Green University of Bangladesh

Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Spring, Year: 2023), B.Sc. in CSE (Day)

LAB REPORT NO #04

Course Title: Algorithm Lab

Course Code: CSE 208 Section: PC-213 DA

<u>Student Details</u>

	Name	ID
1	Pankaj Mahanta	213902002

Lab Date	**: 30/05/2023**
Submission Date	**: 11/06/2023**
Course Teacher's Name	**: Md. Sultanul Islam Ovi**

Lab Report Status

Marks: ………………………………… **Signature:.....................**

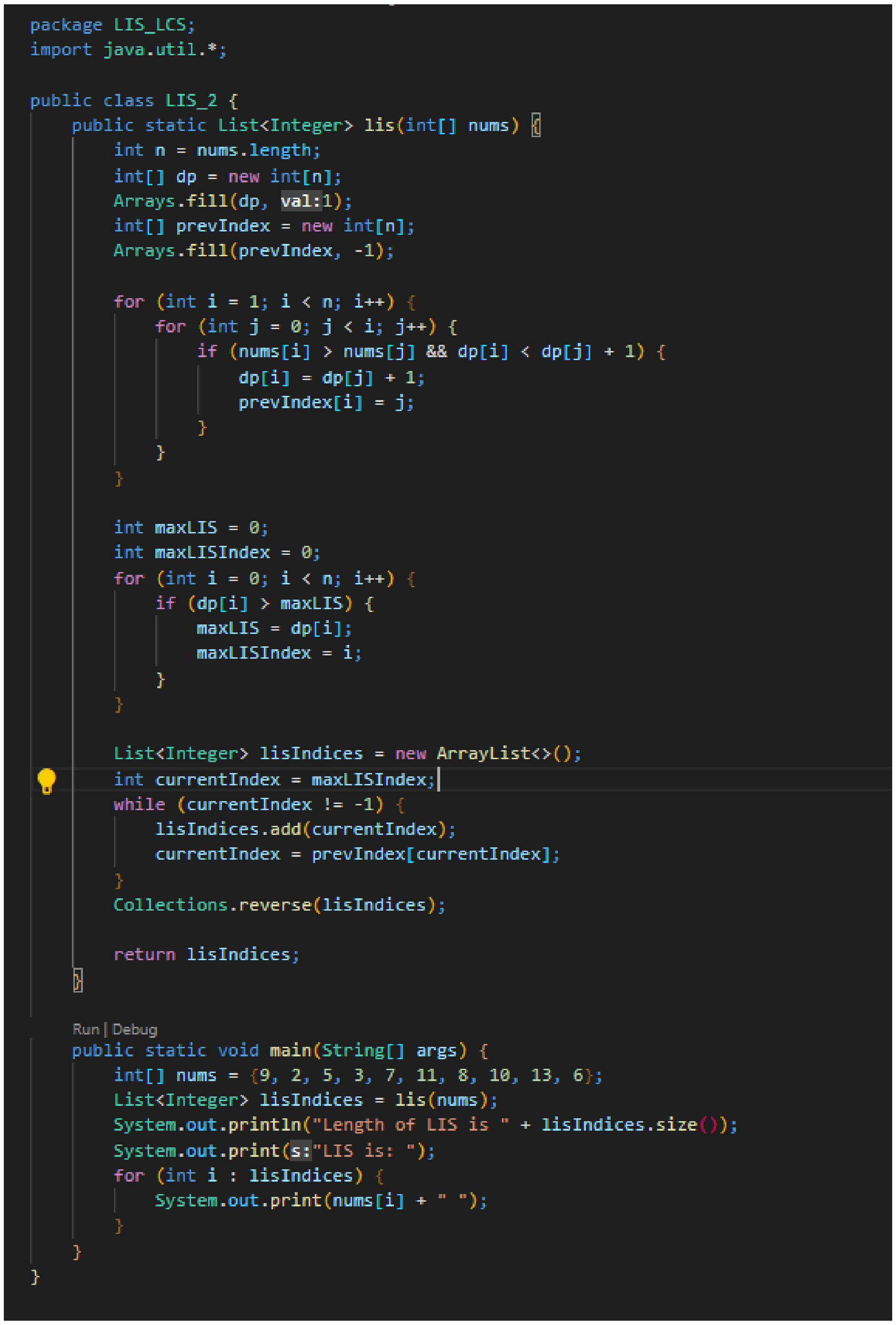
Comments:... **Date:..............................**

