

```

Codeium: Refactor | Explain
3 public class LongestCommonSubsequence {
4     //Given two sequences, find the length of longest subsequence present in both of them.
5     //A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.
6     //For example, "abc", "abg", "bdf", "aeg", "acefg", .. etc are subsequences of "abcdefg".
7     //So a string of length n has 2^n different possible subsequences.
8     //Example: s1 = "abcdgh"; s2 = "abedfhr" -> "abdh" -> 4
9     //Approach: Recursive
10
11
Codeium: Refactor | Explain | Generate Javadoc | X
12 public static void main(String[] args) {
13     String s1 = "abcdgh";
14     String s2 = "abedfhr";
15     System.out.println(lcs(s1, s2, s1.length(), s2.length()));
16 }
17
Codeium: Refactor | Explain | Generate Javadoc | X
18 public static int lcs(String s1, String s2, int m, int n) {
19     if (m == 0 || n == 0) {
20         return 0;
21     }
22
23     if (s1.charAt(m - 1) == s2.charAt(n - 1)) {
24         return 1 + lcs(s1, s2, m - 1, n - 1);
25     } else {
26         return Math.max(lcs(s1, s2, m - 1, n), lcs(s1, s2, m, n - 1));
27     }
28 }
29
30 //Approach: Memoization
Codeium: Refactor | Explain | X
31 public static int lcsMemo(String s1, String s2, int m, int n) {
32     int[][] dp = new int[m + 1][n + 1];
33     return lcsMemo(s1, s2, m, n, dp);
34 }
35
Codeium: Refactor | Explain | X
31 public static int lcsMemo(String s1, String s2, int m, int n) {
32     int[][] dp = new int[m + 1][n + 1];
33     return lcsMemo(s1, s2, m, n, dp);
34 }
35
Codeium: Refactor | Explain | Generate Javadoc | X
36 public static int lcsMemo(String s1, String s2, int m, int n, int[][] dp) {
37     if (m == 0 || n == 0) {
38         return 0;
39     }
40     if (dp[m][n] != 0) {
41         return dp[m][n];
42     }
43
44     if (s1.charAt(m - 1) == s2.charAt(n - 1)) {
45         dp[m][n] = 1 + lcsMemo(s1, s2, m - 1, n - 1, dp);
46         return dp[m][n];
47     } else {
48         dp[m][n] = Math.max(lcsMemo(s1, s2, m - 1, n, dp), lcsMemo(s1, s2, m, n - 1, dp));
49         return dp[m][n];
50     }
51 }
52 //Approach: Dynamic Programming

```

```

52 | //Approach: Dynamic Programming
53 | Codeium: Refactor | Explain | ✕
54 | public static int lcsDP(String s1, String s2) {
55 |     int m = s1.length();
56 |     int n = s2.length();
57 |
58 |     int[][] dp = new int[m + 1][n + 1];
59 |
60 |     //Fill the dp table in bottom up manner
61 |     for (int i = 1; i <= m ; i++) {
62 |         for (int j = 1; j <= n ; j++) {
63 |             if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
64 |                 //If the characters match, then add 1 to the result and move diagonally
65 |                 dp[i][j] = 1 + dp[i - 1][j - 1];
66 |             } else {
67 |                 //If the characters don't match, then take the max of the two results
68 |                 dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
69 |             }
70 |         }
71 |     }
72 |
73 |     return dp[m][n];
74 | }
75 |
76 |
77 |

```

## .LongestCommonSubsequence

```

1 package a1Dynamic.LCS;
2
3 Codeium: Refactor | Explain
4 public class LongestCommonSubsequenceDP {
5     //Given two sequences, find the length of longest subsequence present in both of them.
6     //A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.
7     //Approach: Dynamic Programming
8
9     Codeium: Refactor | Explain | Generate Javadoc | X
10    public static void main(String[] args) {
11        String s1 = "abcdgh";
12        String s2 = "abedfhr";
13        System.out.println(lcs(s1, s2));
14    }
15
16    Codeium: Refactor | Explain | Generate Javadoc | X
17    public static int lcs(String s1, String s2) {
18        int m = s1.length();
19        int n = s2.length();
20
21        int[][] dp = new int[m + 1][n + 1];
22
23        //Fill the dp table in bottom up manner
24        for (int i = 1; i <= m ; i++) {
25            for (int j = 1; j <= n ; j++) {
26                if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
27                    //If the characters match, then add 1 to the result and move diagonally
28                    dp[i][j] = 1 + dp[i - 1][j - 1];
29                } else {
30                    //If the characters don't match, then take the max of the two results
31                    dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
32                }
33            }
34        }
35
36        return dp[m][n];
37    }
38 }

