GeeksforGeeks

A computer science portal for geeks

GeeksQuiz

- Home
- Algorithms
- DS
- GATE
- Interview Corner
- Q&A
- C
- C++
- Java
- Books
- Contribute
- Ask a Q
- About

Array
Bit Magic
C/C++
Articles
GFacts
Linked List
MCQ
Misc
Output
String
Tree
Graph

Dynamic Programming | Set 32 (Word Break Problem)

Given an input string and a dictionary of words, find out if the input string can be segmented into a space-separated sequence of dictionary words. See following examples for more details.
This is a famous Google interview question, also being asked by many other companies now a days.

```
Consider the following dictionary
{ i, like, sam, sung, samsung, mobile, ice,
  cream, icecream, man, go, mango}

Input:  ilike
Output: Yes
The string can be segmented as "i like".

Input:  ilikesamsung
```

```
Output: Yes
The string can be segmented as "i like samsung" or "i like sam sung".
```

Recursive implementation:
The idea is simple, we consider each prefix and search it in dictionary. If the prefix is present in dictionary, we recur for rest of the string (or suffix). If the recursive call for suffix returns true, we return true, otherwise we try next prefix. If we have tried all prefixes and none of them resulted in a solution, we return false.

We strongly recommend to see **substr** function which is used extensively in following implementations.

```
// A recursive program to test whether a given string can be segmented into
// space separated words in dictionary
#include <iostream>
using namespace std;

/* A utility function to check whether a word is present in dictionary or not
  An array of strings is used for dictionary.  Using array of strings for
  dictionary is definitely not a good idea. We have used for simplicity of
  the program*/
int dictionaryContains(string word)
{
    string dictionary[] = {"mobile","samsung","sam","sung","man","mango",
                           "icecream","and","go","i","like","ice","cream"};
    int size = sizeof(dictionary)/sizeof(dictionary[0]);
    for (int i = 0; i < size; i++)
        if (dictionary[i].compare(word) == 0)
           return true;
    return false;
}

// returns true if string can be segmented into space separated
// words, otherwise returns false
bool wordBreak(string str)
{
    int size = str.size();

    // Base case
    if (size == 0)  return true;

    // Try all prefixes of lengths from 1 to size
    for (int i=1; i<=size; i++)
    {
        // The parameter for dictionaryContains is str.substr(0, i)
        // str.substr(0, i) which is prefix (of input string) of
        // length 'i'. We first check whether current prefix is in
        // dictionary. Then we recursively check for remaining string
        // str.substr(i, size-i) which is suffix of length size-i
        if (dictionaryContains( str.substr(0, i) ) &&
            wordBreak( str.substr(i, size-i) ))
            return true;
    }
```

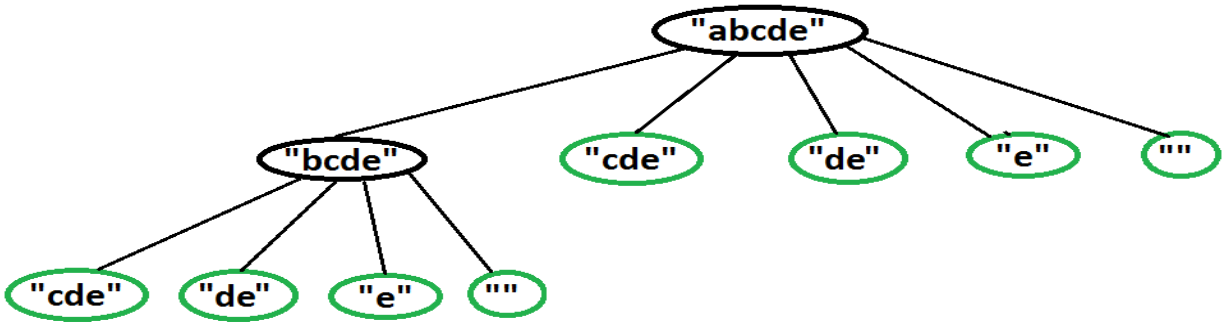
```
    // If we have tried all prefixes and none of them worked
    return false;
}

// Driver program to test above functions
int main()
{
    wordBreak("ilikesamsung")? cout <<"Yes\n": cout << "No\n";
    wordBreak("iiiiiiii")? cout <<"Yes\n": cout << "No\n";
    wordBreak("")? cout <<"Yes\n": cout << "No\n";
    wordBreak("ilikelikeimangoiii")? cout <<"Yes\n": cout << "No\n";
    wordBreak("samsungandmango")? cout <<"Yes\n": cout << "No\n";
    wordBreak("samsungandmangok")? cout <<"Yes\n": cout << "No\n";
    return 0;
}
```