Task-1: Implement Longest increasing subsequence problem using DP technique. (LIS)

Explanation:

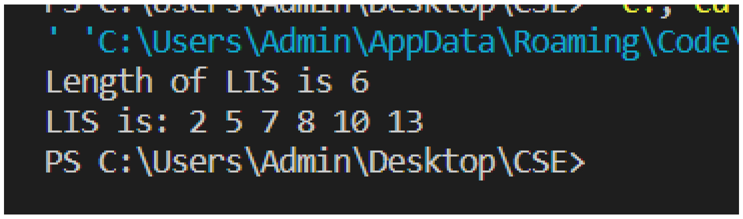
1. The lis method takes an array of integers nums as input and returns a list of indices representing the longest increasing subsequence in the array.
2. The code initializes an array dp of size n (where n is the length of nums) to store the length of the longest increasing subsequence ending at each index. It also initializes another array prevIndex to store the index of the previous element in the LIS sequence.
3. The dp array is initialized with all values set to 1, indicating that each element in the array itself forms a subsequence of length 1.
4. The code then iterates over the array nums from the second element (index 1) to the last element (index n-1).
5. For each element nums[i], it compares it with all the previous elements nums[j] (where j ranges from 0 to i-1).
6. If nums[i] is greater than nums[j] and the length of the LIS ending at nums[i] (dp[i]) is less than the length of the LIS ending at nums[j] plus 1 (dp[j] + 1), it means we can extend the LIS by including nums[i]. In this case, the code updates dp[i] to dp[j] + 1 and sets prevIndex[i] to j, indicating that nums[i] comes after nums[j] in the LIS sequence.
7. After the above loops complete, the code finds the maximum value in the dp array to determine the length of the longest increasing subsequence (maxLIS). It also keeps track of the index (maxLISIndex) where the maximum length is achieved.
8. The code then constructs the LIS sequence by starting from the maxLISIndex and following the prevIndex array until reaching an index of -1 (indicating the end of the sequence). It adds each index to the lisIndices list.
9. Finally, the lisIndices list is reversed using Collections.reverse() to obtain the correct order of indices. The method returns the lisIndices list.
10. In the main method, an example array nums is defined, and the lis method is called to obtain the LIS indices. The length of the LIS and the LIS elements are printed to the console.

Code:



```
package LIS_LCS;
import java.util.*;

public class LIS_2 {
    public static List<Integer> lis(int[] nums) {
        int n = nums.length;
        int[] dp = new int[n];
        Arrays.fill(dp, val:1);
        int[] prevIndex = new int[n];
        Arrays.fill(prevIndex, -1);

        for (int i = 1; i < n; i++) {
            for (int j = 0; j < i; j++) {
                if (nums[i] > nums[j] && dp[i] < dp[j] + 1) {
                    dp[i] = dp[j] + 1;
                    prevIndex[i] = j;
                }
            }
        }

        int maxLIS = 0;
        int maxLISIndex = 0;
        for (int i = 0; i < n; i++) {
            if (dp[i] > maxLIS) {
                maxLIS = dp[i];
                maxLISIndex = i;
            }
        }

        List<Integer> lisIndices = new ArrayList<>();
        int currentIndex = maxLISIndex;
        while (currentIndex != -1) {
            lisIndices.add(currentIndex);
            currentIndex = prevIndex[currentIndex];
        }
        Collections.reverse(lisIndices);

        return lisIndices;
    }

    Run | Debug
    public static void main(String[] args) {
        int[] nums = {9, 2, 5, 3, 7, 11, 8, 10, 13, 6};
        List<Integer> lisIndices = lis(nums);
        System.out.println("Length of LIS is " + lisIndices.size());
        System.out.print(s:"LIS is: ");
        for (int i : lisIndices) {
            System.out.print(nums[i] + " ");
        }
    }
}
```

Output:



```
' 'C:\Users\Admin\AppData\Roaming\Code
Length of LIS is 6
LIS is: 2 5 7 8 10 13
PS C:\Users\Admin\Desktop\CSE>
```

Task-2: Given a list of coins i.e 1 taka, 5 taka and 10 taka, can you determine the total number of combinations of the coins in the given list to make up the number N taka?

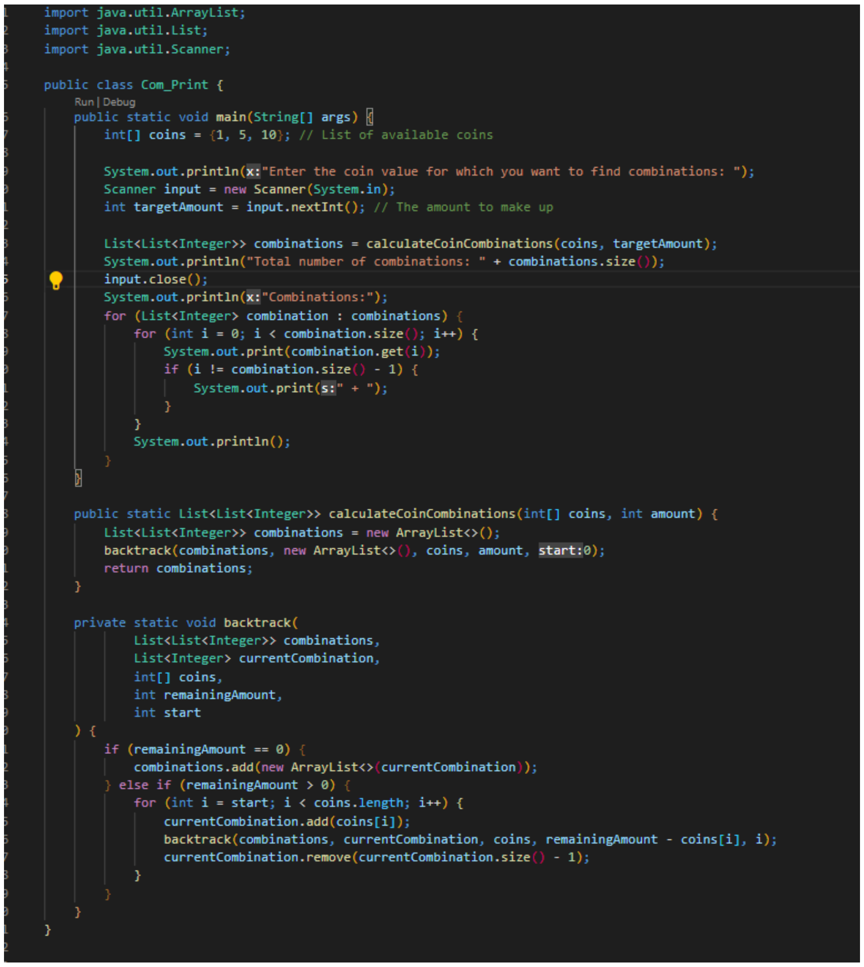
Explanation:

The main theory of the Problem is to calculate all possible combinations of the given coins that can add up to the target amount using a recursive backtracking algorithm. The code accomplishes this by following these steps:

