Dynamic Programming | Set 35 (Longest Arithmetic Progression)

Given a set of numbers, find the **L**ength of the **L**ongest **A**rithmetic **P**rogression (**LLAP**) in it.

Examples:

```
set[] = {1, 7, 10, 15, 27, 29}
output = 3
The longest arithmetic progression is {1, 15, 29}

set[] = {5, 10, 15, 20, 25, 30}
output = 6
The whole set is in AP
```

We strongly recommend that you click here and practice it, before moving on to the solution.

For simplicity, we have assumed that the given set is sorted. We can always add a pre-processing step to first sort the set and then apply the below algorithms.

A **simple solution** is to one by one consider every pair as first two elements of AP and check for the remaining elements in sorted set. To consider all pairs as first two elements, we need to run a O(n^2) nested loop. Inside the nested loops, we need a third loop which linearly looks for the more elements in **A**rithmetic **P**rogression (**AP**). This process takes $O(n^3)$ time.

We can solve this problem in $O(n^2)$ time **using Dynamic Programming**. To get idea of the DP solution, let us first discuss solution of following simpler problem.

Given a sorted set, find if there exist three elements in Arithmetic Progression or not
Please note that, the answer is true if there are 3 or more elements in AP, otherwise false.
To find the three elements, we first fix an element as middle element and search for other two (one smaller and one greater). We start from the second element and fix every element as middle element. For an element set[j] to be middle of AP, there must exist elements 'set[i]' and 'set[k]' such that set[i] + set[k] = 2*set[j] where 0 <= i < j and j < k <=n-1. *How to efficiently find i and k for a given j?* We can find i and k in linear time using following simple algorithm.
1) Initialize i as j-1 and k as j+1
2) Do following while i >= 0 and j <= n-1 ....**a)** If set[i] + set[k] is equal to 2*set[j], then we are done.
........**b)** If set[i] + set[k] > 2*set[j], then decrement i (do i–).
........**c)** Else if set[i] + set[k] < 2*set[j], then increment k (do k++). Following is C++ implementation of the above algorithm for the simpler problem. [sourcecode language="C" highlight="1,2-23"] // The function returns true if there exist three elements in AP // Assumption: set[0..n-1] is sorted. // The code strictly implements the algorithm provided in the reference. bool arithmeticThree(int set[], int n) { // One by fix every element as middle element for (int j=1; j<n-1; j++) { // Initialize i and k for the current j int i = j-1, k = j+1; // Find if there exist i and k that form AP // with j as middle element while (i >= 0 && k <= n-1) { if (set[i] + set[k] == 2*set[j]) return true; (set[i] + set[k] < 2*set[j])? k++ : i--; } } return false; } [/sourcecode] See this for a complete running program.

How to extend the above solution for the original problem?
The above function returns a boolean value. The required output of original problem is **L**ength of the **L**ongest **A**rithmetic **P**rogression (**LLAP**) which is an integer value. If the given set has two or more elements, then the value of LLAP is at least 2 (Why?).
The idea is to create a 2D table L[n][n]. An entry L[i][j] in this table stores LLAP with set[i] and set[j] as first two elements of AP and j > i. The last column of the table is always 2 (Why – see the meaning of L[i][j]). Rest of the table is filled from bottom right to top left. To fill rest of the table, j (second element in AP) is first fixed. i and k are searched for a fixed j. If i and k are found such that i, j, k form an AP, then the value of L[i][j] is set as L[j][k] + 1. Note that the value of L[j][k] must have been filled before as the loop traverses from right to left columns.

Following is C++ implementation of the Dynamic Programming algorithm.

```cpp
// C++ program to find Length of the Longest AP (llap) in a given sorted set.
// The code strictly implements the algorithm provided in the reference.
#include <iostream>
using namespace std;

// Returns length of the longest AP subset in a given set
int lenghtOfLongestAP(int set[], int n)
{
    if (n <= 2)  return n;

    // Create a table and initialize all values as 2. The value of
    // L[i][j] stores LLAP with set[i] and set[j] as first two
    // elements of AP. Only valid entries are the entries where j>i
    int L[n][n];
    int llap = 2;  // Initialize the result

    // Fill entries in last column as 2. There will always be
    // two elements in AP with last number of set as second
    // element in AP
    for (int i = 0; i < n; i++)
        L[i][n-1] = 2;

    // Consider every element as second element of AP
    for (int j=n-2; j>=1; j--)
    {
        // Search for i and k for j
        int i = j-1, k = j+1;
        while (i >= 0 && k <= n-1)
        {
           if (set[i] + set[k] < 2*set[j])
               k++;

           // Before changing i, set L[i][j] as 2
           else if (set[i] + set[k] > 2*set[j])
           {   L[i][j] = 2, i--;   }

           else
           {
               // Found i and k for j, LLAP with i and j as first two
               // elements is equal to LLAP with j and k as first two
               // elements plus 1. L[j][k] must have been filled
               // before as we run the loop from right side
               L[i][j] = L[j][k] + 1;

               // Update overall LLAP, if needed
               llap = max(llap, L[i][j]);

               // Change i and k to fill more L[i][j] values for
               // current j
               i--; k++;
           }
        }

        // If the loop was stopped due to k becoming more than
        // n-1, set the remaining entties in column j as 2
        while (i >= 0)
        {
            L[i][j] = 2;
            i--;
        }
    }
    return llap;
}

/* Drier program to test above function*/
int main()
{
    int set1[] = {1, 7, 10, 13, 14, 19};
    int n1 = sizeof(set1)/sizeof(set1[0]);
    cout <<   lenghtOfLongestAP(set1, n1) << endl;

    int set2[] = {1, 7, 10, 15, 27, 29};
    int n2 = sizeof(set2)/sizeof(set2[0]);
    cout <<   lenghtOfLongestAP(set2, n2) << endl;

    int set3[] = {2, 4, 6, 8, 10};
    int n3 = sizeof(set3)/sizeof(set3[0]);
    cout <<   lenghtOfLongestAP(set3, n3) << endl;

    return 0;
}
```

Run on IDE

Output:

```
4
3
5
```

Time Complexity: $O(n^2)$
Auxiliary Space: $O(n^2)$

Asked in: Snapdeal

References:
http://www.cs.uiuc.edu/~jeffe/pubs/pdf/arith.pdf