997. Find the Town Judge

Easy Topics Companies

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

If the town judge exists, then:

1. The town judge trusts nobody.

2. Everybody (except for the town judge) trusts the town judge.

3. There is exactly one person that satisfies properties **1** and **2**.

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

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

Example 1:

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

Example 2:

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

Example 3:

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

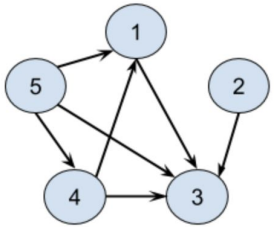
Constraints:

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

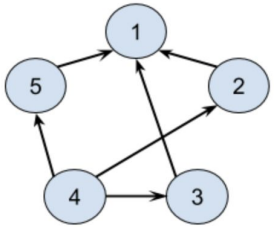
```python
class Solution:
    def findJudge(self, n: int, trust: List[List[int]]) -> int:

```

The `trust` relationships form a graph. Each `trust` pair, `[a, b]` represents a **directed edge** going from `a` to `b`.

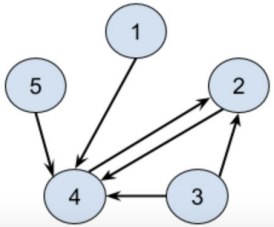
For example, with `N = 5` and `trust = [[1,3],[2,3],[4,3],[4,1],[5,3],[5,1],[5,4]]`, we get the following graph. Who is the town judge?



What about this example, with `trust = [[2,1],[3,1],[4,2],[4,3],[4,5],[5,1]]`?



And what about this example, with `trust = [[1,4],[2,4],[3,2],[3,4],[4,2],[5,4]]`?



Description | Editorial | Solutions | Submissions

997. Find the Town Judge

Solved

Easy | Topics | Companies

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

If the town judge exists, then:

1. The town judge trusts nobody.

2. Everybody (except for the town judge) trusts the town judge.

3. There is exactly one person that satisfies properties **1** and **2**.

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

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

Example 1:

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

Example 2:

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

Example 3:

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

Constraints:

6.7K | 217

Code

Python3 • Auto

```python
class Solution:
    def findJudge(self, N: int, trust: List[List[int]]) -> int:

        if len(trust) < N - 1:
            return -1

        indegree = [0] * (N + 1)
        outdegree = [0] * (N + 1)

        for a, b in trust:
            outdegree[a] += 1
            indegree[b] += 1

        for i in range(1, N + 1):
            if indegree[i] == N - 1 and outdegree[i] == 0:
                return i
        return -1
```

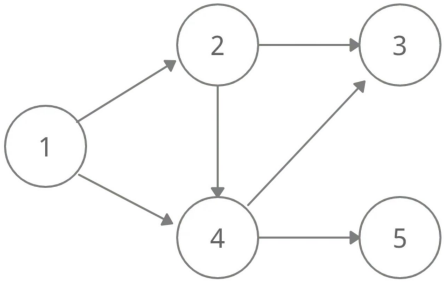
Saved

Ln 17, Col 26

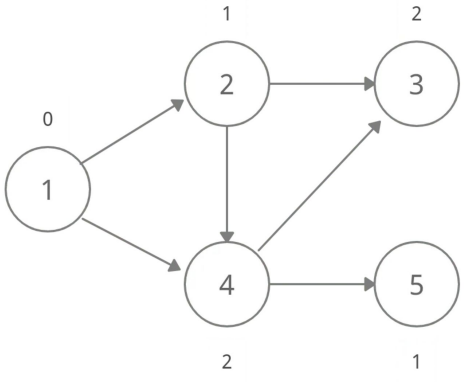
Testcase | Test Result

Kahn's Algo to find Top Sort

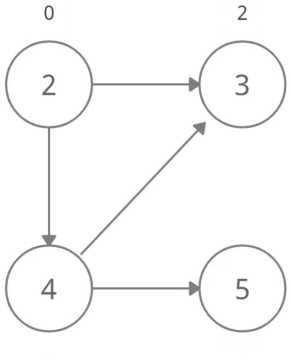
Consider the following graph.



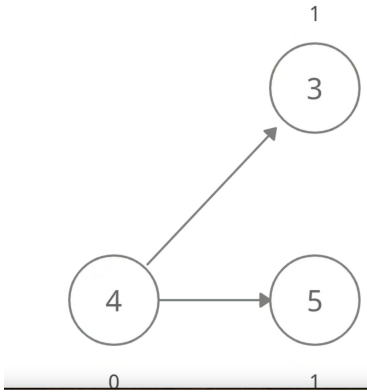
According to Kahn's Algo, First, we need to find the indegree of all the vertices.



The graph will look like this after removing the vertex `1` and its outgoing edges.



Delete the vertex `2` from the graph and all of its outgoing edges. Recalculate the indegree of the remaining vertices, the graph will look like this.



0

If we consider vertex `3` first, the topological order will become -

```
Topological order - 1 2 4 3 5
```

Description | Editorial | Solutions | Submissions

207. Course Schedule

Solved

Medium | Topics | Companies | Hint

There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses - 1`. You are given an array `prerequisites` where `prerequisites[i] = [a_i, b_i]` indicates that you **must** take course `b_i` first if you want to take course `a_i`.

