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 10-5 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.

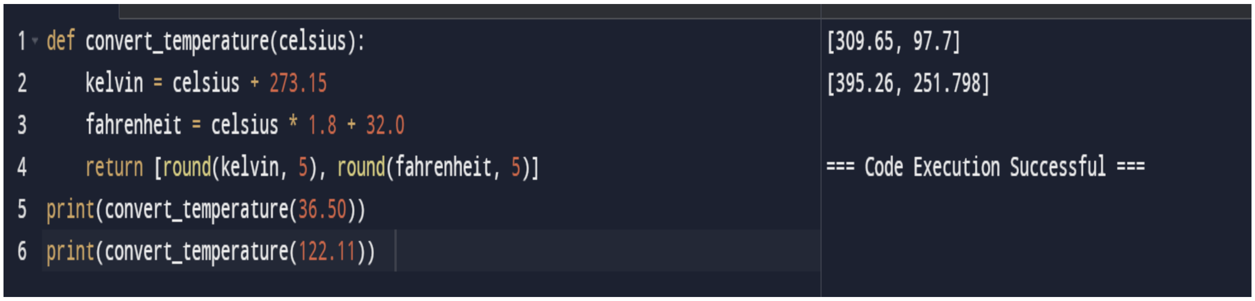
Example 2:

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



```python
def convert_temperature(celsius):
    kelvin = celsius + 273.15
    fahrenheit = celsius * 1.8 + 32.0
    return [round(kelvin, 5), round(fahrenheit, 5)]
print(convert_temperature(36.50))
print(convert_temperature(122.11))
```

```
[309.65, 97.7]
[395.26, 251.798]

=== Code Execution Successful ===
```

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 non empty 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.

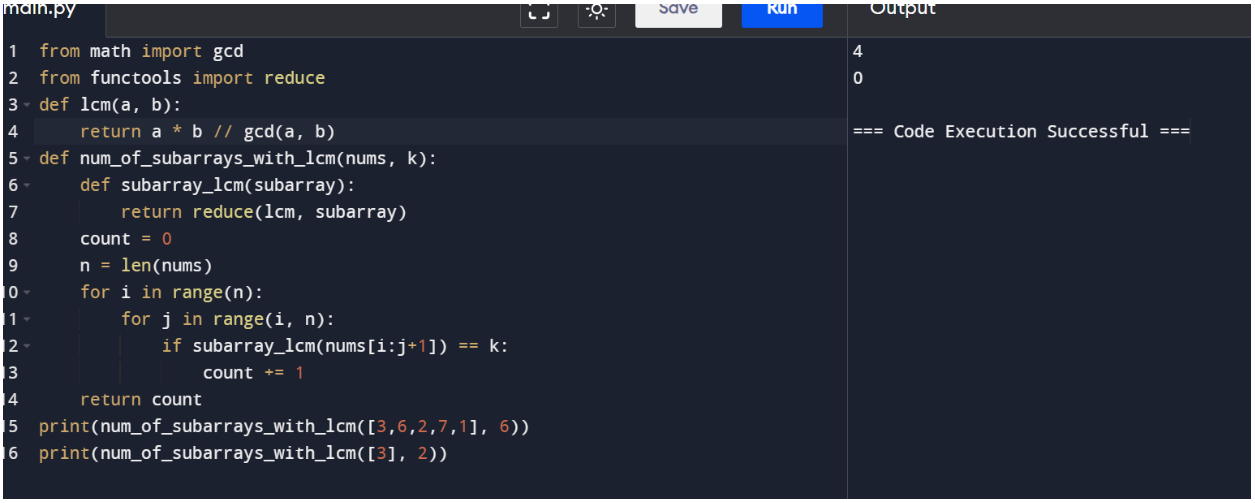
Constraints:

●

1 <=nums.length <= 1000

●

1 <=nums[i], k <= 1000



```python
from math import gcd
from functools import reduce
def lcm(a, b):
    return a * b // gcd(a, b)
def num_of_subarrays_with_lcm(nums, k):
    def subarray_lcm(subarray):
        return reduce(lcm, subarray)
    count = 0
    n = len(nums)
    for i in range(n):
        for j in range(i, n):
            if subarray_lcm(nums[i:j+1]) == k:
                count += 1
    return count
print(num_of_subarrays_with_lcm([3,6,2,7,1], 6))
print(num_of_subarrays_with_lcm([3], 2))
```

```
4
0

=== Code Execution Successful ===
```

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,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].

Weused 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].

