# **359. Logger Rate Limiter**

https://leetcode.com/problems/logger-rate-limiter/description/

Design a logger system that receives a stream of messages along with their timestamps. Each **unique** message should only be printed **at most every 10 seconds** (i.e. a message printed at timestamp t will prevent other identical messages from being printed until timestamp t + 10).

All messages will come in chronological order. Several messages may arrive at the same timestamp.

Implement the Logger class:

* Logger() Initializes the logger object.
* bool shouldPrintMessage(int timestamp, string message) Returns true if the message should be printed in the given timestamp, otherwise returns false.

**Example 1:**

Input  
["Logger", "shouldPrintMessage", "shouldPrintMessage", "shouldPrintMessage", "shouldPrintMessage", "shouldPrintMessage", "shouldPrintMessage"]  
[[], [1, "foo"], [2, "bar"], [3, "foo"], [8, "bar"], [10, "foo"], [11, "foo"]]  
Output  
[null, true, true, false, false, false, true]  
  
Explanation  
Logger logger = new Logger();  
logger.shouldPrintMessage(1, "foo"); // return true, next allowed timestamp for "foo" is 1 + 10 = 11  
logger.shouldPrintMessage(2, "bar"); // return true, next allowed timestamp for "bar" is 2 + 10 = 12  
logger.shouldPrintMessage(3, "foo"); // 3 < 11, return false  
logger.shouldPrintMessage(8, "bar"); // 8 < 12, return false  
logger.shouldPrintMessage(10, "foo"); // 10 < 11, return false  
logger.shouldPrintMessage(11, "foo"); // 11 >= 11, return true, next allowed timestamp for "foo" is 11 + 10 = 21

**Constraints:**

* 0 <= timestamp <= 109
* Every timestamp will be passed in non-decreasing order (chronological order).
* 1 <= message.length <= 30
* At most 104 calls will be made to shouldPrintMessage.

# **1110. Delete Nodes And Return Forest**

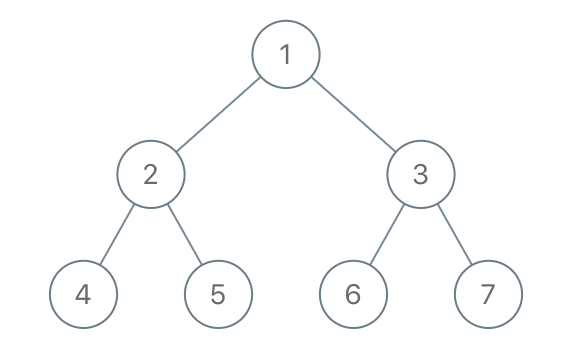
https://leetcode.com/problems/delete-nodes-and-return-forest/description/

Given the root of a binary tree, each node in the tree has a distinct value.

After deleting all nodes with a value in to\_delete, we are left with a forest (a disjoint union of trees).

Return the roots of the trees in the remaining forest. You may return the result in any order.

**Example 1:**



Input: root = [1,2,3,4,5,6,7], to\_delete = [3,5]  
Output: [[1,2,null,4],[6],[7]]

**Example 2:**

Input: root = [1,2,4,null,3], to\_delete = [3]  
Output: [[1,2,4]]

**Constraints:**

* The number of nodes in the given tree is at most 1000.
* Each node has a distinct value between 1 and 1000.
* to\_delete.length <= 1000
* to\_delete contains distinct values between 1 and 1000.

# **2402. Meeting Rooms III**

https://leetcode.com/problems/meeting-rooms-iii/description/

You are given an integer n. There are n rooms numbered from 0 to n - 1.

You are given a 2D integer array meetings where meetings[i] = [starti, endi] means that a meeting will be held during the **half-closed** time interval [starti, endi). All the values of starti are **unique**.

Meetings are allocated to rooms in the following manner:

1. Each meeting will take place in the unused room with the **lowest** number.
2. If there are no available rooms, the meeting will be delayed until a room becomes free. The delayed meeting should have the **same** duration as the original meeting.
3. When a room becomes unused, meetings that have an earlier original **start** time should be given the room.

Return *the* ***number*** *of the room that held the most meetings.* If there are multiple rooms, return *the room with the* ***lowest*** *number.*

A **half-closed interval** [a, b) is the interval between a and b **including** a and **not including** b.

**Example 1:**

Input: n = 2, meetings = [[0,10],[1,5],[2,7],[3,4]]  
Output: 0  
Explanation:  
- At time 0, both rooms are not being used. The first meeting starts in room 0.  
- At time 1, only room 1 is not being used. The second meeting starts in room 1.  
- At time 2, both rooms are being used. The third meeting is delayed.  
- At time 3, both rooms are being used. The fourth meeting is delayed.  
- At time 5, the meeting in room 1 finishes. The third meeting starts in room 1 for the time period [5,10).  
- At time 10, the meetings in both rooms finish. The fourth meeting starts in room 0 for the time period [10,11).  
Both rooms 0 and 1 held 2 meetings, so we return 0.

**Example 2:**

Input: n = 3, meetings = [[1,20],[2,10],[3,5],[4,9],[6,8]]  
Output: 1  
Explanation:  
- At time 1, all three rooms are not being used. The first meeting starts in room 0.  
- At time 2, rooms 1 and 2 are not being used. The second meeting starts in room 1.  
- At time 3, only room 2 is not being used. The third meeting starts in room 2.  
- At time 4, all three rooms are being used. The fourth meeting is delayed.  
- At time 5, the meeting in room 2 finishes. The fourth meeting starts in room 2 for the time period [5,10).  
- At time 6, all three rooms are being used. The fifth meeting is delayed.  
- At time 10, the meetings in rooms 1 and 2 finish. The fifth meeting starts in room 1 for the time period [10,12).  
Room 0 held 1 meeting while rooms 1 and 2 each held 2 meetings, so we return 1.

**Constraints:**

* 1 <= n <= 100
* 1 <= meetings.length <= 105
* meetings[i].length == 2
* 0 <= starti < endi <= 5 \* 105
* All the values of starti are **unique**.

# **2458. Height of Binary Tree After Subtree Removal Queries**

https://leetcode.com/problems/height-of-binary-tree-after-subtree-removal-queries/description/

You are given the root of a **binary tree** with n nodes. Each node is assigned a unique value from 1 to n. You are also given an array queries of size m.

You have to perform m **independent** queries on the tree where in the ith query you do the following:

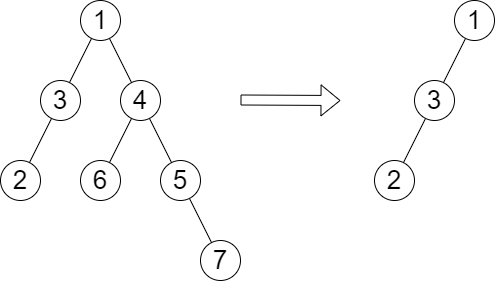
* **Remove** the subtree rooted at the node with the value queries[i] from the tree. It is **guaranteed** that queries[i] will **not** be equal to the value of the root.

Return *an array* answer *of size* m *where* answer[i] *is the height of the tree after performing the* ith *query*.

**Note**:

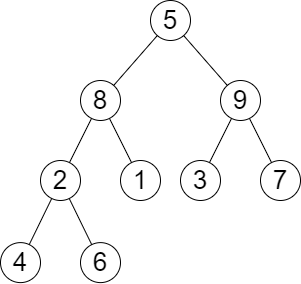
* The queries are independent, so the tree returns to its **initial** state after each query.
* The height of a tree is the **number of edges in the longest simple path** from the root to some node in the tree.

**Example 1:**



Input: root = [1,3,4,2,null,6,5,null,null,null,null,null,7], queries = [4]  
Output: [2]  
Explanation: The diagram above shows the tree after removing the subtree rooted at node with value 4.  
The height of the tree is 2 (The path 1 -> 3 -> 2).

**Example 2:**



Input: root = [5,8,9,2,1,3,7,4,6], queries = [3,2,4,8]  
Output: [3,2,3,2]  
Explanation: We have the following queries:  
- Removing the subtree rooted at node with value 3. The height of the tree becomes 3 (The path 5 -> 8 -> 2 -> 4).  
- Removing the subtree rooted at node with value 2. The height of the tree becomes 2 (The path 5 -> 8 -> 1).  
- Removing the subtree rooted at node with value 4. The height of the tree becomes 3 (The path 5 -> 8 -> 2 -> 6).  
- Removing the subtree rooted at node with value 8. The height of the tree becomes 2 (The path 5 -> 9 -> 3).

**Constraints:**

* The number of nodes in the tree is n.
* 2 <= n <= 105
* 1 <= Node.val <= n
* All the values in the tree are **unique**.
* m == queries.length
* 1 <= m <= min(n, 104)
* 1 <= queries[i] <= n
* queries[i] != root.val

# **715. Range Module**

https://leetcode.com/problems/range-module/description/

A Range Module is a module that tracks ranges of numbers. Design a data structure to track the ranges represented as **half-open intervals** and query about them.

A **half-open interval** [left, right) denotes all the real numbers x where left <= x < right.

Implement the RangeModule class:

* RangeModule() Initializes the object of the data structure.
* void addRange(int left, int right) Adds the **half-open interval** [left, right), tracking every real number in that interval. Adding an interval that partially overlaps with currently tracked numbers should add any numbers in the interval [left, right) that are not already tracked.
* boolean queryRange(int left, int right) Returns true if every real number in the interval [left, right) is currently being tracked, and false otherwise.
* void removeRange(int left, int right) Stops tracking every real number currently being tracked in the **half-open interval** [left, right).

**Example 1:**

Input  
["RangeModule", "addRange", "removeRange", "queryRange", "queryRange", "queryRange"]  
[[], [10, 20], [14, 16], [10, 14], [13, 15], [16, 17]]  
Output  
[null, null, null, true, false, true]  
  
Explanation  
RangeModule rangeModule = new RangeModule();  
rangeModule.addRange(10, 20);  
rangeModule.removeRange(14, 16);  
rangeModule.queryRange(10, 14); // return True,(Every number in [10, 14) is being tracked)  
rangeModule.queryRange(13, 15); // return False,(Numbers like 14, 14.03, 14.17 in [13, 15) are not being tracked)  
rangeModule.queryRange(16, 17); // return True, (The number 16 in [16, 17) is still being tracked, despite the remove operation)

**Constraints:**

* 1 <= left < right <= 109
* At most 104 calls will be made to addRange, queryRange, and removeRange.

# **1825. Finding MK Average**

https://leetcode.com/problems/finding-mk-average/description/

You are given two integers, m and k, and a stream of integers. You are tasked to implement a data structure that calculates the **MKAverage** for the stream.

The **MKAverage** can be calculated using these steps:

1. If the number of the elements in the stream is less than m you should consider the **MKAverage** to be -1. Otherwise, copy the last m elements of the stream to a separate container.
2. Remove the smallest k elements and the largest k elements from the container.
3. Calculate the average value for the rest of the elements **rounded down to the nearest integer**.

Implement the MKAverage class:

* MKAverage(int m, int k) Initializes the **MKAverage** object with an empty stream and the two integers m and k.
* void addElement(int num) Inserts a new element num into the stream.
* int calculateMKAverage() Calculates and returns the **MKAverage** for the current stream **rounded down to the nearest integer**.

**Example 1:**

Input  
["MKAverage", "addElement", "addElement", "calculateMKAverage", "addElement", "calculateMKAverage", "addElement", "addElement", "addElement", "calculateMKAverage"]  
[[3, 1], [3], [1], [], [10], [], [5], [5], [5], []]  
Output  
[null, null, null, -1, null, 3, null, null, null, 5]  
  
Explanation  
MKAverage obj = new MKAverage(3, 1);   
obj.addElement(3); // current elements are [3]  
obj.addElement(1); // current elements are [3,1]  
obj.calculateMKAverage(); // return -1, because m = 3 and only 2 elements exist.  
obj.addElement(10); // current elements are [3,1,10]  
obj.calculateMKAverage(); // The last 3 elements are [3,1,10].  
 // After removing smallest and largest 1 element the container will be [3].  
 // The average of [3] equals 3/1 = 3, return 3  
obj.addElement(5); // current elements are [3,1,10,5]  
obj.addElement(5); // current elements are [3,1,10,5,5]  
obj.addElement(5); // current elements are [3,1,10,5,5,5]  
obj.calculateMKAverage(); // The last 3 elements are [5,5,5].  
 // After removing smallest and largest 1 element the container will be [5].  
 // The average of [5] equals 5/1 = 5, return 5

**Constraints:**

* 3 <= m <= 105
* 1 <= k\*2 < m
* 1 <= num <= 105
* At most 105 calls will be made to addElement and calculateMKAverage.

# **2265. Count Nodes Equal to Average of Subtree**

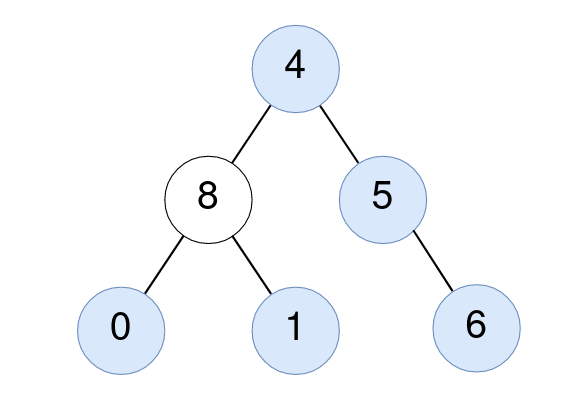
https://leetcode.com/problems/count-nodes-equal-to-average-of-subtree/description/

Given the root of a binary tree, return *the number of nodes where the value of the node is equal to the* ***average*** *of the values in its* ***subtree***.

**Note:**

* The **average** of n elements is the **sum** of the n elements divided by n and **rounded down** to the nearest integer.
* A **subtree** of root is a tree consisting of root and all of its descendants.

**Example 1:**



Input: root = [4,8,5,0,1,null,6]  
Output: 5  
Explanation:   
For the node with value 4: The average of its subtree is (4 + 8 + 5 + 0 + 1 + 6) / 6 = 24 / 6 = 4.  
For the node with value 5: The average of its subtree is (5 + 6) / 2 = 11 / 2 = 5.  
For the node with value 0: The average of its subtree is 0 / 1 = 0.  
For the node with value 1: The average of its subtree is 1 / 1 = 1.  
For the node with value 6: The average of its subtree is 6 / 1 = 6.

**Example 2:**



Input: root = [1]  
Output: 1  
Explanation: For the node with value 1: The average of its subtree is 1 / 1 = 1.

**Constraints:**

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

# **1136. Parallel Courses**

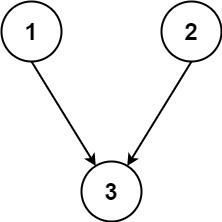
https://leetcode.com/problems/parallel-courses/description/

You are given an integer n, which indicates that there are n courses labeled from 1 to n. You are also given an array relations where relations[i] = [prevCoursei, nextCoursei], representing a prerequisite relationship between course prevCoursei and course nextCoursei: course prevCoursei has to be taken before course nextCoursei.

In one semester, you can take **any number** of courses as long as you have taken all the prerequisites in the **previous** semester for the courses you are taking.

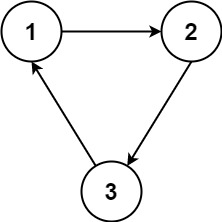
Return *the* ***minimum*** *number of semesters needed to take all courses*. If there is no way to take all the courses, return -1.

**Example 1:**



Input: n = 3, relations = [[1,3],[2,3]]  
Output: 2  
Explanation: The figure above represents the given graph.  
In the first semester, you can take courses 1 and 2.  
In the second semester, you can take course 3.

**Example 2:**



Input: n = 3, relations = [[1,2],[2,3],[3,1]]  
Output: -1  
Explanation: No course can be studied because they are prerequisites of each other.

**Constraints:**

* 1 <= n <= 5000
* 1 <= relations.length <= 5000
* relations[i].length == 2
* 1 <= prevCoursei, nextCoursei <= n
* prevCoursei != nextCoursei
* All the pairs [prevCoursei, nextCoursei] are **unique**.

# **2101. Detonate the Maximum Bombs**

https://leetcode.com/problems/detonate-the-maximum-bombs/description/

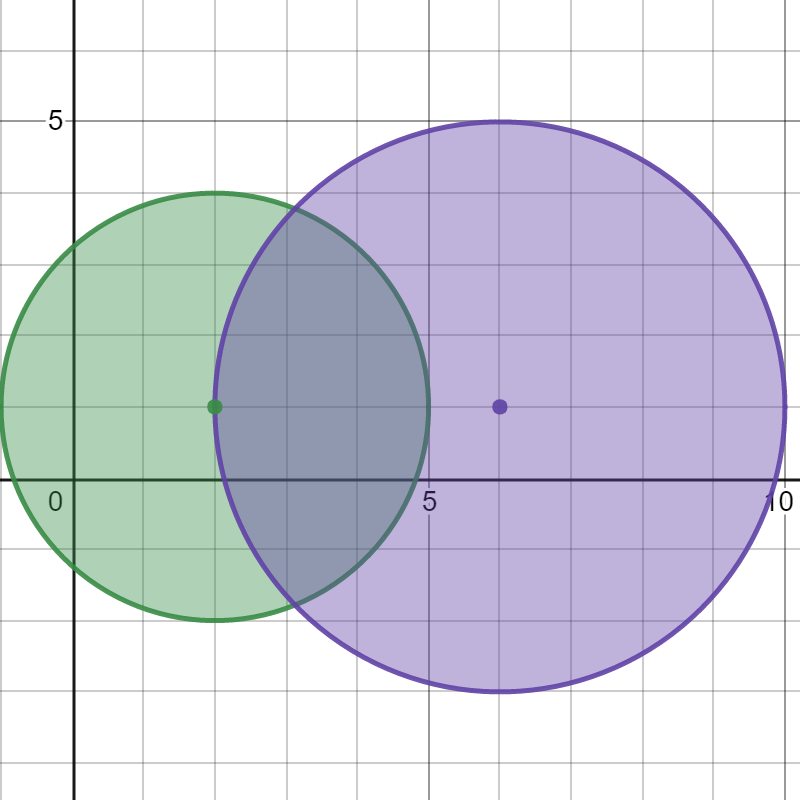
You are given a list of bombs. The **range** of a bomb is defined as the area where its effect can be felt. This area is in the shape of a **circle** with the center as the location of the bomb.

The bombs are represented by a **0-indexed** 2D integer array bombs where bombs[i] = [xi, yi, ri]. xi and yi denote the X-coordinate and Y-coordinate of the location of the ith bomb, whereas ri 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.

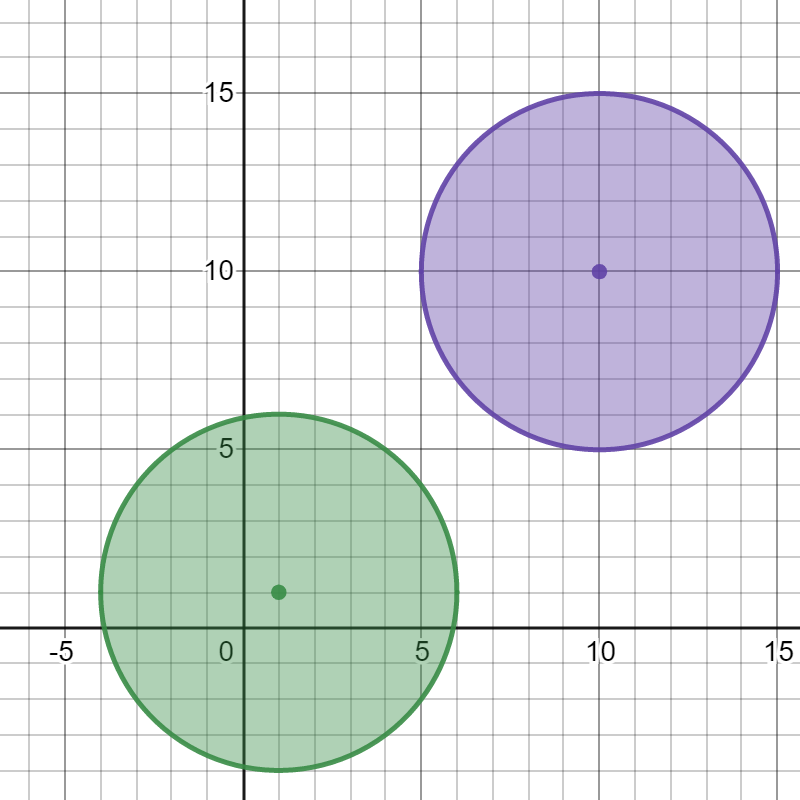
Given the list of bombs, return *the* ***maximum*** *number of bombs that can be detonated if you are allowed to detonate* ***only one*** *bomb*.

