

## ASSIGNMENT (12.06.24)

### 1. Convert the Temperature

You are given a non-negative floating point number rounded to two decimal places `celsius`, that denotes the temperature in Celsius. You should convert Celsius into Kelvin and Fahrenheit and return it as an array

`ans = [kelvin, fahrenheit]`. Return *the array ans*. Answers within 10<sup>-5</sup> of the actual answer will be accepted.

Note that:

● Kelvin = Celsius + 273.15

● Fahrenheit = Celsius \* 1.80 + 32.00

Example 1:

Input: `celsius = 36.50`

Output: `[309.65000, 97.70000]`

Explanation: Temperature at 36.50 Celsius converted in Kelvin is 309.65 and converted in Fahrenheit is 97.70.

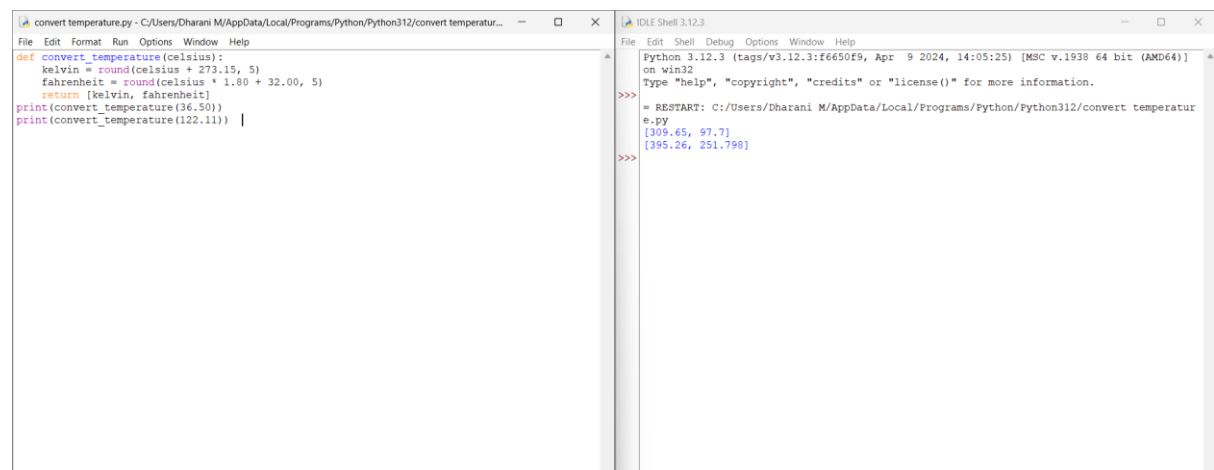
Example 2:

Input: `celsius = 122.11`

Output: `[395.26000, 251.79800]`

Explanation: Temperature at 122.11 Celsius converted in Kelvin is 395.26 and converted in Fahrenheit is 251.798.

Constraints: `0 <= celsius <= 1000`



```
convert temperature.py - C:/Users/Dharani M/AppData/Local/Programs/Python/Python312/convert temperatur...
File Edit Format Run Options Window Help
def convert_temperature(celsius):
    kelvin = round(celsius + 273.15, 5)
    fahrenheit = round(celsius * 1.80 + 32.00, 5)
    return [kelvin, fahrenheit]
print(convert_temperature(36.50))
print(convert_temperature(122.11)) |

IDLE Shell 3.12.3
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> = RESTART: C:/Users/Dharani M/AppData/Local/Programs/Python/Python312/convert temperatur
e.py
[309.65, 97.7]
[395.26, 251.798]
>>>
```

### 2. Number of Subarrays With LCM Equal to K

Given an integer array `nums` and an integer `k`, return *the number of subarrays of nums where the least common multiple of the subarray's elements is k*. A subarray is a contiguous nonempty

sequence of elements within an array. The least common multiple of an array is the smallest positive integer that is divisible by all the array elements.