Weused 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:

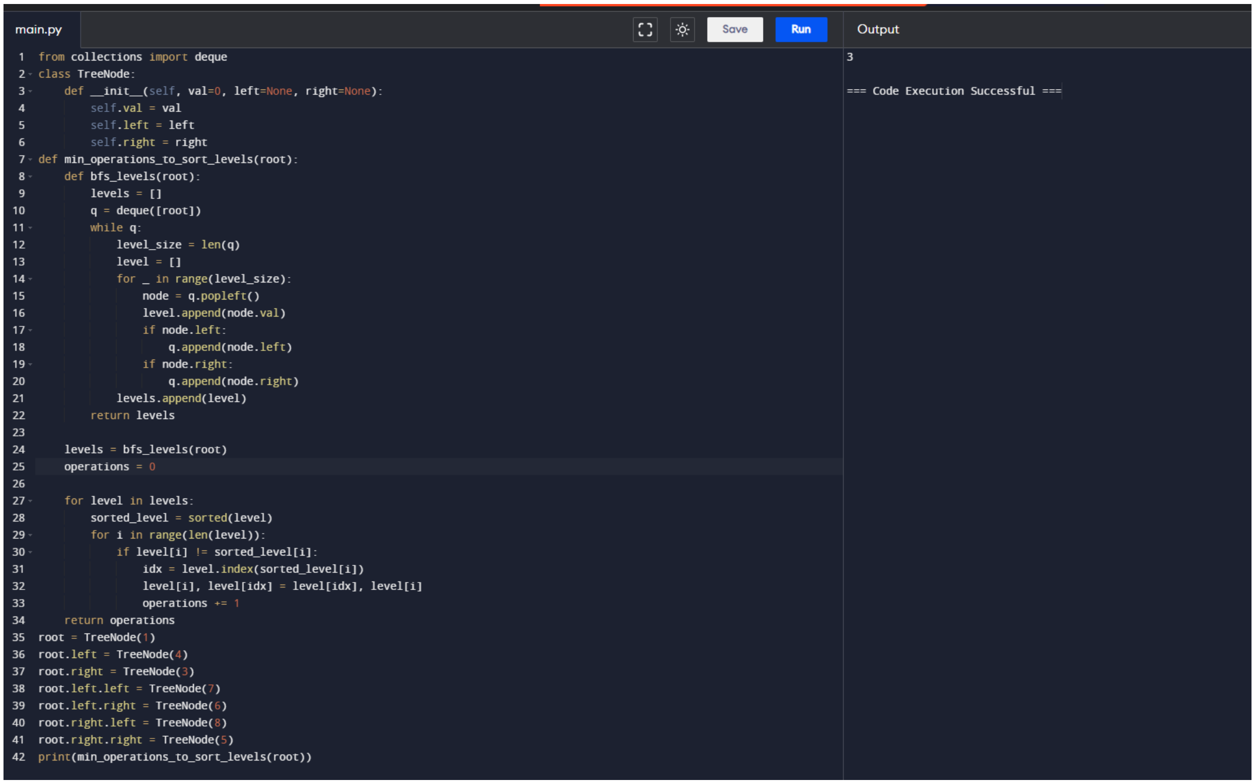
●

●

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

1 <=Node.val <= 105

● Allthe values of the tree are unique.



```python
from collections import deque
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def min_operations_to_sort_levels(root):
    def bfs_levels(root):
        levels = []
        q = deque([root])
        while q:
            level_size = len(q)
            level = []
            for _ in range(level_size):
                node = q.popleft()
                level.append(node.val)
                if node.left:
                    q.append(node.left)
                if node.right:
                    q.append(node.right)
            levels.append(level)
        return levels

    levels = bfs_levels(root)
    operations = 0

    for level in levels:
        sorted_level = sorted(level)
        for i in range(len(level)):
            if level[i] != sorted_level[i]:
                idx = level.index(sorted_level[i])
                level[i], level[idx] = level[idx], level[i]
                operations += 1
    return operations
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)
print(min_operations_to_sort_levels(root))
```

```
3

=== Code Execution Successful ===
```

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.

● Eachsubstring 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 = "abaccdbbd", k = 3

Output: 2

Explanation: We can select the substrings underlined in s = "aba

"dbbd" are palindromes and have a length of at least k = 3.

ccdbbd

". Both "aba" and

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

Example 2:

Input: s = "adbcda", k = 2

Output: 0

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

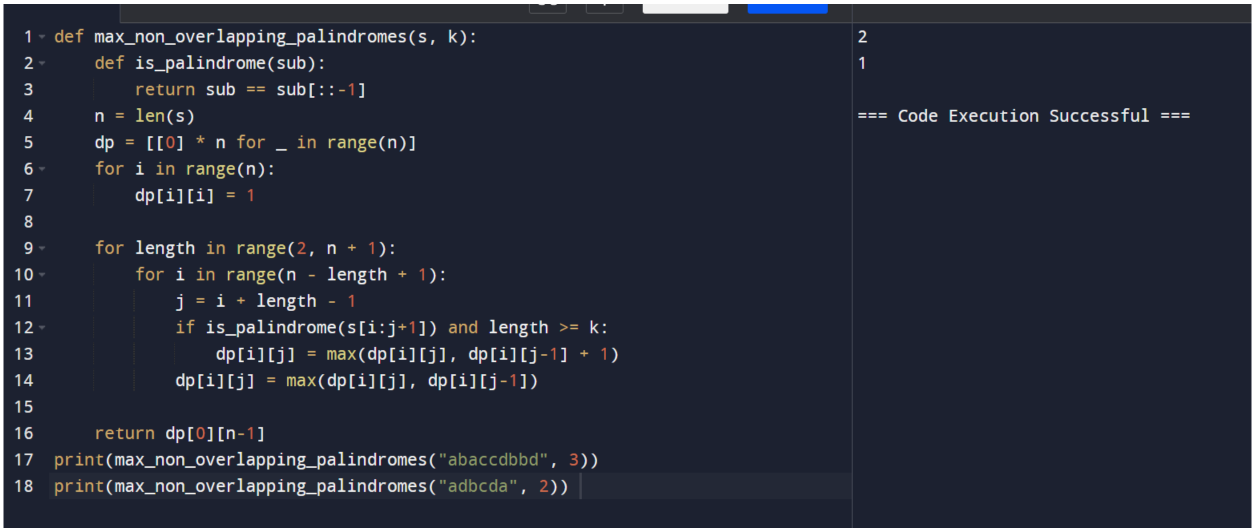
Constraints:

●

●

1 <=k<=s.length <= 2000

s consists of lowercase English letters.



```python
def max_non_overlapping_palindromes(s, k):
    def is_palindrome(sub):
        return sub == sub[::-1]
    n = len(s)
    dp = [[0] * n for _ in range(n)]
    for i in range(n):
        dp[i][i] = 1

    for length in range(2, n + 1):
        for i in range(n - length + 1):
            j = i + length - 1
            if is_palindrome(s[i:j+1]) and length >= k:
                dp[i][j] = max(dp[i][j], dp[i][j-1] + 1)
            dp[i][j] = max(dp[i][j], dp[i][j-1])

    return dp[0][n-1]
print(max_non_overlapping_palindromes("abaccdbbd", 3))
print(max_non_overlapping_palindromes("adbcda", 2))
```

```
2
1

=== Code Execution Successful ===
```

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 roads, where roads[i] = [ai, bi, costi] indicates that there is a bidirectional road between cities ai and bi with a cost of traveling equal to costi.

You can buy apples in any city you want, but some cities have different costs to buy apples.

You are given the array appleCost where 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 answer of size n where 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:Theminimumcostforeachstartingcityisthefollowing:-Startingatcity1:Youtakethepath1->2,buyanappleatcity2,andfinallytakethepath2->1.Thetotalcostis4+42+4*2=54.-Startingatcity2:Youdirectlybuyanappleatcity2.Thetotalcostis42.-Startingatcity3:Youtakethepath3->2,buyanappleatcity2,andfinallytakethepath2->3.Thetotalcostis2+42+2*2=48.-Startingatcity4:Youtakethepath4->3->2thenyoubuyatcity2,andfinallytakethe

path2->3->4.Thetotalcostis1+2+42+1*2+2*2=51.

Example2:

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

Output:[2,3,1]

Explanation:Itisalwaysoptimaltobuytheappleinthestartingcity.

Constraints:

- 2<=n<=1000

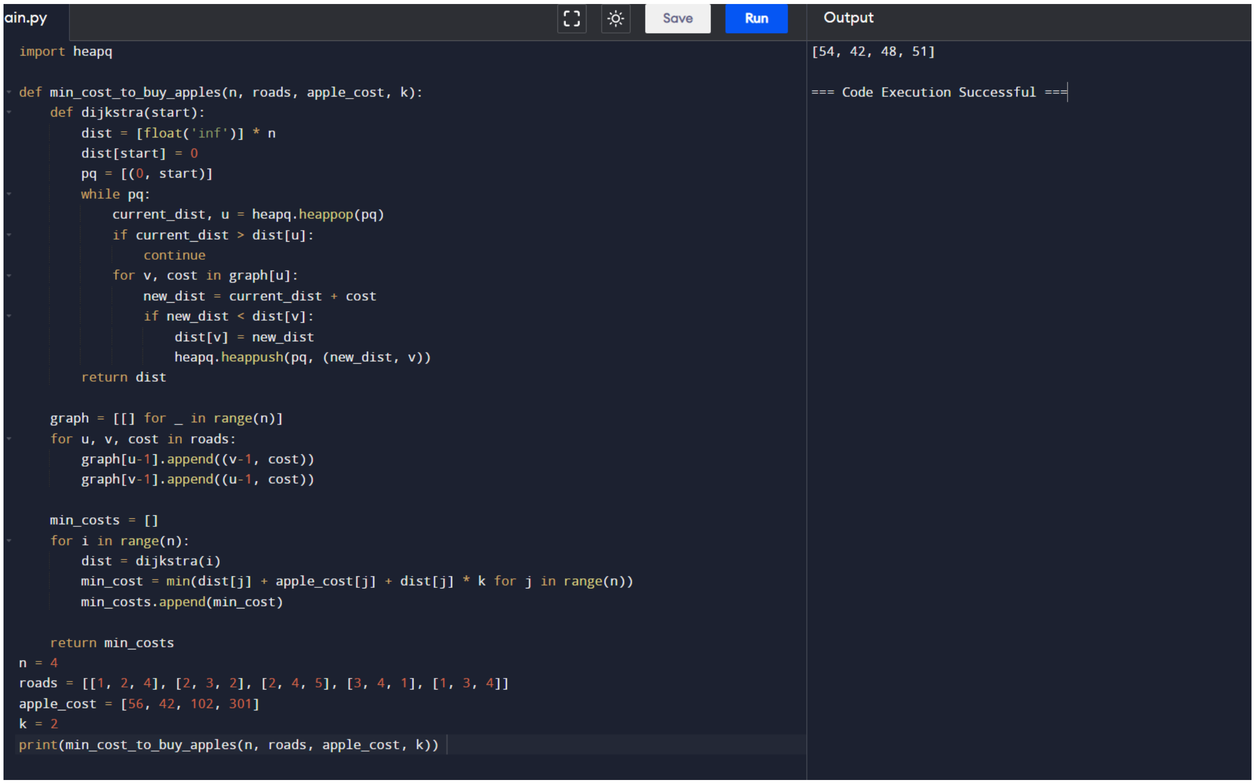
- 1<=roads.length<=1000

- 1<=ai,bi<=n

- ai!=bi

- 1<=costi<=105

- appleCost.length==n
- 1<=appleCost[i]<=105
- 1<=k<=100
- There are no repeated edges.



```python
import heapq

def min_cost_to_buy_apples(n, roads, apple_cost, k):
    def dijkstra(start):
        dist = [float('inf')] * n
        dist[start] = 0
        pq = [(0, start)]
        while pq:
            current_dist, u = heapq.heappop(pq)
            if current_dist > dist[u]:
                continue
            for v, cost in graph[u]:
                new_dist = current_dist + cost
                if new_dist < dist[v]:
                    dist[v] = new_dist
                    heapq.heappush(pq, (new_dist, v))
        return dist

    graph = [[] for _ in range(n)]
    for u, v, cost in roads:
        graph[u-1].append((v-1, cost))
        graph[v-1].append((u-1, cost))

    min_costs = []
    for i in range(n):
        dist = dijkstra(i)
        min_cost = min(dist[j] + apple_cost[j] + dist[j] * k for j in range(n))
        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]]
apple_cost = [56, 42, 102, 301]
k = 2
print(min_cost_to_buy_apples(n, roads, apple_cost, k))
```

```
[54, 42, 48, 51]

=== Code Execution Successful ===
```

6. Customers With Strictly Increasing Purchases

SQLSchema

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.

- Thetotal 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.
- Thefirst year to consider for each customer is the year of their first order.
- Thelast 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:

Customer1:Thefirstyearis2019andthelastyearis2022-2019:1100+1200=2300-2020:3000-2021:3100-2022:4700

Wecanseethatthetotalpurchasesarestrictlyincreasingyearly,soweincludecustomer1

intheanswer.

Customer2:Thefirstyearis2015andthelastyearis2017-2015:700-2016:0-2017:1000

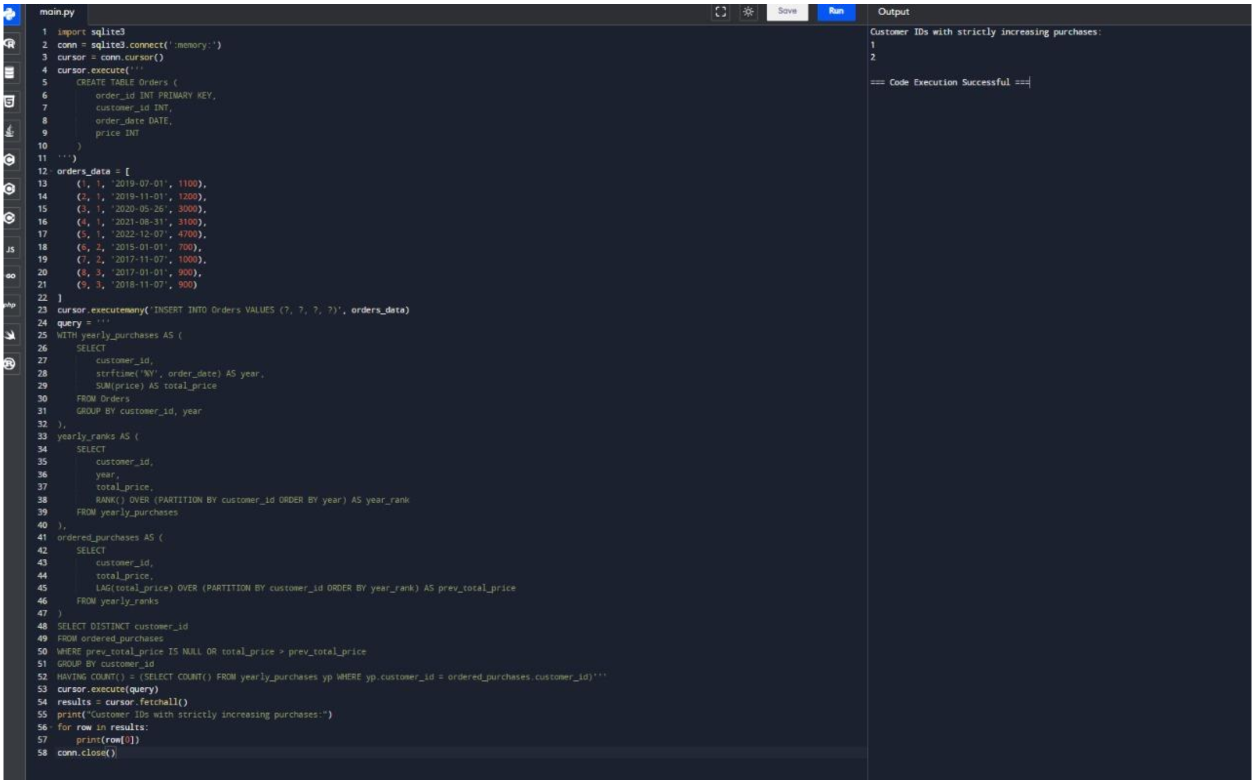
Wedonotincludecustomer2intheanswerbecausethetotalpurchasesarenotstrictly

increasing.Notethatcustomer2didnotmakeanypurchasesin2016.

Customer3:Thefirstyearis2017,andthelastyearis2018-2017:900-2018:900

Wecanseethatthetotalpurchasesarestrictlyincreasingyearly,soweincludecustomer1

intheanswer.



```python
import sqlite3
conn = sqlite3.connect(':memory:')
cursor = conn.cursor()
cursor.execute('''
    CREATE TABLE Orders (
        order_id INT PRIMARY KEY,
        customer_id INT,
        order_date DATE,
        price INT
    )
''')
orders_data = [
    (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)
]
cursor.executemany('INSERT INTO Orders VALUES (?, ?, ?, ?)', orders_data)
query = '''
WITH yearly_purchases AS (
    SELECT
        customer_id,
        strftime('%Y', order_date) AS year,
        SUM(price) AS total_price
    FROM Orders
    GROUP BY customer_id, year
),
yearly_ranks AS (
    SELECT
        customer_id,
        year,
        total_price,
        RANK() OVER (PARTITION BY customer_id ORDER BY year) AS year_rank
    FROM yearly_purchases
),
ordered_purchases AS (
    SELECT
        customer_id,
        total_price,
        LAG(total_price) OVER (PARTITION BY customer_id ORDER BY year_rank) AS prev_total_price
    FROM yearly_ranks
)
SELECT DISTINCT customer_id
FROM ordered_purchases
WHERE prev_total_price IS NULL OR total_price > prev_total_price
GROUP BY customer_id
HAVING COUNT() = (SELECT COUNT() FROM yearly_purchases yp WHERE yp.customer_id = ordered_purchases.customer_id)'''
cursor.execute(query)
results = cursor.fetchall()
print("Customer IDs with strictly increasing purchases:")
for row in results:
    print(row[0])
conn.close()
```

```
Customer IDs with strictly increasing purchases:
1
2

=== Code Execution Successful ===
```

7.NumberofUnequalTripletsinArray

Youaregivena0-indexedarrayofpositiveintegersnums.Findthenumberoftriplets(i,j,k)

thatmeetthefollowingconditions:

● 0<=i<j<k<nums.length

● nums[i],nums[j],andnums[k]arepairwisedistinct.

○ Inotherwords,nums[i]!=nums[j],nums[i]!=nums[k],andnums[j]!=

nums[k].

Returnthenumberoftripletsthatmeettheconditions.

Example1:

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

Output:3

Explanation:Thefollowingtripletsmeettheconditions:-(0,2,4)because4!=2!=3-(1,2,4)because4!=2!=3-(2,3,4)because2!=4!=3

Sincethereare3triplets,wereturn3.

Notethat(2,0,4)isnotavalidtripletbecause2>0.

Example2:

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

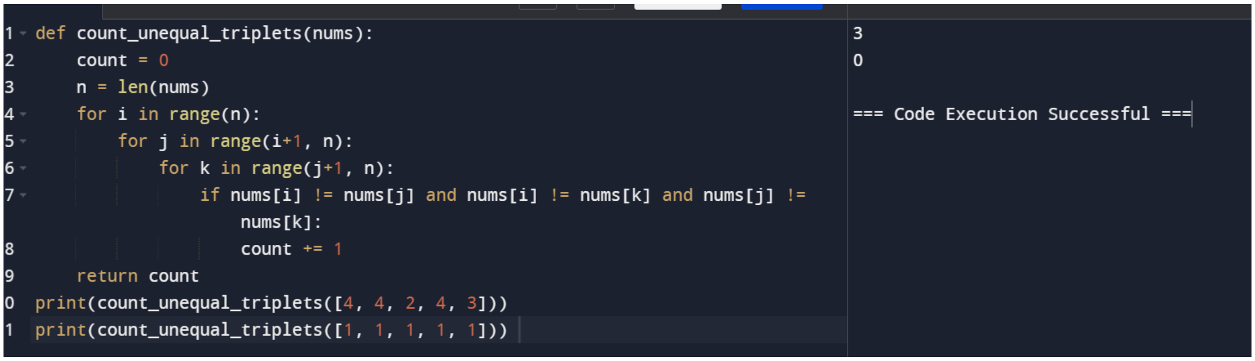
Output:0

Explanation:Notripletsmeettheconditionssowereturn0.

Constraints:

● 3<=nums.length<=100

● 1<=nums[i]<=1000



```python
def count_unequal_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
print(count_unequal_triplets([4, 4, 2, 4, 3]))
print(count_unequal_triplets([1, 1, 1, 1, 1]))
```

```
3
0

=== Code Execution Successful ===
```

8.ClosestNodesQueriesinaBinarySearchTree

Youaregiventherootofabinarysearchtreeandanarrayqueriesofsizenconsistingof

positiveintegers.

Finda2Darrayanswerofsizenwhereanswer[i]=[mini,maxi]:

● miniisthelargestvalueinthetreethatissmallerthanorequaltoqueries[i].Ifasuch

valuedoesnotexist,add-1instead.

● maxiisthesmallestvalueinthetreethatisgreaterthanorequaltoqueries[i].Ifa

suchvaluedoesnotexist,add-1instead.

Returnthearrayanswer.

Example1:

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

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

Explanation:Weanswerthequeriesinthefollowingway:-
Thelargestnumberthatissmallerorequalthan2inthetreeis2,andthesmallestnumber

thatisgreaterorequalthan2isstill2.Sotheanswerforthefirstqueryis[2,2].-
Thelargestnumberthatissmallerorequalthan5inthetreeis4,andthesmallestnumber

thatisgreaterorequalthan5is6.Sotheanswerforthesecondqueryis[4,6].-
Thelargestnumberthatissmallerorequalthan16inthetreeis15,andthesmallestnumber

thatisgreaterorequalthan16doesnotexist.Sotheanswerforthethirdqueryis[15,-1].

Example2:

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

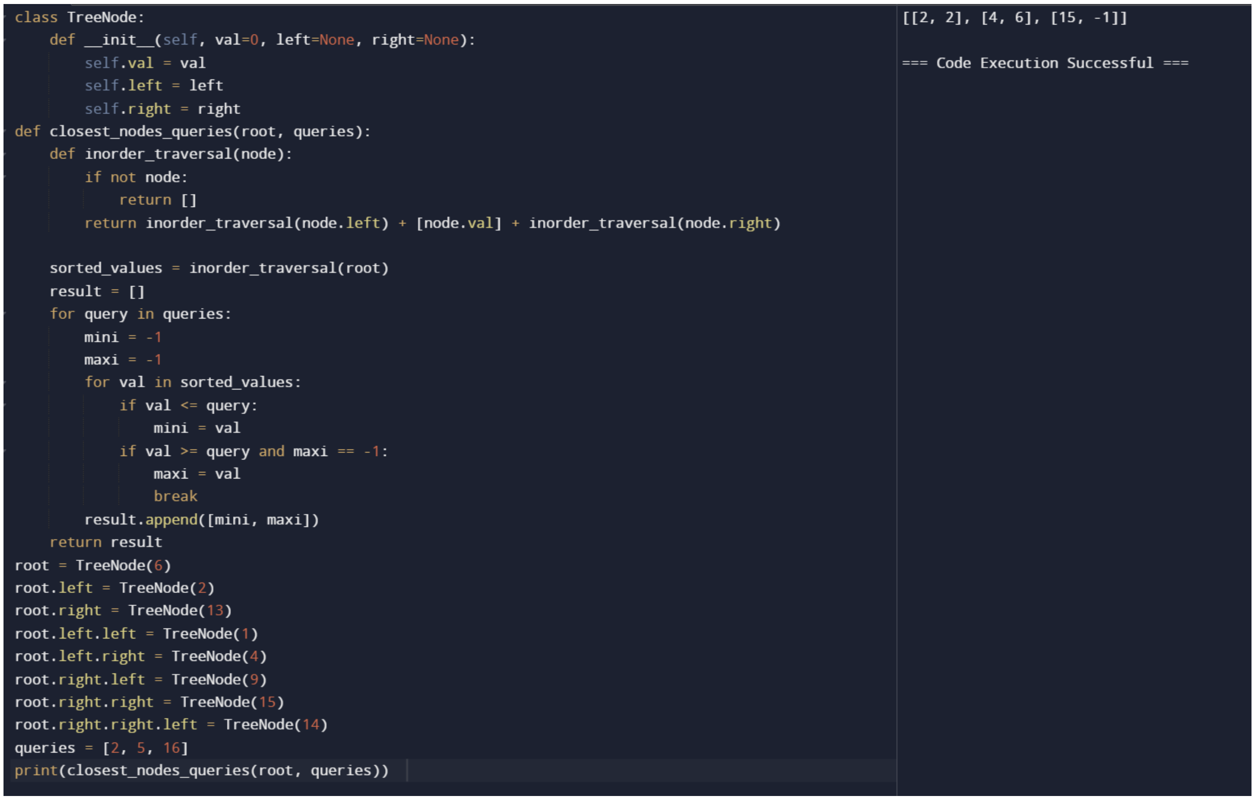
Output:[[-1,4]]

Explanation:Thelargestnumberthatissmallerorequalto3inthetreedoesnotexist,and

thesmallestnumberthatisgreaterorequalto3is4.Sotheanswerforthequeryis[-1,4].

Constraints:

- Thenumberofnodesinthetreeisintherange[2,105].
- 1<=Node.val<=106
- n==queries.length
- 1<=n<=105
- 1<=queries[i]<=106



```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def closest_nodes_queries(root, queries):
    def inorder_traversal(node):
        if not node:
            return []
        return inorder_traversal(node.left) + [node.val] + inorder_traversal(node.right)

    sorted_values = inorder_traversal(root)
    result = []
    for query in queries:
        mini = -1
        maxi = -1
        for val in sorted_values:
            if val <= query:
                mini = val
            if val >= query and maxi == -1:
                maxi = val
                break
        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(closest_nodes_queries(root, queries))
```

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

=== Code Execution Successful ===
```

9.MinimumFuelCosttoReporttotheCapital

Thereisatree(i.e.,aconnected,undirectedgraphwithnocycles)structurecountrynetwork

consistingofncitiesnumberedfrom0ton-1andexactlyn-1roads.Thecapitalcityiscity

0.Youaregivena2Dintegerarrayroadswhereroads[i]=[ai,bi]denotesthatthereexistsa

bidirectionalroadconnectingcitiesaiandbi.

Thereisameetingfortherepresentativesofeachcity.Themeetingisinthecapital

city.Thereisacarineachcity.Youaregivenanintegerseatsthatindicatesthenumberof

seatsineachcar.Arepresentativecanusethecarintheircitytotravelorchangethecarand

ridewithanotherrepresentative.Thecostoftravelingbetweentwocitiesisoneliteroffuel.

Returntheminimumnumberoflitersoffueltoreachthecapitalcity.

Example1:

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

Output:3

Explanation:-Representative1goesdirectlytothecapitalwith1literoffuel.-Representative2goesdirectlytothecapitalwith1literoffuel.-Representative3goesdirectlytothecapitalwith1literoffuel.

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 <=n<=105

roads.length == n- 1

roads[i].length == 2

0 <=ai, bi < n

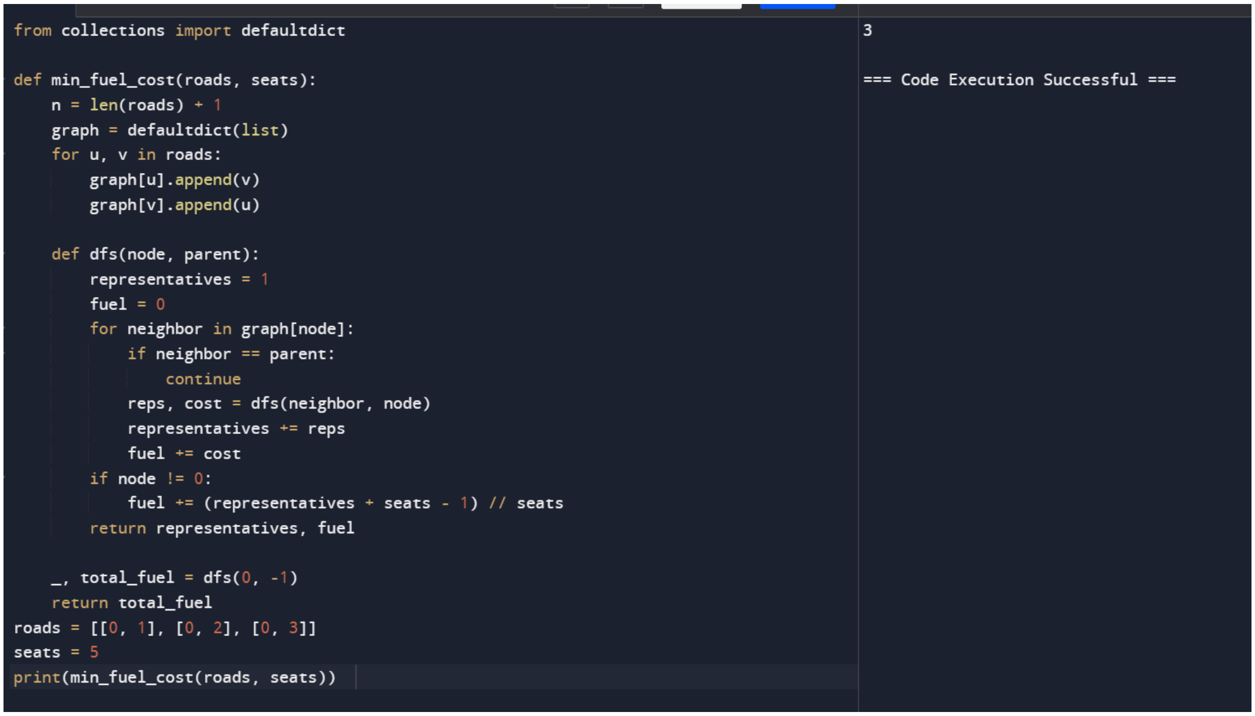
ai != bi

roads represents a valid tree.

●

●

1 <=seats <= 105



```
from collections import defaultdict

def min_fuel_cost(roads, seats):
    n = len(roads) + 1
    graph = defaultdict(list)
    for u, v in roads:
        graph[u].append(v)
        graph[v].append(u)

    def dfs(node, parent):
        representatives = 1
        fuel = 0
        for neighbor in graph[node]:
            if neighbor == parent:
                continue
            reps, cost = dfs(neighbor, node)
            representatives += reps
            fuel += cost
        if node != 0:
            fuel += (representatives + seats - 1) // seats
        return representatives, fuel

    _, total_fuel = dfs(0, -1)
    return total_fuel
roads = [[0, 1], [0, 2], [0, 3]]
seats = 5
print(min_fuel_cost(roads, seats))
```

