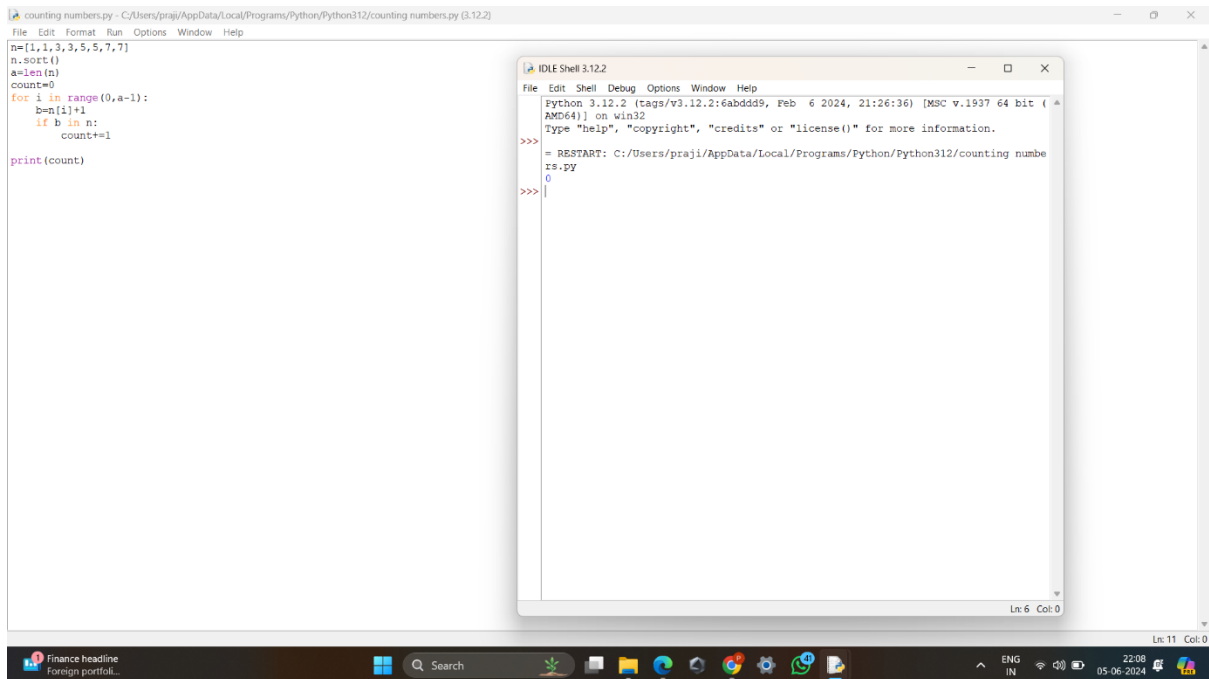


1. Counting Elements Given an integer array arr, count how many elements x there are, such that x + 1 is also in arr. If there are duplicates in arr, count them separately. Example Input: arr = [1,2,3] Output: 2 Explanation: 1 and 2 are counted cause 2 and 3 are in arr. Example 2: Input: arr = [1,1,3,3,5,5,7,7] Output: 0 Explanation: No numbers are counted, cause there is no 2, 4, 6, or 8 in arr.



```
counting numbers.py - C:/Users/praji/AppData/Local/Programs/Python/Python312/counting numbers.py (3.12.2)
File Edit Format Run Options Window Help
arr=[1,1,3,3,5,5,7,7]
n=len(arr)
a=set(arr)
count=0
for i in range(0,n-1):
    b=arr[i]+1
    if b in a:
        count+=1
print(count)
```