**Example 1:**



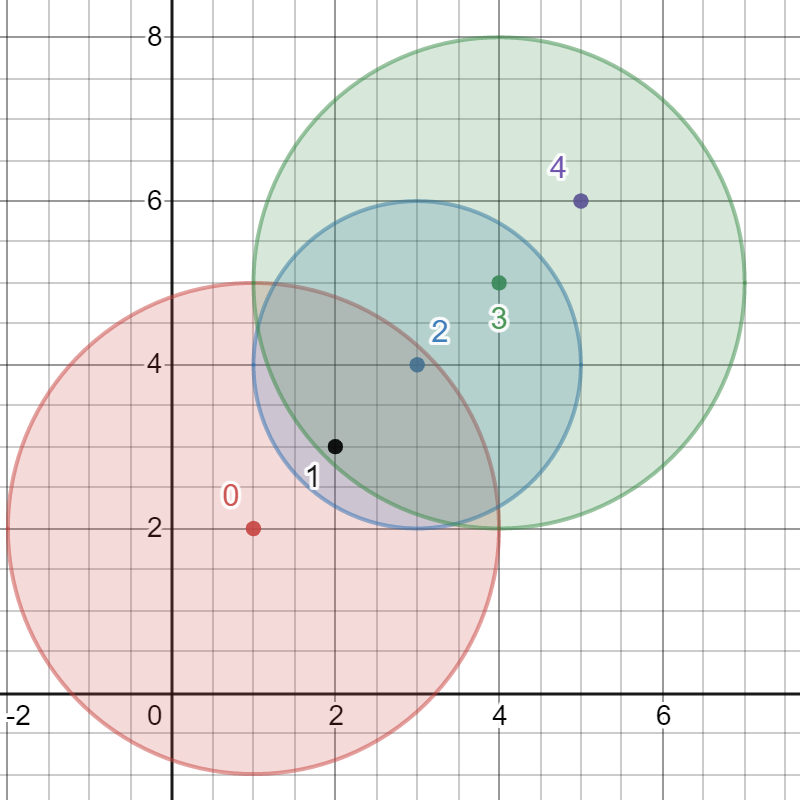
Input: bombs = [[2,1,3],[6,1,4]]  
Output: 2  
Explanation:  
The above figure shows the positions and ranges of the 2 bombs.  
If we detonate the left bomb, the right bomb will not be affected.  
But if we detonate the right bomb, both bombs will be detonated.  
So the maximum bombs that can be detonated is max(1, 2) = 2.

**Example 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.

**Example 3:**



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.

**Constraints:**

* 1 <= bombs.length <= 100
* bombs[i].length == 3
* 1 <= xi, yi, ri <= 105

# **939. Minimum Area Rectangle**

https://leetcode.com/problems/minimum-area-rectangle/description/

You are given an array of points in the **X-Y** plane points where points[i] = [xi, yi].

Return *the minimum area of a rectangle formed from these points, with sides parallel to the X and Y axes*. If there is not any such rectangle, return 0.

**Example 1:**

![](data:text/html; charset=UTF-8;base64,)

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

**Example 2:**

![](data:text/html; charset=UTF-8;base64,)

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

**Constraints:**

* 1 <= points.length <= 500
* points[i].length == 2
* 0 <= xi, yi <= 4 \* 104
* All the given points are **unique**.

# **2437. Number of Valid Clock Times**

https://leetcode.com/problems/number-of-valid-clock-times/description/

You are given a string of length 5 called time, representing the current time on a digital clock in the format "hh:mm". The **earliest** possible time is "00:00" and the **latest** possible time is "23:59".

In the string time, the digits represented by the ? symbol are **unknown**, and must be **replaced** with a digit from 0 to 9.

Return *an integer* answer*, the number of valid clock times that can be created by replacing every* ?*with a digit from* 0 *to* 9.

**Example 1:**

Input: time = "?5:00"  
Output: 2  
Explanation: We can replace the ? with either a 0 or 1, producing "05:00" or "15:00". Note that we cannot replace it with a 2, since the time "25:00" is invalid. In total, we have two choices.

**Example 2:**

Input: time = "0?:0?"  
Output: 100  
Explanation: Each ? can be replaced by any digit from 0 to 9, so we have 100 total choices.

**Example 3:**

Input: time = "??:??"  
Output: 1440  
Explanation: There are 24 possible choices for the hours, and 60 possible choices for the minutes. In total, we have 24 \* 60 = 1440 choices.

**Constraints:**

* time is a valid string of length 5 in the format "hh:mm".
* "00" <= hh <= "23"
* "00" <= mm <= "59"
* Some of the digits might be replaced with '?' and need to be replaced with digits from 0 to 9.

# **1101. The Earliest Moment When Everyone Become Friends**

https://leetcode.com/problems/the-earliest-moment-when-everyone-become-friends/description/

There are n people in a social group labeled from 0 to n - 1. You are given an array logs where logs[i] = [timestampi, xi, yi] indicates that xi and yi will be friends at the time timestampi.

Friendship is **symmetric**. That means if a is friends with b, then b is friends with a. Also, person a is acquainted with a person b if a is friends with b, or a is a friend of someone acquainted with b.

Return *the earliest time for which every person became acquainted with every other person*. If there is no such earliest time, return -1.

**Example 1:**

Input: logs = [[20190101,0,1],[20190104,3,4],[20190107,2,3],[20190211,1,5],[20190224,2,4],[20190301,0,3],[20190312,1,2],[20190322,4,5]], n = 6  
Output: 20190301  
Explanation:   
The first event occurs at timestamp = 20190101, and after 0 and 1 become friends, we have the following friendship groups [0,1], [2], [3], [4], [5].  
The second event occurs at timestamp = 20190104, and after 3 and 4 become friends, we have the following friendship groups [0,1], [2], [3,4], [5].  
The third event occurs at timestamp = 20190107, and after 2 and 3 become friends, we have the following friendship groups [0,1], [2,3,4], [5].  
The fourth event occurs at timestamp = 20190211, and after 1 and 5 become friends, we have the following friendship groups [0,1,5], [2,3,4].  
The fifth event occurs at timestamp = 20190224, and as 2 and 4 are already friends, nothing happens.  
The sixth event occurs at timestamp = 20190301, and after 0 and 3 become friends, we all become friends.

**Example 2:**

Input: logs = [[0,2,0],[1,0,1],[3,0,3],[4,1,2],[7,3,1]], n = 4  
Output: 3  
Explanation: At timestamp = 3, all the persons (i.e., 0, 1, 2, and 3) become friends.

**Constraints:**

* 2 <= n <= 100
* 1 <= logs.length <= 104
* logs[i].length == 3
* 0 <= timestampi <= 109
* 0 <= xi, yi <= n - 1
* xi != yi
* All the values timestampi are **unique**.
* All the pairs (xi, yi) occur at most one time in the input.

# **2700. Differences Between Two Objects**

https://leetcode.com/problems/differences-between-two-objects/description/

Write a function that accepts two deeply nested objects or arrays obj1 and obj2 and returns a new object representing their differences.

The function should compare the properties of the two objects and identify any changes. The returned object should only contains keys where the value is different from obj1 to obj2.

For each changed key, the value should be represented as an array [obj1 value, obj2 value]. Keys that exist in one object but not in the other should not be included in the returned object. When comparing two arrays, the indices of the arrays are considered to be their keys. The end result should be a deeply nested object where each leaf value is a difference array.

You may assume that both objects are the output of JSON.parse.

**Example 1:**

Input:   
obj1 = {}  
obj2 = {  
  "a": 1,   
 "b": 2  
}  
Output: {}  
Explanation: There were no modifications made to obj1. New keys "a" and "b" appear in obj2, but keys that are added or removed should be ignored.

**Example 2:**

Input:   
obj1 = {  
  "a": 1,  
  "v": 3,  
  "x": [],  
  "z": {  
    "a": null  
  }  
}  
obj2 = {  
  "a": 2,  
  "v": 4,  
  "x": [],  
  "z": {  
    "a": 2  
  }  
}  
Output:   
{  
  "a": [1, 2],  
 "v": [3, 4],  
  "z": {  
  "a": [null, 2]  
  }  
}  
Explanation: The keys "a", "v", and "z" all had changes applied. "a" was changed from 1 to 2. "v" was changed from 3 to 4. "z" had a change applied to a child object. "z.a" was changed from null to 2.

**Example 3:**

Input:   
obj1 = {  
  "a": 5,   
  "v": 6,   
  "z": [1, 2, 4, [2, 5, 7]]  
}  
obj2 = {  
  "a": 5,   
  "v": 7,   
  "z": [1, 2, 3, [1]]  
}  
Output:   
{  
  "v": [6, 7],  
  "z": {  
  "2": [4, 3],  
  "3": {  
  "0": [2, 1]  
  }  
  }  
}  
Explanation: In obj1 and obj2, the keys "v" and "z" have different assigned values. "a" is ignored because the value is unchanged. In the key "z", there is a nested array. Arrays are treated like objects where the indices are keys. There were two alterations to the the array: z[2] and z[3][0]. z[0] and z[1] were unchanged and thus not included. z[3][1] and z[3][2] were removed and thus not included.

**Example 4:**

Input:   
obj1 = {  
  "a": {"b": 1},   
}  
obj2 = {  
  "a": [5],  
}  
Output:   
{  
 "a": [{"b": 1}, [5]]  
}  
Explanation: The key "a" exists in both objects. Since the two associated values have different types, they are placed in the difference array.

**Example 5:**

Input:   
obj1 = {  
  "a": [1, 2, {}],   
  "b": false  
}  
obj2 = {     
  "b": false,  
  "a": [1, 2, {}]  
}  
Output:   
{}  
Explanation: Apart from a different ordering of keys, the two objects are identical so an empty object is returned.

**Constraints:**

* obj1 and obj2 are valid JSON objects or arrays
* 2 <= JSON.stringify(obj1).length <= 104
* 2 <= JSON.stringify(obj2).length <= 104

# **2863. Maximum Length of Semi-Decreasing Subarrays**

https://leetcode.com/problems/maximum-length-of-semi-decreasing-subarrays/description/

You are given an integer array nums.

Return *the length of the* ***longest semi-decreasing*** *subarray of* nums*, and* 0 *if there are no such subarrays.*

* A **subarray** is a contiguous non-empty sequence of elements within an array.
* A non-empty array is **semi-decreasing** if its first element is **strictly greater** than its last element.

**Example 1:**

Input: nums = [7,6,5,4,3,2,1,6,10,11]  
Output: 8  
Explanation: Take the subarray [7,6,5,4,3,2,1,6].  
The first element is 7 and the last one is 6 so the condition is met.  
Hence, the answer would be the length of the subarray or 8.  
It can be shown that there aren't any subarrays with the given condition with a length greater than 8.

**Example 2:**

Input: nums = [57,55,50,60,61,58,63,59,64,60,63]  
Output: 6  
Explanation: Take the subarray [61,58,63,59,64,60].  
The first element is 61 and the last one is 60 so the condition is met.  
Hence, the answer would be the length of the subarray or 6.  
It can be shown that there aren't any subarrays with the given condition with a length greater than 6.

**Example 3:**

Input: nums = [1,2,3,4]  
Output: 0  
Explanation: Since there are no semi-decreasing subarrays in the given array, the answer is 0.

**Constraints:**

* 1 <= nums.length <= 105
* -109 <= nums[i] <= 109

# **2842. Count K-Subsequences of a String With Maximum Beauty**

https://leetcode.com/problems/count-k-subsequences-of-a-string-with-maximum-beauty/description/

You are given a string s and an integer k.

A **k-subsequence** is a **subsequence** of s, having length k, and all its characters are **unique**, **i.e**., every character occurs once.

Let f(c) denote the number of times the character c occurs in s.

The **beauty** of a **k-subsequence** is the **sum** of f(c) for every character c in the k-subsequence.

For example, consider s = "abbbdd" and k = 2:

* f('a') = 1, f('b') = 3, f('d') = 2
* Some k-subsequences of s are:
  + "**ab**bbdd" -> "ab" having a beauty of f('a') + f('b') = 4
  + "**a**bbb**d**d" -> "ad" having a beauty of f('a') + f('d') = 3
  + "a**b**bb**d**d" -> "bd" having a beauty of f('b') + f('d') = 5

Return *an integer denoting the number of k-subsequences* *whose* ***beauty*** *is the* ***maximum*** *among all* ***k-subsequences***. Since the answer may be too large, return it modulo 109 + 7.

A subsequence of a string is a new string formed from the original string by deleting some (possibly none) of the characters without disturbing the relative positions of the remaining characters.

**Notes**

* f(c) is the number of times a character c occurs in s, not a k-subsequence.
* Two k-subsequences are considered different if one is formed by an index that is not present in the other. So, two k-subsequences may form the same string.

**Example 1:**

Input: s = "bcca", k = 2  
Output: 4  
Explanation: From s we have f('a') = 1, f('b') = 1, and f('c') = 2.  
The k-subsequences of s are:   
bcca having a beauty of f('b') + f('c') = 3   
bcca having a beauty of f('b') + f('c') = 3   
bcca having a beauty of f('b') + f('a') = 2   
bcca having a beauty of f('c') + f('a') = 3  
bcca having a beauty of f('c') + f('a') = 3   
There are 4 k-subsequences that have the maximum beauty, 3.   
Hence, the answer is 4.

**Example 2:**

Input: s = "abbcd", k = 4  
Output: 2  
Explanation: From s we have f('a') = 1, f('b') = 2, f('c') = 1, and f('d') = 1.   
The k-subsequences of s are:   
abbcd having a beauty of f('a') + f('b') + f('c') + f('d') = 5  
abbcd having a beauty of f('a') + f('b') + f('c') + f('d') = 5   
There are 2 k-subsequences that have the maximum beauty, 5.   
Hence, the answer is 2.

**Constraints:**

* 1 <= s.length <= 2 \* 105
* 1 <= k <= s.length
* s consists only of lowercase English letters.

# **2116. Check if a Parentheses String Can Be Valid**

https://leetcode.com/problems/check-if-a-parentheses-string-can-be-valid/description/

A parentheses string is a **non-empty** string consisting only of '(' and ')'. It is valid if **any** of the following conditions is **true**:

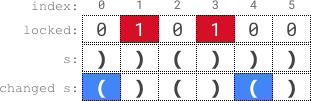
* It is ().
* It can be written as AB (A concatenated with B), where A and B are valid parentheses strings.
* It can be written as (A), where A is a valid parentheses string.

You are given a parentheses string s and a string locked, both of length n. locked is a binary string consisting only of '0's and '1's. For **each** index i of locked,

* If locked[i] is '1', you **cannot** change s[i].
* But if locked[i] is '0', you **can** change s[i] to either '(' or ')'.

Return true *if you can make s a valid parentheses string*. Otherwise, return false.

**Example 1:**



Input: s = "))()))", locked = "010100"  
Output: true  
Explanation: locked[1] == '1' and locked[3] == '1', so we cannot change s[1] or s[3].  
We change s[0] and s[4] to '(' while leaving s[2] and s[5] unchanged to make s valid.

**Example 2:**

Input: s = "()()", locked = "0000"  
Output: true  
Explanation: We do not need to make any changes because s is already valid.

**Example 3:**

Input: s = ")", locked = "0"  
Output: false  
Explanation: locked permits us to change s[0].   
Changing s[0] to either '(' or ')' will not make s valid.

**Constraints:**

* n == s.length == locked.length
* 1 <= n <= 105
* s[i] is either '(' or ')'.
* locked[i] is either '0' or '1'.

# **679. 24 Game**

https://leetcode.com/problems/24-game/description/

You are given an integer array cards of length 4. You have four cards, each containing a number in the range [1, 9]. You should arrange the numbers on these cards in a mathematical expression using the operators ['+', '-', '\*', '/'] and the parentheses '(' and ')' to get the value 24.

You are restricted with the following rules:

* The division operator '/' represents real division, not integer division.
  + For example, 4 / (1 - 2 / 3) = 4 / (1 / 3) = 12.
* Every operation done is between two numbers. In particular, we cannot use '-' as a unary operator.
  + For example, if cards = [1, 1, 1, 1], the expression "-1 - 1 - 1 - 1" is **not allowed**.
* You cannot concatenate numbers together
  + For example, if cards = [1, 2, 1, 2], the expression "12 + 12" is not valid.

Return true if you can get such expression that evaluates to 24, and false otherwise.

**Example 1:**

Input: cards = [4,1,8,7]  
Output: true  
Explanation: (8-4) \* (7-1) = 24

**Example 2:**

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

**Constraints:**

* cards.length == 4
* 1 <= cards[i] <= 9

# **900. RLE Iterator**

https://leetcode.com/problems/rle-iterator/description/

We can use run-length encoding (i.e., **RLE**) to encode a sequence of integers. In a run-length encoded array of even length encoding (**0-indexed**), for all even i, encoding[i] tells us the number of times that the non-negative integer value encoding[i + 1] is repeated in the sequence.

* For example, the sequence arr = [8,8,8,5,5] can be encoded to be encoding = [3,8,2,5]. encoding = [3,8,0,9,2,5] and encoding = [2,8,1,8,2,5] are also valid **RLE** of arr.

Given a run-length encoded array, design an iterator that iterates through it.

Implement the RLEIterator class:

* RLEIterator(int[] encoded) Initializes the object with the encoded array encoded.
* int next(int n) Exhausts the next n elements and returns the last element exhausted in this way. If there is no element left to exhaust, return -1 instead.

**Example 1:**

Input  
["RLEIterator", "next", "next", "next", "next"]  
[[[3, 8, 0, 9, 2, 5]], [2], [1], [1], [2]]  
Output  
[null, 8, 8, 5, -1]  
  
Explanation  
RLEIterator rLEIterator = new RLEIterator([3, 8, 0, 9, 2, 5]); // This maps to the sequence [8,8,8,5,5].  
rLEIterator.next(2); // exhausts 2 terms of the sequence, returning 8. The remaining sequence is now [8, 5, 5].  
rLEIterator.next(1); // exhausts 1 term of the sequence, returning 8. The remaining sequence is now [5, 5].  
rLEIterator.next(1); // exhausts 1 term of the sequence, returning 5. The remaining sequence is now [5].  
rLEIterator.next(2); // exhausts 2 terms, returning -1. This is because the first term exhausted was 5,  
but the second term did not exist. Since the last term exhausted does not exist, we return -1.

**Constraints:**

* 2 <= encoding.length <= 1000
* encoding.length is even.
* 0 <= encoding[i] <= 109
* 1 <= n <= 109
* At most 1000 calls will be made to next.

# **975. Odd Even Jump**

https://leetcode.com/problems/odd-even-jump/description/

You are given an integer array arr. From some starting index, you can make a series of jumps. The (1st, 3rd, 5th, ...) jumps in the series are called **odd-numbered jumps**, and the (2nd, 4th, 6th, ...) jumps in the series are called **even-numbered jumps**. Note that the **jumps** are numbered, not the indices.

You may jump forward from index i to index j (with i < j) in the following way:

* During **odd-numbered jumps** (i.e., jumps 1, 3, 5, ...), you jump to the index j such that arr[i] <= arr[j] and arr[j] is the smallest possible value. If there are multiple such indices j, you can only jump to the **smallest** such index j.
* During **even-numbered jumps** (i.e., jumps 2, 4, 6, ...), you jump to the index j such that arr[i] >= arr[j] and arr[j] is the largest possible value. If there are multiple such indices j, you can only jump to the **smallest** such index j.
* It may be the case that for some index i, there are no legal jumps.

A starting index is **good** if, starting from that index, you can reach the end of the array (index arr.length - 1) by jumping some number of times (possibly 0 or more than once).

Return *the number of* ***good*** *starting indices*.

**Example 1:**

Input: arr = [10,13,12,14,15]  
Output: 2  
Explanation:   
From starting index i = 0, we can make our 1st jump to i = 2 (since arr[2] is the smallest among arr[1], arr[2], arr[3], arr[4] that is greater or equal to arr[0]), then we cannot jump any more.  
From starting index i = 1 and i = 2, we can make our 1st jump to i = 3, then we cannot jump any more.  
From starting index i = 3, we can make our 1st jump to i = 4, so we have reached the end.  
From starting index i = 4, we have reached the end already.  
In total, there are 2 different starting indices i = 3 and i = 4, where we can reach the end with some number of  
jumps.

**Example 2:**

Input: arr = [2,3,1,1,4]  
Output: 3  
Explanation:   
From starting index i = 0, we make jumps to i = 1, i = 2, i = 3:  
During our 1st jump (odd-numbered), we first jump to i = 1 because arr[1] is the smallest value in [arr[1], arr[2], arr[3], arr[4]] that is greater than or equal to arr[0].  
During our 2nd jump (even-numbered), we jump from i = 1 to i = 2 because arr[2] is the largest value in [arr[2], arr[3], arr[4]] that is less than or equal to arr[1]. arr[3] is also the largest value, but 2 is a smaller index, so we can only jump to i = 2 and not i = 3  
During our 3rd jump (odd-numbered), we jump from i = 2 to i = 3 because arr[3] is the smallest value in [arr[3], arr[4]] that is greater than or equal to arr[2].  
We can't jump from i = 3 to i = 4, so the starting index i = 0 is not good.  
In a similar manner, we can deduce that:  
From starting index i = 1, we jump to i = 4, so we reach the end.  
From starting index i = 2, we jump to i = 3, and then we can't jump anymore.  
From starting index i = 3, we jump to i = 4, so we reach the end.  
From starting index i = 4, we are already at the end.  
In total, there are 3 different starting indices i = 1, i = 3, and i = 4, where we can reach the end with some  
number of jumps.

