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
```

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.

```
// 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;
}
```

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.

```
// 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;
}
```

Output:

```
4
3
5
```

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

References:

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

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Related Topics:

- Linearity of Expectation
- Iterative Tower of Hanoi
- Count possible ways to construct buildings
- Build Lowest Number by Removing n digits from a given number
- Set Cover Problem | Set 1 (Greedy Approximate Algorithm)
- Find number of days between two given dates
- How to print maximum number of A's using given four keys
- Write an iterative O(Log y) function for pow(x, y)

Tags: Dynamic Programming

Tweet 3 +1 1

Writing code in comment? Please use **ideone.com** and share the link here.

49 Comments GeeksforGeeks 1 Login

Recommend 2 Share Sort by Newest

Join the discussion…

Coder • 5 months ago

import java.util.HashMap;

public class maxAP {

static int max(int a,int b){

return (a<b)?b:a; }="" static="" int="" getmaxaplength(int[]="" arr){="" int="" n="arr.length;" int[][]="" table="new" int[n][n];="" hashmap<integer,integer=""> hm = new HashMap<integer, integer="">();

int i,j;

int max_val = Integer.MIN_VALUE;

for(i=1;i<n;i++){ for(j="0;j&lt;i;j++){" integer="" diff="arr[i]-arr[j];" if(hm.get(diff)!="null){" integer="" size="hm.get(diff);" hm.put(diff,="" size+1);="" table[j][i]="size+1;" max_val="max(table[j][i],max_val);" }="" else{="" hm.put(diff,="" 2);="" table[j][i]="2;" }="" }="" }="" return="" max_val;="" }="" public="" static="" void="" main(string[]="" args){="" int[]="" arr="{1," 7,="" 10,="" 13,=""

14,="" 19};="" system.out.println(getmaxaplength(arr));="" }="" }="">

• Reply • Share ›

Akash Agrawall • 6 months ago

I have an approach which is quite easy to understand and uses O(n^2) complexity and O(n^2) auxilary space(might be more).
Sort the array of numbers.
Now maintain a dp[i][j] where i denotes the index of the array and j the difference till now and this array gives me the number of elements present in the sequence till i with difference j.
maximum array to create dp[n][arr[n-1]-arr[0]],
where n=length of array
arr=sorted array
we initialize the dp array by putting dp[i][arr[k]-i]=1 and all other elements by zero.

pseudo code for dp:

```
for i=0 to n-1 //Assuming 0 based-index
for j=i-1 to 0
sub=arr[i]-arr[j]
arr[i][sub]=max(arr[i][sub],arr[j][sub]+1)

ans=-1
for i=0 to n-1:
for j=0 to arr[n-1]-arr[0]:
ans=max(ans,dp[i][j])

printf(ans)
```

• Reply • Share ›

kcahdog • 10 months ago

In the initialization of DP you do L [i][n-1] = 2. This will also set L[n-1] [n-1] to 2 which by definition of L[i][j] is wrong. I think L[n-1] [n-1] should be set to 1

1 • Reply • Share ›

zeal deal • 10 months ago

Can anyone tell me what's wrong with this approach? We dont need to 2D table.

http://ideone.com/BgrPDI

• Reply • Share ›

Gaurav Gupta • 10 months ago

O(n^2) solution using hashmaps.
http://ideone.com/IDxDUR

• Reply • Share ›

Sunny Mitra • 10 months ago

@GeeksForGeeks:
This algorithm is failing to provide correct output for the given array:
{1, 4, 3, 2, 5, 7}

The output of the above algorithm is coming as 3, where the correct output should be 4 (for the sequence 1, 3, 5, 7).
I think this is because of the following reason:
when j = 4 (element 5), at first iteration i becomes 3 (element 2) and k becomes 5 (element 7), then a[j]-a[i] = 3 and a[k]-a[j] = 2, so we increment k instead of decrementing i; for this reason we are missing an opportunity to get an optimal AP sequence (3, 5, 7) for this particular subproblem. Hence we are not getting the global optimum either.
Please correct me, if I'm missing something. Otherwise, it would be a great help if you could mail me the correct solution.

• Reply • Share ›

Sunny Mitra Sunny Mitra • 10 months ago

Sorry, I overlooked the sorting portion !!!

• Reply • Share ›

Ravi Shankar Mondal • a year ago

if an unsorted sequence is given and the range of numbers are also known then how can I found the result?

• Reply • Share ›

ravi_pagla Ravi Shankar Mondal • 10 months ago

Sort it first and apply the algorithm for sorted sequence. complexity will still be O(n^2)

• Reply • Share ›

Ravi Shankar Mondal • a year ago

If an unsorted sequence is given and the range of numbers are also given then how can I found the result?

• Reply • Share ›

learner • a year ago

1]
For a input
int set1[] = {5,10,15,20,25};

when I print L[n][n] , I get below output.