- For example, the pair `[0, 1]`, indicates that to take course `0` you have to first take course `1`.

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

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
finished course 1. So it is impossible.
```

Constraints:

- `1 <= numCourses <= 2000`
- `0 <= prerequisites.length <= 5000`
- `prerequisites[i].length == 2`

16.5K | 192

Code

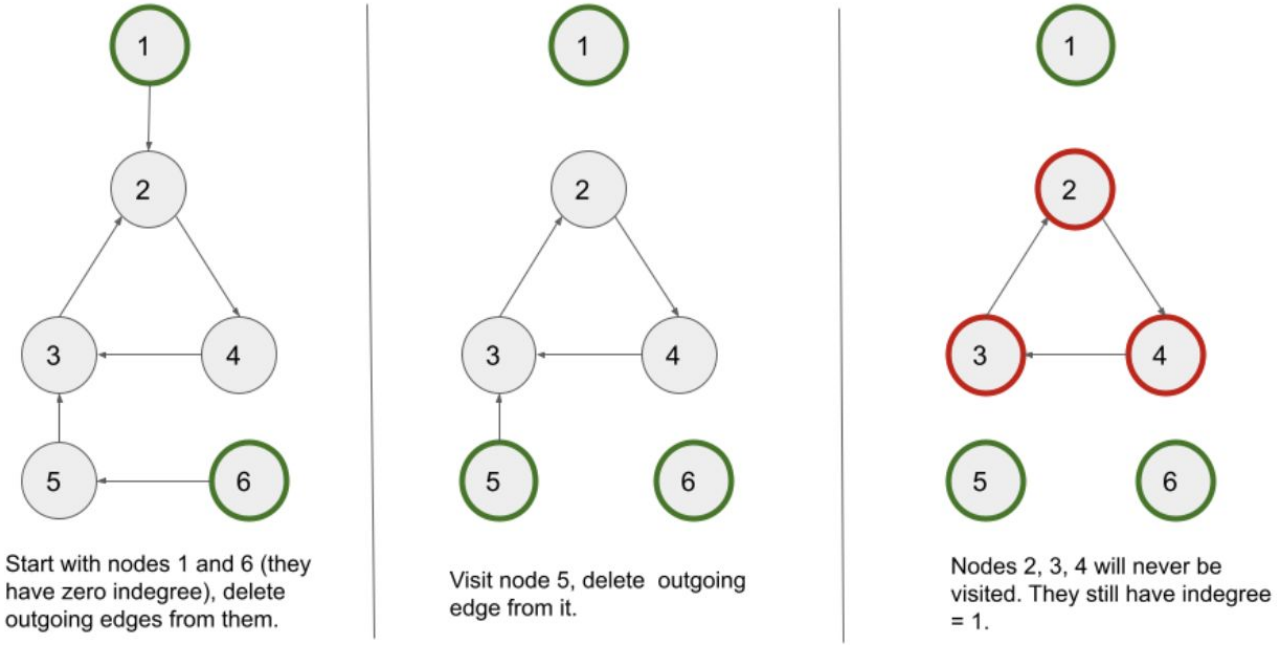
Python3 • Auto

```python
class Solution:
    def canFinish(self, numCourses: int, prerequisites: List[List[int]]) -> bool:

```

Saved | Ln 1, Col 1

Testcase | Test Result

Let's perform Kahn's algorithm on a directed graph having a cycle. Here's a visual step-by-step representation of how it would work:



We can see that if there is a cycle, the indegree of nodes in the cycle cannot be set to `0` due to cyclic dependency. We are unable to visit the cycle's nodes. So, if the number of visited nodes is less than the total number of nodes in the graph, we have a cycle.

207. Course Schedule

Solved

Medium Topics Companies Hint

There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses - 1`. You are given an array `prerequisites` where `prerequisites[i] = [a_i, b_i]` indicates that you **must** take course `b_i` first if you want to take course `a_i`.

- For example, the pair `[0, 1]`, indicates that to take course `0` you have to first take course `1`.

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

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
finished course 1. So it is impossible.
```

Constraints:

- `1 <= numCourses <= 2000`
- `0 <= prerequisites.length <= 5000`
- `prerequisites[i].length == 2`
- `0 <= a_i, b_i < numCourses`

Python3 • Auto

```python
class Solution:
    def canFinish(self, numCourses, prerequisites):
        indegree = [0] * numCourses
        adj = [[] for _ in range(numCourses)]

        for prerequisite in prerequisites:
            adj[prerequisite[1]].append(prerequisite[0])
            indegree[prerequisite[0]] += 1

        queue = deque()
        for i in range(numCourses):
            if indegree[i] == 0:
                queue.append(i)

        nodesVisited = 0
        while queue:
            node = queue.popleft()
            nodesVisited += 1

            for neighbor in adj[node]:
                indegree[neighbor] -= 1
                if indegree[neighbor] == 0:
                    queue.append(neighbor)

        return nodesVisited == numCourses
```