**Example 3:**

Input: arr = [5,1,3,4,2]  
Output: 3  
Explanation: We can reach the end from starting indices 1, 2, and 4.

**Constraints:**

* 1 <= arr.length <= 2 \* 104
* 0 <= arr[i] < 105

# **2973. Find Number of Coins to Place in Tree Nodes**

https://leetcode.com/problems/find-number-of-coins-to-place-in-tree-nodes/description/

You are given an **undirected** tree with n nodes labeled from 0 to n - 1, and rooted at node 0. You are given a 2D integer array edges of length n - 1, where edges[i] = [ai, bi] indicates that there is an edge between nodes ai and bi in the tree.

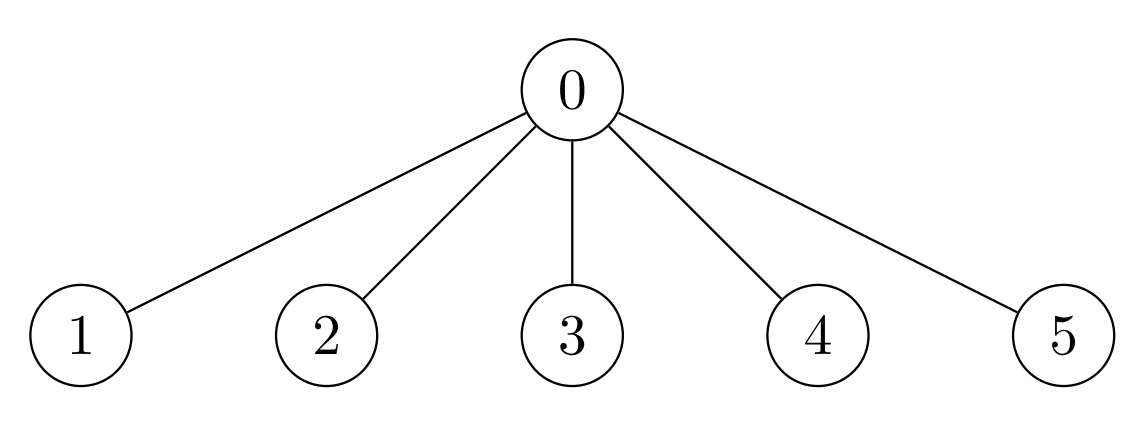
You are also given a **0-indexed** integer array cost of length n, where cost[i] is the **cost** assigned to the ith node.

You need to place some coins on every node of the tree. The number of coins to be placed at node i can be calculated as:

* If size of the subtree of node i is less than 3, place 1 coin.
* Otherwise, place an amount of coins equal to the **maximum** product of cost values assigned to 3 distinct nodes in the subtree of node i. If this product is **negative**, place 0 coins.

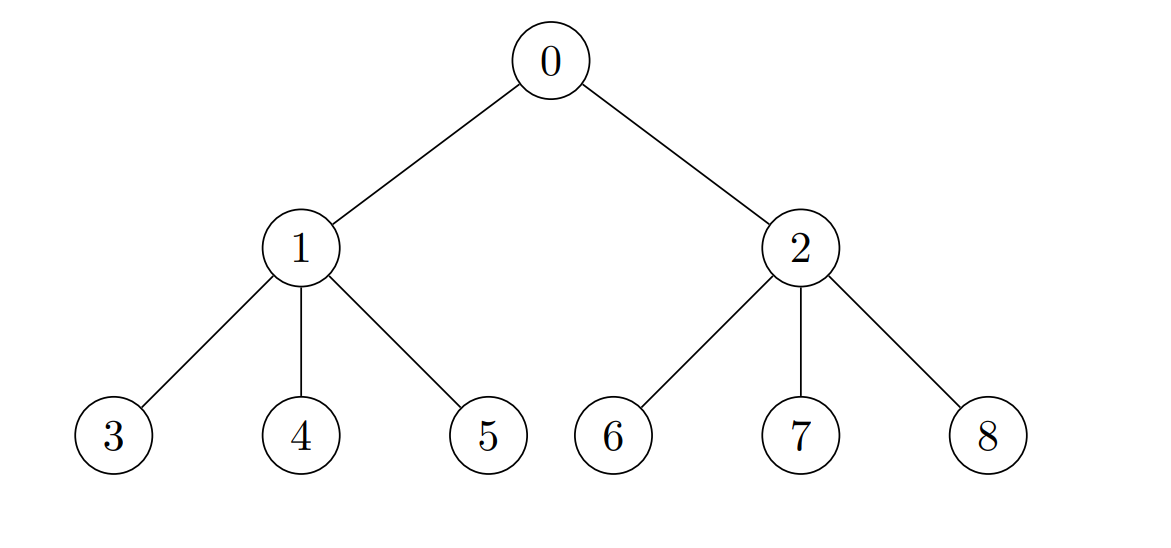
Return *an array* coin *of size* n *such that* coin[i] *is the number of coins placed at node* i*.*

**Example 1:**



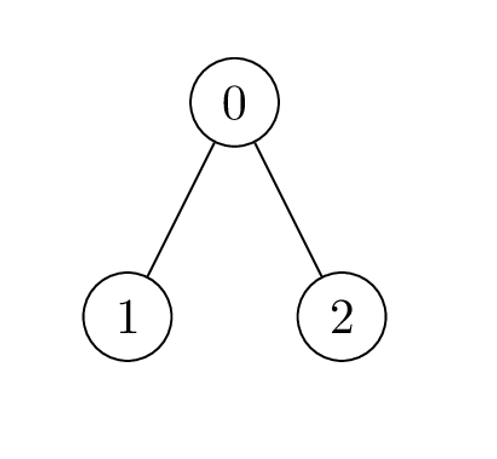
Input: edges = [[0,1],[0,2],[0,3],[0,4],[0,5]], cost = [1,2,3,4,5,6]  
Output: [120,1,1,1,1,1]  
Explanation: For node 0 place 6 \* 5 \* 4 = 120 coins. All other nodes are leaves with subtree of size 1, place 1 coin on each of them.

**Example 2:**



Input: edges = [[0,1],[0,2],[1,3],[1,4],[1,5],[2,6],[2,7],[2,8]], cost = [1,4,2,3,5,7,8,-4,2]  
Output: [280,140,32,1,1,1,1,1,1]  
Explanation: The coins placed on each node are:  
- Place 8 \* 7 \* 5 = 280 coins on node 0.  
- Place 7 \* 5 \* 4 = 140 coins on node 1.  
- Place 8 \* 2 \* 2 = 32 coins on node 2.  
- All other nodes are leaves with subtree of size 1, place 1 coin on each of them.

**Example 3:**



Input: edges = [[0,1],[0,2]], cost = [1,2,-2]  
Output: [0,1,1]  
Explanation: Node 1 and 2 are leaves with subtree of size 1, place 1 coin on each of them. For node 0 the only possible product of cost is 2 \* 1 \* -2 = -4. Hence place 0 coins on node 0.

**Constraints:**

* 2 <= n <= 2 \* 104
* edges.length == n - 1
* edges[i].length == 2
* 0 <= ai, bi < n
* cost.length == n
* 1 <= |cost[i]| <= 104
* The input is generated such that edges represents a valid tree.

# **2667. Create Hello World Function**

https://leetcode.com/problems/create-hello-world-function/description/ Write a function createHelloWorld. It should return a new function that always returns "Hello World".

**Example 1:**

Input: args = []  
Output: "Hello World"  
Explanation:  
const f = createHelloWorld();  
f(); // "Hello World"  
  
The function returned by createHelloWorld should always return "Hello World".

**Example 2:**

Input: args = [{},null,42]  
Output: "Hello World"  
Explanation:  
const f = createHelloWorld();  
f({}, null, 42); // "Hello World"  
  
Any arguments could be passed to the function but it should still always return "Hello World".

**Constraints:**

* 0 <= args.length <= 10

# **1509. Minimum Difference Between Largest and Smallest Value in Three Moves**

https://leetcode.com/problems/minimum-difference-between-largest-and-smallest-value-in-three-moves/description/

You are given an integer array nums.

In one move, you can choose one element of nums and change it to **any value**.

Return *the minimum difference between the largest and smallest value of nums* ***after performing at most three moves***.

**Example 1:**

Input: nums = [5,3,2,4]  
Output: 0  
Explanation: We can make at most 3 moves.  
In the first move, change 2 to 3. nums becomes [5,3,3,4].  
In the second move, change 4 to 3. nums becomes [5,3,3,3].  
In the third move, change 5 to 3. nums becomes [3,3,3,3].  
After performing 3 moves, the difference between the minimum and maximum is 3 - 3 = 0.

**Example 2:**

Input: nums = [1,5,0,10,14]  
Output: 1  
Explanation: We can make at most 3 moves.  
In the first move, change 5 to 0. nums becomes [1,0,0,10,14].  
In the second move, change 10 to 0. nums becomes [1,0,0,0,14].  
In the third move, change 14 to 1. nums becomes [1,0,0,0,1].  
After performing 3 moves, the difference between the minimum and maximum is 1 - 0 = 1.  
It can be shown that there is no way to make the difference 0 in 3 moves.

**Example 3:**

Input: nums = [3,100,20]  
Output: 0  
Explanation: We can make at most 3 moves.  
In the first move, change 100 to 7. nums becomes [3,7,20].  
In the second move, change 20 to 7. nums becomes [3,7,7].  
In the third move, change 3 to 7. nums becomes [7,7,7].  
After performing 3 moves, the difference between the minimum and maximum is 7 - 7 = 0.

**Constraints:**

* 1 <= nums.length <= 105
* -109 <= nums[i] <= 109

# **1834. Single-Threaded CPU**

https://leetcode.com/problems/single-threaded-cpu/description/

You are given n​​​​​​ tasks labeled from 0 to n - 1 represented by a 2D integer array tasks, where tasks[i] = [enqueueTimei, processingTimei] means that the i​​​​​​th​​​​ task will be available to process at enqueueTimei and will take processingTimei to finish processing.

You have a single-threaded CPU that can process **at most one** task at a time and will act in the following way:

* If the CPU is idle and there are no available tasks to process, the CPU remains idle.
* If the CPU is idle and there are available tasks, the CPU will choose the one with the **shortest processing time**. If multiple tasks have the same shortest processing time, it will choose the task with the smallest index.
* Once a task is started, the CPU will **process the entire task** without stopping.
* The CPU can finish a task then start a new one instantly.

Return *the order in which the CPU will process the tasks.*

**Example 1:**

Input: tasks = [[1,2],[2,4],[3,2],[4,1]]  
Output: [0,2,3,1]  
Explanation: The events go as follows:   
- At time = 1, task 0 is available to process. Available tasks = {0}.  
- Also at time = 1, the idle CPU starts processing task 0. Available tasks = {}.  
- At time = 2, task 1 is available to process. Available tasks = {1}.  
- At time = 3, task 2 is available to process. Available tasks = {1, 2}.  
- Also at time = 3, the CPU finishes task 0 and starts processing task 2 as it is the shortest. Available tasks = {1}.  
- At time = 4, task 3 is available to process. Available tasks = {1, 3}.  
- At time = 5, the CPU finishes task 2 and starts processing task 3 as it is the shortest. Available tasks = {1}.  
- At time = 6, the CPU finishes task 3 and starts processing task 1. Available tasks = {}.  
- At time = 10, the CPU finishes task 1 and becomes idle.

**Example 2:**

Input: tasks = [[7,10],[7,12],[7,5],[7,4],[7,2]]  
Output: [4,3,2,0,1]  
Explanation: The events go as follows:  
- At time = 7, all the tasks become available. Available tasks = {0,1,2,3,4}.  
- Also at time = 7, the idle CPU starts processing task 4. Available tasks = {0,1,2,3}.  
- At time = 9, the CPU finishes task 4 and starts processing task 3. Available tasks = {0,1,2}.  
- At time = 13, the CPU finishes task 3 and starts processing task 2. Available tasks = {0,1}.  
- At time = 18, the CPU finishes task 2 and starts processing task 0. Available tasks = {1}.  
- At time = 28, the CPU finishes task 0 and starts processing task 1. Available tasks = {}.  
- At time = 40, the CPU finishes task 1 and becomes idle.

**Constraints:**

* tasks.length == n
* 1 <= n <= 105
* 1 <= enqueueTimei, processingTimei <= 109

# **2713. Maximum Strictly Increasing Cells in a Matrix**

https://leetcode.com/problems/maximum-strictly-increasing-cells-in-a-matrix/description/

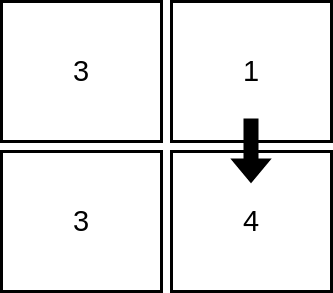
Given a **1-indexed** m x n integer matrix mat, you can select any cell in the matrix as your **starting cell**.

From the starting cell, you can move to any other cell **in the** **same row or column**, but only if the value of the destination cell is **strictly greater** than the value of the current cell. You can repeat this process as many times as possible, moving from cell to cell until you can no longer make any moves.

Your task is to find the **maximum number of cells** that you can visit in the matrix by starting from some cell.

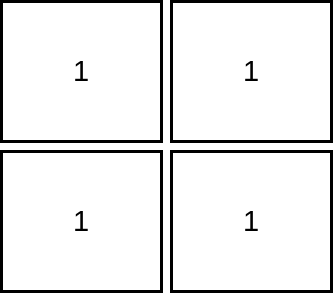
Return *an integer denoting the maximum number of cells that can be visited.*

**Example 1:**



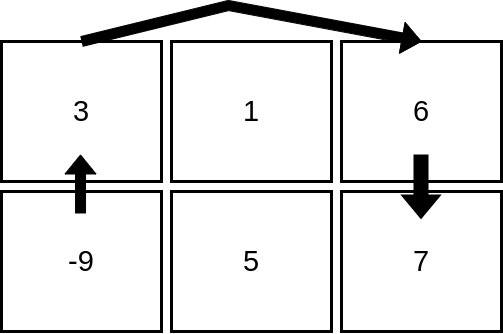
Input: mat = [[3,1],[3,4]]  
Output: 2  
Explanation: The image shows how we can visit 2 cells starting from row 1, column 2. It can be shown that we cannot visit more than 2 cells no matter where we start from, so the answer is 2.

**Example 2:**



Input: mat = [[1,1],[1,1]]  
Output: 1  
Explanation: Since the cells must be strictly increasing, we can only visit one cell in this example.

**Example 3:**



Input: mat = [[3,1,6],[-9,5,7]]  
Output: 4  
Explanation: The image above shows how we can visit 4 cells starting from row 2, column 1. It can be shown that we cannot visit more than 4 cells no matter where we start from, so the answer is 4.

**Constraints:**

* m == mat.length
* n == mat[i].length
* 1 <= m, n <= 105
* 1 <= m \* n <= 105
* -105 <= mat[i][j] <= 105

# **362. Design Hit Counter**

https://leetcode.com/problems/design-hit-counter/description/

Design a hit counter which counts the number of hits received in the past 5 minutes (i.e., the past 300 seconds).

Your system should accept a timestamp parameter (**in seconds** granularity), and you may assume that calls are being made to the system in chronological order (i.e., timestamp is monotonically increasing). Several hits may arrive roughly at the same time.

Implement the HitCounter class:

* HitCounter() Initializes the object of the hit counter system.
* void hit(int timestamp) Records a hit that happened at timestamp (**in seconds**). Several hits may happen at the same timestamp.
* int getHits(int timestamp) Returns the number of hits in the past 5 minutes from timestamp (i.e., the past 300 seconds).

**Example 1:**

Input  
["HitCounter", "hit", "hit", "hit", "getHits", "hit", "getHits", "getHits"]  
[[], [1], [2], [3], [4], [300], [300], [301]]  
Output  
[null, null, null, null, 3, null, 4, 3]  
  
Explanation  
HitCounter hitCounter = new HitCounter();  
hitCounter.hit(1); // hit at timestamp 1.  
hitCounter.hit(2); // hit at timestamp 2.  
hitCounter.hit(3); // hit at timestamp 3.  
hitCounter.getHits(4); // get hits at timestamp 4, return 3.  
hitCounter.hit(300); // hit at timestamp 300.  
hitCounter.getHits(300); // get hits at timestamp 300, return 4.  
hitCounter.getHits(301); // get hits at timestamp 301, return 3.

**Constraints:**

* 1 <= timestamp <= 2 \* 109
* All the calls are being made to the system in chronological order (i.e., timestamp is monotonically increasing).
* At most 300 calls will be made to hit and getHits.

**Follow up:** What if the number of hits per second could be huge? Does your design scale?

# **247. Strobogrammatic Number II**

https://leetcode.com/problems/strobogrammatic-number-ii/description/

Given an integer n, return all the **strobogrammatic numbers** that are of length n. You may return the answer in **any order**.

A **strobogrammatic number** is a number that looks the same when rotated 180 degrees (looked at upside down).

**Example 1:**

Input: n = 2  
Output: ["11","69","88","96"]

**Example 2:**

Input: n = 1  
Output: ["0","1","8"]

**Constraints:**

* 1 <= n <= 14

# **1125. Smallest Sufficient Team**

https://leetcode.com/problems/smallest-sufficient-team/description/

In a project, you have a list of required skills req\_skills, and a list of people. The ith person people[i] contains a list of skills that the person has.

Consider a sufficient team: a set of people such that for every required skill in req\_skills, there is at least one person in the team who has that skill. We can represent these teams by the index of each person.

* For example, team = [0, 1, 3] represents the people with skills people[0], people[1], and people[3].

Return *any sufficient team of the smallest possible size, represented by the index of each person*. You may return the answer in **any order**.

It is **guaranteed** an answer exists.

**Example 1:**

Input: req\_skills = ["java","nodejs","reactjs"], people = [["java"],["nodejs"],["nodejs","reactjs"]]  
Output: [0,2]

**Example 2:**

Input: req\_skills = ["algorithms","math","java","reactjs","csharp","aws"], people = [["algorithms","math","java"],["algorithms","math","reactjs"],["java","csharp","aws"],["reactjs","csharp"],["csharp","math"],["aws","java"]]  
Output: [1,2]

**Constraints:**

* 1 <= req\_skills.length <= 16
* 1 <= req\_skills[i].length <= 16
* req\_skills[i] consists of lowercase English letters.
* All the strings of req\_skills are **unique**.
* 1 <= people.length <= 60
* 0 <= people[i].length <= 16
* 1 <= people[i][j].length <= 16
* people[i][j] consists of lowercase English letters.
* All the strings of people[i] are **unique**.
* Every skill in people[i] is a skill in req\_skills.
* It is guaranteed a sufficient team exists.

# **681. Next Closest Time**

https://leetcode.com/problems/next-closest-time/description/

Given a time represented in the format "HH:MM", form the next closest time by reusing the current digits. There is no limit on how many times a digit can be reused.

You may assume the given input string is always valid. For example, "01:34", "12:09" are all valid. "1:34", "12:9" are all invalid.

**Example 1:**

Input: time = "19:34"  
Output: "19:39"  
Explanation: The next closest time choosing from digits 1, 9, 3, 4, is 19:39, which occurs 5 minutes later.  
It is not 19:33, because this occurs 23 hours and 59 minutes later.

**Example 2:**

Input: time = "23:59"  
Output: "22:22"  
Explanation: The next closest time choosing from digits 2, 3, 5, 9, is 22:22.  
It may be assumed that the returned time is next day's time since it is smaller than the input time numerically.