```
0 5 3 2 2
0 0 4 2 2
0 0 0 3 2
```

0 0 0 0 2
0 0 0 0 2

I did not understand what does 3 , 4 , 5 values in the matrix signify? Can somebody explain ? Can somebody provide better explanation on how ALGORITHM works.

secondly
The entire matrix must be initialized with 2 not only the last column, as a pair of numbers in set1(which is assumed to be sorted) will always be in AP. Please correct me if I am wrong

• Reply • Share ›

Roxanne • a year ago

I found a very simple solution-
First - Create a 2d matrix such that array[i][j] is set[j] - set[i]. eg:
1 4 5 10 15
1 0 3 4 9 14
4 -3 0 1 6 11
5 -4 -1 0 5 10
10 -9 -6 -6 0 5
15 0

Now each element a[i][j] represents the difference(d) of AP from set[j] to set[i].
eg array[3][2] = 5 , is the AP of d= 5 between set[3] = 10 and set[2] = 5.

Now make another pass over this 2d array, neglect all negative numbers, and create a hashmap with size -> number i.e hashmap(key - a[i][j], value - list of all values for this size)
eg
5 -> 10, 15
3 -> 4
10-> 15
11 -> 15
.....

The one size with maximum number of elements in list is the answer.
in this case 5 -> 10, 15
answer is 10-5 = 5 , 10, 15.

Let me know if this has any issues!

3 • Reply • Share ›

Chao Zhou Roxanne • 6 months ago

This method will fail on the input {1, 2, 5, 6}.
Since array[0][2] = 4, array[1][3] = 4
there will be an entry in the hash table:
4 -> 1, 5, 2, 6

and the answer will be: 4
but the correct answer for this input is 2
The reason why it fail is that it only consider the difference of two numbers but neglect the relative position of different set of numbers.

• Reply • Share ›

anonymous Roxanne • a year ago

After seeing the problem, this exact solution came to my mind.

To optimize it a little, instead of filling the 2d with -ve elements, just put a check while filling, that if(j>i), only then perform set[j]-set[i], this can be done by modifying the inner for loop: for j from i+1 to n-1

And my way of storing the numbers was instead of using a hashmap, I was using an array of integers with space as much as the range of numbers(say MAX) and the particular position in the array denotes number of occurrences in the 2d array created.

But the auxiliary space turns out to be O(n^2+MAX), where MAX is the range of numbers.
For some test cases, this might turn out to be worse than the dp soln.
So using a hash table turns out to be better.

1 • Reply • Share ›

Roxanne anonymous • a year ago

Any idea what would we gain in dp solution over this one?

• Reply • Share ›

anonymous Roxanne • a year ago

Just seems more complicated..

• Reply • Share ›

Guest • a year ago

```
function a(){ System.out.println("a"); }
```

• Reply • Share ›

its_dark • 2 years ago

Here is a recursive function that solves this problem.how can we convert this recursion into a dp (OR memoize it), using the same logic ?? :

```
int recur(int a[], int index, int check[], int count, int size){
    if(index==size)
        return count;
    if(count <= 1 ){
        int temp = recur2(a,index+1,check,count,size);
        check[count++] = a[index];
        int temp2 = recur2(a,index+1,check,count,size);
        return max(temp,temp2);
    }
    if(count >= 2){
        if( (a[index] - check[count-1]) == (check[count-1] - check[count-2]) ){
           check[count++] = a[index];
            return recur2(a,index+1,check,count,size);
        }
```

see more

1 • Reply • Share ›

Hiral • 2 years ago

This solution is incorrect. You need to save in the 2*2 matrix, the corresponding difference for a particular L[i][j]. Consider below example:
10 15 20 21 27
You program will output 4 where in the correct answer is 3. This input contains two APs and both of them involve 15.
so one AP is 10 15 20
second one is 15 21 27

1 • Reply • Share ›

Mahdi Belbasi Hiral • 2 years ago

it's ok, the table would be like this :
1 3 2 2 2
0 1 2 3 2
0 0 1 2 2
0 0 0 1 2
0 0 0 0 1

as you see, the answre is 3 :)
where is the mistake?

• Reply • Share ›

Nitesh Kumar Mahdi Belbasi • 6 months ago

llap gives you the correct output and that is 3

• Reply • Share ›

Somebody • 2 years ago

What would be the time and space complexity, if the numbers are given in a sequence instead of a set? Which implies numbers are not in increasing order.

• Reply • Share ›

Alexander Korobeynikov • 2 years ago

Wouldn&#039t it be easier to init L with all 2 from the beginning?
Or maybe even with zeros and, then, return llap+2?

• Reply • Share ›

Sudipto • 2 years ago

How can we find the LLAP if the array is not sorted?
For e.g. if there are 3 elements in the array in the following order : {7, 13, 10}, then sorting would make this {7, 10, 13} and the LLAP will be 3 for the sorted array and not the original array.

• Reply • Share ›

piki • 2 years ago

can we solve this problem using O(n) space ....please reply fast

