Hashing

1.

Write an algorithm to determine if a number `n` is happy.

A **happy number** is a number defined by the following process:

- Starting with any positive integer, replace the number by the sum of the squares of its digits.
- Repeat the process until the number equals 1 (where it will stay), or it **loops endlessly in a cycle** which does not include 1.
- Those numbers for which this process **ends in 1** are happy.

Return `true` *if* `n` *is a happy number, and* `false` *if not.*

Example 1:

> **Input:** n = 19
> **Output:** true
> **Explanation:**
> $1^2 + 9^2 = 82$
> $8^2 + 2^2 = 68$
> $6^2 + 8^2 = 100$
> $1^2 + 0^2 + 0^2 = 1$

Example 2:

> **Input:** n = 2
> **Output:** false

2.

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

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

Example 1:

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

Example 2:

> **Input:** nums = [1,2,3,4]
>
> **Output:** 2
>
> **Explanation:**
>
> The longest harmonious subsequences are `[1,2]`, `[2,3]`, and `[3,4]`, all of which have a length of 2.

Example 3:

> **Input:** nums = [1,1,1,1]
>
> **Output:** 0
>
> **Explanation:**
>
> No harmonic subsequence exists.

3.

The **DNA sequence** is composed of a series of nucleotides abbreviated as `'A'`, `'C'`, `'G'`, and `'T'`.

- For example, `"ACGAATTCCG"` is a **DNA sequence**.

When studying **DNA**, it is useful to identify repeated sequences within the DNA.

Given a string `s` that represents a **DNA sequence**, return all the `10`**-letter-long** sequences (substrings) that occur more than once in a DNA molecule. You may return the answer in **any order**.

Example 1:

> **Input:** s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT"
> **Output:** ["AAAAACCCCC","CCCCCAAAAA"]

Example 2:

> **Input:** s = "AAAAAAAAAAAAA"
> **Output:** ["AAAAAAAAAA"]

Tree, Binary Tree:

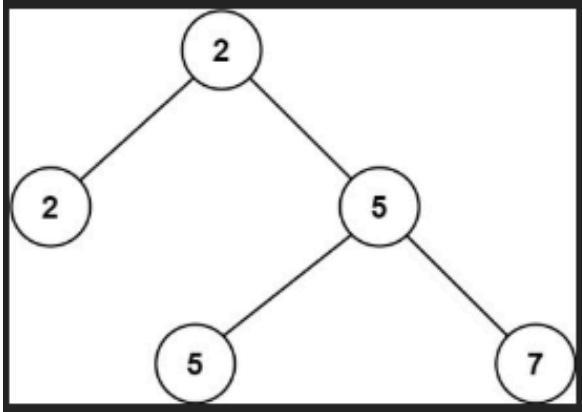
1.

Given a non-empty special binary tree consisting of nodes with the non-negative value, where each node in this tree has exactly `two` or `zero` sub-node. If the node has two sub-nodes, then this node's value is the smaller value among its two sub-nodes. More formally, the property `root.val = min(root.left.val, root.right.val)` always holds.

Given such a binary tree, you need to output the **second minimum** value in the set made of all the nodes' value in the whole tree.

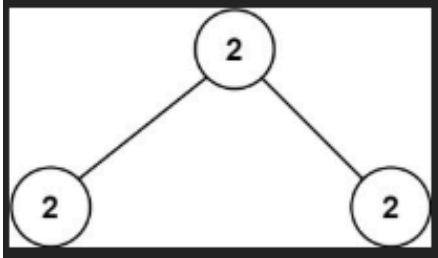
If no such second minimum value exists, output -1 instead.

Example 1:




```
Input: root = [2,2,5,null,null,5,7]
Output: 5
Explanation: The smallest value is 2, the second smallest value is 5.
```

Example 2:




```
Input: root = [2,2,2]
Output: -1
Explanation: The smallest value is 2, but there isn't any second smallest value.
```

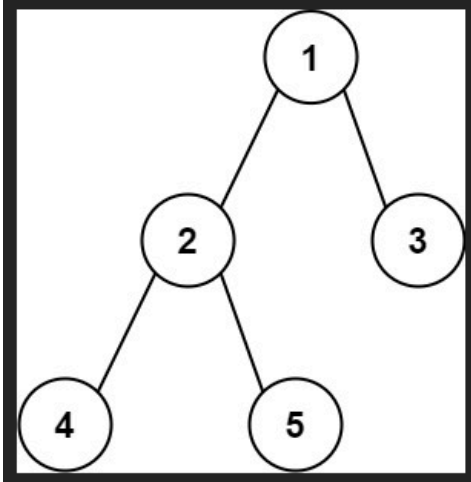
2.

Given the `root` of a binary tree, return *the length of the **diameter** of the tree*.

