Is Subsequence Leetcode Solution

Difficulty Level Easy

Frequently asked in Adobe Amazon Bloomberg Facebook Google Microsoft Pinterest Yandex

Tags algorithms coding Dynamic Programming Greedy Interview interviewprep LeetCode LeetCodeSolutions String Views 17

Recent Posts

Range Sum Query 2D – Immutable Leetcode Solution

Partition Labels LeetCode Solution

Flipping an Image LeetCode Solution

Concatenation of Array LeetCode Solution

Fibonacci Number LeetCode Solution

Algorithm Interview Questions

Array Interview Questions

C Programming Tutorial

Table of Contents

Problem Statement

Examples

Dynamic Programming Interview Questions
Git Tutorial
Graph Interview Questions
Hashing Interview Questions
Interview Experience
Interview Questions
Java Tutorial
JavaScript Tutorial
LeetCode Solutions
LinkedList Interview Questions
Matrix Interview Questions
PHP Tutorial
Python Basics
Queue Interview Questions
R Programming Tutorial
Selenium Tutorial
Sorting Interview Questions
Spring Boot Tutorial
SQL Interview Questions
SQL Tutorial
Stack Interview Questions
String Interview Questions
Technical Interview Questions
Testing Tutorial

Problem Statement

In this problem, we are given two different strings. The goal is to find out whether the first string is a subsequence of the second.

Examples

Input

```
first string = "abc"
second string = "mnagbcd"
```

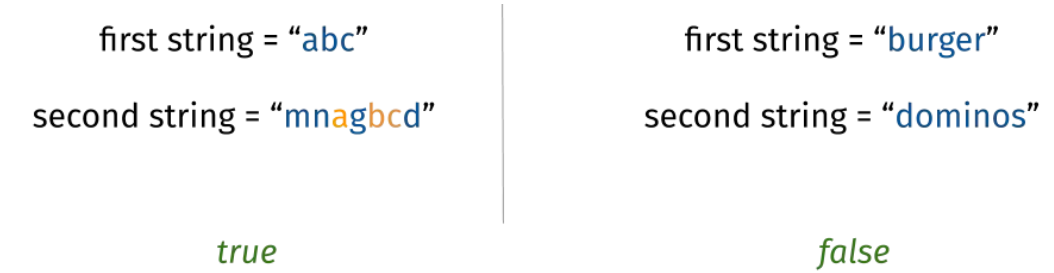
Output

```
true
```

```
first string = "burger"
second string = "dominos"
```

Output

```
false
```



Approach(Recursive)

This is easy to see that we can start matching the strings from their ends. If the characters at the last of the strings match, then we have a reduced sub-problem of finding whether the two strings that can be obtained from the original ones **after** dropping their last characters follow the subsequence criteria. If the end

This can be done recursively by passing the indices of the strings as parameters to compare the characters at every recursive call.

Algorithm

1. Index *i* denotes the last index of the first string and *j* denotes the last index of the second string in recursive calls
2. If *i* == *-1*:
    1. We have completely traversed the first string, return **true**
3. If *j* == *-1*:
    1. We have completely traversed the first string, return **false**
4. If *first_string[i] == second_string[j]:*
    1. call recursive functions with indices *(i – 1 , j – 1)*

---

5. Return the result of recursive function with indices *(i , j – 1)*

Implementation of Is Subsequence Leetcode Solution

C++ Program

Code

```
#include <bits/stdc++.h>
using namespace std;

string S , T;

bool checkSubsequence(int i , int j)
```

```
    if(j == -1)
        return false;
    if(S[i] == T[j])
        return checkSubsequence(i - 1 , j - 1);
    return checkSubsequence(i , j - 1);
}

bool isSubsequence(string s, string t)
{
    S = s , T = t;
    return checkSubsequence((int)s.size() - 1 , (int)t.size() - 1);
}

int main()
{
    string s = "abc";
    string t = "mnagbcd";

    cout << (isSubsequence(s , t) ? "true" : "false") << '\n';
    return 0;
}
```

Java Program

Code

```
class is_subsequence
{
    static String S , T;

    public static void main(String args[])
    {
        String a = "abc" , b = "mnagbcd";
        System.out.println((isSubsequence(a , b) ? "true" : "false"));
    }

    static boolean checkSubsequence(int i , int j)
    {
        if(i == -1)
            return true;
```

```
            return checkSubsequence(i - 1 , j - 1);
        return checkSubsequence(i , j - 1);
    }

    static boolean isSubsequence(String s, String t)
    {
        S = s;
        T = t;
        return checkSubsequence((int)s.length() - 1 , (int)t.length() -
    }
}
```

Output

```
true
```

Complexity Analysis of Is Subsequence Leetcode Solution

Time Complexity

O(min(N, M)) as we need to recur until either of the strings is traversed. **N** and **M** are the lengths of strings.

O(min(M , N) in the worst case due to recursive calls.

Approach(Two-Pointers)

We can use the above technique, iteratively by maintaining two pointers *i* and *j* to store the last indices of respective strings. We can iterate until either of them becomes zero. If *i* (pointer of the first string) is greater than or equal to 0, we have not been able to traverse the first string completely, and hence, it is not a subsequence of the second. Else, we return true.

Algorithm

1. Initialize two pointers *i and j* storing the last indices of both the strings.
2. While *i >= 0 and j >= 0:*
    1. If first_string[i] == second_string[j]:
        1. *i—* , *j—*
    2. Else
        1. *j—*

3. If *i* >= 0:

Implementation of Is Subsequence Leetcode Solution

C++ Program

Code

```
#include <bits/stdc++.h>
using namespace std;

string S , T;

bool isSubsequence(string s , string t)
{
    int i = s.size() , j = t.size();

    i-- , j--;

    while(i >= 0 && j >= 0)
    {
        if(s[i] == t[j])
            i-- , j--;
        else
            j--;
    }

    if(i >= 0)
        return false;
    return true;
}

int main()
{
    string s = "abc";
    string t = "mnagbcd";

    cout << (isSubsequence(s , t) ? "true" : "false") << '\n';
    return 0;
```

Java Program

Code

```
class is_subsequence
{
    static String S , T;

    public static void main(String args[])
    {
        String a = "abc" , b = "mnagbcd";

        System.out.println((isSubsequence(a , b) ? "true" : "false"));
    }

    static boolean isSubsequence(String s , String t)
    {
        int i = s.length() , j = t.length();

        i--;
        j--;

        while(i >= 0 && j >= 0)
        {
            if(s.charAt(i) == t.charAt(j))
            {
                i--;
                j--;
            }
            else
                j--;
        }

        if(i >= 0)
            return false;
        return true;
    }
}
```

Output

Solution

Time Complexity

O(min(M , N)) as we keep iterating until *i or j* becomes zero. M and N are the lengths of the strings.

Space Complexity

O(1) as we use constant space in the memory.

LeetCode Solutions

Adobe, algorithms, Amazon, Bloomberg, coding, Dynamic Programming, Easy, Facebook, Google, Greedy, Interview, interviewprep, LeetCode, LeetCodeSolutions, Microsoft, Pinterest, String, Yandex

‹ Jewels and Stones Leetcode Solution

› Keyboard Row Leetcode Solution