Description | Editorial | Solutions | Submissions

20. Valid Parentheses

Solved

Easy | Topics | Companies | Hint

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

24.6K | 418

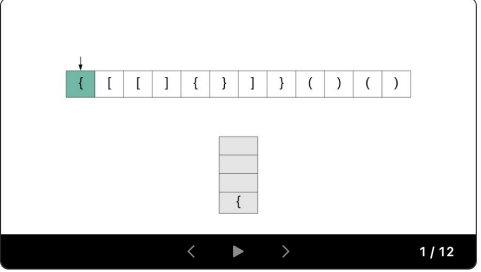
Code

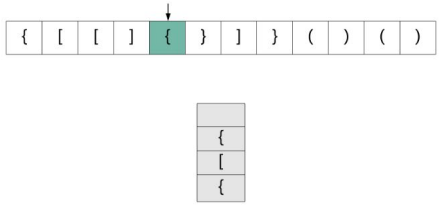
Python3 • Auto

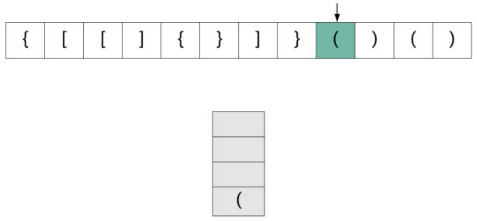
```python
class Solution:
    def isValid(self, s: str) -> bool:

```

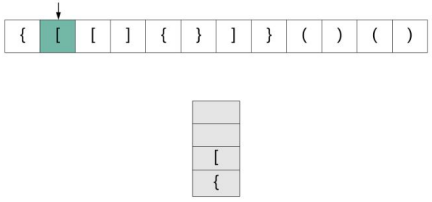
Saved

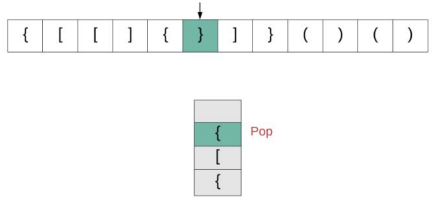
Testcase | Test Result

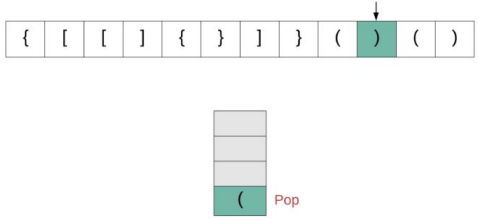

{ [[ ] { }] } ( ) ( )
{
1 / 12

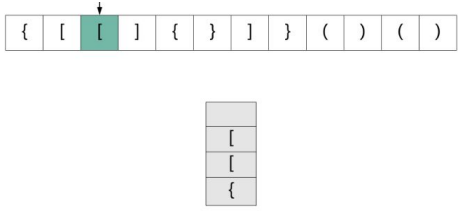

{ [[ ] { }] } ( ) ( )
{
[
{

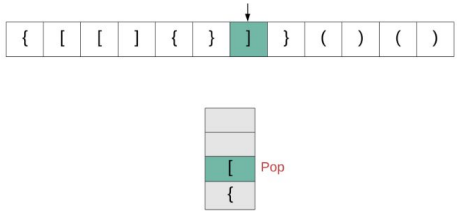

{ [[ ] { }] } ( ) ( )
(

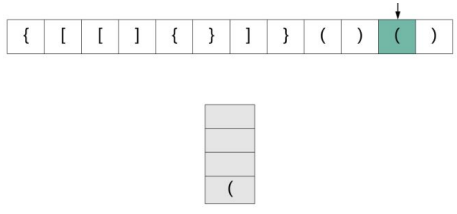

{ [[ ] { }] } ( ) ( )
[
{

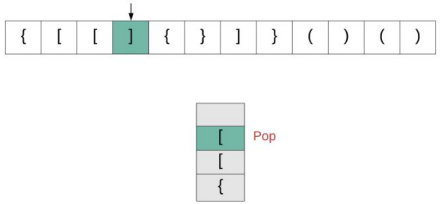

{ [[ ] { }] } ( ) ( )
{
Pop
[
{

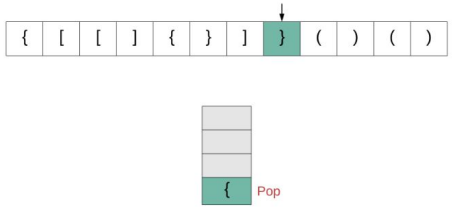

{ [[ ] { }] } ( ) ( )
(
Pop

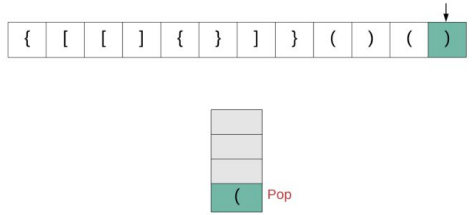

{ [[ ] { }] } ( ) ( )
[
[
{


{ [[ ] { }] } ( ) ( )
[
Pop
{


{ [[ ] { }] } ( ) ( )
(


{ [[ ] { }] } ( ) ( )
[
Pop
[
{


{ [[ ] { }] } ( ) ( )
{
Pop


{ [[ ] { }] } ( ) ( )
(
Pop

Description | Editorial | Solutions | Submissions

20. Valid Parentheses

Solved

Easy | Topics | Companies | Hint

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 = "([])"`

Code

Python3 | Auto

```python
class Solution(object):
    def isValid(self, s: str) -> bool:

        # The stack to keep track of opening brackets.
        stack = []

        # Hash map for keeping track of mappings. This keeps the code very clean.
        # Also makes adding more types of parenthesis easier
        mapping = {")": "(", "}": "{", "]": "["}

        # For every bracket in the expression.
        for char in s:

            # If the character is an closing bracket
            if char in mapping:

                # Pop the topmost element from the stack, if it is non empty
                # Otherwise assign a dummy value of '#' to the top_element variable
                top_element = stack.pop() if stack else "#"

                # The mapping for the opening bracket in our hash and the top
                # element of the stack don't match, return False
                if mapping[char] != top_element:
                    return False
            else:
                # We have an opening bracket, simply push it onto the stack.
                stack.append(char)

        # In the end, if the stack is empty, then we have a valid expression.
        # The stack won't be empty for cases like (((
        return not stack
```

Saved | Ln 31, Col 25

Testcase | Test Result

Description | Editorial | Solutions | Submissions

739. Daily Temperatures

Medium | Topics | Companies | Hint

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

Example 1:

```
Input: temperatures = [73,74,75,71,69,72,76,73]
Output: [1,1,4,2,1,1,0,0]
```

Example 2:

```
Input: temperatures = [30,40,50,60]
Output: [1,1,1,0]
```

Example 3:

```
Input: temperatures = [30,60,90]
Output: [1,1,0]
```

Constraints:

- `1 <= temperatures.length <= 10^5`
- `30 <= temperatures[i] <= 100`

Seen this question in a real interview before? 1/5

Yes | No

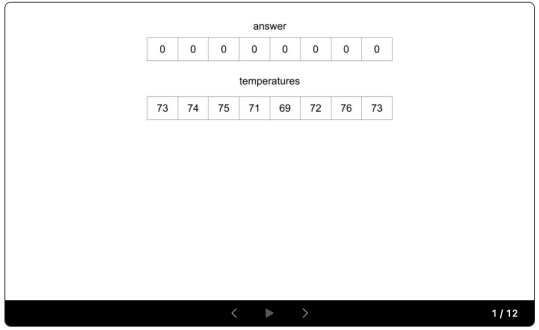
Accepted 1.1M | Submissions 1.7M | Acceptance Rate 66.5%

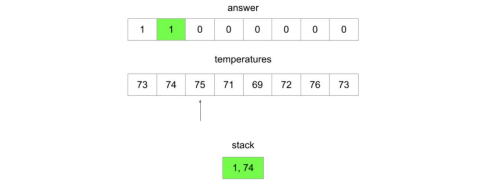
Topics

Code

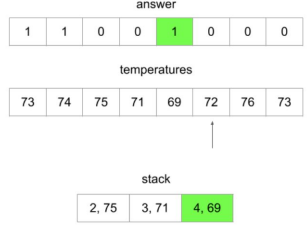
Python3 | Auto

```python
class Solution:
    def dailyTemperatures(self, temperatures: List[int]) -> List[int]:

```

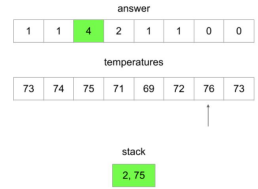




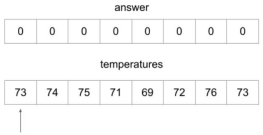
75 > 74. That means today is the first day with a warmer temperature than day 1. Set answer[1] = 2 - 1 = 1 and pop from the stack. After popping, the stack is empty, so push the current day and move on.



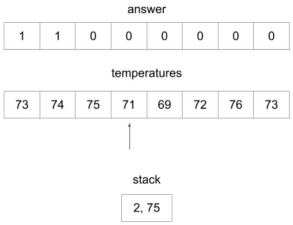
72 is warmer than 69. Set answer[4] = 5 - 4 = 1 and pop from the stack.



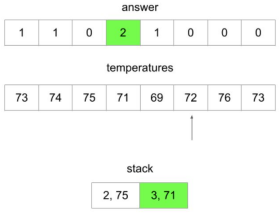
76 is warmer than 75. Set answer[2] = 6 - 2 = 4 and pop from the stack. After popping, the stack is empty, so just push the current day and move on.



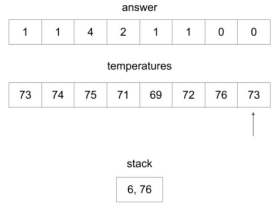
Stack is empty for the first element, so push the first element to stack and move on.



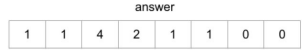
71 is not warmer than 75, so just push it onto the stack and move on.



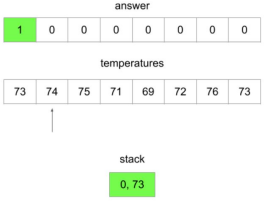
72 is warmer than 71. Set answer[3] = 5 - 3 = 2 and pop from the stack.
After popping, 72 is not warmer than 75, so push the current day onto the stack and move on.



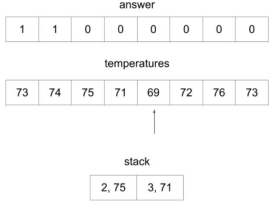
73 is not warmer than 76, so just push the current day and move on.