```
 /* Paste your code here (You may delete these lines if not writing code)#include<stdio.h>
#include<iostream>
#include<limits.h>

using namespace std;

int main()
{
  int N,i,j;
  cin>>N;
  int A[N],LAP[N],diff[N];
  for(i=0;i<N;i++)
     cin>>A[i];
  for(i=0;i<N;i++)
  {
      LAP[i]=1;
```

see more

1 • Reply • Share ›

Mahdi Belbasi ➦ piki • 2 years ago

your soloution is also same as this and is O(n^2)

because of 2 fors nested :

for(i=1;i<n;i++) {="" for(j="0;j&lt;i;j++)" {="" if(lap[i]="=1)" {="" lap[i]="2;" diff[i]="A[i]-A[j];" }="" else="" if(a[i]-a[j]="=diff[j]" &&="" lap[j]+1="">LAP[i])
{
LAP[i]=LAP[j]+1;
diff[i]=A[i]-A[j];

}
}
}

this is also O(n^2)

^ | v • Reply • Share ›

geek • 2 years ago

Do we really need a 2D matrix to store the result. As we are filling the matrix only when we are able to find AP.

^ | v • Reply • Share ›

Nishant Mahajan • 2 years ago

haha

^ | v • Reply • Share ›

hh • 2 years ago

Acc to your code L[n-1][n-1] = 2..please correct it.
It should be 1.

1 ^ | v • Reply • Share ›

prabhu • 2 years ago

here is a solution with O(n^2) time complexity and O(n) space complexity.

```
int lenghtOfLongestAP(int *arr, int n)
{
int aux1[n], aux2[n];

aux1[0] = 1;
memset(aux2, 0xff, sizeof(int)*n);

for(int i=1; i < n; ++i)
{
int longind = -1;
aux1[i] = 1;
for(int j = 0; j arr[j])
{
if (aux2[j] == -1 || arr[i] == (arr[j] + aux2[j]))
{
if (aux1[i] < (aux1[j] + 1))
```

see more

^ | v • Reply • Share ›

Dave • 2 years ago

A more generic solution. Please reply in case of errors.

```
#include <iostream>
using namespace std;
//
int LLAP(int a[], int n)
{
    int diff_max = a[n-1] - a[0];
    int max_Len=1;
    // Len[i][j] is length of AP with last element is i and diff betn adj elements is j
    int Len[n][diff_max+1];
    for (int i=0; i<n; i++)
    {
        for (int j=0; j<=diff_max; j++)
            Len[i][j] = 1;
    }
    for (int i=1; i<n; i++)
```

see more

Reply • Share ›

parbays ➦ Dave • 2 years ago

Your max_diff can go pretty huge, storing such large table will not be prudent.

Reply • Share ›

Dave • 2 years ago

```
#include <stdio.h>
//
int LLAP(int a[], int n)
{
    int diff_max = a[n-1] - a[0];
    int max_Len=1;
    // Len[i][j] is length of AP with last element is i and diff between adj elements is j
    int Len[n][diff_max+1];
    for (int i=0; i<n; i++)
    {
        for (int j=0; j<=diff_max; j++)
            Len[i][j] = 1;
    }
    for (int i=1; i<n; i++)
    {
        for (int k=0; k<i; k++)
        {
```

see more

^ | v • Reply • Share ›

Dhiren • 2 years ago

As I always try to draw the recursion tree for a problem before applying dynamic programming concepts to it. So I tried for this problem too. Here is my solution, but I don’t think there is overlapping sub problem property in here, but may be the optimal substructure property is there, not sure?? If someone?? Reviews it then please point out. Though I don’t like to copy the whole bunch of code in here instead I like to write the logic but in this case as I am not able to copy the recursion tree in this comment so copying the code sample. May be there is some mistake, but I have tasted with different inputs, results is correct. If someone?? Can point out??