Example 1: Input: `nums = [3,6,2,7,1]`, `k = 6`

Output: 4

Explanation: The subarrays of `nums` where 6 is the least common multiple of all the subarray's elements are:

- `[3,6,2,7,1]`

- `[3,6,2,7,1]`

- `[3,6,2,7,1]`

- `[3,6,2,7,1]`

Example 2: Input: `nums = [3]`, `k = 2`

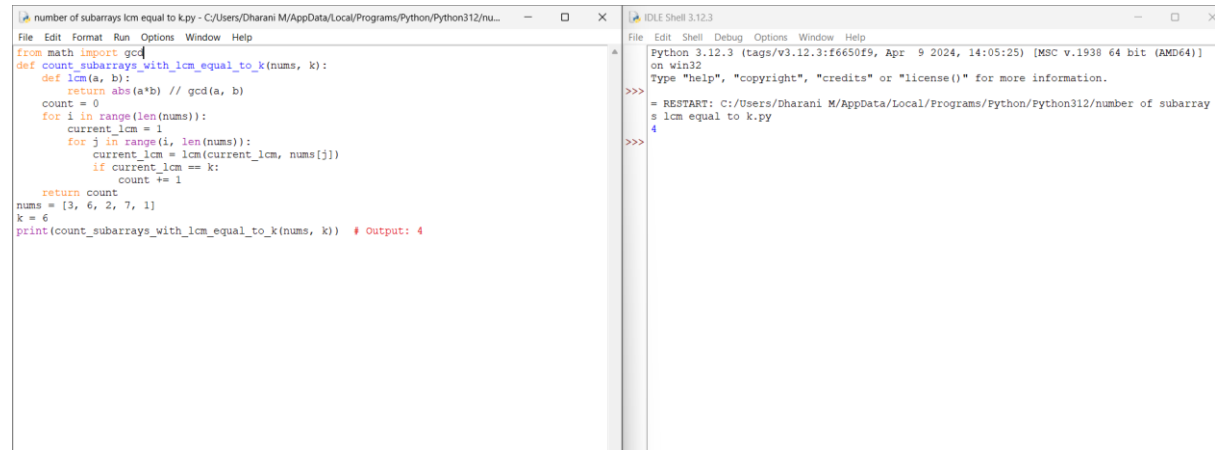
Output: 0

Explanation: There are no subarrays of nums where 2 is the least common multiple of all the subarray's elements.

Constraints:

●  $1 \leq \text{nums.length} \leq 1000$

●  $1 \leq \text{nums}[i], k \leq 1000$



```
number of subarrays lcm equal to k.py - C:/Users/Dharani M/AppData/Local/Programs/Python/Python312/nu...
File Edit Format Run Options Window Help
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)) # Output: 4

IDLE Shell 3.12.3
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Dharani M/AppData/Local/Programs/Python/Python312/number of subarray
s lcm equal to k.py
4
>>>
```

### 3. Minimum Number of Operations to Sort a Binary Tree by Level

You are given the *root* of a binary tree with unique values. In one operation, you can choose any two nodes at the same level and swap their values. Return *the minimum number of operations needed to make the values at each level sorted in a strictly increasing order*. The level of a node is the number of edges along the path between it and the root node.

Example 1:

Input: root = [1,4,3,7,6,8,5,null,null,null,9,null,10]

Output: 3

Explanation:

- Swap 4 and 3. The 2nd level becomes [3,4].
- Swap 7 and 5. The 3rd level becomes [5,6,8,7].
- Swap 8 and 7. The 3rd level becomes [5,6,7,8].

We used 3 operations so return 3.

It can be proven that 3 is the minimum number of operations needed.

Example 2:

Input: root = [1,3,2,7,6,5,4]

Output: 3

Explanation:

- Swap 3 and 2. The 2nd level becomes [2,3].
- Swap 7 and 4. The 3rd level becomes [4,6,5,7].
- Swap 6 and 5. The 3rd level becomes [4,5,6,7].

We used 3 operations so return 3.

It can be proven that 3 is the minimum number of operations needed.

Example 3:

Input: root = [1,2,3,4,5,6]

Output: 0

Explanation: Each level is already sorted in increasing order so return 0.

