

# My LeetCode Submissions - @mohakchugh

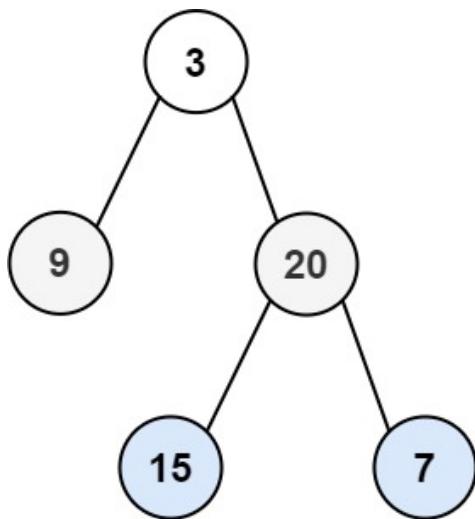
[Download PDF](#)[Follow @TheShubham99 on GitHub](#)[Star on GitHub](#)[View Source Code](#)

## [107 Binary Tree Level Order Traversal II \(link\)](#)

### Description

Given the root of a binary tree, return *the bottom-up level order traversal of its nodes' values*. (i.e., from left to right, level by level from leaf to root).

#### Example 1:



```
Input: root = [3,9,20,null,null,15,7]
Output: [[15,7],[9,20],[3]]
```

#### Example 2:

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

#### Example 3:

```
Input: root = []
Output: []
```

### Constraints:

- The number of nodes in the tree is in the range  $[0, 2000]$ .
- $-1000 \leq \text{Node.val} \leq 1000$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    from collections import deque
    def levelOrderBottom(self, root: Optional[TreeNode]) -> List[List[int]]:
        if not root:
            return []
        queue = deque([(root, 0)])
        levels = deque()
        while queue:
            current, current_level = queue.popleft()
            if len(levels) == current_level:
                levels.append([])
            levels[current_level].append(current.val)
            if current.left:
                queue.append((current.left, current_level+1))
            if current.right:
                queue.append((current.right, current_level+1))
        return list(reversed(levels))
```

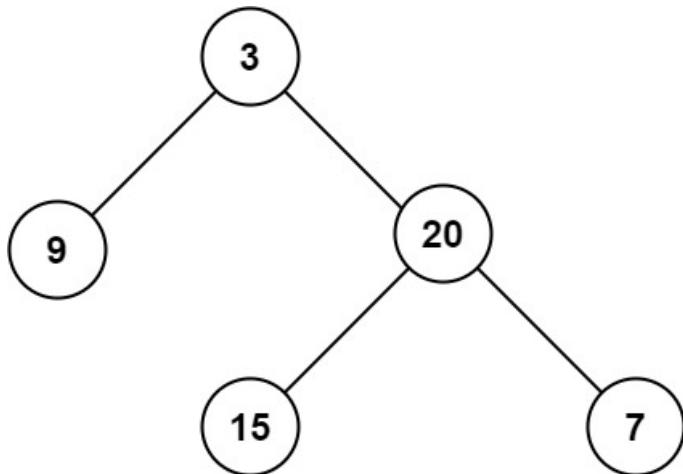
## [104 Maximum Depth of Binary Tree \(link\)](#)

### Description

Given the root of a binary tree, return *its maximum depth*.

A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.

#### Example 1:



```
Input: root = [3,9,20,null,null,15,7]  
Output: 3
```

#### Example 2:

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

### Constraints:

- The number of nodes in the tree is in the range  $[0, 10^4]$ .
- $-100 \leq \text{Node.val} \leq 100$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

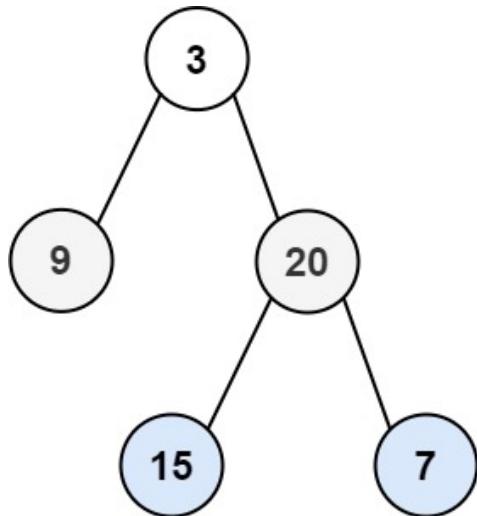
```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def maxDepth(self, root: Optional[TreeNode]) -> int:
        def traverse_max_depth(root, depth):
            if not root:
                return depth
            return max(traverse_max_depth(root.left, depth+1), traverse_max_depth(root.right, depth+1))
        return traverse_max_depth(root, 0)
```

## [103 Binary Tree Zigzag Level Order Traversal \(link\)](#)

### Description

Given the root of a binary tree, return *the zigzag level order traversal of its nodes' values*. (i.e., from left to right, then right to left for the next level and alternate between).

#### Example 1:



```
Input: root = [3,9,20,null,null,15,7]
Output: [[3],[20,9],[15,7]]
```

#### Example 2:

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

#### Example 3:

```
Input: root = []
Output: []
```

#### Constraints:

- The number of nodes in the tree is in the range  $[0, 2000]$ .
- $-100 \leq \text{Node.val} \leq 100$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    from collections import deque
    def zigzagLevelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:
        if not root:
            return []

        queue = deque([(root, 0)])
        levels = []

        while queue:
            # visit the current node
            current_node, current_level = queue.popleft()

            # make space for current level in result array
            if len(levels) == current_level:
                levels.append([])

            # add the current_node in result array
            levels[current_level].append(current_node.val)

            # if left child
            if current_node.left:
                queue.append((current_node.left, current_level + 1))

            # if right child
            if current_node.right:
                queue.append((current_node.right, current_level + 1))

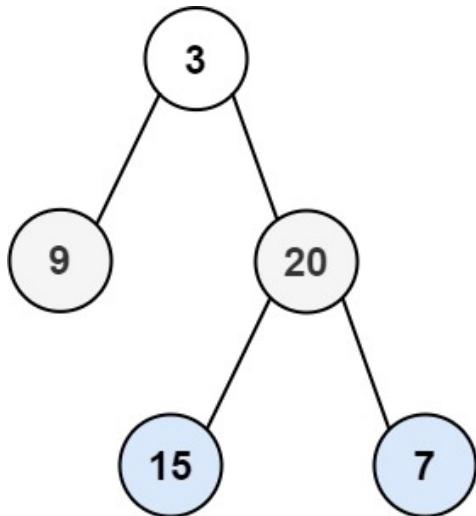
        # make it zig - zag
        for current_level in range(len(levels)):
            if current_level % 2 == 1:
                levels[current_level] = list(reversed(levels[current_level]))
        return levels
```

## [102 Binary Tree Level Order Traversal \(link\)](#)

### Description

Given the root of a binary tree, return *the level order traversal of its nodes' values*. (i.e., from left to right, level by level).

#### Example 1:



```
Input: root = [3,9,20,null,null,15,7]
Output: [[3],[9,20],[15,7]]
```

#### Example 2:

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

#### Example 3:

```
Input: root = []
Output: []
```

#### Constraints:

- The number of nodes in the tree is in the range  $[0, 2000]$ .
- $-1000 \leq \text{Node.val} \leq 1000$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

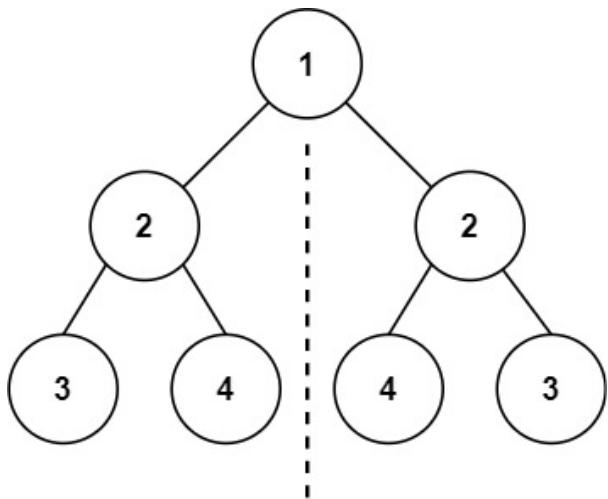
```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    from collections import deque
    def levelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:
        if not root:
            return []
        levels = []
        queue = deque([(root, 0)])
        while queue:
            current_node, current_level = queue.popleft()
            if len(levels) == current_level:
                levels.append([])
            levels[current_level].append(current_node.val)
            if current_node.left:
                queue.append((current_node.left, current_level + 1))
            if current_node.right:
                queue.append((current_node.right, current_level + 1))
        return levels
```

## [101 Symmetric Tree \(link\)](#)

### Description

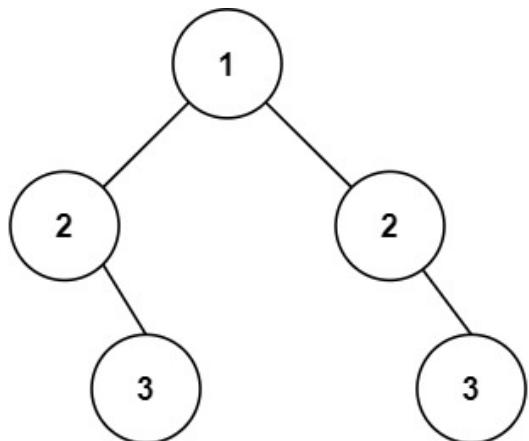
Given the root of a binary tree, *check whether it is a mirror of itself* (i.e., symmetric around its center).

#### Example 1:



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

#### Example 2:



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

#### Constraints:

- The number of nodes in the tree is in the range [1, 1000].
- $-100 \leq \text{Node.val} \leq 100$

**Follow up:** Could you solve it both recursively and iteratively?

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def isSymmetric(self, root: Optional[TreeNode]) -> bool:
        if not root:
            return True
        def breadth_first_search(left, right):
            if (not left and right) or (not right and left):
                return False
            elif not left and not right:
                return True
            elif (left.val != right.val):
                return False
            return breadth_first_search(left.right, right.left) and breadth_first_search(left.left, right.right)
        return breadth_first_search(root.left, root.right)
```

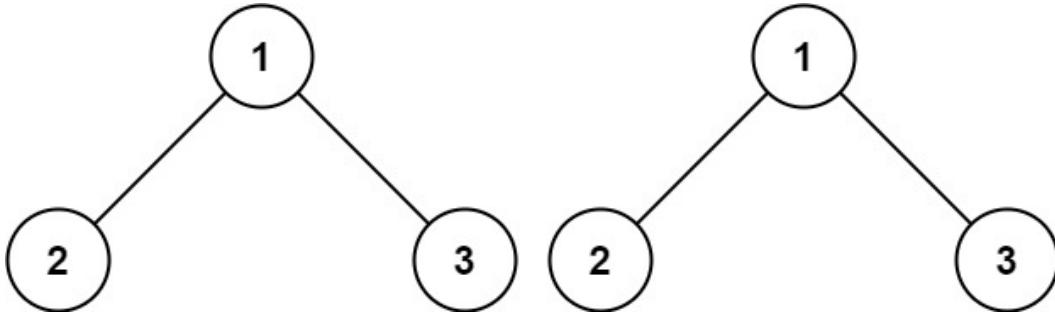
## 100 Same Tree (link)

### Description

Given the roots of two binary trees  $p$  and  $q$ , write a function to check if they are the same or not.

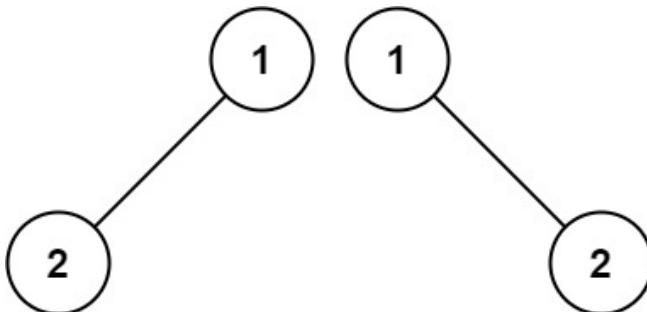
Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

#### Example 1:



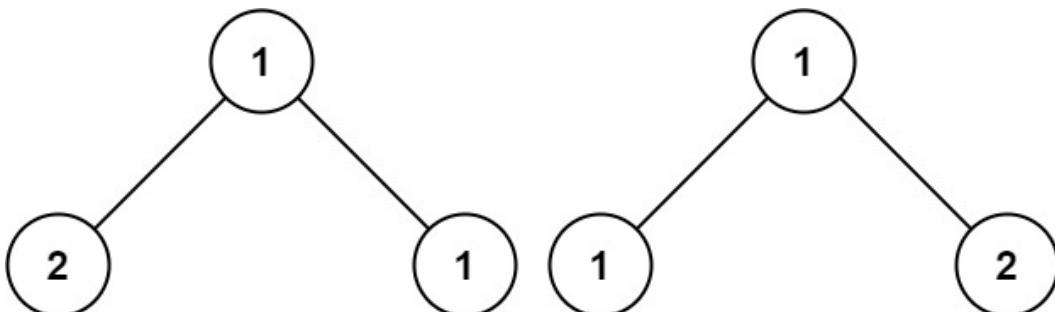
**Input:**  $p = [1,2,3]$ ,  $q = [1,2,3]$   
**Output:** true

#### Example 2:



**Input:**  $p = [1,2]$ ,  $q = [1,null,2]$   
**Output:** false

#### Example 3:



**Input:**  $p = [1,2,1]$ ,  $q = [1,1,2]$   
**Output:** false

#### Constraints:

- The number of nodes in both trees is in the range  $[0, 100]$ .

- $-10^4 \leq \text{Node.val} \leq 10^4$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def isSameTree(self, p: Optional[TreeNode], q: Optional[TreeNode]) -> bool:
        def breadth_first_search(p: Optional[TreeNode], q: Optional[TreeNode]):
            if (not p and q) or (not q and p):
                return False
            elif (not p and not q):
                return True
            elif p.val != q.val:
                return False

            return breadth_first_search(p.left, q.left) and breadth_first_search(p.right, q.right)
        return breadth_first_search(p, q)
```

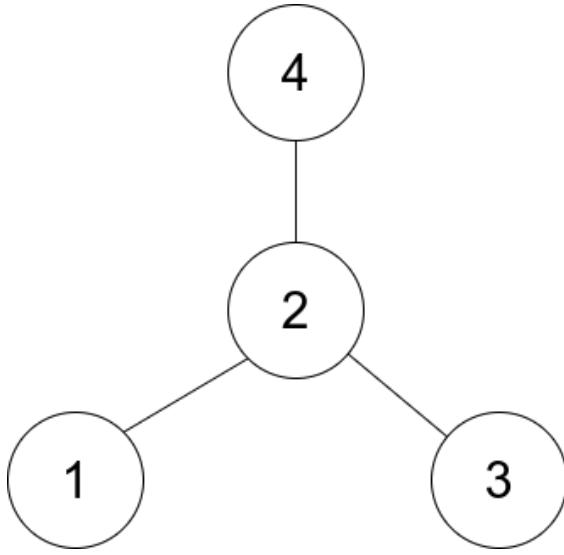
## [1916 Find Center of Star Graph \(link\)](#)

### Description

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] = [ui, vi]` 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 of the star graph.

#### Example 2:

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

**Output:** 1

#### Constraints:

- $3 \leq n \leq 10^5$
- $\text{edges.length} == n - 1$
- $\text{edges}[i].length == 2$
- $1 \leq u_i, v_i \leq n$
- $u_i \neq v_i$
- The given edges represent a valid star graph.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def findCenter(self, edges: List[List[int]]) -> int:
        # Concept: node with max indegree = center

        indegree = collections.defaultdict(int)
        for node1, node2 in edges:
            indegree[node1] += 1
            indegree[node2] += 1

        maxindegree = float('-inf')
        maxindegreeNode = None
        for node in indegree.keys():
            if indegree[node] > maxindegree:
                maxindegree = indegree[node]
                maxindegreeNode = node

        return maxindegreeNode
```

## [1039 Find the Town Judge \(link\)](#)

### Description

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] = [ai, bi]` 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:

- $1 \leq n \leq 1000$
- $0 \leq \text{trust.length} \leq 10^4$
- $\text{trust}[i].length == 2$
- All the pairs of trust are **unique**.
- $a_i \neq b_i$
- $1 \leq a_i, b_i \leq n$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def findJudge(self, n: int, trust: List[List[int]]) -> int:
        if n == 1 and not trust:
            return 1

        graph = collections.defaultdict(list)
        indegree = collections.defaultdict(int)
        outdegree = collections.defaultdict(int)
        nodes = set()

        for doestrust, gettrust in trust:
            graph[gettrust].append(doestrust)
            indegree[gettrust] += 1
            outdegree[doestrust] += 1
            nodes.add(gettrust)
            nodes.add(doestrust)

        for node in nodes:
            if indegree[node] == n-1 and outdegree[node] == 0:
                return node

        return -1
```

## [684 Redundant Connection \(link\)](#)

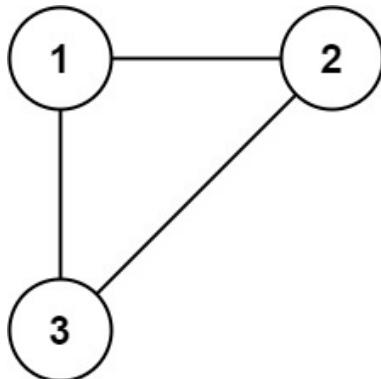
### Description

In this problem, a tree is an **undirected graph** that is connected and has no cycles.

You are given a graph that started as a tree with  $n$  nodes labeled from 1 to  $n$ , with one additional edge added. The added edge has two **different** vertices chosen from 1 to  $n$ , and was not an edge that already existed. The graph is represented as an array `edges` of length  $n$  where `edges[i] = [ai, bi]` indicates that there is an edge between nodes  $a<sub>i</sub>$  and  $b<sub>i</sub>$  in the graph.

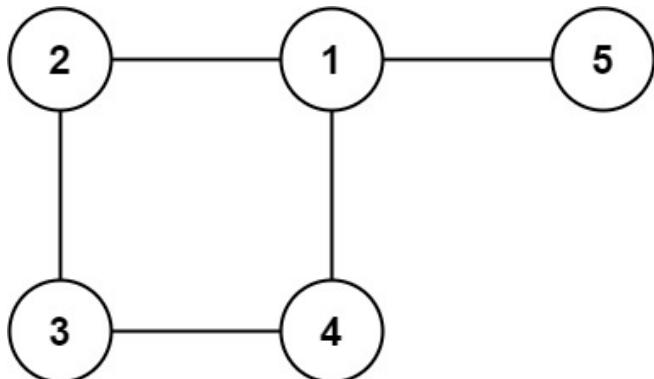
Return *an edge that can be removed so that the resulting graph is a tree of  $n$  nodes*. If there are multiple answers, return the answer that occurs last in the input.

#### Example 1:



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

#### Example 2:



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

#### Constraints:

- $n == \text{edges.length}$
- $3 \leq n \leq 1000$
- $\text{edges}[i].length == 2$
- $1 \leq a_i < b_i \leq \text{edges.length}$
- $a_i \neq b_i$
- There are no repeated edges.
- The given graph is connected.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def findRedundantConnection(self, edges: List[List[int]]) -> List[int]:
        n = len(edges)
        parents = {i: i for i in range(1, n+1)} # parent[x] = x
        rank = {i: 1 for i in range(1, n+1)} # rank[x] = 1

        def FIND(node: int):
            if parents[node] != node:
                parents[node] = FIND(parents[node]) # Path compression
            return parents[node]

        def UNION(node1, node2):
            parent1, parent2 = FIND(node1), FIND(node2)
            if parent1 == parent2:
                return False # Cycle detected
            else:
                if rank[parent1] > rank[parent2]:
                    parents[parent2] = parent1
                elif rank[parent1] < rank[parent2]:
                    parents[parent1] = parent2
                else:
                    parents[parent2] = parent1
                    rank[parent1] += 1

            return True

        for source, destination in edges:
            if UNION(source, destination) == False:
                return [source, destination]
```

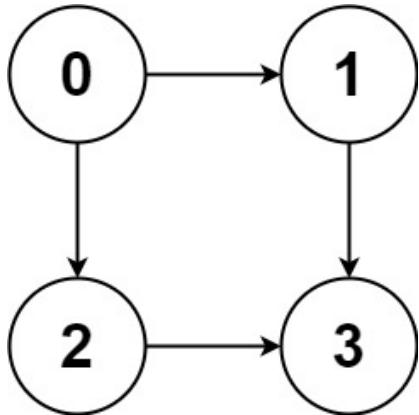
## [813 All Paths From Source to Target \(link\)](#)

### Description

Given a directed acyclic graph (**DAG**) of  $n$  nodes labeled from  $0$  to  $n - 1$ , find all possible paths from node  $0$  to node  $n - 1$  and return them in **any order**.

The graph is given as follows:  $\text{graph}[i]$  is a list of all nodes you can visit from node  $i$  (i.e., there is a directed edge from node  $i$  to node  $\text{graph}[i][j]$ ).

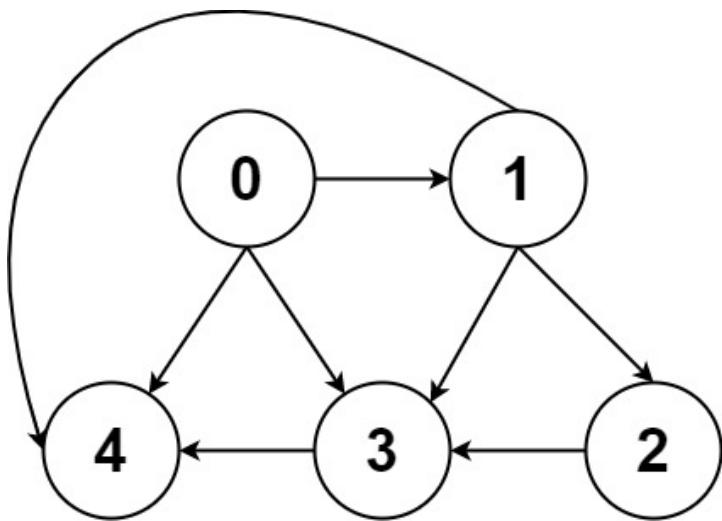
#### Example 1:



**Input:**  $\text{graph} = [[1,2],[3],[3],[]]$   
**Output:**  $[[0,1,3],[0,2,3]]$

**Explanation:** There are two paths:  $0 \rightarrow 1 \rightarrow 3$  and  $0 \rightarrow 2 \rightarrow 3$ .

#### Example 2:



**Input:**  $\text{graph} = [[4,3,1],[3,2,4],[3],[4],[]]$   
**Output:**  $[[0,4],[0,3,4],[0,1,3,4],[0,1,2,3,4],[0,1,4]]$

#### Constraints:

- $n == \text{graph.length}$
- $2 \leq n \leq 15$
- $0 \leq \text{graph}[i][j] < n$
- $\text{graph}[i][j] \neq i$  (i.e., there will be no self-loops).
- All the elements of  $\text{graph}[i]$  are **unique**.
- The input graph is **guaranteed** to be a **DAG**.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def allPathsSourceTarget(self, graph: List[List[int]]) -> List[List[int]]:
        result = []
        N = len(graph)

        def dfs(node: int, path: List[int]):
            path.append(node)

            if node == N - 1:
                result.append(path[:]) # copy the path
            else:
                for neighbor in graph[node]:
                    dfs(neighbor, path)

            path.pop() # backtrack

        dfs(0, [])
        return result

# # initialize variables
# N = len(graph_input)
# graph = collections.defaultdict(list)
# result = []

# # build the graph
# for node in range(N):
#     for neighbors in graph_input[node]:
#         graph[node].append(neighbors)

# # traverse the graph
# def get_path(current_node: int, current_path: list):
#     # Visit current node
#     current_path.append(current_node)

#     # If current node is Target, Add to final results
#     if current_node == (N - 1):
#         result.append(current_path[:])

#     # Traverse all neighbours
#     for neighbor in graph[current_node]:
#         get_path(neighbor, current_path)

# get_path(0, [])
# return result
```

## 1442 Number of Operations to Make Network Connected (link)

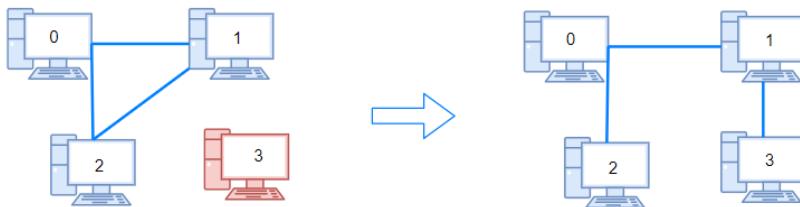
### Description

There are  $n$  computers numbered from  $0$  to  $n - 1$  connected by ethernet cables forming a network where  $\text{connections}[i] = [a_i, b_i]$  represents a connection between computers  $a_i$  and  $b_i$ . Any computer can reach any other computer directly or indirectly through the network.

You are given an initial computer network connections. You can extract certain cables between two directly connected computers, and place them between any pair of disconnected computers to make them directly connected.

Return *the minimum number of times you need to do this in order to make all the computers connected*. If it is not possible, return  $-1$ .

### Example 1:

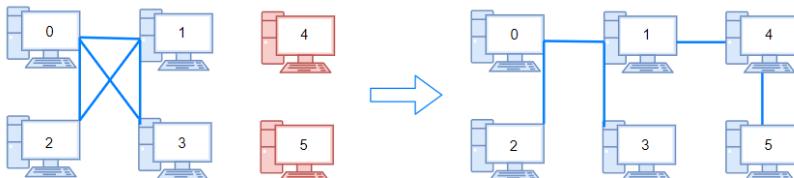


**Input:**  $n = 4$ ,  $\text{connections} = [[0,1], [0,2], [1,2]]$

**Output:** 1

**Explanation:** Remove cable between computer 1 and 2 and place between computers 1 and 3.

### Example 2:



**Input:**  $n = 6$ ,  $\text{connections} = [[0,1], [0,2], [0,3], [1,2], [1,3]]$

**Output:** 2

### Example 3:

**Input:**  $n = 6$ ,  $\text{connections} = [[0,1], [0,2], [0,3], [1,2]]$

**Output:** -1

**Explanation:** There are not enough cables.

### Constraints:

- $1 \leq n \leq 10^5$
- $1 \leq \text{connections.length} \leq \min(n * (n - 1) / 2, 10^5)$
- $\text{connections}[i].length == 2$
- $0 \leq a_i, b_i < n$
- $a_i \neq b_i$
- There are no repeated connections.
- No two computers are connected by more than one cable.

(scroll down for solution)



## Solution

Language: python3

Status: Accepted

```
class Solution:
    def makeConnected(self, n: int, connections: List[List[int]]) -> int:
        # Atleast N-1 cables are required for fully connected graph
        if len(connections) < (n-1):
            return -1

        # initializing variables
        graph = collections.defaultdict(list)
        indegree = collections.defaultdict(int)
        queue = collections.deque()
        visited = set()

        # build graph and calculate indegree
        for source, destination in connections:
            graph[source].append(destination)
            graph[destination].append(source)

            indegree[source] += 1
            indegree[destination] += 1

        # get all islands / components
        def traverse_connected_nodes(node: int):
            for neighbor in graph[node]:
                if neighbor not in visited:
                    visited.add(neighbor)
                    traverse_connected_nodes(neighbor)

        components = 0
        for node in range(n):
            if node not in visited:
                visited.add(node)
                traverse_connected_nodes(node)
                components += 1

        # This is the number of cable changes
        return components - 1
```

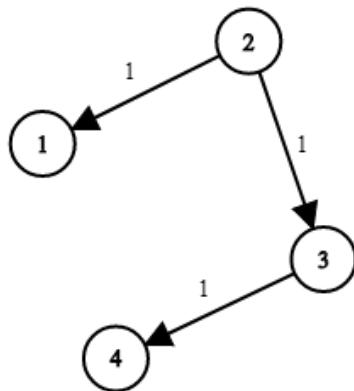
## [744 Network Delay Time \(link\)](#)

### Description

You are given a network of  $n$  nodes, labeled from 1 to  $n$ . You are also given  $\text{times}$ , a list of travel times as directed edges  $\text{times}[i] = (u_i, v_i, w_i)$ , where  $u_i$  is the source node,  $v_i$  is the target node, and  $w_i$  is the time it takes for a signal to travel from source to target.

We will send a signal from a given node  $k$ . Return *the minimum time it takes for all the  $n$  nodes to receive the signal*. If it is impossible for all the  $n$  nodes to receive the signal, return  $-1$ .

#### Example 1:



**Input:**  $\text{times} = [[2,1,1], [2,3,1], [3,4,1]]$ ,  $n = 4$ ,  $k = 2$   
**Output:** 2

#### Example 2:

**Input:**  $\text{times} = [[1,2,1]]$ ,  $n = 2$ ,  $k = 1$   
**Output:** 1

#### Example 3:

**Input:**  $\text{times} = [[1,2,1]]$ ,  $n = 2$ ,  $k = 2$   
**Output:** -1

#### Constraints:

- $1 \leq k \leq n \leq 100$
- $1 \leq \text{times.length} \leq 6000$
- $\text{times}[i].length == 3$
- $1 \leq u_i, v_i \leq n$
- $u_i \neq v_i$
- $0 \leq w_i \leq 100$
- All the pairs  $(u_i, v_i)$  are **unique**. (i.e., no multiple edges.)

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def networkDelayTime(self, times: List[List[int]], n: int, k: int) -> int:
        min_times = [float('inf')] * (n+1)
        min_times[k] = 0
        min_times[0] = 0

        for iteration in range(n):
            current_time = min_times.copy()
            for source, destination, time in times:
                if min_times[source] + time < current_time[destination]:
                    current_time[destination] = min_times[source] + time
            # print(f'Iteration: {iteration}, time: {current_time}')
            min_times = current_time

        # print(min_times)
        return -1 if float('inf') == max(min_times) else max(min_times)
```

## [1325 Path with Maximum Probability](#) (link)

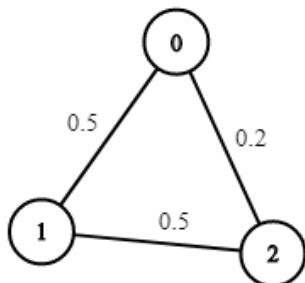
### Description

You are given an undirected weighted graph of  $n$  nodes (0-indexed), represented by an edge list where  $\text{edges}[i] = [a, b]$  is an undirected edge connecting the nodes  $a$  and  $b$  with a probability of success of traversing that edge  $\text{succProb}[i]$ .

Given two nodes  $\text{start}$  and  $\text{end}$ , find the path with the maximum probability of success to go from  $\text{start}$  to  $\text{end}$  and return its success probability.

If there is no path from  $\text{start}$  to  $\text{end}$ , **return 0**. Your answer will be accepted if it differs from the correct answer by at most **1e-5**.

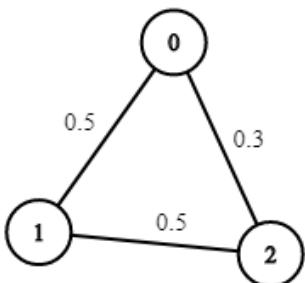
#### Example 1:



**Input:**  $n = 3$ ,  $\text{edges} = [[0,1],[1,2],[0,2]]$ ,  $\text{succProb} = [0.5,0.5,0.2]$ ,  $\text{start} = 0$ ,  $\text{end} = 2$   
**Output:**  $0.25000$

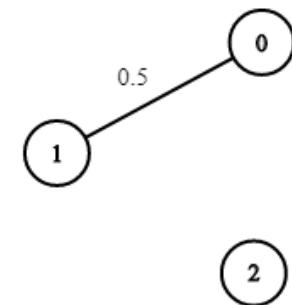
**Explanation:** There are two paths from start to end, one having a probability of success =  $0.25$  and the other  $0.2$ .

#### Example 2:



**Input:**  $n = 3$ ,  $\text{edges} = [[0,1],[1,2],[0,2]]$ ,  $\text{succProb} = [0.5,0.5,0.3]$ ,  $\text{start} = 0$ ,  $\text{end} = 2$   
**Output:**  $0.30000$

#### Example 3:



**Input:** n = 3, edges = [[0,1]], succProb = [0.5], start = 0, end = 2

**Output:** 0.0000

**Explanation:** There is no path between 0 and 2.

### Constraints:

- $2 \leq n \leq 10^4$
- $0 \leq \text{start}, \text{end} < n$
- $\text{start} \neq \text{end}$
- $0 \leq a, b < n$
- $a \neq b$
- $0 \leq \text{succProb.length} == \text{edges.length} \leq 2*10^4$
- $0 \leq \text{succProb}[i] \leq 1$
- There is at most one edge between every two nodes.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```

class Solution:
    def maxProbability(self, n: int, edges: List[List[int]], succProb: List[float], start_node: int, end_node: int) -> float:
        # initialise variable
        graph, queue = collections.defaultdict(list), collections.deque([start_node])
        max_probabilities = [0.0] * n
        max_probabilities[start_node] = 1.0
        probability_updates = False

        # build the graph
        for i in range(len(edges)):
            source, destination = edges[i][0], edges[i][1]
            probability = succProb[i]
            graph[source].append([destination, probability])
            graph[destination].append([source, probability])

        # traverse the graph
        while queue:
            source = queue.popleft()
            for destination, neighbor_probability in graph[source]:
                if max_probabilities[source] * neighbor_probability > max_probabilities[destination]:
                    max_probabilities[destination] = max_probabilities[source] * neighbor_probability
                    queue.append(destination)

        return max_probabilities[end_node]

# # initialise variables
# max_probabilities = [0.0] * n
# max_probabilities[start_node] = 1
# probability_updates = False

# # Bellman ford iterates to n-1
# for node in range(n - 1):
#     current_probabilities = max_probabilities.copy()
#     for i in range(len(edges)):
#         source, destination = edges[i][0], edges[i][1]
#         success_probability = succProb[i]
#
#         if max_probabilities[source] * success_probability > current_probabilities[destination]:
#             current_probabilities[destination] = max_probabilities[source] * success_probability
#             probability_updates = True
#
#         if max_probabilities[destination] * success_probability > current_probabilities[source]:
#             current_probabilities[source] = max_probabilities[destination] * success_probability
#             probability_updates = True
#
#     if not probability_updates:
#         break
#
#     max_probabilities = current_probabilities

# return max_probabilities[end_node]

```

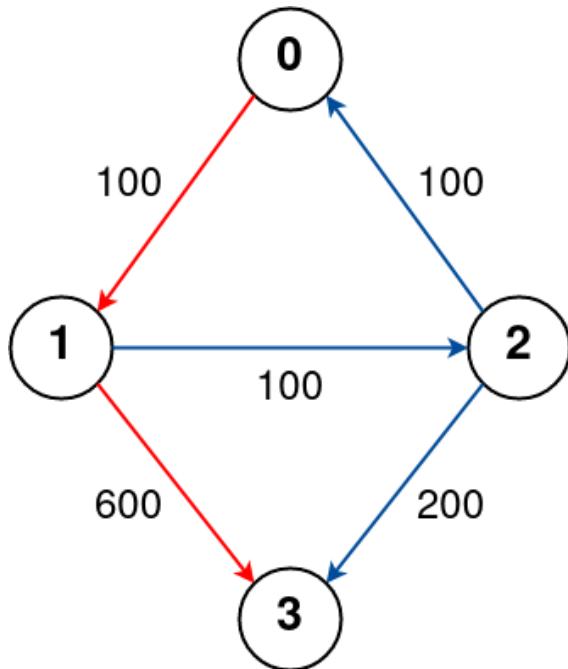
## 803 Cheapest Flights Within K Stops (link)

### Description

There are  $n$  cities connected by some number of flights. You are given an array `flights` where `flights[i] = [fromi, toi, pricei]` indicates that there is a flight from city `fromi` to city `toi` with cost `pricei`.

You are also given three integers `src`, `dst`, and `k`, return **the cheapest price** from `src` to `dst` with at most `k` stops. If there is no such route, return `-1`.

#### Example 1:



**Input:**  $n = 4$ , `flights` =  $[[0,1,100],[1,2,100],[2,0,100],[1,3,600],[2,3,200]]$ , `src` = 0, `dst` = 3,

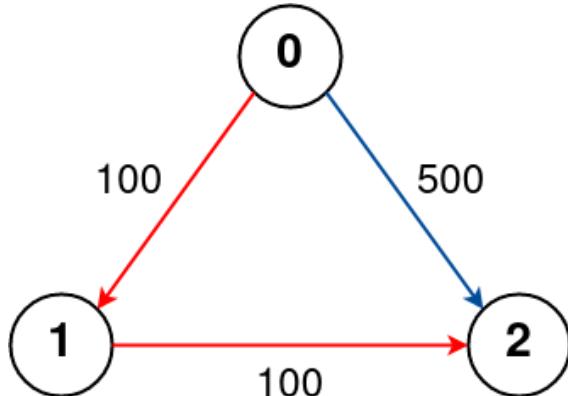
**Output:** 700

**Explanation:**

The graph is shown above.

The optimal path with at most 1 stop from city 0 to 3 is marked in red and has cost  $100 + 600 = 700$ . Note that the path through cities  $[0,1,2,3]$  is cheaper but is invalid because it uses 2 stops.

#### Example 2:



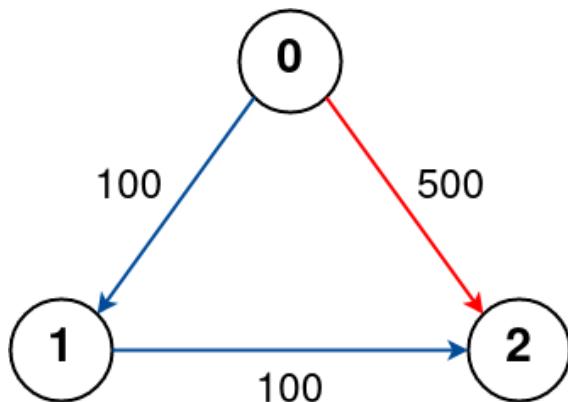
**Input:**  $n = 3$ , `flights` =  $[[0,1,100],[1,2,100],[0,2,500]]$ , `src` = 0, `dst` = 2, `k` = 1

**Output:** 200

**Explanation:**

The graph is shown above.

The optimal path with at most 1 stop from city 0 to 2 is marked in red and has cost  $100 + 100 = 200$ .

**Example 3:**

**Input:** n = 3, flights = [[0,1,100],[1,2,100],[0,2,500]], src = 0, dst = 2, k = 0

**Output:** 500

**Explanation:**

The graph is shown above.

The optimal path with no stops from city 0 to 2 is marked in red and has cost 500.

**Constraints:**

- $2 \leq n \leq 100$
- $0 \leq \text{flights.length} \leq (n * (n - 1) / 2)$
- $\text{flights}[i].length == 3$
- $0 \leq \text{from}_i, \text{to}_i < n$
- $\text{from}_i \neq \text{to}_i$
- $1 \leq \text{price}_i \leq 10^4$
- There will not be any multiple flights between two cities.
- $0 \leq \text{src}, \text{dst}, \text{k} < n$
- $\text{src} \neq \text{dst}$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def findCheapestPrice(self, n: int, flights: List[List[int]], src: int, dst: int, k: int) -> int:
        # starting_node -> BFS -> Every BFS -> Update min distance
        # nodes -> 0 ... n

        cheapest_costs = [float('inf')] * n
        cheapest_costs[src] = 0

        for i in range(k + 1):
            current_costs = cheapest_costs.copy()
            for source, destination, cost in flights:
                if cheapest_costs[source] + cost < current_costs[destination]: # tricky part: current_costs[destination] = cheapest_costs[source] + cost
                    current_costs[destination] = cheapest_costs[source] + cost
            cheapest_costs = current_costs

        return -1 if cheapest_costs[dst] == float('inf') else cheapest_costs[dst]
```

## [97 Interleaving String \(link\)](#)

### Description

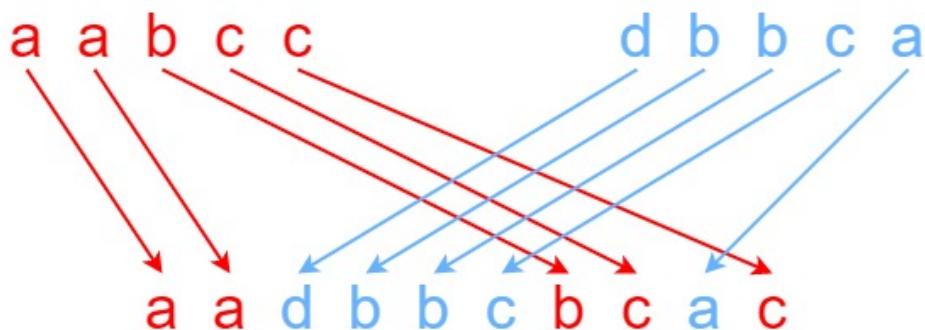
Given strings  $s_1$ ,  $s_2$ , and  $s_3$ , find whether  $s_3$  is formed by an **interleaving** of  $s_1$  and  $s_2$ .

An **interleaving** of two strings  $s$  and  $t$  is a configuration where  $s$  and  $t$  are divided into  $n$  and  $m$  substrings respectively, such that:

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

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

#### Example 1:



**Input:**  $s_1 = "aabcc"$ ,  $s_2 = "dbbca"$ ,  $s_3 = "aadbcbacac"$

**Output:** true

**Explanation:** One way to obtain  $s_3$  is:

Split  $s_1$  into  $s_1 = "aa" + "bc" + "c"$ , and  $s_2$  into  $s_2 = "dbbc" + "a"$ .

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

Since  $s_3$  can be obtained by interleaving  $s_1$  and  $s_2$ , we return true.

#### Example 2:

**Input:**  $s_1 = "aabcc"$ ,  $s_2 = "dbbca"$ ,  $s_3 = "aadbbbaccc"$

**Output:** false

**Explanation:** Notice how it is impossible to interleave  $s_2$  with any other string to obtain  $s_3$ .

#### Example 3:

**Input:**  $s_1 = "", s_2 = "", s_3 = ""$

**Output:** true

#### Constraints:

- $0 \leq s_1.length, s_2.length \leq 100$
- $0 \leq s_3.length \leq 200$
- $s_1$ ,  $s_2$ , and  $s_3$  consist of lowercase English letters.

**Follow up:** Could you solve it using only  $O(s_2.length)$  additional memory space?

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    from functools import lru_cache
    def isInterleave(self, s1: str, s2: str, s3: str) -> bool:
        len_s1, len_s2, len_s3 = len(s1), len(s2), len(s3)
        index1, index2, index3 = 0, 0, 0

        @lru_cache(None)
        def check_string_combinations(index1: int, index2: int):
            index3 = index1 + index2
            if index3 == len_s3:
                return True

            result = False

            # Try taking next char from s1
            if index1 < len_s1 and s1[index1] == s3[index3]:
                result = result or check_string_combinations(index1 + 1, index2)

            # Try taking next char from s2
            if index2 < len_s2 and s2[index2] == s3[index3]:
                result = result or check_string_combinations(index1, index2 + 1)

            return result
        if len_s1 + len_s2 != len_s3:
            return False
        return check_string_combinations(0,0)
```

## [64 Minimum Path Sum \(link\)](#)

### Description

Given a  $m \times n$  grid filled with non-negative numbers, find a path from top left to bottom right, which minimizes the sum of all numbers along its path.

