Arithmetic Slices II - Subsequence

A sequence of numbers 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 sequences:

```
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 **subsequence** slice of that array is any sequence of integers (P_0, P_1, ..., P_k) such that $0 \le P_0$ 1 k

A **subsequence** slice (P_0, P_1, ..., P_k) of array A is called arithmetic if the sequence A[P_0], A[P_1], ..., A[P_{k-1}], A[P_k] is arithmetic. In particular, this means that $k \ge 2$.

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

The input contains N integers. Every integer is in the range of -2^{31} and 2^{31}-1 and $0 \le N \le 1000$. The output is guaranteed to be less than 2^{31}-1.

Example:

```
Input: [2, 4, 6, 8, 10]

Output: 7

Explanation:
All arithmetic subsequence slices are:
[2,4,6]
[4,6,8]
[6,8,10]
[2,4,6,8]
[4,6,8,10]
[2,4,6,8,10]
[2,6,10]
```

Solution 1

```
    public int numberOfArithmeticSlices(int[] A) {
        int re = 0;
        HashMap<Integer, Integer>[] maps = new HashMap[A.length];
        for(int i=0; i<A.length; i++) {
            maps[i] = new HashMap<>();
            int num = A[i];
            for(int j=0; j<i; j++) {
                if((long)num-A[j]>Integer.MAX_VALUE) continue;
                if((long)num-A[j]<Integer.MIN_VALUE) continue;
                int diff = num - A[j];
                int count = maps[j].getOrDefault(diff, 0);
                maps[i].put(diff, maps[i].getOrDefault(diff,0)+count+1);
                re += count;
            }
        }
        return re;
    }
```

written by aaaeeeo original link here

Solution 2

First of all, the standard O(N^2) DP solution written with C++ goes MLE/TLE in LC, but works pretty well when written with JAVA/Python :(
It looks like this (Python version):

```
class Solution(object):
    def numberOfArithmeticSlices(self, A):
        dp = [defaultdict(int) for i in range(len(A))]
        res = 0
        for i in range(1, len(A)):
            for j in range(i):
                step = A[i]-A[j]
                dp[i][step] += 1
                if step in dp[j]:
                    dp[i][step]+= dp[j][step]
                    res += dp[j][step]
        return res
```

Obviously, it maintains an array of dictionary to store the number of arithmetic subsequences (including length 2) ending with A[i]. As a result, the time and space complexity are both O(N^2) which I think is quite reasonable to deal with 0 ≤ N ≤ 1000 in LC. But it fails in C++. So I tweak the meaning of DP equation from:

```
DP[i][d] = the number of arithmetic subsequences ending with A[i], difference is
d. (NOTE here the length of valid subsequences can be 2)
```

to

```
DP[i][d] = the number of arithmetic subsequences whose last but one number is A[i
], difference is d.
```

After that, the length of valid subsequences we record must be at least 3. It does save memory and finally passes LC in 423 ms:

```
class Solution {
public:
    int numberOfArithmeticSlices(vector<int>& A) {
        if (A.empty()) return 0;
        int n = A.size();
        vector<unordered_map<long long, int >> dp(n);
        unordered_set<int> s(A.begin(), A.end());
        int res = 0;
        for (int i = 1; i < n; ++i) {
            for (int j = i-1; j >= 0; --j) {
                long long d = (long long)A[i] - (long long)A[j];
                int tmp = dp[j].count(d) ? dp[j][d] : 0;
                if (tmp) res += tmp;
                if (s.count(A[i]+d)) dp[i][d] += 1 + tmp;
            }
        }
        return res;
    }
};
```

written by vesion original link here

Solution 3

```java
public int numberOfArithmeticSlices(int[] A) {
        if(A==null||A.length<3) return 0;
        List<Map<Integer,Integer>> list=new ArrayList<Map<Integer,Integer>>();
        int res=0;
        for(int i=1;i<A.length;i++){
            Map<Integer,Integer> map=new HashMap<Integer,Integer>();
            for(int j=0;j<i;j++){
                if((long)A[i]-(long)A[j]>Integer.MAX_VALUE) continue;
                if((long)A[i]-(long)A[j]<Integer.MIN_VALUE) continue;
                int dif=A[i]-A[j];
                if(j==0){
                    map.put(dif,1);
                    continue;
                }
                Map<Integer,Integer> temp=list.get(j-1);
                int sum=0;
                if(temp.containsKey(dif)){
                    sum=temp.get(dif);
                }
                if(map.containsKey(dif)){
                    map.put(dif,map.get(dif)+sum+1);
                }else{
                    map.put(dif,sum+1);
                }
                res+=sum;
            }
            list.add(map);
        }
        return res;
    }
```

It seems that there is no better way to solve this problem using O(N^2) space and time. If you have better algorithm, please share your thoughts or solution, thanks!

I have viewed all four posts(at least there are four posts when I am still busy typing mine), and everyone seems to have the same issue facing TLE or MLE or both. Well, for me, I have encountered both.

Since nobody has posted java solution, I will share my ways of tackling this problem.

I will skip the logic and algorithms here, since everybody is doing it the same way, using DP. For each element, store all possible difference and its number of arithmetic sequences(including 2 element sequence).

At first, I was using Map<Long,Integer> instead of Map<Integer,Integer> considering subtraction of two integers could result in overflow. But later I realized if the result is bigger than the max value of integer or smaller than the min value of integer, there is no way to have a valid a 3rd integer element to form a arithmetic sequence. So by adding two if()*continue, and thus replacing Map<Long,Integer> to Map<Integer,Integer> it can save both time and memory. After doing this, my code can pass the judge about 4 out of 5 times, which means it can still have TLE or MLE sometimes. Then I modified the outer loop's parameter i to start from 1 instead of 0 and added if(j==0) statement to do less map operations for every j==0 case. And now it goes through every time! (Well, by saying every time, I mean I submitted 5 times in a row and they all went through.)

I believe the first change is the main cause to help my code get through, and the second change is just the minor. Hope these can give you guys some hints and help, please do share your plans of reducing either time or space here if you have.
Happy coding!

written by KnightY original link here