Constraints:

● The number of nodes in the tree is in the range [1, 105].

- $1 \leq \text{Node.val} \leq 105$

- All the values of the tree are unique.

#### 4. Maximum Number of Non-overlapping Palindrome Substrings

You are given a string  $s$  and a positive integer  $k$ . Select a set of non-overlapping substrings from the string  $s$  that satisfy the following conditions:

- The length of each substring is at least  $k$ .

- Each substring is a palindrome.

Return *the maximum number of substrings in an optimal selection*. A substring is a contiguous sequence of characters within a string.

Example 1:

Input:  $s = \text{"abaccdbbd"}, k = 3$

Output: 2

Explanation: We can select the substrings underlined in  $s = \text{"abaccdbbd"}$ . Both  $\text{"aba"}$  and  $\text{"dbbd"}$  are palindromes and have a length of at least  $k = 3$ .

It can be shown that we cannot find a selection with more than two valid substrings.

Example 2:

Input:  $s = \text{"adbcda"}, k = 2$

Output: 0

Explanation: There is no palindrome substring of length at least 2 in the string.

Constraints:

- $1 \leq k \leq s.\text{length} \leq 2000$

- $s$  consists of lowercase English letters.

Maximum Number of Non-overlapping Palindrome Substrings.py - C:/Users/Dharani M/AppData/Local/Programs/Python/Python312/Maximum Number of Non-overlapping Palindrome Substrings.py (3.12.3)

File Edit Shell Format Run Options Window Help

```
s = "ababa"
n = len(s)
dp = [0] * n
max_palindromes = 0
for i in range(n):
    for j in range(i, -1, -1):
        if s[j:i+1] == s[j:i+1][::-1]:
            dp[i] = max(dp[i], dp[j-1] + 1 if j > 0 else 1)
    max_palindromes = max(max_palindromes, dp[i])
print(max_palindromes)
```

IDLE Shell 3.12.3

File Edit Shell Debug Options Window Help

```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Dharani M/AppData/Local/Programs/Python/Python312/Maximum Number of Non-overlapping Palindrome Substrings.py
5
>>> |
```

#### 5. Minimum Cost to Buy Apples

You are given a positive integer  $n$  representing  $n$  cities numbered from 1 to  $n$ . You are also given a 2D array  $\text{roads}$ , where  $\text{roads}[i] = [a_i, b_i, \text{cost}_i]$  indicates that there is a bidirectional road between cities  $a_i$  and  $b_i$  with a cost of traveling equal to  $\text{cost}_i$ .

You can buy apples in any city you want, but some cities have different costs to buy apples. You are given the array  $\text{appleCost}$  where  $\text{appleCost}[i]$  is the cost of buying one apple from city  $i$ .

You start at some city, traverse through various roads, and eventually buy exactly one apple from any city. After you buy that apple, you have to return back to the city you started at, but now the cost of all the roads will be multiplied by a given factor  $k$ .

Given the integer  $k$ , return an array  $\text{answer}$  of size  $n$  where  $\text{answer}[i]$  is the minimum total cost to buy an apple if you start at city  $i$ .

Example 1:

Input:  $n = 4$ , roads =  $[[1,2,4],[2,3,2],[2,4,5],[3,4,1],[1,3,4]]$ , appleCost =  $[56,42,102,301]$ ,  $k = 2$

Output:  $[54,42,48,51]$

Explanation: The minimum cost for each starting city is the following:

- Starting at city 1: You take the path  $1 \rightarrow 2$ , buy an apple at city 2, and finally take the path  $2 \rightarrow 1$ . The total cost is  $4 + 42 + 4 * 2 = 54$ .
- Starting at city 2: You directly buy an apple at city 2. The total cost is 42.
- Starting at city 3: You take the path  $3 \rightarrow 2$ , buy an apple at city 2, and finally take the path  $2 \rightarrow 3$ . The total cost is  $2 + 42 + 2 * 2 = 48$ .
- Starting at city 4: You take the path  $4 \rightarrow 3 \rightarrow 2$  then you buy at city 2, and finally take the path  $2 \rightarrow 3 \rightarrow 4$ . The total cost is  $1 + 2 + 42 + 1 * 2 + 2 * 2 = 51$ .