**Note:** You can only move either down or right at any point in time.

#### Example 1:

1	3	1
1	5	1
4	2	1

**Input:** grid = [[1,3,1],[1,5,1],[4,2,1]]

**Output:** 7

**Explanation:** Because the path 1 → 3 → 1 → 1 → 1 minimizes the sum.

#### Example 2:

**Input:** grid = [[1,2,3],[4,5,6]]

**Output:** 12

#### Constraints:

- $m == \text{grid.length}$
- $n == \text{grid[i].length}$
- $1 \leq m, n \leq 200$
- $0 \leq \text{grid[i][j]} \leq 200$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def minPathSum(self, grid: List[List[int]]) -> int:
        # tsum(n,m) = min(tsum(m-1,n), tsum(m,n-1))
        m = len(grid)
        n = len(grid[0])

        cache = {}
        def find_path(row: int, col: int) -> int:
            # If in cache
            if cache.get((row, col), False):
                return cache[(row, col)]

            # If out of bounds
            if row < 0 or col < 0 or row >= len(grid) or col >= len(grid[0]):
                return float('inf')

            # If reached target
            if row == 0 and col == 0:
                cache[(row, col)] = grid[row][col]
                return cache[(row, col)]

            # path = current + min_of_rest
            cache[(row, col)] = grid[row][col] + min(
                find_path(row-1, col),
                find_path(row, col-1)
            )
            return cache[(row, col)]
        return find_path(m-1, n-1)
```

## [62 Unique Paths \(link\)](#)

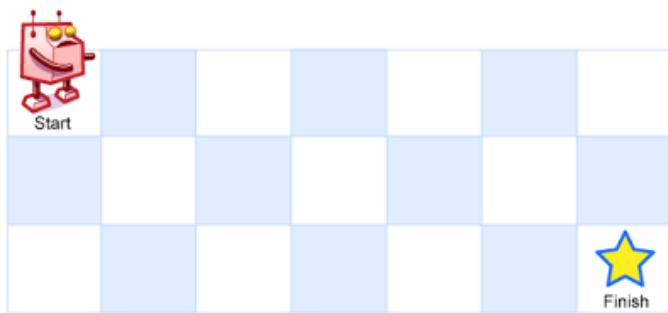
### Description

There is a robot on an  $m \times n$  grid. The robot is initially located at the **top-left corner** (i.e.,  $\text{grid}[0][0]$ ). The robot tries to move to the **bottom-right corner** (i.e.,  $\text{grid}[m - 1][n - 1]$ ). The robot can only move either down or right at any point in time.

Given the two integers  $m$  and  $n$ , return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.

The test cases are generated so that the answer will be less than or equal to  $2 * 10^9$ .

#### Example 1:



**Input:**  $m = 3$ ,  $n = 7$   
**Output:** 28

#### Example 2:

**Input:**  $m = 3$ ,  $n = 2$   
**Output:** 3  
**Explanation:** From the top-left corner, there are a total of 3 ways to reach the bottom-right corner.  
1. Right  $\rightarrow$  Down  $\rightarrow$  Down  
2. Down  $\rightarrow$  Down  $\rightarrow$  Right  
3. Down  $\rightarrow$  Right  $\rightarrow$  Down

### Constraints:

- $1 \leq m, n \leq 100$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        # paths(m,n) = paths(m-1,n) + paths(m, n-1)
        cache = {}
        def total_paths(m, n):
            if cache.get((m,n), False):
                return cache[(m,n)]
            if (m and not n) or (n and not m):
                cache[(m,n)] = 1
                return 1
            if m < 0 or n < 0:
                return 0
            if not m and not n:
                cache[(m,n)] = 0
                return 0

            cache[(m,n)] = total_paths(m-1,n) + total_paths(m,n-1)
            return cache[(m,n)]
        if m == 1 and n == 1:
            return 1
        return total_paths(m-1,n-1)
```

## [72 Edit Distance \(link\)](#)

### 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:

- $0 \leq \text{word1.length}, \text{word2.length} \leq 500$
- `word1` and `word2` consist of lowercase English letters.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def minDistance(self, word1: str, word2: str) -> int:
        cache = {}
        def edit_distance(word1, len1, word2, len2):
            if cache.get((len1, len2), False):
                return cache[(len1, len2)]
            if len1 == 0 or len2 == 0: # if string empty
                cache[(len1, len2)] = max(len1, len2)
                return cache[(len1, len2)]

            if word1[len1-1] == word2[len2-1]:
                cache[(len1, len2)] = edit_distance(word1, len1 - 1, word2, len2 - 1)
                return cache[(len1, len2)]

            cache[(len1, len2)] = 1 + min(
                edit_distance(word1, len1-1, word2, len2),
                edit_distance(word1, len1, word2, len2-1),
                edit_distance(word1, len1-1, word2, len2-1),
            )
            return cache[(len1, len2)]
        return edit_distance(word1, len(word1), word2, len(word2))
```

## 22 Generate Parentheses ([link](#))

### Description

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

#### Example 1:

```
Input: n = 3
Output: ["((()))","(()())","(())()","()((()))","()()()"]
```

#### Example 2:

```
Input: n = 1
Output: ["()"]
```

#### Constraints:

- $1 \leq n \leq 8$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        answer = []
        def generate_parenthesis(remaining_open: int, remaining_closed: int, current_string: str):
            if not remaining_open and not remaining_closed:
                answer.append(current_string)
                return

            if remaining_open:
                generate_parenthesis(remaining_open - 1, remaining_closed, current_string + '(')

            if remaining_closed > remaining_open: # MOST IMPORTANT PART
                generate_parenthesis(remaining_open, remaining_closed - 1, current_string + ')')

        generate_parenthesis(n, n, "")
        return answer
```

## [747 Min Cost Climbing Stairs \(link\)](#)

### Description

You are given an integer array `cost` where `cost[i]` is the cost of  $i^{\text{th}}$  step on a staircase. Once you pay the cost, you can either climb one or two steps.

You can either start from the step with index 0, or the step with index 1.

Return *the minimum cost to reach the top of the floor*.

#### Example 1:

```
Input: cost = [10,15,20]
Output: 15
Explanation: You will start at index 1.
- Pay 15 and climb two steps to reach the top.
The total cost is 15.
```

#### Example 2:

```
Input: cost = [1,100,1,1,1,100,1,1,100,1]
Output: 6
Explanation: You will start at index 0.
- Pay 1 and climb two steps to reach index 2.
- Pay 1 and climb two steps to reach index 4.
- Pay 1 and climb two steps to reach index 6.
- Pay 1 and climb one step to reach index 7.
- Pay 1 and climb two steps to reach index 9.
- Pay 1 and climb one step to reach the top.
The total cost is 6.
```

#### Constraints:

- $2 \leq \text{cost.length} \leq 1000$
- $0 \leq \text{cost}[i] \leq 999$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def minCostClimbingStairs(self, cost: List[int]) -> int:
        # cache = {}
        # def find_min_cost(remaining_steps: int) -> int:
        #     if cache.get(remaining_steps, False):
        #         return cache[remaining_steps]
        #     if remaining_steps < 0:
        #         return 0
        #     if remaining_steps <= 1:
        #         return cost[remaining_steps]

        #     cache[remaining_steps] = cost[remaining_steps] + min(find_min_cost(remaining_steps - 1),
        #                                                       find_min_cost(remaining_steps - 2))

        # return min(find_min_cost(len(cost) - 1), find_min_cost(len(cost) - 2))

        dp = [0] * len(cost)
        for i in range(len(cost)):
            if i < 2:
                dp[i] = cost[i]
            else:
                dp[i] = cost[i] + min(dp[i-1], dp[i-2])

        return min(dp[len(cost)-1], dp[len(cost)-2])
```

## [121 Best Time to Buy and Sell Stock \(link\)](#)

### Description

You are given an array `prices` where `prices[i]` is the price of a given stock on the  $i^{\text{th}}$  day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return *the maximum profit you can achieve from this transaction*. If you cannot achieve any profit, return 0.

#### Example 1:

```
Input: prices = [7,1,5,3,6,4]  
Output: 5
```

**Explanation:** Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.  
Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.

#### Example 2:

```
Input: prices = [7,6,4,3,1]  
Output: 0
```

**Explanation:** In this case, no transactions are done and the max profit = 0.

#### Constraints:

- $1 \leq \text{prices.length} \leq 10^5$
- $0 \leq \text{prices}[i] \leq 10^4$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        max_profit = 0
        if len(prices) <= 1:
            return max_profit

        buy_time, sell_time = 0, 1
        current_profit = prices[sell_time] - prices[buy_time]

        for sell_time in range(len(prices)):
            current_profit = prices[sell_time] - prices[buy_time]
            max_profit = max(max_profit, current_profit)
            if prices[sell_time] < prices[buy_time]:
                buy_time = sell_time

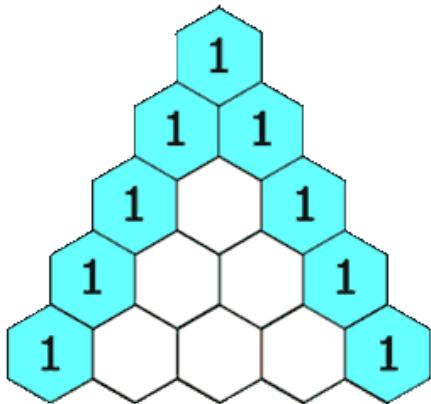
        return max_profit
```

## [119 Pascal's Triangle II \(link\)](#)

### Description

Given an integer `rowIndex`, return the `rowIndexth` (**0-indexed**) row of the **Pascal's triangle**.

In **Pascal's triangle**, each number is the sum of the two numbers directly above it as shown:



#### Example 1:

```
Input: rowIndex = 3
Output: [1,3,3,1]
```

#### Example 2:

```
Input: rowIndex = 0
Output: [1]
```

#### Example 3:

```
Input: rowIndex = 1
Output: [1,1]
```

#### Constraints:

- $0 \leq \text{rowIndex} \leq 33$

**Follow up:** Could you optimize your algorithm to use only  $O(\text{rowIndex})$  extra space?

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def getRow(self, rowIndex: int) -> List[int]:
        result = []
        row_1 = [1]
        row_2 = [1, 1]

        if rowIndex == 0:
            return row_1

        result.append(row_1)
        result.append(row_2)
        if rowIndex == 1:
            return result[-1]

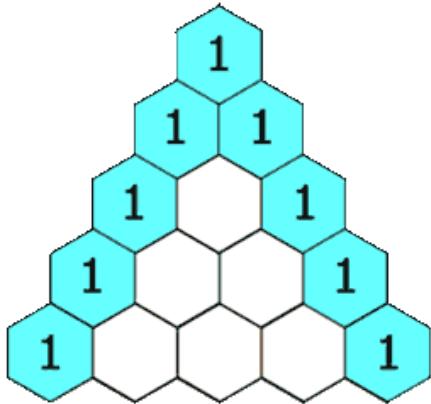
        for i in range(rowIndex + 1):
            number_of_elements = i + 1
            current_pascals_row = []
            for j in range(number_of_elements):
                if j == 0 or j == (number_of_elements - 1):
                    value = 1
                else:
                    value = result[-1][j] + result[-1][j-1]
                current_pascals_row.append(value)
            result.append(current_pascals_row)
        return result[-1]
```

## [118 Pascal's Triangle \(link\)](#)

### Description

Given an integer `numRows`, return the first `numRows` of **Pascal's triangle**.

In **Pascal's triangle**, each number is the sum of the two numbers directly above it as shown:



#### Example 1:

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

#### Example 2:

```
Input: numRows = 1
Output: [[1]]
```

#### Constraints:

- $1 \leq \text{numRows} \leq 30$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def generate(self, numRows: int) -> List[List[int]]:
        result = []
        row_1 = [1]
        row_2 = [1, 1]
        if numRows == 0:
            return []
        if numRows == 1:
            return [row_1]
        result.append(row_1)
        result.append(row_2)
        if numRows == 2:
            return result

        for i in range(2, numRows):
            pascal_current_array = []
            number_of_elements = i + 1
            for j in range(number_of_elements):
                if j == 0 or j == number_of_elements - 1:
                    value = 1
                else:
                    value = result[-1][j] + result[-1][j-1]
                pascal_current_array.append(value)
            result.append(pascal_current_array)

        return result
```

## [70 Climbing Stairs \(link\)](#)

### Description

You are climbing a staircase. It takes  $n$  steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

#### Example 1:

```
Input: n = 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps
```

#### Example 2:

```
Input: n = 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step
```

### Constraints:

- $1 \leq n \leq 45$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def climbStairs(self, n: int) -> int:
        cache = {}
        def number_of_steps(remaining_steps: int) -> int:
            if cache.get(remaining_steps):
                return cache[remaining_steps]
            if remaining_steps <= 0:
                return 0
            if remaining_steps == 1:
                return 1
            if remaining_steps == 2:
                return 2
            cache[remaining_steps] = number_of_steps(remaining_steps - 1) + number_of_steps(remaining_steps - 2)
            return cache[remaining_steps]
        return number_of_steps(n)
```

## [207 Course Schedule \(link\)](#)

### 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 `true` if you can finish all courses. Otherwise, return `false`.

#### Example 1:

```
Input: numCourses = 2, prerequisites = [[1,0]]  
Output: true  
Explanation: There are a total of 2 courses to take.  
To take course 1 you should have finished course 0. So it is possible.
```

#### Example 2:

```
Input: numCourses = 2, prerequisites = [[1,0],[0,1]]  
Output: false  
Explanation: There are a total of 2 courses to take.  
To take course 1 you should have finished course 0, and to take course 0 you should also have
```

#### Constraints:

- `1 <= numCourses <= 2000`
- `0 <= prerequisites.length <= 5000`
- `prerequisites[i].length == 2`
- `0 <= ai, bi < numCourses`
- All the pairs `prerequisites[i]` are **unique**.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    from collections import deque
    def canFinish(self, numCourses: int, prerequisites: List[List[int]]) -> bool:
        # initialise variables
        graph = collections.defaultdict(list)
        indegree = collections.defaultdict(int)
        visited = collections.defaultdict(bool)
        visited_count = 0
        queue = deque()

        # create graph
        for i in range(numCourses):
            graph[i] = []
            indegree[i] = 0

        for edge in prerequisites:
            course, pre_course = edge[0], edge[1]
            graph[pre_course].append(course)
            indegree[course] += 1

        # Add starting nodes to queue
        for course in range(numCourses):
            if indegree[course] == 0:
                queue.append(course)

        # traverse the graph
        while queue:
            current = queue.popleft()
            visited[current] = True
            visited_count += 1
            for neighbor in graph[current]:
                indegree[neighbor] -= 1
                if indegree[neighbor] == 0 and not visited[neighbor]:
                    queue.append(neighbor)

        # if all nodes visited, true, else false
        return (visited_count) == (numCourses)
```

## [347 Top K Frequent Elements \(link\)](#)

### Description

Given an integer array `nums` and an integer `k`, return *the k most frequent elements*. You may return the answer in **any order**.

#### Example 1:

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

**Output:** [1,2]

#### Example 2:

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

**Output:** [1]

#### Example 3:

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

**Output:** [1,2]

#### Constraints:

- $1 \leq \text{nums.length} \leq 10^5$
- $-10^4 \leq \text{nums}[i] \leq 10^4$
- `k` is in the range  $[1, \text{the number of unique elements in the array}]$ .
- It is **guaranteed** that the answer is **unique**.

**Follow up:** Your algorithm's time complexity must be better than  $O(n \log n)$ , where  $n$  is the array's size.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    import heapq
    def topKFrequent(self, nums: List[int], k: int) -> List[int]:
        frequency = Counter(nums)
        result = []
        final_answer = []

        for element, occurrence in frequency.items():
            result.append((occurrence, element))

        heapq._heapify_max(result)
        for i in range(k):
            final_answer.append(heapq._heappop_max(result)[1])
        return final_answer
```

## [128 Longest Consecutive Sequence \(link\)](#)

### Description

Given an unsorted array of integers `nums`, return *the length of the longest consecutive elements sequence*.

You must write an algorithm that runs in  $O(n)$  time.

#### Example 1:

**Input:** `nums = [100, 4, 200, 1, 3, 2]`  
**Output:** 4

**Explanation:** The longest consecutive elements sequence is `[1, 2, 3, 4]`. Therefore its length is 4.

#### Example 2:

**Input:** `nums = [0, 3, 7, 2, 5, 8, 4, 6, 0, 1]`  
**Output:** 9

#### Example 3:

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

### Constraints:

- $0 \leq \text{nums.length} \leq 10^5$
- $-10^9 \leq \text{nums}[i] \leq 10^9$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def longestConsecutive(self, nums: List[int]) -> int:
        visited = set(nums)
        longest = 0

        for n in visited:
            if (n-1) not in visited:
                length = 0
                while (n + length) in visited:
                    length += 1
                longest = max(length, longest)
        return longest
```

## 36 Valid Sudoku (link)

### Description

Determine if a  $9 \times 9$  Sudoku board is valid. Only the filled cells need to be validated **according to the following rules**:

1. Each row must contain the digits 1–9 without repetition.
2. Each column must contain the digits 1–9 without repetition.
3. Each of the nine  $3 \times 3$  sub-boxes of the grid must contain the digits 1–9 without repetition.

#### Note:

- A Sudoku board (partially filled) could be valid but is not necessarily solvable.
- Only the filled cells need to be validated according to the mentioned rules.

#### Example 1:

5	3			7				
6			1	9	5			
	9	8				6		
8				6				3
4			8		3			1
7				2				6
	6					2	8	
		4	1	9				5
			8			7	9	

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

#### Example 2:

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

Explanation: Same as Example 1, except with the 5 in the top left corner being modified to 8.

### Constraints:

- $\text{board.length} == 9$
- $\text{board}[i].length == 9$
- $\text{board}[i][j]$  is a digit 1–9 or '.'.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    from collections import defaultdict
    def isValidSudoku(self, board: List[List[str]]) -> bool:
        rows, cols, square = defaultdict(set), defaultdict(set), defaultdict(set)

        for i in range(9):
            for j in range(9):
                if (board[i][j] != "." and board[i][j] in rows[i] or board[i][j] in cols[j] or
                    rows[i].add(board[i][j])
                    cols[j].add(board[i][j])
                    square[(i//3,j//3)].add(board[i][j])
                    return False
                if board[i][j] != ".":
                    rows[i].add(board[i][j])
                    cols[j].add(board[i][j])
                    square[(i//3,j//3)].add(board[i][j])
        return True
```

## [217 Contains Duplicate \(link\)](#)

### Description

Given an integer array `nums`, return `true` if any value appears **at least twice** in the array, and return `false` if every element is distinct.

#### Example 1:

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

**Output:** `true`

#### Explanation:

The element 1 occurs at the indices 0 and 3.

#### Example 2:

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

**Output:** `false`

#### Explanation:

All elements are distinct.

#### Example 3:

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

**Output:** `true`

#### Constraints:

- $1 \leq \text{nums.length} \leq 10^5$
- $-10^9 \leq \text{nums}[i] \leq 10^9$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:
        seen = set()
        for i in range(len(nums)):
            if nums[i] in seen:
                return True
            seen.add(nums[i])
        return False
```

## [210 Course Schedule II \(link\)](#)

### 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

#### 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 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**.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def findOrder(self, numCourses: int, prerequisites: List[List[int]]) -> List[int]:
        graph, indegree, queue = collections.defaultdict(list), collections.defaultdict(int), collections.defaultdict(bool)

        # build the graph
        for i in range(numCourses):
            graph[i] = []
            indegree[i] = 0

        # Add the edges
        for edge in prerequisites:
            course, pre_course = edge[1], edge[0]
            graph[pre_course].append(course)
            indegree[course] += 1

        # fetch the starting points
        for node in graph.keys():
            if indegree[node] == 0:
                queue.append(node)

        traversal_order = []
        visited_count = 0

        # traverse the graph
        while queue:
            current = queue.popleft()
            visited[current] = True
            visited_count += 1

            for neighbor in graph[current]:
                # Remove the current element edge from its neighbor
                indegree[neighbor] -= 1

                # If all prerequisites of the course are met
                if indegree[neighbor] == 0 and not visited[neighbor]:
                    queue.append(neighbor)

            traversal_order.append(current)

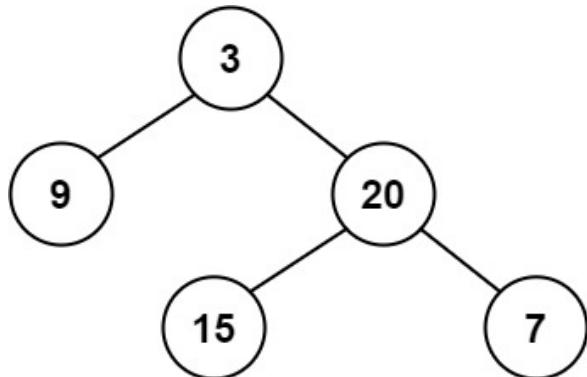
        # If all courses can be completed
        if visited_count == numCourses:
            return traversal_order[::-1]
        else:
            return []
```

## [110 Balanced Binary Tree \(link\)](#)

### Description

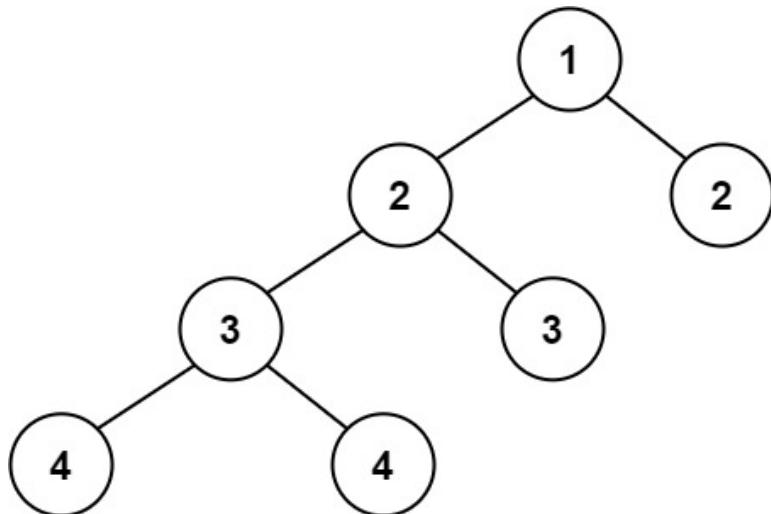
Given a binary tree, determine if it is **height-balanced**.

#### Example 1:



```
Input: root = [3,9,20,null,null,15,7]  
Output: true
```

#### Example 2:



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

#### Example 3:

```
Input: root = []  
Output: true
```

### Constraints:

- The number of nodes in the tree is in the range  $[0, 5000]$ .
- $-10^4 \leq \text{Node.val} \leq 10^4$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def isBalanced(self, root: Optional[TreeNode]) -> bool:
        def height(root):
            if not root:
                return 0
            return 1 + max(height(root.left), height(root.right))

        if not root:
            return True

        left_height = height(root.left)
        right_height = height(root.right)

        if abs(left_height - right_height) <= 1:
            return self.isBalanced(root.left) and self.isBalanced(root.right)
        return False
```

## [49 Group Anagrams \(link\)](#)

### Description

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

#### Example 1:

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

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

#### Explanation:

- There is no string in `strs` that can be rearranged to form "bat".
- The strings "nat" and "tan" are anagrams as they can be rearranged to form each other.
- The strings "ate", "eat", and "tea" are anagrams as they can be rearranged to form each other.

#### Example 2:

**Input:** `strs = [""]`

**Output:** `[[""]]`

#### Example 3:

**Input:** `strs = ["a"]`

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

#### Constraints:

- $1 \leq \text{strs.length} \leq 10^4$
- $0 \leq \text{strs[i].length} \leq 100$
- `strs[i]` consists of lowercase English letters.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
        if not strs:
            return []

        groups = collections.defaultdict(list)
        for word in strs:
            hashkey = str(sorted(word))
            groups[hashkey].append(word)

        result = []
        for hashword in groups.keys():
            result.append(groups[hashword])

        return result
```

## [111 Minimum Depth of Binary Tree \(link\)](#)

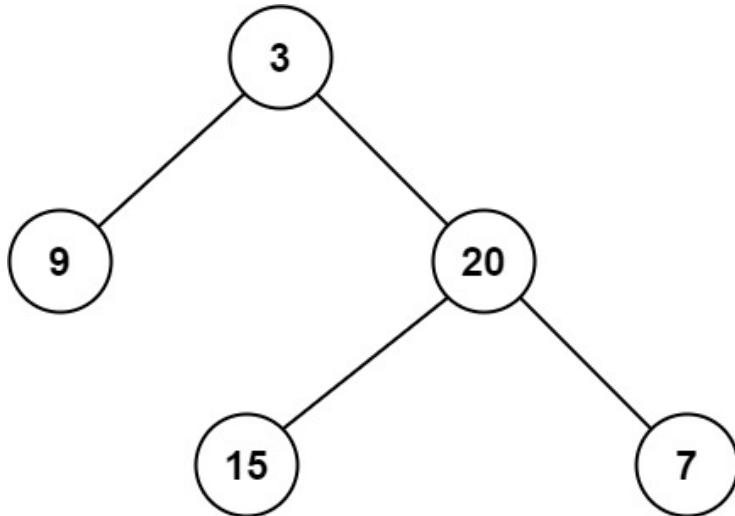
### Description

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

**Note:** A leaf is a node with no children.

#### Example 1:



```
Input: root = [3,9,20,null,null,15,7]
Output: 2
```

#### Example 2:

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

#### Constraints:

- The number of nodes in the tree is in the range  $[0, 10^5]$ .
- $-1000 \leq \text{Node.val} \leq 1000$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def minDepth(self, root: Optional[TreeNode]) -> int:
        if not root:
            return 0

        if not root.left and not root.right:
            return 1

        if not root.left and root.right:
            return 1 + self.minDepth(root.right)

        if not root.right and root.left:
            return 1 + self.minDepth(root.left)

        return 1 + min(self.minDepth(root.left), self.minDepth(root.right))
```

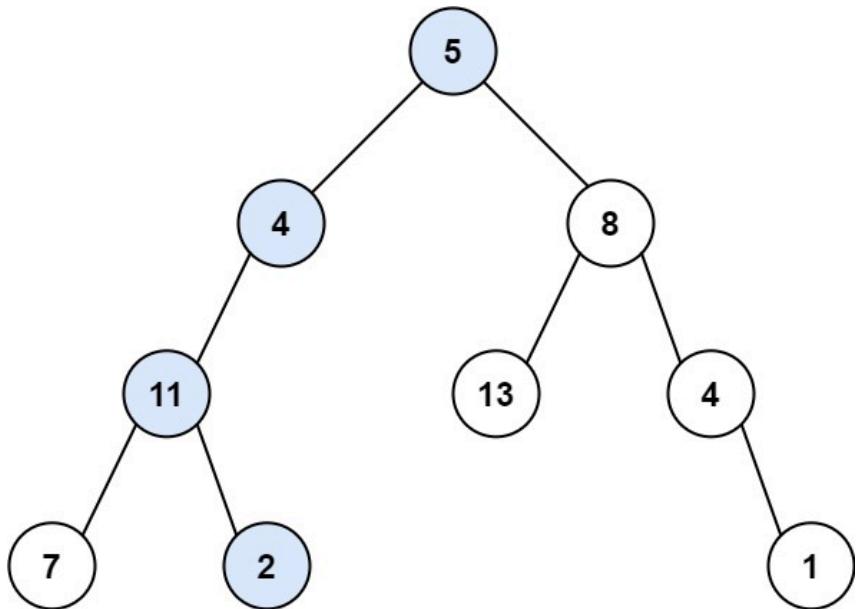
## [112 Path Sum \(link\)](#)

### Description

Given the root of a binary tree and an integer `targetSum`, return true if the tree has a **root-to-leaf** path such that adding up all the values along the path equals `targetSum`.

A **leaf** is a node with no children.

#### Example 1:

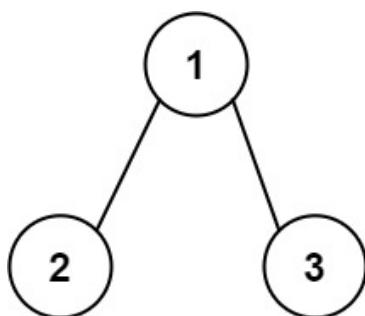


**Input:** root = [5,4,8,11,null,13,4,7,2,null,null,null,1], targetSum = 22

**Output:** true

**Explanation:** The root-to-leaf path with the target sum is shown.

#### Example 2:



**Input:** root = [1,2,3], targetSum = 5

**Output:** false

**Explanation:** There are two root-to-leaf paths in the tree:

(1 --> 2): The sum is 3.

(1 --> 3): The sum is 4.

There is no root-to-leaf path with sum = 5.

#### Example 3:

**Input:** root = [], targetSum = 0

**Output:** false

**Explanation:** Since the tree is empty, there are no root-to-leaf paths.

**Constraints:**

- The number of nodes in the tree is in the range  $[0, 5000]$ .
- $-1000 \leq \text{Node.val} \leq 1000$
- $-1000 \leq \text{targetSum} \leq 1000$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def hasPathSum(self, root: Optional[TreeNode], targetSum: int) -> bool:
        if not root:
            return False

        if root and targetSum - root.val == 0 and (not root.left and not root.right):
            return True

        return self.hasPathSum(root.left, targetSum-root.val) or self.hasPathSum(root.right, ta
```

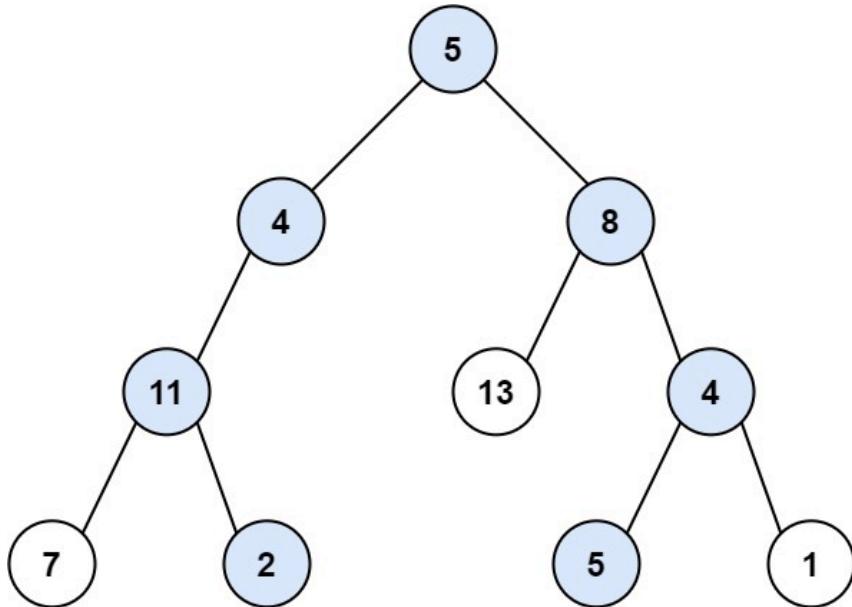
## [113 Path Sum II \(link\)](#)

### Description

Given the root of a binary tree and an integer targetSum, return *all root-to-leaf paths where the sum of the node values in the path equals targetSum*. Each path should be returned as a list of the node values, not node references.

A **root-to-leaf** path is a path starting from the root and ending at any leaf node. A **leaf** is a node with no children.

#### Example 1:



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

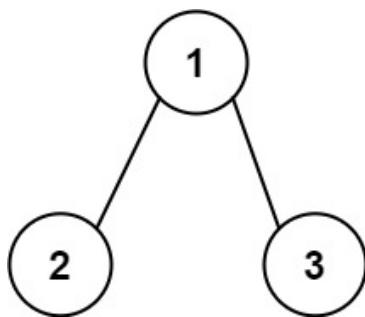
**Output:** [[5,4,11,2],[5,8,4,5]]

**Explanation:** There are two paths whose sum equals targetSum:

5 + 4 + 11 + 2 = 22

5 + 8 + 4 + 5 = 22

#### Example 2:



**Input:** root = [1,2,3], targetSum = 5

**Output:** []

#### Example 3:

**Input:** root = [1,2], targetSum = 0

**Output:** []

**Constraints:**

- The number of nodes in the tree is in the range  $[0, 5000]$ .
- $-1000 \leq \text{Node.val} \leq 1000$
- $-1000 \leq \text{targetSum} \leq 1000$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def pathSum(self, root: Optional[TreeNode], targetSum: int) -> List[List[int]]:
        result = []

        def dfs(root, target_sum, path):
            if not root:
                return

            # If leaf node
            if not root.left and not root.right:
                if target_sum - root.val == 0:
                    path.append(root.val)
                    result.append(path)
                return

            # If not leaf node
            path.append(root.val)
            target_sum = target_sum - root.val
            dfs(root.left, target_sum, path.copy())
            dfs(root.right, target_sum, path.copy())

        dfs(root, targetSum, [])
        return result
```

## 35 Search Insert Position ([link](#))

### Description

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:

- $1 \leq \text{nums.length} \leq 10^4$
- $-10^4 \leq \text{nums}[i] \leq 10^4$
- **nums** contains **distinct** values sorted in **ascending** order.
- $-10^4 \leq \text{target} \leq 10^4$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
        left, right = 0, len(nums) - 1
        mid = (left + right) // 2

        if target > nums[right]:
            return right + 1

        if target < nums[left]:
            return 0

        while left <= right:
            mid = (left + right) // 2
            if target == nums[mid]:
                return mid
            elif (target > nums[mid] and target < nums[mid + 1]):
                return mid + 1
            elif target > nums[mid]:
                left = mid + 1
            elif target < nums[mid]:
                right = mid - 1

        return mid
```

## 141 Linked List Cycle (link)

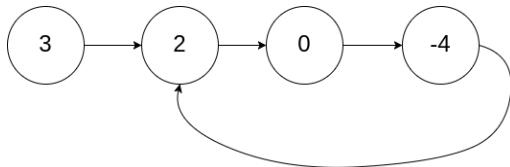
### Description

Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, `pos` is used to denote the index of the node that tail's next pointer is connected to. **Note that `pos` is not passed as a parameter.**

Return `true` if there is a cycle in the linked list. Otherwise, return `false`.

#### Example 1:

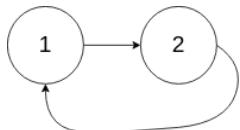


**Input:** `head = [3,2,0,-4]`, `pos = 1`

**Output:** `true`

**Explanation:** There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).

#### Example 2:



**Input:** `head = [1,2]`, `pos = 0`

**Output:** `true`

**Explanation:** There is a cycle in the linked list, where the tail connects to the 0th node.

#### Example 3:



**Input:** `head = [1]`, `pos = -1`

**Output:** `false`

**Explanation:** There is no cycle in the linked list.

#### Constraints:

- The number of the nodes in the list is in the range  $[0, 10^4]$ .
- $-10^5 \leq \text{Node.val} \leq 10^5$
- `pos` is  $-1$  or a **valid index** in the linked-list.

**Follow up:** Can you solve it using  $O(1)$  (i.e. constant) memory?

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def hasCycle(self, head: Optional[ListNode]) -> bool:
        if not head:
            return False
        if not head.next:
            return False
        fast, slow = head, head
        while fast and slow:
            fast = fast.next
            slow = slow.next
            if not fast:
                return False
            fast = fast.next
            if fast == slow:
                return True
        return False
```

## 215 Kth Largest Element in an Array ([link](#))

### Description

Given an integer array `nums` and an integer `k`, return *the k<sup>th</sup> largest element in the array*.

Note that it is the `kth` largest element in the sorted order, not the `kth` distinct element.

Can you solve it without sorting?

#### Example 1:

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

#### Example 2:

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

#### Constraints:

- $1 \leq k \leq \text{nums.length} \leq 10^5$
- $-10^4 \leq \text{nums}[i] \leq 10^4$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    import heapq
    def findKthLargest(self, nums: List[int], k: int) -> int:
        heapq._heapify_max(nums)
        result = float('inf')
        for i in range(k):
            result = min(result, heapq._heappop_max(nums))
        return result
```

## [11 Container With Most Water \(link\)](#)

### Description

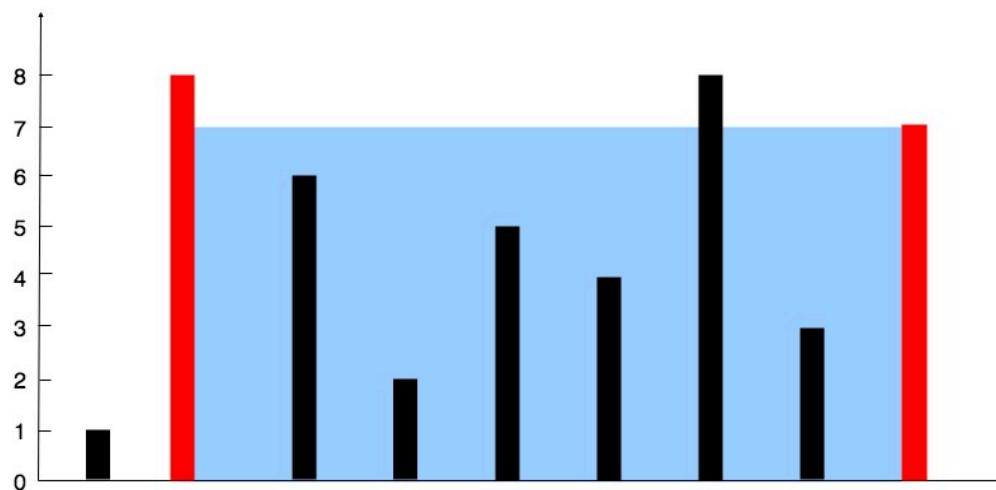
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the  $i^{\text{th}}$  line are  $(i, 0)$  and  $(i, \text{height}[i])$ .

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return *the maximum amount of water a container can store*.

**Notice** that you may not slant the container.

#### Example 1:



**Input:** `height = [1,8,6,2,5,4,8,3,7]`  
**Output:** 49

**Explanation:** The above vertical lines are represented by array `[1,8,6,2,5,4,8,3,7]`. In this case, the maximum area of 49 is formed by the lines at indices 1 and 8.

#### Example 2:

**Input:** `height = [1,1]`  
**Output:** 1

#### Constraints:

- `n == height.length`
- $2 \leq n \leq 10^5$
- $0 \leq \text{height}[i] \leq 10^4$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def maxArea(self, height: List[int]) -> int:
        def calculate_area(bottom: int, left_height: int, right_height: int) -> int:
            return bottom * min(left_height, right_height)

        left, right, max_area = 0, len(height) - 1, 0
        while left < right:
            current_area = calculate_area(right - left, height[left], height[right])
            max_area = max(max_area, current_area)
            if height[left] < height[right]:
                left = left + 1
            else:
                right = right - 1
        return max_area
```

## [7 Reverse Integer \(link\)](#)

### Description

Given a signed 32-bit integer  $x$ , return  $x$  with its digits reversed. If reversing  $x$  causes the value to go outside the signed 32-bit integer range  $[-2^{31}, 2^{31} - 1]$ , then return 0.

**Assume the environment does not allow you to store 64-bit integers (signed or unsigned).**

#### Example 1:

```
Input: x = 123
Output: 321
```

#### Example 2:

```
Input: x = -123
Output: -321
```

#### Example 3:

```
Input: x = 120
Output: 21
```

#### Constraints:

- $-2^{31} \leq x \leq 2^{31} - 1$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def reverse(self, x: int) -> int:
        MIN_INT = -2**31
        MAX_INT = 2**31-1

        if x < MIN_INT or x > MAX_INT:
            return 0

        sign = 1
        if x < 0:
            sign = -1
            x = x * -1

        copy_number = x
        reversed_number = 0
        while copy_number > 0:
            remainder = copy_number % 10
            reversed_number = reversed_number * 10 + remainder
            copy_number = copy_number // 10

        if reversed_number < MIN_INT or reversed_number > MAX_INT:
            return 0

        return reversed_number * sign
```

## 9 Palindrome Number (link)

### Description

Given an integer  $x$ , return `true` if  $x$  is a **palindrome**, and `false` otherwise.

#### Example 1:

**Input:**  $x = 121$

**Output:** `true`

**Explanation:** 121 reads as 121 from left to right and from right to left.

#### Example 2:

**Input:**  $x = -121$

**Output:** `false`

**Explanation:** From left to right, it reads -121. From right to left, it becomes 121-. Therefore

#### Example 3:

**Input:**  $x = 10$

**Output:** `false`

**Explanation:** Reads 01 from right to left. Therefore it is not a palindrome.

### Constraints:

- $-2^{31} \leq x \leq 2^{31} - 1$

**Follow up:** Could you solve it without converting the integer to a string?

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def isPalindrome(self, x: int) -> bool:
        reverse_number = 0
        copy_number = x

        if x < 0:
            return False

        while copy_number > 0:
            remainder = copy_number % 10
            copy_number = copy_number // 10
            reverse_number = reverse_number * 10 + remainder

        if reverse_number == x:
            return True

        return False
```

## [1 Two Sum \(link\)](#)

### 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:

- $2 \leq \text{nums.length} \leq 10^4$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- $-10^9 \leq \text{target} \leq 10^9$
- **Only one valid answer exists.**

**Follow-up:** Can you come up with an algorithm that is less than  $O(n^2)$  time complexity?

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        seen = collections.defaultdict(int)
        for i in range(len(nums)):
            seen[nums[i]] = i

        for i in range(len(nums)):
            if seen[target - nums[i]] and seen[target - nums[i]] != i:
                return [i, seen[target - nums[i]]]
```

## [1646 Kth Missing Positive Number \(link\)](#)

### Description

Given an array `arr` of positive integers sorted in a **strictly increasing order**, and an integer `k`.

Return *the k<sup>th</sup> positive integer that is missing from this array*.

#### Example 1:

**Input:** arr = [2,3,4,7,11], k = 5  
**Output:** 9

**Explanation:** The missing positive integers are [1,5,6,8,9,10,12,13,...]. The 5<sup>th</sup> missing positive integer is

#### Example 2:

**Input:** arr = [1,2,3,4], k = 2  
**Output:** 6

**Explanation:** The missing positive integers are [5,6,7,...]. The 2<sup>nd</sup> missing positive integer is

### Constraints:

- $1 \leq \text{arr.length} \leq 1000$
- $1 \leq \text{arr}[i] \leq 1000$
- $1 \leq k \leq 1000$
- $\text{arr}[i] < \text{arr}[j]$  for  $1 \leq i < j \leq \text{arr.length}$

### Follow up:

Could you solve this problem in less than  $O(n)$  complexity?

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def findKthPositive(self, arr: List[int], k: int) -> int:
        left, right = 0, len(arr) - 1

        while left <= right:
            mid = (left + right) // 2
            missing = arr[mid] - (mid + 1)

            if missing < k:
                left = mid + 1
            else:
                right = mid - 1

        return left + k
```

## 3 Longest Substring Without Repeating Characters ([link](#))

### Description

Given a string  $s$ , find the length of the **longest substring** without duplicate characters.

#### Example 1:

**Input:**  $s = "abcabcbb"$

**Output:** 3

**Explanation:** The answer is "abc", with the length of 3. Note that "bca" and "cab" are also correct.

#### Example 2:

**Input:**  $s = "bbbbbb"$

**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:

- $0 \leq s.length \leq 5 * 10^4$
- $s$  consists of English letters, digits, symbols and spaces.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        left, size = 0, len(s)
        visited = set()
        count = 0

        for right in range(size):
            while s[right] in visited:
                visited.remove(s[left])
                left += 1
            visited.add(s[right])
            count = max(count, right - left + 1)
        return count
```

## [88 Merge Sorted Array \(link\)](#)

### 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

### 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?

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
        """
        Do not return anything, modify nums1 in-place instead.
        """
        # num of ele in nums1 = m
        # num of ele in nums2 = n

        if n == 0:
            return

        last_idx = len(nums1) - 1
        while m > 0 and n > 0:
            if nums1[m-1] >= nums2[n-1]:
                nums1[last_idx] = nums1[m-1]
                m = m - 1
            else:
                nums1[last_idx] = nums2[n-1]
                n = n - 1
            last_idx = last_idx - 1

        while n > 0:
            nums1[last_idx] = nums2[n-1]
            last_idx = last_idx - 1
            n = n - 1
```

## 23 Merge k Sorted Lists (link)

### Description

You are given an array of  $k$  linked-lists `lists`, each linked-list is sorted in ascending order.

*Merge all the linked-lists into one sorted linked-list and return it.*

#### Example 1:

```
Input: lists = [[1,4,5],[1,3,4],[2,6]]  
Output: [1,1,2,3,4,4,5,6]  
Explanation: The linked-lists are:  
[  
    1->4->5,  
    1->3->4,  
    2->6  
]  
merging them into one sorted linked list:  
1->1->2->3->4->4->5->6
```

#### Example 2:

```
Input: lists = []  
Output: []
```

#### Example 3:

```
Input: lists = [[]]  
Output: []
```

### Constraints:

- $k == \text{lists.length}$
- $0 \leq k \leq 10^4$
- $0 \leq \text{lists}[i].\text{length} \leq 500$
- $-10^4 \leq \text{lists}[i][j] \leq 10^4$
- $\text{lists}[i]$  is sorted in **ascending order**.
- The sum of  $\text{lists}[i].\text{length}$  will not exceed  $10^4$ .

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    import heapq
    def mergeKLists(self, lists: List[Optional[ListNode]]) -> Optional[ListNode]:
        priority_queue = []
        heapq.heapify(priority_queue)

        for ll in lists:
            while ll:
                heapq.heappush(priority_queue, ll.val)
                ll = ll.next

        head = ListNode()
        dummy = head
        for i in range(len(priority_queue)):
            dummy.next = ListNode(heapq.heappop(priority_queue))
            dummy = dummy.next
        return head.next
```

## [125 Valid Palindrome \(link\)](#)

### Description

A phrase is a **palindrome** if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.

Given a string  $s$ , return `true` if it is a **palindrome**, or `false` otherwise.

#### Example 1:

```
Input: s = "A man, a plan, a canal: Panama"
Output: true
Explanation: "amanaplanacanalpanama" is a palindrome.
```

#### Example 2:

```
Input: s = "race a car"
Output: false
Explanation: "raceacar" is not a palindrome.
```

#### Example 3:

```
Input: s = " "
Output: true
Explanation: s is an empty string "" after removing non-alphanumeric characters.
Since an empty string reads the same forward and backward, it is a palindrome.
```

### Constraints:

- $1 \leq s.length \leq 2 * 10^5$
- $s$  consists only of printable ASCII characters.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def isPalindrome(self, s: str) -> bool:
        s = s.lower()
        left, right = 0, len(s) - 1

        while left < right:
            print(f"left: {s[left]}, right: {s[right]}")
            while s[left] not in "qwertyuiopasdfghjklzxcvbnm0987654321" and left < right:
                left += 1
            while s[right] not in "qwertyuiopasdfghjklzxcvbnm0987654321" and left < right:
                right -= 1
            if s[left] != s[right]:
                return False
            left += 1
            right -= 1

        return True
```

## [43 Multiply Strings \(link\)](#)

### Description

Given two non-negative integers `num1` and `num2` represented as strings, return the product of `num1` and `num2`, also represented as a string.

**Note:** You must not use any built-in BigInteger library or convert the inputs to integer directly.

#### Example 1:

```
Input: num1 = "2", num2 = "3"  
Output: "6"
```

#### Example 2:

```
Input: num1 = "123", num2 = "456"  
Output: "56088"
```

#### Constraints:

- $1 \leq \text{num1.length}, \text{num2.length} \leq 200$
- `num1` and `num2` consist of digits only.
- Both `num1` and `num2` do not contain any leading zero, except the number 0 itself.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def multiply(self, num1: str, num2: str) -> str:
        return str(int(num1) * int(num2))
```

## [2412 Minimum Amount of Time to Fill Cups \(link\)](#)

### Description

You have a water dispenser that can dispense cold, warm, and hot water. Every second, you can either fill up 2 cups with **different** types of water, or 1 cup of any type of water.

You are given a **0-indexed** integer array `amount` of length 3 where `amount[0]`, `amount[1]`, and `amount[2]` denote the number of cold, warm, and hot water cups you need to fill respectively. Return *the minimum number of seconds needed to fill up all the cups*.

#### Example 1:

```
Input: amount = [1,4,2]
Output: 4
Explanation: One way to fill up the cups is:
Second 1: Fill up a cold cup and a warm cup.
Second 2: Fill up a warm cup and a hot cup.
Second 3: Fill up a warm cup and a hot cup.
Second 4: Fill up a warm cup.
It can be proven that 4 is the minimum number of seconds needed.
```

#### Example 2:

```
Input: amount = [5,4,4]
Output: 7
Explanation: One way to fill up the cups is:
Second 1: Fill up a cold cup, and a hot cup.
Second 2: Fill up a cold cup, and a warm cup.
Second 3: Fill up a cold cup, and a warm cup.
Second 4: Fill up a warm cup, and a hot cup.
Second 5: Fill up a cold cup, and a hot cup.
Second 6: Fill up a cold cup, and a warm cup.
Second 7: Fill up a hot cup.
```

#### Example 3:

```
Input: amount = [5,0,0]
Output: 5
Explanation: Every second, we fill up a cold cup.
```

### Constraints:

- `amount.length == 3`
- `0 <= amount[i] <= 100`

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    import heapq
    def fillCups(self, amount: List[int]) -> int:
        amount = [-a for a in amount]
        heapq.heapify(amount)
        seconds = 0

        # first element will be maximum
        while amount[0] != 0:

            # Get top 2 cups
            first = heapq.heappop(amount)
            second = heapq.heappop(amount)

            # take 1 - 1 cup each
            if first: first -= 1
            if second: second -= 1

            # after taking 1 cup, push it back to heap
            heapq.heappush(amount, first)
            heapq.heappush(amount, second)

            seconds += 1

        return seconds
```

## [1014 K Closest Points to Origin \(link\)](#)

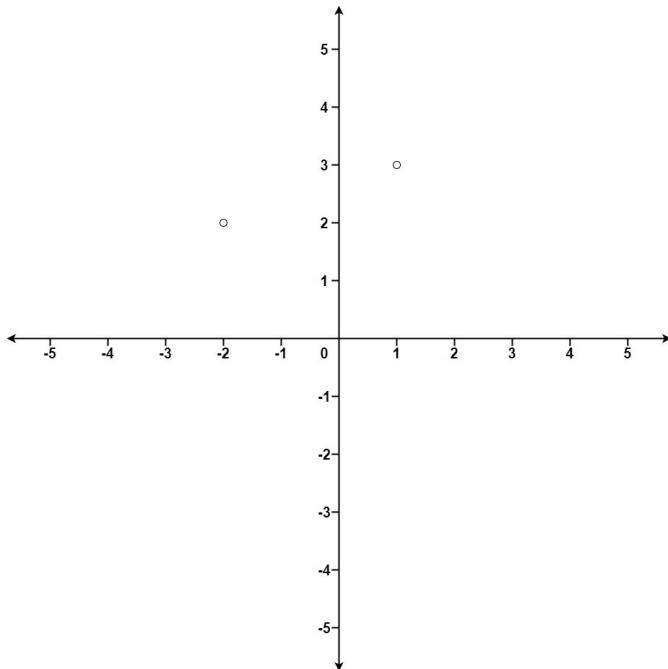
### Description

Given an array of points where  $\text{points}[i] = [x_i, y_i]$  represents a point on the **X-Y** plane and an integer  $k$ , return the  $k$  closest points to the origin  $(0, 0)$ .

The distance between two points on the **X-Y** plane is the Euclidean distance (i.e.,  $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ ).

You may return the answer in **any order**. The answer is **guaranteed** to be **unique** (except for the order that it is in).

#### Example 1:



**Input:** `points = [[1,3], [-2,2]]`,  $k = 1$

**Output:** `[-2,2]`

**Explanation:**

The distance between  $(1, 3)$  and the origin is  $\sqrt{10}$ .

The distance between  $(-2, 2)$  and the origin is  $\sqrt{8}$ .

Since  $\sqrt{8} < \sqrt{10}$ ,  $(-2, 2)$  is closer to the origin.

We only want the closest  $k = 1$  points from the origin, so the answer is just `[-2,2]`.

#### Example 2:

**Input:** `points = [[3,3], [5,-1], [-2,4]]`,  $k = 2$

**Output:** `[3,3], [-2,4]`

**Explanation:** The answer `[-2,4], [3,3]` would also be accepted.

#### Constraints:

- $1 \leq k \leq \text{points.length} \leq 10^4$
- $-10^4 \leq x_i, y_i \leq 10^4$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    import heapq
    from collections import defaultdict
    def kClosest(self, points: List[List[int]], k: int) -> List[List[int]]:
        distance_list = []
        heapq.heapify(distance_list)
        distance_dict = defaultdict(list)

        # Club all the points with same distance together
        for i in range(len(points)):
            x, y = points[i][0], points[i][1]
            distance = (x*x + y*y)
            distance_dict[distance].append([x,y])

        # Put the distance and points in a priority queue
        for distance, point_lists in distance_dict.items():
            heapq.heappush(distance_list, (distance, point_lists))

        # Until k points are added to the output, perform heap pop
        k_closest_answer = []
        while len(k_closest_answer) < k:
            current_closest_point = heapq.heappop(distance_list)
            distance, points = current_closest_point[0], current_closest_point[1]
            for point in points:
                k_closest_answer.append(point)
        return k_closest_answer
```

## [378 Kth Smallest Element in a Sorted Matrix \(link\)](#)

### Description

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

Note that it is the  $k^{\text{th}}$  smallest element **in the sorted order**, not the  $k^{\text{th}}$  **distinct** element.

You must find a solution with a memory complexity better than  $O(n^2)$ .

#### Example 1:

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

**Explanation:** The elements in the matrix are [1,5,9,10,11,12,13,13,15], and the 8<sup>th</sup> smallest num

#### Example 2:

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

### Constraints:

- $n == \text{matrix.length} == \text{matrix[i].length}$
- $1 \leq n \leq 300$
- $-10^9 \leq \text{matrix[i][j]} \leq 10^9$
- All the rows and columns of matrix are **guaranteed** to be sorted in **non-decreasing order**.
- $1 \leq k \leq n^2$

### Follow up:

- Could you solve the problem with a constant memory (i.e.,  $O(1)$  memory complexity)?
- Could you solve the problem in  $O(n)$  time complexity? The solution may be too advanced for an interview but you may find reading [this paper](#) fun.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    from collections import deque
    import heapq
    def kthSmallest(self, matrix: List[List[int]], k: int) -> int:
        result = []
        heapq.heapify(result)
        for i in range(len(matrix)):
            for j in range(len(matrix[0])):
                heapq.heappush(result, matrix[i][j])

        return_element = None
        for i in range(k):
            return_element = heapq.heappop(result)
        return return_element
```

## [2692 Take Gifts From the Richest Pile \(link\)](#)

### Description

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:

- $1 \leq \text{gifts.length} \leq 10^3$
- $1 \leq \text{gifts}[i] \leq 10^9$
- $1 \leq k \leq 10^3$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def pickGifts(self, gifts: List[int], k: int) -> int:
        storage = []
        for index, value in enumerate(gifts):
            storage.append(-1 * value)

        heapq.heapify(storage)

        for i in range(k):
            value = heapq.heappop(storage)
            value = -1 * value
            print(f"value popped: {value}")

            decrement = math.floor(math.sqrt(value))
            heapq.heappush(storage, -1 * decrement)

        return -1 * sum(storage)
```

## [506 Relative Ranks \(link\)](#)

### Description

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

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

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

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

#### Example 1:

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

#### Example 2:

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

#### Constraints:

- `n == score.length`
- $1 \leq n \leq 10^4$
- $0 \leq \text{score}[i] \leq 10^6$
- All the values in `score` are **unique**.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    import heapq
    def findRelativeRanks(self, score: List[int]) -> List[str]:
        result, output = [], [0] * len(score)
        for index, value in enumerate(score):
            result.append((value, index))

        heapq._heapify_max(result)

        # gold medal
        if result:
            value, index = heapq._heappop_max(result)
            output[index] = "Gold Medal"

        # silver medal
        if result:
            value, index = heapq._heappop_max(result)
            output[index] = "Silver Medal"

        # bronze medal
        if result:
            value, index = heapq._heappop_max(result)
            output[index] = "Bronze Medal"

        count = 4
        while result:
            value, index = heapq._heappop_max(result)
            output[index] = "" + str(count)
            count += 1

        return output
```

## [2094 Remove Stones to Minimize the Total \(link\)](#)

### Description

You are given a **0-indexed** integer array `piles`, where `piles[i]` represents the number of stones in the  $i^{\text{th}}$  pile, and an integer `k`. You should apply the following operation **exactly** `k` times:

- Choose any `piles[i]` and **remove**  $\text{floor}(piles[i] / 2)$  stones from it.

**Notice** that you can apply the operation on the **same** pile more than once.

Return *the minimum possible total number of stones remaining after applying the `k` operations*.

`floor(x)` is the **largest** integer that is **smaller** than or **equal** to `x` (i.e., rounds `x` down).

#### Example 1:

**Input:** `piles = [5,4,9]`, `k = 2`  
**Output:** 12

**Explanation:** Steps of a possible scenario are:

- Apply the operation on pile 2. The resulting piles are `[5,4,5]`.
  - Apply the operation on pile 0. The resulting piles are `[3,4,5]`.
- The total number of stones in `[3,4,5]` is 12.

#### Example 2:

**Input:** `piles = [4,3,6,7]`, `k = 3`  
**Output:** 12

**Explanation:** Steps of a possible scenario are:

- Apply the operation on pile 2. The resulting piles are `[4,3,3,7]`.
  - Apply the operation on pile 3. The resulting piles are `[4,3,3,4]`.
  - Apply the operation on pile 0. The resulting piles are `[2,3,3,4]`.
- The total number of stones in `[2,3,3,4]` is 12.

#### Constraints:

- $1 \leq \text{piles.length} \leq 10^5$
- $1 \leq \text{piles}[i] \leq 10^4$
- $1 \leq k \leq 10^5$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    import heapq
    def minStoneSum(self, piles: List[int], k: int) -> int:
        maxPiles = [-i for i in piles]
        heapq.heapify(maxPiles)
        for i in range(k):
            temp = -heapq.heappop(maxPiles)
            val = temp // 2
            heapq.heappush(maxPiles, -(temp - val))
        return -sum(maxPiles)
```

## [1464 Reduce Array Size to The Half \(link\)](#)

### Description

You are given an integer array `arr`. You can choose a set of integers and remove all the occurrences of these integers in the array.

Return *the minimum size of the set so that at least half of the integers of the array are removed*.

#### Example 1:

**Input:** arr = [3,3,3,3,5,5,5,2,2,7]

**Output:** 2

**Explanation:** Choosing {3,7} will make the new array [5,5,5,2,2] which has size 5 (i.e equal to Possible sets of size 2 are {3,5},{3,2},{5,2}.

Choosing set {2,7} is not possible as it will make the new array [3,3,3,3,5,5,5] which has a size 7.

#### Example 2:

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

**Output:** 1

**Explanation:** The only possible set you can choose is {7}. This will make the new array empty.

#### Constraints:

- $2 \leq \text{arr.length} \leq 10^5$
- `arr.length` is even.
- $1 \leq \text{arr}[i] \leq 10^5$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def minSetSize(self, arr: List[int]) -> int:
        initial_length = len(arr)
        target_length = initial_length // 2
        hashmap = collections.defaultdict(int)
        for num in arr:
            hashmap[num] += 1

        storage = []
        for num, frequency in hashmap.items():
            storage.append((frequency, num))

        heapq._heapify_max(storage)
        removed_elements = 0
        while initial_length > target_length:
            frequency, num = heapq._heappop_max(storage)
            initial_length -= frequency
            removed_elements += 1
        return removed_elements
```

## [451 Sort Characters By Frequency \(link\)](#)

### Description

Given a string  $s$ , sort it in **decreasing order** based on the **frequency** of the characters. The **frequency** of a character is the number of times it appears in the string.

Return *the sorted string*. If there are multiple answers, return *any of them*.

#### Example 1:

**Input:**  $s = \text{"tree"}$

**Output:**  $\text{"eert"}$

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

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

#### Example 2:

**Input:**  $s = \text{"cccaaa"}$

**Output:**  $\text{"aaaccc"}$

**Explanation:** Both 'c' and 'a' appear three times, so both "cccaaa" and "aaaccc" are valid answers. Note that "cacaca" is incorrect, as the same characters must be together.

#### Example 3:

**Input:**  $s = \text{"Aabb"}$

**Output:**  $\text{"bbAa"}$

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

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

### Constraints:

- $1 \leq s.length \leq 5 * 10^5$
- $s$  consists of uppercase and lowercase English letters and digits.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    import heapq
    def frequencySort(self, s: str) -> str:
        hashmap = collections.defaultdict(int)
        for ch in s:
            hashmap[ch] += 1

        result = []
        for key, value in hashmap.items():
            result.append((value, key))

        heapq._heapify_max(result)
        result_string = ""

        while len(result) != 0:
            char_freq = heapq._heappop_max(result)
            char, freq = char_freq[0], char_freq[1]
            result_string += char * freq

        return result_string
```

## [1463 The K Weakest Rows in a Matrix \(link\)](#)

### Description

You are given an  $m \times n$  binary matrix  $\text{mat}$  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:

- $m == \text{mat.length}$
- $n == \text{mat[i].length}$
- $2 \leq n, m \leq 100$
- $1 \leq k \leq m$
- $\text{matrix[i][j]}$  is either 0 or 1.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def kWeakestRows(self, mat: List[List[int]], k: int) -> List[int]:
        arr = [(sum(mat[i]),i) for i in range(len(mat))]
        arr.sort()
        return [arr[i][1] for i in range(k)]
```

## [17 Letter Combinations of a Phone Number \(link\)](#)

### Description

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

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



#### Example 1:

```
Input: digits = "23"
Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"]
```

#### Example 2:

```
Input: digits = "2"
Output: ["a", "b", "c"]
```

#### Constraints:

- $1 \leq \text{digits.length} \leq 4$
- $\text{digits}[i]$  is a digit in the range  $['2', '9']$ .

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        if not digits:
            return []
        number_dict = {
            '0': '',
            '1': '',
            '2': "abc",
            '3': "def",
            '4': "ghi",
            '5': "jkl",
            '6': "mno",
            '7': "pqrs",
            '8': "tuv",
            '9': "wxyz"
        }
        result = []

        def recursive_string_generation(index, curr_string):
            if index > len(digits):
                return
            if index == len(digits):
                result.append(curr_string)
                return
            curr_string_list = number_dict[digits[index]]
            for char in curr_string_list:
                new_string = curr_string + char
                recursive_string_generation(index+1, new_string)
        recursive_string_generation(0, "")
        return result
```

## [3226 Minimum Number Game \(link\)](#)

### Description

You are given a **0-indexed** integer array `nums` of **even** length and there is also an empty array `arr`. Alice and Bob decided to play a game where in every round Alice and Bob will do one move. The rules of the game are as follows:

- Every round, first Alice will remove the **minimum** element from `nums`, and then Bob does the same.
- Now, first Bob will append the removed element in the array `arr`, and then Alice does the same.
- The game continues until `nums` becomes empty.

Return *the resulting array arr*.

#### Example 1:

**Input:** `nums` = [5,4,2,3]  
**Output:** [3,2,5,4]

**Explanation:** In round one, first Alice removes 2 and then Bob removes 3. Then in `arr` firstly Bob appends 3. At the begining of round two, `nums` = [5,4]. Now, first Alice removes 4 and then Bob removes 5. Then in `arr` firstly Alice appends 4.

#### Example 2:

**Input:** `nums` = [2,5]  
**Output:** [5,2]

**Explanation:** In round one, first Alice removes 2 and then Bob removes 5. Then in `arr` firstly Bob appends 5.

#### Constraints:

- $2 \leq \text{nums.length} \leq 100$
- $1 \leq \text{nums}[i] \leq 100$
- $\text{nums.length} \% 2 == 0$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    import heapq
    def numberGame(self, nums: List[int]) -> List[int]:
        alice_turn, bob_turn = None, None
        arr = []
        heapq.heapify(nums) # in place and in linear time
        for i in range(len(nums) // 2):
            alice_turn = heapq.heappop(nums)
            bob_turn = heapq.heappop(nums)
            arr.append(bob_turn)
            arr.append(alice_turn)
        return arr
```

## [1574 Maximum Product of Two Elements in an Array \(link\)](#)

### Description

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 of  $(4-1)*(5-1) = 12$ .

#### 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:

- $2 \leq \text{nums.length} \leq 500$
- $1 \leq \text{nums}[i] \leq 10^3$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    import heapq
    def maxProduct(self, nums: List[int]) -> int:
        heapq._heapify_max(nums)
        num1 = heapq._heappop_max(nums)
        num2 = heapq._heappop_max(nums)
        return (num1-1)*(num2-1)
```

## [494 Target Sum \(link\)](#)

### Description

You are given an integer array `nums` and an integer `target`.

You want to build an **expression** out of `nums` by adding one of the symbols `'+'` and `'-'` before each integer in `nums` and then concatenate all the integers.

- For example, if `nums = [2, 1]`, you can add a `'+'` before 2 and a `'-'` before 1 and concatenate them to build the expression `"+2-1"`.

Return the number of different **expressions** that you can build, which evaluates to `target`.

#### Example 1:

```
Input: nums = [1,1,1,1,1], target = 3
Output: 5
Explanation: There are 5 ways to assign symbols to make the sum of nums be target 3.
-1 + 1 + 1 + 1 + 1 = 3
+1 - 1 + 1 + 1 + 1 = 3
+1 + 1 - 1 + 1 + 1 = 3
+1 + 1 + 1 - 1 + 1 = 3
+1 + 1 + 1 + 1 - 1 = 3
```

#### Example 2:

```
Input: nums = [1], target = 1
Output: 1
```

#### Constraints:

- $1 \leq \text{nums.length} \leq 20$
- $0 \leq \text{nums}[i] \leq 1000$
- $0 \leq \text{sum}(\text{nums}[i]) \leq 1000$
- $-1000 \leq \text{target} \leq 1000$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def findTargetSumWays(self, nums: List[int], target: int) -> int:
        dic = {}
        def dfs(index, total):
            key = (index, total)
            if key not in dic:
                if index == len(nums):
                    return 1 if total == target else 0
                else:
                    dic[key] = dfs(index+1, total + nums[index]) + dfs(index+1, total - nums[index])
            return dic[key]
        return dfs(0, 0)
```

## [47 Permutations II \(link\)](#)

### Description

Given a collection of numbers, `nums`, that might contain duplicates, return *all possible unique permutations in any order*.

#### Example 1:

```
Input: nums = [1,1,2]
Output:
[[1,1,2],
 [1,2,1],
 [2,1,1]]
```

#### Example 2:

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

#### Constraints:

- $1 \leq \text{nums.length} \leq 8$
- $-10 \leq \text{nums}[i] \leq 10$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def permuteUnique(self, nums: List[int]) -> List[List[int]]:
        result = set()
        def generate_perms(index: int, curr_perm: list):
            if index == len(nums):
                result.add(tuple(curr_perm.copy()))
                return
            visited = set()
            for i in range(index, len(nums)):
                if curr_perm[i] not in visited:
                    visited.add(curr_perm[i])
                    curr_perm[i], curr_perm[index] = curr_perm[index], curr_perm[i]
                    generate_perms(index + 1, curr_perm)
                    curr_perm[i], curr_perm[index] = curr_perm[index], curr_perm[i]

        generate_perms(0, nums)
        return list(result)
```

## [46 Permutations \(link\)](#)

### Description

Given an array `nums` of distinct integers, return all the possible permutations. You can return the answer in **any order**.

#### Example 1:

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

#### Example 2:

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

#### Example 3:

```
Input: nums = [1]
Output: [[1]]
```

### Constraints:

- $1 \leq \text{nums.length} \leq 6$
- $-10 \leq \text{nums}[i] \leq 10$
- All the integers of `nums` are **unique**.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        result = []
        def generate_permutation(curr_perm: list, index: int):
            if index == len(nums):
                result.append(curr_perm.copy())
                return

            for i in range(index, len(nums)):
                curr_perm[index], curr_perm[i] = curr_perm[i], curr_perm[index]
                generate_permutation(curr_perm, index+1)
                curr_perm[index], curr_perm[i] = curr_perm[i], curr_perm[index]

        generate_permutation(nums, 0)
        return result
```

## [78 Subsets \(link\)](#)

### Description

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

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

#### Example 1:

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

#### Example 2:

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

#### Constraints:

- $1 \leq \text{nums.length} \leq 10$
- $-10 \leq \text{nums}[i] \leq 10$
- All the numbers of `nums` are **unique**.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        result = [[]]
        def generate_subsets(index: int, current_subset):
            if index >= len(nums):
                return
            current_subset.append(nums[index])
            result.append(current_subset.copy())
            generate_subsets(index+1, current_subset)
            current_subset.pop()
            generate_subsets(index+1, current_subset)
        generate_subsets(0, [])
        return result
```

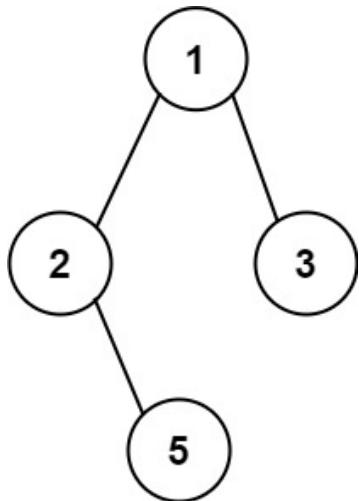
## [257 Binary Tree Paths \(link\)](#)

### Description

Given the root of a binary tree, return *all root-to-leaf paths in any order*.

A **leaf** is a node with no children.

#### Example 1:



```
Input: root = [1,2,3,null,5]
Output: ["1->2->5","1->3"]
```

#### Example 2:

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

### Constraints:

- The number of nodes in the tree is in the range [1, 100].
- $-100 \leq \text{Node.val} \leq 100$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def binaryTreePaths(self, root: Optional[TreeNode]) -> List[str]:
        if not root:
            return []
        result = []
        def generate_path(curr_node, curr_path: str):
            # If current node is a leaf node
            if not curr_node.left and not curr_node.right:
                curr_path += str(curr_node.val) if curr_path == "" else ("->" + str(curr_node.val))
                result.append(curr_path)
                return
            # generate the current string
            curr_path += str(curr_node.val) if curr_path == "" else ("->" + str(curr_node.val))

            # If left node exists
            if curr_node.left:
                generate_path(curr_node.left, curr_path)

            # If right node exists
            if curr_node.right:
                generate_path(curr_node.right, curr_path)

        # Call the function
        generate_path(root, "")
        return result
```

## [39 Combination Sum \(link\)](#)

### Description

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

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

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

#### Example 1:

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

##### Explanation:

2 and 3 are candidates, and  $2 + 2 + 3 = 7$ . Note that 2 can be used multiple times.  
7 is a candidate, and  $7 = 7$ .  
These are the only two combinations.

#### Example 2:

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

#### Example 3:

```
Input: candidates = [2], target = 1
Output: []
```

### Constraints:

- $1 \leq \text{candidates.length} \leq 30$
- $2 \leq \text{candidates}[i] \leq 40$
- All elements of `candidates` are **distinct**.
- $1 \leq \text{target} \leq 40$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:
        result = []
        def generate_combination(index, curr_target, current_selection):
            # If ideal condition is met
            if curr_target == target:
                result.append(current_selection.copy())
                return

            # if over the target
            if curr_target > target:
                return

            # if no more candidates are left
            if index >= len(candidates):
                return

            # choose the current index element
            current_selection.append(candidates[index])
            generate_combination(index, curr_target + candidates[index], current_selection)

            # not choose the current index element
            current_selection.pop()
            generate_combination(index + 1, curr_target, current_selection)

        generate_combination(0, 0, [])
        return result
```

## [820 Find Eventual Safe States \(link\)](#)

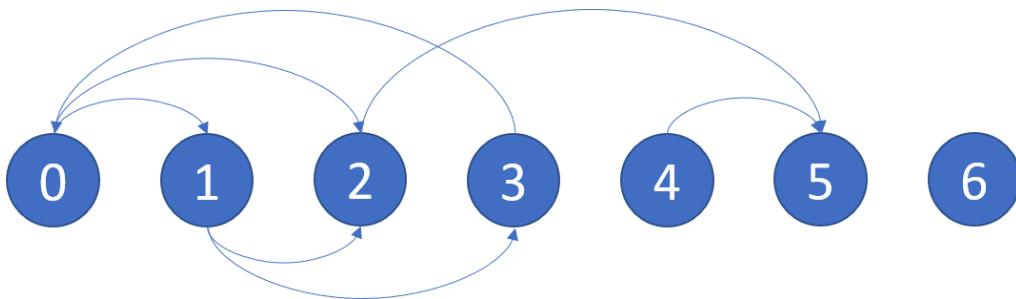
### Description

There is a directed graph of  $n$  nodes with each node labeled from  $0$  to  $n - 1$ . The graph is represented by a **0-indexed** 2D integer array  $\text{graph}$  where  $\text{graph}[i]$  is an integer array of nodes adjacent to node  $i$ , meaning there is an edge from node  $i$  to each node in  $\text{graph}[i]$ .

A node is a **terminal node** if there are no outgoing edges. A node is a **safe node** if every possible path starting from that node leads to a **terminal node** (or another safe node).

Return *an array containing all the safe nodes of the graph*. The answer should be sorted in **ascending** order.

#### Example 1:



**Input:**  $\text{graph} = [[1,2],[2,3],[5],[0],[5],[],[]]$   
**Output:**  $[2,4,5,6]$

**Explanation:** The given graph is shown above.  
 Nodes 5 and 6 are terminal nodes as there are no outgoing edges from either of them.  
 Every path starting at nodes 2, 4, 5, and 6 all lead to either node 5 or 6.

#### Example 2:

**Input:**  $\text{graph} = [[1,2,3,4],[1,2],[3,4],[0,4],[]]$   
**Output:**  $[4]$

**Explanation:**  
 Only node 4 is a terminal node, and every path starting at node 4 leads to node 4.

#### Constraints:

- $n == \text{graph.length}$
- $1 \leq n \leq 10^4$
- $0 \leq \text{graph}[i].length \leq n$
- $0 \leq \text{graph}[i][j] \leq n - 1$
- $\text{graph}[i]$  is sorted in a strictly increasing order.
- The graph may contain self-loops.
- The number of edges in the graph will be in the range  $[1, 4 * 10^4]$ .

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def eventualSafeNodes(self, graph: List[List[int]]) -> List[int]:
        adj_list = collections.defaultdict(list)
        indegree = [0] * len(graph)
        outdegree = [0] * len(graph)

        # create graph and calculate outdegree
        for start_edge in range(len(graph)):
            for end_edge in graph[start_edge]:
                indegree[end_edge] += 1
                adj_list[end_edge].append(start_edge)
                outdegree[start_edge] += 1

        # get all elements with outdegree 0
        queue = collections.deque()
        for node, node_outdegree in enumerate(outdegree):
            if node_outdegree == 0:
                queue.append(node)

        ans = collections.deque()
        # get to all reachable
        while queue:
            curr = queue.popleft()
            ans.appendleft(curr)
            for nei in adj_list[curr]:
                outdegree[nei] -= 1
                if outdegree[nei] == 0:
                    queue.append(nei)

        return sorted(list(ans))
```

## [139 Word Break \(link\)](#)

### Description

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

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

#### Example 1:

```
Input: s = "leetcode", wordDict = ["leet", "code"]
Output: true
Explanation: Return true because "leetcode" can be segmented as "leet code".
```

#### Example 2:

```
Input: s = "applepenapple", wordDict = ["apple", "pen"]
Output: true
Explanation: Return true because "applepenapple" can be segmented as "apple pen apple".
Note that you are allowed to reuse a dictionary word.
```

#### Example 3:

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

### Constraints:

- $1 \leq s.length \leq 300$
- $1 \leq \text{wordDict.length} \leq 1000$
- $1 \leq \text{wordDict}[i].length \leq 20$
- $s$  and  $\text{wordDict}[i]$  consist of only lowercase English letters.
- All the strings of  $\text{wordDict}$  are **unique**.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> bool:
        cache = {}
        def recursive_search(substring):
            if substring in cache:
                return cache[substring]
            if len(substring) == 0:
                cache[substring] = True
                return True
            for word in wordDict:
                if substring.startswith(word):
                    newsubstring = substring[len(word):]
                    if recursive_search(newsubstring):
                        cache[substring] = True
                        return True
            cache[substring] = False
            return cache[substring]
        return recursive_search(s)
```

## [322 Coin Change \(link\)](#)

### Description

You are given an integer array `coins` representing coins of different denominations and an integer `amount` representing a total amount of money.

Return *the fewest number of coins that you need to make up that amount*. If that amount of money cannot be made up by any combination of the coins, return `-1`.

You may assume that you have an infinite number of each kind of coin.

#### Example 1:

```
Input: coins = [1,2,5], amount = 11
Output: 3
Explanation: 11 = 5 + 5 + 1
```

#### Example 2:

```
Input: coins = [2], amount = 3
Output: -1
```

#### Example 3:

```
Input: coins = [1], amount = 0
Output: 0
```

### Constraints:

- $1 \leq \text{coins.length} \leq 12$
- $1 \leq \text{coins}[i] \leq 2^{31} - 1$
- $0 \leq \text{amount} \leq 10^4$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def coinChange(self, coins: List[int], amount: int) -> int:
        cache = {}
        def getNumberOfCoins(target):
            if target in cache:
                return cache[target]
            if target == 0:
                cache[target] = 0
                return cache[target]
            if target < 0:
                cache[target] = float('inf')
                return cache[target]
            min_num_coins = float('inf')
            for coin in coins:
                number_of_coins = 1 + getNumberOfCoins(target - coin)
                min_num_coins = min(min_num_coins, number_of_coins)
            cache[target] = min_num_coins
        return cache[target]

    min_coins = getNumberOfCoins(amount)
    if min_coins == float('inf'):
        return -1
    return min_coins
```

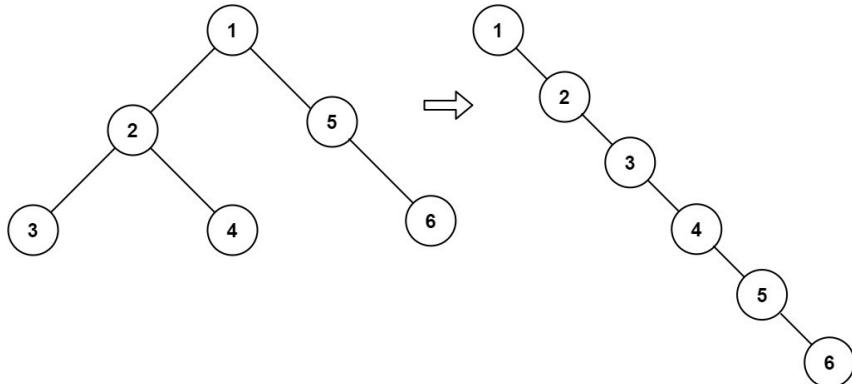
## [114 Flatten Binary Tree to Linked List \(link\)](#)

### Description

Given the root of a binary tree, flatten the tree into a "linked list":

- The "linked list" should use the same `TreeNode` class where the `right` child pointer points to the next node in the list and the `left` child pointer is always `null`.
- The "linked list" should be in the same order as a [pre-order traversal](#) of the binary tree.

#### Example 1:



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

#### Example 2:

```
Input: root = []
Output: []
```

#### Example 3:

```
Input: root = [0]
Output: [0]
```

#### Constraints:

- The number of nodes in the tree is in the range  $[0, 2000]$ .
- $-100 \leq \text{Node.val} \leq 100$

**Follow up:** Can you flatten the tree in-place (with  $O(1)$  extra space)?

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    from collections import deque
    def flatten(self, root: Optional[TreeNode]) -> None:
        """
        Do not return anything, modify root in-place instead.
        """
        curr = root
        while curr:
            if curr.left:
                prev = curr.left
                while prev.right:
                    prev = prev.right
                prev.right = curr.right
                curr.right = curr.left
                curr.left = None
            curr = curr.right
```

## [33 Search in Rotated Sorted Array \(link\)](#)

### Description

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

Prior to being passed to your function, `nums` is **possibly left rotated** at an unknown index `k` ( $1 \leq k < \text{nums.length}$ ) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,5,6,7]` might be left rotated by 3 indices and become `[4,5,6,7,0,1,2]`.

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

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

#### Example 1:

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

#### Example 2:

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

#### Example 3:

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

#### Constraints:

- $1 \leq \text{nums.length} \leq 5000$
- $-10^4 \leq \text{nums}[i] \leq 10^4$
- All values of `nums` are **unique**.
- `nums` is an ascending array that is possibly rotated.
- $-10^4 \leq \text{target} \leq 10^4$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        if len(nums) == 1:
            return 0 if nums[0] == target else -1
        def findPivotElement():
            left, right = 0, len(nums) - 1
            mid = (left + right) // 2

            while left <= right:
                mid = (left + right) // 2
                if nums[mid] > nums[mid+1]:
                    return mid
                elif nums[mid] > nums[len(nums) - 1]:
                    left = mid + 1
                elif nums[mid] < nums[0]:
                    right = mid - 1
            return -1

        def binarysearch(nums, left, right, target):
            mid = (left + right) // 2
            while left <= right:
                mid = (left + right) // 2
                if nums[mid] == target:
                    return mid
                elif nums[mid] > target:
                    right = mid - 1
                elif nums[mid] < target:
                    left = mid + 1
            return -1

        def is_already_sorted():
            if nums[0] <= nums[-1]:
                return True
            return False

        if is_already_sorted():
            return binarysearch(nums, 0, len(nums)-1, target)
        pivot = findPivotElement()
        left = binarysearch(nums, 0, pivot, target)

        if left == -1:
            right = binarysearch(nums, pivot + 1, len(nums) - 1, target)
            return right
        return left
```

## [127 Word Ladder \(link\)](#)

### Description

A **transformation sequence** from word `beginWord` to word `endWord` using a dictionary `wordList` is a sequence of words  $beginWord \rightarrow s_1 \rightarrow s_2 \rightarrow \dots \rightarrow s_k$  such that:

- Every adjacent pair of words differs by a single letter.
- Every  $s_i$  for  $1 \leq i \leq k$  is in `wordList`. Note that `beginWord` does not need to be in `wordList`.
- $s_k == endWord$

Given two words, `beginWord` and `endWord`, and a dictionary `wordList`, return *the number of words in the shortest transformation sequence* from `beginWord` to `endWord`, or 0 if no such sequence exists.

#### Example 1:

```
Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]
Output: 5
Explanation: One shortest transformation sequence is "hit" -> "hot" -> "dot" -> "dog" -> cog",
```

#### Example 2:

```
Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log"]
Output: 0
Explanation: The endWord "cog" is not in wordList, therefore there is no valid transformation sequence.
```

#### Constraints:

- $1 \leq beginWord.length \leq 10$
- $endWord.length == beginWord.length$
- $1 \leq wordList.length \leq 5000$
- $wordList[i].length == beginWord.length$
- `beginWord`, `endWord`, and `wordList[i]` consist of lowercase English letters.
- $beginWord \neq endWord$
- All the words in `wordList` are **unique**.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    from collections import deque
    def ladderLength(self, beginWord: str, endWord: str, wordList: List[str]) -> int:
        if endWord not in wordList:
            return 0

        neighbors = defaultdict(list)
        wordList.append(beginWord)

        for word in wordList:
            for i in range(len(word)):
                pattern = word[:i] + "*" + word[i+1:]
                neighbors[pattern].append(word)

        visited = set([beginWord])
        queue = deque([(beginWord, 1)])

        while queue:
            for i in range(len(queue)):
                curr = queue.popleft()
                word, dist = curr[0], curr[1]
                if word == endWord:
                    return dist

                for j in range(len(word)):
                    pattern = word[:j] + "*" + word[j+1:]
                    for reachable_word in neighbors[pattern]:
                        if reachable_word not in visited:
                            visited.add(reachable_word)
                            queue.append((reachable_word, dist+1))

        return 0
```

## 34 Find First and Last Position of Element in Sorted Array [\(link\)](#)

### Description

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

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

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

#### Example 1:

```
Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]
```

#### Example 2:

```
Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]
```

#### Example 3:

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

### Constraints:

- $0 \leq \text{nums.length} \leq 10^5$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- `nums` is a non-decreasing array.
- $-10^9 \leq \text{target} \leq 10^9$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def searchRange(self, nums: List[int], target: int) -> List[int]:
        def binarysearch(nums, target, direction = "left"):
            left, right = 0, len(nums) - 1
            index = -1

            while left <= right:
                mid = (left + right) // 2
                if nums[mid] > target:
                    right = mid - 1
                elif nums[mid] < target:
                    left = mid + 1
                else:
                    index = mid
                    if direction == "left":
                        right = mid - 1
                    else:
                        left = mid + 1
            return index
        left = binarysearch(nums, target, "left")
        right = binarysearch(nums, target, "right")

        return [left, right]
```

## 28 Find the Index of the First Occurrence in a String [\(link\)](#)

### Description

Given two strings `needle` and `haystack`, return the index of the first occurrence of `needle` in `haystack`, or `-1` if `needle` is not part of `haystack`.

#### Example 1:

```
Input: haystack = "sadbutsad", needle = "sad"
Output: 0
Explanation: "sad" occurs at index 0 and 6.
The first occurrence is at index 0, so we return 0.
```

#### Example 2:

```
Input: haystack = "leetcode", needle = "leeto"
Output: -1
Explanation: "leeto" did not occur in "leetcode", so we return -1.
```

#### Constraints:

- $1 \leq \text{haystack.length}, \text{needle.length} \leq 10^4$
- `haystack` and `needle` consist of only lowercase English characters.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:
        for i in range(len(haystack) - len(needle) + 1):
            substring = haystack[i: i + len(needle)]
            if substring == needle:
                return i
        return -1
```

## 27 Remove Element (link)

### Description

Given an integer array `nums` and an integer `val`, remove all occurrences of `val` in `nums` **in-place**. The order of the elements may be changed. Then return *the number of elements in `nums` which are not equal to `val`*.

Consider the number of elements in `nums` which are not equal to `val` be `k`, to get accepted, you need to do the following things:

- Change the array `nums` such that the first `k` elements of `nums` contain the elements which are not equal to `val`. The remaining elements of `nums` are not important as well as the size of `nums`.
- Return `k`.

#### Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int val = ...; // Value to remove
int[] expectedNums = [...]; // The expected answer with correct length.
                           // It is sorted with no values equaling val.

int k = removeElement(nums, val); // Calls your implementation

assert k == expectedNums.length;
sort(nums, 0, k); // Sort the first k elements of nums
for (int i = 0; i < actualLength; i++) {
    assert nums[i] == expectedNums[i];
}
```

If all assertions pass, then your solution will be **accepted**.

#### Example 1:

**Input:** `nums = [3,2,2,3]`, `val = 3`  
**Output:** `2`, `nums = [2,2,_,_]`  
**Explanation:** Your function should return `k = 2`, with the first two elements of `nums` being `2`. It does not matter what you leave beyond the returned `k` (hence they are underscores).

#### Example 2:

**Input:** `nums = [0,1,2,2,3,0,4,2]`, `val = 2`  
**Output:** `5`, `nums = [0,1,4,0,3,_,_,_]`  
**Explanation:** Your function should return `k = 5`, with the first five elements of `nums` containing `0, 1, 4, 0, 3`. Note that the five elements can be returned in any order. It does not matter what you leave beyond the returned `k` (hence they are underscores).

