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

main.py

```python
def count_good_triplets(nums):
    count = 0
    n = len(nums)
    for i in range(n):
        for j in range(i + 1, n):
            for k in range(j + 1, n):
                if nums[i] != nums[j] and nums[i] != nums[k] and nums[j] != nums[k]:
                    count += 1
    return count

nums = [4, 4, 2, 4, 3]
result = count_good_triplets(nums)
print(result)

```

input

```
3


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

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

main.py

```python
def min_fuel(roads, seats):
    n = len(roads) + 1
    adj_list = [[] for _ in range(n)]

    for a, b in roads:
        adj_list[a].append(b)
        adj_list[b].append(a)

    def dfs(node, parent):
        total_fuel = 0
        for neighbor in adj_list[node]:
            if neighbor != parent:
                total_fuel += dfs(neighbor, node)
        return max(1, total_fuel)

    return sum(dfs(city, -1) * seats for city in adj_list[0])

roads = [[0, 1], [0, 2], [0, 3]]
seats = 5
print(min_fuel(roads, seats))
```

input

```
30


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

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

main.py

```python
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
def count_partitions(s, k, minLength, memo, mod):
    if k == 0:
        return 1 if len(s) == 0 else 0
    if len(s) < k * minLength:
        return 0
    if (s, k) in memo:
        return memo[(s, k)]
    count = 0
    for i in range(minLength, len(s) + 1):
        if s[i - 1] in {'2', '3', '5', '7'} and not is_prime(int(s[:i])):
            count = (count + count_partitions(s[i:], k - 1, minLength, memo, mod)) % mod
    memo[(s, k)] = count
    return count
def beautiful_partitions(s, k, minLength):
    mod = 10**9 + 7
    memo = {}
    return count_partitions(s, k, minLength, memo, mod)
s = "23542185131"
k = 3
minLength = 2
print(beautiful_partitions(s, k, minLength))
```

input

```
0


...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_palindrome_substrings(s: str, k: int) -> int:
    def is_palindrome(sub: str) -> bool:
        return sub == sub[::-1]

    n = len(s)
    dp = [0] * n

    for i in range(n):
        for j in range(i, n):
            if j - i + 1 >= k and is_palindrome(s[i:j+1]):
                dp[j] = max(dp[j], dp[i-1] + 1 if i > 0 else 1)

    return max(dp)

s = "abaccdbbd"
k = 3
output = count_palindrome_substrings(s, k)
print(output)
```

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
from math import gcd

def count_subarrays_with_lcm_equal_to_k(nums, k):
    def lcm(a, b):
        return abs(a*b) // gcd(a, b)

    count = 0
    for i in range(len(nums)):
        current_lcm = 1
        for j in range(i, len(nums)):
            current_lcm = lcm(current_lcm, nums[j])
            if current_lcm == k:
                count += 1

    return count

nums = [3, 6, 2, 7, 1]
k = 6
print(count_subarrays_with_lcm_equal_to_k(nums, k))
```

input

```
4


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

Run Debug Stop Share Save {} Beautify Language Python 3

main.py

```python
def convert_temperature(celsius):
    kelvin = round(celsius + 273.15, 5)
    fahrenheit = round(celsius * 1.80 + 32.00, 5)
    return [kelvin, fahrenheit]

celsius = 36.50
output = convert_temperature(celsius)
print(output)
```

input

```
[309.65, 97.7]


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

```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def closest_values(root, target):
    if not root:
        return -1, -1
    min_val = float('-inf')
    max_val = float('inf')
    while root:
        if root.val == target:
            return target, target
        elif root.val < target:
            min_val = max(min_val, root.val)
            root = root.right
        else:
            max_val = min(max_val, root.val)
            root = root.left

    return min_val if min_val != float('-inf') else -1, max_val if max_val != float('inf') else -1
def find_closest_values(root, queries):
    result = []
    for query in queries:
        mini, maxi = closest_values(root, query)
        result.append([mini, maxi])
    return result
root = TreeNode(6)
root.left = TreeNode(2)
root.right = TreeNode(13)
root.left.left = TreeNode(1)
root.left.right = TreeNode(4)
root.right.left = TreeNode(9)
root.right.right = TreeNode(15)
root.right.right.left = TreeNode(14)
queries = [2, 5, 16]
print(find_closest_values(root, queries))
```

input

```
[[2, 2], [4, 6], [15, -1]]


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

```python
import heapq

def min_cost_to_buy_apple(n, roads, appleCost, k):
    graph = {i: [] for i in range(1, n+1)}
    for road in roads:
        a, b, cost = road
        graph[a].append((b, cost))
        graph[b].append((a, cost))

    def dijkstra(start):
        distances = [float('inf')] * (n + 1)
        distances[start] = 0
        heap = [(0, start)]

        while heap:
            curr_dist, node = heapq.heappop(heap)
            if curr_dist > distances[node]:
                continue
            for neighbor, edge_cost in graph[node]:
                new_dist = curr_dist + edge_cost
                if new_dist < distances[neighbor]:
                    distances[neighbor] = new_dist
                    heapq.heappush(heap, (new_dist, neighbor))
        return distances

    shortest_distances = {}
    for i in range(1, n + 1):
        shortest_distances[i] = dijkstra(i)

    min_costs = []
    for i in range(1, n + 1):
        min_cost = float('inf')
        for j in range(1, n + 1):
            if j != i:
                cost = shortest_distances[i][j] * k + appleCost[j - 1]
                min_cost = min(min_cost, cost)
        min_costs.append(min_cost)

    return min_costs

n = 4
roads = [[1,2,4],[2,3,2],[2,4,5],[3,4,1],[1,3,4]]
appleCost = [56,42,102,301]
k = 2
print(min_cost_to_buy_apple(n, roads, appleCost, k))
```

input

```
[50, 64, 46, 48]


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

main.py

```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def min_swaps_to_sort_levels(root):
    if not root:
        return 0
    queue = [(root, 0)]
    level_values = {}
    while queue:
        node, level = queue.pop(0)
        if level not in level_values:
            level_values[level] = []
        level_values[level].append(node.val)
        if node.left:
            queue.append((node.left, level + 1))
        if node.right:
            queue.append((node.right, level + 1))
    swaps_needed = 0
    for level in level_values.values():
        sorted_level = sorted(level)
        for i in range(len(level)):
            if level[i] != sorted_level[i]:
                swaps_needed += 1
    return swaps_needed
root = TreeNode(1)
```

input

```
5


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

main.py

```python
    while queue:
        node, level = queue.pop(0)
        if level not in level_values:
            level_values[level] = []
        level_values[level].append(node.val)
        if node.left:
            queue.append((node.left, level + 1))
        if node.right:
            queue.append((node.right, level + 1))
    swaps_needed = 0
    for level in level_values.values():
        sorted_level = sorted(level)
        for i in range(len(level)):
            if level[i] != sorted_level[i]:
                swaps_needed += 1
    return swaps_needed
root = TreeNode(1)
root.left = TreeNode(4)
root.right = TreeNode(3)
root.left.left = TreeNode(7)
root.left.right = TreeNode(6)
root.right.left = TreeNode(8)
root.right.right = TreeNode(5)
root.left.left.right = TreeNode(9)
root.right.right.left = TreeNode(10)

print(min_swaps_to_sort_levels(root))
```

input

```
5


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