The **diameter** of a binary tree is the **length** of the longest path between any two nodes in a tree. This path may or may not pass through the `root`.

The **length** of a path between two nodes is represented by the number of edges between them.

Example 1:



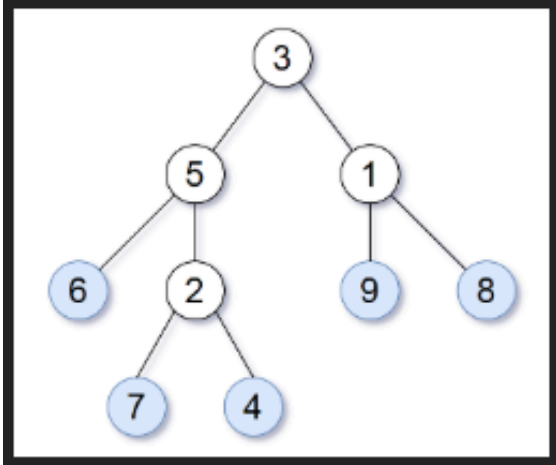
> **Input:** root = [1,2,3,4,5]
> **Output:** 3
> **Explanation:** 3 is the length of the path [4,2,1,3] or [5,2,1,3].

Example 2:

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

3.

Consider all the leaves of a binary tree, from left to right order, the values of those leaves form a **leaf value sequence**.

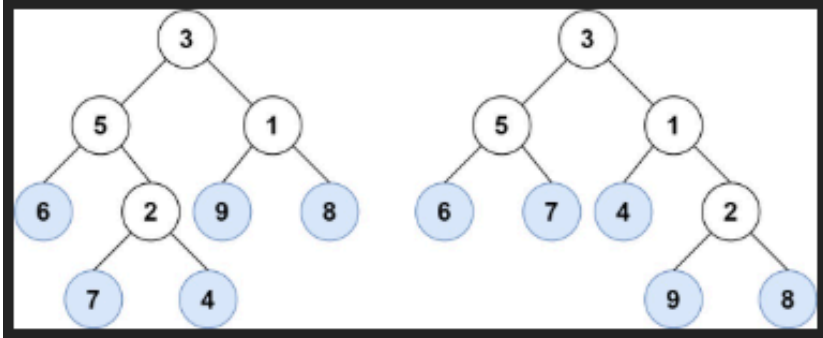


For example, in the given tree above, the leaf value sequence is `(6, 7, 4, 9, 8)`.

Two binary trees are considered *leaf-similar* if their leaf value sequence is the same.

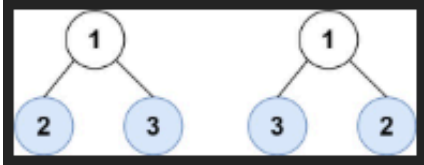
Return `true` if and only if the two given trees with head nodes `root1` and `root2` are leaf-similar.

Example 1:




```
Input: root1 = [3,5,1,6,2,9,8,null,null,7,4], root2 = [3,5,1,6,7,4,2,null,null,null,null,null,null,9,8]
Output: true
```

Example 2:



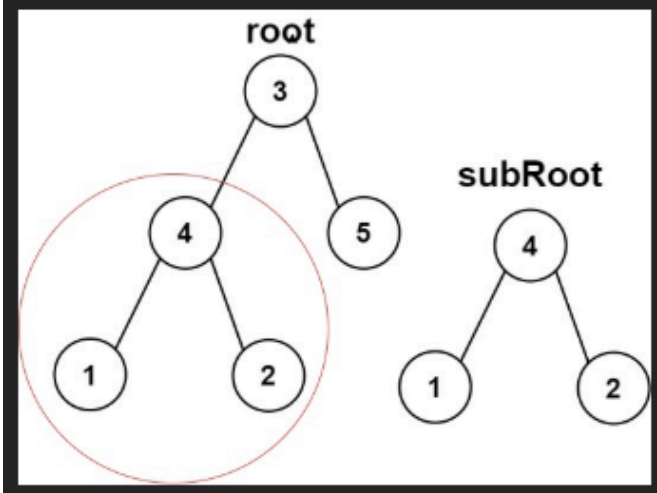

```
Input: root1 = [1,2,3], root2 = [1,3,2]
Output: false
```

4.

Given the roots of two binary trees `root` and `subRoot`, return `true` if there is a subtree of `root` with the same structure and node values of `subRoot` and `false` otherwise.