**Constraints:**

* time.length == 5
* time is a valid time in the form "HH:MM".
* 0 <= HH < 24
* 0 <= MM < 60

# **1601. Maximum Number of Achievable Transfer Requests**

https://leetcode.com/problems/maximum-number-of-achievable-transfer-requests/description/

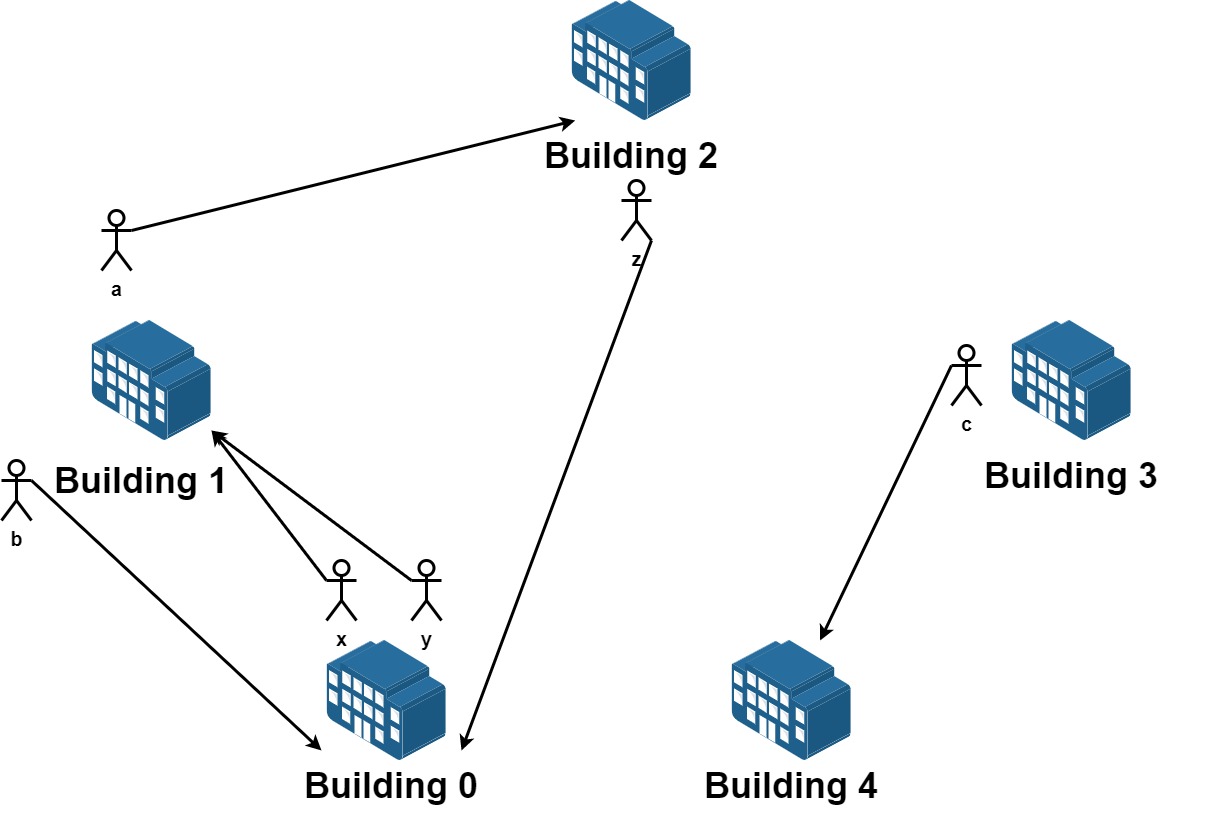
We have n buildings numbered from 0 to n - 1. Each building has a number of employees. It's transfer season, and some employees want to change the building they reside in.

You are given an array requests where requests[i] = [fromi, toi] represents an employee's request to transfer from building fromi to building toi.

**All buildings are full**, so a list of requests is achievable only if for each building, the **net change in employee transfers is zero**. This means the number of employees **leaving** is **equal** to the number of employees **moving in**. For example if n = 3 and two employees are leaving building 0, one is leaving building 1, and one is leaving building 2, there should be two employees moving to building 0, one employee moving to building 1, and one employee moving to building 2.

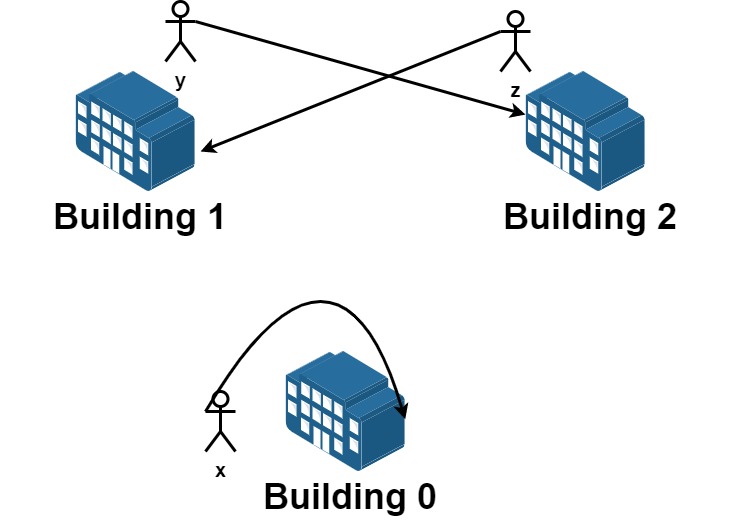
Return *the maximum number of achievable requests*.

**Example 1:**



Input: n = 5, requests = [[0,1],[1,0],[0,1],[1,2],[2,0],[3,4]]  
Output: 5  
Explantion: Let's see the requests:  
From building 0 we have employees x and y and both want to move to building 1.  
From building 1 we have employees a and b and they want to move to buildings 2 and 0 respectively.  
From building 2 we have employee z and they want to move to building 0.  
From building 3 we have employee c and they want to move to building 4.  
From building 4 we don't have any requests.  
We can achieve the requests of users x and b by swapping their places.  
We can achieve the requests of users y, a and z by swapping the places in the 3 buildings.

**Example 2:**



Input: n = 3, requests = [[0,0],[1,2],[2,1]]  
Output: 3  
Explantion: Let's see the requests:  
From building 0 we have employee x and they want to stay in the same building 0.  
From building 1 we have employee y and they want to move to building 2.  
From building 2 we have employee z and they want to move to building 1.  
We can achieve all the requests.

**Example 3:**

Input: n = 4, requests = [[0,3],[3,1],[1,2],[2,0]]  
Output: 4

**Constraints:**

* 1 <= n <= 20
* 1 <= requests.length <= 16
* requests[i].length == 2
* 0 <= fromi, toi < n

# **315. Count of Smaller Numbers After Self**

https://leetcode.com/problems/count-of-smaller-numbers-after-self/description/

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].

**Example 1:**

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.

**Example 2:**

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

**Example 3:**

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

**Constraints:**

* 1 <= nums.length <= 105
* -104 <= nums[i] <= 104

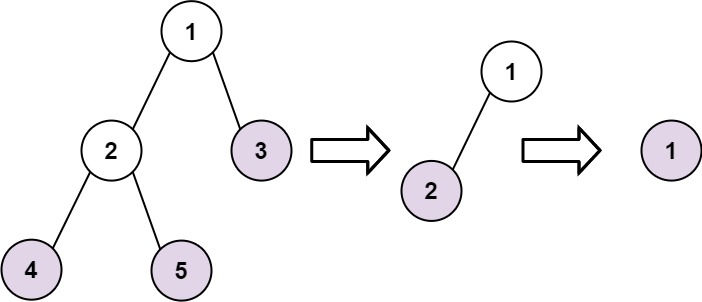
# **366. Find Leaves of Binary Tree**

https://leetcode.com/problems/find-leaves-of-binary-tree/description/

Given the root of a binary tree, collect a tree's nodes as if you were doing this:

* Collect all the leaf nodes.
* Remove all the leaf nodes.
* Repeat until the tree is empty.

**Example 1:**



Input: root = [1,2,3,4,5]  
Output: [[4,5,3],[2],[1]]  
Explanation:  
[[3,5,4],[2],[1]] and [[3,4,5],[2],[1]] are also considered correct answers since per each level it does not matter the order on which elements are returned.

**Example 2:**

Input: root = [1]  
Output: [[1]]

**Constraints:**

* The number of nodes in the tree is in the range [1, 100].
* -100 <= Node.val <= 100

# **424. Longest Repeating Character Replacement**

https://leetcode.com/problems/longest-repeating-character-replacement/description/

You are given a string s and an integer k. You can choose any character of the string and change it to any other uppercase English character. You can perform this operation at most k times.

Return *the length of the longest substring containing the same letter you can get after performing the above operations*.

**Example 1:**

Input: s = "ABAB", k = 2  
Output: 4  
Explanation: Replace the two 'A's with two 'B's or vice versa.

**Example 2:**

Input: s = "AABABBA", k = 1  
Output: 4  
Explanation: Replace the one 'A' in the middle with 'B' and form "AABBBBA".  
The substring "BBBB" has the longest repeating letters, which is 4.  
There may exists other ways to achieve this answer too.

**Constraints:**

* 1 <= s.length <= 105
* s consists of only uppercase English letters.
* 0 <= k <= s.length

# **833. Find And Replace in String**

https://leetcode.com/problems/find-and-replace-in-string/description/

You are given a **0-indexed** string s that you must perform k replacement operations on. The replacement operations are given as three **0-indexed** parallel arrays, indices, sources, and targets, all of length k.

To complete the ith replacement operation:

1. Check if the **substring** sources[i] occurs at index indices[i] in the **original string** s.
2. If it does not occur, **do nothing**.
3. Otherwise if it does occur, **replace** that substring with targets[i].

For example, if s = "abcd", indices[i] = 0, sources[i] = "ab", and targets[i] = "eee", then the result of this replacement will be "eeecd".

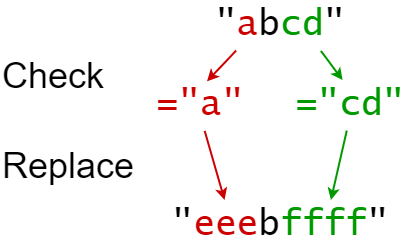
All replacement operations must occur **simultaneously**, meaning the replacement operations should not affect the indexing of each other. The testcases will be generated such that the replacements will **not overlap**.

* For example, a testcase with s = "abc", indices = [0, 1], and sources = ["ab","bc"] will not be generated because the "ab" and "bc" replacements overlap.

Return *the* ***resulting string*** *after performing all replacement operations on* s.

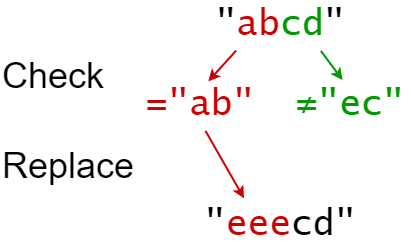
A **substring** is a contiguous sequence of characters in a string.

**Example 1:**



Input: s = "abcd", indices = [0, 2], sources = ["a", "cd"], targets = ["eee", "ffff"]  
Output: "eeebffff"  
Explanation:  
"a" occurs at index 0 in s, so we replace it with "eee".  
"cd" occurs at index 2 in s, so we replace it with "ffff".

**Example 2:**



Input: s = "abcd", indices = [0, 2], sources = ["ab","ec"], targets = ["eee","ffff"]  
Output: "eeecd"  
Explanation:  
"ab" occurs at index 0 in s, so we replace it with "eee".  
"ec" does not occur at index 2 in s, so we do nothing.

**Constraints:**

* 1 <= s.length <= 1000
* k == indices.length == sources.length == targets.length
* 1 <= k <= 100
* 0 <= indexes[i] < s.length
* 1 <= sources[i].length, targets[i].length <= 50
* s consists of only lowercase English letters.
* sources[i] and targets[i] consist of only lowercase English letters.

# **1. Two Sum**

https://leetcode.com/problems/two-sum/description/

Given an array of integers nums and an integer target, return *indices of the two numbers such that they add up to target*.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

**Example 1:**

Input: nums = [2,7,11,15], target = 9  
Output: [0,1]  
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

**Example 2:**

Input: nums = [3,2,4], target = 6  
Output: [1,2]

**Example 3:**

Input: nums = [3,3], target = 6  
Output: [0,1]

**Constraints:**

* 2 <= nums.length <= 104
* -109 <= nums[i] <= 109
* -109 <= target <= 109
* **Only one valid answer exists.**

**Follow-up:**Can you come up with an algorithm that is less than O(n2) time complexity?

# **1740. Find Distance in a Binary Tree**

https://leetcode.com/problems/find-distance-in-a-binary-tree/description/

Given the root of a binary tree and two integers p and q, return *the* ***distance*** *between the nodes of value* p *and value* q *in the tree*.

The **distance** between two nodes is the number of edges on the path from one to the other.

**Example 1:**



Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 0  
Output: 3  
Explanation: There are 3 edges between 5 and 0: 5-3-1-0.

**Example 2:**



Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 7  
Output: 2  
Explanation: There are 2 edges between 5 and 7: 5-2-7.

**Example 3:**



Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 5  
Output: 0  
Explanation: The distance between a node and itself is 0.

**Constraints:**

* The number of nodes in the tree is in the range [1, 104].
* 0 <= Node.val <= 109
* All Node.val are **unique**.
* p and q are values in the tree.

# **365. Water and Jug Problem**

https://leetcode.com/problems/water-and-jug-problem/description/

You are given two jugs with capacities x liters and y liters. You have an infinite water supply. Return whether the total amount of water in both jugs may reach target using the following operations:

* Fill either jug completely with water.
* Completely empty either jug.
* Pour water from one jug into another until the receiving jug is full, or the transferring jug is empty.

**Example 1:**

**Input:** x = 3, y = 5, target = 4

**Output:** true

**Explanation:**

Follow these steps to reach a total of 4 liters:

1. Fill the 5-liter jug (0, 5).
2. Pour from the 5-liter jug into the 3-liter jug, leaving 2 liters (3, 2).
3. Empty the 3-liter jug (0, 2).
4. Transfer the 2 liters from the 5-liter jug to the 3-liter jug (2, 0).
5. Fill the 5-liter jug again (2, 5).
6. Pour from the 5-liter jug into the 3-liter jug until the 3-liter jug is full. This leaves 4 liters in the 5-liter jug (3, 4).
7. Empty the 3-liter jug. Now, you have exactly 4 liters in the 5-liter jug (0, 4).

Reference: The [Die Hard](https://www.youtube.com/watch?v=BVtQNK_ZUJg&ab_channel=notnek01) example.

**Example 2:**

**Input:** x = 2, y = 6, target = 5

**Output:** false

**Example 3:**

**Input:** x = 1, y = 2, target = 3

**Output:** true

**Explanation:** Fill both jugs. The total amount of water in both jugs is equal to 3 now.

**Constraints:**

* 1 <= x, y, target <= 103

# **818. Race Car**

https://leetcode.com/problems/race-car/description/

Your car starts at position 0 and speed +1 on an infinite number line. Your car can go into negative positions. Your car drives automatically according to a sequence of instructions 'A' (accelerate) and 'R' (reverse):

* When you get an instruction 'A', your car does the following:
  + position += speed
  + speed \*= 2
* When you get an instruction 'R', your car does the following:
  + If your speed is positive then speed = -1
  + otherwise speed = 1
* Your position stays the same.

For example, after commands "AAR", your car goes to positions 0 --> 1 --> 3 --> 3, and your speed goes to 1 --> 2 --> 4 --> -1.

Given a target position target, return *the length of the shortest sequence of instructions to get there*.

**Example 1:**

Input: target = 3  
Output: 2  
Explanation:   
The shortest instruction sequence is "AA".  
Your position goes from 0 --> 1 --> 3.

**Example 2:**

Input: target = 6  
Output: 5  
Explanation:   
The shortest instruction sequence is "AAARA".  
Your position goes from 0 --> 1 --> 3 --> 7 --> 7 --> 6.

**Constraints:**

* 1 <= target <= 104

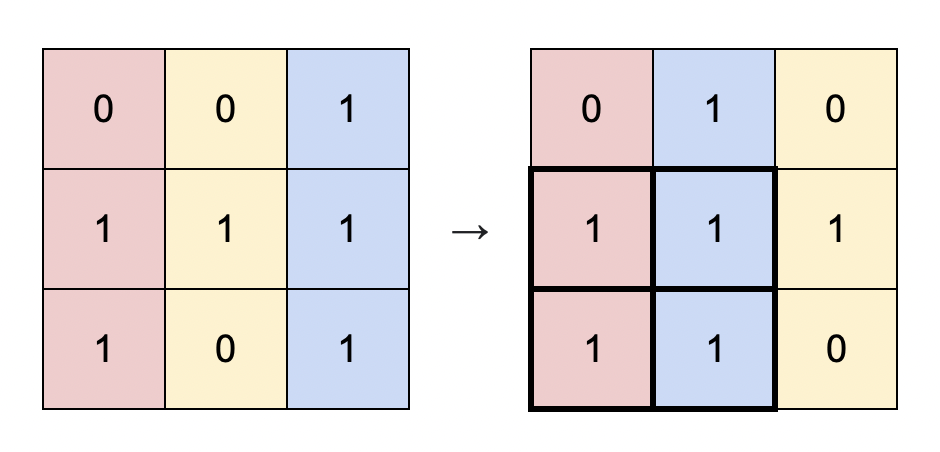
# **1727. Largest Submatrix With Rearrangements**

https://leetcode.com/problems/largest-submatrix-with-rearrangements/description/

You are given a binary matrix matrix of size m x n, and you are allowed to rearrange the **columns** of the matrix in any order.

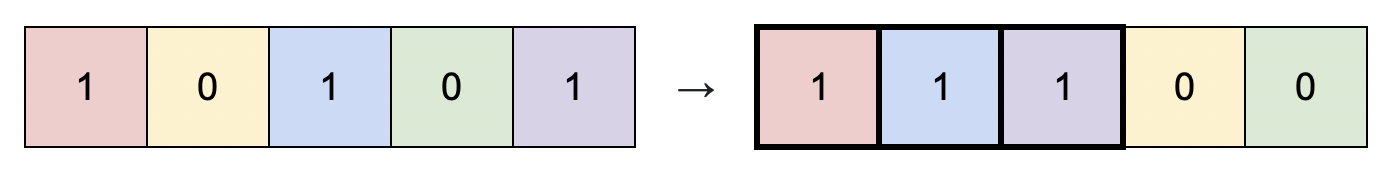
Return *the area of the largest submatrix within* matrix *where* ***every*** *element of the submatrix is* 1 *after reordering the columns optimally.*

**Example 1:**



Input: matrix = [[0,0,1],[1,1,1],[1,0,1]]  
Output: 4  
Explanation: You can rearrange the columns as shown above.  
The largest submatrix of 1s, in bold, has an area of 4.

**Example 2:**



Input: matrix = [[1,0,1,0,1]]  
Output: 3  
Explanation: You can rearrange the columns as shown above.  
The largest submatrix of 1s, in bold, has an area of 3.

**Example 3:**

Input: matrix = [[1,1,0],[1,0,1]]  
Output: 2  
Explanation: Notice that you must rearrange entire columns, and there is no way to make a submatrix of 1s larger than an area of 2.

**Constraints:**

* m == matrix.length
* n == matrix[i].length
* 1 <= m \* n <= 105
* matrix[i][j] is either 0 or 1.

# **564. Find the Closest Palindrome**

https://leetcode.com/problems/find-the-closest-palindrome/description/

Given a string n representing an integer, return *the closest integer (not including itself), which is a palindrome*. If there is a tie, return ***the smaller one***.

The closest is defined as the absolute difference minimized between two integers.

**Example 1:**

Input: n = "123"  
Output: "121"

**Example 2:**

Input: n = "1"  
Output: "0"  
Explanation: 0 and 2 are the closest palindromes but we return the smallest which is 0.

**Constraints:**

* 1 <= n.length <= 18
* n consists of only digits.
* n does not have leading zeros.
* n is representing an integer in the range [1, 1018 - 1].

# **146. LRU Cache**

https://leetcode.com/problems/lru-cache/description/

Design a data structure that follows the constraints of a [**Least Recently Used (LRU) cache**](https://en.wikipedia.org/wiki/Cache_replacement_policies#LRU).

Implement the LRUCache class:

* LRUCache(int capacity) Initialize the LRU cache with **positive** size capacity.
* int get(int key) Return the value of the key if the key exists, otherwise return -1.
* void put(int key, int value) Update the value of the key if the key exists. Otherwise, add the key-value pair to the cache. If the number of keys exceeds the capacity from this operation, **evict** the least recently used key.

The functions get and put must each run in O(1) average time complexity.

**Example 1:**

Input  
["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]  
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]  
Output  
[null, null, null, 1, null, -1, null, -1, 3, 4]  
  
Explanation  
LRUCache lRUCache = new LRUCache(2);  
lRUCache.put(1, 1); // cache is {1=1}  
lRUCache.put(2, 2); // cache is {1=1, 2=2}  
lRUCache.get(1); // return 1  
lRUCache.put(3, 3); // LRU key was 2, evicts key 2, cache is {1=1, 3=3}  
lRUCache.get(2); // returns -1 (not found)  
lRUCache.put(4, 4); // LRU key was 1, evicts key 1, cache is {4=4, 3=3}  
lRUCache.get(1); // return -1 (not found)  
lRUCache.get(3); // return 3  
lRUCache.get(4); // return 4

**Constraints:**

* 1 <= capacity <= 3000
* 0 <= key <= 104
* 0 <= value <= 105
* At most 2 \* 105 calls will be made to get and put.

# **1425. Constrained Subsequence Sum**

https://leetcode.com/problems/constrained-subsequence-sum/description/

Given an integer array nums and an integer k, return the maximum sum of a **non-empty** subsequence of that array such that for every two **consecutive** integers in the subsequence, nums[i] and nums[j], where i < j, the condition j - i <= k is satisfied.

A *subsequence* of an array is obtained by deleting some number of elements (can be zero) from the array, leaving the remaining elements in their original order.

**Example 1:**

Input: nums = [10,2,-10,5,20], k = 2  
Output: 37  
Explanation: The subsequence is [10, 2, 5, 20].

**Example 2:**

Input: nums = [-1,-2,-3], k = 1  
Output: -1  
Explanation: The subsequence must be non-empty, so we choose the largest number.

**Example 3:**

Input: nums = [10,-2,-10,-5,20], k = 2  
Output: 23  
Explanation: The subsequence is [10, -2, -5, 20].

