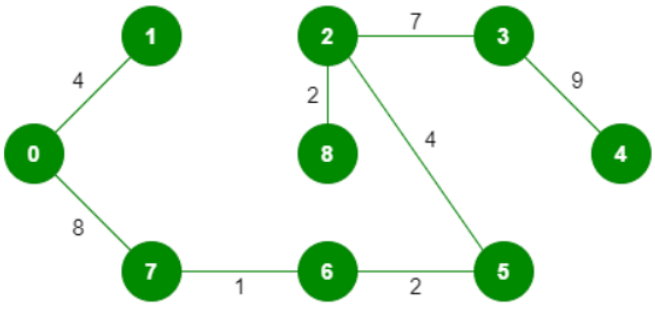
The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.

After sorting:

Weight	Source	Destination
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6
7	2	3
7	7	8
8	0	7
8	1	2
9	3	4
10	5	4
11	1	7
14	3	5

Now pick all edges one by one from the sorted list of edges.

Output:



Output: Following are the edges in the constructed MST

2 -- 3 == 4

0 -- 3 == 5

0 -- 1 == 10

Minimum Cost Spanning Tree: 19

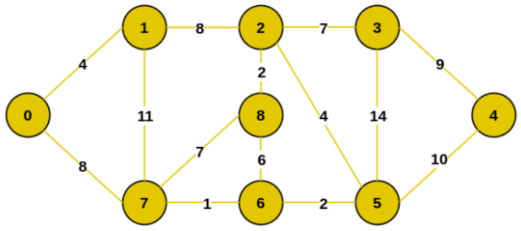
27.2 Prim's Algorithm

The Prim's algorithm starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, and the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

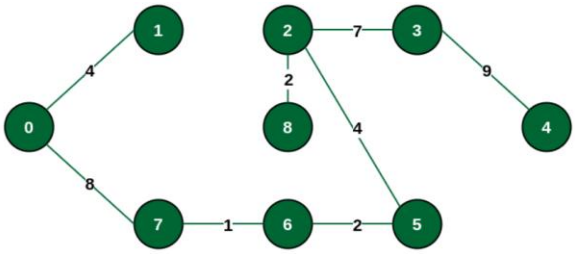
Prim's Algorithm:
The working of Prim's algorithm can be described by using the following steps:

1. Determine an arbitrary vertex as the starting vertex of the MST.
2. Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).
3. Find edges connecting any tree vertex with the fringe vertices.
4. Find the minimum among these edges.
5. Add the chosen edge to the MST if it does not form any cycle.
6. Return the MST and exit

Input: For the given graph G find the minimum cost spanning tree.



Output: The final structure of the MST is as follows and the weight of the edges of the MST is (4 + 8 + 1 + 2 + 4 + 2 + 7 + 9) = 37.



Output:

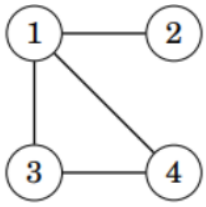
Edge	Weight
0 - 1	2
1 - 2	3
0 - 3	6
1 - 4	5

27.3 Total Number of Spanning Trees in a Graph

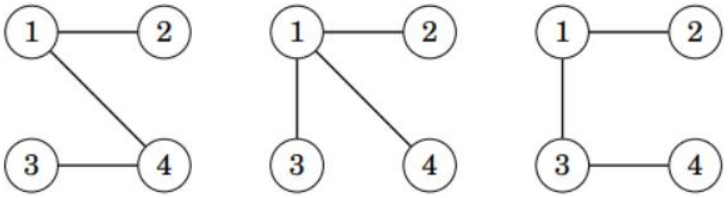
If a graph is a complete graph with n vertices, then total number of spanning trees is $n^{(n-2)}$ where n is the number of nodes in the graph. In complete graph, the task is equal to counting different labeled trees with n nodes for which have Cayley's formula.

Laplacian matrix:
A Laplacian matrix L, where L[i, i] is the degree of node i and L[i, j] = −1 if there is an edge between nodes i and j, and otherwise L[i, j] = 0.

Kirchhoff's theorem provides a way to calculate the number of spanning trees for a given graph as a determinant of a special matrix. Consider the following graph,



All possible spanning trees are as follows:



In order to calculate the number of spanning trees, construct a Laplacian matrix L, where L[i, i] is the degree of node i and L[i, j] = −1 if there is an edge between nodes i and j, and otherwise L[i, j] = 0.
for the above graph, The Laplacian matrix will look like this

$$L = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 2 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix}$$

The number of spanning trees equals the determinant of a matrix.

The Determinant of a matrix that can be obtained when we remove any row and any column from L.
For example, if we remove the first row and column, the result will be,

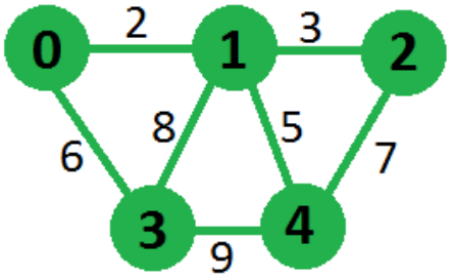
$$\det\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}\right) = 3.$$

The determinant is always the same, regardless of which row and column we remove from L.

27.4 Minimum Product Spanning Tree

A minimum product spanning tree for a weighted, connected, and undirected graph is a spanning tree with a weight product less than or equal to the weight product of every other spanning tree. The weight product of a spanning tree is the product of weights corresponding to each edge of the spanning tree. All weights of the given graph will be positive for simplicity.

Input:



Output: Minimum Product that we can obtain is 180 for above graph by choosing edges 0-1, 1-2, 0-3 and 1-4

This problem can be solved using standard minimum spanning tree algorithms like Kruskal and prim's algorithm, but we need to modify our graph to use these algorithms. Minimum spanning tree algorithms tries to minimize the total sum of weights, here we need to minimize the total product of weights. We can use the property of logarithms to overcome this problem.

log(w1* w2 * w3 * …. * wN) = log(w1) + log(w2) + log(w3) ….. + log(wN)

We can replace each weight of the graph by its log value, then we apply any minimum spanning tree algorithm which will try to minimize the sum of log(wi) which in turn minimizes the weight product.

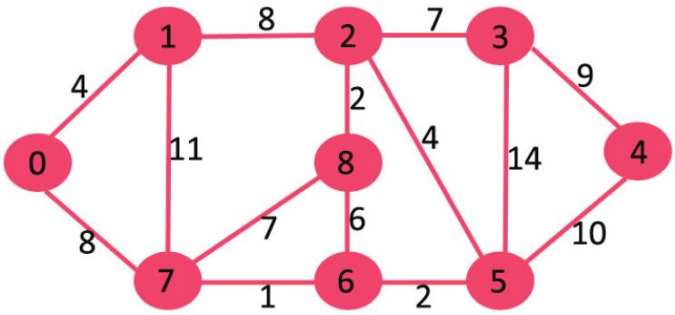
27.5 Reverse Delete Algorithm for Minimum Spanning Tree

In Reverse Delete algorithm, we sort all edges in decreasing order of their weights. After sorting, we one by one pick edges in decreasing order. We include current picked edge if excluding current edge causes disconnection in current graph. The main idea is delete edge if its deletion does not lead to disconnection of graph.

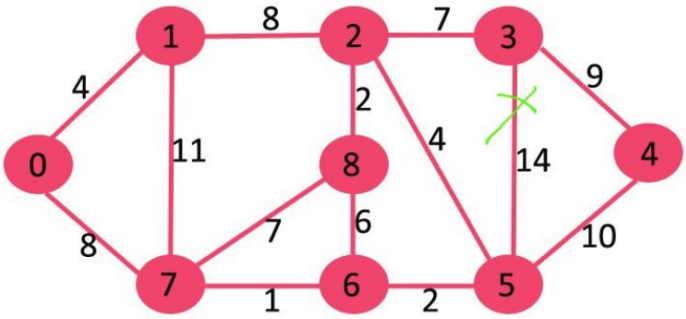
Algorithm:

1. Sort all edges of graph in non-increasing order of edge weights.
2. Initialize MST as original graph and remove extra edges using step 3.
3. Pick highest weight edge from remaining edges and check if deleting the edge disconnects the graph or not.
   If disconnects, then we don't delete the edge.
   Else we delete the edge and continue.

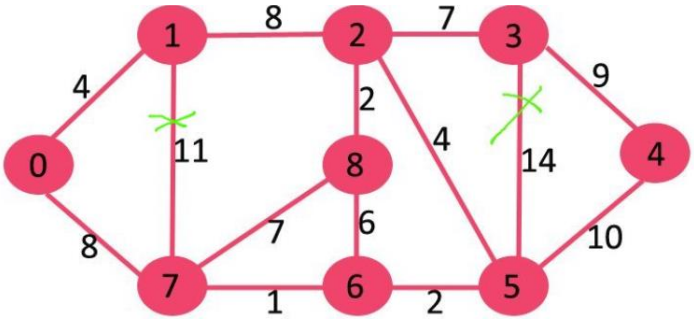
Input: Consider the graph below



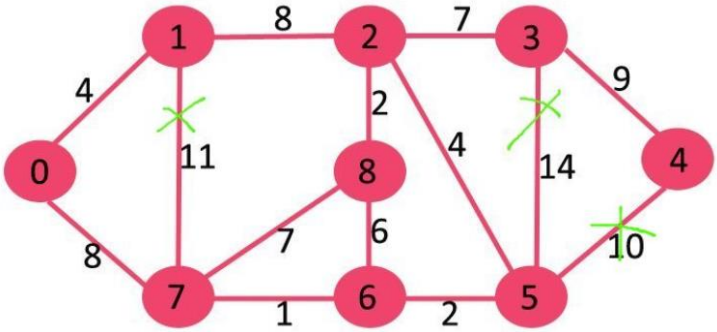
If we delete highest weight edge of weight 14, graph doesn't become disconnected, so we remove it.



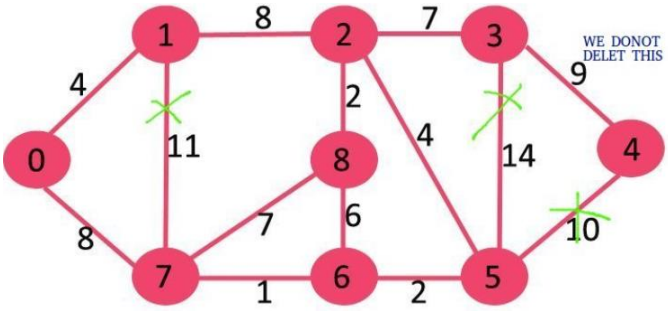
Next we delete 11 as deleting it doesn't disconnect the graph.



Next we delete 10 as deleting it doesn't disconnect the graph.



Next is 9. We cannot delete 9 as deleting it causes disconnection.



We continue this way and following edges remain in final MST.

Edges in MST

(3, 4)
(0, 7)
(2, 3)
(2, 5)
(0, 1)
(5, 6)
(2, 8)
(6, 7)

Output: Yes
Explanation: The diagram clearly shows a cycle 0 -> 2 -> 0

28. Segment Trees

28.1 Queue Reconstruction by Height

You are given an array of people, people, which are the attributes of some people in a queue (not necessarily in order). Each people[i] = [h_i, k_i] represents the i^{th} person of height h_i with **exactly** k_i other people in front who have a height greater than or equal to h_i.

Reconstruct and return the queue that is represented by the input array people. The returned queue should be formatted as an array queue, where queue[j] = [h_j, k_j] is the attributes of the j^{th} person in the queue (queue[0] is the person at the front of the queue).

Input: people = [[7, 0], [4, 4], [7, 1], [5, 0], [6, 1], [5, 2]]

Output: [[5, 0], [7, 0], [5, 2], [6, 1], [4, 4], [7, 1]]

Explanation:

Person 0 has height 5 with no other people taller or the same height in front.

Person 1 has height 7 with no other people taller or the same height in front.

Person 2 has height 5 with two persons taller or the same height in front, which is person 0 and 1.

Person 3 has height 6 with one person taller or the same height in front, which is person 1.

Person 4 has height 4 with four people taller or the same height in front, which are people 0, 1, 2, and 3.

Person 5 has height 7 with one person taller or the same height in front, which is person 1.

Hence [[5, 0], [7, 0], [5, 2], [6, 1], [4, 4], [7, 1]] is the reconstructed queue.

Input: people = [[6, 0], [5, 0], [4, 0], [3, 2], [2, 2], [1, 4]]

Output: [[4, 0], [5, 0], [2, 2], [3, 2], [1, 4], [6, 0]]

28.2 Number of Longest Increasing Subsequence

Given an integer array nums, return *the number of longest increasing subsequences.*

Notice that the sequence has to be **strictly** increasing.

Input: nums = [1, 3, 5, 4, 7]

Output: 2

Explanation: The two longest increasing subsequences are [1, 3, 4, 7] and [1, 3, 5, 7].

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

Output: 5

Explanation: The length of the longest increasing subsequence is 1, and there are 5 increasing subsequences of length 1, so output 5.

28.3 Longest Uploaded Prefix

You are given a stream of n videos, each represented by a **distinct** number from 1 to n that you need to "upload" to a server. You need to implement a data structure that calculates the length of the **longest uploaded prefix** at various points in the upload process.

We consider i to be an uploaded prefix if all videos in the range 1 to i (**inclusive**) have been uploaded to the server. The longest uploaded prefix is the **maximum** value of i that satisfies this definition.

Implement the LUPrefix class:

- LUPrefix(int n) Initializes the object for a stream of n videos.
- void upload(int video) Uploads video to the server.
- int longest() Returns the length of the **longest uploaded prefix** defined above.

Input

["LUPrefix", "upload", "longest", "upload", "longest", "upload", "longest"]

[[4], [3], [], [1], [], [2], []]

Output

[null, null, 0, null, 1, null, 3]

Explanation

LUPrefix server = new LUPrefix(4); // Initialize a stream of 4 videos.

server.upload(3); // Upload video 3.

server.longest(); // Since video 1 has not been uploaded yet, there is no prefix.

// So, we return 0.

server.upload(1); // Upload video 1.

server.longest(); // The prefix [1] is the longest uploaded prefix, so we return 1.

server.upload(2); // Upload video 2.

server.longest(); // The prefix [1,2,3] is the longest uploaded prefix, so we return 3.

28.4 Falling Squares

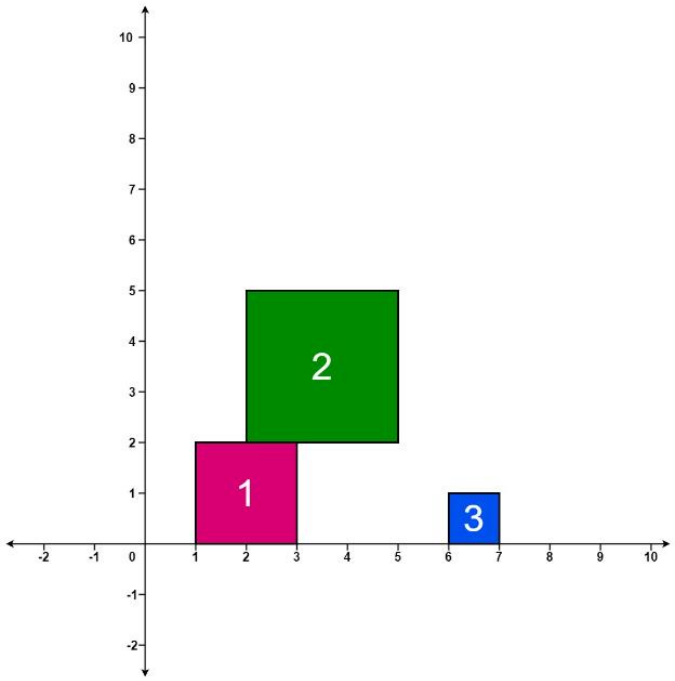
There are several squares being dropped onto the X-axis of a 2D plane.

You are given a 2D integer array positions where positions[i] = [$left_i$, $sideLength_i$] represents the i^{th} square with a side length of $sideLength_i$ that is dropped with its left edge aligned with X-coordinate $left_i$.

Each square is dropped one at a time from a height above any landed squares. It then falls downward (negative Y direction) until it either lands **on the top side of another square** or **on the X-axis**. A square brushing the left/right side of another square does not count as landing on it. Once it lands, it freezes in place and cannot be moved.