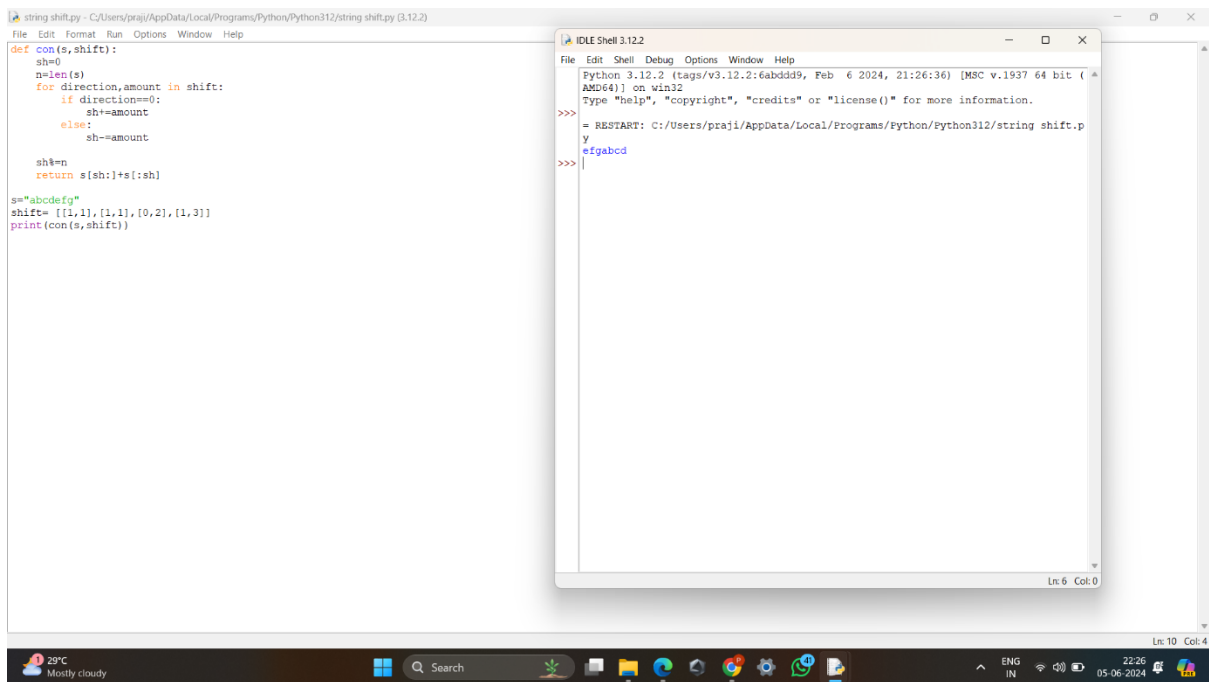
```
IDLE Shell 3.12.2
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb 6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/praji/AppData/Local/Programs/Python/Python312/counting numbers.py
0
>>>
```

Time:  $O(n)$

2. Perform String Shifts You are given a string  $s$  containing lowercase English letters, and a matrix  $shift$ , where  $shift[i] = [direction_i, amount_i]$ :

- $direction_i$  can be 0 (for left shift) or 1 (for right shift).
- $amount_i$  is the amount by which string  $s$  is to be shifted.
- A left shift by 1 means remove the first character of  $s$  and append it to the end.
- Similarly, a right shift by 1 means remove the last character of  $s$  and add it to the beginning.

Return the final string after all operations.



The screenshot shows a Python IDE with two windows. The main window displays a Python script for performing string shifts. The script defines a function `con(s, shift)` that iterates through a list of shifts, applying left or right shifts to a string `s`. The initial string is `abcdefg` and the shift matrix is `[[1,1], [1,1], [0,2], [1,3]]`. The output of the function is printed. A second window, titled 'IDLE Shell 3.12.2', shows the execution of the script, displaying the final string `efgabcd`.

```
string shift.py - C:/Users/praji/AppData/Local/Programs/Python/Python312/string shift.py (3.12.2)
File Edit Format Run Options Window Help
def con(s, shift):
    sh=0
    n=len(s)
    for direction, amount in shift:
        if direction==0:
            sh+=amount
        else:
            sh-=amount
    sh%=n
    return s[sh:] + s[:sh]

s="abcdefg"
shift= [[1,1], [1,1], [0,2], [1,3]]
print(con(s, shift))

IDLE Shell 3.12.2
File Edit Shell Debug Options Window Help
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb 6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> = RESTART: C:/Users/praji/AppData/Local/Programs/Python/Python312/string shift.p
>>> efgabcd
>>> |

Ln: 6 Col: 0
```

Time:  $O(n)$

3. Leftmost Column with at Least a One A row-sorted binary matrix means that all elements are 0 or 1 and each row of the matrix is sorted in non-decreasing order. Given a row-sorted binary matrix `binaryMatrix`, return the index (0-indexed) of the leftmost column with a 1 in it. If such an index does not exist, return -1. You can't access the Binary Matrix directly. You may only access the matrix using a `BinaryMatrix` interface:

- `BinaryMatrix.get(row, col)` returns the element of the matrix at index `(row, col)` (0-indexed).
- `BinaryMatrix.dimensions()` returns the dimensions of the matrix as a list of 2 elements `[rows, cols]`, which means the matrix is `rows` x `cols`.

Submissions making more than 1000 calls to `BinaryMatrix.get` will be judged Wrong Answer. Also, any solutions that attempt to circumvent the judge will result in disqualification. For custom testing purposes, the input will be the entire binary matrix `mat`. You will not have access to the binary matrix directly.