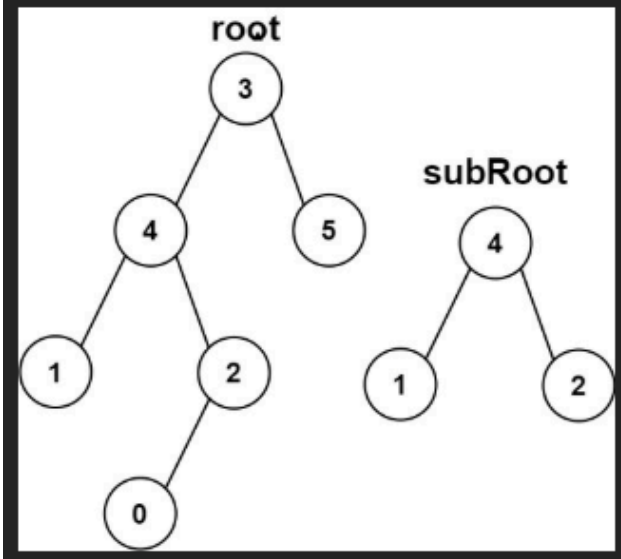
A subtree of a binary tree `tree` is a tree that consists of a node in `tree` and all of this node's descendants. The tree `tree` could also be considered as a subtree of itself.

Example 1:




```
Input: root = [3,4,5,1,2], subRoot = [4,1,2]
Output: true
```

Example 2:



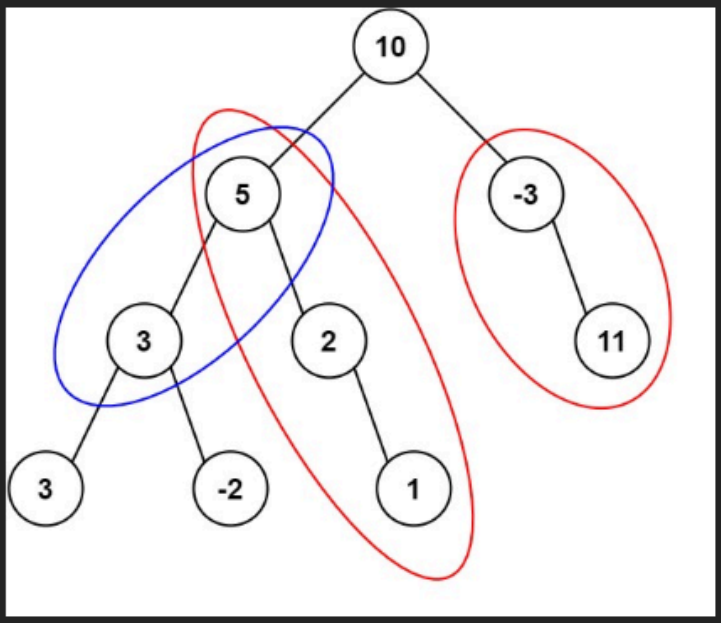

```
Input: root = [3,4,5,1,2,null,null,null,null,0], subRoot = [4,1,2]
Output: false
```

5.

Given the `root` of a binary tree and an integer `targetSum`, return *the number of paths where the sum of the values along the path equals* `targetSum`.

The path does not need to start or end at the root or a leaf, but it must go downwards (i.e., traveling only from parent nodes to child nodes).

Example 1:




```
Input: root = [10,5,-3,3,2,null,11,3,-2,null,1], targetSum = 8
Output: 3
Explanation: The paths that sum to 8 are shown.
```

Example 2:

```
Input: root = [5,4,8,11,null,13,4,7,2,null,null,5,1], targetSum = 22
Output: 3
```

6.

Two players play a turn based game on a binary tree. We are given the `root` of this binary tree, and the number of nodes `n` in the tree. `n` is odd, and each node has a distinct value from `1` to `n`.

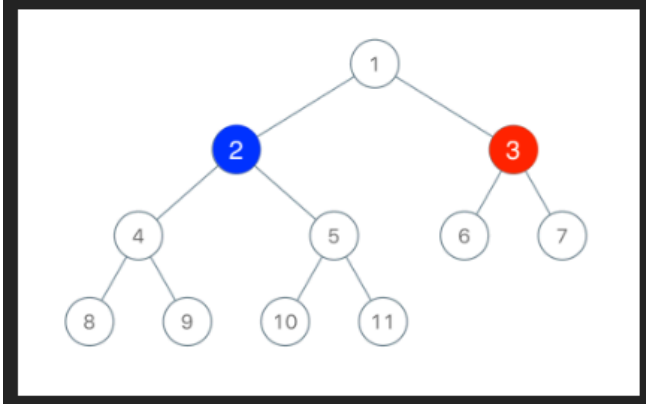
Initially, the first player names a value `x` with `1 <= x <= n`, and the second player names a value `y` with `1 <= y <= n` and `y != x`. The first player colors the node with value `x` red, and the second player colors the node with value `y` blue.

Then, the players take turns starting with the first player. In each turn, that player chooses a node of their color (red if player 1, blue if player 2) and colors an **uncolored** neighbor of the chosen node (either the left child, right child, or parent of the chosen node.)