Output:

```
Yes
Yes
Yes
Yes
Yes
No
```

Dynamic Programming

Why Dynamic Programming? The above problem exhibits overlapping sub-problems. For example, see the following partial recursion tree for string “abcde” in worst case.



Partial recursion tree for input string "abcde". The subproblems encircled with green color are overlapping subproblems

```
// A Dynamic Programming based program to test whether a given string can
// be segmented into space separated words in dictionary
#include <iostream>
#include <string.h>
using namespace std;

/* A utility function to check whether a word is present in dictionary or not
  An array of strings is used for dictionary.  Using array of strings for
  dictionary is definitely not a good idea. We have used for simplicity of
  the program*/
```

```
int dictionaryContains(string word)
{
    string dictionary[] = {"mobile","samsung","sam","sung","man","mango",
                           "icecream","and","go","i","like","ice","cream"};
    int size = sizeof(dictionary)/sizeof(dictionary[0]);
    for (int i = 0; i < size; i++)
        if (dictionary[i].compare(word) == 0)
           return true;
    return false;
}

// Returns true if string can be segmented into space separated
// words, otherwise returns false
bool wordBreak(string str)
{
    int size = str.size();
    if (size == 0)   return true;

    // Create the DP table to store results of subroblems. The value wb[i]
    // will be true if str[0..i-1] can be segmented into dictionary words,
    // otherwise false.
    bool wb[size+1];
    memset(wb, 0, sizeof(wb)); // Initialize all values as false.

    for (int i=1; i<=size; i++)
    {
        // if wb[i] is false, then check if current prefix can make it true.
        // Current prefix is "str.substr(0, i)"
        if (wb[i] == false && dictionaryContains( str.substr(0, i) ))
            wb[i] = true;

        // wb[i] is true, then check for all substrings starting from
        // (i+1)th character and store their results.
        if (wb[i] == true)
        {
            // If we reached the last prefix
            if (i == size)
                return true;

            for (int j = i+1; j <= size; j++)
            {
                // Update wb[j] if it is false and can be updated
                // Note the parameter passed to dictionaryContains() is
                // substring starting from index 'i' and length 'j-i'
                if (wb[j] == false && dictionaryContains( str.substr(i, j-i)
                    wb[j] = true;

                // If we reached the last character
                if (j == size && wb[j] == true)
                    return true;
            }
        }
    }
```

```
    /* Uncomment these lines to print DP table "wb[]"
     for (int i = 1; i <= size; i++)
        cout << " " << wb[i]; */

    // If we have tried all prefixes and none of them worked
    return false;
}

// Driver program to test above functions
int main()
{
    wordBreak("ilikesamsung")? cout <<"Yes\n": cout << "No\n";
    wordBreak("iiiiiiii")? cout <<"Yes\n": cout << "No\n";
    wordBreak("")? cout <<"Yes\n": cout << "No\n";
    wordBreak("ilikelikeimangoiii")? cout <<"Yes\n": cout << "No\n";
    wordBreak("samsungandmango")? cout <<"Yes\n": cout << "No\n";
    wordBreak("samsungandmangok")? cout <<"Yes\n": cout << "No\n";
    return 0;
}
```

Output:

```
Yes
Yes
Yes
Yes
Yes
No
```

Exercise:
The above solutions only finds out whether a given string can be segmented or not. Extend the above Dynamic Programming solution to print all possible partitions of input string. See extended recursive solution for reference.

