```
/**
 * Name: Andy
 * Date: November, 23, 2021
 * Class: Main
 * Description: A class demonstrating the Knuth Morris Pratt Algorithm
 */

// Imported Classes
import java.util.Scanner;
import java.util.ArrayList;

// Class
public class Main {

  // Main
  public static void main(String[] args) {

    // Prompt
-------------------------------------------------------------

    // Initialization
    String input;
    String target;
    Scanner in = new Scanner(System.in);

    // Input string prompt
    input = in.nextLine();

    // Target word prompt
    target = in.nextLine();

    // Algorithm
-------------------------------------------------------------

    // Initialization
    int[] table = new int[target.length()];
    ArrayList<Integer> indices = new ArrayList<Integer> ();

    // Algorithm, creates table for the kmp algorithm based on the target
string
    table(target, table, 1, 0);

    // Algorithm, collects the indices of the target word in the input
word
```

```
    kmp(table, indices, 0, 0, input, target);

    if (indices.size() == 0) indices.add(-1);

    // Output
--------------------------------------------------------------

    System.out.println(indices.get(0));

  } // End Main


   /**
    * Method: Recursively completes the Knuth Morris Pratt Algorithm to
locate the target string in the input string
    * @param int[] table, a pre made table based on the target word to
assist with optimization
    * @return void
    */
    public static void kmp (int[] table, ArrayList<Integer> indices, int
inputPointer, int targetPointer, String input, String target) {

      if (input.length() < target.length()) return;

      // Loop until all instances of the target word in the input word are
found
      while (inputPointer < input.length()) {

        // Initialization
        // inputPointer index character (of String input)
        String i = input.substring(inputPointer, inputPointer + 1);
        // targetPointer index character (of String target)
        String t = target.substring(targetPointer, targetPointer + 1);

        // Main computation
        // This algorithm runs from the left to the right of the input
string, constantly checking for matches to the target word. If there is a
match it will pursue it. If there is no match it will either simply move
on to the next index or, if it was already pursuing a match, it will also
change the pointer on the target word to look for any potential matches
that may have contained the previous characters analyzed.

        // If the input and target characters match, increase both
pointers by one
```

```
        if (i.equals(t)) {

          inputPointer++;
          targetPointer++;

          // If this is the final character of the target string, add an
entry into the indices arraylist to indicate that this index contains an
instance of the target word in the input string
          if (targetPointer == target.length()) {

            indices.add(inputPointer - targetPointer);

            // Alters target pointer based on table
            return;

          } // End if

        } else {

          // Alters target pointer based on table
          targetPointer = table[targetPointer];

          // If the target pointer is set to negative, this means it must
completely reset to the base value
          // Also means if if everything has been reset, move the
inputPointer forward to continue the search
          if (targetPointer < 0) {

            inputPointer++;
            targetPointer++;

          } // End if

        } // End if

      } // End while

    } // End kmp

    /**
    * Method: Creates a supporting table required for the operation of the
kmp algorithm
    * @param String target, the string being analyzed
    * @param int[] table, where the table will be created
    * @param int pos, the current position in the table being analyzed
    * @param int cnd, the current position in the target string being
compared to using pos
    * @return void
    */
    public static void table (String target, int[] table, int pos, int
cnd) {

      // Declaration
      table[0] = -1;

      // Loop until table is complete
      while (pos < target.length()) {

        // Initialization
        // Pos index character
        String p = target.substring(pos, pos + 1);
        // Cnd index character
        String c = target.substring(cnd, cnd + 1);

        // Main computation
        // This algorithm works on the basis that repeated character
chains in the target word may result in the word appearing in the input
string while already pursuing a match.The table ensures that the kmp
algorithm moves its pointers accurately so that every character in the
input string is only checked once but all matches are still found

        // If the targeted characters are equal, set the value in the
table of the pos index to the value in the table of the cnd index
        if (p.equals(c)) {

          table[pos] = table[cnd];

        } else {
        // If the targeted characters are not equal, set the value in the
table of the pos index to the value of cnd

          table[pos] = cnd;

          // Continually set the value of cnd to the value in the table of
the cnd index until either cnd is positive or cnd is equal to the value in
the table of the pos index
          while (cnd >= 0 && !p.equals(target.substring(cnd, cnd + 1))) {
            cnd = table[cnd];

          } // End while

        } // End if

        // Increase pos and cnd by 1
        pos++;
        cnd++;

      } // End while

    } // End table

} // End Class
```