If (and only if) a player cannot choose such a node in this way, they must pass their turn. If both players pass their turn, the game ends, and the winner is the player that colored more nodes.

You are the second player. If it is possible to choose such a `y` to ensure you win the game, return `true`. If it is not possible, return `false`.

Example 1:




```
Input: root = [1,2,3,4,5,6,7,8,9,10,11], n = 11, x = 3
Output: true
Explanation: The second player can choose the node with value 2.
```

Example 2:

```
Input: root = [1,2,3], n = 3, x = 1
Output: false
```

Binary Search Tree

1. Missing Number

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

Example 1:

Input: nums = [3,0,1]
Output: 2
Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.

Example 2:

Input: nums = [0,1]
Output: 2
Explanation: n = 2 since there are 2 numbers, so all numbers are in the range [0,2]. 2 is the missing number in the range since it does not appear in nums.

Example 3:

Input: nums = [9,6,4,2,3,5,7,0,1]
Output: 8
Explanation: n = 9 since there are 9 numbers, so all numbers are in the range [0,9]. 8 is the missing number in the range since it does not appear in nums.

Constraints:

- n == nums.length
- 1 <= n <= 10^4
- 0 <= nums[i] <= n
- All the numbers of nums are **unique**.

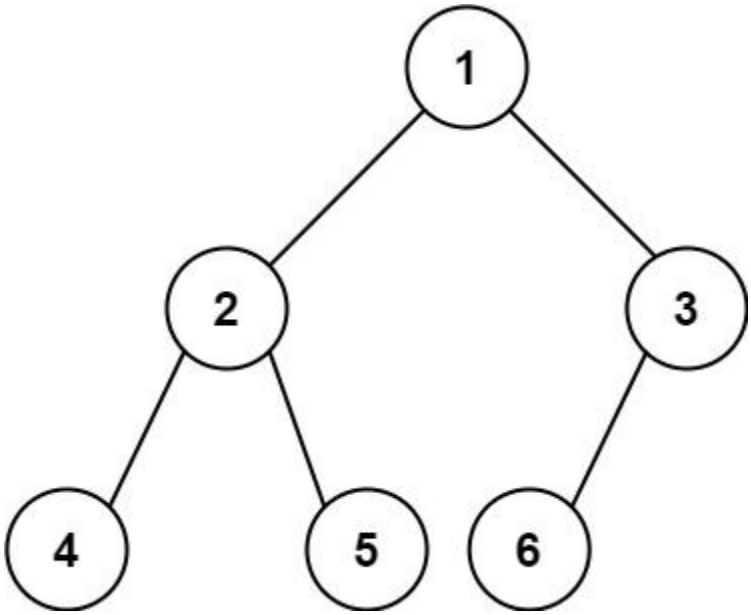
2. Count Complete Tree Nodes

Given the root of a **complete** binary tree, return the number of the nodes in the tree.

According to **Wikipedia**, every level, except possibly the last, is completely filled in a complete binary tree, and all nodes in the last level are as far left as possible. It can have between 1 and 2^h nodes inclusive at the last level h.

Design an algorithm that runs in less than O(n) time complexity.

Example 1:



Input: root = [1,2,3,4,5,6]
Output: 6

Example 2:

Input: root = []
Output: 0

Example 3:

Input: root = [1]
Output: 1

Constraints:

- The number of nodes in the tree is in the range [0, $5 * 10^4$].
- 0 <= Node.val <= $5 * 10^4$
- The tree is guaranteed to be **complete**.

3. Sqrt(x)

Given a non-negative integer x, return *the square root of* x *rounded down to the nearest integer*. The returned integer should be **non-negative** as well.

You **must not use** any built-in exponent function or operator.

- For example, do not use pow(x, 0.5) in c++ or x ** 0.5 in python.

Example 1:

Input: x = 4
Output: 2
Explanation: The square root of 4 is 2, so we return 2.

Example 2:

Input: x = 8
Output: 2
Explanation: The square root of 8 is 2.82842..., and since we round it down to the nearest integer, 2 is returned.

Constraints:

- $0 <= x <= 2_{31} - 1$

4. Search Insert Position

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

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

Example 1:

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

Example 2:

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

Example 3:

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

Constraints:

- 1 <= nums.length <= 10^4
- -10^4 <= nums[i] <= 10^4
- nums contains **distinct** values sorted in **ascending** order.
- -10^4 <= target <= 10^4

5. Best Sightseeing Pair

You are given an integer array values where values[i] represents the value of the i^{th} sightseeing spot. Two sightseeing spots i and j have a **distance** j - i between them.

The score of a pair (i < j) of sightseeing spots is values[i] + values[j] + i - j: the sum of the values of the sightseeing spots, minus the distance between them.