```

## .LongestCommonSubsequenceDP

```

3 public class LongestCommonString01 {
4     // Given two strings 'X' and 'Y', find the length of the longest common substring.
5     // Input : X = "abcdxyz", y = "xyzabcd"
6     // Output : 4
7     // The longest common substring is "abcd" and is of length 4.
8
9     // Input : X = "zxabcdezy", y = "yzabcdez"    // Output : 6
10
11 Codeium: Refactor | Explain | Generate Javadoc | ✕
12 public static void main(String[] args) {
13     String s1 = "abcdxyz";
14     String s2 = "xyzabcd";
15     System.out.println(lcs(s1, s2));
16 }
17
18 Codeium: Refactor | Explain | Generate Javadoc | ✕
19 public static int lcs(String s1, String s2) {
20     int m = s1.length();
21     int n = s2.length();
22
23     int[][] dp = new int[m + 1][n + 1];
24
25     //Fill the dp table in bottom up manner
26     int max = 0;
27     for (int i = 1; i <= m ; i++) {
28         for (int j = 1; j <= n ; j++) {
29             if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
30                 //If the characters match, then add 1 to the result and move diagonally
31                 dp[i][j] = 1 + dp[i - 1][j - 1];
32                 max = Math.max(max, dp[i][j]);
33             } else {
34                 //If the characters don't match, then take the max of the two results
35                 dp[i][j] = 0;
36             }
37         }
38     }
39     return max;
40 }
41 }
42

```

.LongestCommonString01

```

Codeium: Refactor | Explain
3 public class PrintLongestCommonSubsequence02 {
4 //Given two sequences, print the longest subsequence present in both of them.
5 //A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.
6 //Approach: Dynamic Programming
7
Codeium: Refactor | Explain | Generate Javadoc | X
8 public static void main(String[] args) {
9     String s1 = "abcdgh";
10    String s2 = "abedfhr";
11    System.out.println(lcs(s1, s2));
12 }
13

```

```

Codeium: Refactor | Explain | Generate Javadoc | X
13 public static String lcs(String s1, String s2) {
14     int m = s1.length();
15     int n = s2.length();
16
17     int[][] dp = new int[m + 1][n + 1];
18
19     //Fill the dp table in bottom up manner
20     for (int i = 1; i <= m ; i++) {
21         for (int j = 1; j <= n ; j++) {
22             if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
23                 //If the characters match, then add 1 to the result and move diagonally
24                 dp[i][j] = 1 + dp[i - 1][j - 1];
25             } else {
26                 //If the characters don't match, then take the max of the two results
27                 dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
28             }
29         }
30     }
31
32     //Now, we need to print the longest common subsequence
33     //Start from the bottom right corner of the dp table
34     int i = m;
35     int j = n;
36     StringBuilder sb = new StringBuilder();
37     while (i > 0 && j > 0) {
38         //If the characters match, then add the character to the result and move diagonally
39         if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
40             sb.append(s1.charAt(i - 1));
41             i--;
42             j--;
43         } else {
44             //If the characters don't match, then move in the direction of the larger result
45             if (dp[i - 1][j] > dp[i][j - 1]) {
46                 i--;
47             } else {
48                 j--;
49             }
50         }
51     }
52
53     return sb.reverse().toString();
54 }
55