After each square is dropped, you must record the **height of the current tallest stack of squares**.

Return an integer array ans where ans[i] represents the height described above after dropping the i^{th} square.



Input: positions = [[1, 2], [2, 3], [6, 1]]

Output: [2, 5, 5]

Explanation:

After the first drop, the tallest stack is square 1 with a height of 2.

After the second drop, the tallest stack is squares 1 and 2 with a height of 5.

After the third drop, the tallest stack is still squares 1 and 2 with a height of 5.

Thus, we return an answer of [2, 5, 5].

Input: positions = [[100,100], [200,100]]

Output: [100,100]

Explanation:

After the first drop, the tallest stack is square 1 with a height of 100.

After the second drop, the tallest stack is either square 1 or square 2, both with heights of 100.

Thus, we return an answer of [100, 100].

Note that square 2 only brushes the right side of square 1, which does not count as landing on it.

28.5 My Calendar I

You are implementing a program to use as your calendar. We can add a new event if adding the event will not cause a **double booking**.

A **double booking** happens when two events have some non-empty intersection (i.e., some moment is common to both events.).

The event can be represented as a pair of integers start and end that represents a booking on the half-open interval [start, end), the range of real numbers x such that start <= x < end.

Implement the MyCalendar class:

- MyCalendar() Initializes the calendar object.
- boolean book(int start, int end) Returns true if the event can be added to the calendar successfully without causing a **double booking**. Otherwise, return false and do not add the event to the calendar.

Input: ["MyCalendar", "book", "book", "book"]

[[], [10, 20], [15, 25], [20, 30]]

Output: [null, true, false, true]

Explanation:

MyCalendar myCalendar = new MyCalendar();

myCalendar.book(10, 20); // return True

myCalendar.book(15, 25); // return False, It can not be booked because time 15 is already booked by another event.

myCalendar.book(20, 30); // return True, The event can be booked, as the first event takes every time less than 20, but not including 20.

29. Paths and Circuits

29.1 Eulerian Paths

An Eulerian path1 is a path that goes exactly once through each edge of the graph. Given an adjacency matrix representation of an undirected graph. Find if there is any Eulerian Path in the graph. If there is no path print "No Solution". If there is any path print the path.

Input: [[0, 1, 0, 0, 1],

[1, 0, 1, 1, 0],

[0, 1, 0, 1, 0],

[0, 1, 1, 0, 0],

[1, 0, 0, 0, 0]]

Output: 5 -> 1 -> 2 -> 4 -> 3 -> 2

Input: [[0, 1, 0, 1, 1],

[1, 0, 1, 0, 1],

[0, 1, 0, 1, 1],

[1, 1, 1, 0, 0],

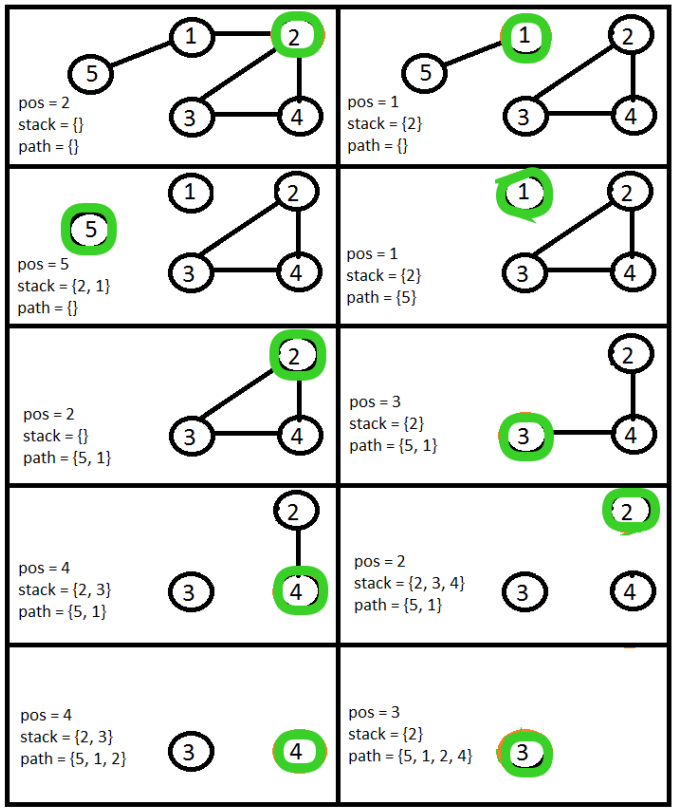
[1, 0, 1, 0, 0]]

Output: "No Solution"

- The base case of this problem is if the number of vertices with an odd number of edges(i.e. odd degree) is greater than 2 then there is no Eulerian path.
- If it has the solution and all the nodes have an even number of edges then we can start our path from any of the nodes.
- If it has the solution and exactly two vertices have an odd number of edges then we have to start our path from one of these two vertices.
- There will not be the case where exactly one vertex has an odd number of edges, as there is an even number of edges in total.

The process to find the Path:

- First, take an empty stack and an empty path.
- If all the vertices have an even number of edges then start from any of them. If two of the vertices have an odd number of edges then start from one of them. Set variable current to this starting vertex.
- If the current vertex has at least one adjacent node then first discover that node and then discover the current node by backtracking. To do so add the current node to stack, remove the edge between the current node and neighbor node, set current to the neighbor node.
- If the current node has not any neighbor then add it to the path and pop stack set current to popped vertex.
- Repeat process 3 and 4 until the stack is empty and the current node has not any neighbor.



In next iteration: pos = 2, stack = {}, path = {5, 1, 2, 4, 3}
And in last iteration: stack = {}, path = {5, 1, 2, 4, 3, 2}

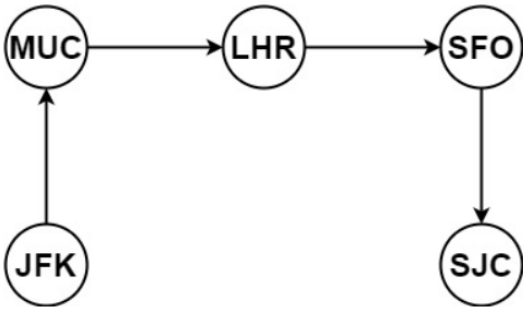
29.2 Reconstruct Itinerary

You are given a list of airline tickets where tickets[i] = [$from_i$, to_i] represent the departure and the arrival airports of one flight. Reconstruct the itinerary in order and return it.

All of the tickets belong to a man who departs from "JFK", thus, the itinerary must begin with "JFK". If there are multiple valid itineraries, you should return the itinerary that has the smallest lexical order when read as a single string.

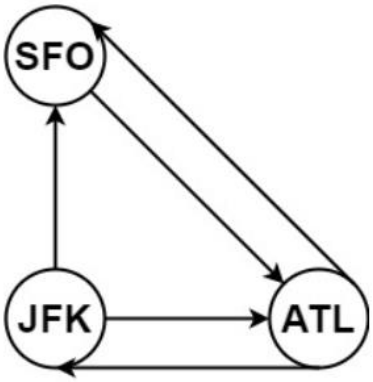
- For example, the itinerary ["JFK", "LGA"] has a smaller lexical order than ["JFK", "LGB"].

You may assume all tickets form at least one valid itinerary. You must use all the tickets once and only once.



Input: tickets = [["MUC","LHR"], ["JFK","MUC"], ["SFO","SJC"], ["LHR","SFO"]]

Output: ["JFK","MUC","LHR","SFO","SJC"]



Input: tickets = [["JFK", "SFO"], ["JFK", "ATL"], ["SFO", "ATL"], ["ATL", "JFK"], ["ATL", "SFO"]]

Output: ["JFK", "ATL", "JFK", "SFO", "ATL", "SFO"]

Explanation: Another possible reconstruction is ["JFK", "SFO", "ATL", "JFK", "ATL", "SFO"] but it is larger in lexical order.

29.3 Cracking the Safe

There is a safe protected by a password. The password is a sequence of n digits where each digit can be in the range [0, k - 1].

The safe has a peculiar way of checking the password. When you enter in a sequence, it checks the **most recent** n **digits** that were entered each time you type a digit.

- For example, the correct password is "345" and you enter in "012345":
    - After typing 0, the most recent 3 digits is "0", which is incorrect.
    - After typing 1, the most recent 3 digits is "01", which is incorrect.

- After typing 2, the most recent 3 digits is "012", which is incorrect.
- After typing 3, the most recent 3 digits is "123", which is incorrect.
- After typing 4, the most recent 3 digits is "234", which is incorrect.
- After typing 5, the most recent 3 digits is "345", which is correct and the safe unlocks.

Return any string of **minimum length** that will unlock the safe **at some point** of entering it.

Input: n = 1, k = 2

Output: "10"

Explanation: The password is a single digit, so enter each digit. "01" would also unlock the safe.

Input: n = 2, k = 2

Output: "01100"

Explanation: For each possible password:

- "00" is typed in starting from the 4^{th} digit.

- "01" is typed in starting from the 1^{st} digit.

- "10" is typed in starting from the 3^{rd} digit.

- "11" is typed in starting from the 2^{nd} digit.

Thus "01100" will unlock the safe. "10011", and "11001" would also unlock the safe.

29.4 Valid Arrangement of Pairs

You are given a **0-indexed** 2D integer array pairs where pairs[i] = [$start_i$, end_i]. An arrangement of pairs is **valid** if for every index i where 1 <= i < pairs.length, we have end_{i-1} == $start_i$.

Return **any** valid arrangement of pairs.

Note: The inputs will be generated such that there exists a valid arrangement of pairs.

Input: pairs = [[5, 1], [4, 5], [11, 9], [9, 4]]

Output: [[11, 9], [9, 4], [4, 5], [5, 1]]

Explanation:

This is a valid arrangement since end_{i-1} always equals $start_i$.

$end_0 = 9 == 9 = start_1$

$end_1 = 4 == 4 = start_2$

$end_2 = 5 == 5 = start_3$

Input: pairs = [[1, 3], [3, 2], [2, 1]]

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

Explanation:

This is a valid arrangement since end_{i-1} always equals $start_i$.

$end_0 = 3 == 3 = start_1$

$end_1 = 2 == 2 = start_2$

The arrangements [[2, 1], [1, 3], [3, 2]] and [[3, 2], [2, 1], [1, 3]] are also valid.

Input: pairs = [[1, 2], [1, 3], [2, 1]]

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

Explanation:

This is a valid arrangement since end_{i-1} always equals $start_i$.

$end_0 = 2 == 2 = start_1$

$end_1 = 1 == 1 = start_2$

29.5 Find the Shortest Superstring

Given an array of strings words, return *the smallest string that contains each string in* words *as a substring*. If there are multiple valid strings of the smallest length, return **any of them**.

You may assume that no string in words is a substring of another string in words.

Input: words = ["alex","loves","leetcode"]

Output: "alexlovesleetcode"

Explanation: All permutations of "alex","loves","leetcode" would also be accepted.

Input: words = ["catg","ctaagt","gcta","ttca","atgcatc"]

Output: "gctaagttcatgcatc"

30. Two Heaps

30.1 Process Tasks using Servers

You are given two **0-indexed** integer arrays servers and tasks of lengths n and m respectively. servers[i] is the **weight** of the i^{th} server, and tasks[j] is the **time needed** to process the j^{th} task **in seconds**.

Tasks are assigned to the servers using a **task queue**. Initially, all servers are free, and the queue is **empty**.

At second j, the j^{th} task is **inserted** into the queue (starting with the 0^{th} task being inserted at second 0). As long as there are free servers and the queue is not empty, the task in the front of the queue will be assigned to a free server with the **smallest weight**, and in case of a tie, it is assigned to a free server with the **smallest index**.

If there are no free servers and the queue is not empty, we wait until a server becomes free and immediately assign the next task. If multiple servers become free at the same time, then multiple tasks from the queue will be assigned **in order of insertion** following the weight and index priorities above.

A server that is assigned task j at second t will be free again at second t + tasks[j].

Build an array ans of length m, where ans[j] is the **index** of the server the j^{th} task will be assigned to.

Return the array ans.

Input: servers = [3, 3, 2], tasks = [1, 2, 3, 2, 1, 2]

Output: [2, 2, 0, 2, 1, 2]

Explanation: Events in chronological order go as follows:

- At second 0, task 0 is added and processed using server 2 until second 1.

- At second 1, server 2 becomes free. Task 1 is added and processed using server 2 until second 3.

- At second 2, task 2 is added and processed using server 0 until second 5.

- At second 3, server 2 becomes free. Task 3 is added and processed using server 2 until second 5.

- At second 4, task 4 is added and processed using server 1 until second 5.

- At second 5, all servers become free. Task 5 is added and processed using server 2 until second 7.

Input: servers = [5, 1, 4, 3, 2], tasks = [2, 1, 2, 4, 5, 2, 1]

Output: [1, 4, 1, 4, 1, 3, 2]

Explanation: Events in chronological order go as follows:

- At second 0, task 0 is added and processed using server 1 until second 2.

- At second 1, task 1 is added and processed using server 4 until second 2.

- At second 2, servers 1 and 4 become free. Task 2 is added and processed using server 1 until second 4.

- At second 3, task 3 is added and processed using server 4 until second 7.

- At second 4, server 1 becomes free. Task 4 is added and processed using server 1 until second 9.

- At second 5, task 5 is added and processed using server 3 until second 7.

- At second 6, task 6 is added and processed using server 2 until second 7.

30.2 Median of Two Sorted Arrays

Given two sorted arrays nums1 and nums2 of size m and n respectively, return **the median** of the two sorted arrays.

The overall run time complexity should be O(log (m+n)).

Input: nums1 = [1, 3], nums2 = [2]

Output: 2.00000

Explanation: merged array = [1, 2, 3] and median is 2.

Input: nums1 = [1, 2], nums2 = [3,4]

Output: 2.50000

Explanation: merged array = [1, 2, 3, 4] and median is (2 + 3) / 2 = 2.5.

30.3 Find Median from Data Stream

The **median** is the middle value in an ordered integer list. If the size of the list is even, there is no middle value, and the median is the mean of the two middle values.

- For example, for arr = [2, 3, 4], the median is 3.
- For example, for arr = [2, 3], the median is (2 + 3) / 2 = 2.5.

Implement the MedianFinder class:

- MedianFinder() initializes the MedianFinder object.
- void addNum(int num) adds the integer num from the data stream to the data structure.
- double findMedian() returns the median of all elements so far. Answers within 10^{-5} of the actual answer will be accepted.

Input: ["MedianFinder", "addNum", "addNum", "findMedian", "addNum", "findMedian"]

[[], [1], [2], [], [3], []]

Output: [null, null, null, 1.5, null, 2.0]

Explanation

MedianFinder medianFinder = new MedianFinder();

medianFinder.addNum(1); // arr = [1]

medianFinder.addNum(2); // arr = [1, 2]

medianFinder.findMedian(); // return 1.5 (i.e., (1 + 2) / 2)

medianFinder.addNum(3); // arr[1, 2, 3]

medianFinder.findMedian(); // return 2.0

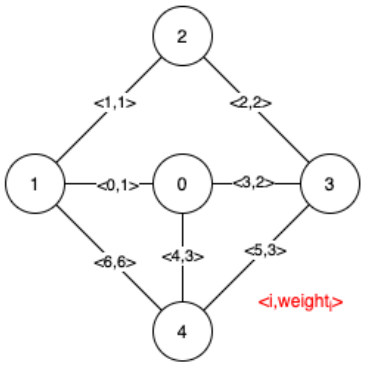
31. Strong Connectivity

31.1 Find Critical and Pseudo-Critical Edges in Minimum Spanning Tree

Given a weighted undirected connected graph with n vertices numbered from 0 to n - 1, and an array edges where edges[i] = [a_i, b_i, $weight_i$] represents a bidirectional and weighted edge between nodes a_i and b_i. A minimum spanning tree (MST) is a subset of the graph's edges that connects all vertices without cycles and with the minimum possible total edge weight.

Find all the critical and pseudo-critical edges in the given graph's minimum spanning tree (MST). An MST edge whose deletion from the graph would cause the MST weight to increase is called a critical edge. On the other hand, a pseudo-critical edge is that which can appear in some MSTs but not all.

Note that you can return the indices of the edges in any order.

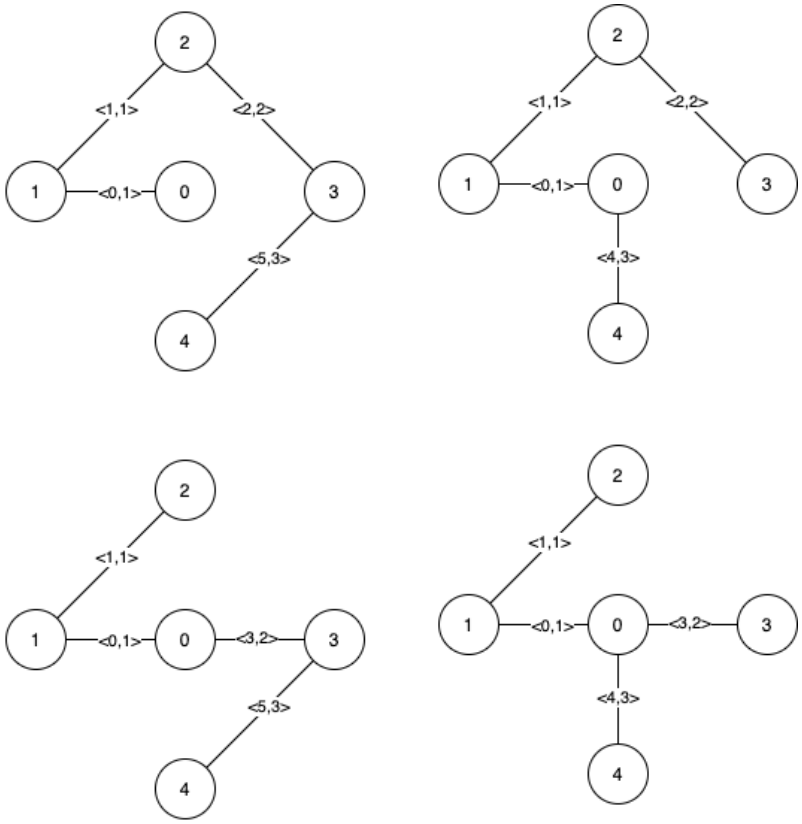


Input: n = 5, edges = [[0, 1, 1], [1, 2, 1], [2, 3, 2], [0, 3, 2], [0, 4, 3], [3, 4, 3], [1, 4, 6]]

Output: [[0, 1], [2, 3, 4, 5]]

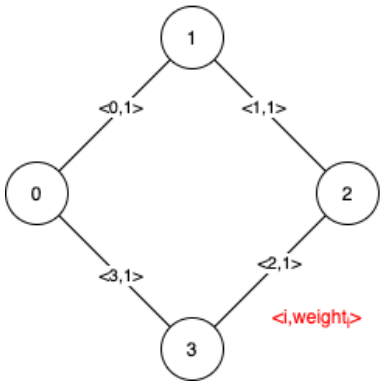
Explanation: The figure above describes the graph.

The following figure shows all the possible MSTs:



Notice that the two edges 0 and 1 appear in all MSTs, therefore they are critical edges, so we return them in the first list of the output.

The edges 2, 3, 4, and 5 are only part of some MSTs, therefore they are considered pseudo-critical edges. We add them to the second list of the output.



Input: n = 4, edges = [[0, 1, 1], [1, 2, 1], [2, 3, 1], [0, 3, 1]]

Output: [[], [0, 1, 2, 3]]

Explanation: We can observe that since all 4 edges have equal weight, choosing any 3 edges from the given 4 will yield an MST. Therefore all 4 edges are pseudo-critical.

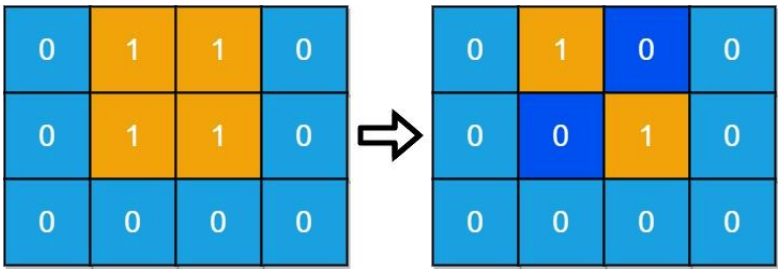
31.2 Minimum Number of Days to Disconnect Island

You are given an m x n binary grid grid where 1 represents land and 0 represents water. An **island** is a maximal **4-directionally** (horizontal or vertical) connected group of 1's.

The grid is said to be **connected** if we have **exactly one island**, otherwise is said **disconnected**.

In one day, we are allowed to change **any** single land cell (1) into a water cell (0).

Return the minimum number of days to disconnect the grid.

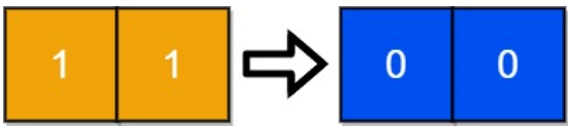


Input: grid = [[0, 1, 1, 0], [0, 1, 1, 0], [0, 0, 0, 0]]

Output: 2

Explanation: We need at least 2 days to get a disconnected grid.

Change land grid[1][1] and grid[0][2] to water and get 2 disconnected island.



Input: grid = [[1, 1]]

Output: 2

Explanation: Grid of full water is also disconnected ([[1, 1]] -> [[0, 0]]), 0 islands.

31.3 Minimum Edge Weight Equilibrium Queries in a Tree

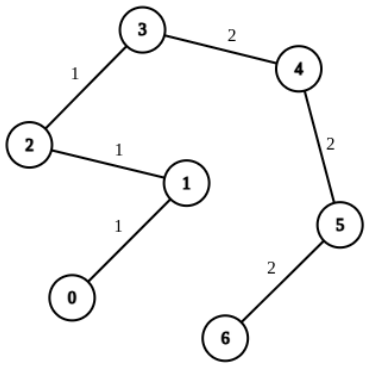
There is an undirected tree with n nodes labeled from 0 to n - 1. You are given the integer n and a 2D integer array edges of length n - 1, where edges[i] = [u_i, v_i, w_i] indicates that there is an edge between nodes u_i and v_i with weight w_i in the tree.

You are also given a 2D integer array queries of length m, where queries[i] = [a_i, b_i]. For each query, find the **minimum number of operations** required to make the weight of every edge on the path from a_i to b_i equal. In one operation, you can choose any edge of the tree and change its weight to any value.

Note that:

- Queries are **independent** of each other, meaning that the tree returns to its **initial state** on each new query.
- The path from a_i to b_i is a sequence of **distinct** nodes starting with node a_i and ending with node b_i such that every two adjacent nodes in the sequence share an edge in the tree.

Return an array answer of length m where answer[i] is the answer to the i^{th} query.



Input: n = 7, edges = [[0,1,1],[1,2,1],[2,3,1],[3,4,2],[4,5,2],[5,6,2]], queries = [[0,3],[3,6],[2,6],[0,6]]

Output: [0, 0, 1, 3]

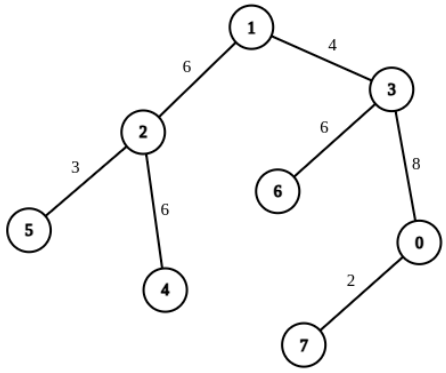
Explanation: In the first query, all the edges in the path from 0 to 3 have a weight of 1. Hence, the answer is 0.

In the second query, all the edges in the path from 3 to 6 have a weight of 2. Hence, the answer is 0.

In the third query, we change the weight of edge [2, 3] to 2. After this operation, all the edges in the path from 2 to 6 have a weight of 2. Hence, the answer is 1.

In the fourth query, we change the weights of edges [0, 1], [1, 2] and [2, 3] to 2. After these operations, all the edges in the path from 0 to 6 have a weight of 2. Hence, the answer is 3.

For each queries[i], it can be shown that answer[i] is the minimum number of operations needed to equalize all the edge weights in the path from a_i to b_i.



Input: n = 8, edges = [[1, 2, 6], [1, 3, 4], [2, 4, 6], [2, 5, 3], [3, 6, 6], [3, 0, 8], [7, 0, 2]], queries = [[4, 6], [0, 4], [6, 5], [7, 4]]

Output: [1, 2, 2, 3]

Explanation: In the first query, we change the weight of edge [1, 3] to 6. After this operation, all the edges in the path from 4 to 6 have a weight of 6. Hence, the answer is 1.

In the second query, we change the weight of edges [0, 3] and [3, 1] to 6. After these operations, all the edges in the path from 0 to 4 have a weight of 6. Hence, the answer is 2.

In the third query, we change the weight of edges [1, 3] and [5, 2] to 6. After these operations, all the edges in the path from 6 to 5 have a weight of 6. Hence, the answer is 2.

In the fourth query, we change the weights of edges [0, 7], [0, 3] and [1, 3] to 6. After these operations, all the edges in the path from 7 to 4 have a weight of 6. Hence, the answer is 3.

For each queries[i], it can be shown that answer[i] is the minimum number of operations needed to equalize all the edge weights in the path from a_i to b_i.

32. Number Theory

32.1 Count Primes

Given an integer n, return the number of prime numbers that are strictly less than n.

Input: n = 10

Output: 4

Explanation: There are 4 prime numbers less than 10, they are 2, 3, 5, 7.

Input: n = 0

Output: 0

Input: n = 1

Output: 0

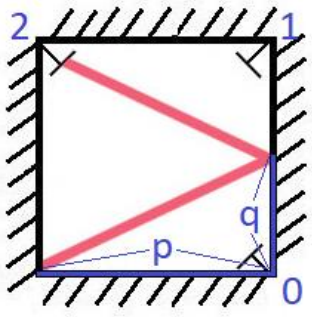
32.2 Mirror Reflection

There is a special square room with mirrors on each of the four walls. Except for the southwest corner, there are receptors on each of the remaining corners, numbered 0, 1, and 2.

The square room has walls of length p and a laser ray from the southwest corner first meets the east wall at a distance q from the 0^{th} receptor.

Given the two integers p and q, return the number of the receptor that the ray meets first.

The test cases are guaranteed so that the ray will meet a receptor eventually.



Input: p = 2, q = 1

Output: 2

Explanation: The ray meets receptor 2 the first time it gets reflected back to the left wall.

Input: p = 3, q = 1

Output: 1

32.3 Largest Component Size by Common Factor

You are given an integer array of unique positive integers nums. Consider the following graph:

- There are nums.length nodes, labeled nums[0] to nums[nums.length - 1],

- There is an undirected edge between nums[i] and nums[j] if nums[i] and nums[j] share a common factor greater than 1.

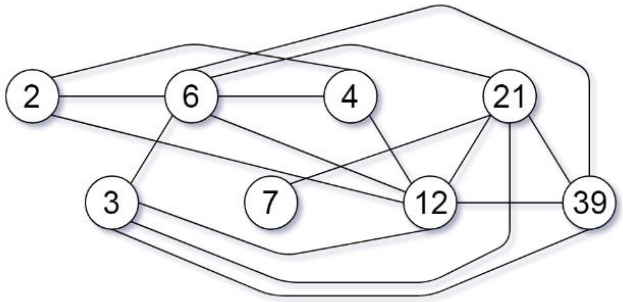
Return the size of the largest connected component in the graph.

Input: nums = [4, 6, 15, 35]

Output: 4

Input: nums = [20, 50, 9, 63]

Output: 2



Input: nums = [2, 3, 6, 7, 4, 12, 21, 39]

Output: 8

32.4 Simplified Fractions

Given an integer n, return a list of all **simplified** fractions between 0 and 1 (exclusive) such that the denominator is less-than-or-equal-to n. You can return the answer in **any order**.

Input: n = 2

Output: ["1/2"]

Explanation: "1/2" is the only unique fraction with a denominator less-than-or-equal-to 2.

Input: n = 3

Output: ["1/2","1/3","2/3"]

Input: n = 4

Output: ["1/2","1/3","1/4","2/3","3/4"]

Explanation: "2/4" is not a simplified fraction because it can be simplified to "1/2".

32.5 The kth Factor of n

You are given two positive integers n and k. A factor of an integer n is defined as an integer i where n % i == 0.

Consider a list of all factors of n sorted in **ascending order**, return *the* kth *factor* in this list or return -1 if n has less than k factors.

Input: n = 12, k = 3

Output: 3

Explanation: Factors list is [1, 2, 3, 4, 6, 12], the 3rd factor is 3.

Input: n = 7, k = 2

Output: 7

Explanation: Factors list is [1, 7], the 2nd factor is 7.

Input: n = 4, k = 4

Output: -1

Explanation: Factors list is [1, 2, 4], there is only 3 factors. We should return -1.

32.6 Number of Different Subsequences GCDs

You are given an array nums that consists of positive integers.

The **GCD** of a sequence of numbers is defined as the greatest integer that divides **all** the numbers in the sequence evenly.

- For example, the GCD of the sequence [4, 6, 16] is 2.

A **subsequence** of an array is a sequence that can be formed by removing some elements (possibly none) of the array.

- For example, [2, 5, 10] is a subsequence of [1, 2, 1, **2**, 4, 1, **5**, **10**].

Return the **number** of **different** GCDs among all **non-empty** subsequences of nums.

Example 1:

Subsequence	GCD
[6]	6
[10]	10
[3]	3
[6,10]	2
[6,3]	3
[10,3]	1
[6,10,3]	1

Input: nums = [6, 10, 3]

Output: 5

Explanation: The figure shows all the non-empty subsequences and their GCDs.

The different GCDs are 6, 10, 3, 2, and 1.

Input: nums = [5, 15, 40, 5, 6]

Output: 7

32.7 Find Greatest Common Divisor of Array

Given an integer array nums, return the **greatest common divisor** of the smallest number and largest number in nums.

The **greatest common divisor** of two numbers is the largest positive integer that evenly divides both numbers.

Input: nums = [2, 5, 6, 9, 10]

Output: 2

Explanation:

The smallest number in nums is 2.

The largest number in nums is 10.

The greatest common divisor of 2 and 10 is 2.

Input: nums = [7, 5, 6, 8, 3]

Output: 1

Explanation:

The smallest number in nums is 3.

The largest number in nums is 8.

The greatest common divisor of 3 and 8 is 1.

Input: nums = [3, 3]

Output: 3

Explanation:

The smallest number in nums is 3.

The largest number in nums is 3.

The greatest common divisor of 3 and 3 is 3.

32.8 Number of Pairs of Interchangeable Rectangles

You are given n rectangles represented by a **0-indexed** 2D integer array rectangles, where rectangles[i] = [$width_i$, $height_i$] denotes the width and height of the i^{th} rectangle.

Two rectangles i and j (i < j) are considered **interchangeable** if they have the **same** width-to-height ratio. More formally, two rectangles are **interchangeable** if $width_i/height_i == width_j/height_j$ (using decimal division, not integer division).

Return the **number** of pairs of **interchangeable** rectangles in rectangles.

Input: rectangles = [[4, 8], [3, 6], [10, 20], [15, 30]]

Output: 6

Explanation: The following are the interchangeable pairs of rectangles by index (0-indexed):

- Rectangle 0 with rectangle 1: 4/8 == 3/6.

- Rectangle 0 with rectangle 2: 4/8 == 10/20.

- Rectangle 0 with rectangle 3: 4/8 == 15/30.

- Rectangle 1 with rectangle 2: 3/6 == 10/20.

- Rectangle 1 with rectangle 3: 3/6 == 15/30.

- Rectangle 2 with rectangle 3: 10/20 == 15/30.

Input: rectangles = [[4, 5], [7, 8]]

Output: 0

Explanation: There are no interchangeable pairs of rectangles.

32.9 Replace Non-Coprime Numbers in Array

You are given an array of integers nums. Perform the following steps:

1. Find **any** two **adjacent** numbers in nums that are **non-coprime**.
2. If no such numbers are found, **stop** the process.
3. Otherwise, delete the two numbers and **replace** them with their **LCM (Least Common Multiple)**.
4. **Repeat** this process as long as you keep finding two adjacent non-coprime numbers.

Return the **final** modified array. It can be shown that replacing adjacent non-coprime numbers in **any** arbitrary order will lead to the same result.

The test cases are generated such that the values in the final array are **less than or equal** to 10^8.