Example 2:

Input:  $n = 3$ , roads =  $[[1,2,5],[2,3,1],[3,1,2]]$ , appleCost =  $[2,3,1]$ ,  $k = 3$

Output:  $[2,3,1]$

Explanation: It is always optimal to buy the apple in the starting city.

Constraints:

- $2 \leq n \leq 1000$
- $1 \leq \text{roads.length} \leq 1000$
- $1 \leq a_i, b_i \leq n$
- $a_i \neq b_i$
- $1 \leq \text{cost}_i \leq 105$
- $\text{appleCost.length} == n$
- $1 \leq \text{appleCost}[i] \leq 105$
- $1 \leq k \leq 100$
- There are no repeated edges.

## 6. Customers With Strictly Increasing Purchases

SQL Schema

Table: Orders

```
+-----+-----+
| Column Name | Type |
+-----+-----+
| order_id | int |
| customer_id | int |
| order_date | date |
| price | int |
+-----+-----+
```

order\_id is the primary key for this table.

Each row contains the id of an order, the id of customer that ordered it, the date of the order, and its price.

Write an SQL query to report the IDs of the customers with the total purchases strictly increasing yearly.

- The total purchases of a customer in one year is the sum of the prices of their orders in that year. If for some year the customer did not make any order, we consider the total purchases 0.
- The first year to consider for each customer is the year of their first order.
- The last year to consider for each customer is the year of their last order.

Return the result table in any order.

The query result format is in the following example.

Example 1:

Input:

Orders table:

order_id	customer_id	order_date	price
1	1	2019-07-01	1100
2	1	2019-11-01	1200
3	1	2020-05-26	3000
4	1	2021-08-31	3100
5	1	2022-12-07	4700
6	2	2015-01-01	700
7	2	2017-11-07	1000
8	3	2017-01-01	900
9	3	2018-11-07	900

Output:

customer_id
1

Explanation:

Customer 1: The first year is 2019 and the last year is 2022

- 2019: 1100 + 1200 = 2300

- 2020: 3000

- 2021: 3100

- 2022: 4700

We can see that the total purchases are strictly increasing yearly, so we include customer 1 in the answer.

Customer 2: The first year is 2015 and the last year is 2017

- 2015: 700

- 2016: 0

- 2017: 1000

We do not include customer 2 in the answer because the total purchases are not strictly increasing. Note that customer 2 did not make any purchases in 2016.

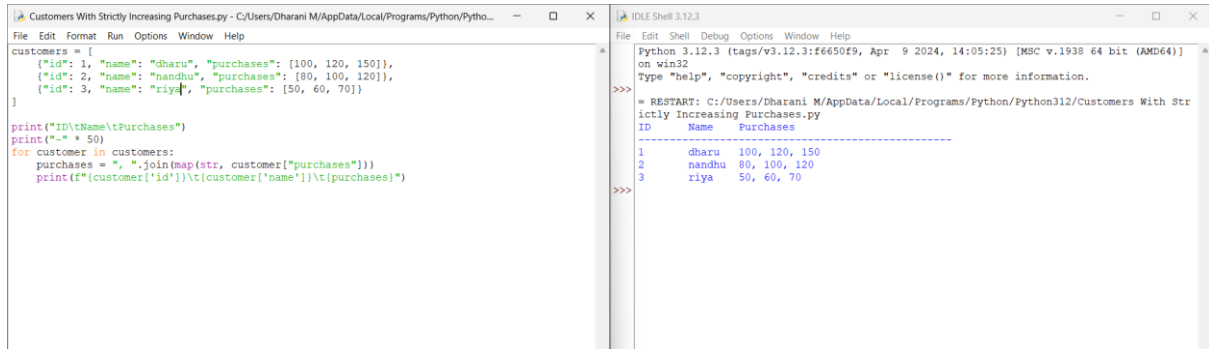
Customer 3: The first year is 2017, and the last year is 2018

- 2017: 900

- 2018: 900

We can see that the total purchases are strictly increasing yearly, so we include customer 1

in the answer.