```

**.PrintLongestCommonSubsequence02**

**abdh**

```

3 public class ShortestCommonSupersequence03 {
4     //Given two strings str1 and str2, find the shortest string that has both str1 and str2 as subsequences.
5     // Examples : Input: str1 = "geek", str2 = "eke" Output: "geeke"
6     //Approach: Dynamic Programming
7
8     Codeium: Refactor | Explain | Generate Javadoc | ✕
9     public static void main(String[] args) {
10         // String s1 = "abcdgh";
11         // String s2 = "abedfhr";
12         String s1 = "geek";
13         String s2 = "eke";
14         // System.out.println(shortestCommonSupersequence(s1, s2));
15         System.out.println(printShortestCommonSupersequence(s1, s2));
16     }
17
18     Codeium: Refactor | Explain | Generate Javadoc | ✕
19     public static int shortestCommonSupersequence(String s1, String s2) {
20         int m = s1.length();
21         int n = s2.length();
22
23         int[][] dp = new int[m + 1][n + 1];
24
25         //Fill the dp table in bottom up manner
26         for (int i = 1; i <= m; i++) {
27             for (int j = 1; j <= n; j++) {
28                 if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
29                     //If the characters match, then add 1 to the result and move diagonally
30                     dp[i][j] = 1 + dp[i - 1][j - 1];
31                 } else {
32                     //If the characters don't match, then take the max of the two results
33                     dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
34                 }
35             }
36         }
37
38         return m + n - dp[m][n];
39     }
40
41     //print the shortest common supersequence
42     Codeium: Refactor | Explain | ✕
43     public static String printShortestCommonSupersequence(String s1, String s2) {
44         int m = s1.length();
45         int n = s2.length();
46
47         int[][] dp = new int[m + 1][n + 1];
48
49         //Fill the dp table in bottom up manner
50         for (int i = 1; i <= m; i++) {
51             for (int j = 1; j <= n; j++) {
52                 if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
53                     //If the characters match, then add 1 to the result and move diagonally
54                     dp[i][j] = 1 + dp[i - 1][j - 1];
55                 } else {
56                     //If the characters don't match, then take the max of the two results
57                     dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
58                 }
59             }
60         }
61
62         //print the shortest common supersequence
63         int i = m;
64         int j = n;
65         StringBuilder sb = new StringBuilder();
66         while (i > 0 && j > 0) {
67             if (s1.charAt(i - 1) == s2.charAt(j - 1)) { //if the characters match, then add the character to the result and move diagonally
68                 sb.append(s1.charAt(i - 1));
69                 i--;
70                 j--;
71             } else {
72                 if (dp[i - 1][j] > dp[i][j - 1]) { //if the character doesn't match, then move in the direction of the max result
73                     sb.append(s1.charAt(i - 1));
74                     i--;
75                 } else {
76                     sb.append(s2.charAt(j - 1));
77                     j--;
78                 }
79             }
80         }
81     }

```

```

80
81     while (i > 0) { //if there are any characters left in s1, then add them to the result
82         sb.append(s1.charAt(i - 1));
83         i--;
84     }
85
86     while (j > 0) { //if there are any characters left in s2, then add them to the result
87         sb.append(s2.charAt(j - 1));
88         j--;
89     }
90
91     return sb.reverse().toString();
92 }
93
94 //print the shortest common supersequence
95
96 }
97

```

**.ShortestCommonSupersequence03**

**geeke**

```

3 public class MinimumInsertionDeletionToConvertStrAToStrB04 {
4     //Given two strings 'str1' and 'str2' of size m and n respectively.
5     // The task is to remove/delete and insert the minimum number of characters from/in str1 to transform it into str2.
6     // It could be possible that the same character needs to be removed/deleted from one point of str1 and inserted to some another point.
7     // Example 1: Input : str1 = "heap", str2 = "pea"
8     // Output : Minimum Deletion = 2 and
9     //           Minimum Insertion = 1
10    // p and h deleted from heap
11    // Then, p is inserted at the beginning
12    // One thing to note, though p was required yet
13    // it was removed/deleted first from its position and
14    // Example 2: Input : str1 = "geeksforgeeks", str2 = "geeks" \ Output : Minimum Deletion = 8
15    //           Minimum Insertion = 0
16    // Explanation: String s1 = "geeksforgeeks" and s2 = "geeks" 1. Delete 7 characters from s1 so that string becomes "geeks"
17    //Approach: Dynamic Programming
18    Codeium: Refactor | Explain | X
19    public static void main(String[] args) {
20        String s1 = "heap";
21        String s2 = "pea";
22        System.out.println(minimumInsertionDeletionToConvertStrAToStrB(s1, s2));
23    }
24
25    Codeium: Refactor | Explain | Generate Javadoc | X
26    public static int minimumInsertionDeletionToConvertStrAToStrB(String s1, String s2) {
27        int m = s1.length();
28        int n = s2.length();
29
30        int[][] dp = new int[m + 1][n + 1];
31
32        //Fill the dp table in bottom up manner
33        for (int i = 1; i <= m ; i++) {
34            for (int j = 1; j <= n ; j++) {
35                if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
36                    //If the characters match, then add 1 to the result and move diagonally
37                    dp[i][j] = 1 + dp[i - 1][j - 1];
38                } else {
39                    //If the characters don't match, then take the max of the two results
40                    dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
41                }
42            }
43        }
44
45        return m + n - 2 * dp[m][n];
46    }
47 }
48