Return *the maximum score of a pair of sightseeing spots*.

Example 1:

Input: values = [8,1,5,2,6]
Output: 11
Explanation: i = 0, j = 2, values[i] + values[j] + i - j = 8 + 5 + 0 - 2 = 11

Example 2:

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

Constraints:

- 2 <= values.length <= 5 * 10^4
- 1 <= values[i] <= 1000

Heap

1. 1046. Last Stone Weight

You are given an array of integers stones where stones[i] is the weight of the i^{th} stone.

We are playing a game with the stones. On each turn, we choose the **heaviest two stones** and smash them together. Suppose the heaviest two stones have weights x and y with x <= y. The result of this smash is:

- If x == y, both stones are destroyed, and
- If x != y, the stone of weight x is destroyed, and the stone of weight y has new weight y - x.

At the end of the game, there is **at most one** stone left.

Return *the weight of the last remaining stone*. If there are no stones left, return 0.

Example 1:

Input: stones = [2,7,4,1,8,1]
Output: 1
Explanation:
We combine 7 and 8 to get 1 so the array converts to [2,4,1,1,1] then,
we combine 2 and 4 to get 2 so the array converts to [2,1,1,1] then,
we combine 2 and 1 to get 1 so the array converts to [1,1,1] then,
we combine 1 and 1 to get 0 so the array converts to [1] then that's the value of the last stone.

Example 2:

Input: stones = [1]
Output: 1

Constraints:

- 1 <= stones.length <= 30
- 1 <= stones[i] <= 1000

2. The K Weakest Rows in a Matrix

You are given an m x n binary matrix mat of 1's (representing soldiers) and 0's (representing civilians). The soldiers are positioned **in front** of the civilians. That is, all the 1's will appear to the **left** of all the 0's in each row.

A row i is **weaker** than a row j if one of the following is true:

- The number of soldiers in row i is less than the number of soldiers in row j.
- Both rows have the same number of soldiers and i < j.

Return *the indices of the* k ***weakest*** *rows in the matrix ordered from weakest to strongest*.

Example 1:

Input: mat =
[[1,1,0,0,0],
[1,1,1,1,0],
[1,0,0,0,0],
[1,1,0,0,0],
[1,1,1,1,1]],
k = 3
Output: [2,0,3]
Explanation:
The number of soldiers in each row is:
- Row 0: 2
- Row 1: 4
- Row 2: 1
- Row 3: 2
- Row 4: 5
The rows ordered from weakest to strongest are [2,0,3,1,4].

Example 2:

Input: mat =
[[1,0,0,0],
[1,1,1,1],
[1,0,0,0],
[1,0,0,0]],
k = 2
Output: [0,2]
Explanation:
The number of soldiers in each row is:

- Row 0: 1
- Row 1: 4
- Row 2: 1
- Row 3: 1

The rows ordered from weakest to strongest are [0,2,3,1].

Constraints:

- m == mat.length
- n == mat[i].length
- 2 <= n, m <= 100
- 1 <= k <= m
- matrix[i][j] is either 0 or 1.

3. Maximum Product of Two Elements in an Array

Given the array of integers nums, you will choose two different indices i and j of that array. *Return the maximum value of* (nums[i]-1)*(nums[j]-1).

Example 1:

Input: nums = [3,4,5,2]
Output: 12
Explanation: If you choose the indices i=1 and j=2 (indexed from 0), you will get the maximum value, that is, (nums[1]-1)*(nums[2]-1) = (4-1)*(5-1) = 3*4 = 12.

Example 2:

Input: nums = [1,5,4,5]
Output: 16
Explanation: Choosing the indices i=1 and j=3 (indexed from 0), you will get the maximum value of (5-1)*(5-1) = 16.

Example 3:

Input: nums = [3,7]
Output: 12

Constraints:

- 2 <= nums.length <= 500

- 1 <= nums[i] <= 10^3

4. Take Gifts From the Richest Pile

You are given an integer array gifts denoting the number of gifts in various piles. Every second, you do the following:

- Choose the pile with the maximum number of gifts.
- If there is more than one pile with the maximum number of gifts, choose any.
- Reduce the number of gifts in the pile to the floor of the square root of the original number of gifts in the pile.

Return *the number of gifts remaining after* k *seconds.*

Example 1:

Input: gifts = [25,64,9,4,100], k = 4
Output: 29
Explanation:
The gifts are taken in the following way:
- In the first second, the last pile is chosen and 10 gifts are left behind.
- Then the second pile is chosen and 8 gifts are left behind.
- After that the first pile is chosen and 5 gifts are left behind.
- Finally, the last pile is chosen again and 3 gifts are left behind.
The final remaining gifts are [5,8,9,4,3], so the total number of gifts remaining is 29.

