**Q2: Knapsack Problem (Dynamic Programming)**: Solve the **0/1 Knapsack Problem** using **dynamic programming**, where you are given a set of items, each with a weight and value, and must determine the most valuable combination that can fit within a weight limit.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Knapsack Problem</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 20px;

}

#output {

margin-top: 20px;

font-weight: bold;

}

</style>

</head>

<body>

<h1>Knapsack Problem</h1>

<div>

<label for="items">Enter items as weight,value pairs (e.g., "2,3 3,4 4,5"):</label><br>

<input type="text" id="items" placeholder="Enter items">

<label for="capacity">Enter knapsack capacity:</label><br>

<input type="number" id="capacity" placeholder="Capacity">

<button onclick="solveKnapsack()">Solve</button>

</div>

<div id="output"></div>

<script>

function parseItems(input) {

return input.split(' ').map(pair => {

const [weight, value] = pair.split(',').map(Number);

return { weight, value };

});

}

function solveKnapsack() {

const input = document.getElementById("items").value.trim();

const capacity = parseInt(document.getElementById("capacity").value, 10);

if (!input || isNaN(capacity) || capacity <= 0) {

alert("Please enter valid items and capacity.");

return;

}

const items = parseItems(input);

const n = items.length;

const dp = Array.from({ length: n + 1 }, () => Array(capacity + 1).fill(0));

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

const { weight, value } = items[i - 1];

for (let w = 1; w <= capacity; w++) {

if (weight <= w) {

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

} else {

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

}

}

}

const result = dp[n][capacity];

document.getElementById("output").innerHTML = `Maximum Value: ${result}`;

}

</script>

</body>

</html>