Examples:

```
Input: ilikeicecreamandmango
Output:
i like ice cream and man go
i like ice cream and mango
i like icecream and man go
i like icecream and mango

Input: ilikesamsungmobile
Output:
i like sam sung mobile
i like samsung mobile
```

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

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

Niraj • a month ago

We can do a slight modification to the code which will avoid un-neccessary looping.

Existing Code snippet:

```
for (int j = i+1; j <= size; j++)
{
// Update wb[j] if it is false and can be updated
// Note the parameter passed to dictionaryContains() is
// substring starting from index 'i' and length 'j-i'
if (wb[j] == false && dictionaryContains( str.substr(i, j-i) ))
wb[j] = true;

// If we reached the last character
if (j == size && wb[j] == true)
return true;
}
```

In this part of the code if can can insert the line , i = j then it will work better.

New Code Snippet :

Mission Peace • a month ago

Check out my video on this qs

https://www.youtube.com/watch?...

• Reply • Share ›

Prakhar • a month ago

I tried to use Tries but failed.
#1. "aaaaaaa", ["aaaa", "aaa"] needs longest prefix
#2. "abcd", ["a", "abc", "b", "cd"] - needs other than longer.

My mind blows away -

http://ideone.com/NjHiD5

Any suggestions with minimal changes to what trie return.

Thnx in advance.

• Reply • Share ›

Truong Khanh Nguyen • 2 months ago

My discussion and java source code is here http://www.capacode.com/?p=335

```
private static String EMPTY = "";
private static String SPACE = " ";

public static String breakingWords(Set<string> dictionary, String s) {
// compute shortest, longest word
int minLength = Integer.MAX_VALUE;
int maxLength = Integer.MIN_VALUE;
for (String word : dictionary) {
if (word.length() > maxLength) {
maxLength = word.length();
}

if (word.length() < minLength) {
minLength = word.length();
}
}
```

see more

• Reply • Share ›

Aspire • 3 months ago

I am unable to conclude if the following method and above method are equally optimal

boolean isBreakable(String sentence,Set<string> dict)

{

boolean isSeg[]=new boolean[sentence.length()+1]; //isSet[i]=true if string(i,end) is segmentable.

for(int i=0;i<isseg.length;i++) isseg[i]="false;" isseg[sentence.length()]="true;" for(int="" i="sentence.length()-1;i">=0;i--)

{

if(dict.contains(sentence.substring(i)))

{

isSeg[i]=true;

}

if(isSeg[i]==false)

{

for(int j=i+1;j<sentence.length();j++) {="" if(dict.contains(sentence.substring(i,j))="" &&="" isseg[j])="" {="" isseg[i]="true;" break;="" }="" }="" }="" }="" return="" isseg[0];="" }="">

• Reply • Share ›

Aspire Aspire • 3 months ago

http://ideone.com/zQueIQ

same code as above but formatted

• Reply • Share ›

codegeeek • 3 months ago

Cannot see any overlapping in the recursion program.

• Reply • Share ›

Vib • 4 months ago

Dynamic program comes into effect here when we stored intermediate results (set wb[j] to true) for intermediate dictionary words. These are picked by wb[i] check

• Reply • Share ›

abhay kumar • 4 months ago

Hi Can any one please explain why are we calculating suffix length as size-i.

```
wordBreak( str.substr(i, size-i) ))
```

Can't we replace size-i by size.

• Reply • Share ›

Vib abhay kumar • 4 months ago

No, we know already that str.substr(0,i) prefix is dictionary word and we want to recursively check for all the possible suffixes with the above code. Also substr second parameter is length of the suffix.

• Reply • Share ›

abhay kumar Vib • 4 months ago

Thanks, I got my ans. I thought "size - i" as end index.

In java for getting sub string we need to pass start and end index. But in C++ we need to give starting index and length for substring.

1 • Reply • Share ›

Vikas Gupta • 6 months ago

We can do this using Trie and backtracking also.

Store all the words in Trie. Start from the first char and check whether the word is in trie or not. If yes then check for remaining string else check by extending the character. In some case it might happen you have found a word in trie but there is also one bigger word from that. e,g. {“Sam”, “Samsung} , Assume "Sung" is not in the array, so it will search Sam but when it will try to search remaining string i.e. sung, it will return false. So, in that case you backtrack to the previous word found (“Sam") and again check word by extending the char.

