Day 15 and 16

1] 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.

Solution:-

Code -

KnapsackProblem01.java

```java
package com.wipro.dynamicprogram;

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;
```

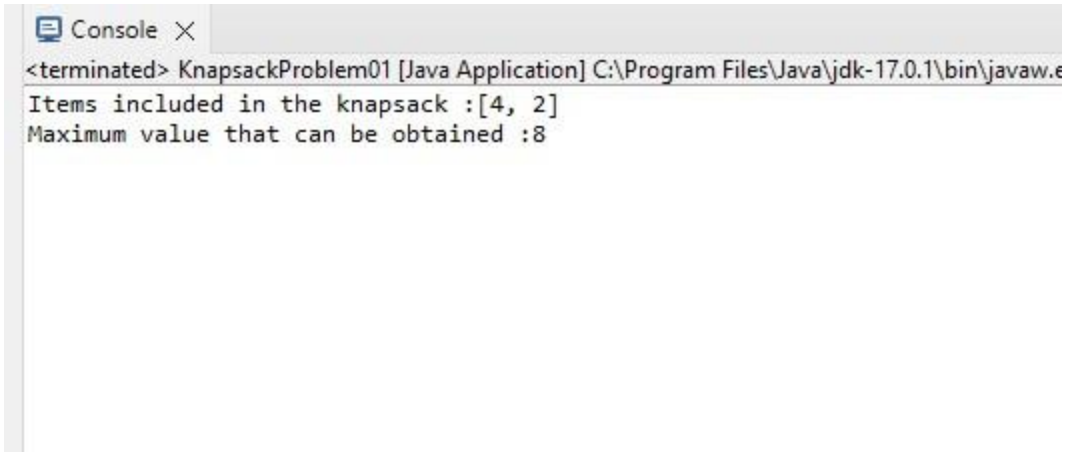
KnapsackProblem01.java

```
                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 row = n;
        int col = capacity;

        while (row > 0 && col > 0) {
            if (t[row][col] != t[row - 1][col]) {
                itemsIncluded.add(row);
```

KnapsackProblem01.java

```
        }

        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 row = n;
        int col = capacity;

        while (row > 0 && col > 0) {
            if (t[row][col] != t[row - 1][col]) {
                itemsIncluded.add(row);
                col -= weights[row - 1];
                row--;
            } else {
                row--;
            }
        }
        return itemsIncluded;
    }

}
```

Output -



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

Solution:-
Code -

LongestCommomSubsequence.java

```java
package com.wipro.dynamicprogram;

public class LongestCommomSubsequence {
    private static int[][] dp;

    public static void main(String[] args) {
        String str1 = "babbab";
        String str2 = "abaaba";

        int length = longestCommonSubsequence(str1, str2);
        System.out.println("Length of the common substr :" + length);


    }


    private static int longestCommonSubsequence(String str1, String str2) {
        int m = str1.length();
        int n = str2.length();
        dp = new int[m + 1][n + 1];

        for (int i = 0; i <= m; i++) {
            for (int j = 0; j <= n; j++) {
                if (i == 0 || j == 0) {

                    dp[i][j] = 0;
```

LongestCommomSubsequence.java

```
16
17      private static int longestCommonSubsequence(String str1, String str2) {
18          int m = str1.length();
19          int n = str2.length();
20          dp = new int[m + 1][n + 1];
21
22          for (int i = 0; i <= m; i++) {
23              for (int j = 0; j <= n; j++) {
24                  if (i == 0 || j == 0) {
25
26                      dp[i][j] = 0;
27                  } else if (str1.charAt(i - 1) == str2.charAt(j - 1)) {
28                      dp[i][j] = 1 + dp[i - 1][j - 1];
29                  } else {
30                      dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
31                  }
32              }
33          }
34          return dp[m][n];
35      }
36
37  }
38
39
```

Output -

Console

<terminated> LongestCommomSubsequence [Java Application] C:\Program Files\Java\jdk-17.

```
Length of the common substr :4
```