Two values x and y are **non-coprime** if GCD(x, y) > 1 where GCD(x, y) is the **Greatest Common Divisor** of x and y.

Input: nums = [6, 4, 3, 2, 7, 6, 2]

Output: [12, 7, 6]

Explanation:

- (6, 4) are non-coprime with LCM(6, 4) = 12. Now, nums = [**12**, 3, 2, 7, 6, 2].

- (12, 3) are non-coprime with LCM(12, 3) = 12. Now, nums = [**12**, 2, 7, 6, 2].

- (12, 2) are non-coprime with LCM(12, 2) = 12. Now, nums = [**12**, 7, 6, 2].

- (6, 2) are non-coprime with LCM(6, 2) = 6. Now, nums = [12, 7, **6**].

There are no more adjacent non-coprime numbers in nums.

Thus, the final modified array is [12, 7, 6].

Note that there are other ways to obtain the same resultant array.

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

Output: [2, 1, 1, 3]

Explanation:

- (3, 3) are non-coprime with LCM(3, 3) = 3. Now, nums = [2, 2, 1, 1, **3**, 3].

- (3, 3) are non-coprime with LCM(3, 3) = 3. Now, nums = [2, 2, 1, 1, **3**].

- (2, 2) are non-coprime with LCM(2, 2) = 2. Now, nums = [**2**, 1, 1, 3].

There are no more adjacent non-coprime numbers in nums.

Thus, the final modified array is [2, 1, 1, 3].

Note that there are other ways to obtain the same resultant array.

32.10 Number of Subarrays with LCM Equal to K

Given an integer array nums and an integer k, return the number of **subarrays** of nums where the least common multiple of the subarray's elements is k.

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

The **least common multiple of an array** is the smallest positive integer that is divisible by all the array elements.

Input: nums = [3, 6, 2, 7, 1], k = 6

Output: 4

Explanation: The subarrays of nums where 6 is the least common multiple of all the subarray's elements are:

- [**3**, **6**, 2, 7, 1]

- [**3**, **6**, **2**, 7, 1]

- [3, **6**, 2, 7, 1]

- [3, **6**, **2**, 7, 1]

Input: nums = [3], k = 2

Output: 0

Explanation: There are no subarrays of nums where 2 is the least common multiple of all the subarray's elements.

33. Combinatorics

33.1 Vowels of All Substrings

Given a string word, return *the* ***sum of the number of vowels*** *(*'a', 'e', 'i', 'o', *and* 'u'*) in every substring of* word.

A **substring** is a contiguous (non-empty) sequence of characters within a string.

Note: Due to the large constraints, the answer may not fit in a signed 32-bit integer. Please be careful during the calculations.

Input: word = "aba"

Output: 6

Explanation:

All possible substrings are: "a", "ab", "aba", "b", "ba", and "a".

- "b" has 0 vowels in it

- "a", "ab", "ba", and "a" have 1 vowel each

- "aba" has 2 vowels in it

Hence, the total sum of vowels = 0 + 1 + 1 + 1 + 1 + 2 = 6.

Input: word = "abc"

Output: 3

Explanation:

All possible substrings are: "a", "ab", "abc", "b", "bc", and "c".

- "a", "ab", and "abc" have 1 vowel each

- "b", "bc", and "c" have 0 vowels each

Hence, the total sum of vowels = 1 + 1 + 1 + 0 + 0 + 0 = 3.

Input: word = "ltcd"

Output: 0

Explanation: There are no vowels in any substring of "ltcd".

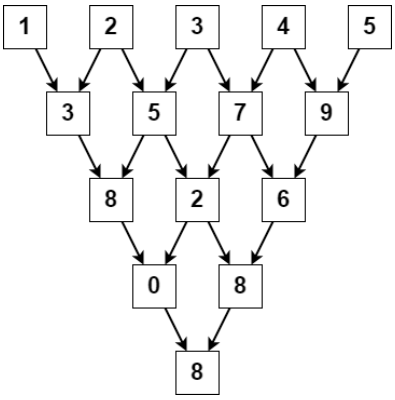
33.2 Find Triangular Sum of an Array

You are given a **0-indexed** integer array nums, where nums[i] is a digit between 0 and 9 (**inclusive**).

The **triangular sum** of nums is the value of the only element present in nums after the following process terminates:

1. Let nums comprise of n elements. If n == 1, **end** the process. Otherwise, **create** a new **0-indexed** integer array newNums of length n - 1.
2. For each index i, where 0 <= i < n - 1, **assign** the value of newNums[i] as (nums[i] + nums[i+1]) % 10, where % denotes modulo operator.
3. **Replace** the array nums with newNums.
4. **Repeat** the entire process starting from step 1.

Return the triangular sum of nums.



Input: nums = [1, 2, 3, 4, 5]

Output: 8

Explanation:

The above diagram depicts the process from which we obtain the triangular sum of the array.

Input: nums = [5]

Output: 5

Explanation:

Since there is only one element in nums, the triangular sum is the value of that element itself.

33.3 Number of Ways to Reach a Position after Exactly k Steps

You are given two **positive** integers startPos and endPos. Initially, you are standing at position startPos on an **infinite** number line. With one step, you can move either one position to the left, or one position to the right.

Given a positive integer k, return the number of **different** ways to reach the position endPos starting from startPos, such that you perform **exactly** k steps. Since the answer may be very large, return it **modulo** $10^9 + 7$.

Two ways are considered different if the order of the steps made is not exactly the same.

Note that the number line includes negative integers.

Input: startPos = 1, endPos = 2, k = 3

Output: 3

Explanation: We can reach position 2 from 1 in exactly 3 steps in three ways:

- 1 -> 2 -> 3 -> 2.

- 1 -> 2 -> 1 -> 2.

- 1 -> 0 -> 1 -> 2.

It can be proven that no other way is possible, so we return 3.

Input: startPos = 2, endPos = 5, k = 10

Output: 0

Explanation: It is impossible to reach position 5 from position 2 in exactly 10 steps.

33.4 Distribute Candies among Children II

You are given two positive integers n and limit.

Return the **total number** of ways to distribute n candies among 3 children such that no child gets more than limit candies.

Input: n = 5, limit = 2

Output: 3

Explanation: There are 3 ways to distribute 5 candies such that no child gets more than 2 candies: (1, 2, 2), (2, 1, 2) and (2, 2, 1).

Input: n = 3, limit = 3

Output: 10

Explanation: There are 10 ways to distribute 3 candies such that no child gets more than 3 candies: (0, 0, 3), (0, 1, 2), (0, 2, 1), (0, 3, 0), (1, 0, 2), (1, 1, 1), (1, 2, 0), (2, 0, 1), (2, 1, 0) and (3, 0, 0).

33.5 Poor Pigs

There are buckets buckets of liquid, where **exactly one** of the buckets is poisonous. To figure out which one is poisonous, you feed some number of (poor) pigs the liquid to see whether they will die or not. Unfortunately, you only have minutesToTest minutes to determine which bucket is poisonous.

You can feed the pigs according to these steps:

1. Choose some live pigs to feed.
2. For each pig, choose which buckets to feed it. The pig will consume all the chosen buckets simultaneously and will take no time. Each pig can feed from any number of buckets, and each bucket can be fed from by any number of pigs.
3. Wait for minutesToDie minutes. You may **not** feed any other pigs during this time.
4. After minutesToDie minutes have passed, any pigs that have been fed the poisonous bucket will die, and all others will survive.
5. Repeat this process until you run out of time.

Given buckets, minutesToDie, and minutesToTest, return the **minimum** number of pigs needed to figure out which bucket is poisonous within the allotted time.

Input: buckets = 4, minutesToDie = 15, minutesToTest = 15

Output: 2

Explanation: We can determine the poisonous bucket as follows:

At time 0, feed the first pig buckets 1 and 2, and feed the second pig buckets 2 and 3.

At time 15, there are 4 possible outcomes:

- If only the first pig dies, then bucket 1 must be poisonous.

- If only the second pig dies, then bucket 3 must be poisonous.

- If both pigs die, then bucket 2 must be poisonous.

- If neither pig dies, then bucket 4 must be poisonous.

Input: buckets = 4, minutesToDie = 15, minutesToTest = 30

Output: 2

Explanation: We can determine the poisonous bucket as follows:

At time 0, feed the first pig bucket 1, and feed the second pig bucket 2.

At time 15, there are 2 possible outcomes:

- If either pig dies, then the poisonous bucket is the one it was fed.

- If neither pig dies, then feed the first pig bucket 3, and feed the second pig bucket 4.

At time 30, one of the two pigs must die, and the poisonous bucket is the one it was fed.

33.6 Number of Music Playlists

Your music player contains n different songs. You want to listen to goal songs (not necessarily different) during your trip. To avoid boredom, you will create a playlist so that:

- Every song is played **at least once**.
- A song can only be played again only if k other songs have been played.

Given n, goal, and k, return the number of possible playlists that you can create. Since the answer can be very large, return it **modulo** $10^9 + 7$.

Input: n = 3, goal = 3, k = 1

Output: 6

Explanation: There are 6 possible playlists: [1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], and [3, 2, 1].

Input: n = 2, goal = 3, k = 0

Output: 6

Explanation: There are 6 possible playlists: [1, 1, 2], [1, 2, 1], [2, 1, 1], [2, 2, 1], [2, 1, 2], and [1, 2, 2].

Input: n = 2, goal = 3, k = 1

Output: 2

Explanation: There are 2 possible playlists: [1, 2, 1] and [2, 1, 2].

33.7 Count All Valid Pickup and Delivery Options

Given n orders, each order consists of a pickup and a delivery service.

Count all valid pickup/delivery possible sequences such that delivery(i) is always after of pickup(i).

Since the answer may be too large, return it modulo 10^9 + 7.

Input: n = 1

Output: 1

Explanation: Unique order (P1, D1), Delivery 1 always is after of Pickup 1.

Input: n = 2

Output: 6

Explanation: All possible orders:

(P1, P2, D1, D2), (P1, P2, D2, D1), (P1, D1, P2, D2), (P2, P1, D1, D2), (P2, P1, D2, D1) and (P2, D2, P1, D1).

This is an invalid order (P1, D2, P2, D1) because Pickup 2 is after of Delivery 2.

Input: n = 3

Output: 90

33.8 Count Sorted Vowel Strings

Given an integer n, return the number of strings of length n that consist only of vowels (a, e, i, o, u) and are **lexicographically sorted**.

A string s is **lexicographically sorted** if for all valid i, s[i] is the same as or comes before s[i+1] in the alphabet.

Input: n = 1

Output: 5

Explanation: The 5 sorted strings that consist of vowels only are ["a","e","i","o","u"].

Input: n = 2

Output: 15

Explanation: The 15 sorted strings that consist of vowels only are

["aa","ae","ai","ao","au","ee","ei","eo","eu","ii","io","iu","oo","ou","uu"].

Note that "ea" is not a valid string since 'e' comes after 'a' in the alphabet.

Input: n = 33

Output: 66045

33.9 Count Ways to Make Array with Product

You are given a 2D integer array, queries. For each queries[i], where queries[i] = [n_i, k_i], find the number of different ways you can place positive integers into an array of size n_i such that the product of the integers is k_i. As the number of ways may be too large, the answer to the i^{th} query is the number of ways **modulo** $10^9 + 7$.

Return an integer array answer where answer.length == queries.length, and answer[i] is the answer to the i^{th} query.

Input: queries = [[2, 6], [5, 1], [73, 660]]

Output: [4, 1, 50734910]

Explanation: Each query is independent.

[2, 6]: There are 4 ways to fill an array of size 2 that multiply to 6: [1, 6], [2, 3], [3, 2], [6, 1].

[5, 1]: There is 1 way to fill an array of size 5 that multiply to 1: [1, 1, 1, 1, 1].

[73, 660]: There are 1050734917 ways to fill an array of size 73 that multiply to 660. 1050734917 modulo $10^9 + 7$ = 50734910.

Input: queries = [[1, 1], [2, 2], [3, 3], [4, 4], [5, 5]]

Output: [1, 2, 3, 10, 5]

33.10 Minimum Number of Operations to Make String Sorted

You are given a string s (**0-indexed**). You are asked to perform the following operation on s until you get a sorted string:

1. Find **the largest index** i such that 1 <= i < s.length and s[i] < s[i - 1].
2. Find **the largest index** j such that i <= j < s.length and s[k] < s[i - 1] for all the possible values of k in the range [i, j] inclusive.
3. Swap the two characters at indices i - 1 and j.
4. Reverse the suffix starting at index i.

Return the number of operations needed to make the string sorted. Since the answer can be too large, return it **modulo** $10^9 + 7$.

Input: s = "cba"

Output: 5

Explanation: The simulation goes as follows:

Operation 1: i=2, j=2. Swap s[1] and s[2] to get s="cab", then reverse the suffix starting at 2. Now, s="cab".

Operation 2: i=1, j=2. Swap s[0] and s[2] to get s="bac", then reverse the suffix starting at 1. Now, s="bca".

Operation 3: i=2, j=2. Swap s[1] and s[2] to get s="bac", then reverse the suffix starting at 2. Now, s="bac".

Operation 4: i=1, j=1. Swap s[0] and s[1] to get s="abc", then reverse the suffix starting at 1. Now, s="acb".

Operation 5: i=2, j=2. Swap s[1] and s[2] to get s="abc", then reverse the suffix starting at 2. Now, s="abc".

Input: s = "aabaa"

Output: 2

Explanation: The simulation goes as follows:

Operation 1: i=3, j=4. Swap s[2] and s[4] to get s="aaaab", then reverse the substring starting at 3. Now, s="aaaba".

Operation 2: i=4, j=4. Swap s[3] and s[4] to get s="aaaab", then reverse the substring starting at 4. Now, s="aaaab".

34. Probability

34.1 Soup Servings

There are two types of soup: **type A** and **type B**. Initially, we have n ml of each type of soup. There are four kinds of operations:

1. Serve 100 ml of **soup A** and 0 ml of **soup B**,
2. Serve 75 ml of **soup A** and 25 ml of **soup B**,
3. Serve 50 ml of **soup A** and 50 ml of **soup B**, and
4. Serve 25 ml of **soup A** and 75 ml of **soup B**.

When we serve some soup, we give it to someone, and we no longer have it. Each turn, we will choose from the four operations with an equal probability 0.25. If the remaining volume of soup is not enough to complete the operation, we will serve as much as possible. We stop once we no longer have some quantity of both types of soup.

Note that we do not have an operation where all 100 ml's of **soup B** are used first.

Return the probability that **soup A** will be empty first, plus half the probability that **A** and **B** become empty at the same time. Answers within 10^{-5} of the actual answer will be accepted.

Input: n = 50

Output: 0.62500

Explanation: If we choose the first two operations, A will become empty first.

For the third operation, A and B will become empty at the same time.

For the fourth operation, B will become empty first.

So the total probability of A becoming empty first plus half the probability that A and B become empty at the same time, is 0.25 * (1 + 1 + 0.5 + 0) = 0.625.

Input: n = 100

Output: 0.71875

34.2 New 21 Game

Alice plays the following game, loosely based on the card game **"21"**.

Alice starts with 0 points and draws numbers while she has less than k points. During each draw, she gains an integer number of points randomly from the range [1, maxPts], where maxPts is an integer. Each draw is independent and the outcomes have equal probabilities.

Alice stops drawing numbers when she gets k **or more points**.

Return the probability that Alice has n or fewer points.

Answers within 10^{-5} of the actual answer are considered accepted.

Input: n = 10, k = 1, maxPts = 10

Output: 1.00000

Explanation: Alice gets a single card, then stops.

Input: n = 6, k = 1, maxPts = 10

Output: 0.60000

Explanation: Alice gets a single card, then stops.

In 6 out of 10 possibilities, she is at or below 6 points.

Input: n = 21, k = 17, maxPts = 10

Output: 0.73278

34.3 Statistics from a Large Sample

You are given a large sample of integers in the range [0, 255]. Since the sample is so large, it is represented by an array count where count[k] is the **number of times** that k appears in the sample.

Calculate the following statistics:

- minimum: The minimum element in the sample.
- maximum: The maximum element in the sample.
- mean: The average of the sample, calculated as the total sum of all elements divided by the total number of elements.
- median:
    - If the sample has an odd number of elements, then the median is the middle element once the sample is sorted.
    - If the sample has an even number of elements, then the median is the average of the two middle elements once the sample is sorted.
- mode: The number that appears the most in the sample. It is guaranteed to be **unique**.