The screenshot shows a Python IDE with a file named `binary_matrix.py`. The script defines a function `generate_row_sorted_binary_matrix` that takes `rows` and `cols` as arguments. It generates a random binary matrix and sorts each row. An example usage is provided, creating a 2x3 matrix and printing its dimensions. An `IDLE Shell 3.12.2` window is open, showing the output of the script: `[[0, 0, 0], [0, 1, 1]]`.

```
def generate_row_sorted_binary_matrix(rows, cols):
    import random
    matrix = [[random.randint(0, 1) for _ in range(cols)] for _ in range(rows)]
    for row in matrix:
        row.sort()
    return matrix

# Example Usage
rows = 2
cols = 3
binary_matrix = generate_row_sorted_binary_matrix(rows, cols)
print(Binary_matrix)
```

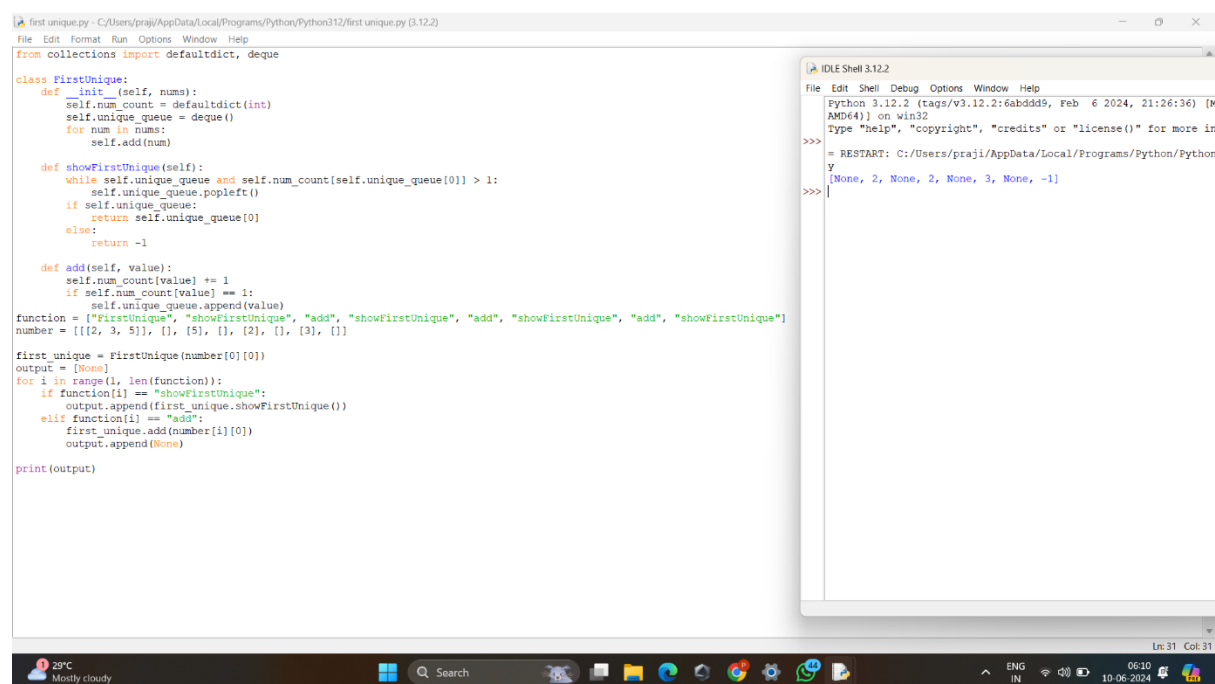
```
>>>
[[0, 0, 0], [0, 1, 1]]
>>>
```

Time:  $O(m \cdot n)$

4. First Unique Number You have a queue of integers, you need to retrieve the first unique integer in the queue. Implement the FirstUnique class: • FirstUnique(int[] nums) Initializes the object with the numbers in the queue. • int showFirstUnique() returns the value of the first unique integer of the queue, and returns -1 if there is no such integer. • void add(int value) insert value to the queue.

Example 1: Input:

["FirstUnique", "showFirstUnique", "add", "showFirstUnique", "add", "showFirstUnique", "add", "showFirstUnique"] [[2,3,5]], [], [5], [], [2], [], [3], [] Output: [null,2,null,2,null,3,null,-1] Explanation: FirstUnique firstUnique = new FirstUnique([2,3,5]); firstUnique.showFirstUnique(); // return 2 firstUnique.add(5); // the queue is now [2,3,5,5] firstUnique.showFirstUnique(); // return 2 firstUnique.add(2); // the queue is now [2,3,5,5,2] firstUnique.showFirstUnique(); // return 3 firstUnique.add(3); // the queue is now [2,3,5,5,2,3] firstUnique.showFirstUnique(); // return -1



```
first_unique.py - C:/Users/praji/AppData/Local/Programs/Python/Python312/first_unique.py (3.12.2)
File Edit Format Run Options Window Help
from collections import defaultdict, deque

class FirstUnique:
    def __init__(self, nums):
        self.num_count = defaultdict(int)
        self.unique_queue = deque()
        for num in nums:
            self.add(num)

    def showFirstUnique(self):
        while self.unique_queue and self.num_count[self.unique_queue[0]] > 1:
            self.unique_queue.popleft()
        if self.unique_queue:
            return self.unique_queue[0]
        else:
            return -1

    def add(self, value):
        self.num_count[value] += 1
        if self.num_count[value] == 1:
            self.unique_queue.append(value)

function = ["FirstUnique", "showFirstUnique", "add", "showFirstUnique", "add", "showFirstUnique", "add", "showFirstUnique"]
number = [[2, 3, 5]], [], [5], [], [2], [], [3], []

first_unique = FirstUnique(number[0][0])
output = [None]
for i in range(1, len(function)):
    if function[i] == "showFirstUnique":
        output.append(first_unique.showFirstUnique())
    elif function[i] == "add":
        first_unique.add(number[i][0])
        output.append(None)

print(output)
```

```
IDLE Shell 3.12.2
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb 6 2024, 21:26:36) [M
AMD64] on win32
Type "help", "copyright", "credits" or "license()" for more in
>>>
= RESTART: C:/Users/praji/AppData/Local/Programs/Python/Python
y
[None, 2, None, 2, None, 3, None, -1]
>>>
```

5. Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree Given a binary tree where each path going from the root to any leaf form a valid sequence, check if a given string is a valid sequence in such binary tree. We get the given string from the concatenation of an array of integers arr and the concatenation of all values of the nodes along a path results in a sequence in the given binary tree. Example 1: Input: root = [0,1,0,0,1,0,null,null,1,0,0], arr = [0,1,0,1] Output: true Explanation: The path 0 -> 1 -> 0 -> 1 is a valid sequence (green color in the figure). Other valid sequences are: 0 -> 1 -> 1 -> 1 -> 0 0 -> 0 -> 0

The image shows a Python IDE with two windows. The background window displays a Python script for a binary tree structure. The script defines a `TreeNode` class, a `isValidSequence` function, and a main execution block. The `isValidSequence` function checks if a sequence of values can form a valid binary search tree. The main block creates a tree with root 0 and children 1 and 1, and then prints the result of `isValidSequence` for the sequence `[0, 1, 0, 1]`.

```
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None

def isValidSequence(root: TreeNode, arr: list) -> bool:
    def dfs(node, index):
        if not node or index >= len(arr) or node.val != arr[index]:
            return False
        if index == len(arr) - 1:
            return not node.left and not node.right
        return dfs(node.left, index + 1) or dfs(node.right, index + 1)

    return dfs(root, 0)

root = TreeNode(0)
root.left = TreeNode(1)
root.right = TreeNode(1)
root.left.left = TreeNode(0)
root.left.right = TreeNode(1)
root.right.left = TreeNode(0)
root.left.left.right = TreeNode(1)
root.left.right.left = TreeNode(0)
root.left.right.right = TreeNode(0)

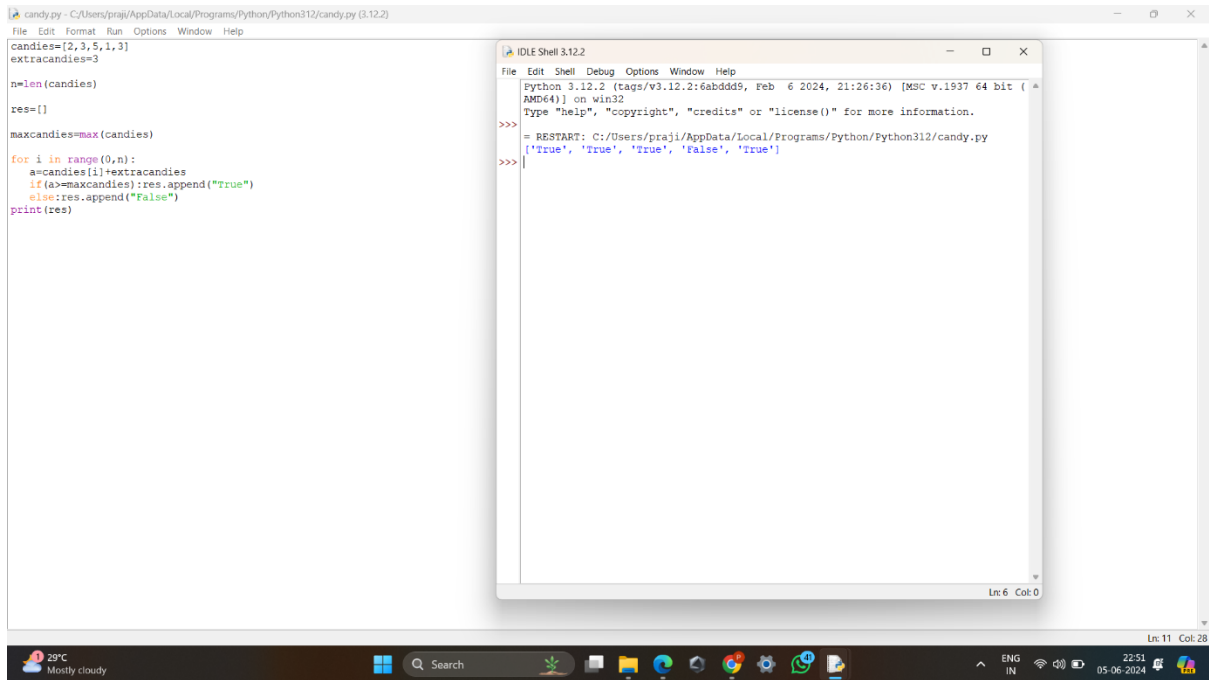
arr = [0, 1, 0, 1]
print(isValidSequence(root, arr))
```

