Squared Subsequences - Codechef

1 Mathematical Reduction

- **Fact 1** : Any integer can be written as the product of 2 integers.
- **Proof** : Observe that in the worst case we can always write it as $num = 1 \cdot num$

Now, consider a number $num = p^2 - q^2$, where p and q are integers. By using High School Mathematics Formula, we have

$$num = (p + q) \cdot (p - q) \tag{1}$$

Also, by **Fact 1**, num has a factorization as the product of 2 integers. Hence,

$$num = x \cdot y = (p + q) \cdot (p - q) \tag{2}$$

Notice that we have not placed any restrictions on x and y. Hence, it covers all the possibilities.
Comparing LHS and RHS and solving the equations

$$x = p + q \tag{3}$$

$$y = p - q \tag{4}$$

leads us to the result

$$p = \frac{x + y}{2} \tag{5}$$

$$q = \frac{x - y}{2} \tag{6}$$

For the original assumption to be valid, (i.e p and q are integers), $(x + y)$ as well as $(x - y)$ must be divisible by 2, i.e, they must be even numbers. Given this information, we would like to know the parity of x and y.
Here's a recall on how the parity of 2 numbers change when we take their sum or difference.

	Even	**Odd**
Even	Even	Odd
Odd	Odd	Even

From the table, we conclude that there is only one restriction on x and y (i.e. they must be of the same parity) for the original assumption to be valid

- If the original number was odd, we can write it as the product of 2 odd integers (i.e 1 and the number itself).
- If the original number was even, it means we need to write it as the product of 2 even integers, ie. $num = (2m) \cdot (2n)$. This implies that the original number has to be divisible by 4.

1.1 Conclusion

Any number can be written in the form $(p^2 - q^2)$ **iff** the number is **odd** or the number is divisible by **4**. So the original question reduces to "**Find the number of subarrays whose product is either divisible by 4 or not divisible by 2**"

2 Dynamic Programming

We would like to answer the question, "**How many subarrays are there whose product is divisible by** mod".

By now, I assume that you know the basics of **DP**. Let us define $dp[i][rem]$ as the "**Number of Subarrays ending at** i **with remainder** rem". It's obvious that to answer the original question, we need to sum up the values $dp[any][0]$ where any is a valid array index.

Let us see how to perform the state transitions for $dp[i][rem]$. The resulting subarray has to end at i. So, it can either be the single element situated at i, or it can be the continuation of the subarray ending at $(i - 1)$.

- If it is a single element subarray, we need to increment the value of $dp[i][a[i]\%mod]$ by 1.
- If it is a continuation, we need to increment the value of $dp[i][new\_rem]$ by $dp[i-1][old\_rem]$, where $new\_rem = (old\_rem \cdot a[i])\%mod$. We iterate over all possible old_rem and update the DP table accordinlgy.

2.1 Caution

Note that the modulo of a negative number can be negative (as per C++). So, to avoid segmentation fault due to *negative-index-access*, we can deal with the absolute values instead. This wont change the answer because if a negative number can be written as $(p^2 - q^2)$, then the absolute value of the number can be written as $(q^2 - p^2)$. Of course, we can also define our version of the modulo which does not generate negative numbers.

3 Putting it all Together

Recall that the total number of subarrays of an array of length n is

$$n + \binom{n}{2} = \frac{n \cdot (n+1)}{2} \tag{7}$$

Using the DP approach, we can find out the number of subarrays which are divisible by 4. To find out the number of subarrays with odd product, we just do $Total\_Subarray\_Count - Even\_Product\_Subarrays$

4 Related Problems

If you want to practice a question with similar DP formulation and transistion(minus the mathematical reduction), you should check out the question titled **Vacations** from **AtCoder Educational DP Contest**.

5 Code

You find the **C++** implementation here and here

6 Pseudocode

Algorithm Find the number of subarrays whose product can be written as the difference of 2 squares

Ensure: Zero Based Indexing for the array and matrix

```
procedure Subarrays_Divisible_By(mod, arr)
▷ dp[i][rem] stores the number of subarrays ending at i with remainder rem

    dp[i][j] ← 0        ∀i        ∀j        ▷ Initialise the DP Matrix
    dp[0][a[0]%mod] ← 1

    for i ∈ [1 : arr.len) do
        dp[i][a[i]%mod] += 1
        for old_rem ∈ [0, mod) do
            new_rem ← (old_rem · a[i])%mod
            dp[i][new_rem] += dp[i − 1][old_rem]

    count ← 0
    for i ∈ [0 : arr.len) do
        count += dp[i][0]
    return count
end procedure

function Good_Subarray(arr)
    for each ele ∈ arr do
        ele ← |ele|

    ans ← Subarrays_Divisible_By(4, arr)
    ans ← n·(n+1)/2 − Subarrays_Divisible_By(2, arr)
    return ans
end function
```