Day 15 and 16

Task 1: Knapsack Problem

Write a function int Knapsack(int W, int[] weights, int[] values) in C# that determines the maximum value of items that can fit into a knapsack with a capacity W. The function should handle up to 100 items. Find the optimal way to fill the knapsack with the given items to achieve the maximum total value. You must consider that you cannot break items, but have to include them whole.

Ans)

Code:-

```
package WiprpTask;
import java.util.ArrayList;
import java.util.List;
public class KnapsackProblem01 {
  public static void main(String[] args) {
     int capacity = 8;
     int[] values = {1, 2, 5, 6};
     int[] weights = {2, 3, 4, 5};
     int n = values.length;
     int maxValue = knapsack(capacity, weights, values, n);
     System.out.println("Maximum value that can be obtained: " + maxValue);
  }
  private static int knapsack(int capacity, int[] weights, int[] profits, int n) {
     int[][] t = new int[n + 1][capacity + 1];
     for (int rownum = 0; rownum <= n; rownum++) {
        for (int colnum = 0; colnum <= capacity; colnum++) {
           if (rownum == 0 || colnum == 0) {
              t[rownum][colnum] = 0;
           } else if (weights[rownum - 1] <= colnum) {
              t[rownum][colnum] = Math.max(t[rownum - 1][colnum],
                    profits[rownum - 1] + t[rownum - 1][colnum - weights[rownum - 1]]);
           } else {
              t[rownum][colnum] = t[rownum - 1][colnum];
           }
        }
     }
     List<Integer> itemsIncluded = findItemsIncluded(t, weights, profits, n, capacity);
     System.out.println("Items included in the knapsack: " + itemsIncluded);
     return t[n][capacity];
  }
  private static List<Integer> findItemsIncluded(int[][] t, int[] weights, int[] profits, int
n, int capacity) {
     List<Integer> itemsIncluded = new ArrayList<>();
     int i = n, j = capacity;
     while (i > 0 && j > 0) {
        if (t[i][j] != t[i - 1][j]) {
           itemsIncluded.add(i - 1);
```

```
            j -= weights[i - 1];
        }
        i--;
    }
    return itemsIncluded;
  }
}
```

OUTPUT:-

```
Items included in the knapsack: [3, 1]
Maximum value that can be obtained: 8
```

Task 2: Longest Common Subsequence
Implement int LCS(string text1, string text2) to find the length of the longest common subsequence between two strings.
Ans)
Code:-

```
package WiprpTask;
public class LongestCommonSubsequence {
  public static void main(String[] args) {
    String text1 = "abccba";
    String text2 = "aceeca";
    int length = LCS(text1, text2);
    System.out.println("Length of the longest common subsequence: " + length);
  }
  public static int LCS(String text1, String text2) {
    int m = text1.length();
    int n = text2.length();
    int[][] dp = new int[m + 1][n + 1];
    for (int i = 1; i <= m; i++) {
      for (int j = 1; j <= n; j++) {
        if (text1.charAt(i - 1) == text2.charAt(j - 1)) {
          dp[i][j] = dp[i - 1][j - 1] + 1;
        } else {
          dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
        }
      }
      printDPTable(dp, m, n);
    }
    int lcsLength = dp[m][n];
    char[] lcs = new char[lcsLength];
    int i = m, j = n, k = lcsLength - 1;
    while (k >= 0) {
      if (text1.charAt(i - 1) == text2.charAt(j - 1)) {
        lcs[k--] = text1.charAt(i - 1);
        i--;
        j--;
```

```
        } else if (dp[i - 1][j] > dp[i][j - 1]) {
            i--;
        } else {
            j--;
        }
    }
    System.out.println("Longest common subsequence: " + new String(lcs));
    return lcsLength;
  }
  private static void printDPTable(int[][] dp, int m, int n) {
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            System.out.print(dp[i][j] + " ");
        }
        System.out.println();
    }
    System.out.println();
  }
}
```

OUTPUT:-

```
0 0 0 0 0 0 0
0 1 1 1 1 1 1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 1 1 1 1 1 1
0 1 1 1 1 1 1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 1 1 1 1 1 1
0 1 1 1 1 1 1
0 1 2 2 2 2 2
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 1 1 1 1 1 1
0 1 1 1 1 1 1
0 1 2 2 2 2 2
0 1 2 2 2 3 3
```

```
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 1 1 1 1 1 1
0 1 1 1 1 1 1
0 1 2 2 2 2 2
0 1 2 2 2 3 3
0 1 2 2 2 3 3
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 1 1 1 1 1 1
0 1 1 1 1 1 1
0 1 2 2 2 2 2
0 1 2 2 2 3 3
0 1 2 2 2 3 3
0 1 2 2 2 3 4
Longest common subsequence: acca
Length of the longest common subsequence: 4
```