**Strings Interleaving**

**We'll cover the following**

* + - [Problem Statement](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK#problem-statement)
    - [Basic Solution](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK#basic-solution)
      * [Code](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK#code)
    - [Top-down Dynamic Programming with Memoization](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK#top-down-dynamic-programming-with-memoization)
      * [Code](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK#code-2)
    - [Bottom-up Dynamic Programming](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK#bottom-up-dynamic-programming)
      * [Code](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK#code-3)

**Problem Statement**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK#problem-statement)

Give three strings ‘m’, ‘n’, and ‘p’, write a method to find out if ‘p’ has been formed by interleaving ‘m’ and ‘n’. ‘p’ would be considered interleaving ‘m’ and ‘n’ if it contains all the letters from ‘m’ and ‘n’ and the order of letters is preserved too.

**Example 1:**

Input: m="abd", n="cef", p="abcdef"  
Output: true  
Explanation: 'p' contains all the letters from 'm' and 'n' and preserves their order too.

**Example 2:**

Input: m="abd", n="cef", p="adcbef"  
Output: false  
Explanation: 'p' contains all the letters from 'm' and 'n' but does not preserve the order.

**Example 3:**

Input: m="abc", n="def", p="abdccf"  
Output: false  
Explanation: 'p' does not contain all the letters from 'm' and 'n'.

**Example 4:**

Input: m="abcdef", n="mnop", p="mnaobcdepf"  
Output: true  
Explanation: 'p' contains all the letters from 'm' and 'n' and preserves their order too.

**Basic Solution**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK#basic-solution)

The problem follows the [Longest Common Subsequence](https://www.educative.io/collection/page/5668639101419520/5633779737559040/5657535201673216) (LCS) pattern and has some similarities with [Subsequence Pattern Matching](https://www.educative.io/collection/page/5668639101419520/5633779737559040/5718922095493120/).

A basic brute-force solution could be to try matching ‘m’ and ‘n’ with ‘p’ one letter at a time. Let’s assume mIndex, nIndex, and pIndex represent the current indexes of ‘m’, ‘n’, and ‘p’ strings respectively. Therefore, we have two options at any step:

1. If the letter at mIndex matches with the letter at pIndex, we can recursively match for the remaining lengths of ‘m’ and ‘p’.
2. If the letter at nIndex matches with the letter at ‘pIndex’, we can recursively match for the remaining lengths of ‘n’ and ‘p’.

**Code**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK#code)

Here is the code:

* [Java](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK)
* [JS](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK)
* [Python3](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK)
* [C++](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK)

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

  }

  private boolean findSIRecursive(String m, String n, String p, int mIndex, int nIndex, int pIndex) {

    // if we have reached the end of the all the strings

    if(mIndex == m.length() && nIndex == n.length() && pIndex == p.length())

      return true;

    // if we have reached the end of 'p' but 'm' or 'n' still have some characters left

    if(pIndex == p.length())

      return false;

    boolean b1=false, b2=false;

    if(mIndex < m.length() && m.charAt(mIndex) == p.charAt(pIndex))

      b1 = findSIRecursive(m, n, p, mIndex+1, nIndex, pIndex+1);

    if(nIndex < n.length() && n.charAt(nIndex) == p.charAt(pIndex))

      b2 = findSIRecursive(m, n, p, mIndex, nIndex+1, pIndex+1);

    return b1 || b2;

  }

  public static void main(String[] args) {

    SI si = new SI();

    System.out.println(si.findSI("abd", "cef", "abcdef"));

    System.out.println(si.findSI("abd", "cef", "adcbef"));

    System.out.println(si.findSI("abc", "def", "abdccf"));

    System.out.println(si.findSI("abcdef", "mnop", "mnaobcdepf"));

  }

}





RUN

SAVERESET

The time complexity of the above algorithm is exponential O(2^{m+n})*O*(2​*m*+*n*​​), where ‘m’ and ‘n’ are the lengths of the two interleaving strings. The space complexity is O(m+n)*O*(*m*+*n*), the value that is used to store the recursion stack.

**Top-down Dynamic Programming with Memoization**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK#top-down-dynamic-programming-with-memoization)

This problem can have overlapping subproblems only when there are some common letters between ‘m’ and ‘n’ at the same index. Because whenever we hit such a scenario, we get an option to match with any one of them.

The three changing values in our recursive function are the three indexes mIndex, nIndex, and pIndex. Therefore, we can store the results of all the subproblems in a three-dimensional array. Alternately, we can use a hash-table whose key would be a string (mIndex + “|” + nIndex + “|” + pIndex).