The foreground window shows the IDLE Shell 3.12.2, which displays the output of the script. The output indicates that the sequence `[0, 1, 0, 1]` is not a valid binary search tree sequence, returning `False`.

```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb 6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> = RESTART: C:/Users/praji/AppData/Local/Programs/Python/Python312/binary tree.py
True
>>>
```

TIMEO(n)

6. Kids With the Greatest Number of Candies There are  $n$  kids with candies. You are given an integer array `candies`, where each `candies[i]` represents the number of candies the  $i$ th kid has, and an integer `extraCandies`, denoting the number of extra candies that you have. Return a boolean array `result` of length  $n$ , where `result[i]` is `true` if, after giving the  $i$ th kid all the `extraCandies`, they will have the greatest number of candies among all the kids, or `false` otherwise. Note that multiple kids can have the greatest number of candies.



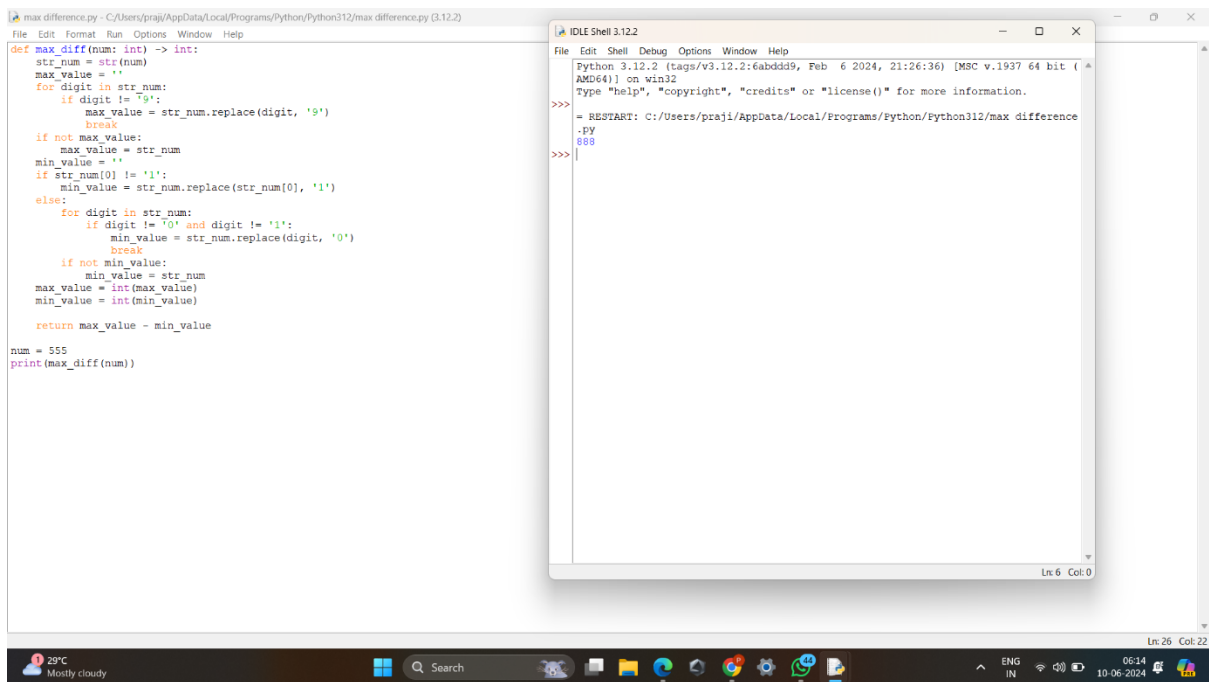
The screenshot shows a Python IDE with a file named `candy.py` and an interactive shell window. The code in `candy.py` defines a function to solve the problem by iterating through the `candies` array, adding `extraCandies` to each element, and comparing it to the maximum value in the array. The shell window shows the execution of the code, resulting in the output `['True', 'True', 'True', 'False', 'True']`.

```
candy.py - C:/Users/praji/AppData/Local/Programs/Python/Python312/candy.py (3.12.2)
File Edit Format Run Options Window Help
candies=[2,3,5,1,3]
extracandies=3
n=len(candies)
res=[]
maxcandies=max(candies)
for i in range(0,n):
    a=candies[i]+extracandies
    if(a>=maxcandies):res.append("True")
    else:res.append("False")
print(res)
```

```
IDLE Shell 3.12.2
File Edit Shell Debug Options Window Help
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb 6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> = RESTART: C:/Users/praji/AppData/Local/Programs/Python/Python312/candy.py
>>> ['True', 'True', 'True', 'False', 'True']
>>>
```