Example 2:

Input: gifts = [1,1,1,1], k = 4
Output: 4
Explanation:
In this case, regardless which pile you choose, you have to leave behind 1 gift in each pile.
That is, you can't take any pile with you.
So, the total gifts remaining are 4.

Constraints:

- 1 <= gifts.length <= 10^3
- 1 <= gifts[i] <= 10^9
- 1 <= k <= 10^3

5. Final Array State After K Multiplication Operations I

You are given an integer array nums, an integer k, and an integer multiplier.

You need to perform k operations on nums. In each operation:

- Find the **minimum** value x in nums. If there are multiple occurrences of the minimum value, select the one that appears **first**.
- Replace the selected minimum value x with x * multiplier.

Return an integer array denoting the *final state* of nums after performing all k operations.

Example 1:

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

Output: [8,4,6,5,6]

Explanation:

Operation	Result
After operation 1	[2, 2, 3, 5, 6]
After operation 2	[4, 2, 3, 5, 6]
After operation 3	[4, 4, 3, 5, 6]
After operation 4	[4, 4, 6, 5, 6]
After operation 5	[8, 4, 6, 5, 6]

Example 2:

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

Output: [16,8]

Explanation:

Operation	Result
After operation 1	[4, 2]
After operation 2	[4, 8]
After operation 3	[16, 8]

Constraints:

- 1 <= nums.length <= 100
- 1 <= nums[i] <= 100
- 1 <= k <= 10
- 1 <= multiplier <= 5

Graph

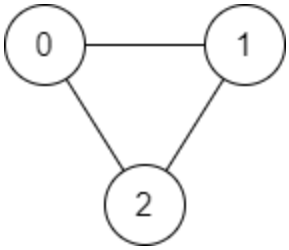
1. Find if Path Exists in Graph

There is a **bi-directional** graph with n vertices, where each vertex is labeled from 0 to n - 1 (**inclusive**). The edges in the graph are represented as a 2D integer array edges, where each edges[i] = [u_i, v_i] denotes a bi-directional edge between vertex u_i and vertex v_i. Every vertex pair is connected by **at most one** edge, and no vertex has an edge to itself.

You want to determine if there is a **valid path** that exists from vertex source to vertex destination.

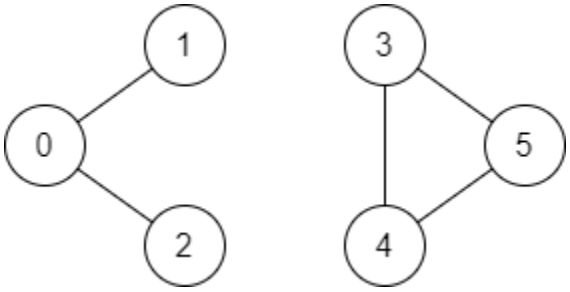
Given edges and the integers n, source, and destination, return true *if there is a **valid path** from* source *to* destination*, or* false *otherwise.*

Example 1:



Input: n = 3, edges = [[0,1],[1,2],[2,0]], source = 0, destination = 2
Output: true
Explanation: There are two paths from vertex 0 to vertex 2:
- 0 → 1 → 2
- 0 → 2

Example 2:



Input: n = 6, edges = [[0,1],[0,2],[3,5],[5,4],[4,3]], source = 0, destination = 5
Output: false
Explanation: There is no path from vertex 0 to vertex 5.

Constraints:

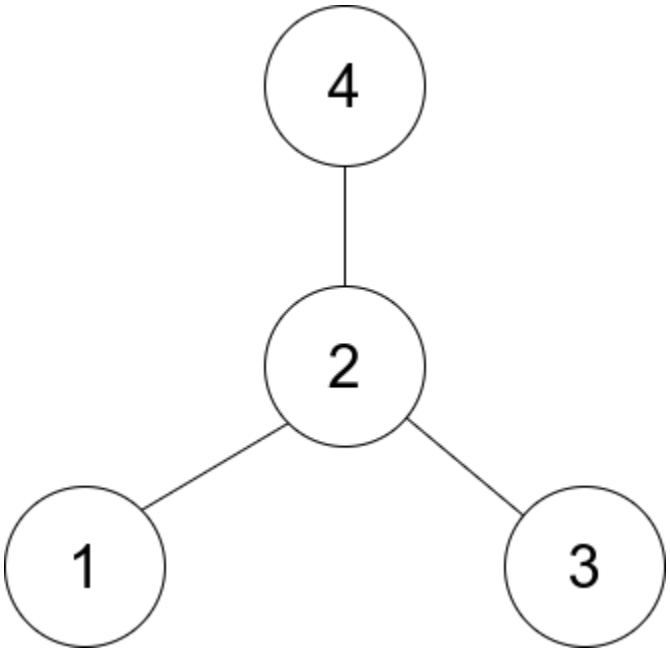
- $1 <= n <= 2 * 10^5$
- $0 <= edges.length <= 2 * 10^5$
- edges[i].length == 2
- $0 <= u_i, v_i <= n - 1$
- $u_i != v_i$
- 0 <= source, destination <= n - 1
- There are no duplicate edges.
- There are no self edges.