1. The user is prompted to enter the target coin value for which combinations need to be found.
2. The **calculateCoinCombinations** method is called, passing the **coins** array and the target amount.
3. In the **calculateCoinCombinations** method, an empty list **combinations** is created to store the combinations.
4. The **backtrack** method is called with initial parameters: the **combinations** list, an empty **currentCombination** list, the **coins** array, the target amount, and the **start** index (which is initially 0).
5. The **backtrack** method is a recursive helper method responsible for generating combinations using backtracking.
6. Inside the **backtrack** method, there are two base cases:
    - If the **remainingAmount** is 0, it means a valid combination has been found. In this case, the current combination is added to the **combinations** list.
    - If the **remainingAmount** is greater than 0, the method proceeds with the backtracking process.
7. In the backtracking process, the method iterates over the **coins** array starting from the **start** index.
8. For each coin, it adds the current coin to the **currentCombination** list.
9. The **backtrack** method is recursively called with updated parameters:
    - The **remainingAmount** is decreased by the value of the current coin.
    - The **start** index remains the same, as multiple occurrences of the same coin are allowed in a combination.

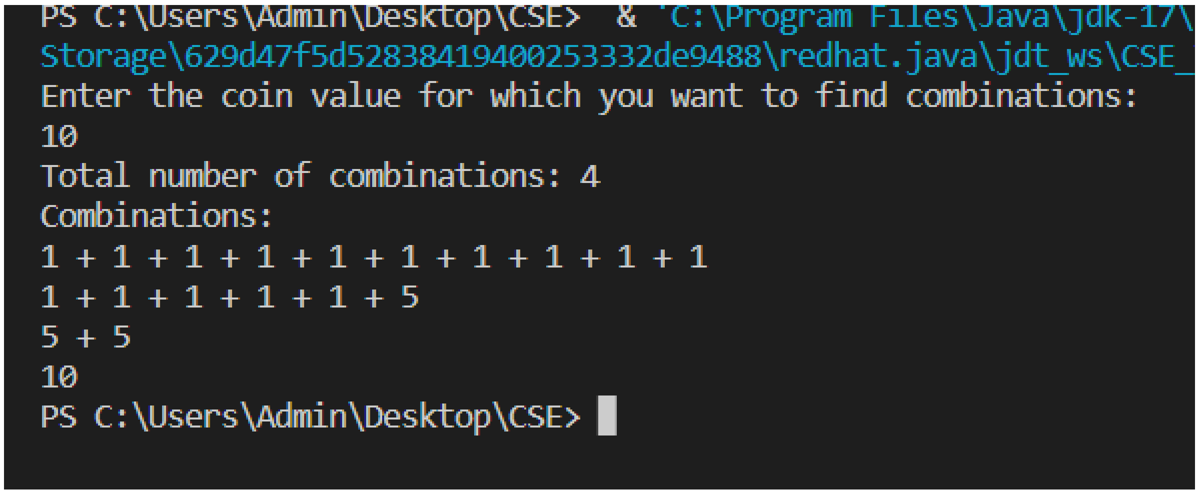
10. After the recursive call, the last coin is removed from the **currentCombination** list. This step allows exploring other possibilities by removing the last coin and considering the next coin in the array.
11. Once all combinations have been generated, the **combinations** list contains all possible combinations that add up to the target amount.
12. In the **main** method, the size of the **combinations** list is printed, which represents the total number of combinations found.
13. The code then iterates over each combination in the **combinations** list.
14. For each combination, it iterates over the coins and prints each coin value, separated by a "+" symbol.
15. The result is displayed on the console, showing the total number of combinations and the individual combinations themselves.