**Constraints:**

* 1 <= k <= nums.length <= 105
* -104 <= nums[i] <= 104

# **1730. Shortest Path to Get Food**

https://leetcode.com/problems/shortest-path-to-get-food/description/

You are starving and you want to eat food as quickly as possible. You want to find the shortest path to arrive at any food cell.

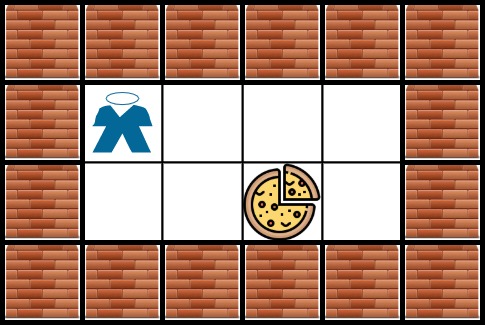
You are given an m x n character matrix, grid, of these different types of cells:

* '\*' is your location. There is **exactly one** '\*' cell.
* '#' is a food cell. There may be **multiple** food cells.
* 'O' is free space, and you can travel through these cells.
* 'X' is an obstacle, and you cannot travel through these cells.

You can travel to any adjacent cell north, east, south, or west of your current location if there is not an obstacle.

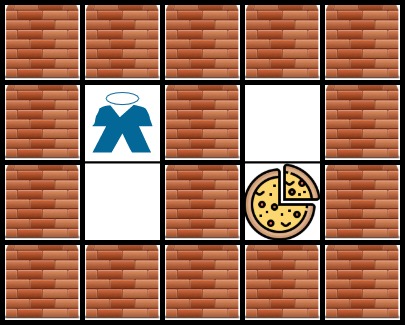
Return *the* ***length*** *of the shortest path for you to reach* ***any*** *food cell*. If there is no path for you to reach food, return -1.

**Example 1:**



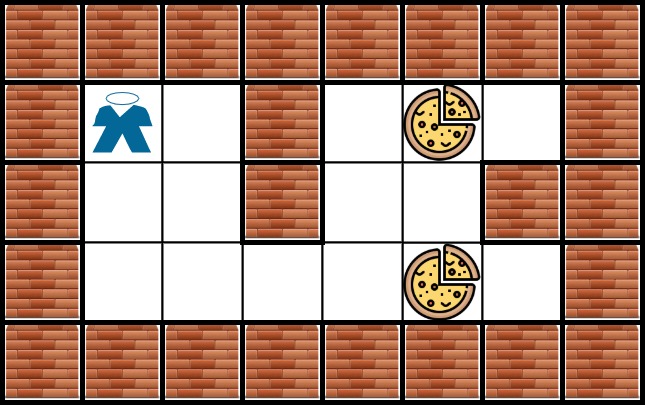
Input: grid = [["X","X","X","X","X","X"],["X","\*","O","O","O","X"],["X","O","O","#","O","X"],["X","X","X","X","X","X"]]  
Output: 3  
Explanation: It takes 3 steps to reach the food.

**Example 2:**



Input: grid = [["X","X","X","X","X"],["X","\*","X","O","X"],["X","O","X","#","X"],["X","X","X","X","X"]]  
Output: -1  
Explanation: It is not possible to reach the food.

**Example 3:**



Input: grid = [["X","X","X","X","X","X","X","X"],["X","\*","O","X","O","#","O","X"],["X","O","O","X","O","O","X","X"],["X","O","O","O","O","#","O","X"],["X","X","X","X","X","X","X","X"]]  
Output: 6  
Explanation: There can be multiple food cells. It only takes 6 steps to reach the bottom food.

**Constraints:**

* m == grid.length
* n == grid[i].length
* 1 <= m, n <= 200
* grid[row][col] is '\*', 'X', 'O', or '#'.
* The grid contains **exactly one** '\*'.

# **655. Print Binary Tree**

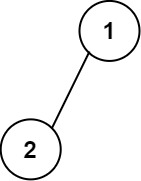
https://leetcode.com/problems/print-binary-tree/description/

Given the root of a binary tree, construct a **0-indexed** m x n string matrix res that represents a **formatted layout** of the tree. The formatted layout matrix should be constructed using the following rules:

* The **height** of the tree is height and the number of rows m should be equal to height + 1.
* The number of columns n should be equal to 2height+1 - 1.
* Place the **root node** in the **middle** of the **top row** (more formally, at location res[0][(n-1)/2]).
* For each node that has been placed in the matrix at position res[r][c], place its **left child** at res[r+1][c-2height-r-1] and its **right child** at res[r+1][c+2height-r-1].
* Continue this process until all the nodes in the tree have been placed.
* Any empty cells should contain the empty string "".

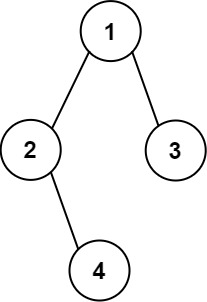
Return *the constructed matrix* res.

**Example 1:**



Input: root = [1,2]  
Output:   
[["","1",""],  
 ["2","",""]]

**Example 2:**



Input: root = [1,2,3,null,4]  
Output:   
[["","","","1","","",""],  
 ["","2","","","","3",""],  
 ["","","4","","","",""]]

**Constraints:**

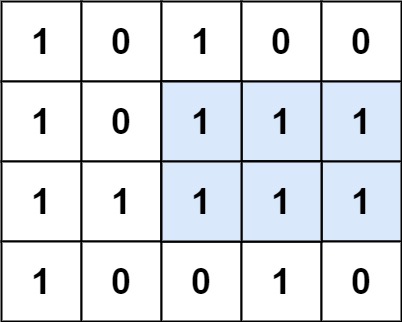
* The number of nodes in the tree is in the range [1, 210].
* -99 <= Node.val <= 99
* The depth of the tree will be in the range [1, 10].

# **85. Maximal Rectangle**

https://leetcode.com/problems/maximal-rectangle/description/

Given a rows x cols binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return *its area*.

**Example 1:**



Input: matrix = [["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]  
Output: 6  
Explanation: The maximal rectangle is shown in the above picture.

**Example 2:**

Input: matrix = [["0"]]  
Output: 0

**Example 3:**

Input: matrix = [["1"]]  
Output: 1

**Constraints:**

* rows == matrix.length
* cols == matrix[i].length
* 1 <= row, cols <= 200
* matrix[i][j] is '0' or '1'.

# **642. Design Search Autocomplete System**

https://leetcode.com/problems/design-search-autocomplete-system/description/

Design a search autocomplete system for a search engine. Users may input a sentence (at least one word and end with a special character '#').

You are given a string array sentences and an integer array times both of length n where sentences[i] is a previously typed sentence and times[i] is the corresponding number of times the sentence was typed. For each input character except '#', return the top 3 historical hot sentences that have the same prefix as the part of the sentence already typed.

Here are the specific rules:

* The hot degree for a sentence is defined as the number of times a user typed the exactly same sentence before.
* The returned top 3 hot sentences should be sorted by hot degree (The first is the hottest one). If several sentences have the same hot degree, use ASCII-code order (smaller one appears first).
* If less than 3 hot sentences exist, return as many as you can.
* When the input is a special character, it means the sentence ends, and in this case, you need to return an empty list.

Implement the AutocompleteSystem class:

* AutocompleteSystem(String[] sentences, int[] times) Initializes the object with the sentences and times arrays.
* List<String> input(char c) This indicates that the user typed the character c.
  + Returns an empty array [] if c == '#' and stores the inputted sentence in the system.
  + Returns the top 3 historical hot sentences that have the same prefix as the part of the sentence already typed. If there are fewer than 3 matches, return them all.

**Example 1:**

Input  
["AutocompleteSystem", "input", "input", "input", "input"]  
[[["i love you", "island", "iroman", "i love leetcode"], [5, 3, 2, 2]], ["i"], [" "], ["a"], ["#"]]  
Output  
[null, ["i love you", "island", "i love leetcode"], ["i love you", "i love leetcode"], [], []]  
  
Explanation  
AutocompleteSystem obj = new AutocompleteSystem(["i love you", "island", "iroman", "i love leetcode"], [5, 3, 2, 2]);  
obj.input("i"); // return ["i love you", "island", "i love leetcode"]. There are four sentences that have prefix "i". Among them, "ironman" and "i love leetcode" have same hot degree. Since ' ' has ASCII code 32 and 'r' has ASCII code 114, "i love leetcode" should be in front of "ironman". Also we only need to output top 3 hot sentences, so "ironman" will be ignored.  
obj.input(" "); // return ["i love you", "i love leetcode"]. There are only two sentences that have prefix "i ".  
obj.input("a"); // return []. There are no sentences that have prefix "i a".  
obj.input("#"); // return []. The user finished the input, the sentence "i a" should be saved as a historical sentence in system. And the following input will be counted as a new search.

**Constraints:**

* n == sentences.length
* n == times.length
* 1 <= n <= 100
* 1 <= sentences[i].length <= 100
* 1 <= times[i] <= 50
* c is a lowercase English letter, a hash '#', or space ' '.
* Each tested sentence will be a sequence of characters c that end with the character '#'.
* Each tested sentence will have a length in the range [1, 200].
* The words in each input sentence are separated by single spaces.
* At most 5000 calls will be made to input.

# **759. Employee Free Time**

https://leetcode.com/problems/employee-free-time/description/

We are given a list schedule of employees, which represents the working time for each employee.

Each employee has a list of non-overlapping Intervals, and these intervals are in sorted order.

Return the list of finite intervals representing **common, positive-length free time** for *all* employees, also in sorted order.

(Even though we are representing Intervals in the form [x, y], the objects inside are Intervals, not lists or arrays. For example, schedule[0][0].start = 1, schedule[0][0].end = 2, and schedule[0][0][0] is not defined).  Also, we wouldn't include intervals like [5, 5] in our answer, as they have zero length.

**Example 1:**

Input: schedule = [[[1,2],[5,6]],[[1,3]],[[4,10]]]  
Output: [[3,4]]  
Explanation: There are a total of three employees, and all common  
free time intervals would be [-inf, 1], [3, 4], [10, inf].  
We discard any intervals that contain inf as they aren't finite.

**Example 2:**

Input: schedule = [[[1,3],[6,7]],[[2,4]],[[2,5],[9,12]]]  
Output: [[5,6],[7,9]]

**Constraints:**

* 1 <= schedule.length , schedule[i].length <= 50
* 0 <= schedule[i].start < schedule[i].end <= 10^8

# **1062. Longest Repeating Substring**

https://leetcode.com/problems/longest-repeating-substring/description/

Given a string s, return *the length of the longest repeating substrings*. If no repeating substring exists, return 0.

**Example 1:**

Input: s = "abcd"  
Output: 0  
Explanation: There is no repeating substring.

**Example 2:**

Input: s = "abbaba"  
Output: 2  
Explanation: The longest repeating substrings are "ab" and "ba", each of which occurs twice.

**Example 3:**

Input: s = "aabcaabdaab"  
Output: 3  
Explanation: The longest repeating substring is "aab", which occurs 3 times.

**Constraints:**

* 1 <= s.length <= 2000
* s consists of lowercase English letters.

# **2982. Find Longest Special Substring That Occurs Thrice II**

https://leetcode.com/problems/find-longest-special-substring-that-occurs-thrice-ii/description/

You are given a string s that consists of lowercase English letters.

A string is called **special** if it is made up of only a single character. For example, the string "abc" is not special, whereas the strings "ddd", "zz", and "f" are special.

Return *the length of the* ***longest special substring*** *of* s *which occurs* ***at least thrice***, *or* -1 *if no special substring occurs at least thrice*.

A **substring** is a contiguous **non-empty** sequence of characters within a string.

**Example 1:**

Input: s = "aaaa"  
Output: 2  
Explanation: The longest special substring which occurs thrice is "aa": substrings "aaaa", "aaaa", and "aaaa".  
It can be shown that the maximum length achievable is 2.

**Example 2:**

Input: s = "abcdef"  
Output: -1  
Explanation: There exists no special substring which occurs at least thrice. Hence return -1.

**Example 3:**

Input: s = "abcaba"  
Output: 1  
Explanation: The longest special substring which occurs thrice is "a": substrings "abcaba", "abcaba", and "abcaba".  
It can be shown that the maximum length achievable is 1.

**Constraints:**

* 3 <= s.length <= 5 \* 105
* s consists of only lowercase English letters.

# **1793. Maximum Score of a Good Subarray**

https://leetcode.com/problems/maximum-score-of-a-good-subarray/description/

You are given an array of integers nums **(0-indexed)** and an integer k.

The **score** of a subarray (i, j) is defined as min(nums[i], nums[i+1], ..., nums[j]) \* (j - i + 1). A **good** subarray is a subarray where i <= k <= j.

Return *the maximum possible* ***score*** *of a* ***good*** *subarray.*

**Example 1:**

Input: nums = [1,4,3,7,4,5], k = 3  
Output: 15  
Explanation: The optimal subarray is (1, 5) with a score of min(4,3,7,4,5) \* (5-1+1) = 3 \* 5 = 15.

**Example 2:**

Input: nums = [5,5,4,5,4,1,1,1], k = 0  
Output: 20  
Explanation: The optimal subarray is (0, 4) with a score of min(5,5,4,5,4) \* (4-0+1) = 4 \* 5 = 20.

**Constraints:**

* 1 <= nums.length <= 105
* 1 <= nums[i] <= 2 \* 104
* 0 <= k < nums.length

# **2131. Longest Palindrome by Concatenating Two Letter Words**

https://leetcode.com/problems/longest-palindrome-by-concatenating-two-letter-words/description/

You are given an array of strings words. Each element of words consists of **two** lowercase English letters.

Create the **longest possible palindrome** by selecting some elements from words and concatenating them in **any order**. Each element can be selected **at most once**.

Return *the* ***length*** *of the longest palindrome that you can create*. If it is impossible to create any palindrome, return 0.

A **palindrome** is a string that reads the same forward and backward.

**Example 1:**

Input: words = ["lc","cl","gg"]  
Output: 6  
Explanation: One longest palindrome is "lc" + "gg" + "cl" = "lcggcl", of length 6.  
Note that "clgglc" is another longest palindrome that can be created.

**Example 2:**

Input: words = ["ab","ty","yt","lc","cl","ab"]  
Output: 8  
Explanation: One longest palindrome is "ty" + "lc" + "cl" + "yt" = "tylcclyt", of length 8.  
Note that "lcyttycl" is another longest palindrome that can be created.

**Example 3:**

Input: words = ["cc","ll","xx"]  
Output: 2  
Explanation: One longest palindrome is "cc", of length 2.  
Note that "ll" is another longest palindrome that can be created, and so is "xx".

**Constraints:**

* 1 <= words.length <= 105
* words[i].length == 2
* words[i] consists of lowercase English letters.

# **1146. Snapshot Array**

https://leetcode.com/problems/snapshot-array/description/

Implement a SnapshotArray that supports the following interface:

* SnapshotArray(int length) initializes an array-like data structure with the given length. **Initially, each element equals 0**.
* void set(index, val) sets the element at the given index to be equal to val.
* int snap() takes a snapshot of the array and returns the snap\_id: the total number of times we called snap() minus 1.
* int get(index, snap\_id) returns the value at the given index, at the time we took the snapshot with the given snap\_id

**Example 1:**

Input: ["SnapshotArray","set","snap","set","get"]  
[[3],[0,5],[],[0,6],[0,0]]  
Output: [null,null,0,null,5]  
Explanation:   
SnapshotArray snapshotArr = new SnapshotArray(3); // set the length to be 3  
snapshotArr.set(0,5); // Set array[0] = 5  
snapshotArr.snap(); // Take a snapshot, return snap\_id = 0  
snapshotArr.set(0,6);  
snapshotArr.get(0,0); // Get the value of array[0] with snap\_id = 0, return 5

**Constraints:**

* 1 <= length <= 5 \* 104
* 0 <= index < length
* 0 <= val <= 109
* 0 <= snap\_id < (the total number of times we call snap())
* At most 5 \* 104 calls will be made to set, snap, and get.

# **1930. Unique Length-3 Palindromic Subsequences**

https://leetcode.com/problems/unique-length-3-palindromic-subsequences/description/

Given a string s, return *the number of* ***unique palindromes of length three*** *that are a* ***subsequence*** *of* s.

Note that even if there are multiple ways to obtain the same subsequence, it is still only counted **once**.

A **palindrome** is a string that reads the same forwards and backwards.

A **subsequence** of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

* For example, "ace" is a subsequence of "abcde".

**Example 1:**

Input: s = "aabca"  
Output: 3  
Explanation: The 3 palindromic subsequences of length 3 are:  
- "aba" (subsequence of "aabca")  
- "aaa" (subsequence of "aabca")  
- "aca" (subsequence of "aabca")

**Example 2:**

Input: s = "adc"  
Output: 0  
Explanation: There are no palindromic subsequences of length 3 in "adc".

**Example 3:**

Input: s = "bbcbaba"  
Output: 4  
Explanation: The 4 palindromic subsequences of length 3 are:  
- "bbb" (subsequence of "bbcbaba")  
- "bcb" (subsequence of "bbcbaba")  
- "bab" (subsequence of "bbcbaba")  
- "aba" (subsequence of "bbcbaba")

**Constraints:**

* 3 <= s.length <= 105
* s consists of only lowercase English letters.

# **721. Accounts Merge**

https://leetcode.com/problems/accounts-merge/description/

Given a list of accounts where each element accounts[i] is a list of strings, where the first element accounts[i][0] is a name, and the rest of the elements are **emails** representing emails of the account.

Now, we would like to merge these accounts. Two accounts definitely belong to the same person if there is some common email to both accounts. Note that even if two accounts have the same name, they may belong to different people as people could have the same name. A person can have any number of accounts initially, but all of their accounts definitely have the same name.

After merging the accounts, return the accounts in the following format: the first element of each account is the name, and the rest of the elements are emails **in sorted order**. The accounts themselves can be returned in **any order**.

**Example 1:**

Input: accounts = [["John","johnsmith@mail.com","john\_newyork@mail.com"],["John","johnsmith@mail.com","john00@mail.com"],["Mary","mary@mail.com"],["John","johnnybravo@mail.com"]]  
Output: [["John","john00@mail.com","john\_newyork@mail.com","johnsmith@mail.com"],["Mary","mary@mail.com"],["John","johnnybravo@mail.com"]]  
Explanation:  
The first and second John's are the same person as they have the common email "johnsmith@mail.com".  
The third John and Mary are different people as none of their email addresses are used by other accounts.  
We could return these lists in any order, for example the answer [['Mary', 'mary@mail.com'], ['John', 'johnnybravo@mail.com'],   
['John', 'john00@mail.com', 'john\_newyork@mail.com', 'johnsmith@mail.com']] would still be accepted.

**Example 2:**

Input: accounts = [["Gabe","Gabe0@m.co","Gabe3@m.co","Gabe1@m.co"],["Kevin","Kevin3@m.co","Kevin5@m.co","Kevin0@m.co"],["Ethan","Ethan5@m.co","Ethan4@m.co","Ethan0@m.co"],["Hanzo","Hanzo3@m.co","Hanzo1@m.co","Hanzo0@m.co"],["Fern","Fern5@m.co","Fern1@m.co","Fern0@m.co"]]  
Output: [["Ethan","Ethan0@m.co","Ethan4@m.co","Ethan5@m.co"],["Gabe","Gabe0@m.co","Gabe1@m.co","Gabe3@m.co"],["Hanzo","Hanzo0@m.co","Hanzo1@m.co","Hanzo3@m.co"],["Kevin","Kevin0@m.co","Kevin3@m.co","Kevin5@m.co"],["Fern","Fern0@m.co","Fern1@m.co","Fern5@m.co"]]

**Constraints:**

* 1 <= accounts.length <= 1000
* 2 <= accounts[i].length <= 10
* 1 <= accounts[i][j].length <= 30
* accounts[i][0] consists of English letters.
* accounts[i][j] (for j > 0) is a valid email.

# **834. Sum of Distances in Tree**

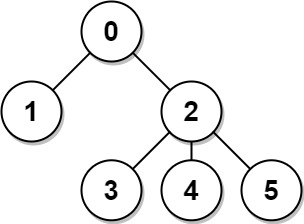
https://leetcode.com/problems/sum-of-distances-in-tree/description/

There is an undirected connected tree with n nodes labeled from 0 to n - 1 and n - 1 edges.

You are given the integer n and the array edges where edges[i] = [ai, bi] indicates that there is an edge between nodes ai and bi in the tree.

Return an array answer of length n where answer[i] is the sum of the distances between the ith node in the tree and all other nodes.

**Example 1:**



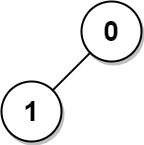
Input: n = 6, edges = [[0,1],[0,2],[2,3],[2,4],[2,5]]  
Output: [8,12,6,10,10,10]  
Explanation: The tree is shown above.  
We can see that dist(0,1) + dist(0,2) + dist(0,3) + dist(0,4) + dist(0,5)  
equals 1 + 1 + 2 + 2 + 2 = 8.  
Hence, answer[0] = 8, and so on.