Return the statistics of the sample as an array of floating-point numbers [minimum, maximum, mean, median, mode]. Answers within 10^{-5} of the actual answer will be accepted.

Input: count =
[0,1,3,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0]

Output: [1.00000, 3.00000, 2.37500, 2.50000, 3.00000]

Explanation: The sample represented by count is [1, 2, 2, 2, 3, 3, 3, 3].

The minimum and maximum are 1 and 3 respectively.

The mean is (1+2+2+2+3+3+3+3) / 8 = 19 / 8 = 2.375.

Since the size of the sample is even, the median is the average of the two middle elements 2 and 3, which is 2.5.

The mode is 3 as it appears the most in the sample.

Input: count =
[0,4,3,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0]

Output: [1.00000, 4.00000, 2.18182, 2.00000, 1.00000]

Explanation: The sample represented by count is [1,1,1,1,2,2,2,3,3,4,4].

The minimum and maximum are 1 and 4 respectively.

The mean is (1+1+1+1+2+2+2+3+3+4+4) / 11 = 24 / 11 = 2.18181818... (for display purposes, the output shows the rounded number 2.18182).

Since the size of the sample is odd, the median is the middle element 2.

The mode is 1 as it appears the most in the sample.

34.4 Airplane Seat Assignment Probability

n passengers board an airplane with exactly n seats. The first passenger has lost the ticket and picks a seat randomly. But after that, the rest of the passengers will:

- Take their own seat if it is still available, and
- Pick other seats randomly when they find their seat occupied

Return the probability that the n^{th} person gets his own seat.

Input: n = 1

Output: 1.00000

Explanation: The first person can only get the first seat.

Input: n = 2

Output: 0.50000

Explanation: The second person has a probability of 0.5 to get the second seat (when first person gets the first seat).

34.5 Implement Rand10() Using Rand7()

Given the **API** rand7() that generates a uniform random integer in the range [1, 7], write a function rand10() that generates a uniform random integer in the range [1, 10]. You can only call the API rand7(), and you shouldn't call any other API. Please **do not** use a language's built-in random API.

Each test case will have one **internal** argument n, the number of times that your implemented function rand10() will be called while testing. Note that this is **not an argument** passed to rand10().

Input: n = 1

Output: [2]

Input: n = 2

Output: [2, 8]

Input: n = 3

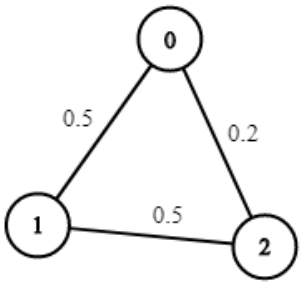
Output: [3, 8, 10]

34.6 Path with Maximum Probability

You are given an undirected weighted graph of n nodes (0-indexed), represented by an edge list where edges[i] = [a, b] is an undirected edge connecting the nodes a and b with a probability of success of traversing that edge succProb[i].

Given two nodes start and end, find the path with the maximum probability of success to go from start to end and return its success probability.

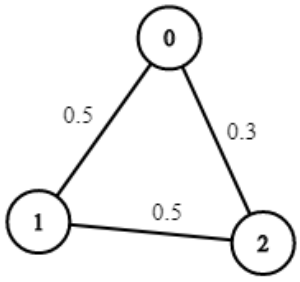
If there is no path from start to end, **return 0**. Your answer will be accepted if it differs from the correct answer by at most **1e-5**.



Input: n = 3, edges = [[0, 1], [1, 2], [0, 2]], succProb = [0.5,0.5,0.2], start = 0, end = 2

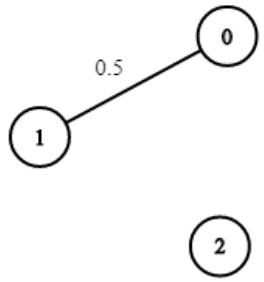
Output: 0.25000

Explanation: There are two paths from start to end, one having a probability of success = 0.2 and the other has 0.5 * 0.5 = 0.25.



Input: n = 3, edges = [[0, 1], [1, 2], [0, 2]], succProb = [0.5, 0.5, 0.3], start = 0, end = 2

Output: 0.30000



Input: n = 3, edges = [[0, 1]], succProb = [0.5], start = 0, end = 2

Output: 0.00000

Explanation: There is no path between 0 and 2.

34.7 Probability of a Two Boxes having the Same Number of Distinct Balls

Given 2n balls of k distinct colors. You will be given an integer array balls of size k where balls[i] is the number of balls of color i.

All the balls will be **shuffled uniformly at random**, then we will distribute the first n balls to the first box and the remaining n balls to the other box (Please read the explanation of the second example carefully).

Please note that the two boxes are considered different. For example, if we have two balls of colors a and b, and two boxes [] and (), then the distribution [a] (b) is considered different than the distribution [b] (a) (Please read the explanation of the first example carefully).

Return *the probability* that the two boxes have the same number of distinct balls. Answers within 10^{-5} of the actual value will be accepted as correct.

Input: balls = [1, 1]

Output: 1.00000

Explanation: Only 2 ways to divide the balls equally:

- A ball of color 1 to box 1 and a ball of color 2 to box 2

- A ball of color 2 to box 1 and a ball of color 1 to box 2

In both ways, the number of distinct colors in each box is equal. The probability is 2/2 = 1

Input: balls = [2, 1, 1]

Output: 0.66667

Explanation: We have the set of balls [1, 1, 2, 3]

This set of balls will be shuffled randomly and we may have one of the 12 distinct shuffles with equal probability (i.e. 1/12):

[1,1 / 2,3], [1,1 / 3,2], [1,2 / 1,3], [1,2 / 3,1], [1,3 / 1,2], [1,3 / 2,1], [2,1 / 1,3], [2,1 / 3,1], [2,3 / 1,1], [3,1 / 1,2], [3,1 / 2,1], [3,2 / 1,1]

After that, we add the first two balls to the first box and the second two balls to the second box.

We can see that 8 of these 12 possible random distributions have the same number of distinct colors of balls in each box.

Probability is 8/12 = 0.66667

Input: balls = [1, 2, 1, 2]

Output: 0.60000

Explanation: The set of balls is [1, 2, 2, 3, 4, 4]. It is hard to display all the 180 possible random shuffles of this set but it is easy to check that 108 of them will have the same number of distinct colors in each box.

Probability = 108 / 180 = 0.6

35. Cycle-Finding

35.1 Detect Cycles in 2D Grid

Given a 2D array of characters grid of size m x n, you need to find if there exists any cycle consisting of the **same value** in grid.

A cycle is a path of **length 4 or more** in the grid that starts and ends at the same cell. From a given cell, you can move to one of the cells adjacent to it - in one of the four directions (up, down, left, or right), if it has the **same value** of the current cell.

Also, you cannot move to the cell that you visited in your last move. For example, the cycle (1, 1) -> (1, 2) -> (1, 1) is invalid because from (1, 2) we visited (1, 1) which was the last visited cell.

Return true if any cycle of the same value exists in grid, otherwise, return false.

Input: grid = [["a", "a", "a", "a"], ["a", "b", "b", "a"], ["a", "b", "b", "a"], ["a", "a", "a", "a"]]

Output: true

Explanation: There are two valid cycles shown in different colors in the image below:

Input: grid = [["c", "c", "c", "a"], ["c", "d", "c", "c"], ["c", "c", "e", "c"], ["f", "c", "c", "c"]]

Output: true

Explanation: There is only one valid cycle highlighted in the image below:

Input: grid = [["a", "b", "b"], ["b", "z", "b"], ["b", "b", "a"]]

Output: false

35.2 Course Schedule II

There are a total of numCourses courses you have to take, labeled from 0 to numCourses - 1. You are given an array prerequisites where prerequisites[i] = [a_i, b_i] indicates that you **must** take course b_i first if you want to take course a_i.

- For example, the pair [0, 1], indicates that to take course 0 you have to first take course 1.

Return the ordering of courses you should take to finish all courses. If there are many valid answers, return **any** of them. If it is impossible to finish all courses, return **an empty array**.

Input: numCourses = 2, prerequisites = [[1, 0]]

Output: [0, 1]

Explanation: There are a total of 2 courses to take. To take course 1 you should have finished course 0. So the correct course order is [0, 1].

Input: numCourses = 4, prerequisites = [[1, 0], [2, 0], [3, 1], [3, 2]]

Output: [0, 2, 1, 3]

Explanation: There are a total of 4 courses to take. To take course 3 you should have finished both courses 1 and 2. Both courses 1 and 2 should be taken after you finished course 0.

So one correct course order is [0, 1, 2, 3]. Another correct ordering is [0, 2, 1, 3].

Input: numCourses = 1, prerequisites = []

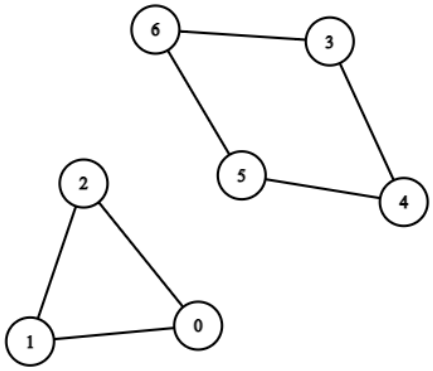
Output: [0]

35.3 Shortest Cycle in a Graph

There is a **bi-directional** graph with n vertices, where each vertex is labeled from 0 to n - 1. The edges in the graph are represented by a given 2D integer array edges, where edges[i] = [u_i, v_i] denotes an edge between vertex u_i and vertex v_i. Every vertex pair is connected by at most one edge, and no vertex has an edge to itself.

Return the length of the **shortest** cycle in the graph. If no cycle exists, return -1.

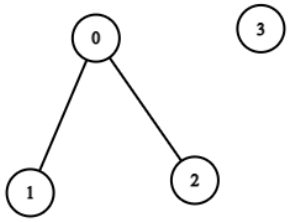
A cycle is a path that starts and ends at the same node, and each edge in the path is used only once.



Input: n = 7, edges = [[0, 1], [1, 2], [2, 0], [3, 4], [4, 5], [5, 6], [6, 3]]

Output: 3

Explanation: The cycle with the smallest length is : 0 -> 1 -> 2 -> 0



Input: n = 4, edges = [[0, 1], [0, 2]]

Output: -1

Explanation: There are no cycles in this graph.

36. Matrices

36.1 Valid Sudoku

Determine if a 9 x 9 Sudoku board is valid. Only the filled cells need to be validated **according to the following rules**:

1. Each row must contain the digits 1-9 without repetition.
2. Each column must contain the digits 1-9 without repetition.
3. Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without repetition.

Note:

- A Sudoku board (partially filled) could be valid but is not necessarily solvable.
- Only the filled cells need to be validated according to the mentioned rules.

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

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: true

Input: board =

[["8","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"]]

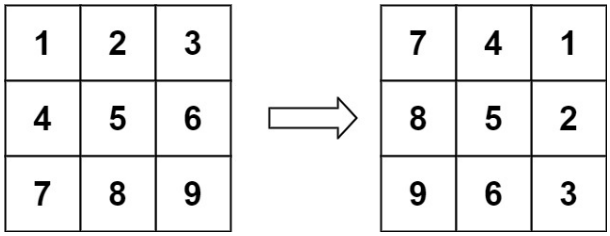
Output: false

Explanation: Same as Example 1, except with the **5** in the top left corner being modified to **8**. Since there are two 8's in the top left 3x3 sub-box, it is invalid.

36.2 Rotate Image

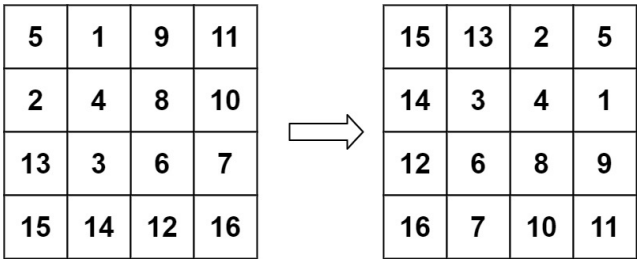
You are given an n x n 2D matrix representing an image, rotate the image by **90** degrees (clockwise).

You have to rotate the image **in-place**, which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.



Input: matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

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

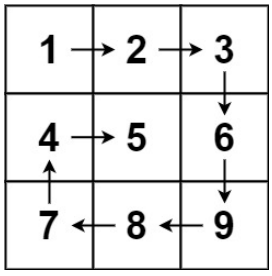


Input: matrix = [[5, 1, 9, 11], [2, 4, 8, 10], [13, 3, 6, 7], [15, 14, 12, 16]]

Output: [[15, 13, 2, 5], [14, 3, 4, 1], [12, 6, 8, 9], [16, 7, 10, 11]]

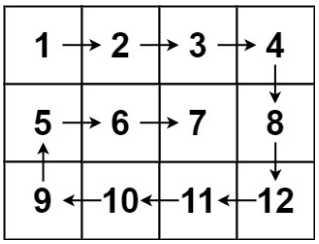
36.3 Spiral Matrix

Given an m x n matrix, return all elements of the matrix in spiral order.



Input: matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

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



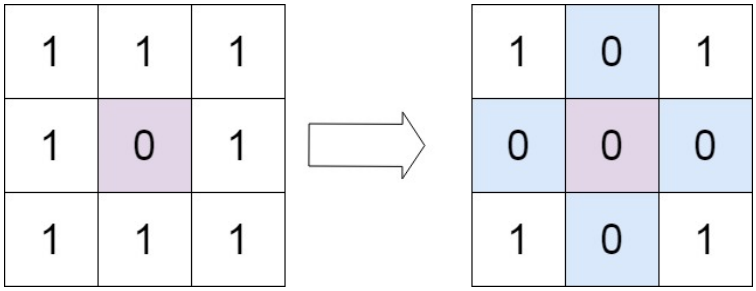
Input: matrix = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]

Output: [1, 2, 3, 4, 8, 12, 11, 10, 9, 5, 6, 7]

36.4 Set Matrix Zeroes

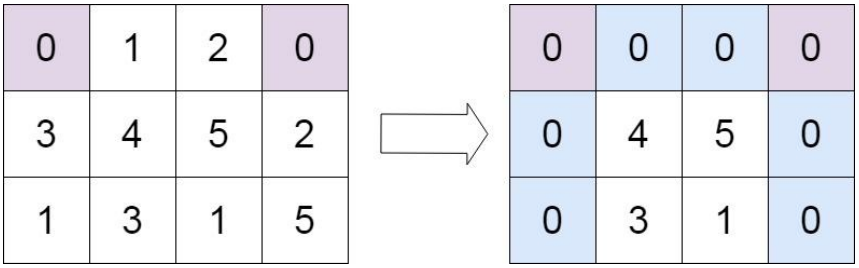
Given an m x n integer matrix matrix, if an element is 0, set its entire row and column to 0's.

You must do it in place.



Input: matrix = [[1, 1, 1], [1, 0, 1], [1, 1, 1]]

Output: [[1, 0, 1], [0, 0, 0], [1, 0, 1]]



Input: matrix = [[0, 1, 2, 0], [3, 4, 5, 2], [1, 3, 1, 5]]

Output: [[0, 0, 0, 0], [0, 4, 5, 0], [0, 3, 1, 0]]

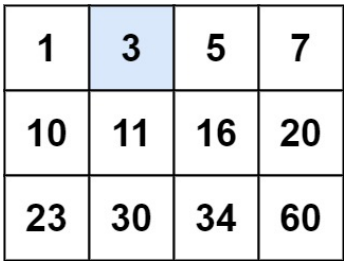
36.5 Search a 2D Matrix

You are given an m x n integer matrix matrix with the following two properties:

- Each row is sorted in non-decreasing order.
- The first integer of each row is greater than the last integer of the previous row.

Given an integer target, return true if target is in matrix or false otherwise.

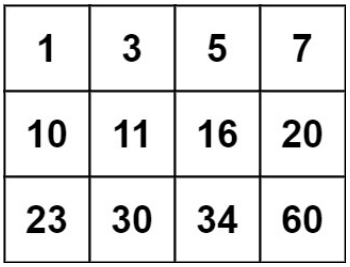
You must write a solution in O(log(m * n)) time complexity.



