Find if a string is interleaved of two other strings | DP-33

Last Updated : 13 Apr, 2024

Given three strings A, B and C. Write a function that checks whether C is an interleaving of A and B. C is said to be interleaving A and B, if it contains all and only characters of A and B and order of all characters in individual strings is preserved.

Example:

```
Input: strings: "XXXXZY", "XXY", "XXZ"
Output: XXXXZY is interleaved of XXY and XXZ
The string XXXXZY can be made by
interleaving XXY and XXZ
String:    XXXXZY
String 1:    XX Y
String 2:  XX  Z
Input: strings: "XXY", "YX", "X"
Output: XXY is not interleaved of YX and X
XXY cannot be formed by interleaving YX and X.
The strings that can be formed are YXX and XYX
```

Method 1: Recursion.

Approach: A simple solution is discussed here: Check whether a given string is an interleaving of two other given string.

The simple solution will also work if the strings A and B have some common characters. For example, let the given string be A and the other strings be B and C. Let A = “XXY”, string B = “XXZ” and string C = “XXZXXXY”. Create a recursive function that takes parameters A, B, and C. To handle all cases, two possibilities need to be considered.

1. If the first character of C matches the first character of A, we move one character ahead in A and C and recursively check.

2. If the first character of C matches the first character of B, we move one character ahead in B and C and recursively check.

If any of the above function returns true or A, B and C are empty then return true else return false.

Thanks to Frederic for suggesting this approach.

C++ C Java Python3 C# JavaScript

```python
def isInterleaved(A, B, C):
    # Base Case: If all strings are empty
    if not A and not B and not C:
        return True

    # If C is empty and any of the two strings is not empty
    if not C:
        return False

    # If the first character of C matches the first character of A,
    # then recursively check the rest of A and C
    r1, r2 = False, False
    if A and C[0] == A[0]:
        r1 = isInterleaved(A[1:], B, C[1:])

    # If the first character of C matches the first character of B,
    # then recursively check the rest of B and C
    if B and C[0] == B[0]:
        r2 = isInterleaved(A, B[1:], C[1:])

    # If none of the above conditions are met, then C cannot be
interleaved
    # from A and B
    return r1 or r2


# A function to run test cases
def test(A, B, C):

    if (isInterleaved(A, B, C)):
        print(C, "is interleaved of", A, "and", B)
    else:
        print(C, "is not interleaved of", A, "and", B)

# Driver Code
if __name__ == '__main__':
    test("XXY", "XXZ", "XXZXXXY")
    test("XY", "WZ", "WZXY")
    test("XY", "X", "XXY")
    test("YX", "X", "XXY")
    test("XXY", "XXZ", "XXXXZY")
```

Complexity Analysis:

- **Time Complexity:** O(2^n), where n is the length of the given string.
- **Space Complexity:** O(1).
  The space complexity is constant.

Method 2: Dynamic Programming.
Approach: The above recursive solution certainly has many overlapping sub-problems. For example, if we consider A = “XXX”, B = “XXX” and C = “XXXXXX” and draw a recursion tree, there will be many overlapping subproblems. Therefore, like any other typical Dynamic Programming problems, we can solve it by creating a table and store results of sub-problems in a **bottom-up manner**. The top-down approach of the above solution can be modified by adding a Hash Map.

Algorithm:

1. Create a DP array (matrix) of size M*N, where m is the size of the first string and n is the size of the second string. Initialize the matrix to false.
2. If the sum of sizes of smaller strings is not equal to the size of the larger string then return false and break the array as they cant be the interleaved to form the larger string.
3. Run a nested loop the outer loop from 0 to m and the inner loop from 0 to n. Loop counters are i and j.
4. If the values of i and j are both zeroes then mark dp[i][j] as true. If the value of i is zero and j is non zero and the j-1 character of B is equal to j-1 character of C the assign dp[i][j] as dp[i][j-1] and similarly if j is 0 then match i-1 th character of C and A and if it matches then assign dp[i][j] as dp[i-1][j].
5. Take three characters x, y, z as (i-1)th character of A and (j-1)th character of B and (i + j – 1)th character of C.
6. if x matches with z and y does not match with z then assign dp[i][j] as dp[i-1][j] similarly if x is not equal to z and y is equal to z then assign dp[i][j] as dp[i][j-1]
7. if x is equal to y and y is equal to z then assign dp[i][j] as bitwise OR of dp[i][j-1] and dp[i-1][j].

8. return value of dp[m][n].