```
#include "stdafx.h"

namespace longest_arithmatic_progression
{
        int lap(int a[], int n, int prev_index, int diff)
        {
                if(n==1 || n==2)
                        return n;
                int lap_count = 0, max_lap_count =0;
```

see more

^ | v • Reply • Share ›

Mohan ➦ Dhiren • 2 years ago

Your solution runs at O(maxdiff*n^2). Even if you use binary search to find the next index of the AP, the complexity is unaffected.

^ | v • Reply • Share ›

RiTZ • 2 years ago

You can also point out a "n^2 log n" solution if you use 2 loops with a binary search instead of a linear search.

^ | v • Reply • Share ›

hh • 2 years ago

@geeksforgeeks:In the portion of your article :"Given a sorted set, find if there exist three elements in Arithmetic Progression or not", how your complexity is linear??I think it is O(n^2). Say you have 2 3 5 6 10.Then one such triplet is 2 6 10.So clearly outer loop runs n-2 times and inner loop n-2 times.

So it should be O(n^2 - 2*n + 4) i.e. O(n^2) Please take a look.Thanks

• Reply • Share ›

csj hh • 2 years ago
how to efficiently find i and k for a 'given j' ? We can find i and k in linear time using following simple algorithm. (For given j)
• Reply • Share ›

Silent • 2 years ago
Without auxillary space..

int FindAP(int *a,int size)
{
int i,j,fix;
int len,max_len,diff,left,right;

len = max_len = 0;

for(fix=1;fix=0 && j 2*a[fix])
i--;
else if(a[i]+a[j] =0)
{
if(left-a[i] == diff)
{
left = a[i];
len++;
}

see more

• Reply • Share ›

csj • 2 years ago
Given code doesn't work correctly when given set has duplicates. following edit resolves the issue.

```
 .
.
.
// Change i and k to fill more L[i][j] values for
// current j
int temp = s[i];
i--;
while(i >= 0 && s[i] == temp) {
L[i][j] = L[j][k] + 1;
i--;
}
k++;
.
.
.
```

Thanks for the problem!

• Reply • Share ›

Savan Popat • 2 years ago

Thank you very much,

• Reply • Share ›

asitdhal • 2 years ago

I have a small doubt here

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

Why are we assigning 2 when set[i]+set[k] is greater ?

• Reply • Share ›

Hitesh • 2 years ago

It could have been better explained (in terms of how to reuse overlapping subproblems and underlying optimal substructure). By the way, thanks for updating ourselves by such a fresh problem!

• Reply • Share ›

Dhiren Hitesh • 2 years ago

I agree with Hitesh. As you cant think the solution of a problem directly in line of dynamic programming. You always try to do it with recursion and then make the solution efficient by applying dynamic programming paradigm to it.

```
/* Paste your code here (You may delete these lines if not writing code) */
```

• Reply • Share ›

sourabh • 2 years ago

Another solution would be:-

1.) Find the minimum and maximum in the sorted set.

2.) Let L(i, j, d) be the length of the sequence starting at a[i] and ending at a[j] with a common difference of d
3.) The maximum value in the table L(I, J, d) gives the maximum subsequence length for common difference d.
4.) Construct the tables for a common difference of 0 to (max- min) found in the first step
5.) Choose the maximum from all the tables
6.) Complexity is O((max - min) * n * n) or O(n*n) since max- min is constant

• Reply • Share ›

Vu Duc Minh sourabh • a year ago

6) It is not correct. For example max-min = 10^9 while n=100. The value of "max-min" is a parameter, not a constant.

• Reply • Share ›

Vikrant • 2 years ago

Edit this

……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++).

1 • Reply • Share ›

GeeksforGeeks Vikrant • 2 years ago

@vikrant: Thanks for pointing this out. We have updated the algorithm.

2 • Reply • Share ›

Tux Ansari GeeksforGeeks • 2 years ago

(set[i] + set[k] < 2*set[j])? k++ : i–;

should be i--;

2 • Reply • Share ›

Subscribe Add Disqus to your site Privacy

-
-
-
-
    - Interview Experiences
    - Advanced Data Structures
    - Dynamic Programming
    - Greedy Algorithms
    - Backtracking
    - Pattern Searching
    - Divide & Conquer
    - Mathematical Algorithms
    - Recursion
    - Geometric Algorithms
-

- ## Popular Posts

    - All permutations of a given string
    - Memory Layout of C Programs
    - Understanding “extern” keyword in C
    - Median of two sorted arrays
    - Tree traversal without recursion and without stack!
    - Structure Member Alignment, Padding and Data Packing
    - Intersection point of two Linked Lists
    - Lowest Common Ancestor in a BST.
    - Check if a binary tree is BST or not
    - Sorted Linked List to Balanced BST
- Follow @GeeksforGeeks

- ## Recent Comments

    - lt_k

i need help for coding this function in java...

Java Programming Language · 2 hours ago

- Piyush

  What is the purpose of else if (recStack[*i])...

  Detect Cycle in a Directed Graph · 2 hours ago

- Andy Toh

  My compile-time solution, which agrees with the...

  Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 2 hours ago

- lucy

  because we first fill zero in first col and...

  Dynamic Programming | Set 29 (Longest Common Substring) · 2 hours ago

- lucy

  @GeeksforGeeks i don't n know what is this long...

  Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 3 hours ago

- manish

  Because TAN is not a subsequence of RANT. ANT...

  Given two strings, find if first string is a subsequence of second · 3 hours ago

•