1	3	5	7
10	11	16	20
23	30	34	60

Input: matrix = [[1, 3, 5, 7], [10, 11, 16, 20], [23, 30, 34, 60]], target = 3

Output: true



1	3	5	7
10	11	16	20
23	30	34	60

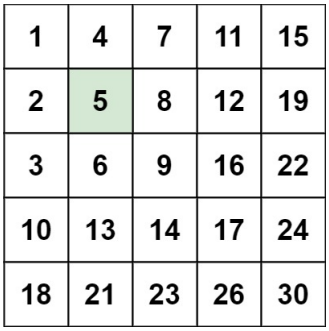
Input: matrix = [[1, 3, 5, 7], [10, 11, 16, 20], [23, 30, 34, 60]], target = 13

Output: false

36.6 Search a 2D Matrix II

Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:

- Integers in each row are sorted in ascending from left to right.
- Integers in each column are sorted in ascending from top to bottom.



1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

Input: matrix = [[1, 4, 7, 11, 15], [2, 5, 8, 12, 19], [3, 6, 9, 16, 22], [10, 13, 14, 17, 24], [18, 21, 23, 26, 30]], target = 5

Output: true

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

Input: matrix = [[1, 4, 7, 11, 15], [2, 5, 8, 12, 19], [3, 6, 9, 16, 22], [10, 13, 14, 17, 24], [18, 21, 23, 26, 30]], target = 20

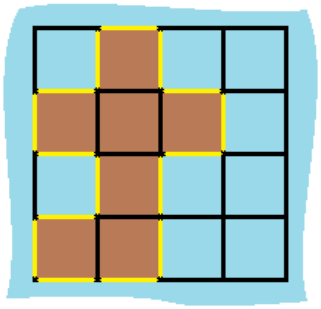
Output: false

36.7 Island Perimeter

You are given row x col grid representing a map where grid[i][j] = 1 represents land and grid[i][j] = 0 represents water.

Grid cells are connected **horizontally/vertically** (not diagonally). The grid is completely surrounded by water, and there is exactly one island (i.e., one or more connected land cells).

The island doesn't have "lakes", meaning the water inside isn't connected to the water around the island. One cell is a square with side length 1. The grid is rectangular, width and height don't exceed 100. Determine the perimeter of the island.



Input: grid = [[0, 1, 0, 0], [1, 1, 1, 0], [0, 1, 0, 0], [1, 1, 0, 0]]

Output: 16

Explanation: The perimeter is the 16 yellow stripes in the image above.

Input: grid = [[1]]

Output: 4

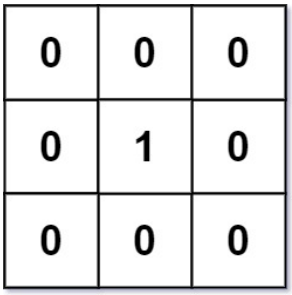
Input: grid = [[1, 0]]

Output: 4

36.8 01 Matrix

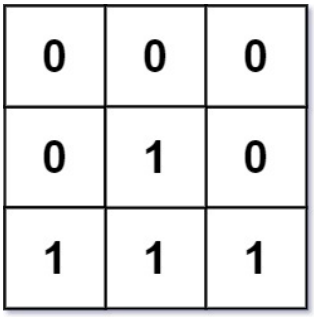
Given an m x n binary matrix mat, return the distance of the nearest 0 for each cell.

The distance between two adjacent cells is 1.



Input: mat = [[0, 0, 0], [0, 1, 0], [0, 0, 0]]

Output: [[0, 0, 0], [0, 1, 0], [0, 0, 0]]



Input: mat = [[0, 0, 0], [0, 1, 0], [1, 1, 1]]

Output: [[0, 0, 0], [0, 1, 0], [1, 2, 1]]

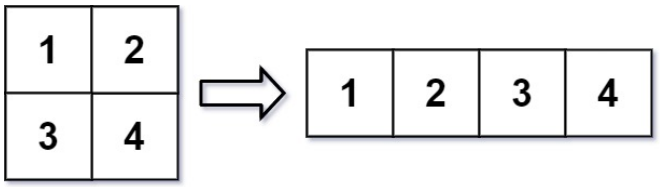
36.9 Reshape the Matrix

In MATLAB, there is a handy function called reshape which can reshape an m x n matrix into a new one with a different size r x c keeping its original data.

You are given an m x n matrix mat and two integers r and c representing the number of rows and the number of columns of the wanted reshaped matrix.

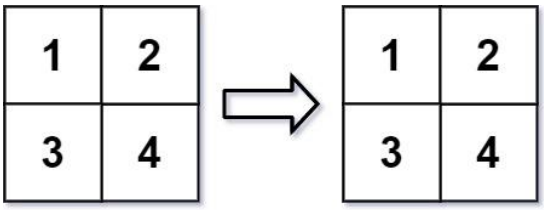
The reshaped matrix should be filled with all the elements of the original matrix in the same row-traversing order as they were.

If the reshape operation with given parameters is possible and legal, output the new reshaped matrix; Otherwise, output the original matrix.



Input: mat = [[1, 2], [3, 4]], r = 1, c = 4

Output: [[1, 2, 3, 4]]



Input: mat = [[1, 2], [3, 4]], r = 2, c = 4

Output: [[1, 2], [3, 4]]

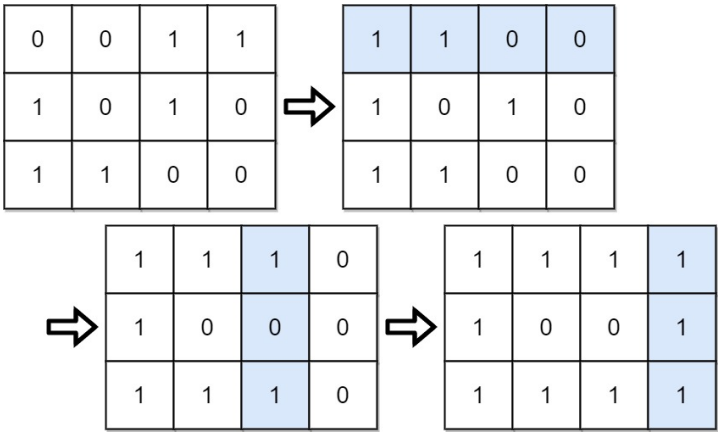
36.10 Score after Flipping Matrix

You are given an m x n binary matrix grid.

A **move** consists of choosing any row or column and toggling each value in that row or column (i.e., changing all 0's to 1's, and all 1's to 0's).

Every row of the matrix is interpreted as a binary number, and the **score** of the matrix is the sum of these numbers.

Return the highest possible **score** after making any number of **moves** (including zero moves).



Input: grid = [[0, 0, 1, 1], [1, 0, 1, 0], [1, 1, 0, 0]]

Output: 39

Explanation: 0b1111 + 0b1001 + 0b1111 = 15 + 9 + 15 = 39

Input: grid = [[0]]

Output: 1

37. Game Theory

37.1 Nim Game

You are playing the following Nim Game with your friend:

- Initially, there is a heap of stones on the table.
- You and your friend will alternate taking turns, and **you go first**.
- On each turn, the person whose turn it is will remove 1 to 3 stones from the heap.
- The one who removes the last stone is the winner.

Given n, the number of stones in the heap, return true if you can win the game assuming both you and your friend play optimally, otherwise return false.

Input: n = 4

Output: false

Explanation: These are the possible outcomes:

1. You remove 1 stone. Your friend removes 3 stones, including the last stone. Your friend wins.

2. You remove 2 stones. Your friend removes 2 stones, including the last stone. Your friend wins.

3. You remove 3 stones. Your friend removes the last stone. Your friend wins.

In all outcomes, your friend wins.

Input: n = 1

Output: true

Example 3:

Input: n = 2

Output: true

37.2 Can I Win

In the "100 game" two players take turns adding, to a running total, any integer from 1 to 10. The player who first causes the running total to **reach or exceed** 100 wins.

What if we change the game so that players **cannot** re-use integers?

For example, two players might take turns drawing from a common pool of numbers from 1 to 15 without replacement until they reach a total >= 100.

Given two integers maxChoosableInteger and desiredTotal, return true if the first player to move can force a win, otherwise, return false. Assume both players play **optimally**.

Input: maxChoosableInteger = 10, desiredTotal = 11

Output: false

Explanation:

No matter which integer the first player choose, the first player will lose.

The first player can choose an integer from 1 up to 10.

If the first player choose 1, the second player can only choose integers from 2 up to 10.

The second player will win by choosing 10 and get a total = 11, which is >= desiredTotal.

Same with other integers chosen by the first player, the second player will always win.

Input: maxChoosableInteger = 10, desiredTotal = 0

Output: true

Input: maxChoosableInteger = 10, desiredTotal = 1

Output: true

37.3 Predict the Winner

You are given an integer array nums. Two players are playing a game with this array: player 1 and player 2.

Player 1 and player 2 take turns, with player 1 starting first. Both players start the game with a score of 0. At each turn, the player takes one of the numbers from either end of the array

(i.e., nums[0] or nums[nums.length - 1]) which reduces the size of the array by 1. The player adds the chosen number to their score. The game ends when there are no more elements in the array.

Return true if Player 1 can win the game. If the scores of both players are equal, then player 1 is still the winner, and you should also return true. You may assume that both players are playing optimally.

Input: nums = [1, 5, 2]

Output: false

Explanation: Initially, player 1 can choose between 1 and 2.

If he chooses 2 (or 1), then player 2 can choose from 1 (or 2) and 5. If player 2 chooses 5, then player 1 will be left with 1 (or 2).

So, final score of player 1 is 1 + 2 = 3, and player 2 is 5.

Hence, player 1 will never be the winner and you need to return false.

Input: nums = [1, 5, 233, 7]

Output: true

Explanation: Player 1 first chooses 1. Then player 2 has to choose between 5 and 7. No matter which number player 2 choose, player 1 can choose 233.

Finally, player 1 has more score (234) than player 2 (12), so you need to return True representing player1 can win.

37.4 Stone Game

Alice and Bob play a game with piles of stones. There are an **even** number of piles arranged in a row, and each pile has a **positive** integer number of stones piles[i].

The objective of the game is to end with the most stones. The **total** number of stones across all the piles is **odd**, so there are no ties.

Alice and Bob take turns, with **Alice starting first**. Each turn, a player takes the entire pile of stones either from the **beginning** or from the **end** of the row. This continues until there are no more piles left, at which point the person with the **most stones wins**.

Assuming Alice and Bob play optimally, return true if Alice wins the game, or false if Bob wins.

Input: piles = [5, 3, 4, 5]

Output: true

Explanation:

Alice starts first, and can only take the first 5 or the last 5.

Say she takes the first 5, so that the row becomes [3, 4, 5].

If Bob takes 3, then the board is [4, 5], and Alice takes 5 to win with 10 points.

If Bob takes the last 5, then the board is [3, 4], and Alice takes 4 to win with 9 points.

This demonstrated that taking the first 5 was a winning move for Alice, so we return true.

Input: piles = [3, 7, 2, 3]

Output: true

37.5 Cat and Mouse

A game on an **undirected** graph is played by two players, Mouse and Cat, who alternate turns.

The graph is given as follows: graph[a] is a list of all nodes b such that ab is an edge of the graph.

The mouse starts at node 1 and goes first, the cat starts at node 2 and goes second, and there is a hole at node 0.

During each player's turn, they **must** travel along one edge of the graph that meets where they are. For example, if the Mouse is at node 1, it **must** travel to any node in graph[1].

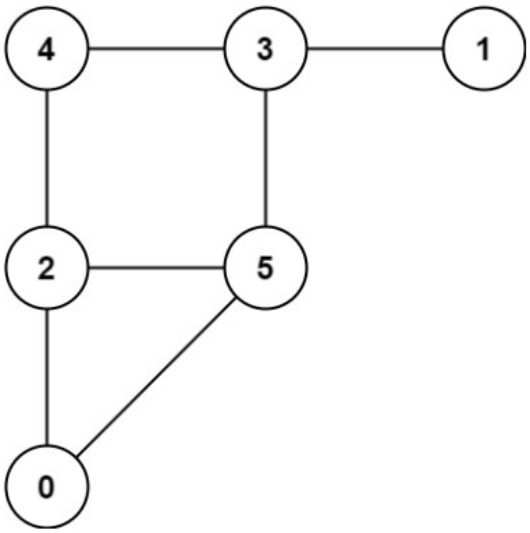
Additionally, it is not allowed for the Cat to travel to the Hole (node 0.)

Then, the game can end in three ways:

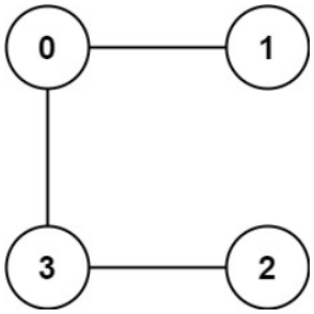
- If ever the Cat occupies the same node as the Mouse, the Cat wins.
- If ever the Mouse reaches the Hole, the Mouse wins.
- If ever a position is repeated (i.e., the players are in the same position as a previous turn, and it is the same player's turn to move), the game is a draw.

Given a graph, and assuming both players play optimally, return

- 1 if the mouse wins the game,
- 2 if the cat wins the game, or
- 0 if the game is a draw.



Input: graph = [[2, 5], [3], [0, 4, 5], [1, 4, 5], [2, 3], [0, 2, 3]]
Output: 0



Input: graph = [[1, 3], [0], [3], [0, 2]]

Output: 1

37.6 Divisor Game

Alice and Bob take turns playing a game, with Alice starting first.

Initially, there is a number n on the chalkboard. On each player's turn, that player makes a move consisting of:

- Choosing any x with 0 < x < n and n % x == 0.
- Replacing the number n on the chalkboard with n - x.

Also, if a player cannot make a move, they lose the game.

Return true if and only if Alice wins the game, assuming both players play optimally.

Input: n = 2

Output: true

Explanation: Alice chooses 1, and Bob has no more moves.

Input: n = 3

Output: false

Explanation: Alice chooses 1, Bob chooses 1, and Alice has no more moves.

37.7 Maximum Number of Coins You Can Get

There are 3n piles of coins of varying size, you and your friends will take piles of coins as follows:

- In each step, you will choose **any** 3 piles of coins (not necessarily consecutive).
- Of your choice, Alice will pick the pile with the maximum number of coins.
- You will pick the next pile with the maximum number of coins.
- Your friend Bob will pick the last pile.
- Repeat until there are no more piles of coins.

Given an array of integers piles where piles[i] is the number of coins in the i^{th} pile.

Return the maximum number of coins that you can have.

Input: piles = [2, 4, 1, 2, 7, 8]

Output: 9

Explanation: Choose the triplet (2, 7, 8), Alice Pick the pile with 8 coins, you the pile with **7** coins and Bob the last one.

Choose the triplet (1, 2, 4), Alice Pick the pile with 4 coins, you the pile with **2** coins and Bob the last one.

The maximum number of coins which you can have are: 7 + 2 = 9.

On the other hand if we choose this arrangement (1, **2**, 8), (2, **4**, 7) you only get 2 + 4 = 6 coins which is not optimal.

Input: piles = [2, 4, 5]

Output: 4

Input: piles = [9, 8, 7, 6, 5, 1, 2, 3, 4]

Output: 18

37.8 Sum Game

Alice and Bob take turns playing a game, with **Alice starting first**.

You are given a string num of **even length** consisting of digits and '?' characters. On each turn, a player will do the following if there is still at least one '?' in num:

1. Choose an index i where num[i] == '?'.
2. Replace num[i] with any digit between '0' and '9'.

The game ends when there are no more '?' characters in num.

For Bob to win, the sum of the digits in the first half of num must be **equal** to the sum of the digits in the second half. For Alice to win, the sums must **not be equal**.

- For example, if the game ended with num = "243801", then Bob wins because 2+4+3 = 8+0+1. If the game ended with num = "243803", then Alice wins because 2+4+3 != 8+0+3.

Assuming Alice and Bob play **optimally**, return true if Alice will win and false if Bob will win.

Input: num = "5023"

Output: false

Explanation: There are no moves to be made.

The sum of the first half is equal to the sum of the second half: 5 + 0 = 2 + 3.

Input: num = "25??"

Output: true

Explanation: Alice can replace one of the '?'s with '9' and it will be impossible for Bob to make the sums equal.

Input: num = "?3295???"

Output: false

Explanation: It can be proven that Bob will always win. One possible outcome is:

- Alice replaces the first '?' with '9'. num = "93295???".