Thanks to Abhinav Ramana for suggesting this method of implementation.

C++ Java Python3 C# JavaScript

```python
# A Dynamic Programming based python3 program
# to check whether a string C is an interleaving
# of two other strings A and B.

# The main function that returns true if C is
# an interleaving of A and B, otherwise false.
def isInterleaved(A, B, C):

    # Find lengths of the two strings
    M = len(A)
    N = len(B)

    # Let us create a 2D table to store solutions of
    # subproblems. C[i][j] will be true if C[0..i + j-1]
    # is an interleaving of A[0..i-1] and B[0..j-1].
    # Initialize all values as false.
    IL = [[False] * (N + 1) for i in range(M + 1)]

    # C can be an interleaving of A and B only of sum
    # of lengths of A & B is equal to length of C.
    if ((M + N) != len(C)):
        return False

    # Process all characters of A and B
    for i in range(0, M + 1):
        for j in range(0, N + 1):

            # two empty strings have an empty string
            # as interleaving
            if (i == 0 and j == 0):
                IL[i][j] = True

            # A is empty
            elif (i == 0):
                if (B[j - 1] == C[j - 1]):
                    IL[i][j] = IL[i][j - 1]

            # B is empty
            elif (j == 0):
                if (A[i - 1] == C[i - 1]):
                    IL[i][j] = IL[i - 1][j]

            # Current character of C matches with
            # current character of A, but doesn't match
            # with current character of B
            elif (A[i - 1] == C[i + j - 1] and
                  B[j - 1] != C[i + j - 1]):
                IL[i][j] = IL[i - 1][j]
```

```
                # Current character of C matches with
                # current character of B, but doesn't match
                # with current character of A
                elif (A[i - 1] != C[i + j - 1] and
                      B[j - 1] == C[i + j - 1]):
                    IL[i][j] = IL[i][j - 1]

                # Current character of C matches with
                # that of both A and B
                elif (A[i - 1] == C[i + j - 1] and
                      B[j - 1] == C[i + j - 1]):
                    IL[i][j] = (IL[i - 1][j] or IL[i][j - 1])

        return IL[M][N]

    # A function to run test cases
    def test(A, B, C):

        if (isInterleaved(A, B, C)):
            print(C, "is interleaved of", A, "and", B)
        else:
            print(C, "is not interleaved of", A, "and", B)

    # Driver Code
    if __name__ == '__main__':
        test("XXY", "XXZ", "XXZXXXY")
        test("XY", "WZ", "WZXY")
        test("XY", "X", "XXY")
        test("YX", "X", "XXY")
        test("XXY", "XXZ", "XXXXZY")

    # This code is contributed by ashutosh450
```

Output:

```
XXZXXXY is not interleaved of XXY and XXZ
WZXY is interleaved of XY and WZ
XXY is interleaved of XY and X
XXY is not interleaved of YX and X
XXXXZY is interleaved of XXY and XXZ
```

See this for more test cases.

Complexity Analysis:

- **Time Complexity:** O(M*N).
  Since a traversal of the DP array is needed, so the time complexity is O(M*N).
- **Space Complexity:** O(M*N).
  This is the space required to store the DP array.

https://youtu.be/WBXy-sztEwI

Method 3: Dynamic Programming(Memoization)

Approach: We can make a matrix where rows and columns represent the characters of the string A and B. If C is the interleaved string of A and B then there exists a path from the top left of the Matrix to the bottom right. That is if we can go from index 0,0 to n,m while matching characters of all A and B with C then C is interleaved of A and B.

Let A be “XXY”, B be “XXZ” and C be “XXZXXY” then the path would look something like this:

	X	X	Y	
X	1	0	0	0
X	1	0	0	0
Z	1	0	0	0
	1	1	1	R

let us consider one more example, let A be “ABC”, B be “DEF” and C be “ADBECF”, then the path would look something like this:

	D	E	F	
A	1	0	0	0
B	1	1	0	0
C	0	1	1	0
	0	0	1	R

If there exists a path through which we can reach R, then C is the interleaved strings of A and B.

Algorithm:

1. We will first create a matrix dp to store the path since one path can be explored multiple times, the Matrix index dp[i][j] will store if there exists a path from this index or not.

2. If we are at i'th index of A and j'th index of B and C[i+j] matches both A[i] and B[j] then we explore both the paths that are we will go right and down i.e. we will explore index i+1,j and j+1,i.