<u>Code:</u>

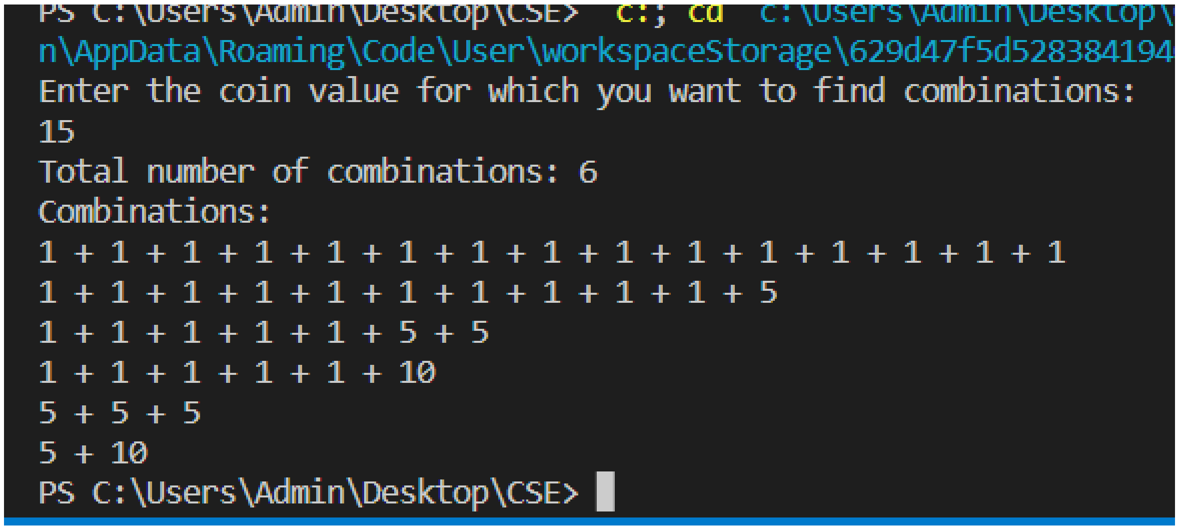


```java
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Com_Print {
    Run | Debug
    public static void main(String[] args) {
        int[] coins = {1, 5, 10}; // List of available coins

        System.out.println(x:"Enter the coin value for which you want to find combinations: ");
        Scanner input = new Scanner(System.in);
        int targetAmount = input.nextInt(); // The amount to make up

        List<List<Integer>> combinations = calculateCoinCombinations(coins, targetAmount);
        System.out.println("Total number of combinations: " + combinations.size());
        input.close();
        System.out.println(x:"Combinations:");
        for (List<Integer> combination : combinations) {
            for (int i = 0; i < combination.size(); i++) {
                System.out.print(combination.get(i));
                if (i != combination.size() - 1) {
                    System.out.print(s:" + ");
                }
            }
            System.out.println();
        }
    }

    public static List<List<Integer>> calculateCoinCombinations(int[] coins, int amount) {
        List<List<Integer>> combinations = new ArrayList<>();
        backtrack(combinations, new ArrayList<>(), coins, amount, start:0);
        return combinations;
    }

    private static void backtrack(
            List<List<Integer>> combinations,
            List<Integer> currentCombination,
            int[] coins,
            int remainingAmount,
            int start
    ) {
        if (remainingAmount == 0) {
            combinations.add(new ArrayList<>(currentCombination));
        } else if (remainingAmount > 0) {
            for (int i = start; i < coins.length; i++) {
                currentCombination.add(coins[i]);
                backtrack(combinations, currentCombination, coins, remainingAmount - coins[i], i);
                currentCombination.remove(currentCombination.size() - 1);
            }
        }
    }
}
```

Output:



```
PS C:\Users\Admin\Desktop\CSE>  & 'C:\Program Files\Java\jdk-17\
Storage\629d47f5d528384194002533332de9488\redhat.java\jdt_ws\CSE_
Enter the coin value for which you want to find combinations:
10
Total number of combinations: 4
Combinations:
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1
1 + 1 + 1 + 1 + 1 + 5
5 + 5
10
PS C:\Users\Admin\Desktop\CSE> 
```



```
PS C:\Users\Admin\Desktop\CSE>  c:; cd  c:\Users\Admin\Desktop\
n\AppData\Roaming\Code\User\workspaceStorage\629d47f5d528384194
Enter the coin value for which you want to find combinations:
15
Total number of combinations: 6
Combinations:
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 5
1 + 1 + 1 + 1 + 1 + 5 + 5
1 + 1 + 1 + 1 + 1 + 10
5 + 5 + 5
5 + 10
PS C:\Users\Admin\Desktop\CSE> 
```

Task-3: Print all the common subsequences according to the descending order of the lengths for two given sequences. (LCS)

Explanation:

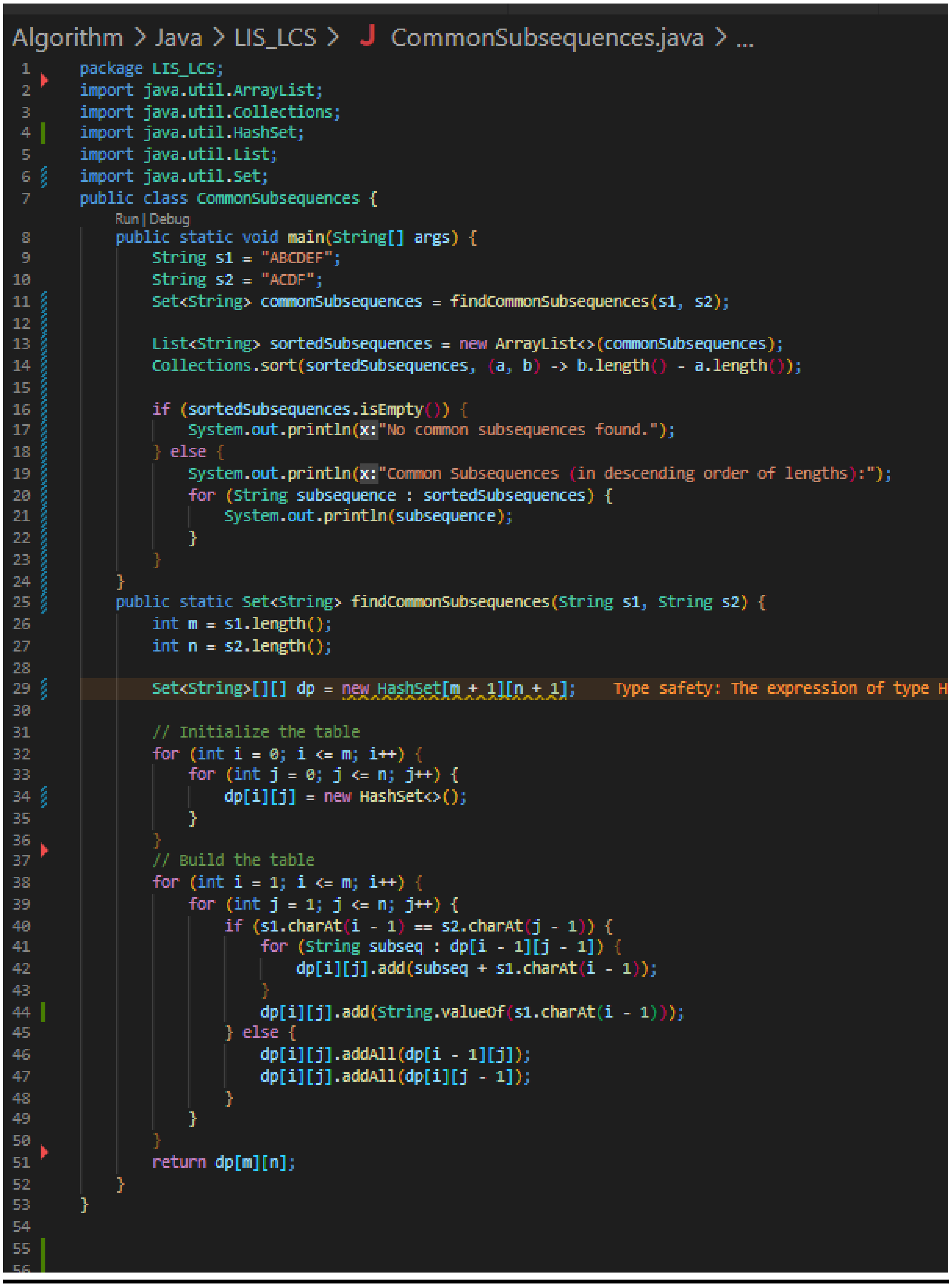
1. Initialize the input strings **s1** and **s2**.
2. Create a **Set<String>** called **commonSubsequences** to store the common subsequences between **s1** and **s2**.
3. Create a 2D array **dp** of type **HashSet** to store the dynamic programming table. The dimensions of the table are **m+1** by **n+1**, where **m** and **n** are the

lengths of **s1** and **s2**, respectively. Each cell of the table will contain a **HashSet** to store the subsequences.