5 • Reply • Share ›

anuj Vikas Gupta • 4 months ago

time complexity??

• Reply • Share ›

Karshit Jaiswal • 7 months ago

http://ideone.com/TUMovr

Explanation commented.

• Reply • Share ›

Karshit Jaiswal • 7 months ago

http://changhaz.wordpress.com/...

a really good code with simple explanation.

3 • Reply • Share ›

aa1992 • 7 months ago

how to print all the words with blank spaces with dynamic programming?

• Reply • Share ›

codeexperiment • 7 months ago

Can it be solved by below in c#

public bool CanWordBreak(string word)

{

string[] arrayOfWords =

{

"mobile","samsung","sam","sung","man","mango",

"icecream","and","go","i","like","ice","cream"

};

// likesamsungk

// untill you don't get a match , keep on iterating

int i = 0;

see more

• Reply • Share ›

Harman Patel • 7 months ago

Dynamic Programming Solution in java:

public boolean wordBreak(String s, Set<string> dict) {
boolean[] ar=new boolean[s.length()+1];
ar[0]=true;
for(int i=1; i<=s.length(); i++){
for(int j=0; j<i; j++){="" if(ar[j]="" &&="" dict.contains(s.substring(j,i))){="" ar[i]="true;" }="" }="" }="" return="" ar[s.length()];="">

2 • Reply • Share ›

Guest • 8 months ago

what is the output for string s = "ilikeansamsung" ..?
is it true or false..? string contains word "an" which is not present in dictionary.

• Reply • Share ›

Priyal Rathi • 8 months ago

Another Dynamic Programming solution:

For each char store the start point of the substr (present in dict) which will end at that character and then go ahead breaking into words with new_startpoint= current_endpoint+1, if we reach the end of string without successfull breaking then again start from prev start point (DP[start-1]) and again start traversing without breaking at the points where we broke previously (as that leaded to unsuccessfull work break)

Link: http://ideone.com/gVcNZO

• Reply • Share ›

Hemant • 9 months ago

if question is :
Given a string find the number of meaningful words (which add to original length of the string) which could be formed from the string (A function called isWord() was provided which had tell you if the word was a dictionary word).
for eg.
programmerit forms:
pro+gram+merit
program+merit
programmer+it
pro+grammer+it

my solution is : Correct me if i'm wrong

```
#include <iostream>
#include <set>
#include <string>
#include <cstring>
using namespace std;
```

see more

• Reply • Share ›

anonymous ➚ Hemant • 9 months ago

your code is not working for programmeritit

• Reply • Share ›

Hemant ➚ anonymous • 9 months ago

there is small change in workBreak(). Check it out. i think it works now.

```
int wordBreak(string str)
{
int size = str.size();
if (size == 0) return 0;
int wb[size+1];
memset(wb, 0, sizeof(wb));
```

```
memset(wb, 0, sizeof(wb));

for (int i=1; i<=size; i++) {
if (dictionaryContains( str.substr(0, i) )) wb[i]++;
if (wb[i]) {
for (int j = i+1; j <= size; j++) {
if (dictionaryContains( str.substr(i, j-i))) {
wb[j] += wb[i];
}
}
}
}
return wb[size];
}
```

• Reply • Share ›

Anurag Singh • 9 months ago

DP solution to print possible partitions (Keeping track of matching words start and end):
http://ideone.com/tAjQvF

• Reply • Share ›

HT ➚ Anurag Singh • 9 months ago

Your codes failed with word "abcd" with dict "a","ab","bc","c","d"

• Reply • Share ›

Anurag Singh ➚ HT • 9 months ago

Thanks. Yes. I see that my code will not print all possible word breaking combinations.
Will post right code soon.

• Reply • Share ›

geekyandgirly • 10 months ago

I found this link that explain exactly the DP algorithm implemented above.
http://www.cs.berkeley.edu/~va...

