Experiment Number: 5

NAME: Harshwardhan Patil  ROLLNO: 50

CLASS: TY-IT BATCH: B1

DATE OF PERFORMANCE: 03-03-2025

**Problem Statement:**

**WAP for 0/1 Knapsack problem using dynamic programming.**

**Perform Time complexity analysis.**

A notebook with writing on it

AI-generated content may be incorrect.

A notebook with writing on it

AI-generated content may be incorrect.

A notebook with writing on it

AI-generated content may be incorrect.

A notebook with writing on it

AI-generated content may be incorrect.

A notebook with writing on it

AI-generated content may be incorrect.

import java.util.Arrays;

public class DaaAs5 {

    // Memoization Approach (Top-Down)

    static int knapsackMemo(int[] wt, int[] val, int W, int n, int[][] dp) {

        if (n == 0 || W == 0) return 0;

        if (dp[n][W] != -1) return dp[n][W];

        if (wt[n - 1] <= W) {

            return dp[n][W] = Math.max(

                val[n - 1] + knapsackMemo(wt, val, W - wt[n - 1], n - 1, dp),

                knapsackMemo(wt, val, W, n - 1, dp)

            );

        } else {

            return dp[n][W] = knapsackMemo(wt, val, W, n - 1, dp);

        }

    }

    // Tabulation Approach (Bottom-Up)

    static int knapsackTabulation(int[] wt, int[] val, int W, int n) {

        int[][] dp = new int[n + 1][W + 1];

        for (int i = 1; i <= n; i++) {

            for (int w = 0; w <= W; w++) {

                if (wt[i - 1] <= w) {

                    dp[i][w] = Math.max(val[i - 1] + dp[i - 1][w - wt[i - 1]], dp[i - 1][w]);

                } else {

                    dp[i][w] = dp[i - 1][w];

                }

            }

        }

        return dp[n][W];

    }

    // Space Optimized Approach

    static int knapsackOptimized(int[] wt, int[] val, int W, int n) {

        int[] prev = new int[W + 1];

        for (int i = 0; i < n; i++) {

            for (int w = W; w >= wt[i]; w--) {

                prev[w] = Math.max(prev[w], val[i] + prev[w - wt[i]]);

            }

        }

        return prev[W];

    }

    public static void main(String[] args) {

        int[] wt = {1, 3, 4, 5};

        int[] val = {10, 40, 50, 70};

        int W = 8;

        int n = wt.length;

        int[][] dp = new int[n + 1][W + 1];

        for (int[] row : dp) Arrays.fill(row, -1);

        long start, end;

        // Memoization Execution

        start = System.nanoTime();

        int memoResult = knapsackMemo(wt, val, W, n, dp);

        end = System.nanoTime();

        System.out.println("--------------------------------------------------");

        System.out.println("Memoization Approach (Top-Down)");

        System.out.println("Result: " + memoResult);

        System.out.println("Time Complexity: O(n \* W)");

        System.out.println("Execution Time: " + (end - start) + " ns");

        System.out.println("--------------------------------------------------");

        // Tabulation Execution

        start = System.nanoTime();

        int tabResult = knapsackTabulation(wt, val, W, n);

        end = System.nanoTime();

        System.out.println("Tabulation Approach (Bottom-Up)");

        System.out.println("Result: " + tabResult);

        System.out.println("Time Complexity: O(n \* W)");

        System.out.println("Execution Time: " + (end - start) + " ns");

        System.out.println("--------------------------------------------------");

        // Space Optimized Execution

        start = System.nanoTime();

        int optResult = knapsackOptimized(wt, val, W, n);

        end = System.nanoTime();

        System.out.println("Space Optimized Approach");

        System.out.println("Result: " + optResult);

        System.out.println("Time Complexity: O(n \* W)");

        System.out.println("Execution Time: " + (end - start) + " ns");

        System.out.println("--------------------------------------------------");

    }

}