```

**.MinimumInsertionDeletionToConvertStrAToStrB04**



Codeium: Refactor | Explain

```
3 public class LongestPalindromicSubsequence05 {
4     //Given a sequence, find the length of the longest palindromic subsequence in it.
5     //As another example, if the given sequence is "BBABCBAB", then the output should be 7 as
6     // "BABCBAB" is the longest palindromic subsequence in it.
7     //"BBBBB" and "BBCBB" are also palindromic subsequences of the given sequence, but not the longest ones.
8     //Approach: Dynamic Programming
9
```

Codeium: Refactor | Explain | Generate Javadoc | X

```
10 public static void main(String[] args) {
11     String s1 = "BBABCBAB";
12     System.out.println(lcs(s1));
13     printLCS(s1);
14 }
15
```

Codeium: Refactor | Explain | Generate Javadoc | X

```
16 public static int lcs(String s1) {
17     int m = s1.length();
18     int n = m;
19
20     int[][] dp = new int[m + 1][n + 1];
21
22     //Fill the dp table in bottom up manner
23     for (int i = 1; i <= m ; i++) {
24         for (int j = 1; j <= n ; j++) {
25             if (s1.charAt(i - 1) == s1.charAt(n - j)) {
26                 //If the characters match, then add 1 to the result and move diagonally
27                 dp[i][j] = 1 + dp[i - 1][j - 1];
28             } else {
29                 //If the characters don't match, then take the max of the two results
30                 dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
31             }
32         }
33     }
34
35     return dp[m][n];
36 }
37
38
```

```

39 //Print the longest pallindromic subsequence
40 //Approach: Dynamic Programming
41
42 Codeium: Refactor | Explain | Generate Javadoc | ✕
43 public static void printLCS(String s1) {
44     int m = s1.length();
45     int n = m;
46
47     int[][] dp = new int[m + 1][n + 1];
48
49     //Fill the dp table in bottom up manner
50     for (int i = 1; i <= m ; i++) {
51         for (int j = 1; j <= n ; j++) {
52             if (s1.charAt(i - 1) == s1.charAt(n - j)) {
53                 //If the characters match, then add 1 to the result and move diagonally
54                 dp[i][j] = 1 + dp[i - 1][j - 1];
55             } else {
56                 //If the characters don't match, then take the max of the two results
57                 dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
58             }
59         }
60     }
61
62     //Print the longest pallindromic subsequence
63     int i = m;
64     int j = n;
65     String result = "";
66     while (i > 0 && j > 0) {
67         if (s1.charAt(i - 1) == s1.charAt(n - j)) {
68             result = s1.charAt(i - 1) + result;
69             i--;
70             j--;
71         } else {
72             if (dp[i - 1][j] > dp[i][j - 1]) {
73                 i--;
74             } else {
75                 j--;
76             }
77         }
78     }
79
80     System.out.println(result);
81 }
82
83 }