#### Constraints:

- $0 \leq \text{nums.length} \leq 100$
- $0 \leq \text{nums}[i] \leq 50$
- $0 \leq \text{val} \leq 100$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:
        if nums == []:
            return 0
        left, right, count = 0, len(nums) - 1, len(nums)
        while right > -1 and nums[right] == val:
            right -= 1
            count -= 1

        while left < right:
            if nums[left] == val:
                nums[right], nums[left] = nums[left], nums[right]
                while nums[right] == val and right > left:
                    right -= 1
                    count -= 1
            left += 1
        return count
```

## 21 Merge Two Sorted Lists (link)

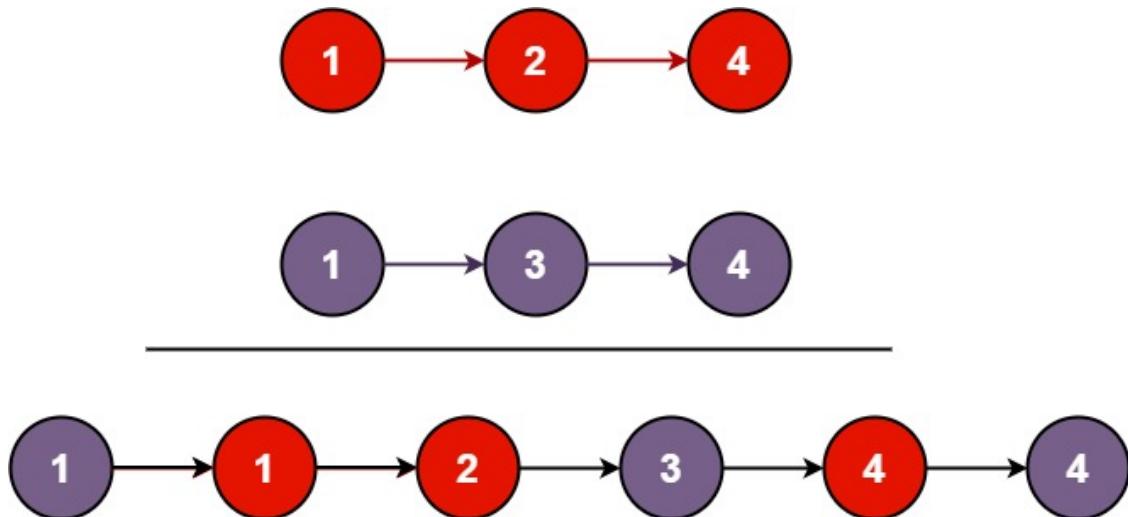
### Description

You are given the heads of two sorted linked lists `list1` and `list2`.

Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list*.

#### Example 1:



```
Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]
```

#### Example 2:

```
Input: list1 = [], list2 = []
Output: []
```

#### Example 3:

```
Input: list1 = [], list2 = [0]
Output: [0]
```

#### Constraints:

- The number of nodes in both lists is in the range  $[0, 50]$ .
- $-100 \leq \text{Node.val} \leq 100$
- Both `list1` and `list2` are sorted in **non-decreasing** order.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def mergeTwoLists(self, list1: Optional[ListNode], list2: Optional[ListNode]) -> Optional[ListNode]:
        if not list1 and not list2:
            return None

        dummy = ListNode(0)
        head = dummy
        while list1 and list2:
            if list1.val < list2.val:
                dummy.next = ListNode(list1.val)
                list1 = list1.next
            else:
                dummy.next = ListNode(list2.val)
                list2 = list2.next
            dummy = dummy.next

        if not list2:
            while list1:
                dummy.next = ListNode(list1.val)
                dummy = dummy.next
                list1 = list1.next

        if not list1:
            while list2:
                dummy.next = ListNode(list2.val)
                dummy = dummy.next
                list2 = list2.next
        return head.next
```

## [5 Longest Palindromic Substring \(link\)](#)

### 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:

- $1 \leq s.length \leq 1000$
- $s$  consist of only digits and English letters.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def longestPalindrome(self, s: str) -> str:
        if len(s) <= 1:
            return s
        def expand_from_center(left, right):
            while left >= 0 and right < len(s) and s[left] == s[right]:
                left -= 1
                right += 1
            return s[left + 1: right]

        max_str = s[0]

        for i in range(len(s) - 1):
            odd = expand_from_center(i, i)
            even = expand_from_center(i, i+1)

            if len(odd) > len(max_str):
                max_str = odd
            if len(even) > len(max_str):
                max_str = even

        return max_str
```

## [647 Palindromic Substrings \(link\)](#)

### Description

Given a string  $s$ , return *the number of palindromic substrings in it*.

A string is a **palindrome** when it reads the same backward as forward.

A **substring** is a contiguous sequence of characters within the string.

#### Example 1:

```
Input: s = "abc"
Output: 3
Explanation: Three palindromic strings: "a", "b", "c".
```

#### Example 2:

```
Input: s = "aaa"
Output: 6
Explanation: Six palindromic strings: "a", "a", "a", "aa", "aa", "aaa".
```

#### Constraints:

- $1 \leq s.length \leq 1000$
- $s$  consists of lowercase English letters.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def countSubstrings(self, s: str) -> int:
        cache = {}
        def isPal(left, right):
            if (left, right) in cache:
                return cache[(left, right)]
            if left > right:
                cache[(left, right)] = True
                return True
            if s[left] != s[right]:
                cache[(left, right)] = False
                return False
            cache[(left, right)] = isPal(left + 1, right - 1)
            return cache[(left, right)]

        count = 0
        for i in range(len(s)):
            for j in range(i, len(s)):
                if isPal(i, j):
                    count += 1
        return count
```

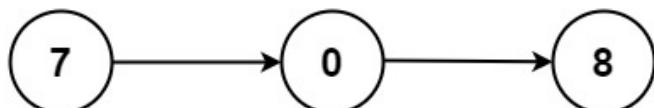
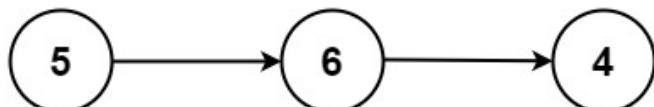
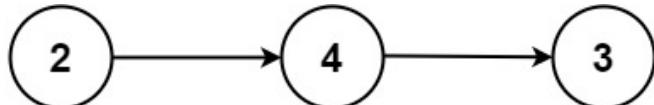
## [2 Add Two Numbers \(link\)](#)

### 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].
- $0 \leq \text{Node.val} \leq 9$
- It is guaranteed that the list represents a number that does not have leading zeros.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def addTwoNumbers(self, l1: Optional[ListNode], l2: Optional[ListNode]) -> Optional[ListNode]:
        carry_flag = 0
        headNode = ListNode(0)
        dummyNode = headNode
        while l1 and l2:
            curr_val = l1.val + l2.val
            curr_val = curr_val + carry_flag
            if curr_val > 9:
                carry_flag = 1
                curr_val = curr_val % 10
            else:
                carry_flag = 0
            curr_node = ListNode(curr_val)
            dummyNode.next = curr_node

            l1 = l1.next
            l2 = l2.next
            dummyNode = dummyNode.next

        while l1:
            curr_val = l1.val + carry_flag
            if curr_val > 9:
                carry_flag = 1
                curr_val = curr_val % 10
            else:
                carry_flag = 0
            curr_node = ListNode(curr_val)
            dummyNode.next = curr_node
            curr_node = curr_node.next
            l1 = l1.next
            dummyNode = dummyNode.next

        while l2:
            curr_val = l2.val + carry_flag
            if curr_val > 9:
                carry_flag = 1
                curr_val = curr_val % 10
            else:
                carry_flag = 0
            curr_node = ListNode(curr_val)
            dummyNode.next = curr_node
            curr_node = curr_node.next
            l2 = l2.next
            dummyNode = dummyNode.next

        if carry_flag:
            dummyNode.next = ListNode(carry_flag)

    return headNode.next
```

## [516 Longest Palindromic Subsequence \(link\)](#)

### Description

Given a string  $s$ , find *the longest palindromic subsequence's length in  $s$* .

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

#### Example 1:

```
Input: s = "bbbab"  
Output: 4  
Explanation: One possible longest palindromic subsequence is "bbbb".
```

#### Example 2:

```
Input: s = "cbbd"  
Output: 2  
Explanation: One possible longest palindromic subsequence is "bb".
```

#### Constraints:

- $1 \leq s.length \leq 1000$
- $s$  consists only of lowercase English letters.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def longestPalindromeSubseq(self, s: str) -> int:
        cache = {}
        def dp(left, right):
            if (left, right) in cache:
                return cache[(left, right)]
            elif left > right:
                return 0
            elif s[left] == s[right]:
                if left == right:
                    return 1 + dp(left+1, right-1)
                return 2 + dp(left + 1, right - 1)
            else:
                cache[(left, right)] = max(
                    dp(left + 1, right),
                    dp(left, right - 1)
                )
            return cache[(left, right)]
        return dp(0, len(s)-1)
```

## [1250 Longest Common Subsequence \(link\)](#)

### Description

Given two strings `text1` and `text2`, return *the length of their longest common subsequence*. If there is no **common subsequence**, return 0.

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".

A **common subsequence** of two strings is a subsequence that is common to both strings.

#### Example 1:

```
Input: text1 = "abcde", text2 = "ace"  
Output: 3
```

**Explanation:** The longest common subsequence is "ace" and its length is 3.

#### Example 2:

```
Input: text1 = "abc", text2 = "abc"  
Output: 3
```

**Explanation:** The longest common subsequence is "abc" and its length is 3.

#### Example 3:

```
Input: text1 = "abc", text2 = "def"  
Output: 0
```

**Explanation:** There is no such common subsequence, so the result is 0.

### Constraints:

- $1 \leq \text{text1.length}, \text{text2.length} \leq 1000$
- `text1` and `text2` consist of only lowercase English characters.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def longestCommonSubsequence(self, text1: str, text2: str) -> int:
        memo = {}
        def lcs(string1, string2, size1, size2):
            if memo.get((size1, size2)):
                return memo[(size1, size2)]
            if size1 == 0 or size2 == 0:
                memo[(size1, size2)] = 0
                return 0
            elif string1[size1 - 1] == string2[size2 - 1]:
                memo[(size1, size2)] = 1 + lcs(string1, string2, size1 - 1, size2 - 1)
            else:
                memo[(size1, size2)] = max(lcs(string1, string2, size1-1, size2), lcs(string1,
                return memo[(size1, size2)]

    def contains_any_common_char(text1, text2):
        set1, set2 = set(), set()
        for char in text1:
            set1.add(char)
        for char in text2:
            set2.add(char)
        return len(set1.intersection(set2)) > 0
    if not contains_any_common_char(text1, text2):
        return 0
    return lcs(text1, text2, len(text1), len(text2))
```

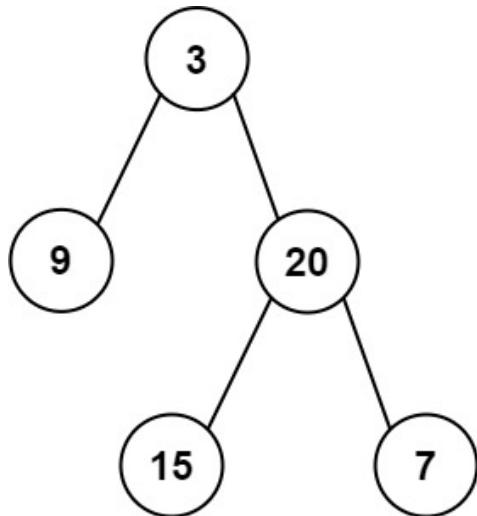
## [404 Sum of Left Leaves \(link\)](#)

### Description

Given the root of a binary tree, return *the sum of all left leaves*.

A **leaf** is a node with no children. A **left leaf** is a leaf that is the left child of another node.

#### Example 1:



**Input:** root = [3,9,20,null,null,15,7]

**Output:** 24

**Explanation:** There are two left leaves in the binary tree, with values 9 and 15 respectively.

#### Example 2:

**Input:** root = [1]

**Output:** 0

#### Constraints:

- The number of nodes in the tree is in the range [1, 1000].
- $-1000 \leq \text{Node.val} \leq 1000$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    from collections import deque
    def sumOfLeftLeaves(self, root: Optional[TreeNode]) -> int:
        if not root:
            return 0

        ans = 0
        if root.left:
            if not root.left.left and not root.left.right:
                ans += root.left.val
            else:
                ans += self.sumOfLeftLeaves(root.left)
        ans += self.sumOfLeftLeaves(root.right)
        return ans
```

## 130 Surrounded Regions (link)

### Description

You are given an  $m \times n$  matrix board containing **letters** 'X' and '0', **capture regions** that are **surrounded**:

- **Connect:** A cell is connected to adjacent cells horizontally or vertically.
- **Region:** To form a region **connect every** '0' cell.
- **Surround:** The region is surrounded with 'X' cells if you can **connect the region** with 'X' cells and none of the region cells are on the edge of the board.

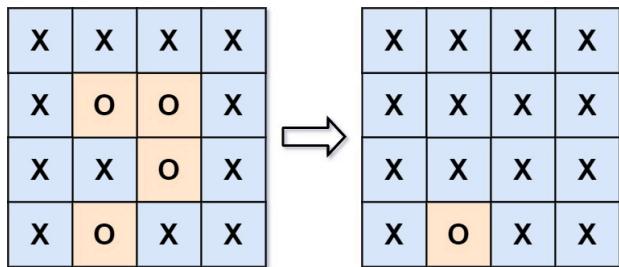
To capture a **surrounded region**, replace all '0's with 'X's **in-place** within the original board. You do not need to return anything.

#### Example 1:

**Input:** board = [["X", "X", "X", "X"], ["X", "O", "O", "X"], ["X", "X", "O", "X"], ["X", "O", "X", "X"]]

**Output:** [[["X", "X", "X", "X"], ["X", "X", "X", "X"], ["X", "X", "X", "X"], ["X", "O", "X", "X"]]]

#### Explanation:



In the above diagram, the bottom region is not captured because it is on the edge of the board and cannot be surrounded.

#### Example 2:

**Input:** board = [["X"]]

**Output:** [[["X"]]]

#### Constraints:

- $m == \text{board.length}$
- $n == \text{board}[i].length$
- $1 \leq m, n \leq 200$
- $\text{board}[i][j]$  is 'X' or '0'.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def solve(self, board: List[List[str]]) -> None:
        """
        Do not return anything, modify board in-place instead.
        """
        visited = set()
        def dfs(curr):
            row, col = curr[0], curr[1]
            if row < 0 or row >= len(board) or col < 0 or col >= len(board[0]) or (row, col) in visited:
                return
            visited.add((row, col))
            if board[row][col] == "0":
                board[row][col] = "M"

            dfs([row-1, col])
            dfs([row+1, col])
            dfs([row, col+1])
            dfs([row, col-1])

        # visit all corner edge with 0
        top_row, bottom_row, left_col, right_col = 0, len(board)-1, 0, len(board[0])-1
        for row in range(len(board)):
            if board[row][left_col] == "0":
                board[row][left_col] = "M"
            if board[row][right_col] == "0":
                board[row][right_col] = "M"

        for col in range(len(board[0])):
            if board[top_row][col] == "0":
                board[top_row][col] = "M"
            if board[bottom_row][col] == "0":
                board[bottom_row][col] = "M"

        # Convert all connected 0 to D
        for row in range(len(board)):
            for col in range(len(board[0])):
                if board[row][col] == "M":
                    dfs([row, col])

        # Convert rest of all 0 to X
        for row in range(len(board)):
            for col in range(len(board[0])):
                if board[row][col] == "0":
                    board[row][col] = "X"
        # Convert all D to 0
        for row in range(len(board)):
            for col in range(len(board[0])):
                if board[row][col] == "M":
                    board[row][col] = "0"
```

## [310 Minimum Height Trees \(link\)](#)

### Description

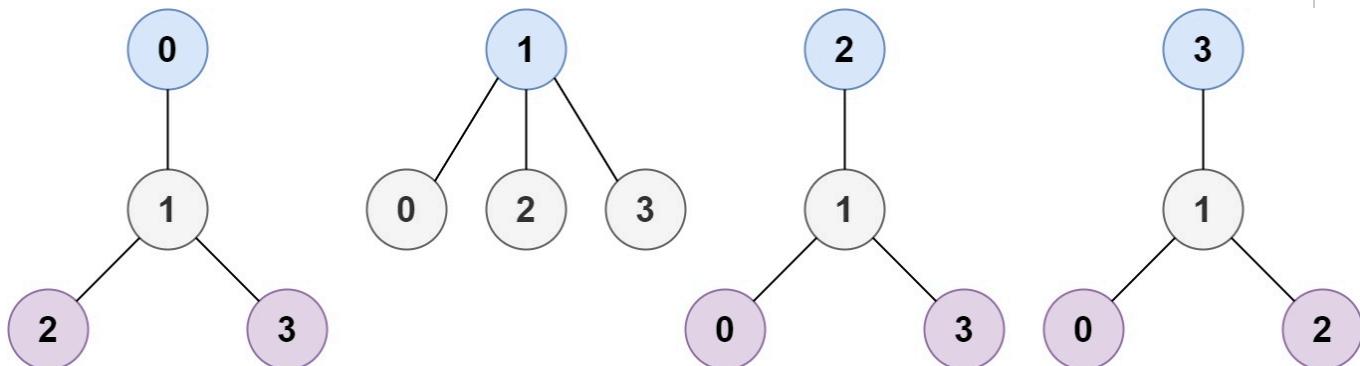
A tree is an undirected graph in which any two vertices are connected by *exactly* one path. In other words, any connected graph without simple cycles is a tree.

Given a tree of  $n$  nodes labelled from  $0$  to  $n - 1$ , and an array of  $n - 1$  edges where  $\text{edges}[i] = [a_i, b_i]$  indicates that there is an undirected edge between the two nodes  $a_i$  and  $b_i$  in the tree, you can choose any node of the tree as the root. When you select a node  $x$  as the root, the result tree has height  $h$ . Among all possible rooted trees, those with minimum height (i.e.  $\min(h)$ ) are called **minimum height trees** (MHTs).

Return a *list of all MHTs' root labels*. You can return the answer in **any order**.

The **height** of a rooted tree is the number of edges on the longest downward path between the root and a leaf.

#### Example 1:

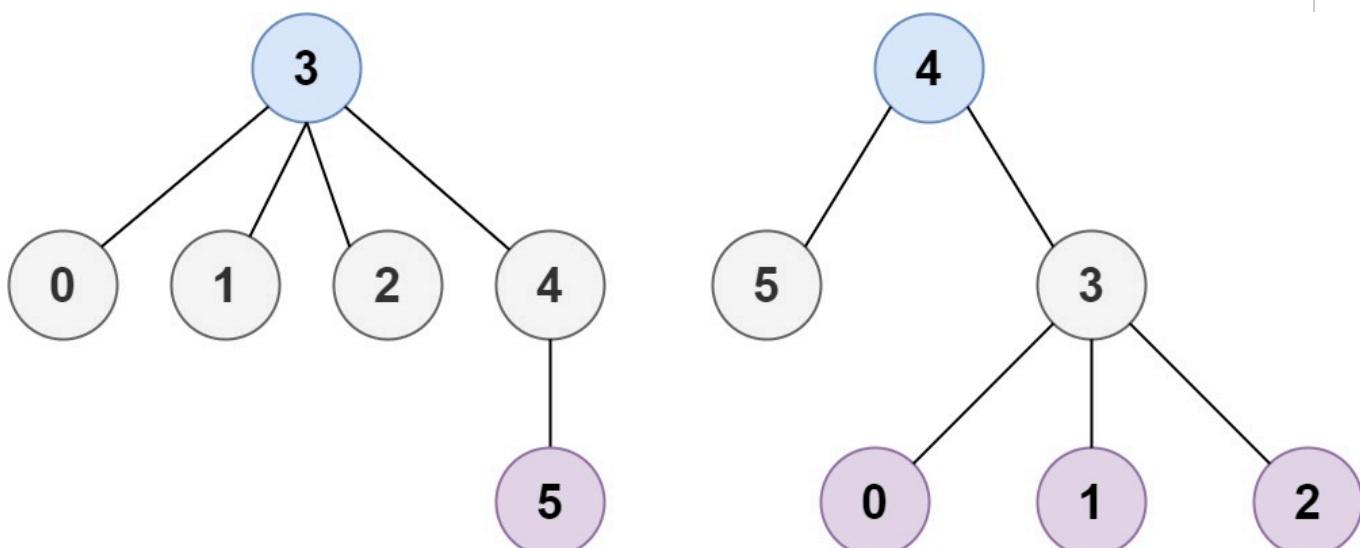


**Input:**  $n = 4$ ,  $\text{edges} = [[1,0], [1,2], [1,3]]$

**Output:** [1]

**Explanation:** As shown, the height of the tree is 1 when the root is the node with label 1 which

#### Example 2:



**Input:**  $n = 6$ ,  $\text{edges} = [[3,0], [3,1], [3,2], [3,4], [5,4]]$

**Output:** [3,4]

**Constraints:**

- $1 \leq n \leq 2 * 10^4$
- $\text{edges.length} == n - 1$
- $0 \leq a_i, b_i < n$
- $a_i \neq b_i$
- All the pairs  $(a_i, b_i)$  are distinct.
- The given input is **guaranteed** to be a tree and there will be **no repeated edges**.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    from collections import deque
    def findMinHeightTrees(self, n: int, edges: List[List[int]]) -> List[int]:
        if n == 1:
            return [0]
        graph = defaultdict(list)
        for edge in edges:
            node1, node2 = edge[0], edge[1]
            graph[node1].append(node2)
            graph[node2].append(node1)

        leaves = deque()
        edge_count = {}
        for node in graph.keys():
            if len(graph[node]) == 1:
                leaves.append(node)
                edge_count[node] = len(graph[node])

        count = 0
        while leaves:
            if count >= n - 2:
                return list(leaves)
            for i in range(len(leaves)):
                node = leaves.popleft()
                count += 1
                for neighbor in graph[node]:
                    edge_count[neighbor] -= 1
                    if edge_count[neighbor] == 1:
                        leaves.append(neighbor)
```

## [48 Rotate Image \(link\)](#)

### Description

You are given an  $n \times n$  2D matrix representing an image, rotate the image by **90** degrees (clockwise).

You have to rotate the image **in-place**, which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

#### Example 1:

1	2	3
4	5	6
7	8	9

7	4	1
8	5	2
9	6	3

**Input:** matrix = [[1,2,3],[4,5,6],[7,8,9]]  
**Output:** [[7,4,1],[8,5,2],[9,6,3]]

#### Example 2:

5	1	9	11
2	4	8	10
13	3	6	7
15	14	12	16

15	13	2	5
14	3	4	1
12	6	8	9
16	7	10	11

**Input:** matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]  
**Output:** [[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]

#### Constraints:

- $n == \text{matrix.length} == \text{matrix[i].length}$
- $1 \leq n \leq 20$
- $-1000 \leq \text{matrix[i][j]} \leq 1000$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def rotate(self, matrix: List[List[int]]) -> None:
        """
        Do not return anything, modify matrix in-place instead.
        """
        # rowStart, colStart, rowEnd, colEnd = 0, 0, len(matrix) - 1, len(matrix[0]) - 1
        # while rowStart < rowEnd and colStart < colEnd:
        #     # store first row
        #     firstRow = []
        #     for i in range(colEnd - colStart):
        #         firstRow.append(matrix[rowStart][colStart + i])
        #
        #     # first row becomes last column
        #
        #     # last column becomes last row
        #     # last row becomes first column
        #
        #     # first column becomes first row
        matrix[:] = zip(*matrix[::-1])
        # matrix[:] = zip(*matrix[::-1])
        return matrix
```

## [2121 Find if Path Exists in Graph \(link\)](#)

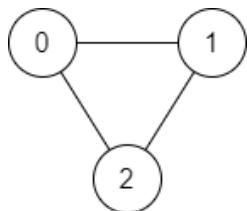
### Description

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  $\text{edges}$ , where each  $\text{edges}[i] = [u_i, v_i]$  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  $\text{source}$  to vertex  $\text{destination}$ .

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

#### Example 1:



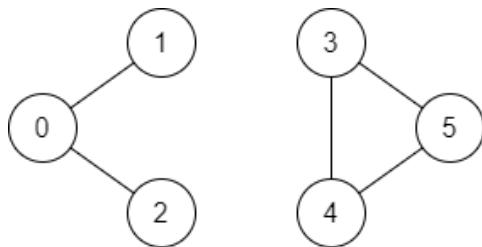
**Input:**  $n = 3$ ,  $\text{edges} = [[0,1],[1,2],[2,0]]$ ,  $\text{source} = 0$ ,  $\text{destination} = 2$

**Output:** `true`

**Explanation:** There are two paths from vertex  $0$  to vertex  $2$ :

- $0 \rightarrow 1 \rightarrow 2$
- $0 \rightarrow 2$

#### Example 2:



**Input:**  $n = 6$ ,  $\text{edges} = [[0,1],[0,2],[3,5],[5,4],[4,3]]$ ,  $\text{source} = 0$ ,  $\text{destination} = 5$

**Output:** `false`

**Explanation:** There is no path from vertex  $0$  to vertex  $5$ .

#### Constraints:

- $1 \leq n \leq 2 * 10^5$
- $0 \leq \text{edges.length} \leq 2 * 10^5$
- $\text{edges}[i].length == 2$
- $0 \leq u_i, v_i \leq n - 1$
- $u_i \neq v_i$
- $0 \leq \text{source}, \text{destination} \leq n - 1$
- There are no duplicate edges.
- There are no self edges.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

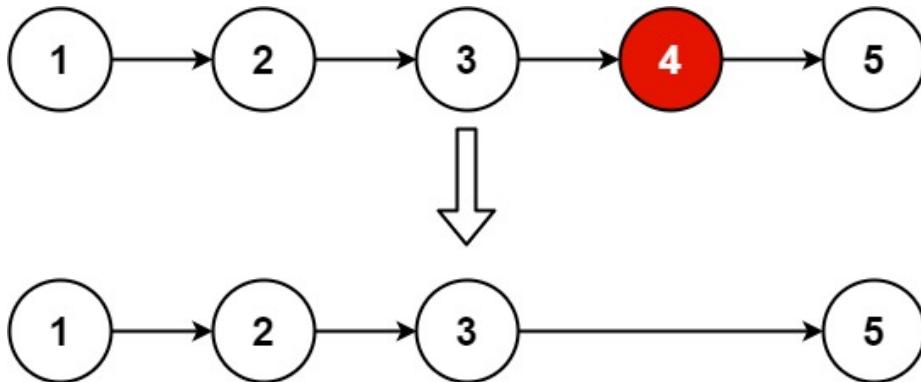
```
class Solution:
    from collections import deque
    def validPath(self, n: int, edges: List[List[int]], source: int, destination: int) -> bool:
        def generateGraph():
            graph = {}
            for edge in edges:
                if not graph.get(edge[0]):
                    graph[edge[0]] = []
                if not graph.get(edge[1]):
                    graph[edge[1]] = []
                graph[edge[0]].append(edge[1])
                graph[edge[1]].append(edge[0])
            return graph
        graph = generateGraph()
        visited = set()
        queue = deque()
        queue.append(source)
        while queue:
            curr = queue.popleft()
            visited.add(curr)
            if curr == destination:
                return True
            for node in graph[curr]:
                if node not in visited:
                    queue.append(node)
        return False
```

## 19 Remove Nth Node From End of List (link)

### Description

Given the head of a linked list, remove the  $n^{\text{th}}$  node from the end of the list and return its head.

#### Example 1:



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

#### Example 2:

**Input:** head = [1], n = 1  
**Output:** []

#### Example 3:

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

#### Constraints:

- The number of nodes in the list is  $sz$ .
- $1 \leq sz \leq 30$
- $0 \leq \text{Node.val} \leq 100$
- $1 \leq n \leq sz$

**Follow up:** Could you do this in one pass?

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def removeNthFromEnd(self, head: Optional[ListNode], n: int) -> Optional[ListNode]:
        dummy = ListNode(0)
        dummy.next = head
        slowRunner, fastRunner = dummy, dummy
        for i in range(n + 1):
            fastRunner = fastRunner.next

        while fastRunner is not None:
            slowRunner = slowRunner.next
            fastRunner = fastRunner.next
        slowRunner.next = slowRunner.next.next
        return dummy.next
```

## [1635 Number of Good Pairs \(link\)](#)

### Description

Given an array of integers `nums`, return *the number of good pairs*.

A pair  $(i, j)$  is called *good* if  $nums[i] == nums[j]$  and  $i < j$ .

#### Example 1:

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

**Explanation:** There are 4 good pairs  $(0,3)$ ,  $(0,4)$ ,  $(3,4)$ ,  $(2,5)$  0-indexed.

#### Example 2:

**Input:** `nums = [1,1,1,1]`  
**Output:** 6

**Explanation:** Each pair in the array are *good*.

#### Example 3:

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

### Constraints:

- $1 \leq \text{nums.length} \leq 100$
- $1 \leq \text{nums}[i] \leq 100$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def numIdenticalPairs(self, nums: List[int]) -> int:
        count = 0
        for i in range(len(nums)):
            for j in range(i+1, len(nums)):
                if nums[i] == nums[j]:
                    count += 1
        return count
```

## [1013 Fibonacci Number \(link\)](#)

### Description

The **Fibonacci numbers**, commonly denoted  $F(n)$  form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

```
 $F(0) = 0, F(1) = 1$   
 $F(n) = F(n - 1) + F(n - 2), \text{ for } n > 1.$ 
```

Given  $n$ , calculate  $F(n)$ .

#### Example 1:

```
Input: n = 2  
Output: 1  
Explanation:  $F(2) = F(1) + F(0) = 1 + 0 = 1.$ 
```

#### Example 2:

```
Input: n = 3  
Output: 2  
Explanation:  $F(3) = F(2) + F(1) = 1 + 1 = 2.$ 
```

#### Example 3:

```
Input: n = 4  
Output: 3  
Explanation:  $F(4) = F(3) + F(2) = 2 + 1 = 3.$ 
```

### Constraints:

- $0 \leq n \leq 30$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    seenFib = {}
    def fib(self, n: int) -> int:
        if n == 1:
            return 1
        if n == 0:
            return 0
        if self.seenFib.get(n):
            return self.seenFib[n]
        self.seenFib[n-1] = self.fib(n-1)
        self.seenFib[n-2] = self.fib(n-2)
        return self.seenFib[n-1] + self.seenFib[n-2]
```

## [199 Binary Tree Right Side View \(link\)](#)

### Description

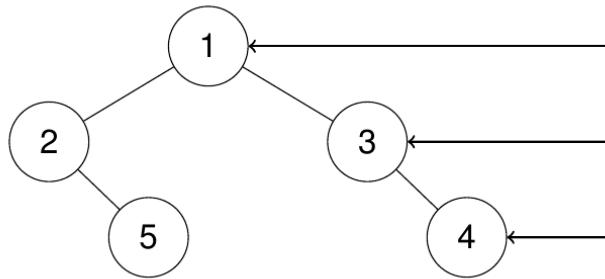
Given the root of a binary tree, imagine yourself standing on the **right side** of it, return *the values of the nodes you can see ordered from top to bottom*.

#### Example 1:

**Input:** root = [1,2,3,null,5,null,4]

**Output:** [1,3,4]

**Explanation:**

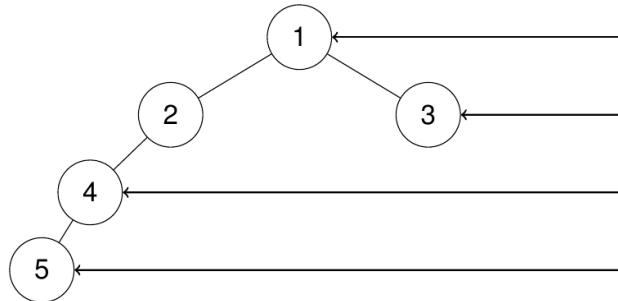


#### Example 2:

**Input:** root = [1,2,3,4,null,null,null,5]

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

**Explanation:**



#### Example 3:

**Input:** root = [1,null,3]

**Output:** [1,3]

#### Example 4:

**Input:** root = []

**Output:** []

#### Constraints:

- The number of nodes in the tree is in the range [0, 100].

- $-100 \leq \text{Node.val} \leq 100$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    from collections import deque
    def rightSideView(self, root: Optional[TreeNode]) -> List[int]:
        def levelordertraversal(root):
            if not root:
                return []
            levels = []
            queue = deque([root])
            while queue:
                curr_level = len(queue)
                curr_nodes = []
                for i in range(curr_level):
                    curr_node = queue.popleft()
                    curr_nodes.append(curr_node.val)

                    if curr_node.left:
                        queue.append(curr_node.left)

                    if curr_node.right:
                        queue.append(curr_node.right)
                levels.append(curr_nodes)
        return levels
    lot = levelordertraversal(root)

    ans = []
    for i in range(len(lot)):
        ans.append(lot[i][len(lot[i]) - 1])
    return ans
```

## [463 Island Perimeter \(link\)](#)

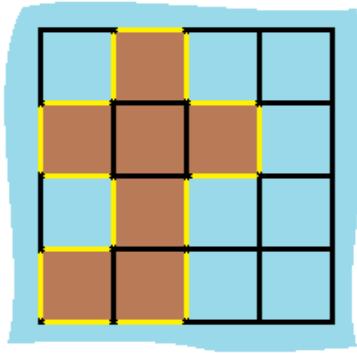
### Description

You are given `row x col` grid representing a map where `grid[i][j] = 1` represents land and `grid[i][j] = 0` represents water.

Grid cells are connected **horizontally/vertically** (not diagonally). The grid is completely surrounded by water, and there is exactly one island (i.e., one or more connected land cells).

The island doesn't have "lakes", meaning the water inside isn't connected to the water around the island. One cell is a square with side length 1. The grid is rectangular, width and height don't exceed 100. Determine the perimeter of the island.

#### Example 1:



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

**Output:** 16

**Explanation:** The perimeter is the 16 yellow stripes in the image above.

#### Example 2:

**Input:** grid = [[1]]

**Output:** 4

#### Example 3:

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

**Output:** 4

### Constraints:

- `row == grid.length`
- `col == grid[i].length`
- `1 <= row, col <= 100`
- `grid[i][j]` is 0 or 1.
- There is exactly one island in grid.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

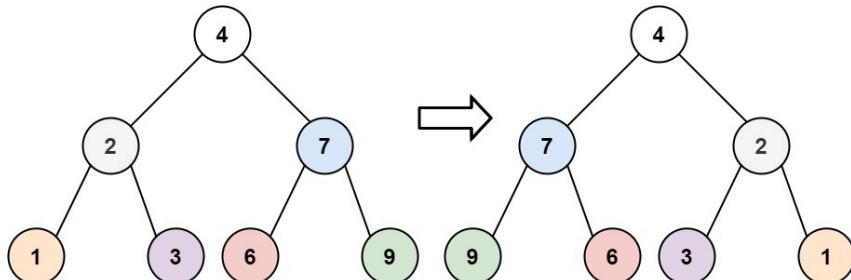
```
class Solution:
    def islandPerimeter(self, grid: List[List[int]]) -> int:
        perimiter = 0
        for i in range(len(grid)):
            for j in range(len(grid[0])):
                curr = grid[i][j]
                if curr == 1:
                    # if left of curr is edge or water
                    if i == 0:
                        perimiter = perimiter + 1
                    elif grid[i-1][j] == 0:
                        perimiter = perimiter + 1
                    # if right of curr is edge or water
                    if i == len(grid) - 1:
                        perimiter = perimiter + 1
                    elif grid[i+1][j] == 0:
                        perimiter = perimiter + 1
                    # if top of curr is edge or water
                    if j == 0:
                        perimiter = perimiter + 1
                    elif grid[i][j - 1] == 0:
                        perimiter = perimiter + 1
                    # if bottom of curr is edge or water
                    if j == len(grid[0]) - 1:
                        perimiter = perimiter + 1
                    elif grid[i][j + 1] == 0:
                        perimiter = perimiter + 1
        return perimiter
```

## [226 Invert Binary Tree \(link\)](#)

### Description

Given the root of a binary tree, invert the tree, and return *its root*.

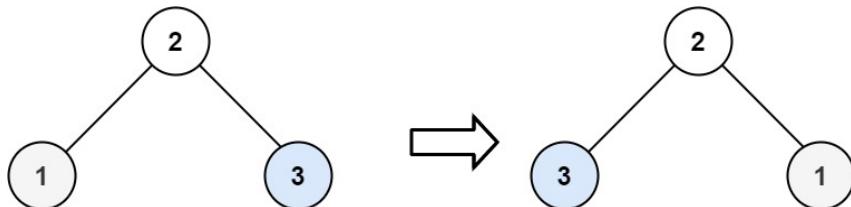
#### Example 1:



**Input:** root = [4,2,7,1,3,6,9]

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

#### Example 2:



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

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

#### Example 3:

**Input:** root = []

**Output:** []

### Constraints:

- The number of nodes in the tree is in the range [0, 100].
- $-100 \leq \text{Node.val} \leq 100$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def invertTree(self, root: Optional[TreeNode]) -> Optional[TreeNode]:
        if root == None:
            return root
        self.invertTree(root.left)
        self.invertTree(root.right)
        root.left, root.right = root.right, root.left
        return root
```

## 133 Clone Graph (link)

### Description

Given a reference of a node in a **connected** undirected graph.

Return a **deep copy** (clone) of the graph.

Each node in the graph contains a value (`int`) and a list (`List<Node>`) of its neighbors.

```
class Node {
    public int val;
    public List<Node> neighbors;
}
```

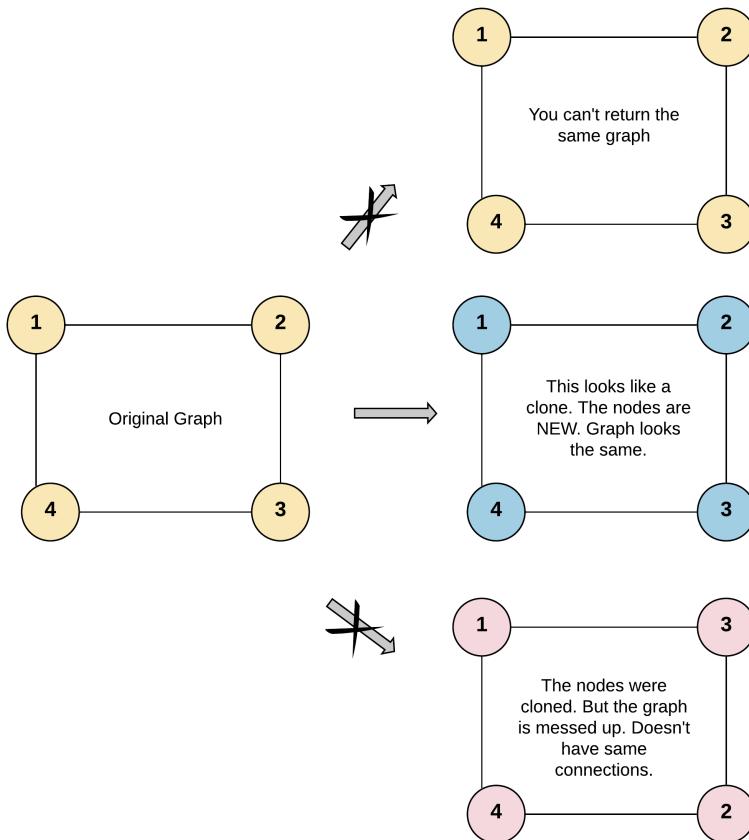
#### Test case format:

For simplicity, each node's value is the same as the node's index (1-indexed). For example, the first node with `val == 1`, the second node with `val == 2`, and so on. The graph is represented in the test case using an adjacency list.

**An adjacency list** is a collection of unordered **lists** used to represent a finite graph. Each list describes the set of neighbors of a node in the graph.

The given node will always be the first node with `val = 1`. You must return the **copy of the given node** as a reference to the cloned graph.

#### Example 1:



**Input:** `adjList = [[2,4],[1,3],[2,4],[1,3]]`

**Output:** `[[2,4],[1,3],[2,4],[1,3]]`

**Explanation:** There are 4 nodes in the graph.

1st node (`val = 1`)'s neighbors are 2nd node (`val = 2`) and 4th node (`val = 4`).  
2nd node (`val = 2`)'s neighbors are 1st node (`val = 1`) and 3rd node (`val = 3`).

3rd node (val = 3)'s neighbors are 2nd node (val = 2) and 4th node (val = 4).  
4th node (val = 4)'s neighbors are 1st node (val = 1) and 3rd node (val = 3).

### Example 2:



**Input:** adjList = []  
**Output:** []

**Explanation:** Note that the input contains one empty list. The graph consists of only one node.

### Example 3:

**Input:** adjList = []  
**Output:** []

**Explanation:** This is an empty graph, it does not have any nodes.

### Constraints:

- The number of nodes in the graph is in the range [0, 100].
- $1 \leq \text{Node.val} \leq 100$
- `Node.val` is unique for each node.
- There are no repeated edges and no self-loops in the graph.
- The Graph is connected and all nodes can be visited starting from the given node.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
.....
# Definition for a Node.
class Node:
    def __init__(self, val = 0, neighbors = None):
        self.val = val
        self.neighbors = neighbors if neighbors is not None else []
.....
from typing import Optional
class Solution:
    def cloneGraph(self, node: Optional['Node']) -> Optional['Node']:
        if not node:
            return None

        visited = {}

        def dfs(node):
            if node in visited:
                return visited[node]

            clone = Node(node.val)
            visited[node] = clone

            for peer in node.neighbors:
                clone.neighbors.append(dfs(peer))

            return clone
        return dfs(node)
```

## 200 Number of Islands (link)

### Description

Given an  $m \times n$  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:

- $m == \text{grid.length}$
- $n == \text{grid[i].length}$
- $1 \leq m, n \leq 300$
- $\text{grid[i][j]}$  is '0' or '1'.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
        count = 0
        def floodfill(curr_pos, grid):
            x, y = curr_pos[0], curr_pos[1]
            if x < 0 or x >= len(grid) or y < 0 or y >= len(grid[0]):
                return
            if grid[x][y] == "0":
                return

            grid[x][y] = "0"
            floodfill([x+1, y], grid)
            floodfill([x-1, y], grid)
            floodfill([x, y+1], grid)
            floodfill([x, y-1], grid)

        for i in range(len(grid)):
            for j in range(len(grid[0])):
                if grid[i][j] == "1":
                    count = count + 1
                    floodfill([i, j], grid)
        return count
```

## [129 Sum Root to Leaf Numbers \(link\)](#)

### Description

You are given the root of a binary tree containing digits from 0 to 9 only.

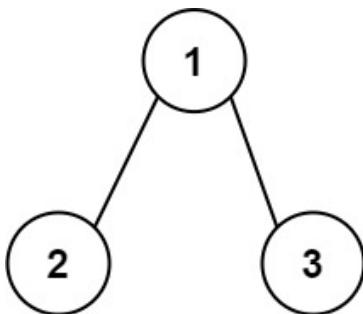
Each root-to-leaf path in the tree represents a number.

- For example, the root-to-leaf path 1 → 2 → 3 represents the number 123.

Return *the total sum of all root-to-leaf numbers*. Test cases are generated so that the answer will fit in a **32-bit** integer.

A **leaf** node is a node with no children.

#### Example 1:



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

**Output:** 25

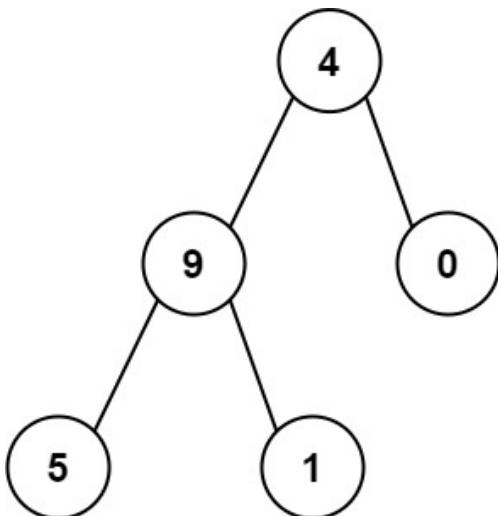
**Explanation:**

The root-to-leaf path 1→2 represents the number 12.

The root-to-leaf path 1→3 represents the number 13.

Therefore, sum = 12 + 13 = 25.

#### Example 2:



**Input:** root = [4,9,0,5,1]

**Output:** 1026

**Explanation:**

The root-to-leaf path 4→9→5 represents the number 495.

The root-to-leaf path 4→9→1 represents the number 491.

The root-to-leaf path 4→0 represents the number 40.

Therefore, sum = 495 + 491 + 40 = 1026.

**Constraints:**

- The number of nodes in the tree is in the range  $[1, 1000]$ .
- $0 \leq \text{Node.val} \leq 9$
- The depth of the tree will not exceed 10.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def sumNumbers(self, root: Optional[TreeNode]) -> int:
        numbers = []
        def get_all_numbers(root, path = 0):
            if not root:
                return
            path = path * 10 + root.val
            if not root.left and not root.right:
                numbers.append(path)
            get_all_numbers(root.left, path)
            get_all_numbers(root.right, path)

        get_all_numbers(root)
        sum = 0
        for num in numbers:
            sum = sum + num
        return sum
```

## 117 Populating Next Right Pointers in Each Node II (link)

### Description

Given a binary tree

```
struct Node {
    int val;
    Node *left;
    Node *right;
    Node *next;
}
```

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.

Initially, all next pointers are set to NULL.

#### Example 1:

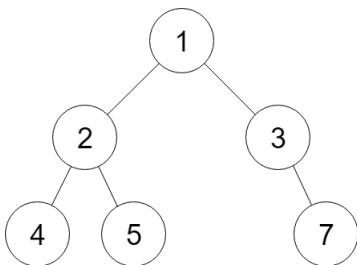


Figure A

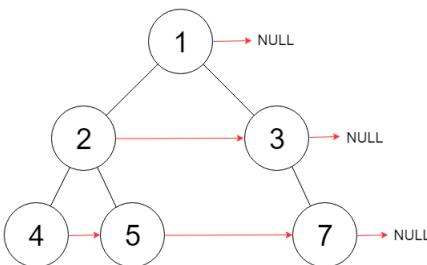


Figure B

**Input:** root = [1,2,3,4,5,null,7]  
**Output:** [1,#,2,3,#,4,5,7,#]

**Explanation:** Given the above binary tree (Figure A), your function should populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.

#### Example 2:

**Input:** root = []  
**Output:** []

#### Constraints:

- The number of nodes in the tree is in the range [0, 6000].
- $-100 \leq \text{Node.val} \leq 100$

#### Follow-up:

- You may only use constant extra space.
- The recursive approach is fine. You may assume implicit stack space does not count as extra space for this problem.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
"""
# Definition for a Node.
class Node:
    def __init__(self, val: int = 0, left: 'Node' = None, right: 'Node' = None, next: 'Node' = None):
        self.val = val
        self.left = left
        self.right = right
        self.next = next
"""

class Solution:
    from collections import deque
    def connect(self, root: 'Node') -> 'Node':
        if not root:
            return root

        def level_order_traversal(root):
            levels = []
            queue = deque([root])

            while queue:
                curr_size = len(queue)
                curr_nodes = []
                for i in range(curr_size):
                    front = queue.popleft()
                    curr_nodes.append(front)

                    if front.left:
                        queue.append(front.left)
                    if front.right:
                        queue.append(front.right)

                levels.append(curr_nodes)
            return levels

        level_order_traversal_nodes = level_order_traversal(root)
        for level in level_order_traversal_nodes:
            for node_idx in range(len(level) - 1):
                level[node_idx].next = level[node_idx + 1]
        return root
```

## 116 Populating Next Right Pointers in Each Node (link)

### Description

You are given a **perfect binary tree** where all leaves are on the same level, and every parent has two children. The binary tree has the following definition:

```
struct Node {
    int val;
    Node *left;
    Node *right;
    Node *next;
}
```

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.

Initially, all next pointers are set to NULL.

#### Example 1:

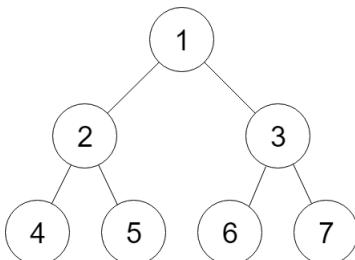


Figure A

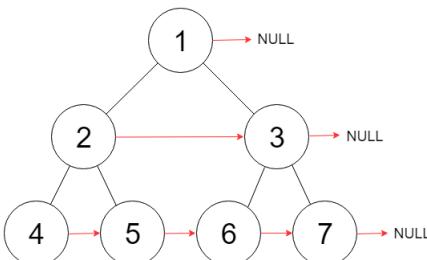


Figure B

**Input:** root = [1,2,3,4,5,6,7]  
**Output:** [1,#,2,3,#,4,5,6,7,#]

**Explanation:** Given the above perfect binary tree (Figure A), your function should populate each

#### Example 2:

**Input:** root = []  
**Output:** []

#### Constraints:

- The number of nodes in the tree is in the range  $[0, 2^{12} - 1]$ .
- $-1000 \leq \text{Node.val} \leq 1000$

#### Follow-up:

- You may only use constant extra space.
- The recursive approach is fine. You may assume implicit stack space does not count as extra space for this problem.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
"""
# Definition for a Node.
class Node:
    def __init__(self, val: int = 0, left: 'Node' = None, right: 'Node' = None, next: 'Node' = None):
        self.val = val
        self.left = left
        self.right = right
        self.next = next
"""

class Solution:
    from collections import deque
    def connect(self, root: 'Optional[Node]') -> 'Optional[Node]':
        if not root:
            return root

        def level_order_traversal(root):
            queue = deque([root])
            levels = []

            while queue:
                curr_level_size = len(queue)
                curr_level_nodes = []

                for i in range(curr_level_size):
                    front = queue.popleft()
                    curr_level_nodes.append(front)

                    if front.left:
                        queue.append(front.left)

                    if front.right:
                        queue.append(front.right)
                levels.append(curr_level_nodes)
            return levels

        level_order_traversal_nodes = level_order_traversal(root)

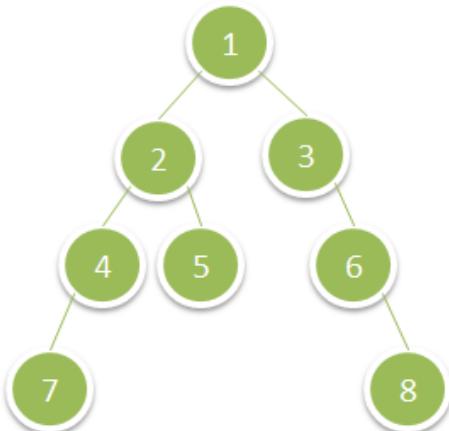
        for level in level_order_traversal_nodes:
            for node_idx in range(len(level)):
                # if right most node
                if node_idx == len(level) - 1:
                    level[node_idx].next = None
                else:
                    level[node_idx].next = level[node_idx + 1]
        return root
```

## [1254 Deepest Leaves Sum \(link\)](#)

### Description

Given the root of a binary tree, return *the sum of values of its deepest leaves*.

#### Example 1:



**Input:** root = [1,2,3,4,5,null,6,7,null,null,null,8]  
**Output:** 15

#### Example 2:

**Input:** root = [6,7,8,2,7,1,3,9,null,1,4,null,null,null,5]  
**Output:** 19

#### Constraints:

- The number of nodes in the tree is in the range  $[1, 10^4]$ .
- $1 \leq \text{Node.val} \leq 100$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

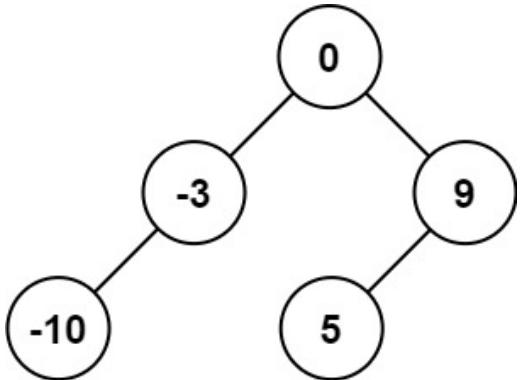
```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    from collections import deque
    def deepestLeavesSum(self, root: Optional[TreeNode]) -> int:
        if root == None:
            return 0
        def level_order_traversal(root: TreeNode):
            if not root:
                return []
            levels = []
            queue = deque([root])
            while queue:
                curr_level_size = len(queue)
                curr_level_nodes = []
                for i in range(curr_level_size):
                    front = queue.popleft()
                    curr_level_nodes.append(front)
                    if front.left:
                        queue.append(front.left)
                    if front.right:
                        queue.append(front.right)
                levels.append(curr_level_nodes)
            return levels
        arrsum = 0
        levels = level_order_traversal(root)
        for node in levels[-1]:
            arrsum += node.val
        return arrsum
```

## [108 Convert Sorted Array to Binary Search Tree \(link\)](#)

### Description

Given an integer array `nums` where the elements are sorted in **ascending order**, convert *it to a **height-balanced** binary search tree*.

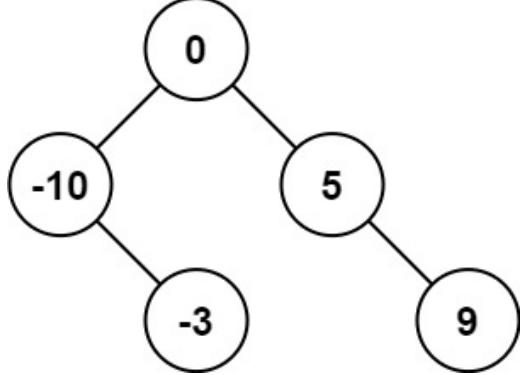
#### Example 1:



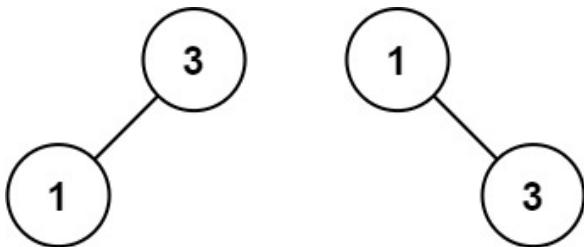
**Input:** `nums = [-10, -3, 0, 5, 9]`

**Output:** `[0, -3, 9, -10, null, 5]`

**Explanation:** `[0, -10, 5, null, -3, null, 9]` is also accepted:



#### Example 2:



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

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

**Explanation:** `[1, null, 3]` and `[3, 1]` are both height-balanced BSTs.

### Constraints:

- $1 \leq \text{nums.length} \leq 10^4$
- $-10^4 \leq \text{nums}[i] \leq 10^4$
- `nums` is sorted in a **strictly increasing order**.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    import math
    def sortedArrayToBST(self, nums: List[int]) -> Optional[TreeNode]:
        if len(nums) == 0:
            return None

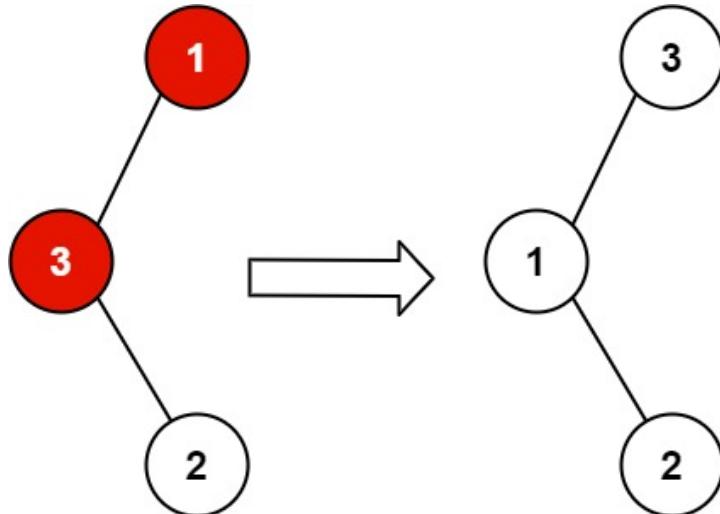
        return TreeNode(
            nums[len(nums) // 2],
            self.sortedArrayToBST(nums[:len(nums)//2]),
            self.sortedArrayToBST(nums[len(nums)//2 + 1:]))
    )
```

## [99 Recover Binary Search Tree \(link\)](#)

### Description

You are given the root of a binary search tree (BST), where the values of **exactly** two nodes of the tree were swapped by mistake. *Recover the tree without changing its structure.*

#### Example 1:

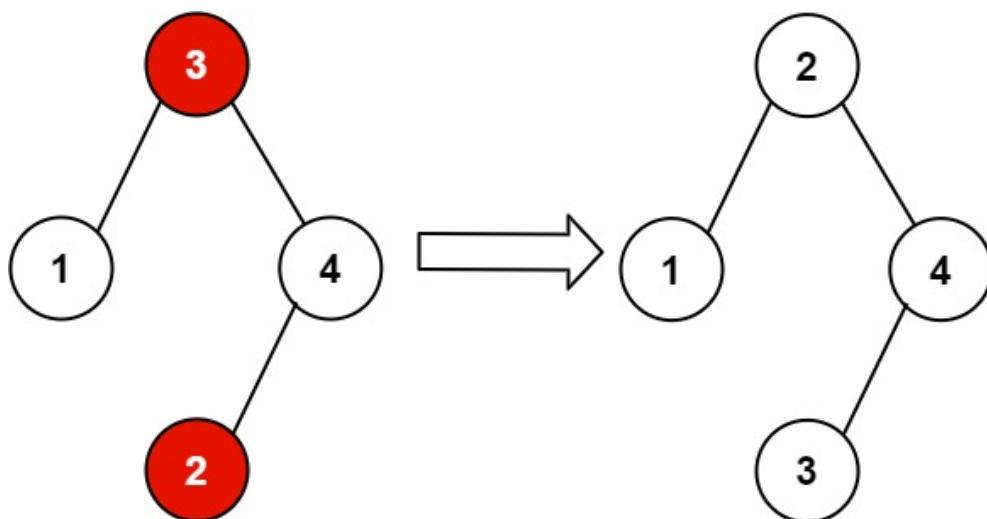


**Input:** root = [1,3,null,null,2]

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

**Explanation:** 3 cannot be a left child of 1 because  $3 > 1$ . Swapping 1 and 3 makes the BST valid.

#### Example 2:



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

**Output:** [2,1,4,null,null,3]

**Explanation:** 2 cannot be in the right subtree of 3 because  $2 < 3$ . Swapping 2 and 3 makes the BST valid.

### Constraints:

- The number of nodes in the tree is in the range  $[2, 1000]$ .
- $-2^{31} \leq \text{Node.val} \leq 2^{31} - 1$

**Follow up:** A solution using  $O(n)$  space is pretty straight-forward. Could you devise a constant  $O(1)$  space solution?

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def recoverTree(self, root: Optional[TreeNode]) -> None:
        """
        Do not return anything, modify root in-place instead.
        """
        inorderList = []
        currInorderList = []
        def inorder(tree_root):
            if tree_root == None:
                return
            inorder(tree_root.left)
            inorderList.append(tree_root.val)
            inorder(tree_root.right)
        inorder(root)
        inorderList.sort()

        def traverse_tree(tree_root):
            if tree_root == None:
                return
            traverse_tree(tree_root.left)
            currInorderList.append(tree_root.val)
            if currInorderList[-1] != inorderList[len(currInorderList) - 1]:
                tree_root.val = inorderList[len(currInorderList)-1]
            traverse_tree(tree_root.right)

        traverse_tree(root)
```

## [98 Validate Binary Search Tree \(link\)](#)

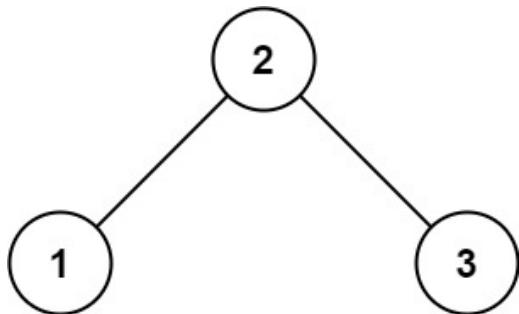
### Description

Given the root of a binary tree, determine if it is a valid binary search tree (BST).

A **valid BST** is defined as follows:

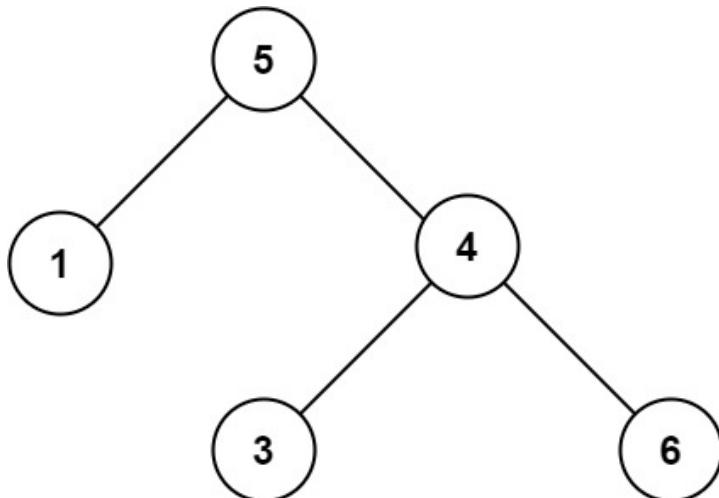
- The left subtree of a node contains only nodes with keys **strictly less than** the node's key.
- The right subtree of a node contains only nodes with keys **strictly greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

#### Example 1:



**Input:** root = [2,1,3]  
**Output:** true

#### Example 2:



**Input:** root = [5,1,4,null,null,3,6]  
**Output:** false  
**Explanation:** The root node's value is 5 but its right child's value is 4.

### Constraints:

- The number of nodes in the tree is in the range  $[1, 10^4]$ .
- $-2^{31} \leq \text{Node.val} \leq 2^{31} - 1$

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def isValidBST(self, root: Optional[TreeNode]) -> bool:
        inorder = []
        def checkValidity(tree_root: Optional[TreeNode]):
            if tree_root == None:
                return
            checkValidity(tree_root.left)
            inorder.append(tree_root.val)
            checkValidity(tree_root.right)
        checkValidity(root)
        for i in range(len(inorder) - 1):
            if inorder[i] >= inorder[i + 1]:
                return False
        return True
```

## [3068 Rename Columns \(link\)](#)

### Description

```
DataFrame students
+-----+-----+
| Column Name | Type |
+-----+-----+
| id          | int   |
| first       | object |
| last        | object |
| age          | int   |
+-----+-----+
```

Write a solution to rename the columns as follows:

- `id` to `student_id`
- `first` to `first_name`
- `last` to `last_name`
- `age` to `age_in_years`

The result format is in the following example.

#### Example 1:

##### Input:

```
+-----+-----+-----+
| id  | first | last  | age |
+-----+-----+-----+
| 1   | Mason | King   | 6   |
| 2   | Ava   | Wright | 7   |
| 3   | Taylor | Hall   | 16  |
| 4   | Georgia | Thompson | 18 |
| 5   | Thomas | Moore  | 10 |
+-----+-----+-----+
```

##### Output:

```
+-----+-----+-----+
| student_id | first_name | last_name | age_in_years |
+-----+-----+-----+
| 1          | Mason      | King      | 6           |
| 2          | Ava        | Wright    | 7           |
| 3          | Taylor     | Hall      | 16          |
| 4          | Georgia    | Thompson  | 18          |
| 5          | Thomas     | Moore     | 10          |
+-----+-----+-----+
```

##### Explanation:

The column names are changed accordingly.

(scroll down for solution)

## Solution

Language: python

Status: Accepted

```
import pandas as pd

def renameColumns(students: pd.DataFrame) -> pd.DataFrame:
    students.rename(columns={'id': 'student_id', 'first': 'first_name', 'last': 'last_name', 'age': 'age'}, inplace=True)
    return students
```

## [3067 Modify Columns \(link\)](#)

### Description

```
DataFrame employees
+-----+-----+
| Column Name | Type   |
+-----+-----+
| name        | object |
| salary       | int    |
+-----+-----+
```

A company intends to give its employees a pay rise.

Write a solution to **modify** the `salary` column by multiplying each salary by 2.

The result format is in the following example.

#### Example 1:

```
Input:
DataFrame employees
+-----+-----+
| name    | salary |
+-----+-----+
| Jack    | 19666  |
| Piper   | 74754  |
| Mia     | 62509  |
| Ulysses | 54866  |
+-----+-----+
Output:
+-----+-----+
| name    | salary |
+-----+-----+
| Jack    | 39332  |
| Piper   | 149508 |
| Mia     | 125018 |
| Ulysses | 109732 |
+-----+-----+
Explanation:
Every salary has been doubled.
```

(scroll down for solution)

## Solution

Language: python

Status: Accepted

```
import pandas as pd

def modifySalaryColumn(employees: pd.DataFrame) -> pd.DataFrame:
    employees['salary'] *= 2
    return employees
```

## [3075 Drop Missing Data \(link\)](#)

### Description

```
DataFrame students
+-----+-----+
| Column Name | Type   |
+-----+-----+
| student_id  | int    |
| name        | object |
| age         | int    |
+-----+-----+
```

There are some rows having missing values in the `name` column.

Write a solution to remove the rows with missing values.

The result format is in the following example.

#### Example 1:

**Input:**

```
+-----+-----+-----+
| student_id | name   | age   |
+-----+-----+-----+
| 32        | Piper   | 5     |
| 217       | None    | 19    |
| 779       | Georgia | 20    |
| 849       | Willow  | 14    |
+-----+-----+-----+
```

**Output:**

```
+-----+-----+-----+
| student_id | name   | age   |
+-----+-----+-----+
| 32        | Piper   | 5     |
| 779       | Georgia | 20    |
| 849       | Willow  | 14    |
+-----+-----+-----+
```

**Explanation:**

Student with id 217 has empty value in the `name` column, so it will be removed.

(scroll down for solution)

## Solution

Language: python

Status: Accepted

```
import pandas as pd

def dropMissingData(students: pd.DataFrame) -> pd.DataFrame:
    return students.dropna(subset = 'name')
```

## [3071 Drop Duplicate Rows \(link\)](#)

### Description

```
DataFrame customers
+-----+-----+
| Column Name | Type |
+-----+-----+
| customer_id | int  |
| name         | object |
| email        | object |
+-----+-----+
```

There are some duplicate rows in the DataFrame based on the `email` column.

Write a solution to remove these duplicate rows and keep only the **first** occurrence.

The result format is in the following example.

#### Example 1:

##### Input:

```
+-----+-----+
| customer_id | name     | email      |
+-----+-----+
| 1           | Ella     | emily@example.com |
| 2           | David    | michael@example.com |
| 3           | Zachary  | sarah@example.com |
| 4           | Alice    | john@example.com |
| 5           | Finn     | john@example.com |
| 6           | Violet   | alice@example.com |
+-----+-----+
```

##### Output:

```
+-----+-----+
| customer_id | name     | email      |
+-----+-----+
| 1           | Ella     | emily@example.com |
| 2           | David    | michael@example.com |
| 3           | Zachary  | sarah@example.com |
| 4           | Alice    | john@example.com |
| 6           | Violet   | alice@example.com |
+-----+-----+
```

##### Explanation:

Alic (`customer_id = 4`) and Finn (`customer_id = 5`) both use `john@example.com`, so only the first

(scroll down for solution)

## Solution

Language: python

Status: Accepted

```
import pandas as pd

def dropDuplicateEmails(customers: pd.DataFrame) -> pd.DataFrame:
    return customers.drop_duplicates(subset = 'email')
```

## [3066 Create a New Column \(link\)](#)

### Description

```
DataFrame employees
+-----+-----+
| Column Name | Type. |
+-----+-----+
| name        | object |
| salary       | int.  |
+-----+-----+
```

A company plans to provide its employees with a bonus.

Write a solution to create a new column name **bonus** that contains the **doubled values** of the **salary** column.

The result format is in the following example.

#### Example 1:

**Input:**

```
DataFrame employees
+-----+-----+
| name | salary |
+-----+-----+
| Piper | 4548  |
| Grace | 28150 |
| Georgia | 1103 |
| Willow | 6593  |
| Finn   | 74576 |
| Thomas | 24433 |
+-----+-----+
```

**Output:**

```
+-----+-----+-----+
| name | salary | bonus  |
+-----+-----+-----+
| Piper | 4548  | 9096  |
| Grace | 28150 | 56300 |
| Georgia | 1103 | 2206  |
| Willow | 6593  | 13186 |
| Finn   | 74576 | 149152 |
| Thomas | 24433 | 48866 |
+-----+-----+-----+
```

**Explanation:**

A new column **bonus** is created by doubling the value in the column **salary**.

(scroll down for solution)

## Solution

Language: python

Status: Accepted

```
import pandas as pd

def createBonusColumn(employees: pd.DataFrame) -> pd.DataFrame:
    employees['bonus'] = employees['salary'] * 2
    return employees
```

## [3074 Select Data \(link\)](#)

### Description

```
DataFrame students
+-----+-----+
| Column Name | Type   |
+-----+-----+
| student_id  | int    |
| name         | object |
| age          | int    |
+-----+-----+
```

Write a solution to select the name and age of the student with `student_id = 101`.

The result format is in the following example.

#### Example 1:

##### Input:

```
+-----+-----+-----+
| student_id | name     | age    |
+-----+-----+-----+
| 101        | Ulysses  | 13     |
| 53         | William   | 10     |
| 128        | Henry    | 6      |
| 3          | Henry    | 11     |
+-----+-----+-----+
```

##### Output:

```
+-----+-----+
| name   | age   |
+-----+-----+
| Ulysses | 13   |
+-----+-----+
```

##### Explanation:

Student Ulysses has `student_id = 101`, we select the name and age.

(scroll down for solution)

## Solution

Language: python

Status: Accepted

```
import pandas as pd

def selectData(students: pd.DataFrame) -> pd.DataFrame:
    return students.loc[students['student_id'] == 101, ['name', 'age']]
```

## [3065 Display the First Three Rows \(link\)](#)

### Description

```
DataFrame: employees
+-----+-----+
| Column Name | Type   |
+-----+-----+
| employee_id | int    |
| name         | object |
| department   | object |
| salary       | int    |
+-----+-----+
```

Write a solution to display the **first 3** rows of this DataFrame.

#### Example 1:

**Input:**

```
DataFrame employees
```

```
+-----+-----+-----+-----+
| employee_id | name      | department          | salary   |
+-----+-----+-----+-----+
| 3           | Bob       | Operations          | 48675   |
| 90          | Alice     | Sales               | 11096   |
| 9           | Tatiana   | Engineering         | 33805   |
| 60          | Annabelle | InformationTechnology | 37678   |
| 49          | Jonathan  | HumanResources     | 23793   |
| 43          | Khaled    | Administration       | 40454   |
+-----+-----+-----+-----+
```

**Output:**

```
+-----+-----+-----+-----+
| employee_id | name      | department          | salary   |
+-----+-----+-----+-----+
| 3           | Bob       | Operations          | 48675   |
| 90          | Alice     | Sales               | 11096   |
| 9           | Tatiana   | Engineering         | 33805   |
+-----+-----+-----+-----+
```

**Explanation:**

Only the first 3 rows are displayed.

(scroll down for solution)

## Solution

Language: python

Status: Accepted

```
import pandas as pd

def selectFirstRows(employees: pd.DataFrame) -> pd.DataFrame:
    return employees.head(3)
```

## 3076 Get the Size of a DataFrame ([link](#))

### Description

```
DataFrame players:
+-----+-----+
| Column Name | Type |
+-----+-----+
| player_id | int  |
| name       | object |
| age        | int  |
| position   | object |
| ...        | ...  |
+-----+-----+
```

Write a solution to calculate and display the **number of rows and columns** of players.

Return the result as an array:

[number of rows, number of columns]

The result format is in the following example.

### Example 1:

**Input:**

player_id	name	age	position	team
846	Mason	21	Forward	RealMadrid
749	Riley	30	Winger	Barcelona
155	Bob	28	Striker	ManchesterUnited
583	Isabella	32	Goalkeeper	Liverpool
388	Zachary	24	Midfielder	BayernMunich
883	Ava	23	Defender	Chelsea
355	Violet	18	Striker	Juventus
247	Thomas	27	Striker	ParisSaint-Germain
761	Jack	33	Midfielder	ManchesterCity
642	Charlie	36	Center-back	Arsenal

**Output:**

[10, 5]

**Explanation:**

This DataFrame contains 10 rows and 5 columns.

(scroll down for solution)

## Solution

Language: python

Status: Accepted

```
import pandas as pd

def getDataframeSize(players: pd.DataFrame) -> List[int]:
    return list(players.shape)
```

## [3062 Create a DataFrame from List \(link\)](#)

### Description

Write a solution to **create** a DataFrame from a 2D list called `student_data`. This 2D list contains the IDs and ages of some students.

The DataFrame should have two columns, `student_id` and `age`, and be in the same order as the original 2D list.

The result format is in the following example.

#### Example 1:

```
Input:  
student_data:  
[  
    [1, 15],  
    [2, 11],  
    [3, 11],  
    [4, 20]  
]
```

Output:

student_id	age
1	15
2	11
3	11
4	20

Explanation:

A DataFrame was created on top of `student_data`, with two columns named `student_id` and `age`.

(scroll down for solution)

## Solution

Language: python

Status: Accepted

```
import pandas as pd

def createDataframe(student_data: List[List[int]]) -> pd.DataFrame:
    labels=["student_id","age"]
    return pd.DataFrame(student_data, columns = labels)
```

## [94 Binary Tree Inorder Traversal \(link\)](#)

### Description

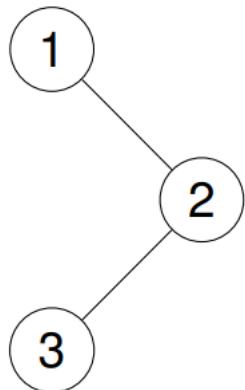
Given the root of a binary tree, return *the inorder traversal of its nodes' values*.

#### Example 1:

**Input:** root = [1,null,2,3]

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

**Explanation:**

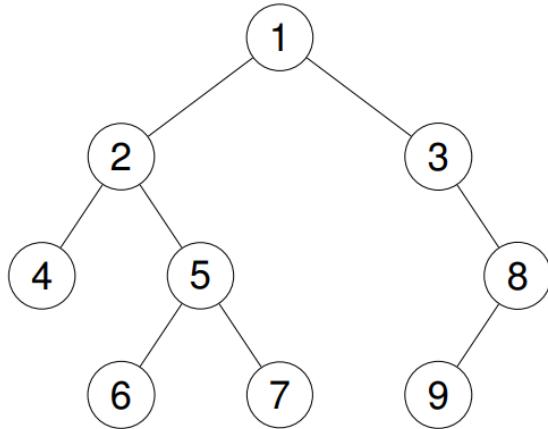


#### Example 2:

**Input:** root = [1,2,3,4,5,null,8,null,null,6,7,9]

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

**Explanation:**



#### Example 3:

**Input:** root = []

**Output:** []

#### Example 4:

**Input:** root = [1]

**Output:** [1]

**Constraints:**

- The number of nodes in the tree is in the range  $[0, 100]$ .
- $-100 \leq \text{Node.val} \leq 100$

**Follow up:** Recursive solution is trivial, could you do it iteratively?

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def inorderTraversal(self, root: Optional[TreeNode]) -> List[int]:
        res = []

        def inorder(root):
            if not root:
                return
            inorder(root.left)
            res.append(root.val)
            inorder(root.right)

        inorder(root)
        return res
```

## 26 Remove Duplicates from Sorted Array (link)

### Description

Given an integer array `nums` sorted in **non-decreasing order**, remove the duplicates **in-place** such that each unique element appears only **once**. The **relative order** of the elements should be kept the **same**.

Consider the number of *unique elements* in `nums` to be `k`. After removing duplicates, return the number of unique elements `k`.

The first `k` elements of `nums` should contain the unique numbers in **sorted order**. The remaining elements beyond index `k - 1` can be ignored.

#### Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int[] expectedNums = [...]; // The expected answer with correct length

int k = removeDuplicates(nums); // Calls your implementation

assert k == expectedNums.length;
for (int i = 0; i < k; i++) {
    assert nums[i] == expectedNums[i];
}
```

If all assertions pass, then your solution will be **accepted**.

#### Example 1:

**Input:** `nums = [1,1,2]`  
**Output:** `2, nums = [1,2,]`  
**Explanation:** Your function should return `k = 2`, with the first two elements of `nums` being `1` and It does not matter what you leave beyond the returned `k` (hence they are underscores).

#### Example 2:

**Input:** `nums = [0,0,1,1,1,2,2,3,3,4]`  
**Output:** `5, nums = [0,1,2,3,4,_,_,_,_,_]`  
**Explanation:** Your function should return `k = 5`, with the first five elements of `nums` being `0, 1, 2, 3, 4`. It does not matter what you leave beyond the returned `k` (hence they are underscores).

#### Constraints:

- $1 \leq \text{nums.length} \leq 3 * 10^4$
- $-100 \leq \text{nums}[i] \leq 100$
- `nums` is sorted in **non-decreasing** order.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        next_index_to_place = 1
        for current_index in range(1, len(nums)):
            if nums[current_index] != nums[current_index - 1]:
                nums[next_index_to_place] = nums[current_index]
                next_index_to_place += 1
        return next_index_to_place
```

## 20 Valid Parentheses ([link](#))

### Description

Given a string  $s$  containing just the characters  $'('$ ,  $)'$ ,  $'{'$ ,  $'}'$ ,  $'[$  and  $']'$ , determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

#### Example 1:

**Input:**  $s = "()"$

**Output:** true

#### Example 2:

**Input:**  $s = "()[]{}"$

**Output:** true

#### Example 3:

**Input:**  $s = "()"$

**Output:** false

#### Example 4:

**Input:**  $s = "()"$

**Output:** true

#### Example 5:

**Input:**  $s = "()"$

**Output:** false

#### Constraints:

- $1 \leq s.length \leq 10^4$
- $s$  consists of parentheses only  $'()' [] {}'$ .

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def isValid(self, s: str) -> bool:
        stack = []

        for char in s:
            if char in ['(', '{', '[']:
                stack.append(char)
            elif char in [')', '}', ']']:
                if len(stack) == 0:
                    return False
                last_element = stack.pop()
                if (char == ')' and last_element != '(') or \
                   (char == '}' and last_element != '{') or \
                   (char == ']' and last_element != '[') :
                    return False
        return len(stack) == 0
```

## 14 Longest Common Prefix (link)

### Description

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

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

#### Example 1:

```
Input: strs = ["flower", "flow", "flight"]
Output: "fl"
```

#### Example 2:

```
Input: strs = ["dog", "racecar", "car"]
Output: ""
Explanation: There is no common prefix among the input strings.
```

#### Constraints:

- $1 \leq \text{strs.length} \leq 200$
- $0 \leq \text{strs[i].length} \leq 200$
- $\text{strs[i]}$  consists of only lowercase English letters if it is non-empty.

(scroll down for solution)

## Solution

Language: python3

Status: Accepted

```
class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str:
        if len(strs) == 0:
            return ""

        # get shortest string
        smallest_string_length = float(inf)
        for word in strs:
            if len(word) < smallest_string_length:
                smallest_string_length = len(word)

        # Check for common chars in strings
        current_smallest_substring = ""
        for i in range(smallest_string_length):
            for j in range(len(strs) - 1):
                if strs[j][i] != strs[j+1][i]:
                    return current_smallest_substring
            current_smallest_substring += strs[0][i]
        return current_smallest_substring
```

## 395 Longest Substring with At Least K Repeating Characters (link)

### Description

Given a string  $s$  and an integer  $k$ , return *the length of the longest substring of  $s$  such that the frequency of each character in this substring is greater than or equal to  $k$* .

if no such substring exists, return 0.

#### Example 1:

**Input:**  $s = "aaabb"$ ,  $k = 3$

**Output:** 3

**Explanation:** The longest substring is "aaa", as 'a' is repeated 3 times.

#### Example 2:

**Input:**  $s = "ababbc"$ ,  $k = 2$

**Output:** 5

**Explanation:** The longest substring is "ababb", as 'a' is repeated 2 times and 'b' is repeated 3 times.

#### Constraints:

- $1 \leq s.length \leq 10^4$
- $s$  consists of only lowercase English letters.
- $1 \leq k \leq 10^5$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    //bbaaacbc | k = 3
    /*
     [bbaaacbc]
    */

    int longestSubstring(string s, int k) {
        int n = s.size();
        return longestSubstringUtil(s, 0, n, k);
    }
    int longestSubstringUtil(string &s, int start, int end, int k) {
        if (end < k) return 0;
        int countMap[26] = {0};
        // update the countMap with the count of each character
        for (int i = start; i < end; i++)
            countMap[s[i] - 'a']++;
        for (int mid = start; mid < end; mid++) {
            if (countMap[s[mid] - 'a'] >= k) continue;
            int midNext = mid + 1;
            while (midNext < end && countMap[s[midNext] - 'a'] < k) midNext++;
            return max(longestSubstringUtil(s, start, mid, k),
                      longestSubstringUtil(s, midNext, end, k));
        }
        return (end - start);
    }
};
```

## [219 Contains Duplicate II \(link\)](#)

### Description

Given an integer array `nums` and an integer `k`, return `true` if there are two *distinct indices* `i` and `j` in the array such that `nums[i] == nums[j]` and `abs(i - j) <= k`.

#### Example 1:

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

#### Example 2:

```
Input: nums = [1,0,1,1], k = 1
Output: true
```

#### Example 3:

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

### Constraints:

- $1 \leq \text{nums.length} \leq 10^5$
- $-10^9 \leq \text{nums[i]} \leq 10^9$
- $0 \leq k \leq 10^5$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    bool containsNearbyDuplicate(vector<int>& nums, int k) {
        unordered_map<int,int> m;
        for(int i=0; i<nums.size(); i++){
            if(m.find(nums[i]) == m.end()) m[nums[i]] = i;
            else {
                if (i - m[nums[i]] <= k) return true;
                else m[nums[i]] = i;
            }
        }
        return false;
    }
};
```

## 328 Odd Even Linked List (link)

### Description

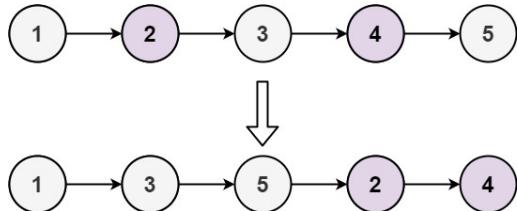
Given the head of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return *the reordered list*.

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

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

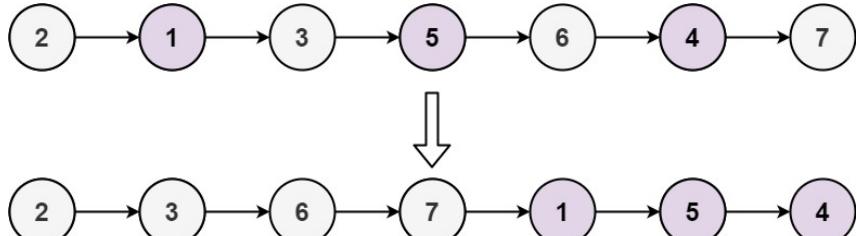
You must solve the problem in  $O(1)$  extra space complexity and  $O(n)$  time complexity.

#### Example 1:



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

#### Example 2:



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

#### Constraints:

- The number of nodes in the linked list is in the range  $[0, 10^4]$ .
- $-10^6 \leq \text{Node.val} \leq 10^6$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
/*
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* oddEvenList(ListNode* head) {
        if (head == nullptr) return nullptr;

        ListNode *headcopy1 = head;
        ListNode *headcopy2 = head->next;
        ListNode *cp2Head = headcopy2;

        while (headcopy2 != nullptr) {
            if (headcopy2->next == nullptr) {
                break;
            }

            headcopy1->next = headcopy2->next;
            headcopy1 = headcopy1->next;

            headcopy2->next = headcopy1->next;
            headcopy2 = headcopy2->next;
        }

        headcopy1->next = cp2Head;
        return head;
    }
};
```

## [152 Maximum Product Subarray \(link\)](#)

### Description

Given an integer array `nums`, find a subarray that has the largest product, and return *the product*.

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

**Note** that the product of an array with a single element is the value of that element.

#### Example 1:

```
Input: nums = [2,3,-2,4]
Output: 6
Explanation: [2,3] has the largest product 6.
```

#### Example 2:

```
Input: nums = [-2,0,-1]
Output: 0
Explanation: The result cannot be 2, because [-2,-1] is not a subarray.
```

#### Constraints:

- $1 \leq \text{nums.length} \leq 2 * 10^4$
- $-10 \leq \text{nums}[i] \leq 10$
- The product of any subarray of `nums` is **guaranteed** to fit in a **32-bit** integer.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    int maxProduct(vector<int>& nums) {
        if (nums.size() == 0)
            return 0;

        int prev = 0, prodSoFar = 1, maxProd = nums[0];

        for (int i = nums.size() - 1; i >= 0; i--) {
            if (nums[i] == 0) {
                prodSoFar = 1;
                maxProd = max(maxProd, nums[i]);
                continue;
            }

            prodSoFar *= nums[i];
            maxProd = max(maxProd, nums[i]);
            maxProd = max(maxProd, prodSoFar);
        }

        prodSoFar = 1;

        for (int i = 0; i < nums.size(); i++) {
            if (nums[i] == 0) {
                prodSoFar = 1;
                maxProd = max(maxProd, nums[i]);
                continue;
            }

            prodSoFar *= nums[i];
            maxProd = max(maxProd, nums[i]);
            maxProd = max(maxProd, prodSoFar);
        }

        return maxProd;
    };
};
```

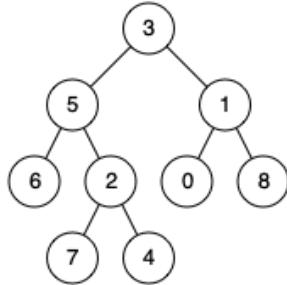
## [236 Lowest Common Ancestor of a Binary Tree \(link\)](#)

### Description

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the [definition of LCA on Wikipedia](#): “The lowest common ancestor is defined between two nodes  $p$  and  $q$  as the lowest node in  $T$  that has both  $p$  and  $q$  as descendants (where we allow a node to be a descendant of itself).”

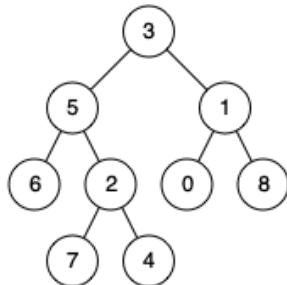
#### Example 1:



**Input:** root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1  
**Output:** 3

**Explanation:** The LCA of nodes 5 and 1 is 3.

#### Example 2:



**Input:** root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4  
**Output:** 5

**Explanation:** The LCA of nodes 5 and 4 is 5, since a node can be a descendant of itself according to the problem statement.

