LeetCode 413

https://leetcode.com/problems/arithmetic-slices/description/

Yifeng Zeng

Description

413. Arithmetic Slices

A sequence of number is called arithmetic if it consists of at least three elements and if the difference between any two consecutive elements is the same.

```
For example, these are arithmetic sequence:
1, 3, 5, 7, 9
7, 7, 7, 7
3, -1, -5, -9
The following sequence is not arithmetic.
1, 1, 2, 5, 7
```

A zero-indexed array A consisting of N numbers is given. A slice of that array is any pair of integers (P, Q) such that 0 <= P < Q < N.

A slice (P, Q) of array A is called arithmetic if the sequence:
A[P], A[p + 1], …, A[Q - 1], A[Q] is arithmetic. In particular, this means that P + 1 < Q.

The function should return the number of arithmetic slices in the array A.

Example:

A = [1, 2, 3, 4]

return: 3, for 3 arithmetic slices in A: [1, 2, 3], [2, 3, 4] and [1, 2, 3, 4] itself.

Idea Report

The primitive idea is the brute force, we fix 1 numebr and see how far can it go for being a arithmetic sequence. The time coplexity is O(n^2)

Code:

```
class Solution {
    public int numberOfArithmeticSlices(int[] A) {

        int res = 0;
        for (int i = 0; i < A.length - 1; i++) {
            int j = i + 1;
            int diff = A[j] - A[i];
            for (; j + 1 < A.length; j++) {
                if (A[j + 1] - A[j] == diff) {
                    res++;
                } else {
                    break;
                }
            }
        }
        return res;
    }
}
```

After j + 1 position no longer make the sub array a arithmetic sequence, we don't have to go back to i + 1, we can just continue to search from j, so we make i = j - 1;, this will make the whole algorithm to be O(n).

Code:

```
class Solution {
    public int numberOfArithmeticSlices(int[] A) {

        int res = 0;
        for (int i = 0; i < A.length - 2; i++) {
            int j = i + 1;
            int diff = A[j] - A[i];
            int count = 0;
            for (; j + 1 < A.length; j++) {
                if (A[j + 1] - A[j] == diff) {
                    count++;
                } else {
                    break;
                }
            }
            i = j - 1;
            count = getTotalCount(count);
            // System.out.println("count = " + count + ", i = " + i);
            res += count;
        }
        return res;
    }

    private int getTotalCount(int c) {
        if (c <= 1) {
            return c;
        }
        return (1 + c) * c / 2;
    }
}
```

Summary

- Two pointers, chasing pointer.
- sum from a to b = (a + b) * n / 2