Shri Ramdeobaba College of Engineering & Management Nagpur-13

Department of Computer Application

Session: 2023-2024

Submission for

Course Name: Design Analysis and Algorithm Lab

Course Code: MCP546

Name of the Student: Jayesh Lalit Nandanwar

Class Roll No: 26

Semester: MCA II semester

Shift: 2

Batch: 2

Under the Guidance of

Prof. Manda Ukey

Date of submission: 13/04/2024

Practical 10

Aim: Perform a 0-1 knapsack using the DYNAMIC PROGRAMMING method to find the optimum load that the knapsack can carry.
Input the no. of objects, weights, profits, and the maximum capacity of the container.
Display the time taken to perform this and compare it with the greedy knapsack method.

Code:

```
import java.util.*;

public class ZeroOneKnapsack {

   static class Item {
      int weight;
      int profit;

      public Item(int weight, int profit) {
         this.weight = weight;
         this.profit = profit;
      }
   }

   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter the number of objects: ");
      int n = scanner.nextInt();

      int[] weights = new int[n];
      int[] profits = new int[n];

      System.out.println("Enter the weights of objects:");
      for (int i = 0; i < n; i++) {
         weights[i] = scanner.nextInt();
      }

      System.out.println("Enter the profits of objects:");
      for (int i = 0; i < n; i++) {
         profits[i] = scanner.nextInt();
      }

      System.out.print("Enter the maximum capacity of the knapsack: ");
      int capacity = scanner.nextInt();
```

```
        long startTime = System.nanoTime();
        KnapsackResult result = knapsack(weights, profits, capacity);
        long endTime = System.nanoTime();

        System.out.println("Maximum profit that can be achieved: " + result.maxProfit);
        System.out.println("Objects picked in order:");
        for (int index : result.pickedItems) {
            System.out.println("Object " + index + " (Weight: " + weights[index] + ",
Profit: " + profits[index] + ")");
        }
        System.out.println("Time taken: " + (endTime - startTime) + " nanoseconds");

        scanner.close();
    }

    static class KnapsackResult {
        int maxProfit;
        List<Integer> pickedItems;

        public KnapsackResult(int maxProfit, List<Integer> pickedItems) {
            this.maxProfit = maxProfit;
            this.pickedItems = pickedItems;
        }
    }

    public static KnapsackResult knapsack(int[] weights, int[] profits, int capacity) {
        int n = weights.length;
        int[][] dp = new int[n + 1][capacity + 1];

        for (int i = 0; i <= n; i++) {
            for (int w = 0; w <= capacity; w++) {
                if (i == 0 || w == 0) {
                    dp[i][w] = 0;
                } else if (weights[i - 1] <= w) {
                    dp[i][w] = Math.max(profits[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i -
1][w]);
                } else {
                    dp[i][w] = dp[i - 1][w];
                }
            }
        }

        int maxProfit = dp[n][capacity];
        List<Integer> pickedItems = new ArrayList<>();

        // Backtrack to find the items picked
```

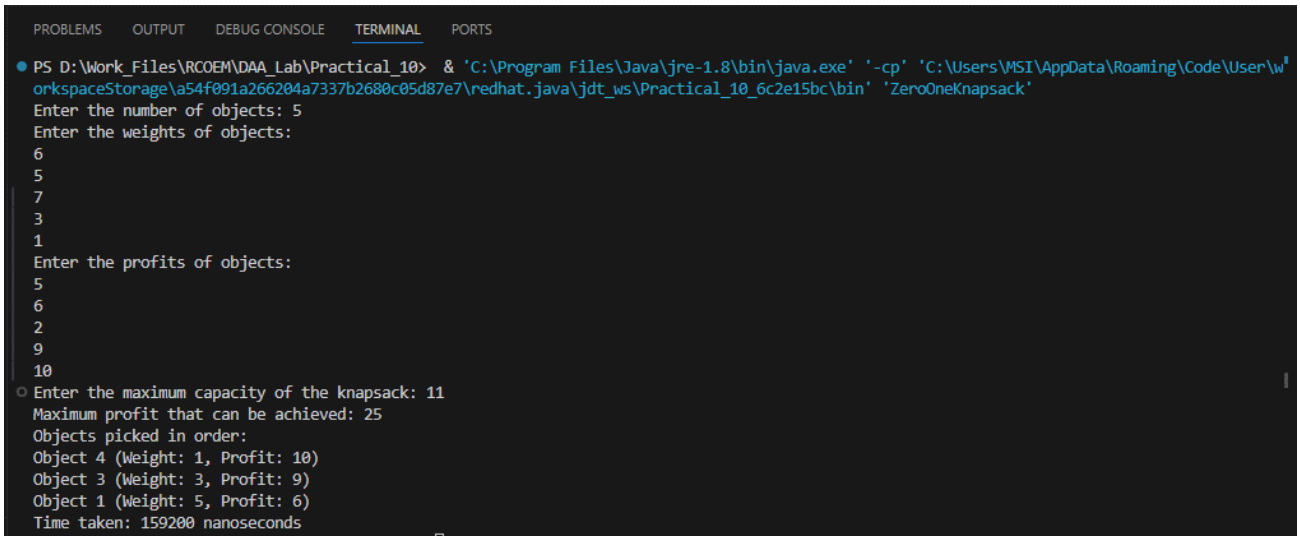
```
        int remainingCapacity = capacity;
        for (int i = n; i > 0 && maxProfit > 0; i--) {
            if (maxProfit != dp[i - 1][remainingCapacity]) {
                pickedItems.add(i - 1);
                maxProfit -= profits[i - 1];
                remainingCapacity -= weights[i - 1];
            }
        }

        return new KnapsackResult(dp[n][capacity], pickedItems);
    }
}
```

Data Used:

Object	Weight	Profit
1	6	5
2	5	6
3	7	2
4	3	9
5	1	10

Knapsack Capacity: 11

Output:



Time Taken: 159200 nanoseconds

Comparison between time taken (For same given problem):

Algorithm	Time Taken (in nano seconds)	Objects Picked (in order)	Maximum Profit
Greedy Knapsack	30049600	1,3,5	26.66668
0 – 1 Knapsack	159200	1,3,5	25