4 • Reply • Share ›

Karshit Jaiswal • 10 months ago

Simple Recursive Implementation which counts and also prints the meaningfull words in string using a dictionary.
http://ideone.com/76aMxs

• Reply • Share ›

Gaurav Gupta ➚ Karshit Jaiswal • 10 months ago

the problem is if the input string can be separated into words that belongs to the given

dictionary of words. Your implementation is just finding out the words but it should return false/no if it is not able to find any substring from input string that doesn't belong to the dictionary. Like for the string that you have chosen it should return false/no.

• Reply • Share ›

Karshit Jaiswal ➚ Gaurav Gupta • 10 months ago

Yes, I just shared the solution to one more version of this problem.
As I have mentioned in my prev post, anyways thanks.

• Reply • Share ›

AlienOnEarth • a year ago

Cant we simply use Trie or Suffix Tree?

1 • Reply • Share ›

GS • a year ago

http://ideone.com/f8yfuE

Here I have written a code that gives the min number of spaces required. I must be missing something as the algo works in O(n).

Please let me know what test cases I have missed.Its working fine with given cases

• Reply • Share ›

Guest • a year ago

There are no overlapping sub problems in this program. Most of the commenters indicate the same. Need to make sure the code is fixed..

1 • Reply • Share ›

dmr • a year ago

Hi
I was searching for other solutions for this problem on net and found this(http://thenoisychannel.com/201... particular approach in a comment:

Build a directed graph consisting of vertices labeled 0, 1, 2, …, n,
where n is the length of the string, where there is an edge from k to j
if and only if there is a dictionary word of length j-k starting at
position k in the string.
This can be done in O(n^2).
Solutions to the problem then correspond to paths through the graph
from 0 to n. Use something like Dijkstra's algorithm to find a minimal
path in O(n^2), which corresponds to a solution that uses the smallest
number of dictionary words to exactly cover the string.

Now, that comment is very old, so not sure if someone will reply there. But does anyone here

think this approach will work? I am particularly doubtful about being able to create that graph in O(n^2). Any comments ?

2 • Reply • Share ›

asde dmr • 3 months ago

Yes, you can do that in O(n^2).

• Reply • Share ›

CodeCode54 • a year ago

I saw a lot of comments about people trying to see sub optimal structure for DP or the need for memoization. I was also skeptical about it first but after some try I have some clarity.

Suppose the string is: "ilovesamsung"

Dict: [i, love, loves, samsung]

String size is 12 so boolean[13]

First iteration i:1
str.substr(0, i) = i present in dictionary so the array is

b[1] = t all false

no we iterate with j from i+1 (2) to 13

l not in dict

li not in dict

---

see more

2 • Reply • Share ›

Brandon White CodeCode54 • a year ago

Thanks for this breakdown -- it was actually very helpful. It would be great if you could correct some of the mistakes, such as 'like' vs. 'love' and you skipped a letter in "i becomes 4 "ilk"" and "i becomes 5 "ilke""

• Reply • Share ›

Ali Nahid • a year ago

I'm sorry .. But I am seeing how the recursion method is causes overlapping ? can someone please explain ?

As far as I could understand .. because I am doing a return after the recursive call .. I will never overlap. If I did not do a "return" then for loop will iterate after the recursive call.

For Example:

"applepie" for dict {"apple","pie"} .. in the no recursion will be called unless i find apple. once i find apple then there will a recursion with pie. After that, I find prefix "pie" another recursion will be called with empty "" (well, yes, this is unnecessary call and with an if statement this can be avoided.. but it wont make any difference in the Big-O analysis cause cost of that if statement and cost of recursion is the same.. right ?[please correct me if i am wrong]).
So, where's the overlapping ?

• Reply • Share ›

pm ➚ Ali Nahid • a year ago

I will try to explain. The provided picture doesn't show the full recursion tree for a specific example. It shows all the possibilities for two nodes and as you can see there are some possibilities overlapping.

This probably didn't really help so here is something more specific. Let's use your example, but add the words a, p, ap.

applepie = a+pplepie OR ap+plepie OR apple+pie

The substring pplepie will be analyzed first. Obviously it will eventually return false. But one of the calls (the first one) will be for the substring plepie. Which overlaps with the second option ap+plepie.