```

.LongestPallindromicSubsequence05

7

BACBCAB

```

Codeium: Refactor | Explain
3  ✓ public class MinInsDelToMakePallindrome06 {
4      //Given a string, find the minimum number of characters to be inserted to convert it to palindrome.
5      //For Example:
6      //ab: Number of insertions required is 1. bab or aba
7      //aa: Number of insertions required is 0. aa
8      //abcd: Number of insertions required is 3. dcbabcd
9      //abcd: Number of insertions required is 3. dcbabcd or abcacba or abcdcba or abcba
10     //abcda: Number of insertions required is 2. adcbcd a which is same as number of deletions in adcbcb a
11     //abcde: Number of insertions required is 4. edcbabcde
12     //Approach: Dynamic Programming
13
Codeium: Refactor | Explain | Generate Javadoc | ✕
14  ✓ public static void main(String[] args) {
15     String s1 = "abcd";
16     String s2 = new StringBuilder(s1).reverse().toString();
17     System.out.println("Min Insertions: " + minInsertions(s1, s2));
18 }
19
Codeium: Refactor | Explain | Generate Javadoc | ✕
20  ✓ public static int minInsertions(String s1, String s2) {
21     int m = s1.length();
22     int n = s2.length();
23
24     int[][] dp = new int[m + 1][n + 1];
25
26     //Fill the dp table in bottom up manner
27     for (int i = 1; i <= m ; i++) {
28         for (int j = 1; j <= n ; j++) {
29             if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
30                 //If the characters match, then add 1 to the result and move diagonally
31                 dp[i][j] = 1 + dp[i - 1][j - 1];
32             } else {
33                 //If the characters don't match, then take the max of the two results
34                 dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
35             }
36         }
37     }
38
39     return m - dp[m][n];
40 }
41
42

```

**.MinInsDelToMakePallindrome06**

**Min Insertions: 3**

```

Codeium: Refactor | Explain
3 public class PrintShortestSuperSequence07 {
4     //Given two strings 'X' and 'Y', yprin the shortest supersequence such that both strings are subsequence
5     // of the supersequence.
6     // Examples : Input: str1 = "geek", str2 = "eke" Output: "geeke"
7     //Approach: Dynamic Programming
8
9     Codeium: Refactor | Explain | Generate Javadoc | ✕
10    public static void main(String[] args) {
11        // String s1 = "abcdgh";
12        // String s2 = "abedfhr";
13        String s1 = "geek";
14        String s2 = "eke";
15        System.out.println(printShortestCommonSupersequence(s1, s2));
16    }
17
18    Codeium: Refactor | Explain | Generate Javadoc | ✕
19    public static String printShortestCommonSupersequence(String s1, String s2) {
20        int m = s1.length();
21        int n = s2.length();
22
23        int[][] dp = new int[m + 1][n + 1];
24
25        //Fill the dp table in bottom up manner
26        for (int i = 1; i <= m; i++) {
27            for (int j = 1; j <= n; j++) {
28                if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
29                    //If the characters match, then add 1 to the result and move diagonally
30                    dp[i][j] = 1 + dp[i - 1][j - 1];
31                } else {
32                    //If the characters don't match, then take the max of the two results
33                    dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
34                }
35            }
36        }
37
38        //print the shortest common supersequence
39        int i = m;
40        int j = n;
41        StringBuilder sb = new StringBuilder();
42        while (i > 0 && j > 0) {
43            if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
44                sb.append(s1.charAt(i - 1));
45                i--;
46                j--;
47            } else {
48                if (dp[i - 1][j] > dp[i][j - 1]) {
49                    sb.append(s1.charAt(i - 1));
50                    i--;
51                } else {
52                    sb.append(s2.charAt(j - 1));
53                    j--;
54                }
55            }
56        }
57
58        while (i > 0) {
59            sb.append(s1.charAt(i - 1));
60            i--;
61        }
62
63        while (j > 0) {
64            sb.append(s2.charAt(j - 1));
65            j--;
66        }
67
68        return sb.reverse().toString();
69    }

