6.kidswith candies

In [1]:
```
def kidsWithCandies(candies, extraCandies):
    max_candies = max(candies)
    result = [candy + extraCandies >= max_candies for candy in candies]
    return result
candies = [2, 3, 5, 1, 3]
extraCandies = 3
print(kidsWithCandies(candies, extraCandies))
```

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

1.count the elements

In [2]:
```
def count_elements(arr):
    return sum(1 for x in arr if x + 1 in arr)
arr1 = [1, 2, 3]
print(count_elements(arr1))
arr2 = [1, 1, 3, 3, 5, 5, 7, 7]
print(count_elements(arr2))
```

```
2
0
```

2.perform string shift

In [3]:
```
def stringShift(s, shift):
    total_shift = 0
    for sh in shift:
        if sh[0] == 0:
            total_shift -= sh[1]
        else:
            total_shift += sh[1]

    total_shift %= len(s)
    return s[-total_shift:] + s[:-total_shift]
s1 = "abc"
shift1 = [[0,1],[1,2]]
print(stringShift(s1, shift1))
s2 = "abcdefg"
shift2 = [[1,1],[1,1],[0,2],[1,3]]
print(stringShift(s2, shift2))
```

```
cab
efgabcd
```

7.max difference

In [5]:
```
def max_diff(num):
    num_str = str(num)
    max_num = int('9' + num_str[1:].replace('9','8'))
    min_num = int(('1' if num_str[0] != '1' else '0') + num_str[1:].replace(
    return max_num - min_num
print(max_diff(9))
```

```
8
```

3.binary matrix

In [8]:
```
def gene_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
rows = 2
cols = 2
binary_matrix = gene_row_sorted_binary_matrix(rows, cols)
print(binary_matrix)
```

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

4.firstunique

In [1]:
```
from collections import deque
class FirstUniq:
    def __init__(self, nums):
        self.queue = deque()
        self.count = {}
        for num in nums:
            self.add(num)
    def showFirstUniq(self):
        while self.queue and self.count[self.queue[0]] > 1:
            self.queue.popleft()
        if self.queue:
            return self.queue[0]
        return -1
    def add(self, value):
        if value in self.count:
            self.count[value] += 1
        else:
            self.count[value] = 1
            self.queue.append(value)
firstUniq = FirstUniq([2, 3, 5])
print(firstUnique.showFirstUniq())
firstUniq.add(5)
print(firstUniq.showFirstUnique())
firstUnique.add(2)
print(firstUnique.showFirstUnique())
firstUnique.add(3)
print(firstUnique.showFirstUnique())
```

```
2
2
3
-1
```

8.Check If a String Can Break Another String

In [4]:
```
def Break(s1, s2):
    sort_s1 = sorted(s1)
    sort_s2 = sorted(s2)
    s1_break_s2 = True
    s2_break_s1 = True
    for c1, c2 in zip(sort_s1, sort_s2):
        if c1 < c2:
            s1_break_s2 = False
        if c2 < c1:
            s2_break_s1 = False
    return s1_break_s2 or s2_break_s1
s1 = "abc"
s2 = "xya"
print(Break(s1, s2))
s1 = "abe"
s2 = "acd"
print(Break(s1, s2))
```

```
True
False
```

9.

In [7]:
```
from collections import defaultdict
def numberWays(hats):
    MOD = 10**9 + 7
    n = len(hats)
    hat_to_people = defaultdict(list)
    for person, hat_list in enumerate(hats):
        for hat in hat_list:
            hat_to_people[hat].append(person)
    memo = {}
    def dp(hat, mask):
        if mask == (1 << n) - 1:
            return 1
        if hat > 40:
            return 0
        if (hat, mask) in memo:
            return memo[(hat, mask)]
        ways = dp(hat + 1, mask)
        for person in hat_to_people[hat]:
            if not (mask & (1 << person)):
                ways += dp(hat + 1, mask | (1 << person))
                ways %= MOD
        memo[(hat, mask)] = ways
        return ways
    return dp(1, 0)
hats = [
    [3,4],
    [3,5,1],
    [5]
]
print(numberWays(hats))
```

```
3
```

10.destination city

In [8]:
```
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))
```

```
Sao Paulo
```

5. binary tree

In [10]:
```
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None
def isValidSequence(root, sequence):
    def dfs(node, index):
        if not node or index >= len(sequence) or node.val != int(sequence[ir
            return False
        if not node.left and not node.right and index == len(sequence) - 1:
            return True
        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(0)
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)
sequence = "001"
print(isValidSequence(root, sequence))
```

```
False
```

In [ ]: