Queue_1

@kbbhatt04

@September 7, 2023

Minimum Multiplications to reach End

Given **start**, **end** and an array **arr** of **n** numbers. At each step, **start** is multiplied with any number in the array and then mod operation with **100000** is done to get the new start.

Your task is to find the minimum steps in which **end** can be achieved starting from **start**. If it is not possible to reach **end**, then return **-1**.

```
Input:
arr[] = {2, 5, 7}
start = 3, end = 30
Output:
2
Explanation:
Step 1: 3*2 = 6 % 100000 = 6
Step 2: 6*5 = 30 % 100000 = 30
```

- Approach
  - Brute-force
    -
    - Time Complexity: $O(n^3)$
    - Space Complexity: $O(1)$
  - Optimal
    - Initialize a queue for BFS traversal
    - Create an array `min_steps_to_value` to keep track of the minimum steps required to reach each value modulo 100000

- Calculate the modulo of the `start` value and mark it as having 0 steps to reach
- Enqueue the `start` value along with the step count (which is 0) into the queue
- While the queue is not empty, dequeue the current value and the number of steps taken to reach it
- Check if the current value is equal to the `end` value. If yes, return the number of steps
- For each multiplier in the `arr` array, calculate the next value by multiplying the current value with the multiplier and taking the modulo operation with 100000
- If the next value has not been visited before (i.e., min_steps_to_value[next_value] == -1), enqueue it along with the updated step count and mark it in the `min_steps_to_value` array
- If the target value is not reached after exploring all possibilities, return -1
- Time Complexity: $O(100000 * n)$ where `n` is the number of elements in the `arr` array. The BFS algorithm explores all possible values up to a maximum of 100000, and for each value, it considers each multiplier from the `arr` array.
- Space Complexity: $O(100000)$

```
# Python3
# Brute-force Solution
```

```
# Python3
# Optimal Solution
from collections import deque
class Solution:
    def minimumMultiplications(self, arr, start, end):
        mod = 100000
```

```
        min_steps = [-1 for _ in range(mod)]
        dq = deque()

        start_mod_value = start % mod
        dq.append((start_mod_value, 0))
        min_steps[start_mod_value] = 0

        while dq:
            current_value, steps = dq.popleft()

            if current_value == end:
                return steps

            for mul in arr:
                next_val = (current_value * mul) % mod

                if min_steps[next_val] == -1:
                    dq.append((next_val, steps + 1))
                    min_steps[next_val] = steps + 1
        return -1
```

```
// C++
// Optimal Solution
```

Template

- Approach
  - Brute-force
    -
    - Time Complexity: $O(n^3)$
    - Space Complexity: $O(1)$

- Better
  -
  - Time Complexity: $O(n^3)$
  - Space Complexity: $O(1)$
- Optimal
  -
  - Time Complexity: $O(n^3)$
  - Space Complexity: $O(1)$

```
# Python3
# Brute-force Solution
```

```
# Python3
# Better Solution
```

```
# Python3
# Optimal Solution
```

```
// C++
// Optimal Solution
```