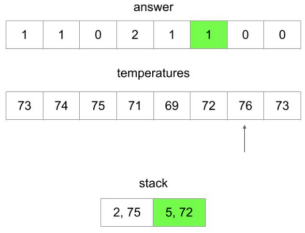
After iteration, this is our final answer.



74 > 73. That means today is the first day with a warmer temperature than day 0. Set answer[0] = 1 - 0 = 1 and pop from the stack. After popping, the stack is empty, so push the current day and move on.



69 is not warmer than 71, so just push it onto the stack and move on.



76 is warmer than 72. Set answer[5] = 6 - 5 = 1 and pop from the stack.

Description | Editorial | Solutions | Submissions

739. Daily Temperatures

Solved

Medium | Topics | Companies | Hint

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

Example 1:

```
Input: temperatures = [73,74,75,71,69,72,76,73]
Output: [1,1,4,2,1,1,0,0]
```

Example 2:

```
Input: temperatures = [30,40,50,60]
Output: [1,1,1,0]
```

Example 3:

```
Input: temperatures = [30,60,90]
Output: [1,1,0]
```

Constraints:

- `1 <= temperatures.length <= 10^5`
- `30 <= temperatures[i] <= 100`

Seen this question in a real interview before? 1/5

Yes | No

Code

Python3 • Auto

```python
class Solution:
    def dailyTemperatures(self, temperatures: List[int]) -> List[int]:
        n = len(temperatures)
        answer = [0] * n
        stack = []

        for curr_day, curr_temp in enumerate(temperatures):
            # Pop until the current day's temperature is not
            # warmer than the temperature at the top of the stack
            while stack and temperatures[stack[-1]] < curr_temp:
                prev_day = stack.pop()
                answer[prev_day] = curr_day - prev_day
            stack.append(curr_day)

        return answer
```

Saved Ln 15, Col 2

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

225. Implement Stack using Queues

Easy | Topics | Companies

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (`push`, `top`, `pop`, and `empty`).

Implement the `MyStack` class:

- `void push(int x)` Pushes element x to the top of the stack.
- `int pop()` Removes the element on the top of the stack and returns it.
- `int top()` Returns the element on the top of the stack.
- `boolean empty()` Returns `true` if the stack is empty, `false` otherwise.

Notes:

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

Example 1:

```
Input
["MyStack", "push", "push", "top", "pop", "empty"]
[[], [1], [2], [], [], []]
Output
[null, null, null, 2, 2, false]

Explanation
MyStack myStack = new MyStack();
myStack.push(1);
myStack.push(2);
myStack.top(); // return 2
myStack.pop(); // return 2
myStack.empty(); // return False
```