#### Example 3:

**Input:** root = [1,2], p = 1, q = 2  
**Output:** 1

#### Constraints:

- The number of nodes in the tree is in the range  $[2, 10^5]$ .
- $-10^9 \leq \text{Node.val} \leq 10^9$
- All `Node.val` are **unique**.
- $p \neq q$
- $p$  and  $q$  will exist in the tree.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int dep(TreeNode *root, TreeNode *child)
    {
        unordered_map<TreeNode*, int> level;
        queue<TreeNode*> q;

        q.push(root);
        level[root] = 0;

        while(!q.empty())
        {
            auto curr = q.front();
            q.pop();

            if (curr == child)
                return level[curr];

            if (curr->left) {
                level[curr->left] = level[curr] + 1;
                q.push(curr->left);
            }

            if (curr->right) {
                level[curr->right] = level[curr] + 1;
                q.push(curr->right);
            }
        }

        return -1;
    }

    unordered_map<TreeNode*, TreeNode*> generateParents(TreeNode *root)
    {
        // child -> parent
        unordered_map<TreeNode*, TreeNode*> getParent;

        queue<TreeNode*> q;
        q.push(root);
        getParent[root] = nullptr;

        while(!q.empty())
        {
            auto curr = q.front();
            q.pop();

            if (curr->left) {
                getParent[curr->left] = curr;
                q.push(curr->left);
            }

            if (curr->right) {
                getParent[curr->right] = curr;
                q.push(curr->right);
            }
        }

        return getParent;
    }

    TreeNode* moveKLevelsUp(TreeNode* child, int k, unordered_map<TreeNode*, TreeNode*> getParent
    {

```

```
        while (k != 0) {
            child = getParents[child];
            k--;
        }
        return child;
    }

TreeNode* returnCommonParent(TreeNode* child1, TreeNode *child2, unordered_map<TreeNode*, int> &getParents) {
    while (child1 != child2) {
        child1 = getParents[child1];
        child2 = getParents[child2];
    }
    return child1;
}

TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* child1, TreeNode* child2) {
    int depthOfChild1 = dep(root, child1);
    int depthOfChild2 = dep(root, child2);

    auto getParents = generateParents(root);

    int depthDifference = abs(depthOfChild1 - depthOfChild2);

    if (depthOfChild1 < depthOfChild2) {
        child2 = moveKLevelsUp(child2, depthDifference, getParents);
    } else if (depthOfChild1 > depthOfChild2) {
        child1 = moveKLevelsUp(child1, depthDifference, getParents);
    }

    return returnCommonParent(child1, child2, getParents, root);
}
};
```

## [16 3Sum Closest \(link\)](#)

### Description

Given an integer array `nums` of length `n` and an integer `target`, find three integers at **distinct indices** in `nums` such that the sum is closest to `target`.

Return *the sum of the three integers*.

You may assume that each input would have exactly one solution.

#### Example 1:

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

**Explanation:** The sum that is closest to the target is 2.  $(-1 + 2 + 1 = 2)$ .

#### Example 2:

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

**Explanation:** The sum that is closest to the target is 0.  $(0 + 0 + 0 = 0)$ .

#### Constraints:

- $3 \leq \text{nums.length} \leq 500$
- $-1000 \leq \text{nums}[i] \leq 1000$
- $-10^4 \leq \text{target} \leq 10^4$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution
{
public:
    int threeSumClosest(vector<int> &nums, int target)
    {
        sort(nums.begin(), nums.end());
        int mindif = INT_MAX, n = nums.size(), ans = 0;

        for (int i = 0; i < n - 2; i++)
        {
            int ptr1 = i + 1, ptr2 = n - 1;
            while (ptr1 < ptr2)
            {
                int sum = nums[i] + nums[ptr1] + nums[ptr2];

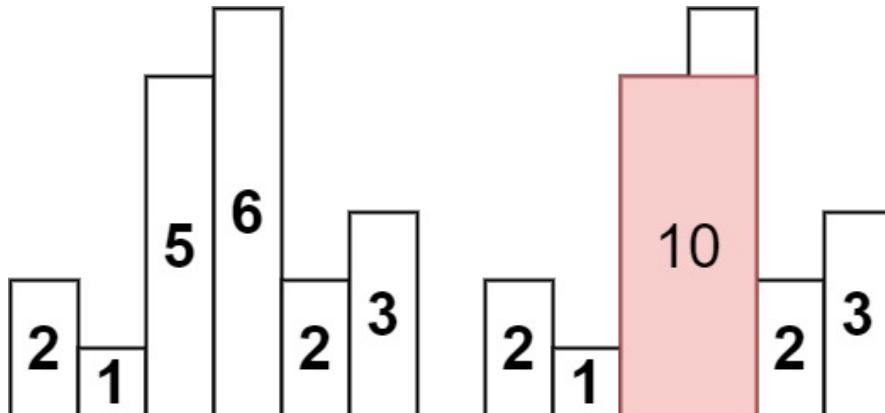
                if (sum == target)
                    return sum;
                if (mindif > abs(target - sum))
                {
                    mindif = abs(target - sum);
                    ans = sum;
                }
                if (sum > target)
                    --ptr2;
                else
                    ++ptr1;
            }
        }
        return ans;
    }
};
```

## [84 Largest Rectangle in Histogram \(link\)](#)

### Description

Given an array of integers heights representing the histogram's bar height where the width of each bar is 1, return *the area of the largest rectangle in the histogram*.

#### Example 1:



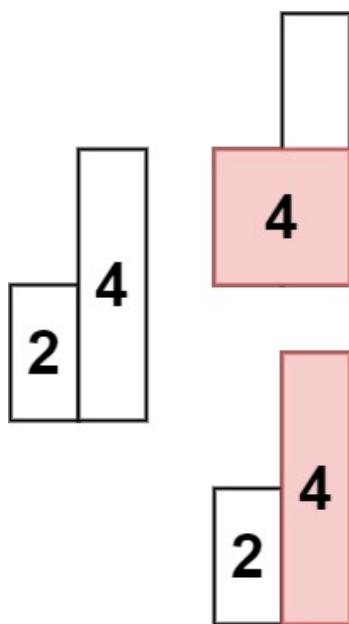
**Input:** heights = [2,1,5,6,2,3]

**Output:** 10

**Explanation:** The above is a histogram where width of each bar is 1.

The largest rectangle is shown in the red area, which has an area = 10 units.

#### Example 2:



**Input:** heights = [2,4]

**Output:** 4

### Constraints:

- $1 \leq \text{heights.length} \leq 10^5$
- $0 \leq \text{heights}[i] \leq 10^4$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    int min(int a, int b) {
        return a < b ? a : b;
    }

    int largestRectangleArea(vector<int> &heights) {
        if (heights.size() == 0) return 0;

        stack<int> height, index;
        int n = heights.size();

        int maxmArea = 0;

        // Initialise
        height.push(heights[0]);
        index.push(0);

        for (int i = 1; i < n; i++)
        {
            if (height.top() < heights[i])
            {
                height.push(heights[i]);
                index.push(i);
            }
            else
            {
                int pos = 0;
                while (height.top() >= heights[i])
                {
                    pos = index.top();
                    int area = height.top() * (i - pos);
                    if (maxmArea < area)
                        maxmArea = area;

                    height.pop();
                    index.pop();

                    if (height.size() == 0 && index.size() == 0)
                        break;
                }

                index.push(pos);
                height.push(heights[i]);
            }
        }

        while (!height.empty())
        {
            int area = height.top() * (heights.size() - index.top());
            if (area > maxmArea)
                maxmArea = area;
            height.pop();
            index.pop();
        }
    }

    return maxmArea;
};
```

## [1572 Subrectangle Queries \(link\)](#)

### Description

Implement the class `SubrectangleQueries` which receives a `rows x cols` rectangle as a matrix of integers in the constructor and supports two methods:

1. `updateSubrectangle(int row1, int col1, int row2, int col2, int newValue)`
  - Updates all values with `newValue` in the subrectangle whose upper left coordinate is `(row1, col1)` and bottom right coordinate is `(row2, col2)`.
2. `getValue(int row, int col)`
  - Returns the current value of the coordinate `(row, col)` from the rectangle.

### Example 1:

```

Input
["SubrectangleQueries","getValue","updateSubrectangle","getValue","getValue","updateSubrectangle"]
[[[[1,2,1],[4,3,4],[3,2,1],[1,1,1]]],[0,2],[0,0,3,2,5],[0,2],[3,1],[3,0,3,2,10],[3,1],[0,2]]
Output
[null,1,null,5,5,null,10,5]
Explanation
SubrectangleQueries subrectangleQueries = new SubrectangleQueries([[1,2,1],[4,3,4],[3,2,1],[1,1,1]]);
// The initial rectangle (4x3) looks like:
// 1 2 1
// 4 3 4
// 3 2 1
// 1 1 1
subrectangleQueries.getValue(0, 2); // return 1
subrectangleQueries.updateSubrectangle(0, 0, 3, 2, 5);
// After this update the rectangle looks like:
// 5 5 5
// 5 5 5
// 5 5 5
// 5 5 5
subrectangleQueries.getValue(0, 2); // return 5
subrectangleQueries.getValue(3, 1); // return 5
subrectangleQueries.updateSubrectangle(3, 0, 3, 2, 10);
// After this update the rectangle looks like:
// 5 5 5
// 5 5 5
// 5 5 5
// 10 10 10
subrectangleQueries.getValue(3, 1); // return 10
subrectangleQueries.getValue(0, 2); // return 5

```

### Example 2:

```

Input
["SubrectangleQueries","getValue","updateSubrectangle","getValue","getValue","updateSubrectangle"]
[[[[1,1,1],[2,2,2],[3,3,3]]],[0,0],[0,0,2,2,100],[0,0],[2,2],[1,1,2,2,20],[2,2]]
Output
[null,1,null,100,100,null,20]
Explanation
SubrectangleQueries subrectangleQueries = new SubrectangleQueries([[1,1,1],[2,2,2],[3,3,3]]);
subrectangleQueries.getValue(0, 0); // return 1
subrectangleQueries.updateSubrectangle(0, 0, 2, 2, 100);
subrectangleQueries.getValue(0, 0); // return 100
subrectangleQueries.getValue(2, 2); // return 100
subrectangleQueries.updateSubrectangle(1, 1, 2, 2, 20);
subrectangleQueries.getValue(2, 2); // return 20

```

### Constraints:

- There will be at most 500 operations considering both methods: `updateSubrectangle` and `getValue`.
- $1 \leq \text{rows}, \text{cols} \leq 100$

- `rows == rectangle.length`
- `cols == rectangle[i].length`
- `0 <= row1 <= row2 < rows`
- `0 <= col1 <= col2 < cols`
- `1 <= newValue, rectangle[i][j] <= 10^9`
- `0 <= row < rows`
- `0 <= col < cols`

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class SubrectangleQueries {
public:
    vector<vector<int>> rect;
    SubrectangleQueries(vector<vector<int>>& rectangle) {
        rect = rectangle;
    }
    void updateSubrectangle(int row1, int col1, int row2, int col2, int newValue) {
        for(int i = row1; i <= row2; i++) {
            for (int j = col1; j <= col2; j++)
                rect[i][j] = newValue;
        }
    }
    int getValue(int row, int col) {
        return rect[row][col];
    }
};

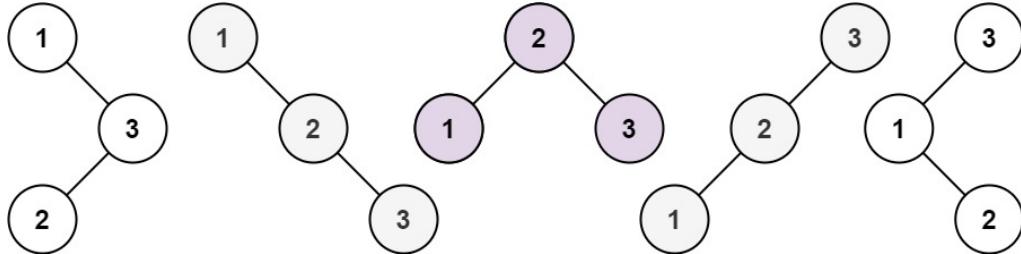
/**
 * Your SubrectangleQueries object will be instantiated and called as such:
 * SubrectangleQueries* obj = new SubrectangleQueries(rectangle);
 * obj->updateSubrectangle(row1,col1,row2,col2,newValue);
 * int param_2 = obj->getValue(row,col);
 */
```

## [96 Unique Binary Search Trees \(link\)](#)

### Description

Given an integer  $n$ , return *the number of structurally unique BST's (binary search trees) which has exactly  $n$  nodes of unique values from 1 to  $n$ .*

#### Example 1:



**Input:**  $n = 3$

**Output:** 5

#### Example 2:

**Input:**  $n = 1$

**Output:** 1

#### Constraints:

- $1 \leq n \leq 19$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

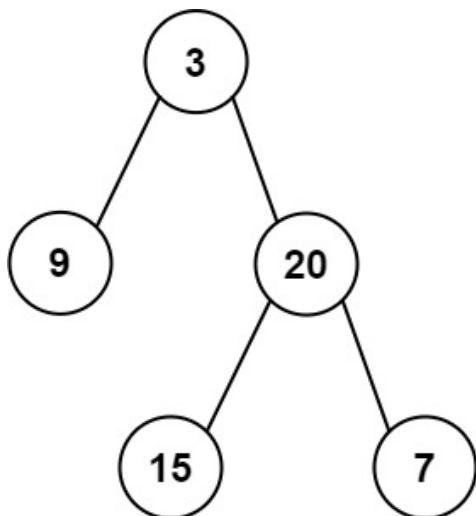
```
class Solution {
public:
    int numTrees(int n) {
        unsigned long c = 1;
        for(int i = 1; i < n; i++) {
            c *= 2;
            c *= (i + 1) * 2 - 1;
            c /= (i + 2);
        }
        return c;
    }
};
```

## 105 Construct Binary Tree from Preorder and Inorder Traversal (link)

### Description

Given two integer arrays `preorder` and `inorder` where `preorder` is the preorder traversal of a binary tree and `inorder` is the inorder traversal of the same tree, construct and return *the binary tree*.

#### Example 1:



**Input:** `preorder = [3,9,20,15,7]`, `inorder = [9,3,15,20,7]`  
**Output:** `[3,9,20,null,null,15,7]`

#### Example 2:

**Input:** `preorder = [-1]`, `inorder = [-1]`  
**Output:** `[-1]`

#### Constraints:

- $1 \leq \text{preorder.length} \leq 3000$
- $\text{inorder.length} == \text{preorder.length}$
- $-3000 \leq \text{preorder}[i], \text{inorder}[i] \leq 3000$
- `preorder` and `inorder` consist of **unique** values.
- Each value of `inorder` also appears in `preorder`.
- `preorder` is **guaranteed** to be the preorder traversal of the tree.
- `inorder` is **guaranteed** to be the inorder traversal of the tree.

(scroll down for solution)

# Solution

Language: cpp

Status: Accepted

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution
{
public:
    int index;
    unordered_map<int, int> table;

    /* Preorder Index
     * 9  -> 0
     * 3  -> 1
     * 15 -> 2
     * 20 -> 3
     * 7   -> 4
     */

    TreeNode *buildTree(vector<int> &preorder, vector<int> &inorder)
    {
        index = 0;

        for (int i = 0; i < inorder.size(); ++i)
            table[inorder[i]] = i;

        return divideAndConquer(preorder, 0, inorder.size() - 1);
    }

    TreeNode *divideAndConquer(vector<int> &pre, int l, int r)
    {
        if (l > r)
            return NULL;

        TreeNode *head = new TreeNode(pre[index]);
        index++;

        int root = table[head->val];

        head->left = divideAndConquer(pre, l, root - 1);
        head->right = divideAndConquer(pre, root + 1, r);

        return head;
    }
};
```

## [172 Factorial Trailing Zeroes \(link\)](#)

### Description

Given an integer  $n$ , return *the number of trailing zeroes in  $n!$* .

Note that  $n! = n * (n - 1) * (n - 2) * \dots * 3 * 2 * 1$ .

#### Example 1:

```
Input: n = 3
Output: 0
Explanation: 3! = 6, no trailing zero.
```

#### Example 2:

```
Input: n = 5
Output: 1
Explanation: 5! = 120, one trailing zero.
```

#### Example 3:

```
Input: n = 0
Output: 0
```

#### Constraints:

- $0 \leq n \leq 10^4$

**Follow up:** Could you write a solution that works in logarithmic time complexity?

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution
{
public:
    int trailingZeroes(int n, int k = 5)
    {
        if ((n / k) == 0)
            return 0;
        return (n / k) + trailingZeroes(n, k * 5);
    };
}
```

## [168 Excel Sheet Column Title \(link\)](#)

### Description

Given an integer `columnNumber`, return *its corresponding column title as it appears in an Excel sheet*.

For example:

```
A -> 1
B -> 2
C -> 3
...
Z -> 26
AA -> 27
AB -> 28
...
```

#### Example 1:

```
Input: columnNumber = 1
Output: "A"
```

#### Example 2:

```
Input: columnNumber = 28
Output: "AB"
```

#### Example 3:

```
Input: columnNumber = 701
Output: "ZY"
```

#### Constraints:

- $1 \leq \text{columnNumber} \leq 2^{31} - 1$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution
{
public:
    string convertToTitle(int n)
    {
        if (n == 0)
            return "";
        return convertToTitle((n - 1) / 26) + (char)((n - 1) % 26 + 'A');
    }
};
```

## [67 Add Binary \(link\)](#)

### Description

Given two binary strings  $a$  and  $b$ , return *their sum as a binary string*.

#### Example 1:

```
Input: a = "11", b = "1"
Output: "100"
```

#### Example 2:

```
Input: a = "1010", b = "1011"
Output: "10101"
```

#### Constraints:

- $1 \leq a.length, b.length \leq 10^4$
- $a$  and  $b$  consist only of '0' or '1' characters.
- Each string does not contain leading zeros except for the zero itself.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution
{
public:
    string addBinary(string a, string b)
    {
        string result = "";
        int len1 = a.length();
        int len2 = b.length();

        int sum = 0, carry = 0;

        for (int i = len1 - 1, j = len2 - 1; i >= 0 || j >= 0; i--, j--)
        {
            sum = carry;
            if (i >= 0)
                sum += a[i] - '0';

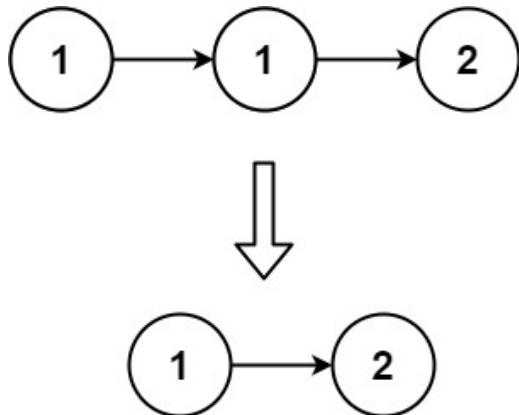
            if (j >= 0)
                sum += b[j] - '0';
            result += to_string(sum % 2);
            carry = sum / 2;
        }
        if (carry > 0)
            result += '1';
        reverse(result.begin(), result.end());
        return result;
    }
};
```

## [83 Remove Duplicates from Sorted List \(link\)](#)

### Description

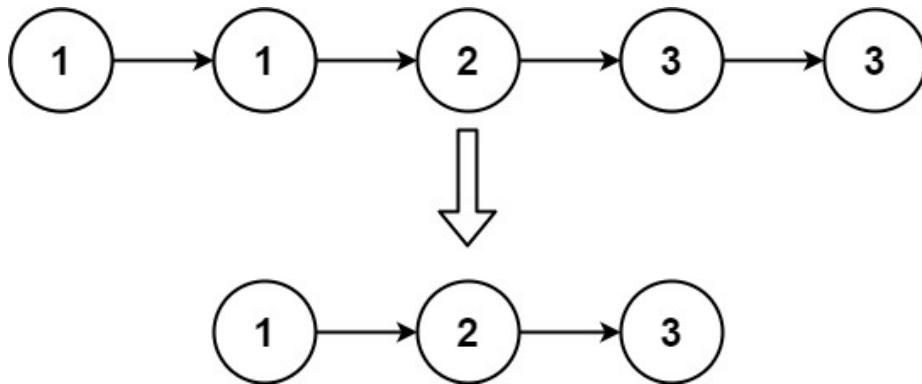
Given the head of a sorted linked list, *delete all duplicates such that each element appears only once*. Return *the linked list sorted* as well.

#### Example 1:



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

#### Example 2:



**Input:** head = [1,1,2,3,3]  
**Output:** [1,2,3]

### Constraints:

- The number of nodes in the list is in the range [0, 300].
- $-100 \leq \text{Node.val} \leq 100$
- The list is guaranteed to be **sorted** in ascending order.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
/*
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution
{
public:
    ListNode *deleteDuplicates(ListNode *head)
    {
        if (head == nullptr)
            return nullptr;

        ListNode *curr = head;
        while (curr->next != nullptr)
        {
            if (curr->next != nullptr && curr->next->val == curr->val)
                curr->next = curr->next->next;
            else
                curr = curr->next;
        }

        return head;
    }
};
```

## [1787 Sum of Absolute Differences in a Sorted Array \(link\)](#)

### Description

You are given an integer array `nums` sorted in **non-decreasing** order.

Build and return *an integer array result with the same length as nums such that result[i] is equal to the summation of absolute differences between nums[i] and all the other elements in the array.*

In other words, `result[i]` is equal to `sum(|nums[i]-nums[j]|)` where  $0 \leq j < \text{nums.length}$  and  $j \neq i$  (**0-indexed**).

#### Example 1:

```
Input: nums = [2,3,5]
Output: [4,3,5]
Explanation: Assuming the arrays are 0-indexed, then
result[0] = |2-2| + |2-3| + |2-5| = 0 + 1 + 3 = 4,
result[1] = |3-2| + |3-3| + |3-5| = 1 + 0 + 2 = 3,
result[2] = |5-2| + |5-3| + |5-5| = 3 + 2 + 0 = 5.
```

#### Example 2:

```
Input: nums = [1,4,6,8,10]
Output: [24,15,13,15,21]
```

#### Constraints:

- $2 \leq \text{nums.length} \leq 10^5$
- $1 \leq \text{nums}[i] \leq \text{nums}[i + 1] \leq 10^4$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    vector<int> getSumAbsoluteDifferences(vector<int>& nums) {
        vector<int> sums(nums.size());
        sums[0] = nums[0];
        for(int i = 1; i < nums.size(); i++)
            sums[i] = nums[i] + sums[i-1];

        vector<int> result(nums.size());

        int totalsum = 0;
        for(auto i: nums) totalsum += i;
        result[0] = totalsum - (nums[0] * nums.size());

        for (int i = 1; i < nums.size(); i++) {
            result[i] = (i * nums[i]) - sums[i-1];
            result[i] += (sums[nums.size() - 1] - sums[i]) - ((nums.size() - 1 - i) * nums[i]);
        }

        return result;
    }
};
```

## [1786 Count the Number of Consistent Strings \(link\)](#)

### Description

You are given a string `allowed` consisting of **distinct** characters and an array of strings `words`. A string is **consistent** if all characters in the string appear in the string `allowed`.

Return *the number of consistent strings in the array words*.

#### Example 1:

```
Input: allowed = "ab", words = ["ad", "bd", "aaab", "baa", "badab"]
Output: 2
```

**Explanation:** Strings "aaab" and "baa" are consistent since they only contain characters 'a' and 'b'.

#### Example 2:

```
Input: allowed = "abc", words = ["a", "b", "c", "ab", "ac", "bc", "abc"]
Output: 7
```

**Explanation:** All strings are consistent.

#### Example 3:

```
Input: allowed = "cad", words = ["cc", "acd", "b", "ba", "bac", "bad", "ac", "d"]
Output: 4
```

**Explanation:** Strings "cc", "acd", "ac", and "d" are consistent.

### Constraints:

- $1 \leq \text{words.length} \leq 10^4$
- $1 \leq \text{allowed.length} \leq 26$
- $1 \leq \text{words[i].length} \leq 10$
- The characters in `allowed` are **distinct**.
- `words[i]` and `allowed` contain only lowercase English letters.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    int countConsistentStrings(string allowed, vector<string>& words) {
        unordered_map<char, bool> isal;
        for (char ch : allowed)
            isal[ch] = true;

        int count = 0;
        for (string word: words) {
            bool flag = false;
            for (char ch: word)
                if (!isal[ch])
                    flag = true;
            if (!flag)
                ++count;
        }
        return count;
    }
};
```

## [75 Sort Colors \(link\)](#)

### Description

Given an array `nums` with `n` objects colored red, white, or blue, sort them [in-place](#) so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

#### Example 1:

```
Input: nums = [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]
```

#### Example 2:

```
Input: nums = [2,0,1]
Output: [0,1,2]
```

#### Constraints:

- `n == nums.length`
- `1 <= n <= 300`
- `nums[i]` is either 0, 1, or 2.

**Follow up:** Could you come up with a one-pass algorithm using only constant extra space?

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution
{
public:
    void sortColors(vector<int> &nums)
    {
        unordered_map<int, int> count;
        for (int i = 0; i < nums.size(); i++)
            count[nums[i]]++;

        int zeros = count[0];
        int ones = count[1];
        int twos = count[2];

        for (int i = 0; i < nums.size(); i++)
        {
            if (i < zeros)
                nums[i] = 0;
            else if (i >= zeros && i < zeros + ones)
                nums[i] = 1;
            else
                nums[i] = 2;
        }
    }
};
```

## [74 Search a 2D Matrix \(link\)](#)

### Description

You are given an  $m \times n$  integer matrix `matrix` with the following two properties:

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

Given an integer `target`, return `true` if `target` is in `matrix` or `false` otherwise.

You must write a solution in  $O(\log(m * n))$  time complexity.

#### Example 1:

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

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

#### Example 2:

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

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

#### Constraints:

- $m == \text{matrix.length}$
- $n == \text{matrix[i].length}$
- $1 \leq m, n \leq 100$
- $-10^4 \leq \text{matrix[i][j]}, \text{target} \leq 10^4$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution
{
public:
    bool searchMatrix(vector<vector<int>> &matrix, int target)
    {

        if (matrix.size() == 0 || matrix[0].size() == 0)
            return false;

        int rowSize = matrix.size();
        int colSize = matrix[0].size();

        // for (int i = 0; i < rowSize;
        int i = 0, j = colSize - 1;
        while (i < rowSize && j >= 0)
        {
            // for (int j = colSize - 1; j >= 0;
            // {
            if (matrix[i][j] == target)
                return true;
            else if (matrix[i][j] < target)
                i++;
            else
                j--;
            // }
        }
        return false;
    }
};
```

## [31 Next Permutation \(link\)](#)

### Description

A **permutation** of an array of integers is an arrangement of its members into a sequence or linear order.

- For example, for `arr = [1,2,3]`, the following are all the permutations of `arr`: `[1,2,3]`, `[1,3,2]`, `[2, 1, 3]`, `[2, 3, 1]`, `[3,1,2]`, `[3,2,1]`.

The **next permutation** of an array of integers is the next lexicographically greater permutation of its integer. More formally, if all the permutations of the array are sorted in one container according to their lexicographical order, then the **next permutation** of that array is the permutation that follows it in the sorted container. If such arrangement is not possible, the array must be rearranged as the lowest possible order (i.e., sorted in ascending order).

- For example, the next permutation of `arr = [1,2,3]` is `[1,3,2]`.
- Similarly, the next permutation of `arr = [2,3,1]` is `[3,1,2]`.
- While the next permutation of `arr = [3,2,1]` is `[1,2,3]` because `[3,2,1]` does not have a lexicographical larger rearrangement.

Given an array of integers `nums`, *find the next permutation of `nums`*.

The replacement must be **in place** and use only constant extra memory.

#### Example 1:

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

#### Example 2:

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

#### Example 3:

```
Input: nums = [1,1,5]
Output: [1,5,1]
```

#### Constraints:

- `1 <= nums.length <= 100`
- `0 <= nums[i] <= 100`

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution
{
public:
    void nextPermutation(vector<int> &nums)
    {
        next_permutation(nums.begin(), nums.end());
    }
};
```

## [56 Merge Intervals \(link\)](#)

### Description

Given an array of intervals where  $\text{intervals}[i] = [\text{start}_i, \text{end}_i]$ , merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

#### Example 1:

```
Input: intervals = [[1,3],[2,6],[8,10],[15,18]]  
Output: [[1,6],[8,10],[15,18]]  
Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].
```

#### Example 2:

```
Input: intervals = [[1,4],[4,5]]  
Output: [[1,5]]  
Explanation: Intervals [1,4] and [4,5] are considered overlapping.
```

#### Example 3:

```
Input: intervals = [[4,7],[1,4]]  
Output: [[1,7]]  
Explanation: Intervals [1,4] and [4,7] are considered overlapping.
```

### Constraints:

- $1 \leq \text{intervals.length} \leq 10^4$
- $\text{intervals}[i].length == 2$
- $0 \leq \text{start}_i \leq \text{end}_i \leq 10^4$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution
{
public:
    int max(int a, int b)
    {
        return a < b ? b : a;
    }

    vector<vector<int>> merge(vector<vector<int>> intervals)
    {
        if (intervals.size() == 0)
            return {};

        sort(intervals.begin(), intervals.end());
        /*
        Comparator Function:
            [](vector<int> a, vector<int> b)
            {
                if (a[0] == b[0])
                    { return a[1] < b[1]; }
                else
                    { return a[0] < b[0]; }
            }
        */

        vector<vector<int>> result;
        vector<int> current_interval(2);

        current_interval[0] = intervals[0][0];
        current_interval[1] = intervals[0][1];

        for (int i = 1; i < intervals.size(); i++)
        {
            // If the ending of the curr interval is greater than the starting of the next
            if (current_interval[1] >= intervals[i][0])
                current_interval[1] = max(intervals[i][1], current_interval[1]);

            else
            {
                result.push_back(current_interval);
                current_interval[0] = intervals[i][0];
                current_interval[1] = max(intervals[i][1], current_interval[1]);
            }
        }

        if (result.size() == 0)
            result.push_back(current_interval);

        if (result[max(result.size() - 1, 0)] != current_interval)
            result.push_back(current_interval);

        return result;
    }
};
```

## [55 Jump Game \(link\)](#)

### Description

You are given an integer array `nums`. You are initially positioned at the array's **first index**, and each element in the array represents your maximum jump length at that position.

Return `true` if you can reach the last index, or `false` otherwise.

#### Example 1:

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

**Output:** `true`

**Explanation:** Jump 1 step from index 0 to 1, then 3 steps to the last index.

#### Example 2:

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

**Output:** `false`

**Explanation:** You will always arrive at index 3 no matter what. Its maximum jump length is 0, which is not enough to reach the last index.

#### Constraints:

- $1 \leq \text{nums.length} \leq 10^4$
- $0 \leq \text{nums}[i] \leq 10^5$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution
{
public:
    int min(int a, int b)
    {
        return a < b ? a : b;
    }

    bool canJump(vector<int> &nums)
    {
        if (nums.size() > 10000 && nums[0] != 1)
            return false;

        vector<bool> canReachTillEnd(nums.size(), false);
        canReachTillEnd[nums.size() - 1] = true;

        for (int i = nums.size() - 2; i >= 0; i--)
        {
            // If from the current Position, we can reach till the end
            if ((i + nums[i]) >= (nums.size() - 1))
                canReachTillEnd[i] = true;

            // If from the current position, we can reach a position from where
            // we can reach till end
            else if (canReachTillEnd[min(i + nums[i], nums.size() - 1)])
                canReachTillEnd[i] = true;

            else
                for (int j = i; j < i + nums[i]; j++)
                    canReachTillEnd[i] = canReachTillEnd[j] || canReachTillEnd[i];

            // // If we cant reach a position which can reach till end, return false
            // else
            //     continue;
        }

        // return true;
        return canReachTillEnd[0];
    }
};
```

## [153 Find Minimum in Rotated Sorted Array \(link\)](#)

### Description

Suppose an array of length  $n$  sorted in ascending order is **rotated** between 1 and  $n$  times. For example, the array `nums` = `[0,1,2,4,5,6,7]` might become:

- `[4,5,6,7,0,1,2]` if it was rotated 4 times.
- `[0,1,2,4,5,6,7]` if it was rotated 7 times.

Notice that **rotating** an array `[a[0], a[1], a[2], ..., a[n-1]]` 1 time results in the array `[a[n-1], a[0], a[1], a[2], ..., a[n-2]]`.

Given the sorted rotated array `nums` of **unique** elements, return *the minimum element of this array*.

You must write an algorithm that runs in  $O(\log n)$  time.

#### Example 1:

```
Input: nums = [3,4,5,1,2]
Output: 1
Explanation: The original array was [1,2,3,4,5] rotated 3 times.
```

#### Example 2:

```
Input: nums = [4,5,6,7,0,1,2]
Output: 0
Explanation: The original array was [0,1,2,4,5,6,7] and it was rotated 4 times.
```

#### Example 3:

```
Input: nums = [11,13,15,17]
Output: 11
Explanation: The original array was [11,13,15,17] and it was rotated 4 times.
```

### Constraints:

- $n == \text{nums.length}$
- $1 \leq n \leq 5000$
- $-5000 \leq \text{nums}[i] \leq 5000$
- All the integers of `nums` are **unique**.
- `nums` is sorted and rotated between 1 and  $n$  times.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution
{
public:
    int findMin(vector<int> &nums)
    {
        if (is_sorted(nums.begin(), nums.end()) || nums.size() == 1)
            return nums[0];

        int left = 0, right = nums.size() - 1, mid = left + (right - left) / 2;
        while (left <= right)
        {
            mid = left + (right - left) / 2;
            if (nums[mid] > nums[mid + 1])
                return nums[mid + 1];
            if (nums[mid - 1] > nums[mid + 1])
                return nums[mid];
            else if (nums[mid] < nums[mid + 1] && nums[mid] < nums[0])
                right = mid - 1;
            else if (nums[mid] < nums[mid + 1] && nums[mid] > nums[0])
                left = mid + 1;
        }
        return -1;
    };
};
```

## [71 Simplify Path \(link\)](#)

### Description

You are given an *absolute* path for a Unix-style file system, which always begins with a slash '/'. Your task is to transform this absolute path into its **simplified canonical path**.

The *rules* of a Unix-style file system are as follows:

- A single period '.' represents the current directory.
- A double period '..' represents the previous/parent directory.
- Multiple consecutive slashes such as '///' and '////' are treated as a single slash '/'.
- Any sequence of periods that does **not match** the rules above should be treated as a **valid directory or file name**. For example, '....' and '.....' are valid directory or file names.

The simplified canonical path should follow these *rules*:

- The path must start with a single slash '/'.
- Directories within the path must be separated by exactly one slash '/'.
- The path must not end with a slash '/', unless it is the root directory.
- The path must not have any single or double periods ('.' and '..') used to denote current or parent directories.

Return the **simplified canonical path**.

#### Example 1:

**Input:** path = "/home/"

**Output:** "/home"

#### Explanation:

The trailing slash should be removed.

#### Example 2:

**Input:** path = "/home//foo/"

**Output:** "/home/foo"

#### Explanation:

Multiple consecutive slashes are replaced by a single one.

#### Example 3:

**Input:** path = "/home/user/Documents/..Pictures"

**Output:** "/home/user/Pictures"

#### Explanation:

A double period ".." refers to the directory up a level (the parent directory).

#### Example 4:

**Input:** path = "/../"

**Output:** "/"

#### Explanation:

Going one level up from the root directory is not possible.

#### Example 5:

**Input:** path = "/.../a/..../b/c/..../d/./"

**Output:** "/.../b/d"

**Explanation:**

"...." is a valid name for a directory in this problem.

**Constraints:**

- $1 \leq \text{path.length} \leq 3000$
- path consists of English letters, digits, period '.', slash '/' or '\_'.
  - path is a valid absolute Unix path.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```

class Solution
{
public:
    string simplifyPath(string path)
    {
        // break into chunks - chunk = anything between //
        // if we encounter a normal token - push to stack
        // if we encounter a . token - do nothing | continue
        // if we encounter a .. token - pop from stack
        // if .. and stack empty - do nothing

        // after the end - build the path from the stack such as :-
        /*
            / token1 / token2 / token3 / token4
        */

        deque<string> tokens;
        string temp = "";

        int len = path.length() - 1;
        if (path[len] != '/')
            path.push_back('/');

        for (int i = 0; i < path.length(); i++)
        {
            if (path[i] == '/')
            {
                tokens.push_back(temp);
                temp = "";
            }

            else if (i == path.length() - 1)
                tokens.push_back(temp);

            else
                temp.push_back(path[i]);
        }

        tokens.pop_front();

        deque<string> structure;
        string properPath = "/";

        for (string token : tokens)
        {
            if (token == "." || (token == ".." && structure.empty()))
                continue;

            if (token == "..")
                structure.pop_back();
            else if (token != "")
                structure.push_back(token);
        }

        for (string dir : structure)
            properPath += dir + "/";

        if (properPath != "/")
            properPath.pop_back();
        return properPath;
    }
};

```

## [50 Pow\(x, n\) \(link\)](#)

### Description

Implement [pow\(x, n\)](#), which calculates  $x$  raised to the power  $n$  (i.e.,  $x^n$ ).

#### Example 1:

```
Input: x = 2.00000, n = 10
Output: 1024.00000
```

#### Example 2:

```
Input: x = 2.10000, n = 3
Output: 9.26100
```

#### Example 3:

```
Input: x = 2.00000, n = -2
Output: 0.25000
Explanation:  $2^{-2} = 1/2^2 = 1/4 = 0.25$ 
```

### Constraints:

- $-100.0 < x < 100.0$
- $-2^{31} \leq n \leq 2^{31}-1$
- $n$  is an integer.
- Either  $x$  is not zero or  $n > 0$ .
- $-10^4 \leq x^n \leq 10^4$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    double myPow(double x, long long int n) {
        if (x == 0)
            return 0;

        else if (n < 0)
        {
            x = myPow(x, -1 * n);
            return 1 / x;
        }

        else if (n == 0)
            return 1.00;

        else if (n % 2 == 0)
        {
            x = myPow(x, n / 2);
            return x * x;
        }

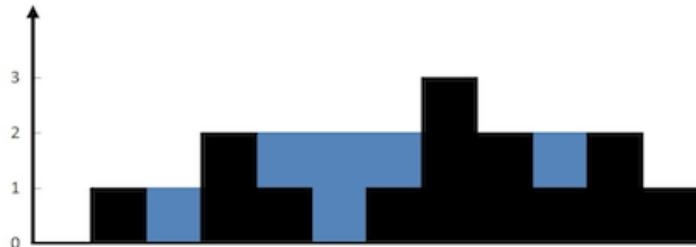
        else
        {
            x = x * myPow(x, n - 1);
            return x;
        }
    };
};
```

## [42 Trapping Rain Water \(link\)](#)

### Description

Given  $n$  non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

#### Example 1:



**Input:** height = [0,1,0,2,1,0,1,3,2,1,2,1]

**Output:** 6

**Explanation:** The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1].

#### Example 2:

**Input:** height = [4,2,0,3,2,5]

**Output:** 9

#### Constraints:

- $n == \text{height.length}$
- $1 \leq n \leq 2 * 10^4$
- $0 \leq \text{height}[i] \leq 10^5$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution
{
public:
    int firstNonZeroIndex(vector<int> arr)
    {
        for (int i = 0; i < arr.size(); i++)
            if (arr[i] > 0)
                return i;

        // all array is zero
        return -1;
    }

    int trap(vector<int> &height)
    {
        if (height.size() == 0)
            return 0;

        int sum = 0;

        vector<int> leftMaxs(height.size());
        vector<int> rightMaxs(height.size());

        leftMaxs[0] = height[0];
        rightMaxs[height.size() - 1] = height[height.size() - 1];

        for (int i = 1; i < height.size(); i++)
            leftMaxs[i] = max(leftMaxs[i - 1], height[i]);

        for (int i = height.size() - 2; i >= 0; i--)
            rightMaxs[i] = max(rightMaxs[i + 1], height[i]);

        for (int i = 0; i < height.size(); i++)
            sum += min(leftMaxs[i], rightMaxs[i]) - height[i];

        return sum;
    }
};
```

## [242 Valid Anagram \(link\)](#)

### Description

Given two strings  $s$  and  $t$ , return `true` if  $t$  is an anagram of  $s$ , and `false` otherwise.

#### Example 1:

**Input:**  $s = \text{"anagram"}$ ,  $t = \text{"nagaram"}$

**Output:** `true`

#### Example 2:

**Input:**  $s = \text{"rat"}$ ,  $t = \text{"car"}$

**Output:** `false`

#### Constraints:

- $1 \leq s.length, t.length \leq 5 * 10^4$
- $s$  and  $t$  consist of lowercase English letters.

**Follow up:** What if the inputs contain Unicode characters? How would you adapt your solution to such a case?

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    bool isAnagram(string s, string t) {
        unordered_map<char, int> freq1;
        unordered_map<char, int> freq2;

        for (auto i: s)
            freq1[i]++;
        for (auto i: t)
            freq2[i]++;
        for (auto i: freq1)
            if (freq2[i.first] != i.second)
                return false;
        for (auto i : freq2)
            if (freq1[i.first] != i.second)
                return false;
        return true;
    };
};
```

## [155 Min Stack \(link\)](#)

### Description

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

Implement the `MinStack` class:

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

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

### Example 1:

**Input**  
["MinStack","push","push","push","getMin","pop","top","getMin"]  
[[], [-2], [0], [-3], [], [], [], []]

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

**Explanation**  
MinStack minStack = new MinStack();  
minStack.push(-2);  
minStack.push(0);  
minStack.push(-3);  
minStack.getMin(); // return -3  
minStack.pop();  
minStack.top(); // return 0  
minStack.getMin(); // return -2

### Constraints:

- $-2^{31} \leq \text{val} \leq 2^{31} - 1$
- Methods `pop`, `top` and `getMin` operations will always be called on **non-empty** stacks.
- At most  $3 * 10^4$  calls will be made to `push`, `pop`, `top`, and `getMin`.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class MinStack
{
    stack<long long int> minVals;
    stack<long long int> actualStack;

public:
    /** initialize your data structure here. */
    MinStack()
    {
        // minVals.push(INT_MAX);
    }

    void push(int x)
    {
        actualStack.push(x);

        if (minVals.empty())
            minVals.push(x);
        else if (x <= minVals.top())
            minVals.push(x);
    }

    void pop()
    {
        int toBeDeleted = actualStack.top();
        actualStack.pop();

        if (toBeDeleted == minVals.top())
            minVals.pop();
    }

    int top()
    {
        return actualStack.top();
    }

    int getMin()
    {
        if (minVals.empty())
            return 0;

        return minVals.top();
    }
};

/**
 * Your MinStack object will be instantiated and called as such:
 * MinStack* obj = new MinStack();
 * obj->push(x);
 * obj->pop();
 * int param_3 = obj->top();
 * int param_4 = obj->getMin();
 */

```

## [204 Count Primes \(link\)](#)

### Description

Given an integer  $n$ , return *the number of prime numbers that are strictly less than  $n$* .

#### Example 1:

**Input:**  $n = 10$

**Output:** 4

**Explanation:** There are 4 prime numbers less than 10, they are 2, 3, 5, 7.

#### Example 2:

**Input:**  $n = 0$

**Output:** 0

#### Example 3:

**Input:**  $n = 1$

**Output:** 0

### Constraints:

- $0 \leq n \leq 5 * 10^6$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution
{
public:
    int sum(vector<int> dp)
    {
        int sum = 0;
        for (auto i : dp)
            sum += i;
        return sum;
    }

    int countPrimes(int n)
    {
        if (n <= 2)
            return 0;
        vector<int> dp(n, 1);

        for (int i = 2; i * i <= n; i++)
            for (int j = i + i; j < n; j += i)
                dp[j] = 0;

        return sum(dp) - 2;
    }
};
```

## [238 Product of Array Except Self \(link\)](#)

### Description

Given an integer array `nums`, return *an array answer such that answer[i] is equal to the product of all the elements of nums except nums[i]*.

The product of any prefix or suffix of `nums` is **guaranteed** to fit in a **32-bit** integer.

You must write an algorithm that runs in  $O(n)$  time and without using the division operation.

#### Example 1:

```
Input: nums = [1,2,3,4]
Output: [24,12,8,6]
```

#### Example 2:

```
Input: nums = [-1,1,0,-3,3]
Output: [0,0,9,0,0]
```

#### Constraints:

- $2 \leq \text{nums.length} \leq 10^5$
- $-30 \leq \text{nums}[i] \leq 30$
- The input is generated such that `answer[i]` is **guaranteed** to fit in a **32-bit** integer.

**Follow up:** Can you solve the problem in  $O(1)$  extra space complexity? (The output array **does not** count as extra space for space complexity analysis.)

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution
{
public:
    vector<int> productExceptSelf(vector<int> &nums)
    {
        vector<int> leftProd(nums.size());
        vector<int> rightProd(nums.size());

        leftProd[0] = 1;
        rightProd[nums.size() - 1] = 1;
        for (int i = 1; i < nums.size(); i++)
            leftProd[i] = leftProd[i - 1] * nums[i - 1];

        for (int i = nums.size() - 2; i >= 0; i--)
            rightProd[i] = rightProd[i + 1] * nums[i + 1];

        for (int i = 0; i < nums.size(); i++)
            leftProd[i] = leftProd[i] * rightProd[i];

        return leftProd;
    }
};
```

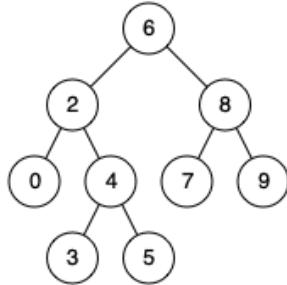
## [235 Lowest Common Ancestor of a Binary Search Tree \(link\)](#)

### Description

Given a binary search tree (BST), find the lowest common ancestor (LCA) node of two given nodes in the BST.