Time:  $O(n)$

7. Max Difference You Can Get From Changing an Integer You are given an integer num. You will apply the following steps exactly two times: • Pick a digit x ( $0 \leq x \leq 9$ ). • Pick another digit y ( $0 \leq y \leq 9$ ). The digit y can be equal to x. • Replace all the occurrences of x in the decimal representation of num by y. • The new integer cannot have any leading zeros, also the new integer cannot be 0. Let a and b be the results of applying the operations to num the first and second times, respectively. Return the max difference between a and b. Example 1: Input: num = 555 Output: 888 Explanation: The first time pick x = 5 and y = 9 and store the new integer in a. The second time pick x = 5 and y = 1 and store the new integer in b. We have now a = 999 and b = 111 and max difference = 888



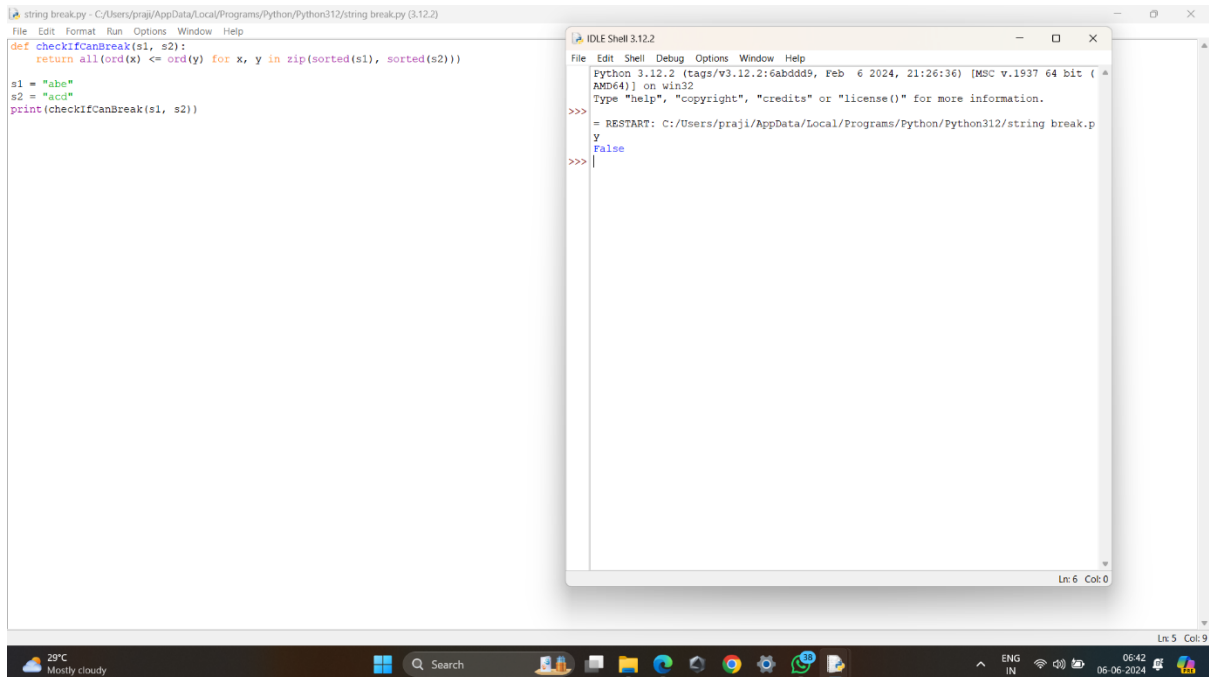
```
def max_diff(num: int) -> int:
    str_num = str(num)
    max_value = ''
    for digit in str_num:
        if digit != '9':
            max_value = str_num.replace(digit, '9')
            break
    if not max_value:
        max_value = str_num
    min_value = ''
    if str_num[0] != '1':
        min_value = str_num.replace(str_num[0], '1')
    else:
        for digit in str_num:
            if digit != '0' and digit != '1':
                min_value = str_num.replace(digit, '0')
                break
    if not min_value:
        min_value = str_num
    max_value = int(max_value)
    min_value = int(min_value)
    return max_value - min_value

num = 555
print(max_diff(num))
```

```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb 6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/praji/AppData/Local/Programs/Python/Python312/max difference
.py
>>>
888
```

TIME:O(n)

8. Check If a String Can Break Another String Given two strings: s1 and s2 with the same size, check if some permutation of string s1 can break some permutation of string s2 or vice-versa. In other words s2 can break s1 or vice-versa. A string x can break string y (both of size n) if  $x[i] \geq y[i]$  (in alphabetical order) for all i between 0 and n-1.