- Bob replaces one of the '?' in the right half with '9'. num = "932959??".

- Alice replaces one of the '?' in the right half with '2'. num = "9329592?".

- Bob replaces the last '?' in the right half with '7'. num = "93295927".

Bob wins because 9 + 3 + 2 + 9 = 5 + 9 + 2 + 7.

37.9 Stone Game II

Alice and Bob continue their games with piles of stones. There are a number of piles **arranged in a row**, and each pile has a positive integer number of stones piles[i]. The objective of the game is to end with the most stones.

Alice and Bob take turns, with Alice starting first. Initially, M = 1.

On each player's turn, that player can take **all the stones** in the **first** X remaining piles, where 1 <= X <= 2M. Then, we set M = max(M, X).

The game continues until all the stones have been taken.

Assuming Alice and Bob play optimally, return the maximum number of stones Alice can get.

Input: piles = [2, 7, 9, 4, 4]

Output: 10

Explanation: If Alice takes one pile at the beginning, Bob takes two piles, then Alice takes 2 piles again. Alice can get 2 + 4 + 4 = 10 piles in total. If Alice takes two piles at the beginning, then Bob can take all three piles left. In this case, Alice get 2 + 7 = 9 piles in total. So we return 10 since it's larger.

Input: piles = [1, 2, 3, 4, 5, 100]

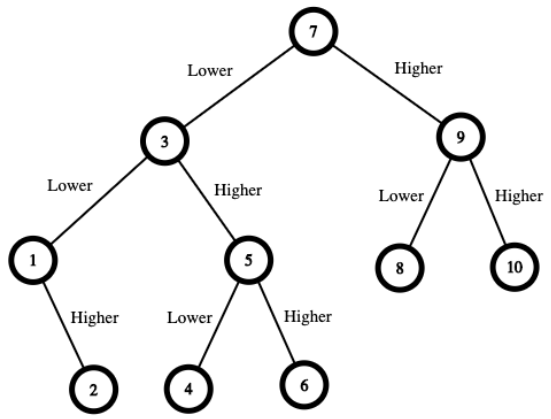
Output: 104

37.10 Guess Number Higher or Lower II

We are playing the Guessing Game. The game will work as follows:

1. I pick a number between 1 and n.
2. You guess a number.
3. If you guess the right number, **you win the game**.
4. If you guess the wrong number, then I will tell you whether the number I picked is **higher or lower**, and you will continue guessing.
5. Every time you guess a wrong number x, you will pay x dollars. If you run out of money, **you lose the game**.

Given a particular n, return the minimum amount of money you need to **guarantee a win regardless of what number I pick**.



Input: n = 10

Output: 16

Explanation: The winning strategy is as follows:

- The range is [1, 10]. Guess 7.

- If this is my number, your total is $0. Otherwise, you pay $7.
- If my number is higher, the range is [8, 10]. Guess 9.
  - If this is my number, your total is $7. Otherwise, you pay $9.
  - If my number is higher, it must be 10. Guess 10. Your total is $7 + $9 = $16.
  - If my number is lower, it must be 8. Guess 8. Your total is $7 + $9 = $16.
- If my number is lower, the range is [1, 6]. Guess 3.
  - If this is my number, your total is $7. Otherwise, you pay $3.
  - If my number is higher, the range is [4,6]. Guess 5.
    - If this is my number, your total is $7 + $3 = $10. Otherwise, you pay $5.
    - If my number is higher, it must be 6. Guess 6. Your total is $7 + $3 + $5 = $15.
    - If my number is lower, it must be 4. Guess 4. Your total is $7 + $3 + $5 = $15.
  - If my number is lower, the range is [1,2]. Guess 1.
    - If this is my number, your total is $7 + $3 = $10. Otherwise, you pay $1.
    - If my number is higher, it must be 2. Guess 2. Your total is $7 + $3 + $1 = $11.

The worst case in all these scenarios is that you pay $16. Hence, you only need $16 to guarantee a win.

Input: n = 1

Output: 0

Explanation: There is only one possible number, so you can guess 1 and not have to pay anything.

Input: n = 2

Output: 1

Explanation: There are two possible numbers, 1 and 2.

- Guess 1.
  - If this is my number, your total is $0. Otherwise, you pay $1.
  - If my number is higher, it must be 2. Guess 2. Your total is $1.

The worst case is that you pay $1.

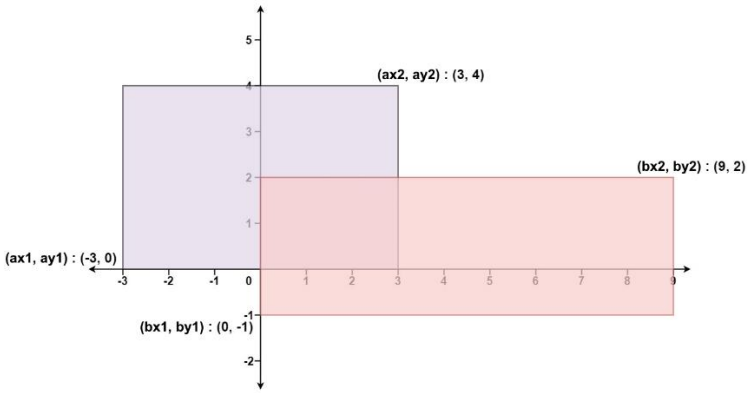
38. Geometry

38.1 Rectangle Area

Given the coordinates of two **rectilinear** rectangles in a 2D plane, return the total area covered by the two rectangles.

The first rectangle is defined by its **bottom-left** corner (ax1, ay1) and its **top-right** corner (ax2, ay2).

The second rectangle is defined by its **bottom-left** corner (bx1, by1) and its **top-right** corner (bx2, by2).



Input: ax1 = -3, ay1 = 0, ax2 = 3, ay2 = 4, bx1 = 0, by1 = -1, bx2 = 9, by2 = 2

Output: 45

Input: ax1 = -2, ay1 = -2, ax2 = 2, ay2 = 2, bx1 = -2, by1 = -2, bx2 = 2, by2 = 2

Output: 16

38.2 Valid Square

Given the coordinates of four points in 2D space p1, p2, p3 and p4, return true if the four points construct a square.

The coordinate of a point p_i is represented as $[x_i, y_i]$. The input is **not** given in any order.

A **valid square** has four equal sides with positive length and four equal angles (90-degree angles).

Input: p1 = [0, 0], p2 = [1, 1], p3 = [1, 0], p4 = [0, 1]

Output: true

Input: p1 = [0, 0], p2 = [1, 1], p3 = [1, 0], p4 = [0, 12]

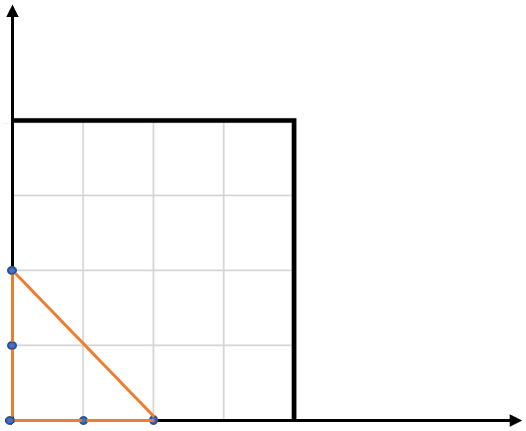
Output: false

Input: p1 = [1, 0], p2 = [-1, 0], p3 = [0, 1], p4 = [0, -1]

Output: true

38.3 Largest Triangle Area

Given an array of points on the **X-Y** plane points where points[i] = $[x_i, y_i]$, return the area of the largest triangle that can be formed by any three different points. Answers within 10^{-5} of the actual answer will be accepted.



Input: points = [[0, 0], [0, 1], [1, 0], [0, 2], [2, 0]]

Output: 2.00000

Explanation: The five points are shown in the above figure. The red triangle is the largest.

Input: points = [[1, 0], [0, 0], [0, 1]]

Output: 0.50000

38.4 Rectangle Overlap

An axis-aligned rectangle is represented as a list [x1, y1, x2, y2], where (x1, y1) is the coordinate of its bottom-left corner, and (x2, y2) is the coordinate of its top-right corner. Its top and bottom edges are parallel to the X-axis, and its left and right edges are parallel to the Y-axis.

Two rectangles overlap if the area of their intersection is **positive**. To be clear, two rectangles that only touch at the corner or edges do not overlap.

Given two axis-aligned rectangles rec1 and rec2, return true if they overlap, otherwise return false.

Input: rec1 = [0, 0, 2, 2], rec2 = [1, 1, 3, 3]

Output: true

Input: rec1 = [0, 0, 1, 1], rec2 = [1, 0, 2, 1]

Output: false

Input: rec1 = [0, 0, 1, 1], rec2 = [2, 2, 3, 3]

Output: false

38.5 Surface Area of 3D Shapes

You are given an n x n grid where you have placed some 1 x 1 x 1 cubes. Each value v = grid[i][j] represents a tower of v cubes placed on top of cell (i, j).

After placing these cubes, you have decided to glue any directly adjacent cubes to each other, forming several irregular 3D shapes.

Return the total surface area of the resulting shapes.

Note: The bottom face of each shape counts toward its surface area.

1	2
3	4

Input: grid = [[1, 2], [3, 4]]

Output: 34

1	1	1
1	0	1
1	1	1

Input: grid = [[1, 1, 1], [1, 0, 1], [1, 1, 1]]

Output: 32

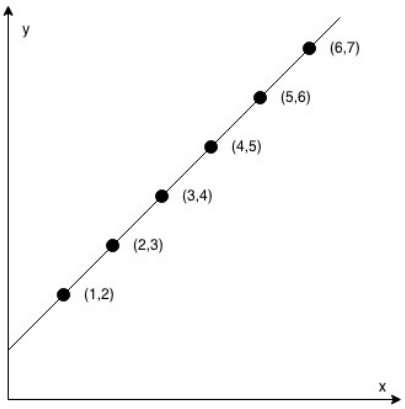
2	2	2
2	1	2
2	2	2

Input: grid = [[2, 2, 2], [2, 1, 2], [2, 2, 2]]

Output: 46

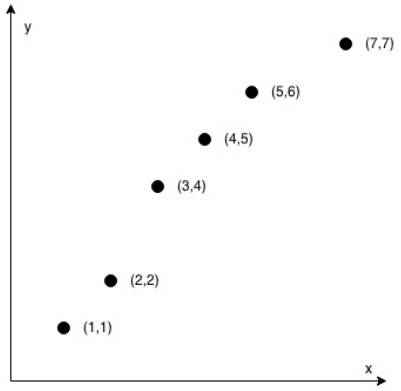
38.6 Check If It Is a Straight Line

You are given an array coordinates, coordinates[i] = [x, y], where [x, y] represents the coordinate of a point. Check if these points make a straight line in the XY plane.



Input: coordinates = [[1,2],[2,3],[3,4],[4,5],[5,6],[6,7]]

Output: true



Input: coordinates = [[1,1],[2,2],[3,4],[4,5],[5,6],[7,7]]

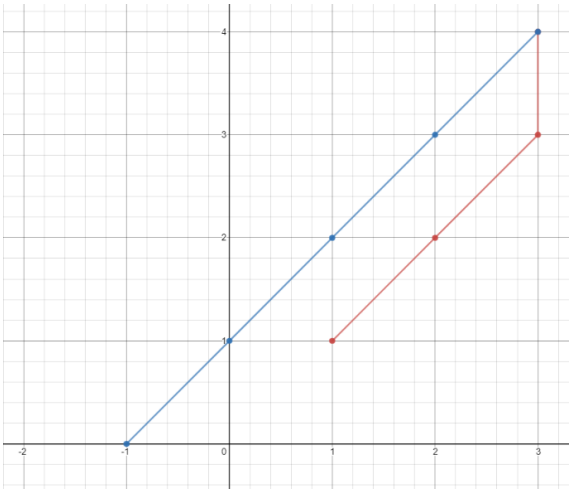
Output: false

38.7 Minimum Time Visiting All Points

On a 2D plane, there are n points with integer coordinates points[i] = [x_i, y_i]. Return the **minimum time** in seconds to visit all the points in the order given by points.

You can move according to these rules:

- In 1 second, you can either:
    - move vertically by one unit,
    - move horizontally by one unit, or
    - move diagonally sqrt(2) units (in other words, move one unit vertically then one unit horizontally in 1 second).
- You have to visit the points in the same order as they appear in the array.
- You are allowed to pass through points that appear later in the order, but these do not count as visits.



Input: points = [[1, 1], [3, 4], [-1, 0]]

Output: 7

Explanation: One optimal path is **[1, 1]** -> [2, 2] -> [3, 3] -> **[3, 4]** -> [2, 3] -> [1, 2] -> [0, 1] -> **[-1, 0]**

Time from [1, 1] to [3, 4] = 3 seconds

Time from [3, 4] to [-1, 0] = 4 seconds

Total time = 7 seconds

Input: points = [[3, 2], [-2, 2]]

Output: 5

38.8 Projection Area of 3D Shapes

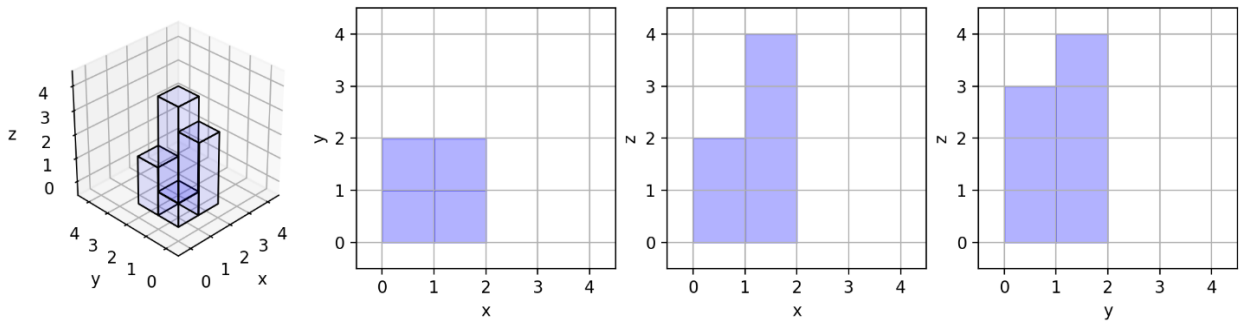
You are given an n x n grid where we place some 1 x 1 x 1 cubes that are axis-aligned with the x, y, and z axes.

Each value v = grid[i][j] represents a tower of v cubes placed on top of the cell (i, j).

We view the projection of these cubes onto the xy, yz, and zx planes.

A **projection** is like a shadow, that maps our **3-dimensional** figure to a **2-dimensional** plane. We are viewing the "shadow" when looking at the cubes from the top, the front, and the side.

Return the total area of all three projections.



Input: grid = [[1, 2], [3, 4]]

Output: 17

Explanation: Here are the three projections ("shadows") of the shape made with each axis-aligned plane.

Input: grid = [[2]]

Output: 5

Input: grid = [[1, 0], [0, 2]]

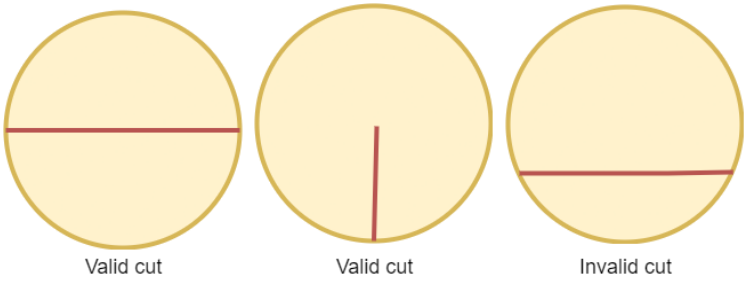
Output: 8

38.9 Minimum Cuts to Divide a Circle

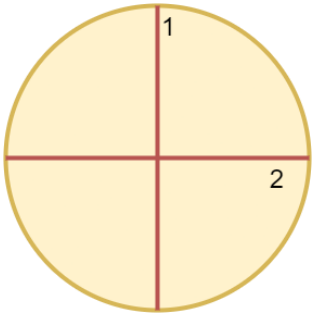
A **valid cut** in a circle can be:

- A cut that is represented by a straight line that touches two points on the edge of the circle and passes through its center, or
- A cut that is represented by a straight line that touches one point on the edge of the circle and its center.