4. Initialize the table by iterating over all cells and initializing each cell with an empty **HashSet**.
5. Build the dynamic programming table by iterating over each character of **s1** and **s2**. If the characters at the current positions match, iterate over the existing subsequences in **dp[i-1][j-1]** and add the current character to each subsequence. Also, add the current character as a separate subsequence. If the characters do not match, combine the subsequences from the previous rows and columns (**dp[i-1][j]** and **dp[i][j-1]**) and add them to the current cell.
6. After building the table, the cell at **dp[m][n]** will contain all the common subsequences between **s1** and **s2**.
7. Convert the **Set<String> commonSubsequences** into a **List<String>** called **sortedSubsequences** to allow sorting of the subsequences.
8. Sort the **sortedSubsequences** in descending order of their lengths using the **Collections.sort** method and a custom comparator that compares the lengths of the subsequences.
9. Check if the **sortedSubsequences** list is empty. If it is, print a message indicating that no common subsequences were found. Otherwise, print the common subsequences in descending order of their lengths by iterating over the **sortedSubsequences** list and printing each subsequence.

In summary, the code uses dynamic programming to build a table of subsequences and finds the common subsequences between two given strings. It then sorts the common subsequences by length and prints them in descending order.

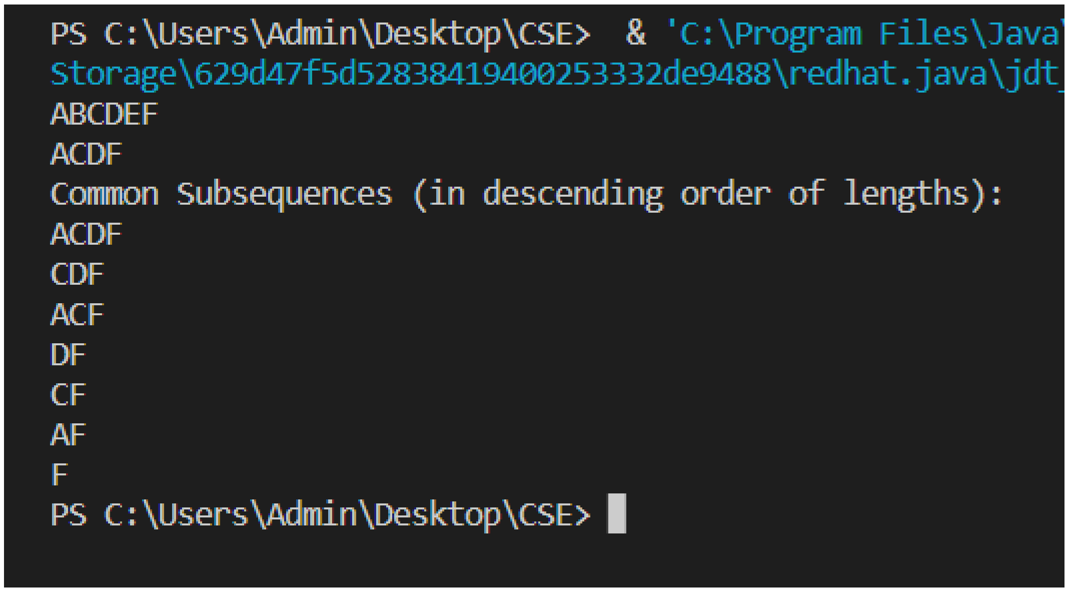
Code:



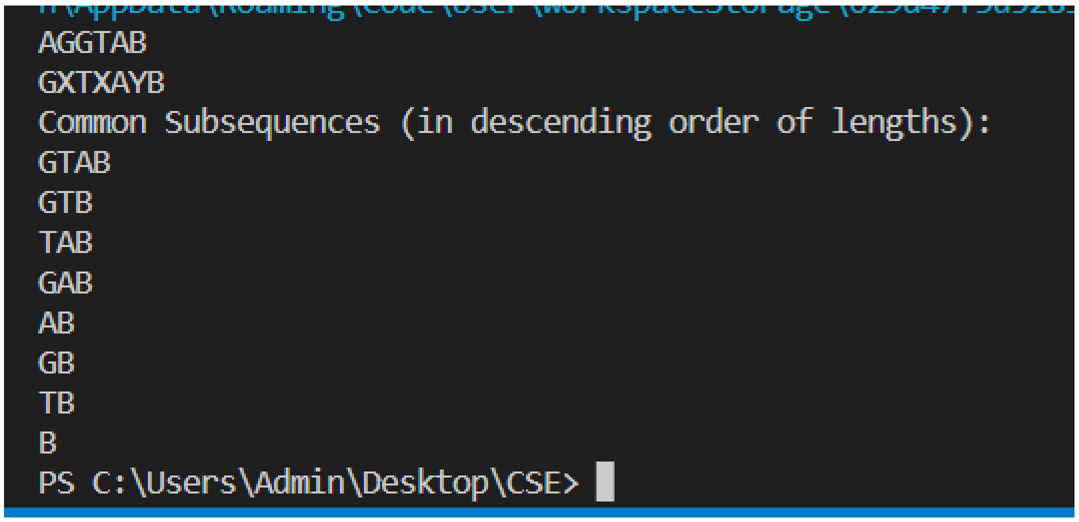
Algorithm > Java > LIS_LCS > CommonSubsequences.java > ...

```java
package LIS_LCS;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
public class CommonSubsequences {
    Run | Debug
    public static void main(String[] args) {
        String s1 = "ABCDEF";
        String s2 = "ACDF";
        Set<String> commonSubsequences = findCommonSubsequences(s1, s2);

        List<String> sortedSubsequences = new ArrayList<>(commonSubsequences);
        Collections.sort(sortedSubsequences, (a, b) -> b.length() - a.length());

        if (sortedSubsequences.isEmpty()) {
            System.out.println(x:"No common subsequences found.");
        } else {
            System.out.println(x:"Common Subsequences (in descending order of lengths):");
            for (String subsequence : sortedSubsequences) {
                System.out.println(subsequence);
            }
        }
    }
    public static Set<String> findCommonSubsequences(String s1, String s2) {
        int m = s1.length();
        int n = s2.length();

        Set<String>[][] dp = new HashSet[m + 1][n + 1];     Type safety: The expression of type H

        // Initialize the table
        for (int i = 0; i <= m; i++) {
            for (int j = 0; j <= n; j++) {
                dp[i][j] = new HashSet<>();
            }
        }
        // Build the table
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
                    for (String subseq : dp[i - 1][j - 1]) {
                        dp[i][j].add(subseq + s1.charAt(i - 1));
                    }
                    dp[i][j].add(String.valueOf(s1.charAt(i - 1)));
                } else {
                    dp[i][j].addAll(dp[i - 1][j]);
                    dp[i][j].addAll(dp[i][j - 1]);
                }
            }
        }
        return dp[m][n];
    }
}
```

Output:



```
PS C:\Users\Admin\Desktop\CSE>  & 'C:\Program Files\Java
Storage\629d47f5d52838419400253332de9488\redhat.java\jdt
ABCDEF
ACDF
Common Subsequences (in descending order of lengths):
ACDF
CDF
ACF
DF
CF
AF
F
PS C:\Users\Admin\Desktop\CSE>
```



```
n\AppData\Roaming\Code\User\workspaceStorage\629d47f5d528
AGGTAB
GXTXAYB
Common Subsequences (in descending order of lengths):
GTAB
GTB
TAB
GAB
AB
GB
TB
B
PS C:\Users\Admin\Desktop\CSE>
```