```

**.PrintShortestSuperSequence07**

geeke

```

Codeium: Refactor | Explain
3 public class LongestRepeatingSubsequence08 {
4     //Given a string, find the length of the longest repeating subsequence such that the two subsequences don't have
5     //same string character at the same position, i.e., any i'th character in the two subsequences shouldn't have the
6     //same index in the original string.
7     //Example: s1 = "aabebccdd" -> "abd" -> 3
8     //Example: s1 = "aabb" -> "ab" -> 2
9     //Approach: Dynamic Programming
10
11     Codeium: Refactor | Explain | Generate Javadoc | X
12     public static void main(String[] args) {
13         String s1 = "aabebccdd";
14         System.out.println(lrs(s1));
15     }
16
17     Codeium: Refactor | Explain | Generate Javadoc | X
18     public static int lrs(String s1) {
19         String s2 = s1;
20         int m = s1.length();
21         int n = s2.length();
22
23         int[][] dp = new int[m + 1][n + 1];
24
25         //Fill the dp table in bottom up manner
26         for (int i = 1; i <= m ; i++) {
27             for (int j = 1; j <= n ; j++) {
28                 if (s1.charAt(i - 1) == s2.charAt(j - 1) && i != j) {
29                     //If the characters match, then add 1 to the result and move diagonally
30                     dp[i][j] = 1 + dp[i - 1][j - 1];
31                 } else {
32                     //If the characters don't match, then take the max of the two results
33                     dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
34                 }
35             }
36         }
37
38         return dp[m][n];
39     }
40 }
41

```

## .LongestRepeatingSubsequence08

Codeium: Refactor | Explain

```
3 public class SequencePatternMatching09 {
4     //Given two strings s and t, return true if s is a subsequence of t, or false otherwise.
5     //A subsequence of a string is a new string that is formed from the original string by
6     // deleting some (can be none) of the characters without disturbing the
7     // relative positions of the
8     // remaining characters. (i.e., "ace" is a subsequence of "abcde" while "aec" is not).
9     //
10    //Example 1:
11    //Input: s = "abc", t = "ahbgdc"
12    //Output: true
13
```

Codeium: Refactor | Explain | Generate Javadoc | X

```
14 public static void main(String[] args) {
15     String s = "abc";
16     String t = "ahbgdc";
17     System.out.println(isSubsequence(s, t));
18 }
19
```

Codeium: Refactor | Explain | Generate Javadoc | X

```
20 public static boolean isSubsequence(String s, String t) {
21     int m = s.length();
22     int n = t.length();
23
24     int[][] dp = new int[m + 1][n + 1];
25
26     //Fill the dp table in bottom up manner
27     for (int i = 1; i <= m ; i++) {
28         for (int j = 1; j <= n ; j++) {
29             if (s.charAt(i - 1) == t.charAt(j - 1)) {
30                 //If the characters match, then add 1 to the result and move diagonally
31                 dp[i][j] = 1 + dp[i - 1][j - 1];
32             } else {
33                 //If the characters don't match, then take the max of the two results
34                 dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
35             }
36         }
37     }
38
39     return dp[m][n] == m;
40 }
41
42
```

.SequencePatternMatching09

true

Codeium: Refactor | Explain

```
3 public class MinDeletionsToMakePallindrome10 {
4     //Minimum number of deletions to make a string palindrome
5     //Given a string of size 'n'. The task is to remove or delete minimum number of
6     // characters from the string so that the resultant string is palindrome.
7     //Examples :
8     //
9     //Input : aebcbda
10    //Output : 2
11    //Remove characters 'e' and 'd'
12    //Resultant string will be 'abcba'
13    //which is a palindromic string
14
15    //Approach: Dynamic Programming
16    Codeium: Refactor | Explain | ✕
17    public static void main(String[] args) {
18        String s1 = "aebcbda";
19        String s2 = new StringBuilder(s1).reverse().toString();
20        System.out.println("Min Deletions: " + minDeletions(s1, s2));
21    }
22
23    Codeium: Refactor | Explain | Generate Javadoc | ✕
24    public static int minDeletions(String s1, String s2) {
25        int m = s1.length();
26        int n = s2.length();
27
28        int[][] dp = new int[m + 1][n + 1];
29
30        //Fill the dp table in bottom up manner
31        for (int i = 1; i <= m ; i++) {
32            for (int j = 1; j <= n ; j++) {
33                if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
34                    //If the characters match, then add 1 to the result and move diagonally
35                    dp[i][j] = 1 + dp[i - 1][j - 1];
36                } else {
37                    //If the characters don't match, then take the max of the two results
38                    dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
39                }
40            }
41        }
42
43        return m - dp[m][n];
44    }
45 }
```

**.MinDeletionsToMakePallindrome10**

**Min Deletions: 2**