**Code**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK#code-2)

Here is the code:

* [Java](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK)
* [JS](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK)
* [Python3](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK)
* [C++](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK)

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

13

12

32

33

34

35

11

10

9

8

7

6

5

4

    // if we have reached the end of the all the strings

    if(mIndex == m.length() && nIndex == n.length() && pIndex == p.length())

      return true;

    // if we have reached the end of 'p' but 'm' or 'n' still has some characters left

    if(pIndex == p.length())

      return false;

    String subProblemKey = mIndex + "-" + nIndex + "-" + pIndex;

    if(!dp.containsKey(subProblemKey)) {

      boolean b1=false, b2=false;

      if(mIndex < m.length() && m.charAt(mIndex) == p.charAt(pIndex))

        b1 = findSIRecursive(dp, m, n, p, mIndex+1, nIndex, pIndex+1);

      if(nIndex < n.length() && n.charAt(nIndex) == p.charAt(pIndex))

        b2 = findSIRecursive(dp, m, n, p, mIndex, nIndex+1, pIndex+1);

      dp.put(subProblemKey, b1 || b2);

    int mIndex, int nIndex, int pIndex) {

    }

    return dp.get(subProblemKey);

  }

  private boolean findSIRecursive(Map<String, Boolean> dp, String m, String n, String p,

  }

    return findSIRecursive(dp, m, n, p, 0, 0, 0);

    Map<String, Boolean> dp = new HashMap<>();

  public Boolean findSI(String m, String n, String p) {

class SI {





RUN

SAVERESET

**Bottom-up Dynamic Programming**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK#bottom-up-dynamic-programming)

Since we want to completely match ‘m’ and ‘n’ (the two interleaving strings) with ‘p’, we can use a two-dimensional array to store our results. The lengths of ‘m’ and ‘n’ will define the dimensions of the result array.

As mentioned above, we will be tracking separate indexes for ‘m’, ‘n’ and ‘p’, so we will have the following options for every value of mIndex, nIndex, and pIndex:

1. If the character m[mIndex] matches the character p[pIndex], we will take the matching result up to mIndex-1 and nIndex.
2. If the character n[nIndex] matches the character p[pIndex], we will take the matching result up to mIndex and nIndex-1.

String ‘p’ will be interleaving strings ‘m’ and ‘n’ if any of the above two options is true. This is also required as there could be some common letters between ‘m’ and ‘n’.

So our recursive formula would look like:

1

2

3

4

5

dp[mIndex][nIndex] = false

if m[mIndex] == p[pIndex]

  dp[mIndex][nIndex] = dp[mIndex-1][nIndex]

if n[nIndex] == p[pIndex]

 dp[mIndex][nIndex] |= dp[mIndex][nIndex-1]





Let’s draw this visually:

a b c d e f Interleaving string a b d T c e f

If 'm' and 'n' are empty, we can always find an empty string that will be interleaving them

**1** of 16

**Code**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK#code-3)

Here is the code for our bottom-up dynamic programming approach:

* [Java](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK)
* [JS](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK)
* [Python3](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK)
* [C++](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMM68YXRwEK)

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

14

12

13

7

8

9

10

11

1

2

3

4

5

6

        if(mIndex==0 && nIndex==0)

          dp[mIndex][nIndex] = true;

        // if 'm' is empty, we need to check the interleaving with 'n' only

        else if (mIndex==0 && n.charAt(nIndex-1) == p.charAt(mIndex+nIndex-1))

          dp[mIndex][nIndex] = dp[mIndex][nIndex-1];

        // if 'n' is empty, we need to check the interleaving with 'm' only

        else if (nIndex==0 && m.charAt(mIndex-1) == p.charAt(mIndex+nIndex-1))

          dp[mIndex][nIndex] = dp[mIndex-1][nIndex];

        else {

          // if the letter of 'm' and 'p' match, we take whatever is matched till mIndex-1

          if(mIndex > 0 && m.charAt(mIndex-1) == p.charAt(mIndex+nIndex-1))

            dp[mIndex][nIndex] = dp[mIndex-1][nIndex];

          // if the letter of 'n' and 'p' match, we take whatever is matched till nIndex-1 too

          // note the '|=', this is required when we have common letters

          if(nIndex > 0 && n.charAt(nIndex-1) == p.charAt(mIndex+nIndex-1))

            dp[mIndex][nIndex] |= dp[mIndex][nIndex-1];

        }

        // if 'm' and 'n' are empty, then 'p' must have been empty too.

    for(int mIndex=0; mIndex<=m.length(); mIndex++) {

      for(int nIndex=0; nIndex<=n.length(); nIndex++) {

    // for the empty pattern, we have one matching

    if(m.length() + n.length() != p.length())

      return false;

class SI {

  public Boolean findSI(String m, String n, String p) {

    // dp[mIndex][nIndex] will be storing the result of string interleaving

    // up to p[0..mIndex+nIndex-1]

    boolean[][] dp = new boolean[m.length()+1][n.length()+1];





RUN

SAVERESET

The time and space complexity of the above algorithm is O(m\*n)*O*(*m*∗*n*), where ‘m’ and ‘n’ are the lengths of the two interleaving strings.

**MARK AS COMPLETED**

[**←    Back**](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/gx2QMvEorYY)

Edit Distance

[**Next    →**](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nVWonJzqVA)

Contact Us

Report an Issue

[Ask a Question](https://discuss.educative.io/c/grokking-dynamic-programming-patterns-for-coding-interviews-design-gurus/pattern-5-longest-common-substring-strings-interleaving" \t "_blank)