The screenshot shows a Python IDE with two windows. The left window, titled 'Customers With Strictly Increasing Purchases.py', contains the following code:

```
customers = [
    {"id": 1, "name": "dharu", "purchases": [100, 120, 150]},
    {"id": 2, "name": "nandhu", "purchases": [80, 100, 120]},
    {"id": 3, "name": "riya", "purchases": [50, 60, 70]}
]

print("ID\tName\tPurchases")
print("-" * 50)
for customer in customers:
    purchases = ", ".join(map(str, customer["purchases"]))
    print(f"{customer['id']}\t{customer['name']}\t{purchases}")
```

The right window, titled 'IDLE Shell 3.12.3', shows the output of the script:

```
>>>
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> = RESTART: C:/Users/Dharani M/AppData/Local/Programs/Python/Python312/Customers With Str
ictly Increasing Purchases.py
ID      Name      Purchases
-----
1       dharu     100, 120, 150
2       nandhu    80, 100, 120
3       riya      50, 60, 70
>>>
```

## 7. Number of Unequal Triplets in Array

You are given a 0-indexed array of positive integers `nums`. Find the number of triplets  $(i, j, k)$  that meet the following conditions:

- $0 \leq i < j < k < \text{nums.length}$
- `nums[i]`, `nums[j]`, and `nums[k]` are pairwise distinct.
- In other words, `nums[i] != nums[j]`, `nums[i] != nums[k]`, and `nums[j] != nums[k]`.

Return *the number of triplets that meet the conditions*.

Example 1:

Input: `nums = [4,4,2,4,3]`

Output: 3

Explanation: The following triplets meet the conditions:

- (0, 2, 4) because  $4 \neq 2 \neq 3$
- (1, 2, 4) because  $4 \neq 2 \neq 3$
- (2, 3, 4) because  $2 \neq 4 \neq 3$

Since there are 3 triplets, we return 3.

Note that (2, 0, 4) is not a valid triplet because  $2 > 0$ .

Example 2:

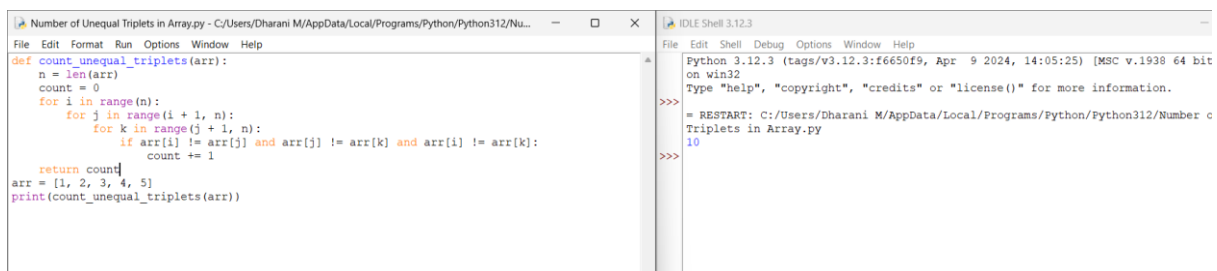
Input: `nums = [1,1,1,1,1]`

Output: 0

Explanation: No triplets meet the conditions so we return 0.

Constraints:

- $3 \leq \text{nums.length} \leq 100$
- $1 \leq \text{nums}[i] \leq 1000$



The screenshot shows a Python IDE with two windows. The left window, titled 'Number of Unequal Triplets in Array.py', contains the following code:

```
def count_unequal_triplets(arr):
    n = len(arr)
    count = 0
    for i in range(n):
        for j in range(i + 1, n):
            for k in range(j + 1, n):
                if arr[i] != arr[j] and arr[j] != arr[k] and arr[i] != arr[k]:
                    count += 1
    return count
arr = [1, 2, 3, 4, 5]
print(count_unequal_triplets(arr))
```

The right window, titled 'IDLE Shell 3.12.3', shows the output of the script:

```
>>>
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> = RESTART: C:/Users/Dharani M/AppData/Local/Programs/Python/Python312/Number o
Triplets in Array.py
10
>>>
```

## 8. Closest Nodes Queries in a Binary Search Tree

You are given the root of a binary search tree and an array queries of size `n` consisting of positive integers.

