Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def is_valid_sequence(root, arr):
    def dfs(node, idx):
        if not node or node.val != arr[idx]:
            return False
        if idx == len(arr) - 1:
            return not node.left and not node.right
        if node.left and dfs(node.left, idx + 1):
            return True
        if node.right and dfs(node.right, idx + 1):
            return True
        return False

    return dfs(root, 0)

root = TreeNode(0, TreeNode(1, TreeNode(0, TreeNode(1)), TreeNode(1, TreeNode(0), TreeNode(0))), TreeNode(0, TreeNode(0)))
arr = [0, 1, 0, 1]
print(is_valid_sequence(root, arr))

```

Input

```
True


...Program finished with exit code 0
Press ENTER to exit console.
```

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def count_elements(arr):
        s = set(arr)
        return sum(1 for x in arr if x + 1 in s)

arr = [1, 2, 3]
print(count_elements(arr))
```

input

```
2


...Program finished with exit code 0
Press ENTER to exit console.
```

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def string_shift(s, shifts):
    total_shift = sum(amount if direction == 1 else -amount for direction, amount in shifts) % len(s)
    return s[-total_shift:] + s[:-total_shift]

s = "abcdefg"
shifts = [[1, 1], [1, 1], [0, 2], [1, 3]]
print(string_shift(s, shifts))
```

input

```
efgabcd


...Program finished with exit code 0
Press ENTER to exit console.
```

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
class BinaryMatrix:
    def __init__(self, matrix):
        self.matrix = matrix

    def get(self, x: int, y: int) -> int:
        return self.matrix[x][y]

    def dimensions(self) -> list:
        return [len(self.matrix), len(self.matrix[0])]

def leftmost_column_with_one(binaryMatrix: 'BinaryMatrix') -> int:
    rows, cols = binaryMatrix.dimensions()
    leftmost_col = cols
    for row in range(rows):
        while leftmost_col > 0 and binaryMatrix.get(row, leftmost_col - 1) == 1:
            leftmost_col -= 1
    return leftmost_col if leftmost_col < cols else -1

# Example usage
matrix = [
    [0, 0, 0, 1],
    [0, 0, 1, 1],
    [0, 1, 1, 1],
    [1, 1, 1, 1]
]

binaryMatrix = BinaryMatrix(matrix)
print("Leftmost column with at least one 1:", leftmost_column_with_one(binaryMatrix))

```

input

```
Leftmost column with at least one 1: 0


...Program finished with exit code 0
Press ENTER to exit console.
```

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
from collections import defaultdict, deque

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

    def showFirstUnique(self) -> int:
        while self.queue and self.freq[self.queue[0]] > 1:
            self.queue.popleft()
        return self.queue[0] if self.queue else -1

    def add(self, value: int) -> None:
        self.freq[value] += 1
        if self.freq[value] == 1:
            self.queue.append(value)

nums = [2, 3, 5, 5, 2, 6]
first_unique = FirstUnique(nums)
print("First Unique:", first_unique.showFirstUnique())

first_unique.add(3)
print("First Unique:", first_unique.showFirstUnique())

first_unique.add(6)
print("First Unique:", first_unique.showFirstUnique())

```

input

```
First Unique: 3
First Unique: 6
First Unique: -1


...Program finished with exit code 0
Press ENTER to exit console.
```

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def kids_with_candies(candies, extraCandies):
        max_candies = max(candies)
        return [c + extraCandies >= max_candies for c in candies]

candies = [2, 3, 5, 1, 3]
extraCandies = 3
print(kids_with_candies(candies, extraCandies))

```

input

```
[True, True, True, False, True]


...Program finished with exit code 0
Press ENTER to exit console.
```

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def max_diff(num):
        num_str = str(num)
        max_num = num_str
        min_num = num_str

        for d in num_str:
            if d != '9':
                max_num = num_str.replace(d, '9')
                break

        if num_str[0] != '1':
            min_num = num_str.replace(num_str[0], '1')
        else:
            for d in num_str[1:]:
                if d != '0' and d != '1':
                    min_num = num_str.replace(d, '0')
                    break

        return int(max_num) - int(min_num)

num = 123456
print(max_diff(num))
```

input

```
820000


...Program finished with exit code 0
Press ENTER to exit console.
```

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def check_if_can_break(s1, s2):
    s1, s2 = sorted(s1), sorted(s2)
    return all(c1 >= c2 for c1, c2 in zip(s1, s2)) or all(c2 >= c1 for c1, c2 in zip(s1, s2))

s1 = "abc"
s2 = "xya"
print(check_if_can_break(s1, s2))  # Output: True

```

input

```
True


...Program finished with exit code 0
Press ENTER to exit console.
```

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def number_of_ways(hats):
        from functools import lru_cache
        MOD = 10**9 + 7
        n = len(hats)
        all_people = (1 << n) - 1
        hat_to_people = [[] for _ in range(41)]

        for person, hat_list in enumerate(hats):
            for hat in hat_list:
                hat_to_people[hat].append(person)
        @lru_cache(None)
        def dp(mask, hat):
            if mask == all_people:
                return 1
            if hat > 40:
                return 0
            result = dp(mask, hat + 1)
            for person in hat_to_people[hat]:
                if not (mask & (1 << person)):
                    result += dp(mask | (1 << person), hat + 1)
                    result %= MOD
            return result
        return dp(0, 1)

hats = [[3, 4], [4, 5], [5]]
print(number_of_ways(hats))
```

input

```
1


...Program finished with exit code 0
Press ENTER to exit console.
```

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def dest_city(paths):
        starts = {path[0] for path in paths}
        for _, dest in paths:
            if dest not in starts:
                return dest

paths = [["London", "New York"], ["New York", "Lima"], ["Lima", "Sao Paulo"]]
print(dest_city(paths))
```

input

```
Sao Paulo


...Program finished with exit code 0
Press ENTER to exit console.
```