Hope this makes at least a little sense.

• Reply • Share ›

Ali Nahid ➚ pm • a year ago

Thanks for the explanation. However, I am still not convinced. I understand what you're saying. But There's only 1 directed recursion going on. So I'm pretty confident that there's no overlapping.
Recursion like fibonacci or BST can have overlapping subproblems because of their tree structured traversal where one child can be visited multiple times.
For example : 1 recursion for left and 1 recursion on right.

BUT in this case it's a linear subproblem and i dont think there's any way they will be overlapping.

1 • Reply • Share ›

Paparao Veeragandham • a year ago

I think this problem not using the small problems results to find out Bigger problems also. Means it was not constructing the small solutions followed by Bigger solutions. (It was not following some hierarchy)

• Reply • Share ›

Paparao Veeragandham ➚ Paparao Veeragandham • a year ago

// Sample code in DP

```
// Sample code in D?
int i = 0
for(; i < s.length(); i++){
for(int j = 0; j > i; j++){
if(T[j] && dictionary.contains(s.substring(j + 1, i)){
T[i] = true;
break;
}
}
if(dictionary.contains(s.substring(0, i + 1)){
T[i] = true;
}
}
```

Create a table T[N] which says that T[i] is true if the substring S[0...i) can be broken into a sequence of valid words. T[i] is true iff i = 0 OR (there exists a j, 0<=j<i, where="" t[j]="" is="" true="" and="" s[j...i)="" is="" a="" valid="" word).="">

• Reply • Share ›

Paparao Veeragandham ➦ Paparao Veeragandham • a year ago

Sorry Above solution does not work for input string : samxyzsam.

• Reply • Share ›

Jeremy • a year ago

You Memoization part just simply doesn't take effect! what's the point of using wb[i] since it's top-bottom recursion.

• Reply • Share ›

Guest • a year ago

```
#include <iostream>
#include <string>
using namespace std;

bool isDict(string str)
{
    string Dict[] = {"mobile","samsung","sam","sung","man","mango","icecream","and","go","i
    int size = sizeof(Dict)/sizeof(Dict[0]);
    for(int i = 0 ; i < size; i++)
    {
        if(Dict[i].compare(str) == 0)
            return true;
    }
    return false;
}
```

```
bool wordBreak(string str)
{
```

see more

1 • Reply • Share ›

Guest • a year ago

Authors of geeksforgeeks, please give explanation of an approach in addition to the code. In this concrete problem it's not obvious how DP works. You only described overlapping subproblems. But I don't understand the actual idea!

10 • Reply • Share ›

Vijay Daultani • a year ago

```
#include <iostream>
#include <string>
#include <stdlib.h>
using namespace std;
#define ALPHABET 26
#define CHAR_TO_INDEX(c) (int)((int)c - (int)'a')
#define INDEX_TO_CHAR(i) (char)(i + (int)'a')

struct TrieNode
{
    TrieNode* children[ALPHABET];
    bool isLeaf;
};

struct TrieTree
{
    struct TrieNode* root;
};
```

see more

• Reply • Share ›

SAGAR JHOBALIA • 2 years ago

The problem can be memoized with a Mat[n][n] matrix.
Mat[i][j] = max(a[i-1][i-1]+dict(Word[i-j]), a[i-1][j]) ; j>=i
Mat[n][n] will hold the max number of words possible (therefore samsung will always be return sam sung). This can be backtracked in O(n) to retrieve the splits.

• Reply • Share ›

Guest • 2 years ago

The solution will not work for

The solution will not work for,
Diction {a, ajax}
Input: aajax
Here for this input outer for loop finds 'a' w[1] = true
then inner for loop finds second 'a' w[2]= true
Then it comes out of inner forloop as 'jax' is not in diction.
Now outer forloop with i as 2 checks w[2] is true so it enters inner forloop and which anyways return false and this keeps repeating till end.
Apologize that I haven't tried running the program with inputs and could be wrong :-)

• Reply • Share ›

its_dark Guest • 2 years ago

You are wrong !
See the examples for which the output is shown.
Print their W[] arrays.

• Reply • Share ›

Load more comments

- 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