2. K-Subarrays

A *k-subarray* of an array is:

- a subarray (contiguous elements)
- the sum of the elements is evenly divisible by *k (sum modulo k = 0)*.

Given an array of integers, determine the number of *k-subarrays* it contains.

Example

k = 5

nums = [5, 10, 11, 9, 5]

The *10 k-subarrays* are *{5}, {5, 10}, {5, 10, 11, 9}, {5, 10, 11, 9, 5}, {10}, {10, 11, 9}, {10, 11, 9, 5}, {11, 9}, {11, 9, 5}, {5}.*

Function Description

Complete the function *kSub* in the editor below. The function must return a long integer that represents the number of *k-subarrays* in the array.

kSub has the following parameter(s):

int k: the divisor of a k-subarray

int nums[n]: an array of integers

Return

long int: the number of k-subarrays in *nums*

Constraints

- $1 \le n \le 3 \times 10^5$

Language C++20

Autocomplete Ready

```
 * The function is expected to return a LONG_INTEGER.
 * The function accepts following parameters:
 *  1. INTEGER k
 *  2. INTEGER_ARRAY nums
 */
long dp[300005][101] = {0};


long kSub(int k, vector<int> nums) {
    // dp[0][0] =
    dp[0][nums[0] % k] = 1;

    int n = nums.size();

    for (int i = 1; i < n; ++i) {
        dp[i][nums[i] % k] = 1;
        for (int j = 0; j < k; ++j) {
            // dp[i][j] =
            // depends on prev index
            // and prev modulo number
            dp[i][(j + nums[i]) % k] += dp[i-1][j];
        }
    }



    long res = 0;
    for (int i = 0; i < n; ++i) {
        // cout << dp[i][0] << ' ';
        res += dp[i][0];
    }

    return res;


```

Line: 9 Col: 1

Test Results | Custom Input | Run Code | Run Tests | Submit

5m left

ALL

3. Visiting Cities

There are a number of cities in a row, and there are two bus lines that go between them. They both visit all cities in order, but one may take longer than the other to go between any two cities. Starting on or moving to the Blue line takes a certain amount of extra time. There is no extra time required to start on or move to the Red line. Determine the minimum cost to move from the first city to each of the cities.

Example

red = [2, 3, 4]

blue = [3, 1, 1]

blueCost = 2

There are *4* cities numbered *0* through *3*. Times from city 0 to cities 1, 2, and 3 are at indices 0, 1, and 2 respectively in the *red* and *blue* arrays.

Through the explanation, an answer array, *ans*, will be created.

- The minimum cost to go from city 0 to itself is 0. Now *ans =[0]*
- The time from city 0 to city 1 is
  - *2* on the Red line
  - *3 + blueCost = 5* on the Blue line.
    - The *blueCost* applies when you start on the Blue line.
  - The minimum time to city 1 is *2* on the Red line, so *ans = [0, 2]*.
- Continuing to city 2:
  - stay on the Red line, arriving at *2 + 3 = 5*
  - switch to the Blue line and arrive at *2 + 1 + 2 =*

Language C++14

Autocomplete Ready

```
#include <bits/stdc++.h>

/*
 * Complete the 'minimumCost' function below.
 *
 * The function is expected to return a LONG_INTEGER_ARRAY.
 * The function accepts following parameters:
 *  1. INTEGER_ARRAY red
 *  2. INTEGER_ARRAY blue
 *  3. INTEGER blueCost
 */

// long dp[2][200005];

long long dp[2][200005];

vector<long> minimumCost(vector<int> red, vector<int> blue, int blueCost) {
    int n = red.size();
    dp[0][0] = 0;
    dp[1][0] = blueCost;
    for (int i = 1; i <= n; i++) {
        dp[0][i] = min({dp[1][i-1] + red[i-1], dp[1][i-1] + blue[i-1], dp[0][i-1] + red[i-1]});

        dp[1][i] = min({dp[0][i-1] + blueCost + blue[i-1], dp[0][i-1] + blueCost + red[i-1], dp[1][i-1] + blue[i-1]});
    }
    vector<long> ans;
    for (int i = 0; i <= n; i++) {
        ans.push_back(min(dp[0][i], dp[1][i]));
    }

    return ans;
}

int main()
```

Line: 15 Col: 25

Test Results | Custom Input | Run Code | Run Tests | Submit