2. Find Center of Star Graph

There is an undirected **star** graph consisting of n nodes labeled from 1 to n. A star graph is a graph where there is one **center** node and **exactly** n - 1 edges that connect the center node with every other node.

You are given a 2D integer array edges where each edges[i] = [u_i, v_i] indicates that there is an edge between the nodes u_i and v_i. Return the center of the given star graph.

Example 1:



Input: edges = [[1,2],[2,3],[4,2]]
Output: 2
Explanation: As shown in the figure above, node 2 is connected to every other node, so 2 is the center.

Example 2:

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

Constraints:

- 3 <= n <= 10^5
- edges.length == n - 1
- edges[i].length == 2
- 1 <= u_i, v_i <= n
- u_i != v_i
- The given edges represent a valid star graph.

3. Find the Town Judge

In a town, there are n people labeled from 1 to n. There is a rumor that one of these people is secretly the town judge.

If the town judge exists, then:

1. The town judge trusts nobody.
2. Everybody (except for the town judge) trusts the town judge.
3. There is exactly one person that satisfies properties **1** and **2**.

You are given an array trust where trust[i] = [a_i, b_i] representing that the person labeled a_i trusts the person labeled b_i. If a trust relationship does not exist in trust array, then such a trust relationship does not exist.

Return *the label of the town judge if the town judge exists and can be identified, or return* -1 *otherwise*.

Example 1:

Input: n = 2, trust = [[1,2]]
Output: 2

Example 2:

Input: n = 3, trust = [[1,3],[2,3]]
Output: 3

Example 3:

Input: n = 3, trust = [[1,3],[2,3],[3,1]]
Output: -1

Constraints:

- 1 <= n <= 1000
- 0 <= trust.length <= 10^4
- trust[i].length == 2
- All the pairs of trust are **unique**.
- a_i != b_i
- 1 <= a_i, b_i <= n

4. Shortest Distance After Road Addition Queries I

You are given an integer n and a 2D integer array queries.

There are n cities numbered from 0 to n - 1. Initially, there is a **unidirectional** road from city i to city i + 1 for all 0 <= i < n - 1.

queries[i] = [u_i, v_i] represents the addition of a new **unidirectional** road from city u_i to city v_i. After each query, you need to find the **length** of the **shortest path** from city 0 to city n - 1.

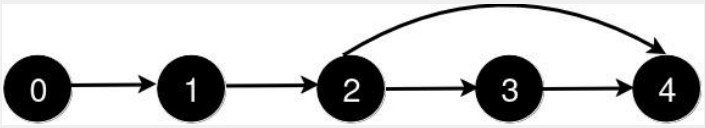
Return an array answer where for each i in the range [0, queries.length - 1], answer[i] is the *length of the shortest path* from city 0 to city n - 1 after processing the **first** i + 1 queries.

Example 1:

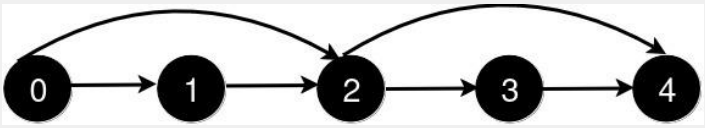
Input: n = 5, queries = [[2,4],[0,2],[0,4]]

Output: [3,2,1]

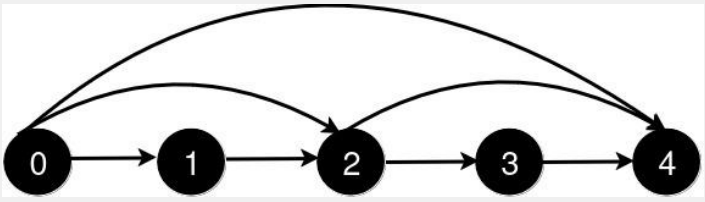
Explanation:



After the addition of the road from 2 to 4, the length of the shortest path from 0 to 4 is 3.



After the addition of the road from 0 to 2, the length of the shortest path from 0 to 4 is 2.



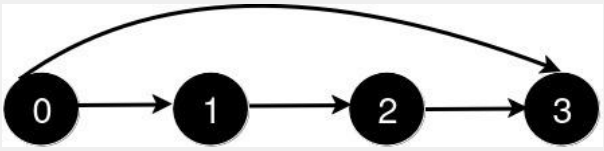
After the addition of the road from 0 to 4, the length of the shortest path from 0 to 4 is 1.