3. If C[i+j] only matches with A[i] or B[j] then we will go just down or right respectively that is i+1,j or i,j+1.

C++ Java Python3 C# JavaScript

```python
# A Python Memoization program
# to check whether a string C is
# an interleaving of two other
# strings A and B.

# Declare dp array
dp = [[0]*101]*101

# This function checks if there exist a valid path from 0,0 to n,m
def dfs(i, j, A, B, C):

    # If path has already been calculated from this index
    # then return calculated value.
    if(dp[i][j]!=-1):
        return dp[i][j]

    # If we reach the destination return 1
    n,m=len(A),len(B)
    if(i==n and j==m):
        return 1

    # If C[i+j] matches with both A[i] and B[j]
    # we explore both the paths

    if (i<n and A[i]==C[i + j] and j<m and B[j]==C[i + j]):
        # go down and store the calculated value in x
        # and go right and store the calculated value in y.
        x = dfs(i + 1, j, A, B, C)
        y = dfs(i, j + 1, A, B, C)

        # return the best of both.
        dp[i][j] = x|y
        return dp[i][j]
```

```
        # If C[i+j] matches with A[i].
        if (i < n and A[i] == C[i + j]):
            # go down
            x = dfs(i + 1, j, A, B, C)

            # Return the calculated value.
            dp[i][j] = x
            return dp[i][j]

        # If C[i+j] matches with B[j].
        if (j < m and B[j] == C[i + j]):
            y = dfs(i, j + 1, A, B, C)

            # Return the calculated value.
            dp[i][j] = y
            return dp[i][j]

        # if nothing matches we return 0
        dp[i][j] = 0
        return dp[i][j]

    # The main function that
    # returns true if C is
    # an interleaving of A
    # and B, otherwise false.
    def isInterleaved(A, B, C):

        # Storing the length in n,m
        n = len(A)
        m = len(B)

        # C can be an interleaving of
        # A and B only of the sum
        # of lengths of A & B is equal
        # to the length of C.

        if((n+m)!=len(C)):
            return 0

        # initializing dp array with -1
        for i in range(n+1):
            for j in range(m+1):
                dp[i][j]=-1

        # calling and returning the answer
        return dfs(0,0,A,B,C)


    # A function to run test cases
    def test(A, B, C):

        if (isInterleaved(A, B, C)):
            print(C, "is interleaved of", A, "and", B)
        else:
            print(C, "is not interleaved of", A, "and", B)

    # Driver Code
    if __name__ == '__main__':
```

```
        test("XXY", "XXZ", "XXZXXXY")
        test("XY", "WZ", "WZXY")
        test("XY", "X", "XXY")
        test("YX", "X", "XXY")
        test("XXY", "XXZ", "XXXXZY")
        test("ACA", "DAS", "DAACSA")

    # This code is contributed by Pushpesh Raj.
```

Output

```
XXZXXXY is not interleaved of XXY and XXZ
WZXY is interleaved of XY and WZ
XXY is interleaved of XY and X
XXY is not interleaved of YX and X
XXXXZY is interleaved of XXY and XXZ
DAACSA is interleaved of ACA and DAS
```

Complexity Analysis:

Time Complexity: **O(m*n).**

This is the worst case of time complexity, if the given strings contain no common character matching with C then the time complexity will be O(n+m).
Space Complexity: **O(m*n).**

This is the space required to store the DP array.

The above method and implementation are contributed by Anirudh Singh.

GeeksforGeeks 80

Next Article

Print all interleavings of given two strings

Similar Reads

Check if given string can be formed by two other strings or their…

Given a string str and an array of strings arr[], the task is to check if the given string can be formed by any of the string pairs from the array or their...

13 min read

Minimum swaps required between two strings to make one string strictly…

Given two strings A and B of length M and N respectively, the task is to find the minimum swapping of two characters required to make string A lexicographicall...

9 min read

Python3 Program for Check if given string can be formed by two other strin…

Given a string str and an array of strings arr[], the task is to check if the given string can be formed by any of the string pair from the array or their permutation...

4 min read

Construct the Largest Palindromic String which lies between two other…

Given two strings A and B of length N which contains only lowercase characters, find the lexicographically largest string C of length N which contains only...

9 min read

Given two numbers as strings, find if one is a power of other

Given two large numbers as strings, find if one is the power of another. Examples: Input : a = "374747", b = "52627712618930723" Output : YES Explanation :...

11 min read

View More Articles

Article Tags : DSA Dynamic Programming Strings FactSet +4 More

Practice Tags : FactSet Google Microsoft Yahoo +3 More