```
3

=== Code Execution Successful ===
```

10. Number of Beautiful Partitions

You are given a string s that consists of the digits '1' to '9' and two integers k and minLength.

Apartition of s is called beautiful if:

●

s is partitioned into k non-intersecting substrings.

●

Each substring has a length of at least minLength.

- Eachsubstring 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 109 + 7.A substring is a contiguous sequence of characters within a string.

Example 1:

Input: s = "23542185131", k = 3, 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, minLength = 3

Output: 1

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

Example 3:

Input: s = "3312958", k = 3, minLength = 1

Output: 1

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

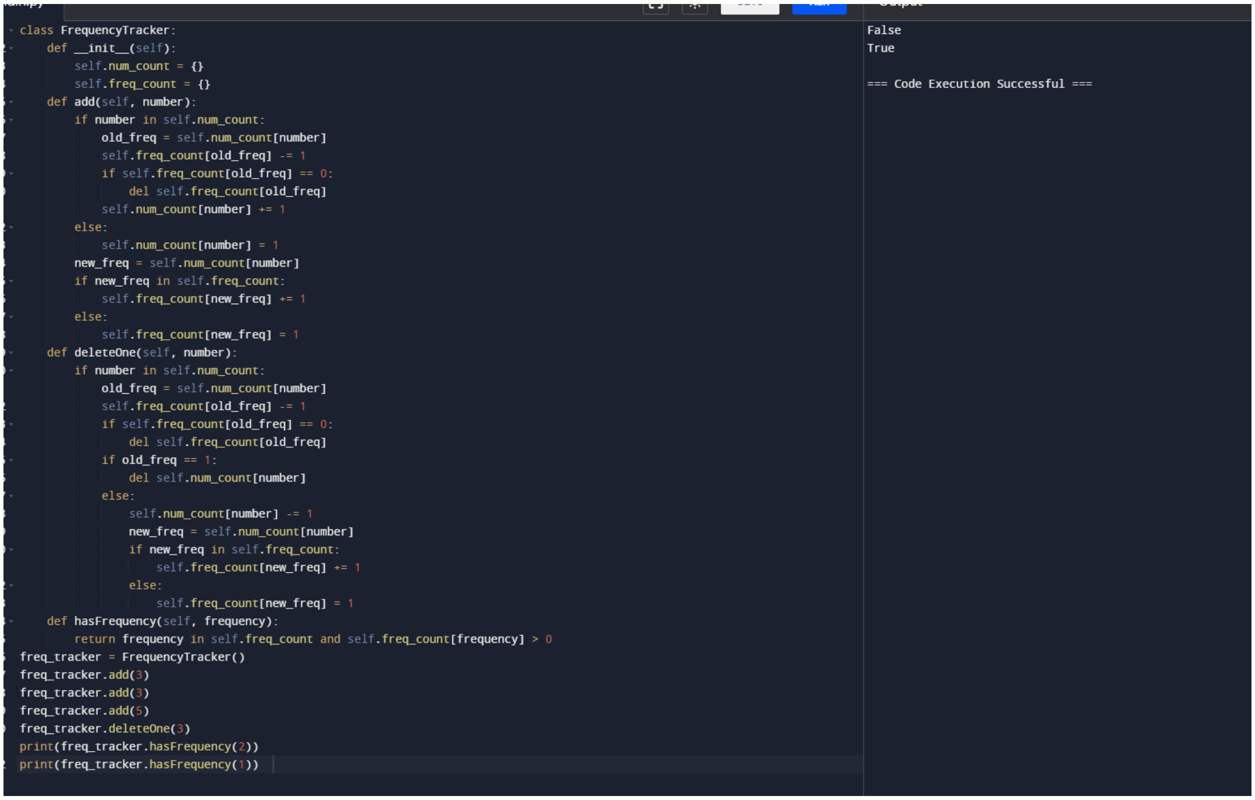
Constraints:

●

●

1 <=k, minLength <= s.length <= 1000

s consists of the digits '1' to '9'.



```python
class FrequencyTracker:
    def __init__(self):
        self.num_count = {}
        self.freq_count = {}
    def add(self, number):
        if number in self.num_count:
            old_freq = self.num_count[number]
            self.freq_count[old_freq] -= 1
            if self.freq_count[old_freq] == 0:
                del self.freq_count[old_freq]
            self.num_count[number] += 1
        else:
            self.num_count[number] = 1
        new_freq = self.num_count[number]
        if new_freq in self.freq_count:
            self.freq_count[new_freq] += 1
        else:
            self.freq_count[new_freq] = 1
    def deleteOne(self, number):
        if number in self.num_count:
            old_freq = self.num_count[number]
            self.freq_count[old_freq] -= 1
            if self.freq_count[old_freq] == 0:
                del self.freq_count[old_freq]
            if old_freq == 1:
                del self.num_count[number]
            else:
                self.num_count[number] -= 1
                new_freq = self.num_count[number]
                if new_freq in self.freq_count:
                    self.freq_count[new_freq] += 1
                else:
                    self.freq_count[new_freq] = 1
    def hasFrequency(self, frequency):
        return frequency in self.freq_count and self.freq_count[frequency] > 0
freq_tracker = FrequencyTracker()
freq_tracker.add(3)
freq_tracker.add(3)
freq_tracker.add(5)
freq_tracker.deleteOne(3)
print(freq_tracker.hasFrequency(2))
print(freq_tracker.hasFrequency(1))
```

```
False
True

=== Code Execution Successful ===
```