The screenshot shows a Python IDE with a script titled 'string break.py'. The script defines a function 'checkIfCanBreak(s1, s2)' that returns 'all(ord(x) <= ord(y) for x, y in zip(sorted(s1), sorted(s2)))'. It then sets s1 = 'abe', s2 = 'acd', and prints the result of checkIfCanBreak(s1, s2). An IDLE Shell window is open, showing the execution output: 'False'.

```
string break.py - C:/Users/praji/AppData/Local/Programs/Python/Python312/string break.py (3.12.2)
File Edit Format Run Options Window Help
def checkIfCanBreak(s1, s2):
    return all(ord(x) <= ord(y) for x, y in zip(sorted(s1), sorted(s2)))

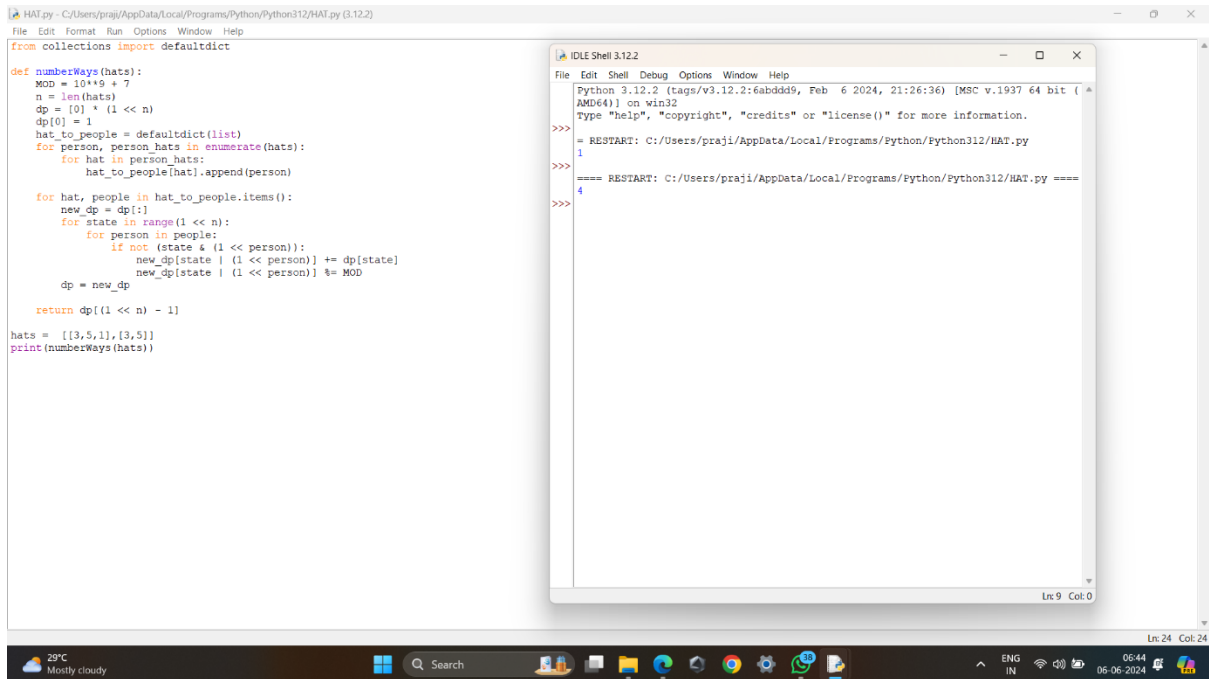
s1 = "abe"
s2 = "acd"
print(checkIfCanBreak(s1, s2))

IDLE Shell 3.12.2
File Edit Shell Debug Options Window Help
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb 6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> = RESTART: C:/Users/praji/AppData/Local/Programs/Python/Python312/string break.p
y
False
>>>
```

Time:  $O(n)$



9. Number of Ways to Wear Different Hats to Each Other There are  $n$  people and 40 types of hats labeled from 1 to 40. Given a 2D integer array `hats`, where `hats[i]` is a list of all hats preferred by the  $i$ th person. Return the number of ways that the  $n$  people wear different hats to each other. Since the answer may be too large, return it modulo  $10^9 + 7$ .



```
from collections import defaultdict

def numberWays(hats):
    MOD = 10**9 + 7
    n = len(hats)
    dp = [0] * (1 << n)
    dp[0] = 1
    hat_to_people = defaultdict(list)
    for person, person_hats in enumerate(hats):
        for hat in person_hats:
            hat_to_people[hat].append(person)

    for hat, people in hat_to_people.items():
        new_dp = dp[:]
        for person in people:
            if not (state & (1 << person)):
                new_dp[state | (1 << person)] += dp[state]
                new_dp[state | (1 << person)] %= MOD
        dp = new_dp

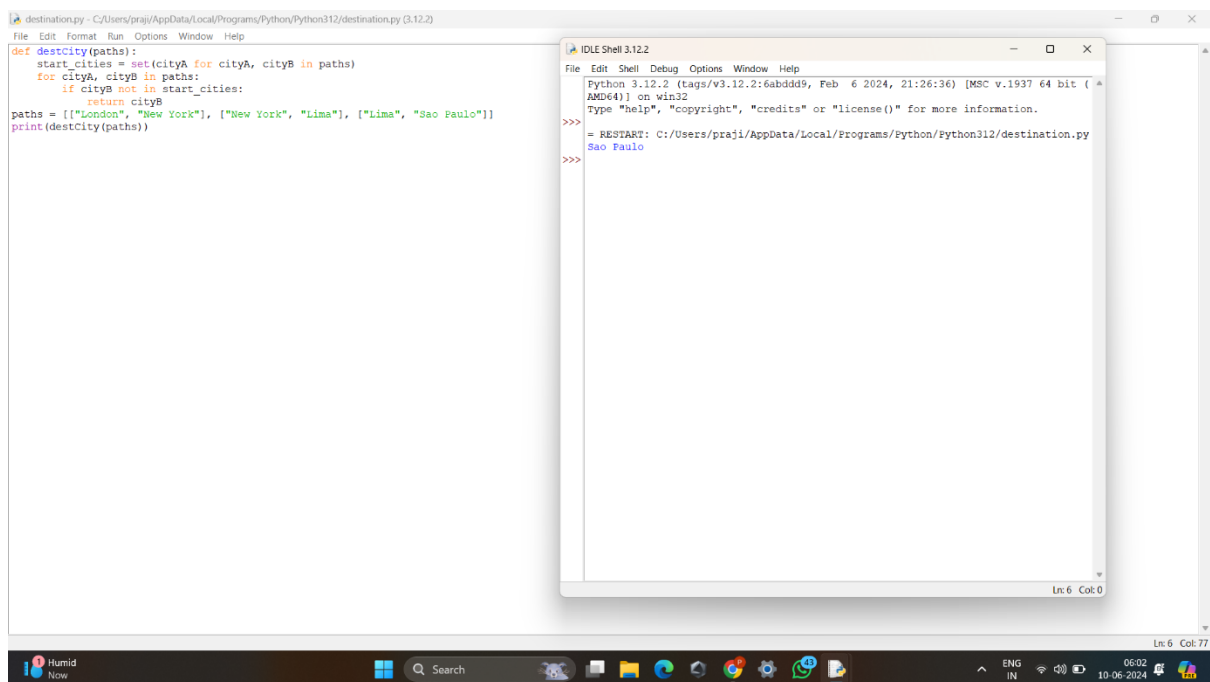
    return dp[(1 << n) - 1]

hats = [[3,5,1],[3,5]]
print(numberWays(hats))
```

```
Python 3.12.2 (tags/v3.12.2:6abdd9, Feb 6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/praji/AppData/Local/Programs/Python/Python312/HAT.py
1
>>>
==== RESTART: C:/Users/praji/AppData/Local/Programs/Python/Python312/HAT.py ====
4
>>>
```

Time:  $O(n)$

10. Destination City You are given the array paths, where paths[i] = [cityAi, cityBi] means there exists a direct path going from cityAi to cityBi. Return the destination city, that is, the city without any path outgoing to another city. It is guaranteed that the graph of paths forms a line without any loop, therefore, there will be exactly one destination city. Example 1: Input: paths = [["London","New York"],["New York","Lima"],["Lima","Sao Paulo"]] Output: "Sao Paulo" Explanation: Starting at "London" city you will reach "Sao Paulo" city which is the destination city. Your trip consist of: "London" -> "New York" -> "Lima" -> "Sao Paulo". Example 2: Input: paths = [["B","C"],["D","B"],["C","A"]] Output: "A" Explanation: All possible trips are: "D" -> "B" -> "C" -> "A". "B" -> "C" -> "A". "C" -> "A". "A". Clearly the destination city is "A".



The screenshot shows a Python IDE with a file named 'destination.py' and an interactive shell window. The code in the file defines a function 'destCity' that takes a list of paths and returns the destination city. The paths are [['London', 'New York'], ['New York', 'Lima'], ['Lima', 'Sao Paulo']]. The shell window shows the execution of the code, resulting in the output 'Sao Paulo'.

```
def destCity(paths):
    start_cities = set(cityA for cityA, cityB in paths)
    for cityA, cityB in paths:
        if cityB not in start_cities:
            return cityB
paths = [["London", "New York"], ["New York", "Lima"], ["Lima", "Sao Paulo"]]
print(destCity(paths))
```

```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb 6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> = RESTART: C:/Users/praji/AppData/Local/Programs/Python/Python312/destination.py
Sao Paulo
>>>
```

Time:  $O(n)$