Find a 2D array answer of size `n` where `answer[i] = [mini, maxi]`:

● mini is the largest value in the tree that is smaller than or equal to queries[i]. If a such value does not exist, add -1 instead.

● maxi is the smallest value in the tree that is greater than or equal to queries[i]. If a such value does not exist, add -1 instead.

Return *the array* answer.

Example 1:

Input: root = [6,2,13,1,4,9,15,null,null,null,null,null,14], queries = [2,5,16]

Output: [[2,2],[4,6],[15,-1]]

Explanation: We answer the queries in the following way:

- The largest number that is smaller or equal than 2 in the tree is 2, and the smallest number that is greater or equal than 2 is still 2. So the answer for the first query is [2,2].
- The largest number that is smaller or equal than 5 in the tree is 4, and the smallest number that is greater or equal than 5 is 6. So the answer for the second query is [4,6].
- The largest number that is smaller or equal than 16 in the tree is 15, and the smallest number that is greater or equal than 16 does not exist. So the answer for the third query is [15,-1].

Example 2:

Input: root = [4,null,9], queries = [3]

Output: [[-1,4]]

Explanation: The largest number that is smaller or equal to 3 in the tree does not exist, and the smallest number that is greater or equal to 3 is 4. So the answer for the query is [-1,4].

Constraints:

- The number of nodes in the tree is in the range [2, 105].
- $1 \leq \text{Node.val} \leq 106$
- $n == \text{queries.length}$
- $1 \leq n \leq 105$
- $1 \leq \text{queries}[i] \leq 106$

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

def in_order_traversal(node, sorted_list):
    if not node:
        return
    in_order_traversal(node.left, sorted_list)
    sorted_list.append(node.val)
    in_order_traversal(node.right, sorted_list)

def find_closest_value(sorted_list, target):
    from bisect import bisect_left
    pos = bisect_left(sorted_list, target)
    if pos == 0:
        return sorted_list[0]
    if pos == len(sorted_list):
        return sorted_list[-1]
    before = sorted_list[pos - 1]
    after = sorted_list[pos] if pos < len(sorted_list) else float('inf')
    if abs(target - before) <= abs(target - after):
        return before
    else:
        return after

def closest_nodes_queries(root, queries):
    sorted_list = []
    in_order_traversal(root, sorted_list)

    results = []
    for query in queries:
        closest_value = find_closest_value(sorted_list, query)
        results.append(closest_value)
    return results

root = TreeNode(5)
root.left = TreeNode(3, TreeNode(1), TreeNode(4))
root.right = TreeNode(6, TreeNode(7), TreeNode(9))