Some valid and invalid cuts are shown in the figures below.



Given the integer n, return *the **minimum** number of cuts needed to divide a circle into* n *equal slices*.

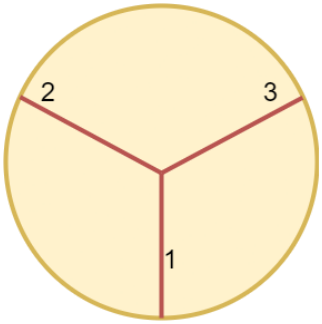


Input: n = 4

Output: 2

Explanation:

The above figure shows how cutting the circle twice through the middle divides it into 4 equal slices.



Input: n = 3

Output: 3

Explanation:

At least 3 cuts are needed to divide the circle into 3 equal slices.

It can be shown that less than 3 cuts cannot result in 3 slices of equal size and shape.

Also note that the first cut will not divide the circle into distinct parts.

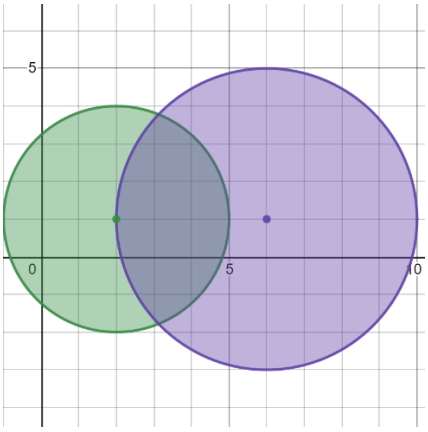
38.10 Detonate the Maximum Bombs

You are given a list of bombs. The **range** of a bomb is defined as the area where its effect can be felt. This area is in the shape of a **circle** with the center as the location of the bomb.

The bombs are represented by a **0-indexed** 2D integer array bombs where bombs[i] = [x_i, y_i, r_i]. x_i and y_i denote the X-coordinate and Y-coordinate of the location of the i^{th} bomb, whereas r_i denotes the **radius** of its range.

You may choose to detonate a **single** bomb. When a bomb is detonated, it will detonate **all bombs** that lie in its range. These bombs will further detonate the bombs that lie in their ranges.

Given the list of bombs, return the **maximum** number of bombs that can be detonated if you are allowed to detonate **only one** bomb.



Input: bombs = [[2, 1, 3], [6, 1, 4]]

Output: 2

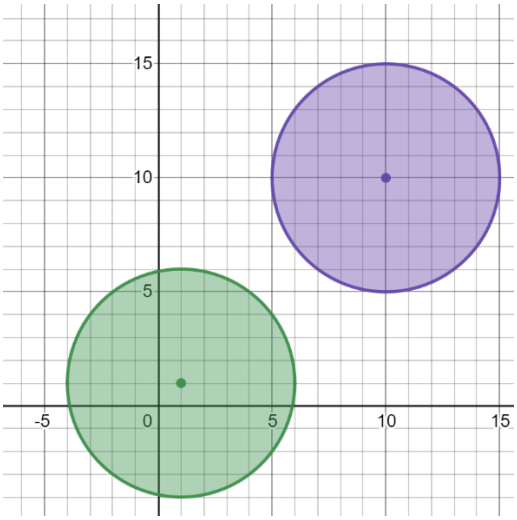
Explanation:

The above figure shows the positions and ranges of the 2 bombs.

If we detonate the left bomb, the right bomb will not be affected.

But if we detonate the right bomb, both bombs will be detonated.

So the maximum bombs that can be detonated is max(1, 2) = 2.

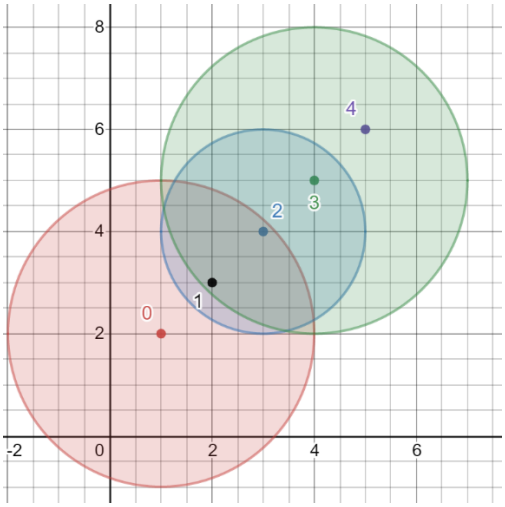


Input: bombs = [[1, 1, 5], [10, 10, 5]]

Output: 1

Explanation:

Detonating either bomb will not detonate the other bomb, so the maximum number of bombs that can be detonated is 1.



Input: bombs = [[1, 2, 3], [2, 3, 1], [3, 4, 2], [4, 5, 3], [5, 6, 4]]

Output: 5

Explanation:

The best bomb to detonate is bomb 0 because:

- Bomb 0 detonates bombs 1 and 2. The red circle denotes the range of bomb 0.

- Bomb 2 detonates bomb 3. The blue circle denotes the range of bomb 2.

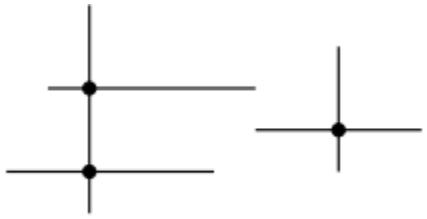
- Bomb 3 detonates bomb 4. The green circle denotes the range of bomb 3.

Thus all 5 bombs are detonated.

39. Sweep Line Algorithms

39.1 Intersection Points

Given a set of n line segments, each of them being either horizontal or vertical, consider the problem of counting the total number of intersection points. When the line segments as shown below have three intersection points.



It is easy to solve the problem in $O(n^2)$ time, because we can go through all possible pairs of line segments and check if they intersect. But we can solve this problem more efficiently in O(n log n) time using a sweep line algorithm and a range query data structure.

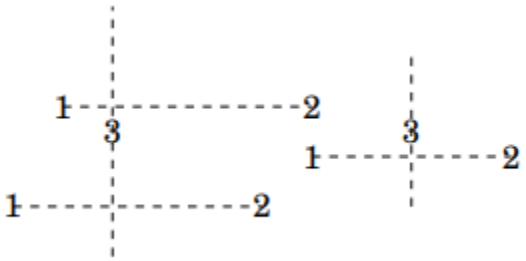
The idea is to process the endpoints of the line segments from left to right and focus on three types of events:

1. Horizontal segment begins

2. Horizontal segment ends

3. Vertical segment

The following events correspond to the example:



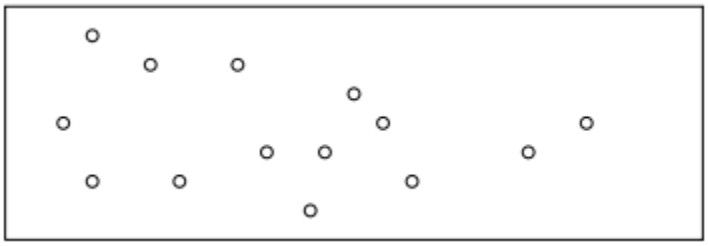
We go through the events from left to right and use a data structure that maintains a set of y coordinates where there is an active horizontal segment. At event 1, we add the y coordinate of the segment to the set, and at event 2, we remove the y coordinate from the set.

Intersection points are calculated at event 3. When there is a vertical segment between points y1 and y2, we count the number of active horizontal segments whose y coordinate is between y1 and y2, and add this number to the total number of intersection points.

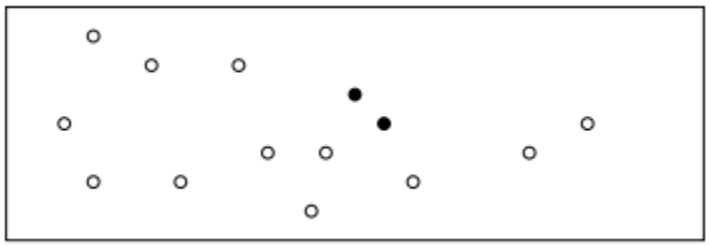
To store y coordinates of horizontal segments, we can use a binary indexed or segment tree, possibly with index compression. When such structures are used, processing each event takes O(logn) time, so the total running time of the algorithm is O(nlogn).

39.2 Closest Pair Problem

Given a set of n points, our next problem is to find two points whose Euclidean distance is minimum. For example, if the points are



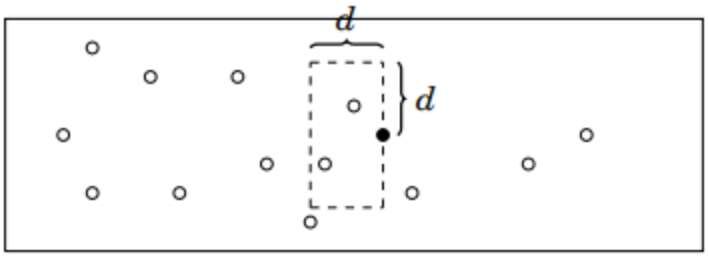
We should find the following points:



This is another example of a problem that can be solved in O(nlogn) time using a sweep line algorithm1 . We go through the points from left to right and maintain a value d: the minimum distance between two points seen so far. At each point, we find the nearest point to the left. If the distance is less than d, it is the new minimum distance and we update the value of d.

If the current point is (x, y) and there is a point to the left within a distance of less than d, the x coordinate of such a point must be between [x − d, x] and the y coordinate must be between [y− d, y+ d]. Thus, it suffices to only consider points that are located in those ranges, which makes the algorithm efficient.

For example, in the following picture, the region marked with dashed lines contains the points that can be within a distance of d from the active point:

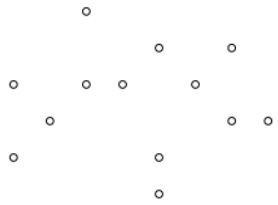


The efficiency of the algorithm is based on the fact that the region always contains only O(1) points. We can go through those points in O(logn) time by maintaining a set of points whose x coordinate is between [x − d, x], in increasing order according to their y coordinates.

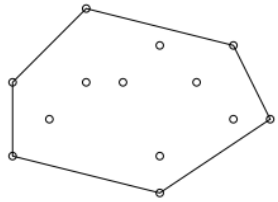
The time complexity of the algorithm is O(nlogn), because we go through n points and find for each point the nearest point to the left in O(logn) time.

39.3 Convex Hull Problem

A convex hull is the smallest convex polygon that contains all points of a given set. Convexity means that a line segment between any two vertices of the polygon is completely inside the polygon. For example, for the points



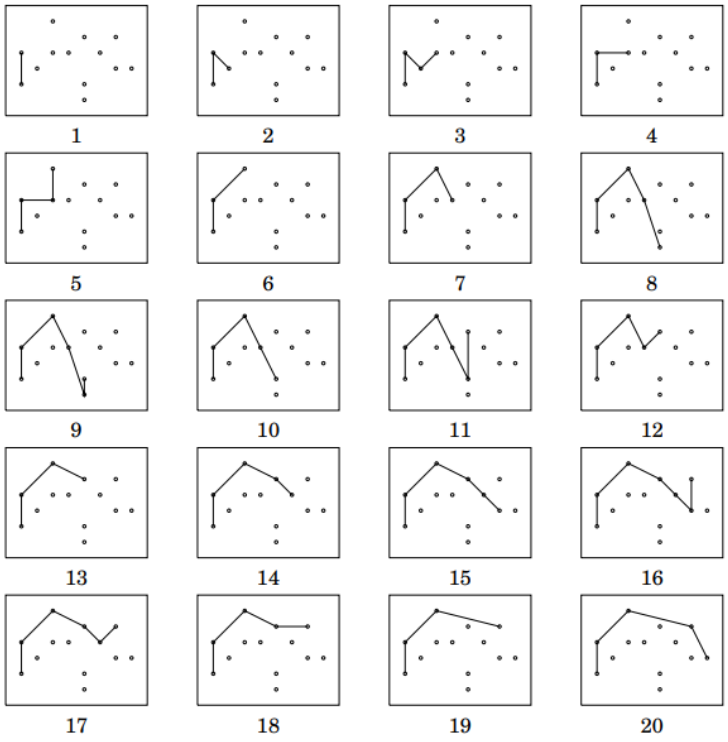
The convex hull is as follows:



Andrew's algorithm provides an easy way to construct the convex hull for a set of points in O(nlogn) time. The algorithm first locates the leftmost and rightmost points, and then constructs the convex hull in two parts: first the upper hull and then the lower hull. Both parts are similar, so we can focus on constructing the upper hull.

First, we sort the points primarily according to x coordinates and secondarily according to y coordinates. After this, we go through the points and add each point to the hull. Always after adding a point to the hull, we make sure that the last line segment in the hull does not turn left. As long as it turns left, we repeatedly remove the second last point from the hull.

The following pictures show how Andrew's algorithm works:

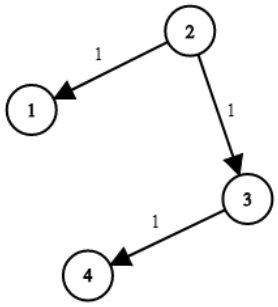


40. Network Flow

40.1 Network Delay Time

You are given a network of n nodes, labeled from 1 to n. You are also given times, a list of travel times as directed edges times[i] = (u_i, v_i, w_i), where u_i is the source node, v_i is the target node, and w_i is the time it takes for a signal to travel from source to target.

We will send a signal from a given node k. Return the **minimum** time it takes for all the n nodes to receive the signal. If it is impossible for all the n nodes to receive the signal, return -1.



Input: times = [[2, 1, 1], [2, 3, 1], [3, 4, 1]], n = 4, k = 2

Output: 2

Input: times = [[1, 2, 1]], n = 2, k = 1

Output: 1

Input: times = [[1, 2, 1]], n = 2, k = 2

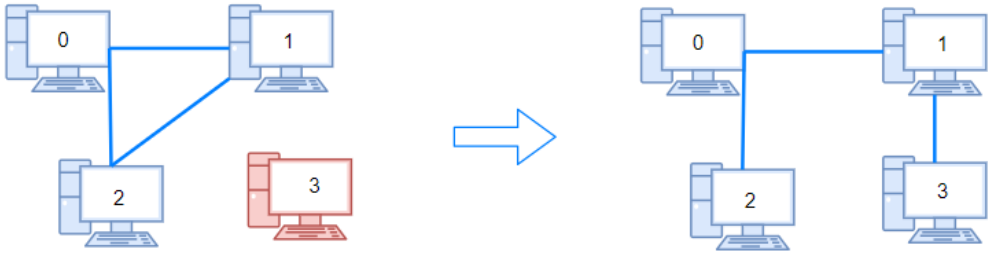
Output: -1

40.2 Number of Operations to Make Network Connected

There are n computers numbered from 0 to n - 1 connected by ethernet cables connections forming a network where connections[i] = [a_i, b_i] represents a connection between computers a_i and b_i. Any computer can reach any other computer directly or indirectly through the network.

You are given an initial computer network connections. You can extract certain cables between two directly connected computers, and place them between any pair of disconnected computers to make them directly connected.

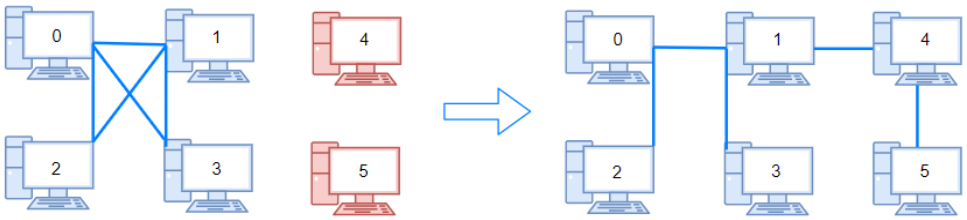
Return the minimum number of times you need to do this in order to make all the computers connected. If it is not possible, return -1.



Input: n = 4, connections = [[0, 1], [0, 2], [1, 2]]

Output: 1

Explanation: Remove cable between computer 1 and 2 and place between computers 1 and 3.



Input: n = 6, connections = [[0, 1], [0, 2], [0, 3], [1, 2], [1, 3]]

Output: 2

Input: n = 6, connections = [[0, 1], [0, 2], [0, 3], [1, 2]]

Output: -1

Explanation: There are not enough cables.