**Example 2:**



Input: n = 1, edges = []  
Output: [0]

**Example 3:**



Input: n = 2, edges = [[1,0]]  
Output: [1,1]

**Constraints:**

* 1 <= n <= 3 \* 104
* edges.length == n - 1
* edges[i].length == 2
* 0 <= ai, bi < n
* ai != bi
* The given input represents a valid tree.

# **406. Queue Reconstruction by Height**

https://leetcode.com/problems/queue-reconstruction-by-height/description/

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] = [hi, ki] represents the ith person of height hi with **exactly** ki other people in front who have a height greater than or equal to hi.

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] = [hj, kj] is the attributes of the jth person in the queue (queue[0] is the person at the front of the queue).

**Example 1:**

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.

**Example 2:**

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]]

**Constraints:**

* 1 <= people.length <= 2000
* 0 <= hi <= 106
* 0 <= ki < people.length
* It is guaranteed that the queue can be reconstructed.

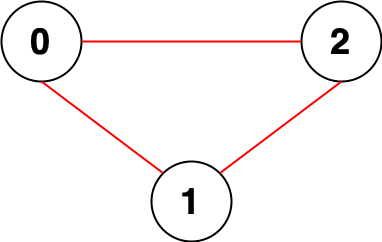
# **2316. Count Unreachable Pairs of Nodes in an Undirected Graph**

https://leetcode.com/problems/count-unreachable-pairs-of-nodes-in-an-undirected-graph/description/

You are given an integer n. There is an **undirected** graph with n nodes, numbered from 0 to n - 1. You are given a 2D integer array edges where edges[i] = [ai, bi] denotes that there exists an **undirected** edge connecting nodes ai and bi.

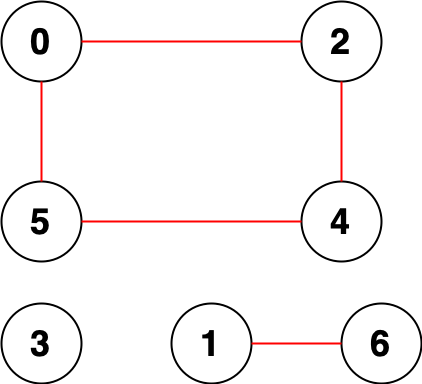
Return *the* ***number of pairs*** *of different nodes that are* ***unreachable*** *from each other*.

**Example 1:**



Input: n = 3, edges = [[0,1],[0,2],[1,2]]  
Output: 0  
Explanation: There are no pairs of nodes that are unreachable from each other. Therefore, we return 0.

**Example 2:**



Input: n = 7, edges = [[0,2],[0,5],[2,4],[1,6],[5,4]]  
Output: 14  
Explanation: There are 14 pairs of nodes that are unreachable from each other:  
[[0,1],[0,3],[0,6],[1,2],[1,3],[1,4],[1,5],[2,3],[2,6],[3,4],[3,5],[3,6],[4,6],[5,6]].  
Therefore, we return 14.

**Constraints:**

* 1 <= n <= 105
* 0 <= edges.length <= 2 \* 105
* edges[i].length == 2
* 0 <= ai, bi < n
* ai != bi
* There are no repeated edges.

# **68. Text Justification**

https://leetcode.com/problems/text-justification/description/

Given an array of strings words and a width maxWidth, format the text such that each line has exactly maxWidth characters and is fully (left and right) justified.

You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces ' ' when necessary so that each line has exactly maxWidth characters.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line does not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

For the last line of text, it should be left-justified, and no extra space is inserted between words.

**Note:**

* A word is defined as a character sequence consisting of non-space characters only.
* Each word's length is guaranteed to be greater than 0 and not exceed maxWidth.
* The input array words contains at least one word.

**Example 1:**

Input: words = ["This", "is", "an", "example", "of", "text", "justification."], maxWidth = 16  
Output:  
[  
   "This    is    an",  
   "example  of text",  
   "justification.  "  
]

**Example 2:**

Input: words = ["What","must","be","acknowledgment","shall","be"], maxWidth = 16  
Output:  
[  
  "What   must   be",  
  "acknowledgment  ",  
  "shall be        "  
]  
Explanation: Note that the last line is "shall be " instead of "shall be", because the last line must be left-justified instead of fully-justified.  
Note that the second line is also left-justified because it contains only one word.

**Example 3:**

Input: words = ["Science","is","what","we","understand","well","enough","to","explain","to","a","computer.","Art","is","everything","else","we","do"], maxWidth = 20  
Output:  
[  
  "Science  is  what we",  
 "understand      well",  
  "enough to explain to",  
  "a  computer.  Art is",  
  "everything  else  we",  
  "do                  "  
]

**Constraints:**

* 1 <= words.length <= 300
* 1 <= words[i].length <= 20
* words[i] consists of only English letters and symbols.
* 1 <= maxWidth <= 100
* words[i].length <= maxWidth

# **1048. Longest String Chain**

https://leetcode.com/problems/longest-string-chain/description/

You are given an array of words where each word consists of lowercase English letters.

wordA is a **predecessor** of wordB if and only if we can insert **exactly one** letter anywhere in wordA **without changing the order of the other characters** to make it equal to wordB.

* For example, "abc" is a **predecessor** of "abac", while "cba" is not a **predecessor** of "bcad".

A **word chain** is a sequence of words [word1, word2, ..., wordk] with k >= 1, where word1 is a **predecessor** of word2, word2 is a **predecessor** of word3, and so on. A single word is trivially a **word chain** with k == 1.

Return *the* ***length*** *of the* ***longest possible word chain*** *with words chosen from the given list of* words.

**Example 1:**

Input: words = ["a","b","ba","bca","bda","bdca"]  
Output: 4  
Explanation: One of the longest word chains is ["a","ba","bda","bdca"].

**Example 2:**

Input: words = ["xbc","pcxbcf","xb","cxbc","pcxbc"]  
Output: 5  
Explanation: All the words can be put in a word chain ["xb", "xbc", "cxbc", "pcxbc", "pcxbcf"].

**Example 3:**

Input: words = ["abcd","dbqca"]  
Output: 1  
Explanation: The trivial word chain ["abcd"] is one of the longest word chains.  
["abcd","dbqca"] is not a valid word chain because the ordering of the letters is changed.

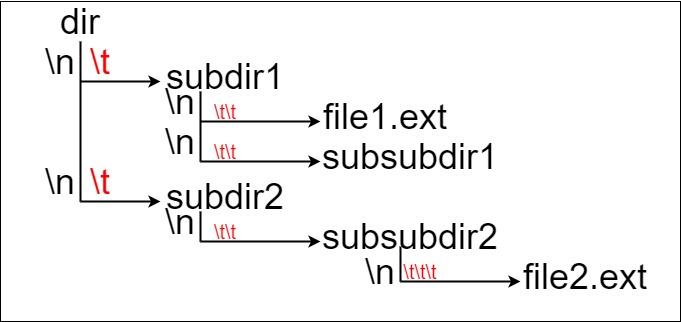
**Constraints:**

* 1 <= words.length <= 1000
* 1 <= words[i].length <= 16
* words[i] only consists of lowercase English letters.

# **388. Longest Absolute File Path**

https://leetcode.com/problems/longest-absolute-file-path/description/

Suppose we have a file system that stores both files and directories. An example of one system is represented in the following picture:



Here, we have dir as the only directory in the root. dir contains two subdirectories, subdir1 and subdir2. subdir1 contains a file file1.ext and subdirectory subsubdir1. subdir2 contains a subdirectory subsubdir2, which contains a file file2.ext.

In text form, it looks like this (with ⟶ representing the tab character):

dir  
⟶ subdir1  
⟶ ⟶ file1.ext  
⟶ ⟶ subsubdir1  
⟶ subdir2  
⟶ ⟶ subsubdir2  
⟶ ⟶ ⟶ file2.ext

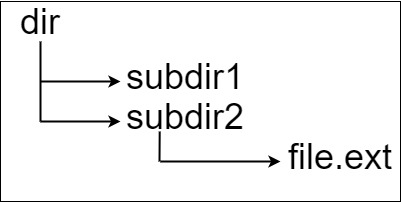
If we were to write this representation in code, it will look like this: "dir\n\tsubdir1\n\t\tfile1.ext\n\t\tsubsubdir1\n\tsubdir2\n\t\tsubsubdir2\n\t\t\tfile2.ext". Note that the '\n' and '\t' are the new-line and tab characters.

Every file and directory has a unique **absolute path** in the file system, which is the order of directories that must be opened to reach the file/directory itself, all concatenated by '/'s. Using the above example, the **absolute path** to file2.ext is "dir/subdir2/subsubdir2/file2.ext". Each directory name consists of letters, digits, and/or spaces. Each file name is of the form name.extension, where name and extension consist of letters, digits, and/or spaces.

Given a string input representing the file system in the explained format, return *the length of the* ***longest absolute path*** *to a* ***file*** *in the abstracted file system*. If there is no file in the system, return 0.

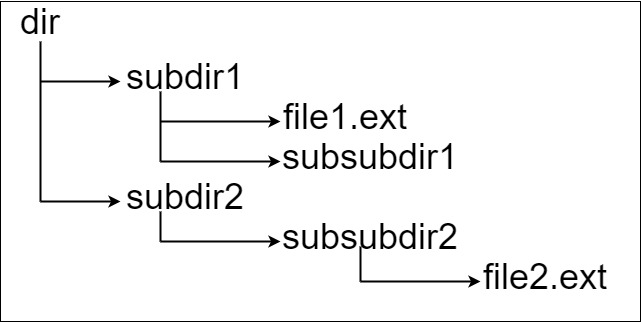
**Note** that the testcases are generated such that the file system is valid and no file or directory name has length 0.

**Example 1:**



Input: input = "dir\n\tsubdir1\n\tsubdir2\n\t\tfile.ext"  
Output: 20  
Explanation: We have only one file, and the absolute path is "dir/subdir2/file.ext" of length 20.

**Example 2:**



Input: input = "dir\n\tsubdir1\n\t\tfile1.ext\n\t\tsubsubdir1\n\tsubdir2\n\t\tsubsubdir2\n\t\t\tfile2.ext"  
Output: 32  
Explanation: We have two files:  
"dir/subdir1/file1.ext" of length 21  
"dir/subdir2/subsubdir2/file2.ext" of length 32.  
We return 32 since it is the longest absolute path to a file.

**Example 3:**

Input: input = "a"  
Output: 0  
Explanation: We do not have any files, just a single directory named "a".

**Constraints:**

* 1 <= input.length <= 104
* input may contain lowercase or uppercase English letters, a new line character '\n', a tab character '\t', a dot '.', a space ' ', and digits.
* All file and directory names have **positive** length.

# **446. Arithmetic Slices II - Subsequence**

https://leetcode.com/problems/arithmetic-slices-ii-subsequence/description/

Given an integer array nums, return *the number of all the* ***arithmetic subsequences*** *of* nums.

A sequence of numbers 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.
* For example, [1, 1, 2, 5, 7] is not an arithmetic sequence.

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**].

The test cases are generated so that the answer fits in **32-bit** integer.

**Example 1:**

Input: nums = [2,4,6,8,10]  
Output: 7  
Explanation: All arithmetic subsequence slices are:  
[2,4,6]  
[4,6,8]  
[6,8,10]  
[2,4,6,8]  
[4,6,8,10]  
[2,4,6,8,10]  
[2,6,10]

**Example 2:**

Input: nums = [7,7,7,7,7]  
Output: 16  
Explanation: Any subsequence of this array is arithmetic.

**Constraints:**

* 1  <= nums.length <= 1000
* -231 <= nums[i] <= 231 - 1

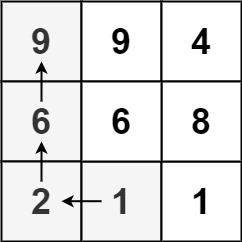
# **329. Longest Increasing Path in a Matrix**

https://leetcode.com/problems/longest-increasing-path-in-a-matrix/description/

Given an m x n integers matrix, return *the length of the longest increasing path in* matrix.

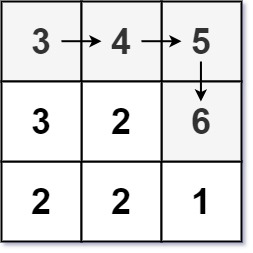
From each cell, you can either move in four directions: left, right, up, or down. You **may not** move **diagonally** or move **outside the boundary** (i.e., wrap-around is not allowed).

**Example 1:**



Input: matrix = [[9,9,4],[6,6,8],[2,1,1]]  
Output: 4  
Explanation: The longest increasing path is [1, 2, 6, 9].

**Example 2:**



Input: matrix = [[3,4,5],[3,2,6],[2,2,1]]  
Output: 4  
Explanation: The longest increasing path is [3, 4, 5, 6]. Moving diagonally is not allowed.

**Example 3:**

Input: matrix = [[1]]  
Output: 1

**Constraints:**

* m == matrix.length
* n == matrix[i].length
* 1 <= m, n <= 200
* 0 <= matrix[i][j] <= 231 - 1

# **690. Employee Importance**

https://leetcode.com/problems/employee-importance/description/

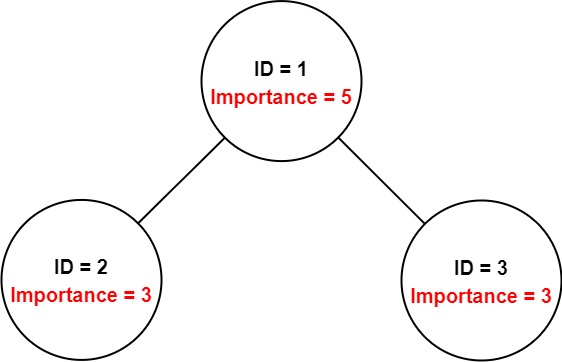
You have a data structure of employee information, including the employee's unique ID, importance value, and direct subordinates' IDs.

You are given an array of employees employees where:

* employees[i].id is the ID of the ith employee.
* employees[i].importance is the importance value of the ith employee.
* employees[i].subordinates is a list of the IDs of the direct subordinates of the ith employee.

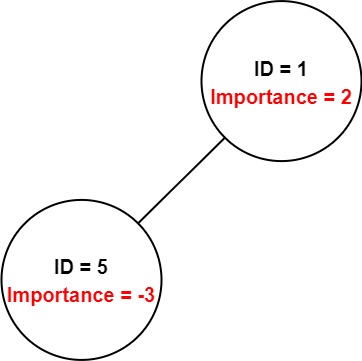
Given an integer id that represents an employee's ID, return *the* ***total*** *importance value of this employee and all their direct and indirect subordinates*.

**Example 1:**



Input: employees = [[1,5,[2,3]],[2,3,[]],[3,3,[]]], id = 1  
Output: 11  
Explanation: Employee 1 has an importance value of 5 and has two direct subordinates: employee 2 and employee 3.  
They both have an importance value of 3.  
Thus, the total importance value of employee 1 is 5 + 3 + 3 = 11.

**Example 2:**



Input: employees = [[1,2,[5]],[5,-3,[]]], id = 5  
Output: -3  
Explanation: Employee 5 has an importance value of -3 and has no direct subordinates.  
Thus, the total importance value of employee 5 is -3.

**Constraints:**

* 1 <= employees.length <= 2000
* 1 <= employees[i].id <= 2000
* All employees[i].id are **unique**.
* -100 <= employees[i].importance <= 100
* One employee has at most one direct leader and may have several subordinates.
* The IDs in employees[i].subordinates are valid IDs.

# **253. Meeting Rooms II**

https://leetcode.com/problems/meeting-rooms-ii/description/

Given an array of meeting time intervals intervals where intervals[i] = [starti, endi], return *the minimum number of conference rooms required*.

**Example 1:**

Input: intervals = [[0,30],[5,10],[15,20]]  
Output: 2

**Example 2:**

Input: intervals = [[7,10],[2,4]]  
Output: 1

**Constraints:**

* 1 <= intervals.length <= 104
* 0 <= starti < endi <= 106

# **2091. Removing Minimum and Maximum From Array**

https://leetcode.com/problems/removing-minimum-and-maximum-from-array/description/

You are given a **0-indexed** array of **distinct** integers nums.

There is an element in nums that has the **lowest** value and an element that has the **highest** value. We call them the **minimum** and **maximum** respectively. Your goal is to remove **both** these elements from the array.

A **deletion** is defined as either removing an element from the **front** of the array or removing an element from the **back** of the array.

Return *the* ***minimum*** *number of deletions it would take to remove* ***both*** *the minimum and maximum element from the array.*

**Example 1:**

Input: nums = [2,10,7,5,4,1,8,6]  
Output: 5  
Explanation:   
The minimum element in the array is nums[5], which is 1.  
The maximum element in the array is nums[1], which is 10.  
We can remove both the minimum and maximum by removing 2 elements from the front and 3 elements from the back.  
This results in 2 + 3 = 5 deletions, which is the minimum number possible.

**Example 2:**

Input: nums = [0,-4,19,1,8,-2,-3,5]  
Output: 3  
Explanation:   
The minimum element in the array is nums[1], which is -4.  
The maximum element in the array is nums[2], which is 19.  
We can remove both the minimum and maximum by removing 3 elements from the front.  
This results in only 3 deletions, which is the minimum number possible.

**Example 3:**

Input: nums = [101]  
Output: 1  
Explanation:   
There is only one element in the array, which makes it both the minimum and maximum element.  
We can remove it with 1 deletion.

**Constraints:**

* 1 <= nums.length <= 105
* -105 <= nums[i] <= 105
* The integers in nums are **distinct**.

# **413. Arithmetic Slices**

https://leetcode.com/problems/arithmetic-slices/description/

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.

**Example 1:**

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.

**Example 2:**

Input: nums = [1]  
Output: 0

**Constraints:**

* 1 <= nums.length <= 5000
* -1000 <= nums[i] <= 1000

# **685. Redundant Connection II**

https://leetcode.com/problems/redundant-connection-ii/description/

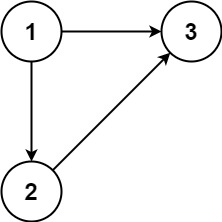
In this problem, a rooted tree is a **directed** graph such that, there is exactly one node (the root) for which all other nodes are descendants of this node, plus every node has exactly one parent, except for the root node which has no parents.

The given input is a directed graph that started as a rooted tree with n nodes (with distinct values from 1 to n), with one additional directed edge added. The added edge has two different vertices chosen from 1 to n, and was not an edge that already existed.

The resulting graph is given as a 2D-array of edges. Each element of edges is a pair [ui, vi] that represents a **directed** edge connecting nodes ui and vi, where ui is a parent of child vi.

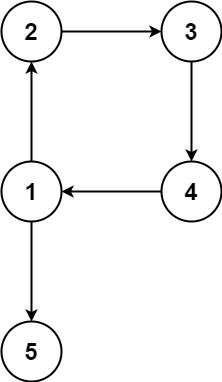
Return *an edge that can be removed so that the resulting graph is a rooted tree of* n *nodes*. If there are multiple answers, return the answer that occurs last in the given 2D-array.

**Example 1:**



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

**Example 2:**



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

**Constraints:**

* n == edges.length
* 3 <= n <= 1000
* edges[i].length == 2
* 1 <= ui, vi <= n
* ui != vi

# **729. My Calendar I**

https://leetcode.com/problems/my-calendar-i/description/

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.

**Example 1:**

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.

**Constraints:**

* 0 <= start < end <= 109
* At most 1000 calls will be made to book.

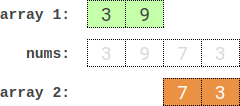
# **2035. Partition Array Into Two Arrays to Minimize Sum Difference**

https://leetcode.com/problems/partition-array-into-two-arrays-to-minimize-sum-difference/description/

You are given an integer array nums of 2 \* n integers. You need to partition nums into **two** arrays of length n to **minimize the absolute difference** of the **sums** of the arrays. To partition nums, put each element of nums into **one** of the two arrays.

Return *the* ***minimum*** *possible absolute difference*.

**Example 1:**

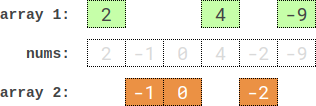


Input: nums = [3,9,7,3]  
Output: 2  
Explanation: One optimal partition is: [3,9] and [7,3].  
The absolute difference between the sums of the arrays is abs((3 + 9) - (7 + 3)) = 2.

**Example 2:**

Input: nums = [-36,36]  
Output: 72  
Explanation: One optimal partition is: [-36] and [36].  
The absolute difference between the sums of the arrays is abs((-36) - (36)) = 72.

**Example 3:**



Input: nums = [2,-1,0,4,-2,-9]  
Output: 0  
Explanation: One optimal partition is: [2,4,-9] and [-1,0,-2].  
The absolute difference between the sums of the arrays is abs((2 + 4 + -9) - (-1 + 0 + -2)) = 0.