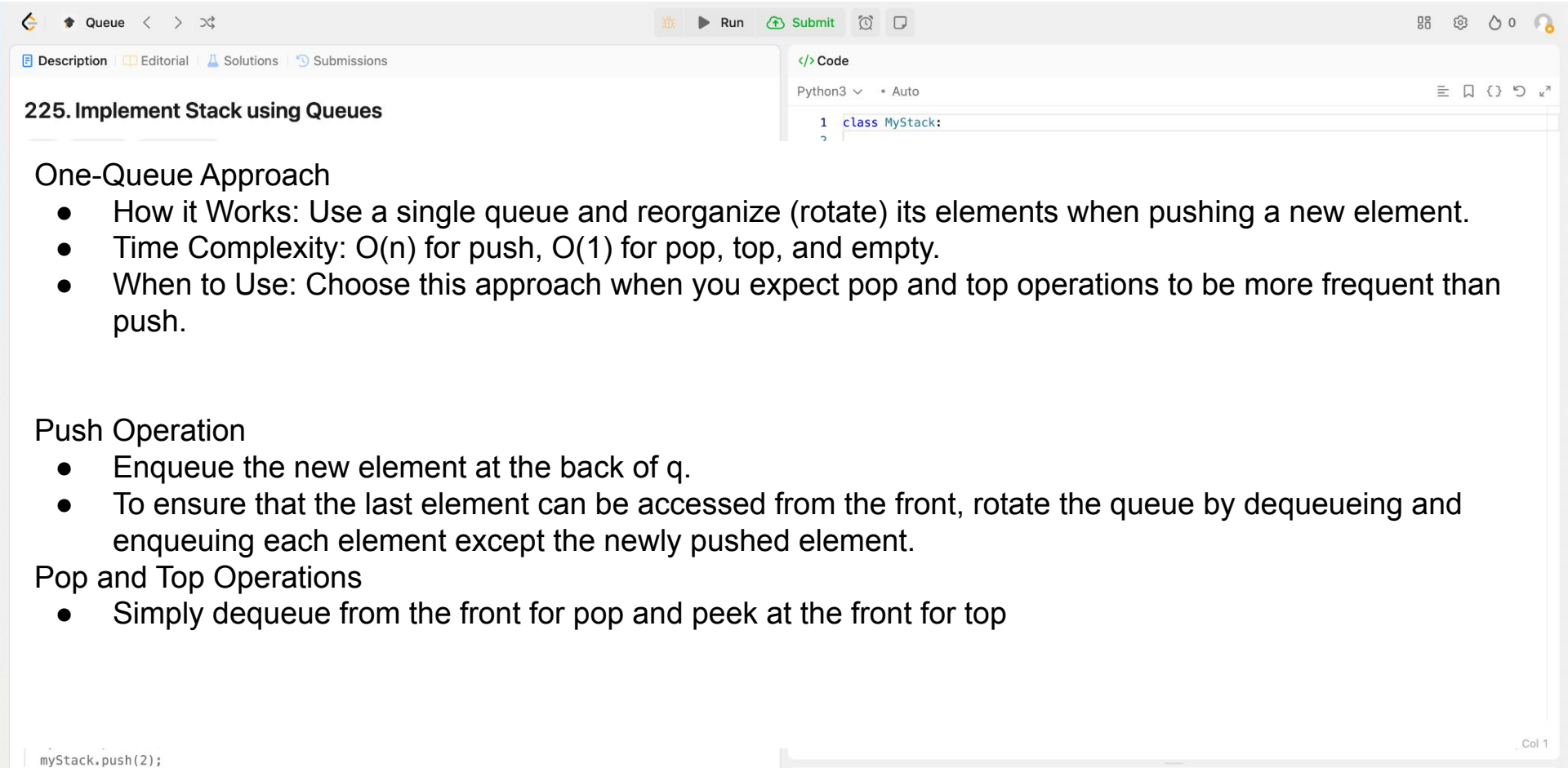
Code

Python3 • Auto

```python
class MyStack:

    def __init__(self):


    def push(self, x: int) -> None:


    def pop(self) -> int:


    def top(self) -> int:


    def empty(self) -> bool:


```

Example 1:

```
Input
["MyStack", "push", "push", "top", "pop", "empty"]
[[], [1], [2], [], [], []]
Output
[null, null, null, 2, 2, false]

Explanation
MyStack myStack = new MyStack();
myStack.push(1);
myStack.push(2);
myStack.top(); // return 2
myStack.pop(); // return 2
myStack.empty(); // return False
```



One-Queue Approach

- How it Works: Use a single queue and reorganize (rotate) its elements when pushing a new element.
- Time Complexity: O(n) for push, O(1) for pop, top, and empty.
- When to Use: Choose this approach when you expect pop and top operations to be more frequent than push.

Push Operation

- Enqueue the new element at the back of q.
- To ensure that the last element can be accessed from the front, rotate the queue by dequeueing and enqueuing each element except the newly pushed element.

Pop and Top Operations

- Simply dequeue from the front for pop and peek at the front for top

Queue

Run Submit

Description | Editorial | Solutions | Submissions

225. Implement Stack using Queues

Easy | Topics | Companies

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (`push`, `top`, `pop`, and `empty`).

Implement the `MyStack` class:

- `void push(int x)` Pushes element x to the top of the stack.
- `int pop()` Removes the element on the top of the stack and returns it.
- `int top()` Returns the element on the top of the stack.
- `boolean empty()` Returns `true` if the stack is empty, `false` otherwise.

Notes:

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

Example 1:

```
Input
["MyStack", "push", "push", "top", "pop", "empty"]
[[], [1], [2], [], [], []]
Output
[null, null, null, 2, 2, false]

Explanation
MyStack myStack = new MyStack();
myStack.push(1);
myStack.push(2);
```

6.2K 61

Code

Python3 • Auto

```python
class MyStack:

    def __init__(self):
        self.q = deque()

    def push(self, x: int) -> None:
        self.q.append(x)
        for _ in range(len(self.q) - 1):
            self.q.append(self.q.popleft())

    def pop(self) -> int:
        return self.q.popleft()

    def top(self) -> int:
        return self.q[0]

    def empty(self) -> bool:
        return len(self.q) == 0
```

Saved Ln 18, Col 32

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1

Input

["MyStack","push","push","top","pop","empty"]

950. Reveal Cards In Increasing Order

Medium | Topics | Companies

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

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

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

1. Take the top card of the deck, reveal it, and take it out of the deck.

2. If there are still cards in the deck then put the next top card of the deck at the bottom of the deck.

3. If there are still unrevealed cards, go back to step 1. Otherwise, stop.

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

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

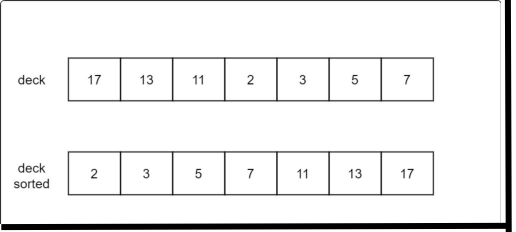
Example 1:

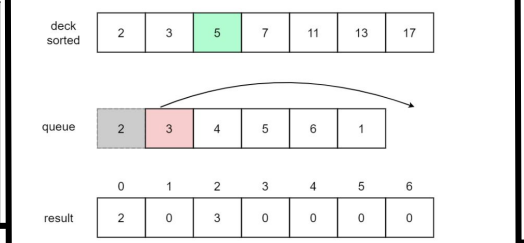
```
Input: deck = [17,13,11,2,3,5,7]
Output: [2,13,3,11,5,17,7]
Explanation:
We get the deck in the order [17,13,11,2,3,5,7] (this order does not matter), and reorder it.
After reordering, the deck starts as [2,13,3,11,5,17,7], where 2 is the top of the deck.
We reveal 2, and move 13 to the bottom.  The deck is now [3,11,5,17,7,13].
We reveal 3, and move 11 to the bottom.  The deck is now [5,17,7,13,11].
We reveal 5, and move 17 to the bottom.  The deck is now [7,13,11,17].
We reveal 7, and move 13 to the bottom.  The deck is now [11,17,13].
We reveal 11, and move 17 to the bottom.  The deck is now [13,17].
We reveal 13, and move 17 to the bottom.  The deck is now [17].
We reveal 17.
Since all the cards revealed are in increasing order, the answer is correct.
```

