Part 2 : CS 3364

**Part 1: Definition of Subproblems**

**Deliverable 2.1:**  
The goal is to maximize the total enjoyment points for a tourist, given a set of experiences and a luggage weight limit. The subproblem can be defined as follows:

**Subproblem Definition:**  
Given an array of experiences Exp=[(w1​,v1​),(w2​,v2​),…,(wn​,vn​)]

Where wi is the weight and vi is the value (enjoyment points) of the ith experience, and a total weight capacity W, the Subproblem K (*i*, w) represents the maximum total enjoyment points we can achieve by selecting from the first I experiences without exceeding the weight W.

**Part 2: Recurrence Relation**

**Deliverable 2.2:**  
The recurrence relation for the 0/1 Knapsack problem is:

A math equations on a white background

Description automatically generated

**Explanation:**

* **Base Case:** If no experiences are left (i=0) or the weight capacity is zero (W=0), the maximum value is zero.
* **Exclusion Case:** If the weight of the current experience wi exceeds the current weight limit W, the experience cannot be included, so the value is the same as the subproblem without this item.
* **Inclusion Case:** If the experience can fit in the luggage (wi ≤ W), we have two options:
  1. **Exclude the experience:** Use the result of the previous i−1 experiences.
  2. **Include the experience:** Add the enjoyment points of the current experience vi and solve the subproblem with reduced capacity W−wi

**Part 3: Implementation**

* **Deliverable 2.3:**

def maximize\_enjoyment(weights, points, capacity):

num\_experiences = len(weights)

dp\_table = [[0 for \_ in range(capacity + 1)] for \_ in range(num\_experiences + 1)]

# Build the dynamic programming table

for i in range(1, num\_experiences + 1):

for current\_weight in range(1, capacity + 1):

if weights[i - 1] <= current\_weight:

# Max of excluding or including the current experience

dp\_table[i][current\_weight] = max(

dp\_table[i - 1][current\_weight],

points[i - 1] + dp\_table[i - 1][current\_weight - weights[i - 1]]

)

else:

dp\_table[i][current\_weight] = dp\_table[i - 1][current\_weight]

# Identify the selected experiences

selected\_experiences = []

remaining\_weight = capacity

for i in range(num\_experiences, 0, -1):

if dp\_table[i][remaining\_weight] != dp\_table[i - 1][remaining\_weight]: # Experience was included

selected\_experiences.append(i)

remaining\_weight -= weights[i - 1]

return dp\_table[num\_experiences][capacity], selected\_experiences

# Experience data

experience\_weights = [8, 7, 6, 5, 4]

enjoyment\_points = [1500, 1600, 1700, 1800, 3000]

luggage\_limit = 20

max\_points, chosen\_experiences = maximize\_enjoyment(experience\_weights, enjoyment\_points, luggage\_limit)

print(f"Maximum Enjoyment Points: {max\_points}")

print(f"Chosen Experiences: {chosen\_experiences}")

**Part 4: Analysis**

**Deliverable 2.4:**  
**Chosen Experiences:**  
The code identifies which experiences to include. Given the weight constraint W=20 and the experiences, the selected ones are:

* **Experience 5 (Weight: 4, Enjoyment: 3000)**
* **Experience 3 (Weight: 6, Enjoyment: 1700)**
* **Experience 4 (Weight: 5, Enjoyment: 1800)**

**Rationale:**

* These three experiences maximize the enjoyment points while keeping the total weight within the limit: 4+6+5=15 (remaining weight: 5 units).
* Experience 5 provides the highest value-to-weight ratio, making it the top choice.
* Experiences 4 and 3 further contribute to maximizing the total enjoyment without exceeding the limit.