According to the [definition of LCA on Wikipedia](#): “The lowest common ancestor is defined between two nodes  $p$  and  $q$  as the lowest node in  $T$  that has both  $p$  and  $q$  as descendants (where we allow a node to be a descendant of itself).”

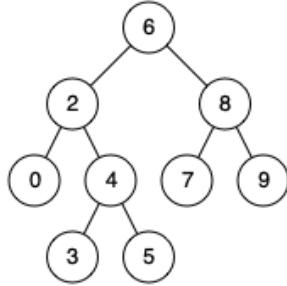
#### Example 1:



**Input:** root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8  
**Output:** 6

**Explanation:** The LCA of nodes 2 and 8 is 6.

#### Example 2:



**Input:** root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4  
**Output:** 2

**Explanation:** The LCA of nodes 2 and 4 is 2, since a node can be a descendant of itself according to the problem description.

#### Example 3:

**Input:** root = [2,1], p = 2, q = 1  
**Output:** 2

#### Constraints:

- The number of nodes in the tree is in the range  $[2, 10^5]$ .
- $-10^9 \leq \text{Node.val} \leq 10^9$
- All `Node.val` are **unique**.
- $p \neq q$
- $p$  and  $q$  will exist in the BST.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */

class Solution
{
public:
    TreeNode *lowestCommonAncestor(TreeNode *root, TreeNode *p, TreeNode *q)
    {
        if (root->val < p->val && root->val < q->val)
            return lowestCommonAncestor(root->right, p, q);

        if (root->val > p->val && root->val > q->val)
            return lowestCommonAncestor(root->left, p, q);

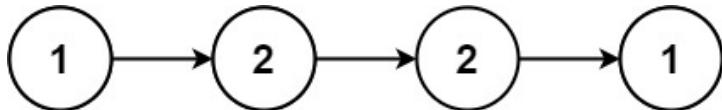
        else
            return root;
    }
};
```

## [234 Palindrome Linked List \(link\)](#)

### Description

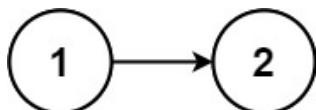
Given the head of a singly linked list, return `true` if it is a palindrome or `false` otherwise.

#### Example 1:



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

#### Example 2:



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

#### Constraints:

- The number of nodes in the list is in the range  $[1, 10^5]$ .
- $0 \leq \text{Node.val} \leq 9$

**Follow up:** Could you do it in  $O(n)$  time and  $O(1)$  space?

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution
{
public:
    bool isPalindrome(ListNode *head)
    {
        if (head == nullptr || head->next == nullptr)
            return true;

        if (head->next->next == nullptr)
        {
            if (head->next->val == head->val)
                return true;
            return false;
        }

        ListNode *slowpointer = head, *fastpointer = head;
        stack<int> storage;

        while (fastpointer->next != nullptr)
        {
            storage.push(slowpointer->val);
            slowpointer = slowpointer->next;

            if (fastpointer->next->next != nullptr)
                fastpointer = fastpointer->next->next;
            else
                break;
        }

        if (!fastpointer->next)
            slowpointer = slowpointer->next;

        while (slowpointer != nullptr)
        {
            int curr = storage.top();
            storage.pop();

            if (curr != slowpointer->val)
                return false;

            slowpointer = slowpointer->next;
        }

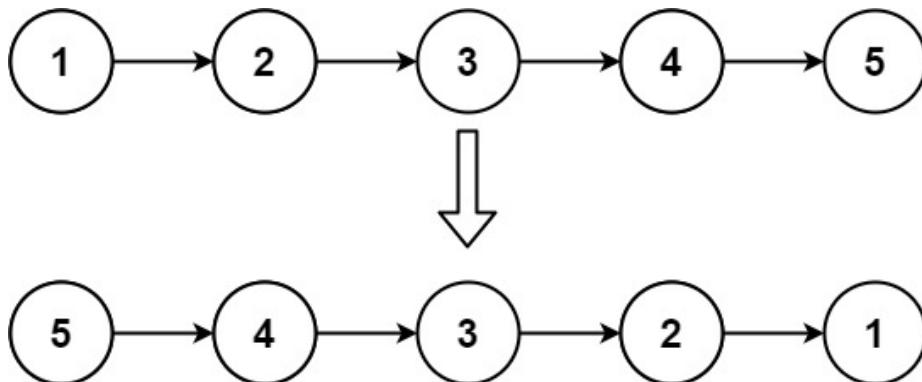
        return storage.empty();
    };
}
```

## 206 Reverse Linked List (link)

### Description

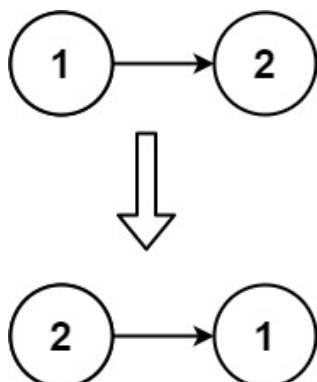
Given the head of a singly linked list, reverse the list, and return *the reversed list*.

#### Example 1:



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

#### Example 2:



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

#### Example 3:

**Input:** head = []  
**Output:** []

### Constraints:

- The number of nodes in the list is in the range [0, 5000].
- $-5000 \leq \text{Node.val} \leq 5000$

**Follow up:** A linked list can be reversed either iteratively or recursively. Could you implement both?

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
/*
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution
{
public:
    ListNode *reverseList(ListNode *head)
    {
        // if (head == nullptr || head->next == nullptr)
        //     return head;

        // ListNode *remainingElements = reverseList(head->next);
        // head->next->next = head;
        // head->next = nullptr;
        // return remainingElements;

        ListNode *current = head;
        ListNode *prev = NULL, *next = NULL;

        while (current != NULL)
        {
            // Store next
            next = current->next;

            // Reverse current node's pointer
            current->next = prev;

            // Move pointers one position ahead.
            prev = current;
            current = next;
        }
        head = prev;
        return head;
    }
};
```

## [189 Rotate Array \(link\)](#)

### Description

Given an integer array `nums`, rotate the array to the right by `k` steps, where `k` is non-negative.

#### Example 1:

```
Input: nums = [1,2,3,4,5,6,7], k = 3
Output: [5,6,7,1,2,3,4]
Explanation:
rotate 1 steps to the right: [7,1,2,3,4,5,6]
rotate 2 steps to the right: [6,7,1,2,3,4,5]
rotate 3 steps to the right: [5,6,7,1,2,3,4]
```

#### Example 2:

```
Input: nums = [-1,-100,3,99], k = 2
Output: [3,99,-1,-100]
Explanation:
rotate 1 steps to the right: [99,-1,-100,3]
rotate 2 steps to the right: [3,99,-1,-100]
```

#### Constraints:

- $1 \leq \text{nums.length} \leq 10^5$
- $-2^{31} \leq \text{nums}[i] \leq 2^{31} - 1$
- $0 \leq k \leq 10^5$

#### Follow up:

- Try to come up with as many solutions as you can. There are at least **three** different ways to solve this problem.
- Could you do it in-place with  $O(1)$  extra space?

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution
{
public:
    void rotate(vector<int> &nums, int k)
    {
        if (k > nums.size())
            k = k % nums.size();

        reverse(nums.begin(), nums.end());
        reverse(nums.begin(), nums.begin() + k);
        reverse(nums.begin() + k, nums.end());
    }
};
```

## [167 Two Sum II - Input Array Is Sorted \(link\)](#)

### Description

Given a **1-indexed** array of integers `numbers` that is already **sorted in non-decreasing order**, find two numbers such that they add up to a specific target number. Let these two numbers be `numbers[index1]` and `numbers[index2]` where  $1 \leq index_1 < index_2 \leq numbers.length$ .

Return *the indices of the two numbers, `index1` and `index2`, added by one as an integer array [`index1`, `index2`] of length 2*.

The tests are generated such that there is **exactly one solution**. You **may not** use the same element twice.

Your solution must use only constant extra space.

#### Example 1:

**Input:** `numbers = [2, 7, 11, 15]`, `target = 9`

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

**Explanation:** The sum of 2 and 7 is 9. Therefore, `index1 = 1`, `index2 = 2`. We return `[1, 2]`.

#### Example 2:

**Input:** `numbers = [2, 3, 4]`, `target = 6`

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

**Explanation:** The sum of 2 and 4 is 6. Therefore `index1 = 1`, `index2 = 3`. We return `[1, 3]`.

#### Example 3:

**Input:** `numbers = [-1, 0]`, `target = -1`

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

**Explanation:** The sum of -1 and 0 is -1. Therefore `index1 = 1`, `index2 = 2`. We return `[1, 2]`.

### Constraints:

- $2 \leq numbers.length \leq 3 * 10^4$
- $-1000 \leq numbers[i] \leq 1000$
- `numbers` is sorted in **non-decreasing order**.
- $-1000 \leq target \leq 1000$
- The tests are generated such that there is **exactly one solution**.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution
{
public:
    vector<int> twoSum(vector<int> &numbers, int target)
    {
        int smallPointer = 0;
        int largePointer = numbers.size() - 1;

        while (smallPointer < largePointer)
        {
            if (numbers[smallPointer] + numbers[largePointer] < target)
                smallPointer++;

            else if (numbers[smallPointer] + numbers[largePointer] > target)
                largePointer--;

            else if (numbers[smallPointer] + numbers[largePointer] == target)
                return {smallPointer + 1, largePointer + 1};
        }

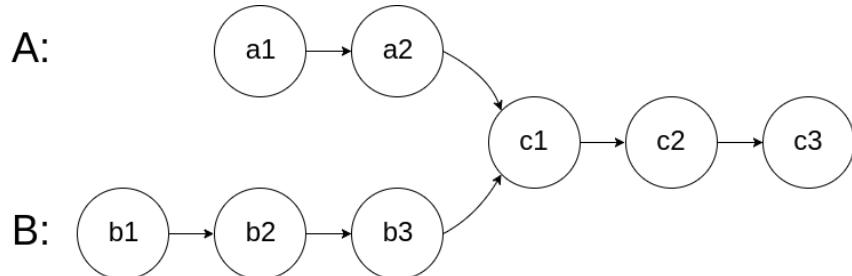
        return {};
    };
}
```

## 160 Intersection of Two Linked Lists (link)

### Description

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

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



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

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

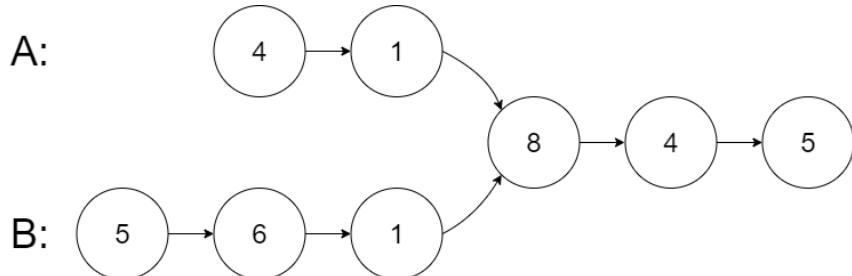
#### Custom Judge:

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

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

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

#### Example 1:



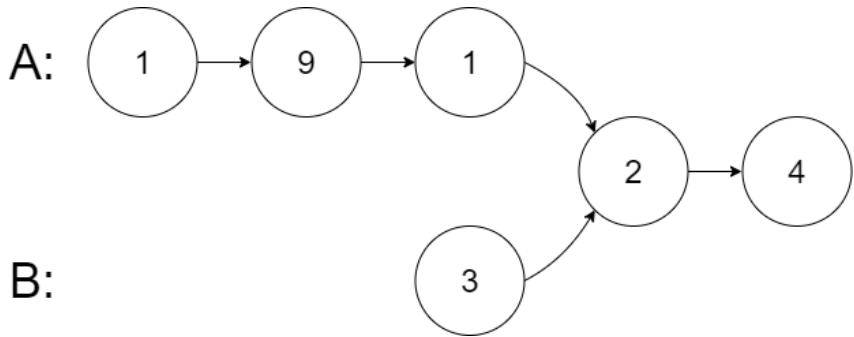
**Input:** `intersectVal = 8, listA = [4,1,8,4,5], listB = [5,6,1,8,4,5], skipA = 2, skipB = 3`

**Output:** Intersected at '8'

**Explanation:** The intersected node's value is 8 (note that this must not be 0 if the two lists intersect). From the head of A, it reads as [4,1,8,4,5]. From the head of B, it reads as [5,6,1,8,4,5]. The nodes in listA are: 4, 1, 8, 4, 5. The nodes in listB are: 5, 6, 1, 8, 4, 5. There are 2 nodes before the intersected node in listA and 3 nodes before the intersected node in listB.

– Note that the intersected node's value is not 1 because the nodes with value 1 in A and B (2nd and 3rd nodes) are different.

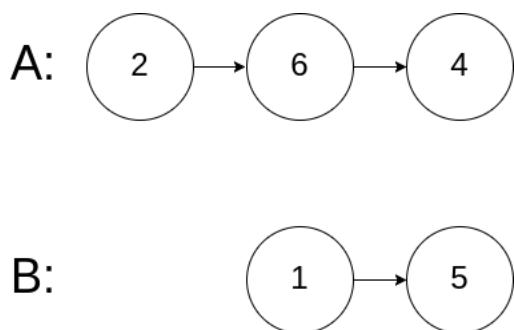
#### Example 2:



**Input:** intersectVal = 2, listA = [1,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1  
**Output:** Intersected at '2'

**Explanation:** The intersected node's value is 2 (note that this must not be 0 if the two lists intersect). From the head of A, it reads as [1,9,1,2,4]. From the head of B, it reads as [3,2,4]. There are 3 nodes ahead of the intersection in A and 1 node ahead of the intersection in B.

### Example 3:



**Input:** intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2  
**Output:** No intersection

**Explanation:** From the head of A, it reads as [2,6,4]. From the head of B, it reads as [1,5]. Since both lists don't have 3 nodes before their intersection, the two lists do not intersect.

### Constraints:

- The number of nodes of listA is in the range [1, 1000].
- The number of nodes of listB is in the range [1, 1000].
- $1 \leq \text{Node.val} \leq 10^5$
- $0 \leq \text{skipA} \leq \text{m}$
- $0 \leq \text{skipB} \leq \text{n}$
- $\text{intersectVal} \neq 0$  if listA and listB do not intersect.
- $\text{intersectVal} == \text{listA}[\text{skipA}] == \text{listB}[\text{skipB}]$  if listA and listB intersect.

**Follow up:** Could you write a solution that runs in  $O(m + n)$  time and use only  $O(1)$  memory?

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
/*
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution
{
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB)
    {
        ListNode *cur1 = headA, *cur2 = headB;
        while (cur1 != cur2)
        {
            cur1 = cur1 ? cur1->next : headB;
            cur2 = cur2 ? cur2->next : headA;
        }
        return cur1;
    }
};
```

## 138 Copy List with Random Pointer (link)

### Description

A linked list of length  $n$  is given such that each node contains an additional random pointer, which could point to any node in the list, or `null`.

Construct a **deep copy** of the list. The deep copy should consist of exactly  $n$  **brand new** nodes, where each new node has its value set to the value of its corresponding original node. Both the `next` and `random` pointer of the new nodes should point to new nodes in the copied list such that the pointers in the original list and copied list represent the same list state. **None of the pointers in the new list should point to nodes in the original list.**

For example, if there are two nodes  $X$  and  $Y$  in the original list, where  $X.\text{random} \rightarrow Y$ , then for the corresponding two nodes  $x$  and  $y$  in the copied list,  $x.\text{random} \rightarrow y$ .

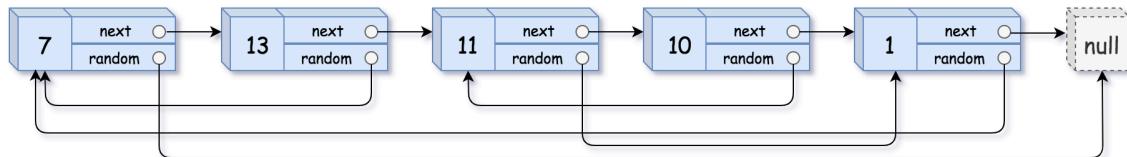
Return *the head of the copied linked list*.

The linked list is represented in the input/output as a list of  $n$  nodes. Each node is represented as a pair of `[val, random_index]` where:

- `val`: an integer representing `Node.val`
- `random_index`: the index of the node (range from 0 to  $n-1$ ) that the `random` pointer points to, or `null` if it does not point to any node.

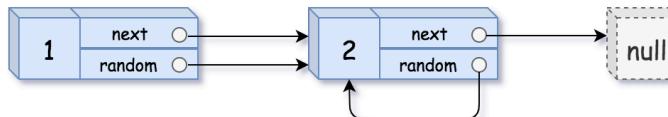
Your code will **only** be given the head of the original linked list.

### Example 1:



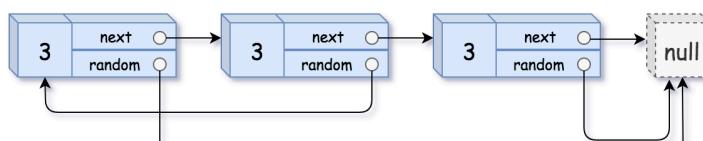
**Input:** head = [[7,null],[13,0],[11,4],[10,2],[1,0]]  
**Output:** [[7,null],[13,0],[11,4],[10,2],[1,0]]

### Example 2:



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

### Example 3:



**Input:** head = [[3,null],[3,0],[3,null]]  
**Output:** [[3,null],[3,0],[3,null]]

**Constraints:**

- $0 \leq n \leq 1000$
- $-10^4 \leq \text{Node.val} \leq 10^4$
- `Node.random` is null or is pointing to some node in the linked list.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
/*
// Definition for a Node.
class Node {
public:
    int val;
    Node* next;
    Node* random;
    Node(int _val) {
        val = _val;
        next = NULL;
        random = NULL;
    }
};

class Solution
{
public:
    // Original Random -> Pointer to which random is required
    unordered_map<Node *, Node *> seen;
    Node *copyRandomList(Node *head)
    {
        if (head == nullptr)
            return nullptr;

        if (!seen[head])
        {
            Node *copy = new Node(head->val);
            // randomOf[head->random] = copy;

            seen[head] = copy;

            copy->next = copyRandomList(head->next);
            copy->random = copyRandomList(head->random);

            // return copy;
        }

        return seen[head];
    }
};
```

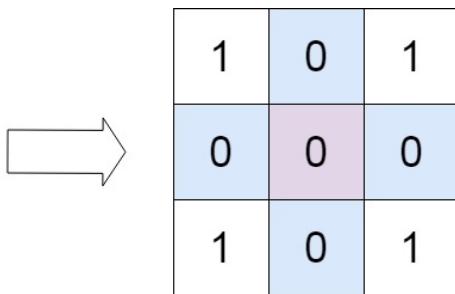
## [73 Set Matrix Zeroes \(link\)](#)

### Description

Given an  $m \times n$  integer matrix `matrix`, if an element is `0`, set its entire row and column to `0`'s.

You must do it [in place](#).

#### Example 1:



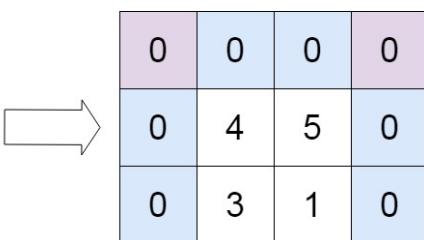
1	1	1
1	0	1
1	1	1

1	0	1
0	0	0
1	0	1

```
Input: matrix = [[1,1,1],[1,0,1],[1,1,1]]
Output: [[1,0,1],[0,0,0],[1,0,1]]
```

#### Example 2:



0	1	2	0
3	4	5	2
1	3	1	5

0	0	0	0
0	4	5	0
0	3	1	0

```
Input: matrix = [[0,1,2,0],[3,4,5,2],[1,3,1,5]]
Output: [[0,0,0,0],[0,4,5,0],[0,3,1,0]]
```

#### Constraints:

- $m == \text{matrix.length}$
- $n == \text{matrix[0].length}$
- $1 \leq m, n \leq 200$
- $-2^{31} \leq \text{matrix}[i][j] \leq 2^{31} - 1$

#### Follow up:

- A straightforward solution using  $O(mn)$  space is probably a bad idea.
- A simple improvement uses  $O(m + n)$  space, but still not the best solution.
- Could you devise a constant space solution?

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution
{
public:
    bool FirstRowContainsZero(vector<vector<int>> &matrix)
    {
        for (int i = 0; i < matrix[0].size(); i++)
            if (matrix[0][i] == 0)
                return true;
        return false;
    }

    bool FirstColContainsZero(vector<vector<int>> &matrix)
    {
        for (int i = 0; i < matrix.size(); i++)
            if (matrix[i][0] == 0)
                return true;
        return false;
    }

    void MakeColoumnZero(vector<vector<int>> &matrix, int coloumn)
    {
        for (int i = 0; i < matrix.size(); i++)
            matrix[i][coloumn] = 0;
    }

    void MakeRowZero(vector<vector<int>> &matrix, int row)
    {
        for (int i = 0; i < matrix[0].size(); i++)
            matrix[row][i] = 0;
    }

    void setZeroes(vector<vector<int>> &matrix)
    {
        bool firstRowContainsZero = FirstRowContainsZero(matrix);
        bool firstColContainsZero = FirstColContainsZero(matrix);

        vector<int> rows;
        vector<int> cols;

        for (int i = 0; i < matrix.size(); i++)
        {
            for (int j = 0; j < matrix[i].size(); j++)
            {
                if (matrix[i][j] == 0)
                {
                    // matrix[i][0] = 0;
                    rows.push_back(i);
                    cols.push_back(j);
                    // matrix[0][j] = 0;
                }
            }
        }

        // for (int i = 0; i < matrix.size(); i++)
        //     if (matrix[i][0] == 0)
        //         MakeRowZero(matrix, i);

        // for (int i = 0; i < matrix[0].size(); i++)
        //     if (matrix[0][i] == 0)
        //         MakeColoumnZero(matrix, i);
        for (auto i: rows)
            MakeRowZero(matrix, i);

        for (auto i : cols)
            MakeColoumnZero(matrix, i);

        if (firstRowContainsZero)
            MakeRowZero(matrix, 0);

        if (firstColContainsZero)
            MakeColoumnZero(matrix, 0);
    }
}
```

{};

## [15 3Sum \(link\)](#)

### Description

Given an integer array `nums`, return all the triplets `[nums[i], nums[j], nums[k]]` such that  $i \neq j$ ,  $i \neq k$ , and  $j \neq k$ , and  $nums[i] + nums[j] + nums[k] == 0$ .

Notice that the solution set must not contain duplicate triplets.

#### Example 1:

**Input:** `nums = [-1,0,1,2,-1,-4]`

**Output:** `[[-1,-1,2], [-1,0,1]]`

**Explanation:**

`nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.`

`nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.`

`nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.`

The distinct triplets are `[-1,0,1]` and `[-1,-1,2]`.

Notice that the order of the output and the order of the triplets does not matter.

#### Example 2:

**Input:** `nums = [0,1,1]`

**Output:** `[]`

**Explanation:** The only possible triplet does not sum up to 0.

#### Example 3:

**Input:** `nums = [0,0,0]`

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

**Explanation:** The only possible triplet sums up to 0.

#### Constraints:

- $3 \leq \text{nums.length} \leq 3000$
- $-10^5 \leq \text{nums}[i] \leq 10^5$

(scroll down for solution)

# Solution

Language: cpp

Status: Accepted

```

class Solution
{
public:
    bool isSameSeq(vector<int> arr)
    {
        int num = arr[0];
        for (auto i : arr)
            if (i != num)
                return false;
        return true;
    }

    bool contains3Zeros(vector<int> a)
    {
        int count = 0;
        for (auto i : a)
        {
            if (i == 0)
                ++count;
            if (count == 3)
                return true;
        }
        return (count >= 3);
    }

    bool onlyZeroOneMinusOneCase(vector<int> a)
    {
        for (auto i : a)
            if (i != 0 || i != 1 || i != -1)
                return false;
        return true;
    }

    vector<vector<int>> threeSum(vector<int> &nums)
    {
        if (nums.size() < 3)
            return {};
        if (isSameSeq(nums))
            if (nums[0] == 0)
                return {{0, 0, 0}};
        if (nums.size() == 3000 && nums[0] == 0)
            return {{-1, 0, 1}, {0, 0, 0}};
        unordered_map<int, bool> seen;
        for (auto i : nums)
            seen[i] = true;
        set<vector<int>> result;
        for (int i = 0; i < nums.size(); i++)
        {
            for (int j = i + 1; j < nums.size(); j++)
            {
                auto req = 0 - (nums[i] + nums[j]);
                if (seen[req] && req != nums[i] && req != nums[j])
                {
                    // auto min1 = ;
                    // auto max1 = max(max(req, nums[i]), nums[j]);
                    // auto third = 0 - min1 + max1;
                    result.insert({min(min(req, nums[i]), nums[j]),
                                  (0 - (max(max(req, nums[i]), nums[j]) + min(min(req, nums[i]),
                                  max(max(req, nums[i]), nums[j]))));
                }
            }
        }
        vector<vector<int>> finalResult;
        for (auto i : result)
            finalResult.push_back(i);
    }
}

```

```
    if (contains3Zeros(nums))
        finalResult.push_back({0, 0, 0});

    return finalResult;
};
```

## [8 String to Integer \(atoi\) \(link\)](#)

### Description

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

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

1. **Whitespace**: Ignore any leading whitespace (" ").
2. **Signedness**: Determine the sign by checking if the next character is '-' or '+', assuming positivity if neither present.
3. **Conversion**: Read the integer by skipping leading zeros until a non-digit character is encountered or the end of the string is reached. If no digits were read, then the result is 0.
4. **Rounding**: If the integer is out of the 32-bit signed integer range  $[-2^{31}, 2^{31} - 1]$ , then round the integer to remain in the range. Specifically, integers less than  $-2^{31}$  should be rounded to  $-2^{31}$ , and integers greater than  $2^{31} - 1$  should be rounded to  $2^{31} - 1$ .

Return the integer as the final result.

#### Example 1:

**Input:** s = "42"

**Output:** 42

**Explanation:**

The underlined characters are what is read in and the caret is the current reader position.  
 Step 1: "42" (no characters read because there is no leading whitespace)  
 ^  
 Step 2: "42" (no characters read because there is neither a '-' nor '+')  
 ^  
 Step 3: "42" ("42" is read in)  
 ^

#### Example 2:

**Input:** s = "-042"

**Output:** -42

**Explanation:**

Step 1: "  -042" (leading whitespace is read and ignored)  
 ^  
 Step 2: "  -042" ('-' is read, so the result should be negative)  
 ^  
 Step 3: "  -042" ("042" is read in, leading zeros ignored in the result)  
 ^

#### Example 3:

**Input:** s = "1337c0d3"

**Output:** 1337

**Explanation:**

Step 1: "1337c0d3" (no characters read because there is no leading whitespace)  
 ^  
 Step 2: "1337c0d3" (no characters read because there is neither a '-' nor '+')  
 ^  
 Step 3: "1337c0d3" ("1337" is read in; reading stops because the next character is a non-digit)  
 ^

**Example 4:****Input:** s = "0-1"**Output:** 0**Explanation:**

```
Step 1: "0-1" (no characters read because there is no leading whitespace)
         ^
Step 2: "0-1" (no characters read because there is neither a '-' nor '+')
         ^
Step 3: "0-1" ("0" is read in; reading stops because the next character is a non-digit)
         ^
```

**Example 5:****Input:** s = "words and 987"**Output:** 0**Explanation:**

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

**Constraints:**

- $0 \leq s.length \leq 200$
- s consists of English letters (lower-case and upper-case), digits (0-9), ' ', '+', '−', and '.'.

(scroll down for solution)

## Solution

Language: cpp

**Status: Accepted**

## [146 LRU Cache \(link\)](#)

### Description

Design a data structure that follows the constraints of a [Least Recently Used \(LRU\) cache](#).

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:

- $1 \leq \text{capacity} \leq 3000$
- $0 \leq \text{key} \leq 10^4$
- $0 \leq \text{value} \leq 10^5$
- At most  $2 * 10^5$  calls will be made to `get` and `put`.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class LRUCache {
public:
    unordered_map<int, pair<list<int>::iterator, int>> ht;
    list<int> dll;
    int cap;
    LRUCache(int capacity) {
        cap=capacity;
    }

    void moveToFront(int key){
        dll.erase(ht[key].first);
        dll.push_front(key);
        ht[key].first=dll.begin();
    }

    int get(int key) {
        if(ht.find(key)==ht.end()) return -1;

        moveToFront(key);
        return ht[key].second;
    }

    void put(int key, int value) {
        if(ht.find(key)!=ht.end()){
            ht[key].second=value;
            moveToFront(key);
        }
        else{
            dll.push_front(key);
            ht[key]={dll.begin(), value};
            cap--;
        }

        if(cap<0){
            ht.erase(dll.back());
            dll.pop_back();
            cap++;
        }
    }
};
```

## [695 Max Area of Island \(link\)](#)

### Description

You are given an  $m \times n$  binary matrix grid. An island is a group of 1's (representing land) connected **4-directionally** (horizontal or vertical.) You may assume all four edges of the grid are surrounded by water.

The **area** of an island is the number of cells with a value 1 in the island.

Return *the maximum area of an island in grid*. If there is no island, return 0.

#### Example 1:

0	0	1	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	0	0	0	0
0	1	1	0	1	0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	0	1	0	1	0	0	0
0	1	0	0	1	1	0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0	0

**Input:** grid = [[0,0,1,0,0,0,0,1,0,0,0,0,0],[0,0,0,0,0,0,0,1,1,1,0,0,0],[0,1,1,0,1,0,0,0,0,0,0,0,0]]  
**Output:** 6  
**Explanation:** The answer is not 11, because the island must be connected 4-directionally.

#### Example 2:

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

#### Constraints:

- $m == \text{grid.length}$
- $n == \text{grid}[i].length$
- $1 \leq m, n \leq 50$
- $\text{grid}[i][j]$  is either 0 or 1.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    long long int calculateArea(vector<vector<int>>& grid, int i, int j) {
        if (i < 0 || i >= grid.size() || j < 0 || j >= grid[i].size() || grid[i][j] == 0) {
            return 0;
        }
        grid[i][j] = 0;
        int left = calculateArea(grid, i - 1, j);
        int right = calculateArea(grid, i + 1, j);
        int up = calculateArea(grid, i, j - 1);
        int down = calculateArea(grid, i, j + 1);
        return up + down + left + right + 1;
    }

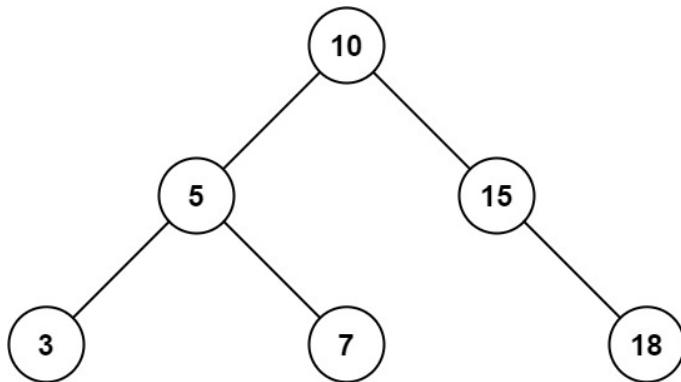
    int maxAreaOfIsland(vector<vector<int>>& grid) {
        long long int maxArea = 0;
        for (int i = 0; i < grid.size(); i++) {
            for (int j = 0; j < grid[i].size(); j++) {
                if (grid[i][j] == 1) {
                    auto current_sum = calculateArea(grid, i, j);
                    if (current_sum > maxArea)
                        maxArea = current_sum;
                }
            }
        }
        return maxArea;
    }
};
```

## [975 Range Sum of BST \(link\)](#)

### Description

Given the root node of a binary search tree and two integers `low` and `high`, return *the sum of values of all nodes with a value in the **inclusive** range*  $[low, high]$ .

#### Example 1:

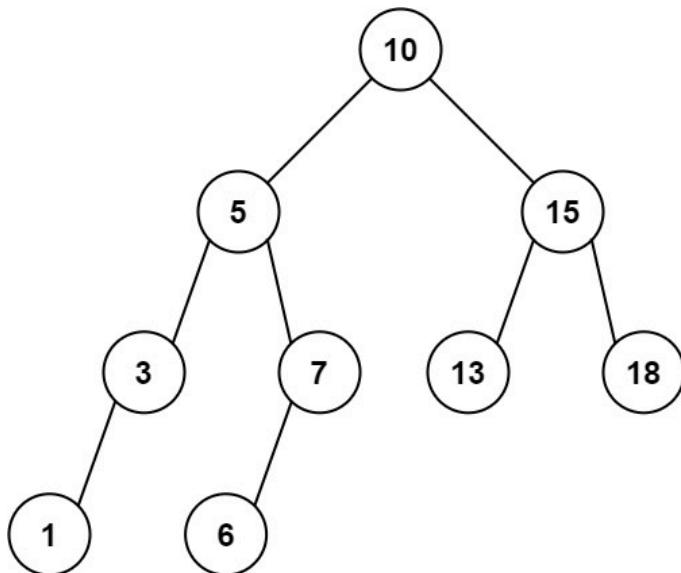


**Input:** root = [10,5,15,3,7,null,18], low = 7, high = 15

**Output:** 32

**Explanation:** Nodes 7, 10, and 15 are in the range [7, 15].  $7 + 10 + 15 = 32$ .

#### Example 2:



**Input:** root = [10,5,15,3,7,13,18,1,null,6], low = 6, high = 10

**Output:** 23

**Explanation:** Nodes 6, 7, and 10 are in the range [6, 10].  $6 + 7 + 10 = 23$ .

#### Constraints:

- The number of nodes in the tree is in the range  $[1, 2 * 10^4]$ .
- $1 \leq \text{Node.val} \leq 10^5$
- $1 \leq \text{low} \leq \text{high} \leq 10^5$
- All `Node.val` are **unique**.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    int rangeSumBST(TreeNode* root, int L, int R) {
        if (root == nullptr) return 0;
        queue<TreeNode*> q;
        q.push(root);

        int sum = 0;

        TreeNode* curr;

        while(!q.empty()) {
            curr = q.front();
            q.pop();

            if (curr->val >= L && curr->val <= R) sum += curr->val;

            if (curr->left) q.push(curr->left);
            if (curr->right) q.push(curr->right);
        }

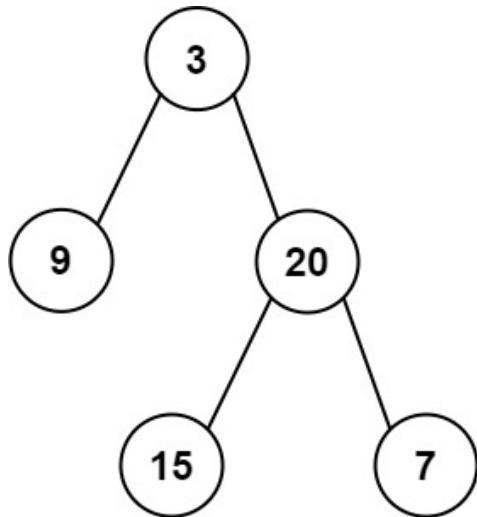
        return sum;
    }
};
```

## [637 Average of Levels in Binary Tree \(link\)](#)

### Description

Given the root of a binary tree, return *the average value of the nodes on each level in the form of an array*. Answers within  $10^{-5}$  of the actual answer will be accepted.

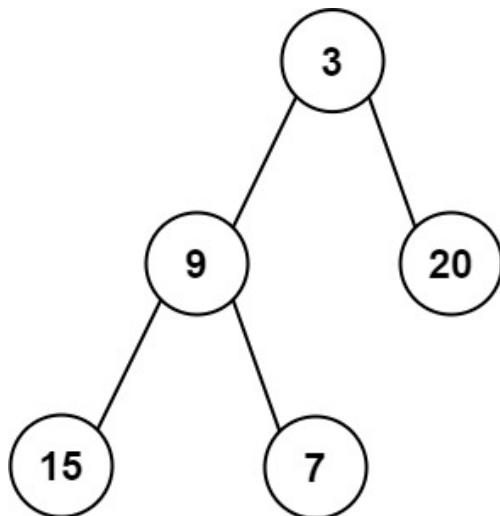
#### Example 1:



**Input:** root = [3,9,20,null,null,15,7]  
**Output:** [3.00000,14.50000,11.00000]

Explanation: The average value of nodes on level 0 is 3, on level 1 is 14.5, and on level 2 is 11. Hence return [3, 14.5, 11].

#### Example 2:



**Input:** root = [3,9,20,15,7]  
**Output:** [3.00000,14.50000,11.00000]

#### Constraints:

- The number of nodes in the tree is in the range  $[1, 10^4]$ .
- $-2^{31} \leq \text{Node.val} \leq 2^{31} - 1$

(scroll down for solution)

# Solution

Language: cpp

Status: Accepted

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:

    int depthCalc(TreeNode* root) {
        if (root == nullptr) return 0;
        return 1 + max(depthCalc(root->left), depthCalc(root->right));
    }

    vector<double> averageOfLevels(TreeNode* root) {
        if (root == nullptr) return {0};

        vector<vector<double>> levelOrder(depthCalc(root));

        queue<TreeNode*> q;
        unordered_map<TreeNode*, int> levels;

        q.push(root);
        levels[root] = 0;

        while(!q.empty()) {
            auto curr = q.front();
            q.pop();

            levelOrder[levels[curr]].push_back(curr->val);

            if (curr->left) {
                levels[curr->left] = 1 + levels[curr];
                q.push(curr->left);
            }

            if (curr->right) {
                levels[curr->right] = 1 + levels[curr];
                q.push(curr->right);
            }
        }

        vector<double> result;
        for(auto i: levelOrder) {
            long long int len = i.size();
            long long int sum = 0;
            for(auto j: i) {
                sum += j;
            }

            result.push_back((double)sum / len);
        }
        return result;
    }
};
```



## [122 Best Time to Buy and Sell Stock II \(link\)](#)

### Description

You are given an integer array `prices` where `prices[i]` is the price of a given stock on the  $i^{\text{th}}$  day.

On each day, you may decide to buy and/or sell the stock. You can only hold **at most one** share of the stock at any time. However, you can sell and buy the stock multiple times on the **same day**, ensuring you never hold more than one share of the stock.

Find and return *the maximum profit you can achieve*.

#### Example 1:

**Input:** `prices = [7,1,5,3,6,4]`

**Output:** 7

**Explanation:** Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4. Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3. Total profit is 4 + 3 = 7.

#### Example 2:

**Input:** `prices = [1,2,3,4,5]`

**Output:** 4

**Explanation:** Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4. Total profit is 4.

#### Example 3:

**Input:** `prices = [7,6,4,3,1]`

**Output:** 0

**Explanation:** There is no way to make a positive profit, so we never buy the stock to achieve th

#### Constraints:

- $1 \leq \text{prices.length} \leq 3 * 10^4$
- $0 \leq \text{prices}[i] \leq 10^4$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        if (prices.size() == 0) return 0;

        int sum = 0, prev = 0;
        for(int i = 0; i < prices.size(); i++) {
            if (i == prices.size() - 1) {
                sum += prices[i] - prices[prev];
            }

            else if (prices[i] > prices[i + 1]) {
                sum += prices[i] - prices[prev];
                prev = i + 1;
            }
        }

        return sum;
    }
};
```

## [387 First Unique Character in a String \(link\)](#)

### Description

Given a string  $s$ , find the **first** non-repeating character in it and return its index. If it **does not** exist, return  $-1$ .

#### Example 1:

**Input:**  $s = \text{"leetcode"}$

**Output:** 0

#### Explanation:

The character '`l`' at index 0 is the first character that does not occur at any other index.

#### Example 2:

**Input:**  $s = \text{"loveleetcode"}$

**Output:** 2

#### Example 3:

**Input:**  $s = \text{"aabb"}$

**Output:** -1

#### Constraints:

- $1 \leq s.\text{length} \leq 10^5$
- $s$  consists of only lowercase English letters.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    int firstUniqChar(string s) {
        unordered_map<char, int> freq;
        for(int i = 0; i < s.length(); i++)
            freq[s[i]]++;

        for(int i = 0; i < s.length(); i++)
            if (freq[s[i]] == 1)
                return i;
        return -1;
    }
};
```

## [344 Reverse String \(link\)](#)

### Description

Write a function that reverses a string. The input string is given as an array of characters  $s$ .

You must do this by modifying the input array [in-place](#) with  $O(1)$  extra memory.

#### Example 1:

```
Input: s = ["h", "e", "l", "l", "o"]
Output: ["o", "l", "l", "e", "h"]
```

#### Example 2:

```
Input: s = ["H", "a", "n", "n", "a", "h"]
Output: ["h", "a", "n", "n", "a", "H"]
```

#### Constraints:

- $1 \leq s.length \leq 10^5$
- $s[i]$  is a [printable ascii character](#).

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    void swap(char &s1, char &s2) {
        char temp = s1;
        s1 = s2;
        s2 = temp;
    }

    void reverseString(vector<char>& s) {
        int length = s.size();
        for(int i = 0; i < length / 2; i++) {
            swap(s[i], s[length-i - 1]);
        }
    }
};
```

## [283 Move Zeroes \(link\)](#)

### Description

Given an integer array `nums`, move all 0's to the end of it while maintaining the relative order of the non-zero elements.

**Note** that you must do this in-place without making a copy of the array.

#### Example 1:

```
Input: nums = [0,1,0,3,12]
Output: [1,3,12,0,0]
```

#### Example 2:

```
Input: nums = [0]
Output: [0]
```

#### Constraints:

- $1 \leq \text{nums.length} \leq 10^4$
- $-2^{31} \leq \text{nums}[i] \leq 2^{31} - 1$

**Follow up:** Could you minimize the total number of operations done?

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    void moveZeroes(vector<int>& nums) {
        int num0fZeros = 0;
        for(auto i: nums) if(i == 0) num0fZeros++;

        vector<int> res;
        for(auto i: nums) {
            if(i != 0) res.push_back(i);
        }

        for (int i = 0; i < num0fZeros; i++) {
            res.push_back(0);
        }

        for(int i = 0; i < nums.size(); i++) {
            nums[i] = res[i];
        }

    }
};
```

## [66 Plus One \(link\)](#)

### Description

You are given a **large integer** represented as an integer array `digits`, where each `digits[i]` is the  $i^{\text{th}}$  digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.

Increment the large integer by one and return *the resulting array of digits*.