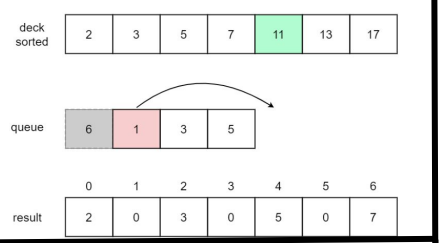
Example 2:

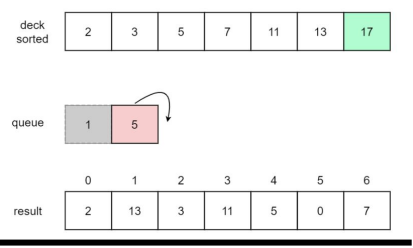
```
Input: deck = [1,1000]
Output: [1,1000]
```

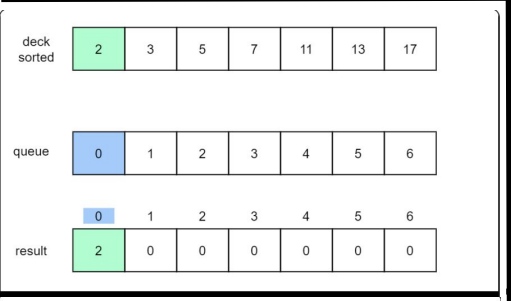
```python
class Solution:
    def deckRevealedIncreasing(self, deck: List[int]) -> List[int]:

```

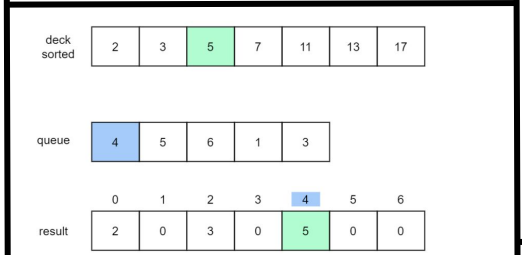

deck
17 13 11 2 3 5 7
deck sorted
2 3 5 7 11 13 17

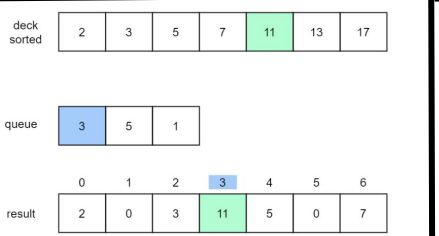

deck sorted
2 3 5 7 11 13 17
queue
2 3 4 5 6 1
0 1 2 3 4 5 6
result
2 0 3 0 0 0 0

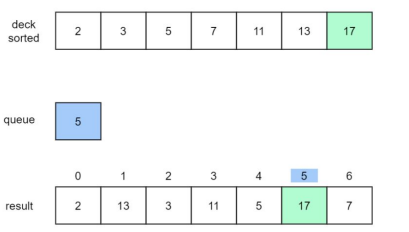

deck sorted
2 3 5 7 11 13 17
queue
6 1 3 5
0 1 2 3 4 5 6
result
2 0 3 0 5 0 7

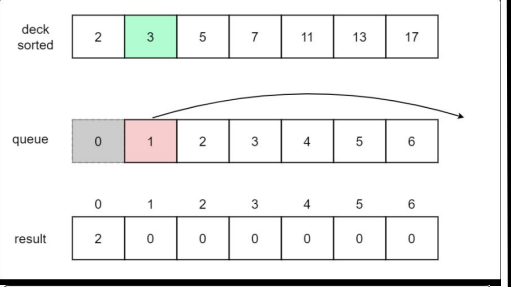

deck sorted
2 3 5 7 11 13 17
queue
1 5
0 1 2 3 4 5 6
result
2 13 3 11 5 0 7

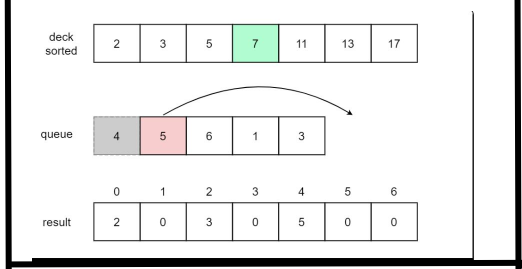

deck sorted
2 3 5 7 11 13 17
queue
0 1 2 3 4 5 6
0 1 2 3 4 5 6
result
2 0 0 0 0 0 0