Example 2:

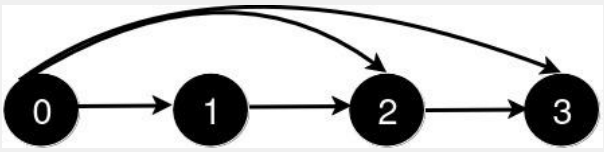
Input: n = 4, queries = [[0,3],[0,2]]

Output: [1,1]

Explanation:



After the addition of the road from 0 to 3, the length of the shortest path from 0 to 3 is 1.



After the addition of the road from 0 to 2, the length of the shortest path remains 1.

Constraints:

- 3 <= n <= 500
- 1 <= queries.length <= 500
- queries[i].length == 2
- 0 <= queries[i][0] < queries[i][1] < n
- 1 < queries[i][1] - queries[i][0]
- There are no repeated roads among the queries.

5. Most Profitable Path in a Tree

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

At every node i, there is a gate. You are also given an array of even integers amount, where amount[i] represents:

- the price needed to open the gate at node i, if amount[i] is negative, or,
- the cash reward obtained on opening the gate at node i, otherwise.

The game goes on as follows:

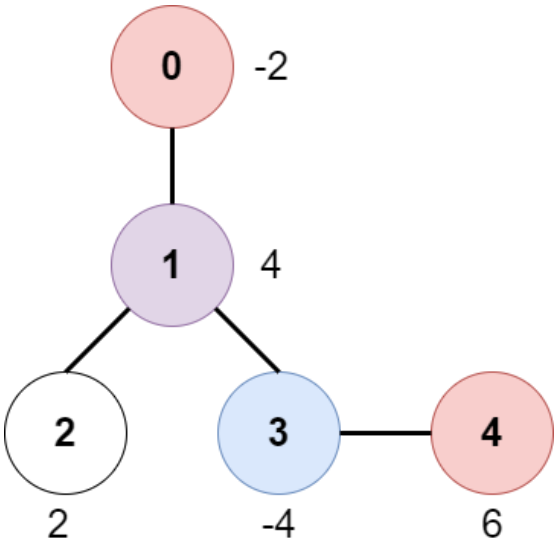
- Initially, Alice is at node 0 and Bob is at node bob.
- At every second, Alice and Bob **each** move to an adjacent node. Alice moves towards some **leaf node**, while Bob moves towards node 0.
- For **every** node along their path, Alice and Bob either spend money to open the gate at that node, or accept the reward. Note that:

- If the gate is **already open**, no price will be required, nor will there be any cash reward.
- If Alice and Bob reach the node **simultaneously**, they share the price/reward for opening the gate there. In other words, if the price to open the gate is c, then both Alice and Bob pay c / 2 each. Similarly, if the reward at the gate is c, both of them receive c / 2 each.

- If Alice reaches a leaf node, she stops moving. Similarly, if Bob reaches node 0, he stops moving. Note that these events are **independent** of each other.

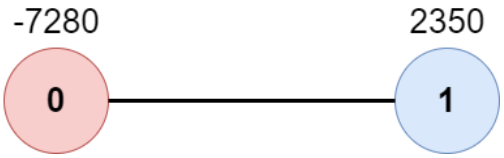
Return *the **maximum** net income Alice can have if she travels towards the optimal leaf node.*

Example 1:



Input: edges = [[0,1],[1,2],[1,3],[3,4]], bob = 3, amount = [-2,4,2,-4,6]
Output: 6
Explanation:
The above diagram represents the given tree. The game goes as follows:
- Alice is initially on node 0, Bob on node 3. They open the gates of their respective nodes.
  Alice's net income is now -2.
- Both Alice and Bob move to node 1.
  Since they reach here simultaneously, they open the gate together and share the reward.
  Alice's net income becomes -2 + (4 / 2) = 0.
- Alice moves on to node 3. Since Bob already opened its gate, Alice's income remains unchanged.
  Bob moves on to node 0, and stops moving.
- Alice moves on to node 4 and opens the gate there. Her net income becomes 0 + 6 = 6.
Now, neither Alice nor Bob can make any further moves, and the game ends.
It is not possible for Alice to get a higher net income.

Example 2:



Input: edges = [[0,1]], bob = 1, amount = [-7280,2350]
Output: -7280
Explanation:
Alice follows the path 0->1 whereas Bob follows the path 1->0.
Thus, Alice opens the gate at node 0 only. Hence, her net income is -7280.

Constraints:

- $2 <= n <= 10^5$
- edges.length == n - 1
- edges[i].length == 2
- $0 <= a_i, b_i < n$
- $a_i != b_i$
- edges represents a valid tree.
- 1 <= bob < n
- amount.length == n
- amount[i] is an **even** integer in the range $[-10^4, 10^4]$.