#### Example 1:

```
Input: digits = [1,2,3]
Output: [1,2,4]
Explanation: The array represents the integer 123.
Incrementing by one gives 123 + 1 = 124.
Thus, the result should be [1,2,4].
```

#### Example 2:

```
Input: digits = [4,3,2,1]
Output: [4,3,2,2]
Explanation: The array represents the integer 4321.
Incrementing by one gives 4321 + 1 = 4322.
Thus, the result should be [4,3,2,2].
```

#### Example 3:

```
Input: digits = [9]
Output: [1,0]
Explanation: The array represents the integer 9.
Incrementing by one gives 9 + 1 = 10.
Thus, the result should be [1,0].
```

### Constraints:

- $1 \leq \text{digits.length} \leq 100$
- $0 \leq \text{digits}[i] \leq 9$
- `digits` does not contain any leading 0's.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:

    bool ifAllNine(vector<int> arr) {
        for(auto i: arr) if(i != 9) return false;
        return true;
    }

    vector<int> plusOne(vector<int>& digits) {
        if (ifAllNine(digits)) {
            vector<int> d(digits.size() + 1);
            d[0] = 1;
            for(int i = 1; i < d.size(); i++) {
                d[i] = 0;
            }
            return d;
        }
        for(int i = digits.size() - 1; i >= 0; i--) {
            if (digits[i] == 9) {
                digits[i] = 0;
            }
            else {
                digits[i]++;
                return digits;
            }
        }
        return digits;
    }
};
```

## 350 Intersection of Two Arrays II (link)

### Description

Given two integer arrays `nums1` and `nums2`, return *an array of their intersection*. Each element in the result must appear as many times as it shows in both arrays and you may return the result in **any order**.

#### Example 1:

```
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2,2]
```

#### Example 2:

```
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [4,9]
Explanation: [9,4] is also accepted.
```

#### Constraints:

- $1 \leq \text{nums1.length}, \text{nums2.length} \leq 1000$
- $0 \leq \text{nums1[i]}, \text{nums2[i]} \leq 1000$

#### Follow up:

- What if the given array is already sorted? How would you optimize your algorithm?
- What if `nums1`'s size is small compared to `nums2`'s size? Which algorithm is better?
- What if elements of `nums2` are stored on disk, and the memory is limited such that you cannot load all elements into the memory at once?

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    vector<int> intersect(vector<int>& nums1, vector<int>& nums2) {
        map<int, int> seen;
        for(auto i: nums1) seen[i]++;
        
        vector<int> result;
        for(int i = 0; i < nums2.size(); i++) {
            if(seen[nums2[i]]) {
                result.push_back(nums2[i]);
                seen[nums2[i]] -= 1;
            }
        }
        return result;
    }
};
```

## [136 Single Number \(link\)](#)

### Description

Given a **non-empty** array of integers `nums`, every element appears *twice* except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space.

#### Example 1:

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

**Output:** 1

#### Example 2:

**Input:** `nums = [4,1,2,1,2]`

**Output:** 4

#### Example 3:

**Input:** `nums = [1]`

**Output:** 1

#### Constraints:

- $1 \leq \text{nums.length} \leq 3 * 10^4$
- $-3 * 10^4 \leq \text{nums}[i] \leq 3 * 10^4$
- Each element in the array appears twice except for one element which appears only once.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        unordered_map<int, int> freq;
        int len = nums.size();
        for(int i = 0; i < len; i++) {
            freq[nums[i]]++;
        }

        for(auto i: freq) {
            if (i.second != 2) {
                return i.first;
            }
        }
        return -1;
    }
};
```

## [300 Longest Increasing Subsequence \(link\)](#)

### 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:

- $1 \leq \text{nums.length} \leq 2500$
- $-10^4 \leq \text{nums}[i] \leq 10^4$

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

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    /*
    [1,2,4,2,1,3,4,2,1,5,6,4,2,4,5,7,4]
    [1] -> 1
    [1, 1] -> 1
    [1, 2] -> 2
    [1, 2, 3] -> 3
    [1, 4, 2, 3] -> 3

    if(arr[i-1] < arr[i]) dp[i] += dp[i-1];
    */

    int lengthOfLIS(vector<int>& nums) {
        if(nums.size() == 0) return 0;
        vector<int> dp(nums.size());
        dp[0] = 1;
        int maxans = 1;
        for(int i = 1; i < dp.size(); i++) {
            int maxval = 0;
            for(int j = 0; j < i; j++) {
                if(nums[i] > nums[j]) {
                    maxval = max(maxval, dp[j]);
                }
            }
            dp[i] = maxval + 1;
            maxans = max(maxans, dp[i]);
        }
        return maxans;
    }
};
```

## [198 House Robber \(link\)](#)

### Description

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and **it will automatically contact the police if two adjacent houses were broken into on the same night.**

Given an integer array `nums` representing the amount of money of each house, return *the maximum amount of money you can rob tonight **without alerting the police**.*

#### Example 1:

```
Input: nums = [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
Total amount you can rob = 1 + 3 = 4.
```

#### Example 2:

```
Input: nums = [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).
Total amount you can rob = 2 + 9 + 1 = 12.
```

#### Constraints:

- $1 \leq \text{nums.length} \leq 100$
- $0 \leq \text{nums}[i] \leq 400$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:

    // try 1: using recursion
    int maxmRobbery(vector<int> arr, int i) {
        if(i == 0) return arr[0];
        if(i == 1) return max(arr[0], arr[1]);

        return max(maxmRobbery(arr, i-1), maxmRobbery(arr, i-2) + arr[i]);
    }

    // try2: memoize

    // unordered_map<int, int> seen;

    // int maxmRobbery(vector<int> arr, int i) {
    //     if(i == 0) return arr[0];
    //     if(i == 1) return max(arr[0], arr[1]);

    //     if(seen[i]) return seen[i];
    //     else seen[i] = max(maxmRobbery(arr, i-1), maxmRobbery(arr, i-2) + arr[i]);

    //     return seen[i];
    // }

    // try 3: bottom up

    int rob(vector<int>& nums) {
        if(nums.size() == 0) return 0;
        vector<int> dp(nums.size() + 1);
        dp[0] = nums[0];

        if(nums.size() >= 2)
            dp[1] = max(nums[0], nums[1]);

        for(int i = 2; i < nums.size(); i++) {
            dp[i] = max(dp[i-1], dp[i-2] + nums[i]);
        }
        return dp[nums.size() - 1];
    }
};
```

## [392 Is Subsequence \(link\)](#)

### Description

Given two strings  $s$  and  $t$ , return `true` if  $s$  is a **subsequence** of  $t$ , or `false` otherwise.

A **subsequence** of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e., "ace" is a subsequence of "abcde" while "aec" is not).

#### Example 1:

```
Input: s = "abc", t = "ahbgdc"
Output: true
```

#### Example 2:

```
Input: s = "axc", t = "ahbgdc"
Output: false
```

#### Constraints:

- $0 \leq s.length \leq 100$
- $0 \leq t.length \leq 10^4$
- $s$  and  $t$  consist only of lowercase English letters.

**Follow up:** Suppose there are lots of incoming  $s$ , say  $s_1, s_2, \dots, s_k$  where  $k \geq 10^9$ , and you want to check one by one to see if  $t$  has its subsequence. In this scenario, how would you change your code?

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    bool isSubsequence(string s, string t) {
        int dp[t.length() + 1][s.length() + 1];

        for(int i = 0; i < t.length() + 1; i++) {
            dp[i][0] = 0;
        }

        for(int i = 0; i < s.length() + 1; i++) {
            dp[0][i] = 0;
        }

        for(int i = 1; i < t.length() + 1; i++) {
            for(int j = 1; j < s.length() + 1; j++) {
                if(s[j - 1] == t[i - 1]) {
                    dp[i][j] = 1 + dp[i-1][j-1];
                }
                else {
                    dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
                }
            }
        }

        return (dp[t.length()][s.length()] == s.length());
    }
};
```

## [1086 Divisor Game \(link\)](#)

### Description

Alice and Bob take turns playing a game, with Alice starting first.

Initially, there is a number  $n$  on the chalkboard. On each player's turn, that player makes a move consisting of:

- Choosing any integer  $x$  with  $0 < x < n$  and  $n \% x == 0$ .
- Replacing the number  $n$  on the chalkboard with  $n - x$ .

Also, if a player cannot make a move, they lose the game.

Return `true` if and only if Alice wins the game, assuming both players play optimally.

#### Example 1:

```
Input: n = 2
Output: true
Explanation: Alice chooses 1, and Bob has no more moves.
```

#### Example 2:

```
Input: n = 3
Output: false
Explanation: Alice chooses 1, Bob chooses 1, and Alice has no more moves.
```

#### Constraints:

- $1 \leq n \leq 1000$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    int flag = 0;
    bool divisorGame(int N) {
        if (N == 1 && flag == 0) return false;
        if (N == 1 && flag == 1) return true;

        if (N % 2 == 0) {
            flag = !flag;
            return divisorGame(N - 1);
        }
        else {
            auto d = 0;
            for(int i = 1; i * i < N; i++) {
                if(N % i == 0)
                    d = i;
            }
            flag = !flag;
            return divisorGame(N - d);
        }
    }
};
```

## [454 4Sum II \(link\)](#)

### Description

Given four integer arrays `nums1`, `nums2`, `nums3`, and `nums4` all of length `n`, return the number of tuples  $(i, j, k, l)$  such that:

- $0 \leq i, j, k, l < n$
- $nums1[i] + nums2[j] + nums3[k] + nums4[l] == 0$

#### Example 1:

**Input:** `nums1 = [1,2]`, `nums2 = [-2,-1]`, `nums3 = [-1,2]`, `nums4 = [0,2]`

**Output:** 2

**Explanation:**

The two tuples are:

1.  $(0, 0, 0, 1) \rightarrow nums1[0] + nums2[0] + nums3[0] + nums4[1] = 1 + (-2) + (-1) + 2 = 0$
2.  $(1, 1, 0, 0) \rightarrow nums1[1] + nums2[1] + nums3[0] + nums4[0] = 2 + (-2) + (-1) + 0 = 0$

#### Example 2:

**Input:** `nums1 = [0]`, `nums2 = [0]`, `nums3 = [0]`, `nums4 = [0]`

**Output:** 1

### Constraints:

- $n == nums1.length$
- $n == nums2.length$
- $n == nums3.length$
- $n == nums4.length$
- $1 \leq n \leq 200$
- $-2^{28} \leq nums1[i], nums2[i], nums3[i], nums4[i] \leq 2^{28}$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    /*
        [1, -2, 3, -4, 5, -6, 7]
        [-9, 8, -7, 6, -5, 4, -3]
    */

    int fourSumCount(vector<int>& A, vector<int>& B, vector<int>& C, vector<int>& D) {
        unordered_map<int, int> v1;
        unordered_map<int, int> v2;

        for(int i = 0; i < A.size(); i++) {
            for(int j = 0; j < B.size(); j++) {
                v1[A[i] + B[j]]++;
            }
        }

        int count = 0;

        for(int i = 0; i < A.size(); i++) {
            for(int j = 0; j < B.size(); j++) {
                int sum = C[i] + D[j];
                if(v1[-sum]) {
                    count += v1[-sum];
                }
            }
        }

        return count;
    };
}
```

## [162 Find Peak Element \(link\)](#)

### Description

A peak element is an element that is strictly greater than its neighbors.

Given a **0-indexed** integer array `nums`, find a peak element, and return its index. If the array contains multiple peaks, return the index to **any of the peaks**.

You may imagine that `nums[-1] = nums[n] = -∞`. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

You must write an algorithm that runs in  $O(\log n)$  time.

#### Example 1:

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

**Explanation:** 3 is a peak element and your function should return the index number 2.

#### Example 2:

**Input:** `nums = [1,2,1,3,5,6,4]`  
**Output:** 5

**Explanation:** Your function can return either index number 1 where the peak element is 2, or index

#### Constraints:

- $1 \leq \text{nums.length} \leq 1000$
- $-2^{31} \leq \text{nums}[i] \leq 2^{31} - 1$
- $\text{nums}[i] \neq \text{nums}[i + 1]$  for all valid  $i$ .

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    bool isIncreasing(vector<int> arr) {
        for(int i = 0; i < arr.size() - 1; i++) {
            if(arr[i] > arr[i+1]) return false;
        }
        return true;
    }

    int findPeakElement(vector<int>& nums) {
        int i;
        for(i = 0; i < nums.size() - 1; i++) {
            if(nums[i] > nums[i+1]) {
                return i;
            }
        }
        if(i == nums.size() - 1) return i;
        return 0;
    }
};
```

## [240 Search a 2D Matrix II \(link\)](#)

### Description

Write an efficient algorithm that searches for a value `target` in an  $m \times n$  integer matrix `matrix`. This matrix has the following properties:

- Integers in each row are sorted in ascending from left to right.
- Integers in each column are sorted in ascending from top to bottom.

#### Example 1:

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

```
Input: matrix = [[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]],  
Output: true
```

#### Example 2:

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

```
Input: matrix = [[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]],  
Output: false
```

#### Constraints:

- $m == \text{matrix.length}$
- $n == \text{matrix[i].length}$
- $1 \leq n, m \leq 300$
- $-10^9 \leq \text{matrix[i][j]} \leq 10^9$

- All the integers in each row are **sorted** in ascending order.
- All the integers in each column are **sorted** in ascending order.
- $-10^9 \leq \text{target} \leq 10^9$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:

    bool search(vector<vector<int>>& matrix, long int i, long int j, int target) {
        if(i < 0 || j < 0 || i >= matrix.size() || j >= matrix[0].size()) {
            return false;
        }

        if(target == matrix[i][j]) {
            return true;
        }
        else if (target < matrix[i][j]) {
            return search(matrix, i, j-1, target);
        }
        else {
            return search(matrix, i+1, j, target);
        }
    }

    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        if(matrix.size() == 0) {
            return {};
        }
        long int i = 0;
        long int j = matrix[i].size() - 1;
        return search(matrix, i, j, target);
    }
};
```

## [287 Find the Duplicate Number \(link\)](#)

### Description

Given an array of integers `nums` containing  $n + 1$  integers where each integer is in the range  $[1, n]$  inclusive.

There is only **one repeated number** in `nums`, return *this repeated number*.

You must solve the problem **without** modifying the array `nums` and using only constant extra space.

#### Example 1:

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

#### Example 2:

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

#### Example 3:

```
Input: nums = [3,3,3,3,3]
Output: 3
```

### Constraints:

- $1 \leq n \leq 10^5$
- `nums.length == n + 1`
- $1 \leq \text{nums}[i] \leq n$
- All the integers in `nums` appear only **once** except for **precisely one integer** which appears **two or more** times.

### Follow up:

- How can we prove that at least one duplicate number must exist in `nums`?
- Can you solve the problem in linear runtime complexity?

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    int findDuplicate(vector<int>& nums) {
        unordered_map<int, int> seen;
        for(int i = 0; i < nums.size(); i++) {
            auto idx = nums[i];
            if(seen[idx]) return idx;
            else {
                seen[idx]++;
            }
        }
        return -1;
    }
};
```

## [671 Second Minimum Node In a Binary Tree \(link\)](#)

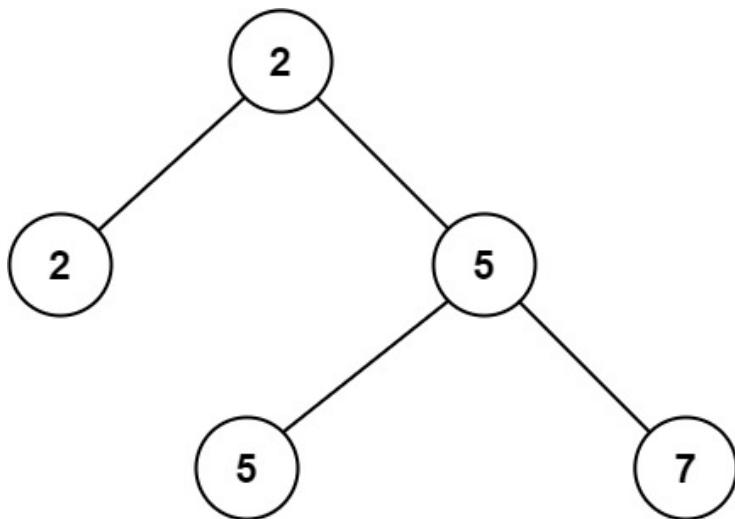
### Description

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:

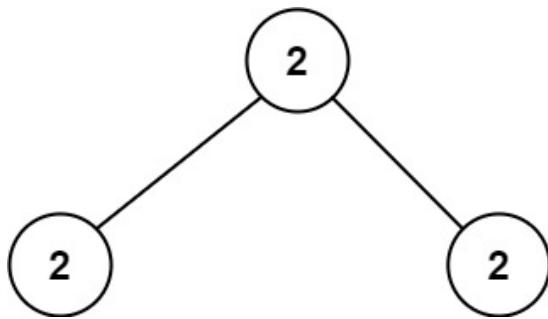


**Input:** root = [2,2,5,null,null,5,7]

**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.

#### Constraints:

- The number of nodes in the tree is in the range [1, 25].
- $1 \leq \text{Node.val} \leq 2^{31} - 1$
- `root.val == min(root.left.val, root.right.val)` for each internal node of the tree.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:

    vector<int> res;
    unordered_map<int, int> seen;
    void inra(TreeNode* root) {

        if(root == nullptr) {
            return;
        }

        inra(root->left);

        if(!seen[root->val]) {
            res.push_back(root->val);
            seen[root->val] = 1;
        }

        inra(root->right);
    }

    int findSecondMinimumValue(TreeNode* root) {

        inra(root);

        sort(res.begin(), res.end());

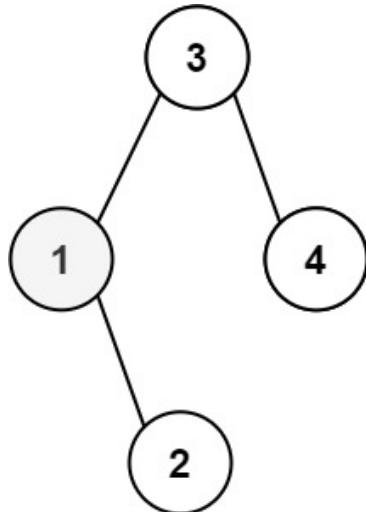
        if(res.size() >= 2) return res[1];
        else return -1;
    }
};
```

## 230 Kth Smallest Element in a BST (link)

### Description

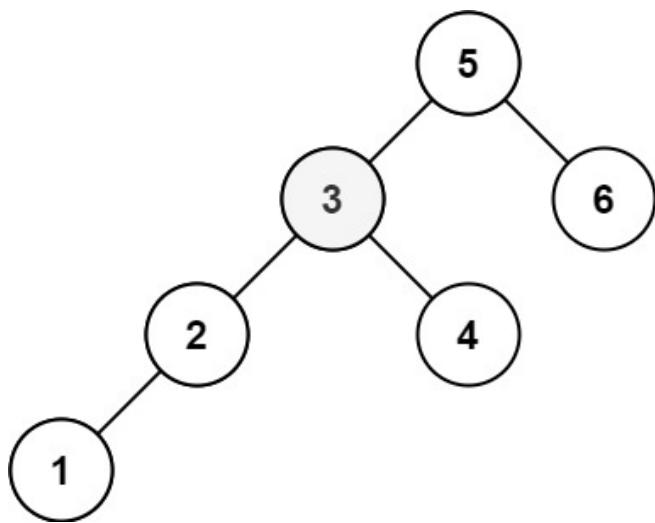
Given the root of a binary search tree, and an integer k, return *the k<sup>th</sup> smallest value (1-indexed) of all the values of the nodes in the tree*.

#### Example 1:



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

#### Example 2:



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

#### Constraints:

- The number of nodes in the tree is n.
- $1 \leq k \leq n \leq 10^4$
- $0 \leq \text{Node.val} \leq 10^4$

**Follow up:** If the BST is modified often (i.e., we can do insert and delete operations) and you need to find the  $k$ th smallest frequently, how would you optimize?

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:

    vector<int> inorder;
    void inorderTraversal(TreeNode* root) {
        if(root == nullptr) return;
        inorderTraversal(root->left);
        inorder.push_back(root->val);
        inorderTraversal(root->right);
    }

    int kthSmallest(TreeNode* root, int k) {
        inorderTraversal(root);
        int i;
        for(i = 0; i < k; i++);
        return inorder[i - 1];
    }
};
```

## [53 Maximum Subarray \(link\)](#)

### Description

Given an integer array `nums`, find the subarray with the largest sum, and return *its sum*.

#### Example 1:

```
Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: The subarray [4,-1,2,1] has the largest sum 6.
```

#### Example 2:

```
Input: nums = [1]
Output: 1
Explanation: The subarray [1] has the largest sum 1.
```

#### Example 3:

```
Input: nums = [5,4,-1,7,8]
Output: 23
Explanation: The subarray [5,4,-1,7,8] has the largest sum 23.
```

### Constraints:

- $1 \leq \text{nums.length} \leq 10^5$
- $-10^4 \leq \text{nums}[i] \leq 10^4$

**Follow up:** If you have figured out the  $O(n)$  solution, try coding another solution using the **divide and conquer** approach, which is more subtle.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    int max(int a, int b) {
        return a >= b ? a : b;
    }
    int maxSubArray(vector<int>& nums) {
        int maxCurrent = nums[0];
        int maxGlobal = nums[0];

        for (int i = 1; i < nums.size(); i++) {
            maxCurrent = max(nums[i], maxCurrent + nums[i]);
            if (maxCurrent > maxGlobal) {
                maxGlobal = maxCurrent;
            }
        }

        return maxGlobal;
    }
};
```

## [202 Happy Number \(link\)](#)

### Description

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:
12 + 92 = 82
82 + 22 = 68
62 + 82 = 100
12 + 02 + 02 = 1
```

#### Example 2:

```
Input: n = 2
Output: false
```

#### Constraints:

- $1 \leq n \leq 2^{31} - 1$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```

class Solution {
public:
    /*
        19 -> 1 + 81 = 82
        82 -> 64 + 4 = 68
        86 -> 64 + 36 = 100

        37 -> 9 + 49 = 58
        58 -> 25 + 64 = 89
        89 -> 64 + 81 = 145
        145 -> 1 + 16 + 25 = 42
        42 = 16 + 4 = 20
        20 = 4;
        4 -> 16;
        16 -> 1 + 36 = 37
        37 (seen before already) hence unhappy
    */

    unordered_map<int, bool> seen;

    //    bool isHappy(int n) {
    //        if (seen[n]) return false;

    //        int sum = 0;
    //        int x = n;
    //        // 123
    //        while(x > 0) {
    //            int rem = x % 10;
    //            rem = rem * rem;
    //            x = x /10;
    //            sum += rem;
    //        }

    //        if (sum == 1) return true;
    //        else return isHappy(sum);
    //    }

    int value(int n) {
        int sum = 0;
        int x = n;
        // 123
        while(x > 0) {
            int rem = x % 10;
            rem = rem * rem;
            x = x /10;
            sum += rem;
        }
        return sum;
    }

    bool isHappy(int n) {
        int temp=n;
        while(1)
        {
            if(temp==89)
                return false;
            if(temp==1)
                return true;
            temp=value(temp);
        }
    }
};

```

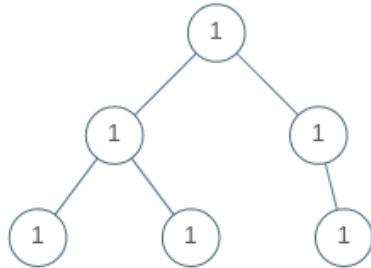
## [1005 Univalued Binary Tree \(link\)](#)

### Description

A binary tree is **uni-valued** if every node in the tree has the same value.

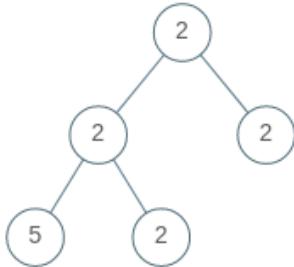
Given the root of a binary tree, return `true` if the given tree is **uni-valued**, or `false` otherwise.

#### Example 1:



```
Input: root = [1,1,1,1,1,null,1]
Output: true
```

#### Example 2:



```
Input: root = [2,2,2,5,2]
Output: false
```

#### Constraints:

- The number of nodes in the tree is in the range  $[1, 100]$ .
- $0 \leq \text{Node.val} < 100$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    bool t(TreeNode* root, int data) {
        if (root != nullptr && root->val != data) return false;
        if (root == nullptr) return true;
        auto t1 = t(root->left, data);
        auto t2 = t(root->right, data);
        return t1 & t2;
    }
    bool isUnivalTree(TreeNode* root) {
        auto data = root->val;
        if (t(root, data)) {
            return true;
        }
        return false;
    }
};
```

## [69 Sqrt\(x\) \(link\)](#)

### Description

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:

- $0 \leq x \leq 2^{31} - 1$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:

    int mySqrt(int x) {
        long long int i;
        for(i = 0; i*i <= x; i++);
        return i-1;
    }
};
```

## [171 Excel Sheet Column Number \(link\)](#)

### Description

Given a string `columnTitle` that represents the column title as appears in an Excel sheet, return *its corresponding column number*.

For example:

```
A -> 1
B -> 2
C -> 3
...
Z -> 26
AA -> 27
AB -> 28
...
```

#### Example 1:

```
Input: columnTitle = "A"
Output: 1
```

#### Example 2:

```
Input: columnTitle = "AB"
Output: 28
```

#### Example 3:

```
Input: columnTitle = "ZY"
Output: 701
```

#### Constraints:

- $1 \leq \text{columnTitle.length} \leq 7$
- `columnTitle` consists only of uppercase English letters.
- `columnTitle` is in the range `["A", "FXSHRXW"]`.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    /*
    AA = 1 * 26^1 + 1 * 26^0 = 26 + 1 = 27;
    A = 1 * 26^0 = 1
    AB = 1 * 26 + 2 * 26^0

    [A, A, A, A]
    len = 4, i => (0, 3)

    val = 1,
    */

    int power(int base, int pow) {
        int p = base;
        for (int i = 0; i < pow; i++) {
            p *= 26;
        }
        return p;
    }

    int val(char ch) {
        return ch - 'A' + 1;
    }

    int titleToNumber(string s) {
        int len = s.length(), sum = 0;
        for (int i = 0; i < len; i++) {
            // auto v = val(s[i]);
            // v = power(v, len - i - 1);
            // sum += v;
            sum += power(val(s[i]), len - i - 1);
        }
        return sum;
    }
};
```

## [268 Missing Number \(link\)](#)

### Description

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:

- $n == \text{nums.length}$
- $1 \leq n \leq 10^4$
- $0 \leq \text{nums}[i] \leq n$
- All the numbers of `nums` are **unique**.

**Follow up:** Could you implement a solution using only  $O(1)$  extra space complexity and  $O(n)$  runtime complexity?

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    int missingNumber(vector<int>& nums) {
        int sum = nums.size()*(nums.size()+1)/2;
        int calsum = 0;
        for(auto i: nums)
            calsum += i;

        return sum - calsum;
    }
};
```

## [169 Majority Element \(link\)](#)

### Description

Given an array `nums` of size  $n$ , return *the majority element*.

The majority element is the element that appears more than  $\lfloor n / 2 \rfloor$  times. You may assume that the majority element always exists in the array.

#### Example 1:

```
Input: nums = [3,2,3]
Output: 3
```

#### Example 2:

```
Input: nums = [2,2,1,1,1,2,2]
Output: 2
```

#### Constraints:

- $n == \text{nums.length}$
- $1 \leq n \leq 5 * 10^4$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- The input is generated such that a majority element will exist in the array.

**Follow-up:** Could you solve the problem in linear time and in  $O(1)$  space?

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        int len = nums.size();
        int majSize = len / 2;

        unordered_map<int, int> freq;
        for(int i = 0; i < len; i++)
            freq[nums[i]]++;

        for(auto i: freq)
        {
            if(i.second > majSize)
                return i.first;
        }

        return -1;
    };
}
```

## [412 Fizz Buzz \(link\)](#)

### Description

Given an integer  $n$ , return a *string array*  $\text{answer}$  (**1-indexed**) where:

- $\text{answer}[i] == \text{"FizzBuzz"}$  if  $i$  is divisible by 3 and 5.
- $\text{answer}[i] == \text{"Fizz"}$  if  $i$  is divisible by 3.
- $\text{answer}[i] == \text{"Buzz"}$  if  $i$  is divisible by 5.
- $\text{answer}[i] == i$  (as a string) if none of the above conditions are true.

#### Example 1:

```
Input: n = 3
Output: ["1","2","Fizz"]
```

#### Example 2:

```
Input: n = 5
Output: ["1","2","Fizz","4","Buzz"]
```

#### Example 3:

```
Input: n = 15
Output: ["1","2","Fizz","4","Buzz","Fizz","7","8","Fizz","Buzz","11","Fizz","13","14","FizzBuzz"]
```

### Constraints:

- $1 \leq n \leq 10^4$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    vector<string> fizzBuzz(int n) {
        // To convert to char -> n + '0';

        vector<string> op(n);

        for(int i = 1; i <= n; i++) {
            if(i % 3 != 0 && i % 5 != 0) {
                op[i-1] = to_string(i);
            }
            else if (i % 3 == 0 && i % 5 != 0){
                op[i-1] = "Fizz";
            }
            else if (i % 5 == 0 && i % 3 != 0){
                op[i-1] = "Buzz";
            }
            else {
                op[i-1] = "FizzBuzz";
            }
        }

        return op;
    }
};
```

## 237 Delete Node in a Linked List (link)

### Description

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

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

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

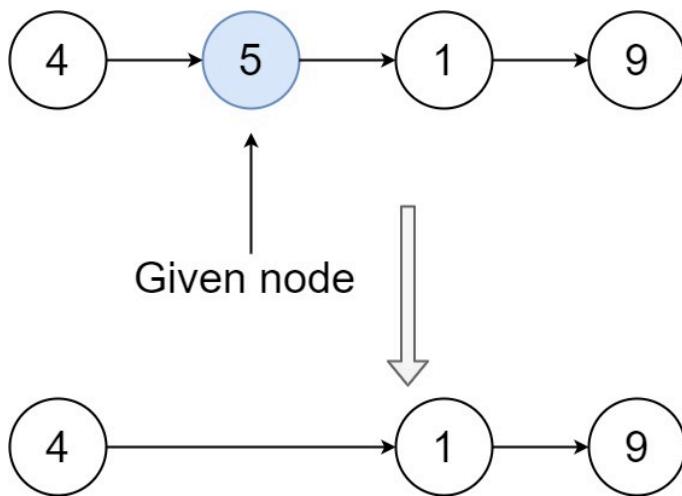
Delete the given node. Note that by deleting the node, we do not mean removing it from memory. We mean:

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

#### Custom testing:

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

#### Example 1:

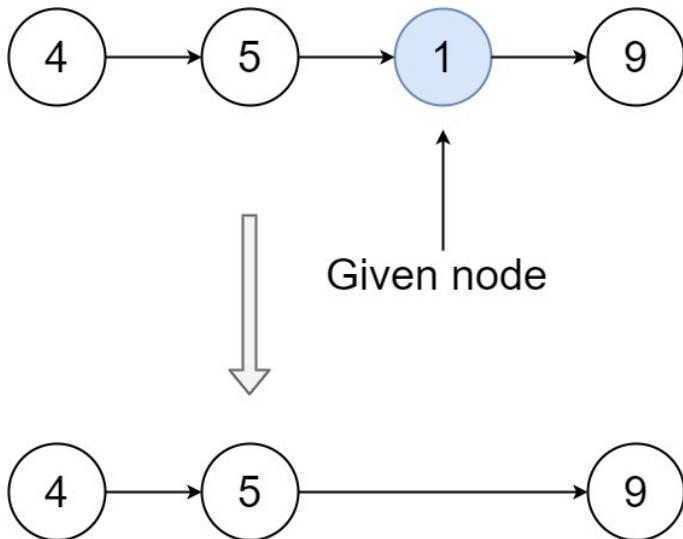


**Input:** head = [4,5,1,9], node = 5

**Output:** [4,1,9]

**Explanation:** You are given the second node with value 5, the linked list should become 4 → 1

#### Example 2:



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

**Output:** [4,5,9]

**Explanation:** You are given the third node with value 1, the linked list should become 4  $\rightarrow$  5  $\rightarrow$

### Constraints:

- The number of the nodes in the given list is in the range [2, 1000].
- $-1000 \leq \text{Node.val} \leq 1000$
- The value of each node in the list is **unique**.
- The node to be deleted is **in the list** and is **not a tail node**.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
/*
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    void deleteNode(ListNode* node) {
        auto next = node->next;
        node->val = next->val;
        node->next = next->next;
        delete next;
    }
};
```

## 345 Reverse Vowels of a String ([link](#))

### Description

Given a string  $s$ , reverse only all the vowels in the string and return it.

The vowels are 'a', 'e', 'i', 'o', and 'u', and they can appear in both lower and upper cases, more than once.

#### Example 1:

**Input:**  $s = \text{"IceCreAm"}$

**Output:** "AceCreIm"

#### Explanation:

The vowels in  $s$  are ['I', 'e', 'e', 'A']. On reversing the vowels,  $s$  becomes "AceCreIm".

#### Example 2:

**Input:**  $s = \text{"leetcode"}$

**Output:** "leotcede"

#### Constraints:

- $1 \leq s.length \leq 3 * 10^5$
- $s$  consist of **printable ASCII** characters.

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    void swap(char &s1, char &s2) {
        char temp = s1;
        s1 = s2;
        s2 = temp;
    }
    string reverseVowels(string s) {
        deque<unsigned long int> forw, back;

        int len = s.length();
        for (int i = 0, j = len - 1; i < len; i++, j--) {
            if(s[i] == 'a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'o' || s[i] == 'u'
            || s[i] == 'A' || s[i] == 'E' || s[i] == 'I' || s[i] == 'O' || s[i] == 'U')
            {
                forw.push_back(i);
            }
            if(s[j] == 'a' || s[j] == 'e' || s[j] == 'i' || s[j] == 'o' || s[j] == 'u'
            || s[j] == 'A' || s[j] == 'E' || s[j] == 'I' || s[j] == 'O' || s[j] == 'U')
            {
                back.push_back(j);
            }
        }

        // char temp;
        for(int i = 0; i < forw.size() / 2; i++) {
            swap(s[forw[i]], s[back[i]]);
        }
        return s;
    }
};
```

## 1604 Least Number of Unique Integers after K Removals ([link](#))

### Description

Given an array of integers `arr` and an integer `k`. Find the *least number of unique integers* after removing **exactly** `k` elements.

#### Example 1:

```
Input: arr = [5,5,4], k = 1
Output: 1
Explanation: Remove the single 4, only 5 is left.
```

#### Example 2:

```
Input: arr = [4,3,1,1,3,3,2], k = 3
Output: 2
Explanation: Remove 4, 2 and either one of the two 1s or three 3s. 1 and 3 will be left.
```

#### Constraints:

- $1 \leq \text{arr.length} \leq 10^5$
- $1 \leq \text{arr}[i] \leq 10^9$
- $0 \leq k \leq \text{arr.length}$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    int findLeastNumOfUniqueInts(vector<int>& arr, int k) {
        int n=arr.size();
        map<int,int>mp;
        for(int i=0;i<n;i++){
            mp[arr[i]]++;
        }

        int cnt=0;
        vector<pair<int,int>> v;
        for(auto it=mp.begin();it!=mp.end();it++){
            v.push_back({it->second,it->first});
        }
        sort(v.begin(),v.end());

        for(int i=0;i<v.size();i++){
            if(cnt+v[i].first<=k){
                cnt+=v[i].first;
                mp.erase(v[i].second);
            }
            else
                break;
        }
        return mp.size();
    }
};
```

## [1603 Running Sum of 1d Array \(link\)](#)

### Description

Given an array `nums`. We define a running sum of an array as `runningSum[i] = sum(nums[0]...nums[i])`.

Return the running sum of `nums`.

#### Example 1:

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

**Output:** `[1,3,6,10]`

**Explanation:** Running sum is obtained as follows: `[1, 1+2, 1+2+3, 1+2+3+4]`.

#### Example 2:

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

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

**Explanation:** Running sum is obtained as follows: `[1, 1+1, 1+1+1, 1+1+1+1, 1+1+1+1+1]`.

#### Example 3:

**Input:** `nums = [3,1,2,10,1]`

**Output:** `[3,4,6,16,17]`

### Constraints:

- $1 \leq \text{nums.length} \leq 1000$
- $-10^6 \leq \text{nums}[i] \leq 10^6$

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    vector<int> runningSum(vector<int>& nums) {
        vector<int> sums(nums.size());
        sums[0] = nums[0];
        for(int i = 1; i < nums.size(); i++) {
            sums[i] = sums[i-1] + nums[i];
        }
        return sums;
    }
};
```

## [58 Length of Last Word \(link\)](#)

### Description

Given a string  $s$  consisting of words and spaces, return *the length of the last word in the string*.

A **word** is a maximal substring consisting of non-space characters only.

#### Example 1:

```
Input: s = "Hello World"
Output: 5
Explanation: The last word is "World" with length 5.
```

#### Example 2:

```
Input: s = "    fly me   to the moon "
Output: 4
Explanation: The last word is "moon" with length 4.
```

#### Example 3:

```
Input: s = "luffy is still joyboy"
Output: 6
Explanation: The last word is "joyboy" with length 6.
```

### Constraints:

- $1 \leq s.length \leq 10^4$
- $s$  consists of only English letters and spaces ' '.
- There will be at least one word in  $s$ .

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution
{
public:
    int lengthOfLastWord(string s)
    {
        int len = 0, tail = s.length() - 1;
        while (tail >= 0 && s[tail] == ' ')
            tail--;
        while (tail >= 0 && s[tail] != ' ')
        {
            len++;
            tail--;
        }
        return len;
    };
};
```

## [178 Rank Scores \(link\)](#)

### Description

Table: Scores

Column Name	Type
id	int
score	decimal

id is the primary key (column with unique values) for this table.

Each row of this table contains the score of a game. Score is a floating point value with two decimal places.

Write a solution to find the rank of the scores. The ranking should be calculated according to the following rules:

- The scores should be ranked from the highest to the lowest.
- If there is a tie between two scores, both should have the same ranking.
- After a tie, the next ranking number should be the next consecutive integer value. In other words, there should be no holes between ranks.

Return the result table ordered by score in descending order.

The result format is in the following example.

### Example 1:

#### Input:

Scores table:

id	score
1	3.50
2	3.65
3	4.00
4	3.85
5	4.00
6	3.65

#### Output:

score	rank
4.00	1
4.00	1
3.85	2
3.65	3
3.65	3
3.50	4

(scroll down for solution)

## Solution

Language: mysql

Status: Accepted

```
select Score, rank
from Scores a
left join
(
    select dscore, @rank := @rank + 1 as rank
    from
    (
        select distinct Score as dscore
        from Scores
        order by dscore desc
    ) s,
    (
        select @rank := 0
    ) r
) as b
on a.Score = b.dsore
order by Score desc
```

## [177 Nth Highest Salary \(link\)](#)

### Description

Table: Employee

Column Name	Type
id	int
salary	int

`id` is the primary key (column with unique values) for this table.  
Each row of this table contains information about the salary of an employee.

Write a solution to find the  $n^{\text{th}}$  highest **distinct** salary from the Employee table. If there are less than  $n$  distinct salaries, return null.

The result format is in the following example.

#### Example 1:

**Input:**  
Employee table:

id	salary
1	100
2	200
3	300

`n = 2`

**Output:**

getNthHighestSalary(2)
200

#### Example 2:

**Input:**  
Employee table:

id	salary
1	100

`n = 2`

**Output:**

getNthHighestSalary(2)
null

(scroll down for solution)

## Solution

Language: mysql

Status: Accepted

```
CREATE FUNCTION getNthHighestSalary(N INT) RETURNS INT
BEGIN
  SET N=N-1;
  RETURN (
    # Write your MySQL query statement below.
    select (select distinct salary from employee
            order by salary desc limit N,1) as Salary
  );
END
```

## [176 Second Highest Salary \(link\)](#)

### Description

Table: Employee

Column Name	Type
id	int
salary	int

id is the primary key (column with unique values) for this table.  
Each row of this table contains information about the salary of an employee.

Write a solution to find the second highest **distinct** salary from the Employee table. If there is no second highest salary, return null (return None in Pandas).

The result format is in the following example.

#### Example 1:

**Input:**  
Employee table:

id	salary
1	100
2	200
3	300

**Output:**

SecondHighestSalary
200

#### Example 2:

**Input:**  
Employee table:

id	salary
1	100

**Output:**

SecondHighestSalary
null

(scroll down for solution)

## Solution

*Language: mysql*

**Status: Accepted**

```
# Write your MySQL query statement below
select (select distinct Salary from Employee
order by Salary desc limit 1 offset 1) as SecondHighestSalary ;
```

## 175 Combine Two Tables (link)

### Description

Table: Person

Column Name	Type
personId	int
lastName	varchar
firstName	varchar

personId is the primary key (column with unique values) for this table.  
This table contains information about the ID of some persons and their first and last names.

Table: Address

Column Name	Type
addressId	int
personId	int
city	varchar
state	varchar

addressId is the primary key (column with unique values) for this table.  
Each row of this table contains information about the city and state of one person with ID = Pe

Write a solution to report the first name, last name, city, and state of each person in the Person table. If the address of a personId is not present in the Address table, report null instead.

Return the result table in **any order**.

The result format is in the following example.

### Example 1:

#### Input:

Person table:

personId	lastName	firstName
1	Wang	Allen
2	Alice	Bob

Address table:

addressId	personId	city	state
1	2	New York City	New York
2	3	Leetcode	California

#### Output:

firstName	lastName	city	state
Allen	Wang	Null	Null
Bob	Alice	New York City	New York

#### Explanation:

There is no address in the address table for the personId = 1 so we return null in their city and addressId = 1 contains information about the address of personId = 2.

(scroll down for solution)



## Solution

*Language: mysql*

**Status: Accepted**

```
select FirstName, LastName, City, State
from Person left join Address
on Person.PersonId = Address.PersonId
;
```

## [13 Roman to Integer \(link\)](#)

### Description

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

- I can be placed before V (5) and X (10) to make 4 and 9.
- X can be placed before L (50) and C (100) to make 40 and 90.
- C can be placed before D (500) and M (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

#### Example 1:

```
Input: s = "III"
Output: 3
Explanation: III = 3.
```

#### Example 2:

```
Input: s = "LVIII"
Output: 58
Explanation: L = 50, V= 5, III = 3.
```

#### Example 3:

```
Input: s = "MCMXCIV"
Output: 1994
Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.
```

#### Constraints:

- $1 \leq s.length \leq 15$
- s contains only the characters ('I', 'V', 'X', 'L', 'C', 'D', 'M').
- It is **guaranteed** that s is a valid roman numeral in the range [1, 3999].

(scroll down for solution)

## Solution

Language: cpp

Status: Accepted

```
class Solution {
public:
    long int romanToInt(string s) {
        long int result = 0;
        for(int i = 0; i < s.length(); i++) {
            if(s[i] == 'M') {
                if(s[i - 1] == 'C') {
                    result += 900;
                }
                else {
                    result += 1000;
                }
            }
            if(s[i] == 'D') {
                if(s[i - 1] == 'C') {
                    result += 400;
                }
                else {
                    result += 500;
                }
            }
            if(s[i] == 'C') {
                if(s[i - 1] == 'X') {
                    result += 90;
                }
                else if (i-1 == -1) {
                    if((s[i+1] == 'M' || s[i+1] == 'D')) {
                        continue;
                    }
                    else {
                        result += 100;
                    }
                }
                else if((s[i+1] == 'M' || s[i+1] == 'D')){
                    continue;
                }
                else {
                    result += 100;
                }
            }
            if(s[i] == 'L') {
                if(s[i - 1] == 'X') {
                    result += 40;
                }
                else {
                    result += 50;
                }
            }
            if(s[i] == 'X') {
                if(s[i - 1] == 'I') {
                    result += 9;
                }
                else if (i-1 == -1) {
                    if(s[i+1] == 'L' || s[i+1] == 'C') {
                        continue;
                    }
                    else {
                        result += 10;
                    }
                }
                else if(s[i+1] == 'L' || s[i+1] == 'C') {
                    continue;
                }
                else {
                    result += 10;
                }
            }
            if(s[i] == 'V') {
                if(s[i - 1] == 'I') {
                    result += 4;
                }
                else {
                    result += 5;
                }
            }
        }
    }
};
```

```
if(s[i] == 'I') {  
    if (s[i+1] == 'V' || s[i+1] == 'X') {  
        continue;  
    } else {  
        result += 1;  
    }  
}  
return result;  
};  
};
```