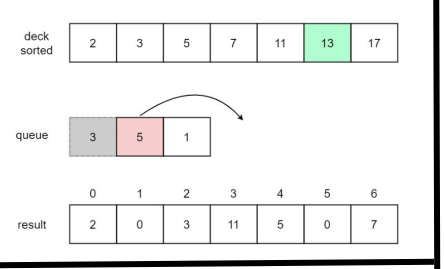

deck sorted
2 3 5 7 11 13 17
queue
4 5 6 1 3
0 1 2 3 4 5 6
result
2 0 3 0 5 0 0

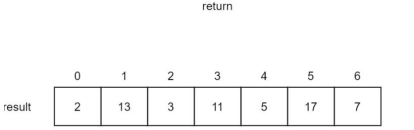

deck sorted
2 3 5 7 11 13 17
queue
3 5 1
0 1 2 3 4 5 6
result
2 0 3 11 5 0 7

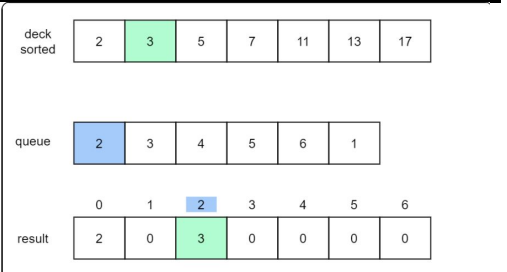

deck sorted
2 3 5 7 11 13 17
queue
5
0 1 2 3 4 5 6
result
2 13 3 11 5 17 7

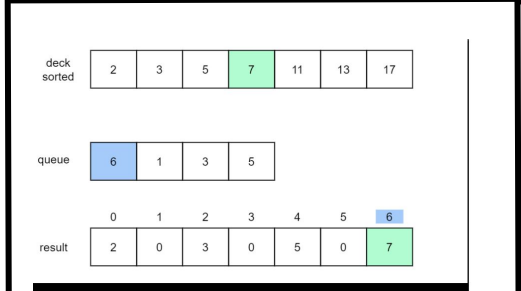

deck sorted
2 3 5 7 11 13 17
queue
0 1 2 3 4 5 6
0 1 2 3 4 5 6
result
2 0 0 0 0 0 0

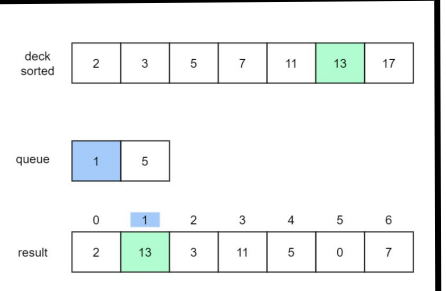

deck sorted
2 3 5 7 11 13 17
queue
4 5 6 1 3
0 1 2 3 4 5 6
result
2 0 3 0 5 0 0


deck sorted
2 3 5 7 11 13 17
queue
3 5 1
0 1 2 3 4 5 6
result
2 0 3 11 5 0 7


return
0 1 2 3 4 5 6
result
2 13 3 11 5 17 7


deck sorted
2 3 5 7 11 13 17
queue
2 3 4 5 6 1
0 1 2 3 4 5 6
result
2 0 3 0 0 0 0


deck sorted
2 3 5 7 11 13 17
queue
6 1 3 5
0 1 2 3 4 5 6
result
2 0 3 0 5 0 7


deck sorted
2 3 5 7 11 13 17
queue
1 5
0 1 2 3 4 5 6
result
2 13 3 11 5 0 7

Description | Editorial | Solutions | Submissions

950. Reveal Cards In Increasing Order

Solved

Medium | Topics | Companies

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

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

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

1. Take the top card of the deck, reveal it, and take it out of the deck.

2. If there are still cards in the deck then put the next top card of the deck at the bottom of the deck.

3. If there are still unrevealed cards, go back to step 1. Otherwise, stop.

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

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

Example 1:

```
Input: deck = [17,13,11,2,3,5,7]
Output: [2,13,3,11,5,17,7]
Explanation:
We get the deck in the order [17,13,11,2,3,5,7] (this order does not matter), and reorder it.
After reordering, the deck starts as [2,13,3,11,5,17,7], where 2 is the top of the deck.
We reveal 2, and move 13 to the bottom.  The deck is now [3,11,5,17,7,13].
We reveal 3, and move 11 to the bottom.  The deck is now [5,17,7,13,11].
We reveal 5, and move 17 to the bottom.  The deck is now [7,13,11,17].
We reveal 7, and move 13 to the bottom.  The deck is now [11,17,13].
We reveal 11, and move 17 to the bottom.  The deck is now [13,17].
We reveal 13, and move 17 to the bottom.  The deck is now [17].
We reveal 17.
Since all the cards revealed are in increasing order, the answer is correct.
```

Example 2:

```
Input: deck = [1,1000]
Output: [1,1000]
```

Code

Python3 | Auto

```python
class Solution:
    def deckRevealedIncreasing(self, deck: List[int]) -> List[int]:
        N = len(deck)
        queue = deque()

        # Create a queue of indexes
        for i in range(N):
            queue.append(i)

        deck.sort()

        # Put cards at correct index in result
        result = [0] * N
        for card in deck:
            # Reveal Card
            result[queue.popleft()] = card

            # Move next card to bottom
            if queue:
                queue.append(queue.popleft())

        return result
```

Saved

Ln 22, Col 22

Testcase | Test Result