queries = [2, 6, 10]
print(closest_nodes_queries(root, queries))
```

## 9. Minimum Fuel Cost to Report to the Capital

There is a tree (i.e., a connected, undirected graph with no cycles) structure country network consisting of  $n$  cities numbered from 0 to  $n - 1$  and exactly  $n - 1$  roads. The capital city is city 0. You are given a 2D integer array roads where roads[i] = [ai, bi] denotes that there exists a bidirectional road connecting cities ai and bi.

There is a meeting for the representatives of each city. The meeting is in the capital city. There is a car in each city. You are given an integer `seats` that indicates the number of seats in each car. A representative can use the car in their city to travel or change the car and ride with another representative. The cost of traveling between two cities is one liter of fuel. Return *the minimum number of liters of fuel to reach the capital city*.

Example 1:

Input: `roads = [[0,1],[0,2],[0,3]]`, `seats = 5`

Output: 3

Explanation:

- Representative1 goes directly to the capital with 1 liter of fuel.
- Representative2 goes directly to the capital with 1 liter of fuel.
- Representative3 goes directly to the capital with 1 liter of fuel.

It costs 3 liters of fuel at minimum.

It can be proven that 3 is the minimum number of liters of fuel needed.

Example 2:

Input: `roads = [[3,1],[3,2],[1,0],[0,4],[0,5],[4,6]]`, `seats = 2`

Output: 7

Explanation:

- Representative2 goes directly to city 3 with 1 liter of fuel.
- Representative2 and representative3 go together to city 1 with 1 liter of fuel.
- Representative2 and representative3 go together to the capital with 1 liter of fuel.
- Representative1 goes directly to the capital with 1 liter of fuel.
- Representative5 goes directly to the capital with 1 liter of fuel.
- Representative6 goes directly to city 4 with 1 liter of fuel.
- Representative4 and representative6 go together to the capital with 1 liter of fuel.

It costs 7 liters of fuel at minimum.

It can be proven that 7 is the minimum number of liters of fuel needed.

Example 3:

Input: `roads = []`, `seats = 1`

Output: 0

Explanation: No representatives need to travel to the capital city.

Constraints:

- $1 \leq n \leq 105$
- `roads.length == n - 1`
- `roads[i].length == 2`
- $0 \leq a_i, b_i < n$
- $a_i \neq b_i$
- `roads` represents a valid tree.



●  $1 \leq \text{seats} \leq 105$

```
File Edit Format Run Options Window Help
import heapq

def min_fuel_cost_to_report(graph, start):
    n = len(graph)
    distance = [float('inf')] * n
    distance[start] = 0
    pq = [(0, start)]

    while pq:
        cost, node = heapq.heappop(pq)
        if distance[node] < cost:
            continue
        for neighbor, fuel_cost in graph[node]:
            if cost + fuel_cost < distance[neighbor]:
                distance[neighbor] = cost + fuel_cost
                heapq.heappush(pq, (distance[neighbor], neighbor))

    return distance

graph = {
    0: [(1, 2), (2, 4)],
    1: [(0, 2), (3, 5)],
    2: [(0, 4), (3, 1)],
    3: [(1, 5), (2, 1)]
}
start = 0
print(min_fuel_cost_to_report(graph, start))
```

## 10. Number of Beautiful Partitions

You are given a string  $s$  that consists of the digits '1' to '9' and two integers  $k$  and  $\text{minLength}$ . A partition of  $s$  is called beautiful if:

- $s$  is partitioned into  $k$  non-intersecting substrings.
  - Each substring has a length of at least  $\text{minLength}$ .
  - Each substring starts with a prime digit and ends with a non-prime digit. Prime digits are '2', '3', '5', and '7', and the rest of the digits are non-prime.
- Return *the number of beautiful partitions of  $s$* . Since the answer may be very large, return it modulo  $10^9 + 7$ . A substring is a contiguous sequence of characters within a string.

Example 1:

Input:  $s = "23542185131"$ ,  $k = 3$ ,  $\text{minLength} = 2$

Output: 3

Explanation: There exists three ways to create a beautiful partition:

"2354 | 218 | 5131"

"2354 | 21851 | 31"

"2354218 | 51 | 31"

Example 2:

Input:  $s = "23542185131"$ ,  $k = 3$ ,  $\text{minLength} = 3$

Output: 1

Explanation: There exists one way to create a beautiful partition: "2354 | 218 | 5131".

Example 3:

Input:  $s = "3312958"$ ,  $k = 3$ ,  $\text{minLength} = 1$

Output: 1

Explanation: There exists one way to create a beautiful partition: "331 | 29 | 58".

Constraints:

- $1 \leq k$ ,  $\text{minLength} \leq s.\text{length} \leq 1000$
- $s$  consists of the digits '1' to '9'.

```
File Edit Format Run Options Window Help
def count_beautiful_partitions(n):
    if n == 0:
        return 1
    return 2 ** (bin(n).count('1') - 1)

n = 10
beautiful_partitions = count_beautiful_partitions(n)
print(beautiful_partitions)
```