**Constraints:**

* 1 <= n <= 15
* nums.length == 2 \* n
* -107 <= nums[i] <= 107

# **2013. Detect Squares**

https://leetcode.com/problems/detect-squares/description/

You are given a stream of points on the X-Y plane. Design an algorithm that:

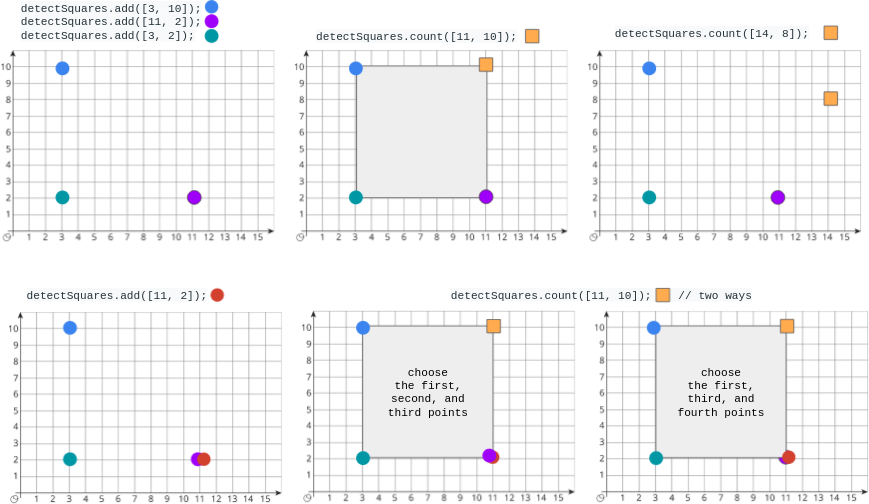
* **Adds** new points from the stream into a data structure. **Duplicate** points are allowed and should be treated as different points.
* Given a query point, **counts** the number of ways to choose three points from the data structure such that the three points and the query point form an **axis-aligned square** with **positive area**.

An **axis-aligned square** is a square whose edges are all the same length and are either parallel or perpendicular to the x-axis and y-axis.

Implement the DetectSquares class:

* DetectSquares() Initializes the object with an empty data structure.
* void add(int[] point) Adds a new point point = [x, y] to the data structure.
* int count(int[] point) Counts the number of ways to form **axis-aligned squares** with point point = [x, y] as described above.

**Example 1:**



Input  
["DetectSquares", "add", "add", "add", "count", "count", "add", "count"]  
[[], [[3, 10]], [[11, 2]], [[3, 2]], [[11, 10]], [[14, 8]], [[11, 2]], [[11, 10]]]  
Output  
[null, null, null, null, 1, 0, null, 2]  
  
Explanation  
DetectSquares detectSquares = new DetectSquares();  
detectSquares.add([3, 10]);  
detectSquares.add([11, 2]);  
detectSquares.add([3, 2]);  
detectSquares.count([11, 10]); // return 1. You can choose:  
 // - The first, second, and third points  
detectSquares.count([14, 8]); // return 0. The query point cannot form a square with any points in the data structure.  
detectSquares.add([11, 2]); // Adding duplicate points is allowed.  
detectSquares.count([11, 10]); // return 2. You can choose:  
 // - The first, second, and third points  
 // - The first, third, and fourth points

**Constraints:**

* point.length == 2
* 0 <= x, y <= 1000
* At most 3000 calls **in total** will be made to add and count.

# **490. The Maze**

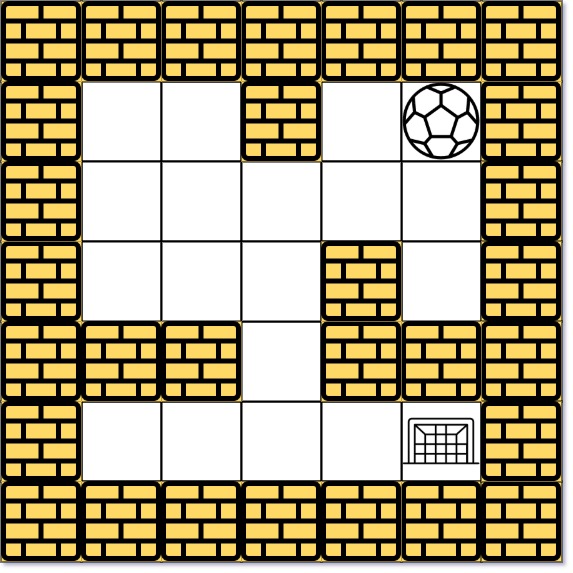
https://leetcode.com/problems/the-maze/description/

There is a ball in a maze with empty spaces (represented as 0) and walls (represented as 1). The ball can go through the empty spaces by rolling **up, down, left or right**, but it won't stop rolling until hitting a wall. When the ball stops, it could choose the next direction.

Given the m x n maze, the ball's start position and the destination, where start = [startrow, startcol] and destination = [destinationrow, destinationcol], return true if the ball can stop at the destination, otherwise return false.

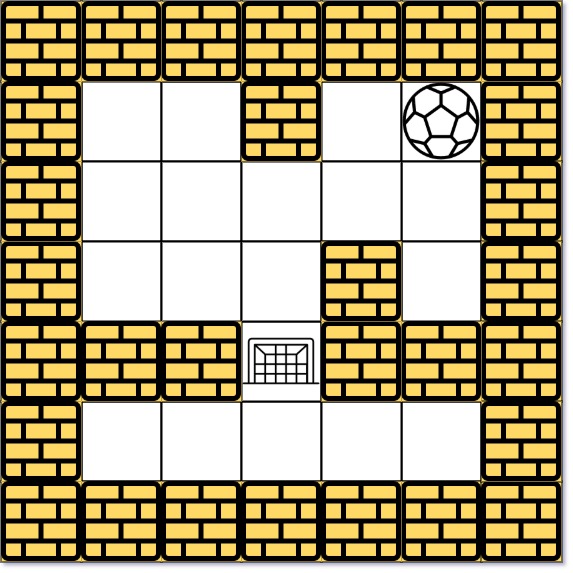
You may assume that **the borders of the maze are all walls** (see examples).

**Example 1:**



Input: maze = [[0,0,1,0,0],[0,0,0,0,0],[0,0,0,1,0],[1,1,0,1,1],[0,0,0,0,0]], start = [0,4], destination = [4,4]  
Output: true  
Explanation: One possible way is : left -> down -> left -> down -> right -> down -> right.

**Example 2:**



Input: maze = [[0,0,1,0,0],[0,0,0,0,0],[0,0,0,1,0],[1,1,0,1,1],[0,0,0,0,0]], start = [0,4], destination = [3,2]  
Output: false  
Explanation: There is no way for the ball to stop at the destination. Notice that you can pass through the destination but you cannot stop there.

**Example 3:**

Input: maze = [[0,0,0,0,0],[1,1,0,0,1],[0,0,0,0,0],[0,1,0,0,1],[0,1,0,0,0]], start = [4,3], destination = [0,1]  
Output: false

**Constraints:**

* m == maze.length
* n == maze[i].length
* 1 <= m, n <= 100
* maze[i][j] is 0 or 1.
* start.length == 2
* destination.length == 2
* 0 <= startrow, destinationrow <= m
* 0 <= startcol, destinationcol <= n
* Both the ball and the destination exist in an empty space, and they will not be in the same position initially.
* The maze contains **at least 2 empty spaces**.

# **2007. Find Original Array From Doubled Array**

https://leetcode.com/problems/find-original-array-from-doubled-array/description/

An integer array original is transformed into a **doubled** array changed by appending **twice the value** of every element in original, and then randomly **shuffling** the resulting array.

Given an array changed, return original *if* changed *is a* ***doubled*** *array. If* changed *is not a* ***doubled*** *array, return an empty array. The elements in* original *may be returned in* ***any*** *order*.

**Example 1:**

Input: changed = [1,3,4,2,6,8]  
Output: [1,3,4]  
Explanation: One possible original array could be [1,3,4]:  
- Twice the value of 1 is 1 \* 2 = 2.  
- Twice the value of 3 is 3 \* 2 = 6.  
- Twice the value of 4 is 4 \* 2 = 8.  
Other original arrays could be [4,3,1] or [3,1,4].

**Example 2:**

Input: changed = [6,3,0,1]  
Output: []  
Explanation: changed is not a doubled array.

**Example 3:**

Input: changed = [1]  
Output: []  
Explanation: changed is not a doubled array.

**Constraints:**

* 1 <= changed.length <= 105
* 0 <= changed[i] <= 105

# **1352. Product of the Last K Numbers**

https://leetcode.com/problems/product-of-the-last-k-numbers/description/

Design an algorithm that accepts a stream of integers and retrieves the product of the last k integers of the stream.

Implement the ProductOfNumbers class:

* ProductOfNumbers() Initializes the object with an empty stream.
* void add(int num) Appends the integer num to the stream.
* int getProduct(int k) Returns the product of the last k numbers in the current list. You can assume that always the current list has at least k numbers.

The test cases are generated so that, at any time, the product of any contiguous sequence of numbers will fit into a single 32-bit integer without overflowing.

**Example:**

Input  
["ProductOfNumbers","add","add","add","add","add","getProduct","getProduct","getProduct","add","getProduct"]  
[[],[3],[0],[2],[5],[4],[2],[3],[4],[8],[2]]  
  
Output  
[null,null,null,null,null,null,20,40,0,null,32]  
  
Explanation  
ProductOfNumbers productOfNumbers = new ProductOfNumbers();  
productOfNumbers.add(3); // [3]  
productOfNumbers.add(0); // [3,0]  
productOfNumbers.add(2); // [3,0,2]  
productOfNumbers.add(5); // [3,0,2,5]  
productOfNumbers.add(4); // [3,0,2,5,4]  
productOfNumbers.getProduct(2); // return 20. The product of the last 2 numbers is 5 \* 4 = 20  
productOfNumbers.getProduct(3); // return 40. The product of the last 3 numbers is 2 \* 5 \* 4 = 40  
productOfNumbers.getProduct(4); // return 0. The product of the last 4 numbers is 0 \* 2 \* 5 \* 4 = 0  
productOfNumbers.add(8); // [3,0,2,5,4,8]  
productOfNumbers.getProduct(2); // return 32. The product of the last 2 numbers is 4 \* 8 = 32

**Constraints:**

* 0 <= num <= 100
* 1 <= k <= 4 \* 104
* At most 4 \* 104 calls will be made to add and getProduct.
* The product of the stream at any point in time will fit in a **32-bit** integer.

# **1768. Merge Strings Alternately**

https://leetcode.com/problems/merge-strings-alternately/description/

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.*

**Example 1:**

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

**Example 2:**

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

**Example 3:**

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

**Constraints:**

* 1 <= word1.length, word2.length <= 100
* word1 and word2 consist of lowercase English letters.

# **980. Unique Paths III**

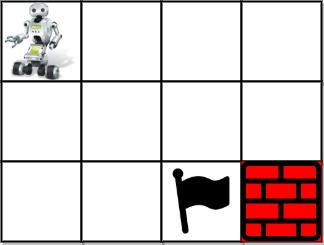
https://leetcode.com/problems/unique-paths-iii/description/

You are given an m x n integer array grid where grid[i][j] could be:

* 1 representing the starting square. There is exactly one starting square.
* 2 representing the ending square. There is exactly one ending square.
* 0 representing empty squares we can walk over.
* -1 representing obstacles that we cannot walk over.

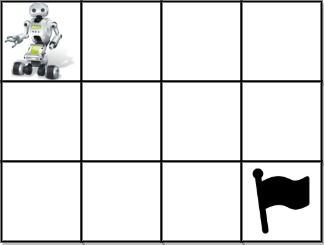
Return *the number of 4-directional walks from the starting square to the ending square, that walk over every non-obstacle square exactly once*.

**Example 1:**



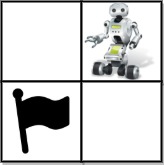
Input: grid = [[1,0,0,0],[0,0,0,0],[0,0,2,-1]]  
Output: 2  
Explanation: We have the following two paths:   
1. (0,0),(0,1),(0,2),(0,3),(1,3),(1,2),(1,1),(1,0),(2,0),(2,1),(2,2)  
2. (0,0),(1,0),(2,0),(2,1),(1,1),(0,1),(0,2),(0,3),(1,3),(1,2),(2,2)

**Example 2:**



Input: grid = [[1,0,0,0],[0,0,0,0],[0,0,0,2]]  
Output: 4  
Explanation: We have the following four paths:   
1. (0,0),(0,1),(0,2),(0,3),(1,3),(1,2),(1,1),(1,0),(2,0),(2,1),(2,2),(2,3)  
2. (0,0),(0,1),(1,1),(1,0),(2,0),(2,1),(2,2),(1,2),(0,2),(0,3),(1,3),(2,3)  
3. (0,0),(1,0),(2,0),(2,1),(2,2),(1,2),(1,1),(0,1),(0,2),(0,3),(1,3),(2,3)  
4. (0,0),(1,0),(2,0),(2,1),(1,1),(0,1),(0,2),(0,3),(1,3),(1,2),(2,2),(2,3)

**Example 3:**



Input: grid = [[0,1],[2,0]]  
Output: 0  
Explanation: There is no path that walks over every empty square exactly once.  
Note that the starting and ending square can be anywhere in the grid.

**Constraints:**

* m == grid.length
* n == grid[i].length
* 1 <= m, n <= 20
* 1 <= m \* n <= 20
* -1 <= grid[i][j] <= 2
* There is exactly one starting cell and one ending cell.

# **3. Longest Substring Without Repeating Characters**

https://leetcode.com/problems/longest-substring-without-repeating-characters/description/

Given a string s, find the length of the **longest**

**substring**

without repeating characters.

**Example 1:**

Input: s = "abcabcbb"  
Output: 3  
Explanation: The answer is "abc", with the length of 3.

**Example 2:**

Input: s = "bbbbb"  
Output: 1  
Explanation: The answer is "b", with the length of 1.

**Example 3:**

Input: s = "pwwkew"  
Output: 3  
Explanation: The answer is "wke", with the length of 3.  
Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.

**Constraints:**

* 0 <= s.length <= 5 \* 104
* s consists of English letters, digits, symbols and spaces.

# **1631. Path With Minimum Effort**

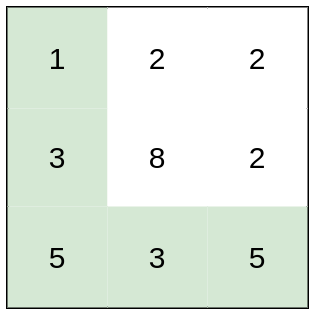
https://leetcode.com/problems/path-with-minimum-effort/description/

You are a hiker preparing for an upcoming hike. You are given heights, a 2D array of size rows x columns, where heights[row][col] represents the height of cell (row, col). You are situated in the top-left cell, (0, 0), and you hope to travel to the bottom-right cell, (rows-1, columns-1) (i.e., **0-indexed**). You can move **up**, **down**, **left**, or **right**, and you wish to find a route that requires the minimum **effort**.

A route's **effort** is the **maximum absolute difference** in heights between two consecutive cells of the route.

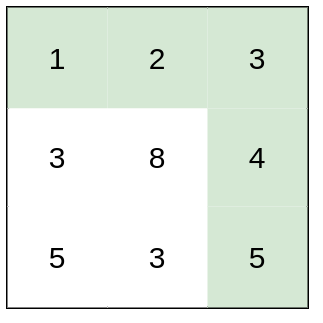
Return *the minimum* ***effort*** *required to travel from the top-left cell to the bottom-right cell.*

**Example 1:**



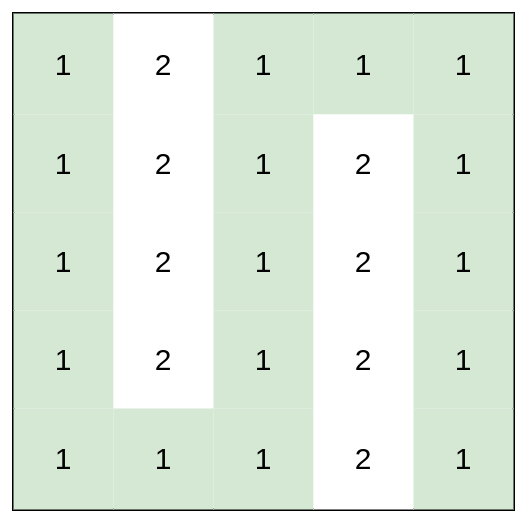
Input: heights = [[1,2,2],[3,8,2],[5,3,5]]  
Output: 2  
Explanation: The route of [1,3,5,3,5] has a maximum absolute difference of 2 in consecutive cells.  
This is better than the route of [1,2,2,2,5], where the maximum absolute difference is 3.

**Example 2:**



Input: heights = [[1,2,3],[3,8,4],[5,3,5]]  
Output: 1  
Explanation: The route of [1,2,3,4,5] has a maximum absolute difference of 1 in consecutive cells, which is better than route [1,3,5,3,5].

**Example 3:**



Input: heights = [[1,2,1,1,1],[1,2,1,2,1],[1,2,1,2,1],[1,2,1,2,1],[1,1,1,2,1]]  
Output: 0  
Explanation: This route does not require any effort.

**Constraints:**

* rows == heights.length
* columns == heights[i].length
* 1 <= rows, columns <= 100
* 1 <= heights[i][j] <= 106

# **410. Split Array Largest Sum**

https://leetcode.com/problems/split-array-largest-sum/description/

Given an integer array nums and an integer k, split nums into k non-empty subarrays such that the largest sum of any subarray is **minimized**.

Return *the minimized largest sum of the split*.

A **subarray** is a contiguous part of the array.

**Example 1:**

Input: nums = [7,2,5,10,8], k = 2  
Output: 18  
Explanation: There are four ways to split nums into two subarrays.  
The best way is to split it into [7,2,5] and [10,8], where the largest sum among the two subarrays is only 18.

**Example 2:**

Input: nums = [1,2,3,4,5], k = 2  
Output: 9  
Explanation: There are four ways to split nums into two subarrays.  
The best way is to split it into [1,2,3] and [4,5], where the largest sum among the two subarrays is only 9.

**Constraints:**

* 1 <= nums.length <= 1000
* 0 <= nums[i] <= 106
* 1 <= k <= min(50, nums.length)

# **862. Shortest Subarray with Sum at Least K**

https://leetcode.com/problems/shortest-subarray-with-sum-at-least-k/description/

Given an integer array nums and an integer k, return *the length of the shortest non-empty* ***subarray*** *of* nums *with a sum of at least* k. If there is no such **subarray**, return -1.

A **subarray** is a **contiguous** part of an array.

**Example 1:**

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

**Example 2:**

Input: nums = [1,2], k = 4  
Output: -1

**Example 3:**

Input: nums = [2,-1,2], k = 3  
Output: 3

**Constraints:**

* 1 <= nums.length <= 105
* -105 <= nums[i] <= 105
* 1 <= k <= 109

# **652. Find Duplicate Subtrees**

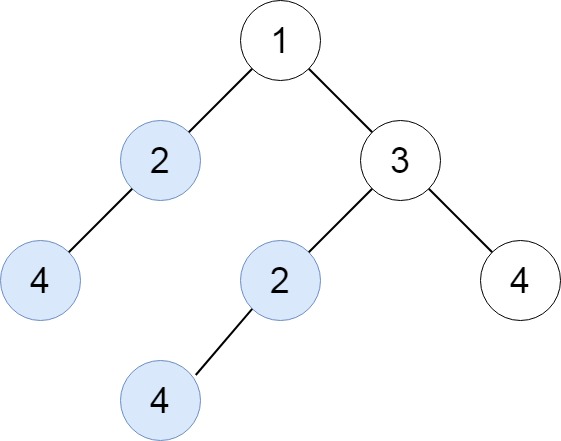
https://leetcode.com/problems/find-duplicate-subtrees/description/

Given the root of a binary tree, return all **duplicate subtrees**.

For each kind of duplicate subtrees, you only need to return the root node of any **one** of them.

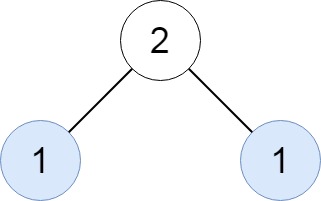
Two trees are **duplicate** if they have the **same structure** with the **same node values**.

**Example 1:**



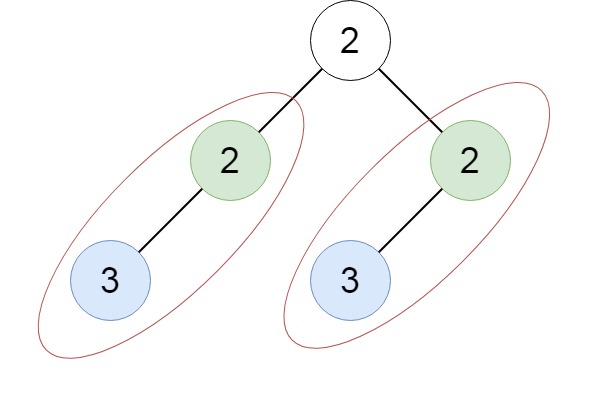
Input: root = [1,2,3,4,null,2,4,null,null,4]  
Output: [[2,4],[4]]

**Example 2:**



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

**Example 3:**



Input: root = [2,2,2,3,null,3,null]  
Output: [[2,3],[3]]

**Constraints:**

* The number of the nodes in the tree will be in the range [1, 5000]
* -200 <= Node.val <= 200

# **200. Number of Islands**

https://leetcode.com/problems/number-of-islands/description/

Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return *the number of islands*.

An **island** is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

**Example 1:**

Input: grid = [  
 ["1","1","1","1","0"],  
 ["1","1","0","1","0"],  
 ["1","1","0","0","0"],  
 ["0","0","0","0","0"]  
]  
Output: 1

**Example 2:**

Input: grid = [  
 ["1","1","0","0","0"],  
 ["1","1","0","0","0"],  
 ["0","0","1","0","0"],  
 ["0","0","0","1","1"]  
]  
Output: 3

**Constraints:**

* m == grid.length
* n == grid[i].length
* 1 <= m, n <= 300
* grid[i][j] is '0' or '1'.

# **1630. Arithmetic Subarrays**

https://leetcode.com/problems/arithmetic-subarrays/description/

A sequence of numbers is called **arithmetic** if it consists of at least two elements, and the difference between every two consecutive elements is the same. More formally, a sequence s is arithmetic if and only if s[i+1] - s[i] == s[1] - s[0] for all valid i.

For example, these are **arithmetic** sequences:

1, 3, 5, 7, 9  
7, 7, 7, 7  
3, -1, -5, -9

The following sequence is not **arithmetic**:

1, 1, 2, 5, 7

You are given an array of n integers, nums, and two arrays of m integers each, l and r, representing the m range queries, where the ith query is the range [l[i], r[i]]. All the arrays are **0-indexed**.

Return *a list of* boolean *elements* answer*, where* answer[i] *is* true *if the subarray* nums[l[i]], nums[l[i]+1], ... , nums[r[i]] *can be* ***rearranged*** *to form an* ***arithmetic*** *sequence, and* false *otherwise.*

**Example 1:**

Input: nums = [4,6,5,9,3,7], l = [0,0,2], r = [2,3,5]  
Output: [true,false,true]  
Explanation:  
In the 0th query, the subarray is [4,6,5]. This can be rearranged as [6,5,4], which is an arithmetic sequence.  
In the 1st query, the subarray is [4,6,5,9]. This cannot be rearranged as an arithmetic sequence.  
In the 2nd query, the subarray is [5,9,3,7]. This can be rearranged as [3,5,7,9], which is an arithmetic sequence.

**Example 2:**

Input: nums = [-12,-9,-3,-12,-6,15,20,-25,-20,-15,-10], l = [0,1,6,4,8,7], r = [4,4,9,7,9,10]  
Output: [false,true,false,false,true,true]

**Constraints:**

* n == nums.length
* m == l.length
* m == r.length
* 2 <= n <= 500
* 1 <= m <= 500
* 0 <= l[i] < r[i] < n
* -105 <= nums[i] <= 105

# **2115. Find All Possible Recipes from Given Supplies**

https://leetcode.com/problems/find-all-possible-recipes-from-given-supplies/description/

You have information about n different recipes. You are given a string array recipes and a 2D string array ingredients. The ith 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.

**Example 1:**

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".

**Example 2:**

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".

**Example 3:**

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".

**Constraints:**

* n == recipes.length == ingredients.length
* 1 <= n <= 100
* 1 <= ingredients[i].length, supplies.length <= 100
* 1 <= recipes[i].length, ingredients[i][j].length, supplies[k].length <= 10
* recipes[i], ingredients[i][j], and supplies[k] consist only of lowercase English letters.
* All the values of recipes and supplies combined are unique.
* Each ingredients[i] does not contain any duplicate values.

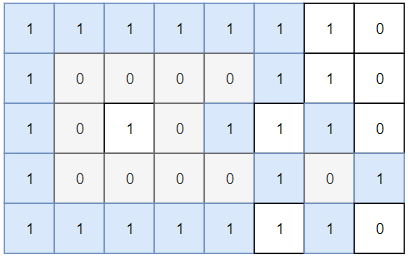
# **1254. Number of Closed Islands**

https://leetcode.com/problems/number-of-closed-islands/description/

Given a 2D grid consists of 0s (land) and 1s (water).  An *island* is a maximal 4-directionally connected group of 0s and a *closed island* is an island **totally** (all left, top, right, bottom) surrounded by 1s.

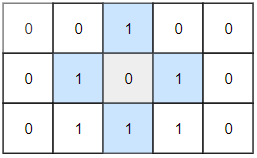
Return the number of *closed islands*.

**Example 1:**



Input: grid = [[1,1,1,1,1,1,1,0],[1,0,0,0,0,1,1,0],[1,0,1,0,1,1,1,0],[1,0,0,0,0,1,0,1],[1,1,1,1,1,1,1,0]]  
Output: 2  
Explanation:   
Islands in gray are closed because they are completely surrounded by water (group of 1s).

**Example 2:**



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

**Example 3:**

Input: grid = [[1,1,1,1,1,1,1],  
  [1,0,0,0,0,0,1],  
  [1,0,1,1,1,0,1],  
  [1,0,1,0,1,0,1],  
  [1,0,1,1,1,0,1],  
  [1,0,0,0,0,0,1],  
 [1,1,1,1,1,1,1]]  
Output: 2

**Constraints:**

* 1 <= grid.length, grid[0].length <= 100
* 0 <= grid[i][j] <=1

# **5. Longest Palindromic Substring**

https://leetcode.com/problems/longest-palindromic-substring/description/

Given a string s, return *the longest*

*palindromic*

*substring*

in s.

**Example 1:**

Input: s = "babad"  
Output: "bab"  
Explanation: "aba" is also a valid answer.

**Example 2:**

Input: s = "cbbd"  
Output: "bb"

**Constraints:**

* 1 <= s.length <= 1000
* s consist of only digits and English letters.

# **459. Repeated Substring Pattern**

https://leetcode.com/problems/repeated-substring-pattern/description/

Given a string s, check if it can be constructed by taking a substring of it and appending multiple copies of the substring together.

**Example 1:**

Input: s = "abab"  
Output: true  
Explanation: It is the substring "ab" twice.

**Example 2:**

Input: s = "aba"  
Output: false

**Example 3:**

Input: s = "abcabcabcabc"  
Output: true  
Explanation: It is the substring "abc" four times or the substring "abcabc" twice.

**Constraints:**

* 1 <= s.length <= 104
* s consists of lowercase English letters.

# **1980. Find Unique Binary String**

https://leetcode.com/problems/find-unique-binary-string/description/

Given an array of strings nums containing n **unique** binary strings each of length n, return *a binary string of length* n *that* ***does not appear*** *in* nums*. If there are multiple answers, you may return* ***any*** *of them*.

**Example 1:**

Input: nums = ["01","10"]  
Output: "11"  
Explanation: "11" does not appear in nums. "00" would also be correct.

**Example 2:**

Input: nums = ["00","01"]  
Output: "11"  
Explanation: "11" does not appear in nums. "10" would also be correct.

**Example 3:**

Input: nums = ["111","011","001"]  
Output: "101"  
Explanation: "101" does not appear in nums. "000", "010", "100", and "110" would also be correct.

**Constraints:**

* n == nums.length
* 1 <= n <= 16
* nums[i].length == n
* nums[i] is either '0' or '1'.
* All the strings of nums are **unique**.

# **650. 2 Keys Keyboard**

https://leetcode.com/problems/2-keys-keyboard/description/

There is only one character 'A' on the screen of a notepad. You can perform one of two operations on this notepad for each step:

* Copy All: You can copy all the characters present on the screen (a partial copy is not allowed).
* Paste: You can paste the characters which are copied last time.

Given an integer n, return *the minimum number of operations to get the character* 'A' *exactly* n *times on the screen*.

**Example 1:**

Input: n = 3  
Output: 3  
Explanation: Initially, we have one character 'A'.  
In step 1, we use Copy All operation.  
In step 2, we use Paste operation to get 'AA'.  
In step 3, we use Paste operation to get 'AAA'.

**Example 2:**

Input: n = 1  
Output: 0

**Constraints:**

* 1 <= n <= 1000

# **417. Pacific Atlantic Water Flow**

https://leetcode.com/problems/pacific-atlantic-water-flow/description/

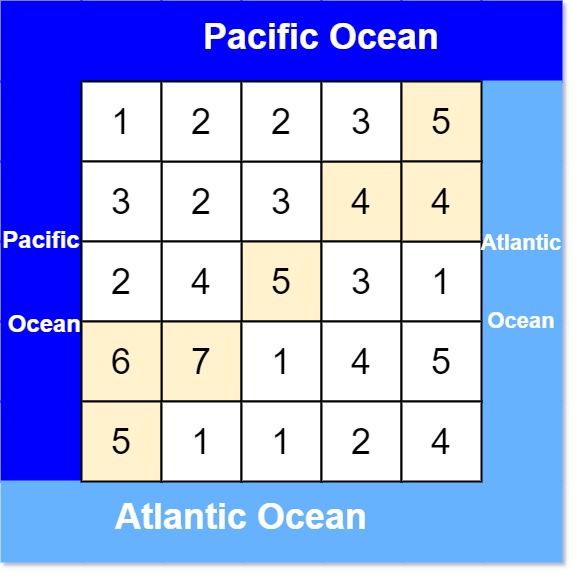
There is an m x n rectangular island that borders both the **Pacific Ocean** and **Atlantic Ocean**. The **Pacific Ocean** touches the island's left and top edges, and the **Atlantic Ocean** touches the island's right and bottom edges.

The island is partitioned into a grid of square cells. You are given an m x n integer matrix heights where heights[r][c] represents the **height above sea level** of the cell at coordinate (r, c).

The island receives a lot of rain, and the rain water can flow to neighboring cells directly north, south, east, and west if the neighboring cell's height is **less than or equal to** the current cell's height. Water can flow from any cell adjacent to an ocean into the ocean.

Return *a* ***2D list*** *of grid coordinates* result *where* result[i] = [ri, ci] *denotes that rain water can flow from cell* (ri, ci) *to* ***both*** *the Pacific and Atlantic oceans*.

**Example 1:**



Input: heights = [[1,2,2,3,5],[3,2,3,4,4],[2,4,5,3,1],[6,7,1,4,5],[5,1,1,2,4]]  
Output: [[0,4],[1,3],[1,4],[2,2],[3,0],[3,1],[4,0]]  
Explanation: The following cells can flow to the Pacific and Atlantic oceans, as shown below:  
[0,4]: [0,4] -> Pacific Ocean   
  [0,4] -> Atlantic Ocean  
[1,3]: [1,3] -> [0,3] -> Pacific Ocean   
  [1,3] -> [1,4] -> Atlantic Ocean  
[1,4]: [1,4] -> [1,3] -> [0,3] -> Pacific Ocean   
  [1,4] -> Atlantic Ocean  
[2,2]: [2,2] -> [1,2] -> [0,2] -> Pacific Ocean   
  [2,2] -> [2,3] -> [2,4] -> Atlantic Ocean  
[3,0]: [3,0] -> Pacific Ocean   
  [3,0] -> [4,0] -> Atlantic Ocean  
[3,1]: [3,1] -> [3,0] -> Pacific Ocean   
  [3,1] -> [4,1] -> Atlantic Ocean  
[4,0]: [4,0] -> Pacific Ocean   
 [4,0] -> Atlantic Ocean  
Note that there are other possible paths for these cells to flow to the Pacific and Atlantic oceans.

**Example 2:**

Input: heights = [[1]]  
Output: [[0,0]]  
Explanation: The water can flow from the only cell to the Pacific and Atlantic oceans.

**Constraints:**

* m == heights.length
* n == heights[r].length
* 1 <= m, n <= 200
* 0 <= heights[r][c] <= 105

# **295. Find Median from Data Stream**

https://leetcode.com/problems/find-median-from-data-stream/description/

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.

**Example 1:**

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

**Constraints:**

* -105 <= num <= 105
* There will be at least one element in the data structure before calling findMedian.
* At most 5 \* 104 calls will be made to addNum and findMedian.

**Follow up:**

* If all integer numbers from the stream are in the range [0, 100], how would you optimize your solution?
* If 99% of all integer numbers from the stream are in the range [0, 100], how would you optimize your solution?

# **2235. Add Two Integers**

https://leetcode.com/problems/add-two-integers/description/ Given two integers num1 and num2, return *the* ***sum*** *of the two integers*.

**Example 1:**

Input: num1 = 12, num2 = 5  
Output: 17  
Explanation: num1 is 12, num2 is 5, and their sum is 12 + 5 = 17, so 17 is returned.

**Example 2:**

Input: num1 = -10, num2 = 4  
Output: -6  
Explanation: num1 + num2 = -6, so -6 is returned.

**Constraints:**

* -100 <= num1, num2 <= 100

# **300. Longest Increasing Subsequence**

https://leetcode.com/problems/longest-increasing-subsequence/description/

Given an integer array nums, return *the length of the longest* ***strictly increasing***

***subsequence***

.

**Example 1:**

Input: nums = [10,9,2,5,3,7,101,18]  
Output: 4  
Explanation: The longest increasing subsequence is [2,3,7,101], therefore the length is 4.

**Example 2:**

Input: nums = [0,1,0,3,2,3]  
Output: 4

**Example 3:**

Input: nums = [7,7,7,7,7,7,7]  
Output: 1

**Constraints:**

* 1 <= nums.length <= 2500
* -104 <= nums[i] <= 104

**Follow up:** Can you come up with an algorithm that runs in O(n log(n)) time complexity?

# **832. Flipping an Image**

https://leetcode.com/problems/flipping-an-image/description/

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].

**Example 1:**

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]]

**Example 2:**

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]]

**Constraints:**

* n == image.length
* n == image[i].length
* 1 <= n <= 20
* images[i][j] is either 0 or 1.

# **88. Merge Sorted Array**

https://leetcode.com/problems/merge-sorted-array/description/

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.

**Example 1:**

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.

**Example 2:**

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].

**Example 3:**

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.

**Constraints:**

* nums1.length == m + n
* nums2.length == n
* 0 <= m, n <= 200
* 1 <= m + n <= 200
* -109 <= nums1[i], nums2[j] <= 109

**Follow up:** Can you come up with an algorithm that runs in O(m + n) time?

# **718. Maximum Length of Repeated Subarray**

https://leetcode.com/problems/maximum-length-of-repeated-subarray/description/

Given two integer arrays nums1 and nums2, return *the maximum length of a subarray that appears in* ***both*** *arrays*.

**Example 1:**

Input: nums1 = [1,2,3,2,1], nums2 = [3,2,1,4,7]  
Output: 3  
Explanation: The repeated subarray with maximum length is [3,2,1].

**Example 2:**

Input: nums1 = [0,0,0,0,0], nums2 = [0,0,0,0,0]  
Output: 5  
Explanation: The repeated subarray with maximum length is [0,0,0,0,0].

**Constraints:**

* 1 <= nums1.length, nums2.length <= 1000
* 0 <= nums1[i], nums2[i] <= 100

# **210. Course Schedule II**

https://leetcode.com/problems/course-schedule-ii/description/

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] = [ai, bi] indicates that you **must** take course bi first if you want to take course ai.

* 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**.

**Example 1:**

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].

**Example 2:**

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].

**Example 3:**

Input: numCourses = 1, prerequisites = []  
Output: [0]

**Constraints:**

* 1 <= numCourses <= 2000
* 0 <= prerequisites.length <= numCourses \* (numCourses - 1)
* prerequisites[i].length == 2
* 0 <= ai, bi < numCourses
* ai != bi
* All the pairs [ai, bi] are **distinct**.

# **1361. Validate Binary Tree Nodes**

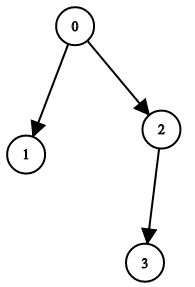
https://leetcode.com/problems/validate-binary-tree-nodes/description/

You have n binary tree nodes numbered from 0 to n - 1 where node i has two children leftChild[i] and rightChild[i], return true if and only if **all** the given nodes form **exactly one** valid binary tree.

If node i has no left child then leftChild[i] will equal -1, similarly for the right child.

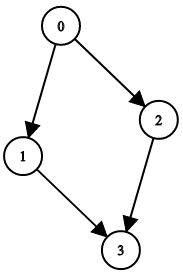
Note that the nodes have no values and that we only use the node numbers in this problem.

**Example 1:**



Input: n = 4, leftChild = [1,-1,3,-1], rightChild = [2,-1,-1,-1]  
Output: true

**Example 2:**



Input: n = 4, leftChild = [1,-1,3,-1], rightChild = [2,3,-1,-1]  
Output: false

**Example 3:**



Input: n = 2, leftChild = [1,0], rightChild = [-1,-1]  
Output: false

**Constraints:**

* n == leftChild.length == rightChild.length
* 1 <= n <= 104
* -1 <= leftChild[i], rightChild[i] <= n - 1

# **72. Edit Distance**

https://leetcode.com/problems/edit-distance/description/

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

**Example 1:**

Input: word1 = "horse", word2 = "ros"  
Output: 3  
Explanation:   
horse -> rorse (replace 'h' with 'r')  
rorse -> rose (remove 'r')  
rose -> ros (remove 'e')

**Example 2:**

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')

**Constraints:**

* 0 <= word1.length, word2.length <= 500
* word1 and word2 consist of lowercase English letters.

# **470. Implement Rand10() Using Rand7()**

https://leetcode.com/problems/implement-rand10-using-rand7/description/

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().

**Example 1:**

Input: n = 1  
Output: [2]

**Example 2:**

Input: n = 2  
Output: [2,8]

**Example 3:**

Input: n = 3  
Output: [3,8,10]

**Constraints:**

* 1 <= n <= 105

**Follow up:**

* What is the [expected value](https://en.wikipedia.org/wiki/Expected_value) for the number of calls to rand7() function?
* Could you minimize the number of calls to rand7()?

# **2. Add Two Numbers**

https://leetcode.com/problems/add-two-numbers/description/

You are given two **non-empty** linked lists representing two non-negative integers. The digits are stored in **reverse order**, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

**Example 1:**



Input: l1 = [2,4,3], l2 = [5,6,4]  
Output: [7,0,8]  
Explanation: 342 + 465 = 807.

**Example 2:**

Input: l1 = [0], l2 = [0]  
Output: [0]

**Example 3:**

Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]  
Output: [8,9,9,9,0,0,0,1]

**Constraints:**

* The number of nodes in each linked list is in the range [1, 100].
* 0 <= Node.val <= 9
* It is guaranteed that the list represents a number that does not have leading zeros.

# **460. LFU Cache**

https://leetcode.com/problems/lfu-cache/description/

Design and implement a data structure for a [Least Frequently Used (LFU)](https://en.wikipedia.org/wiki/Least_frequently_used) cache.

Implement the LFUCache class:

* LFUCache(int capacity) Initializes the object with the capacity of the data structure.
* int get(int key) Gets the value of the key if the key exists in the cache. Otherwise, returns -1.
* void put(int key, int value) Update the value of the key if present, or inserts the key if not already present. When the cache reaches its capacity, it should invalidate and remove the **least frequently used** key before inserting a new item. For this problem, when there is a **tie** (i.e., two or more keys with the same frequency), the **least recently used** key would be invalidated.

To determine the least frequently used key, a **use counter** is maintained for each key in the cache. The key with the smallest **use counter** is the least frequently used key.

When a key is first inserted into the cache, its **use counter** is set to 1 (due to the put operation). The **use counter** for a key in the cache is incremented either a get or put operation is called on it.

The functions get and put must each run in O(1) average time complexity.

**Example 1:**

Input  
["LFUCache", "put", "put", "get", "put", "get", "get", "put", "get", "get", "get"]  
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [3], [4, 4], [1], [3], [4]]  
Output  
[null, null, null, 1, null, -1, 3, null, -1, 3, 4]  
  
Explanation  
// cnt(x) = the use counter for key x  
// cache=[] will show the last used order for tiebreakers (leftmost element is most recent)  
LFUCache lfu = new LFUCache(2);  
lfu.put(1, 1); // cache=[1,\_], cnt(1)=1  
lfu.put(2, 2); // cache=[2,1], cnt(2)=1, cnt(1)=1  
lfu.get(1); // return 1  
 // cache=[1,2], cnt(2)=1, cnt(1)=2  
lfu.put(3, 3); // 2 is the LFU key because cnt(2)=1 is the smallest, invalidate 2.  
  // cache=[3,1], cnt(3)=1, cnt(1)=2  
lfu.get(2); // return -1 (not found)  
lfu.get(3); // return 3  
 // cache=[3,1], cnt(3)=2, cnt(1)=2  
lfu.put(4, 4); // Both 1 and 3 have the same cnt, but 1 is LRU, invalidate 1.  
 // cache=[4,3], cnt(4)=1, cnt(3)=2  
lfu.get(1); // return -1 (not found)  
lfu.get(3); // return 3  
 // cache=[3,4], cnt(4)=1, cnt(3)=3  
lfu.get(4); // return 4  
 // cache=[4,3], cnt(4)=2, cnt(3)=3

**Constraints:**

* 1 <= capacity <= 104
* 0 <= key <= 105
* 0 <= value <= 109
* At most 2 \* 105 calls will be made to get and put.