3202. Find the Maximum Length of Valid Subsequence II

Attempted

Medium | Topics | Companies | Hint

You are given an integer array `nums` and a **positive** integer `k`.
A subsequence `sub` of `nums` with length `x` is called **valid** if it satisfies:

- `(sub[0] + sub[1]) % k == (sub[1] + sub[2]) % k == ... == (sub[x - 2] + sub[x - 1]) % k.`

Return the length of the **longest valid** subsequence of `nums`.

Example 1:

> **Input:** `nums = [1,2,3,4,5], k = 2`
>
> **Output:** 5
>
> **Explanation:**
>
> The longest valid subsequence is `[1, 2, 3, 4, 5]`.

Python3 • Auto

```python
class Solution:
    def maximumLength(self, nums: List[int], k: int) -> int:
        pattern = []
        for i in range(k):
            for j in range(k):
                pattern.append([i,j])

        res = 0

        for p in pattern:
            count = 0
            for n in nums:
                if n % k == p[count % 2]:
                    count += 1

            res = max(res, count)

        return res
```

time limit exceeded (inefficient if k and n are too large)

```python
class Solution:
    def maximumLength(self, nums: List[int], k: int) -> int:
        dp =[[0] * k for _ in range(k)]
        res = 0

        for n in nums:
            n %= k
            for prev in range(k):
                dp[prev][n] = dp[n][prev] + 1
                res = max(res, dp[prev][n])

        return res
```

Comparison

Aspect	DP Approach	Brute Force Approach
Time Complexity	`O(n * k)`	`O(n * k^2)` (slower for large `k`)
Space Complexity	`O(k^2)`	`O(1)`
Optimality	Yes (efficient for large inputs)	No (fails for large inputs)
Use Case	Best for `k` up to ~1000	Only for very small `k` (e.g., `k=2`)

Key Takeaways

- **DP Approach** is optimal and handles larger inputs efficiently.
- **Brute Force** is simpler but impractical due to high time complexity.
- The DP solution leverages the observation that the validity of a subsequence depends only on the last two elements modulo `k`.