Project 3: Dynamic Programming

We have a knapsack of capacity weight *C* (a positive integer) and *n* types of objects. Each object of the ith type has weight w_i and profit p_i (all w_i and all p_i are positive integers, i = 0, 1, …, *n-1*). There are unlimited supplies of each type of objects. Find the largest total profit of any set of the objects that fits in the knapsack.

Let P(*C*) be the maximum profit that can be made by packing objects into the knapsack of capacity *C*.

(1) Give a recursive definition of the function P(*C*).
(2) Draw the subproblem graph for P(14) where *n* is 3 with the weights and profits given below.

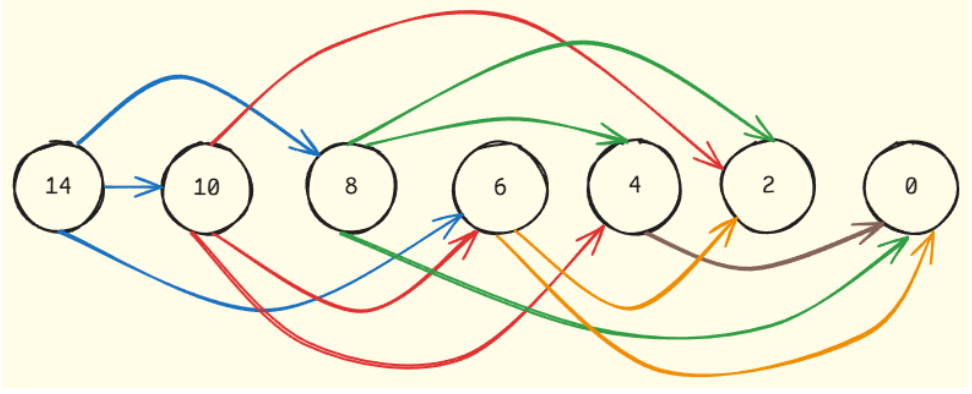
	0	1	2
w_i	4	6	8
p_i	7	6	9

Answers

(1) If capacity is 0, we can't include any item. We can take unlimited items of each type.
Compute all profits for all possible combinations and take the best case
(a) Base case: $P(0) = 0$
(b) Recursive case: $P(C) = max\{P(C - w_i) + p_i\}$, for $C > 0$, where $w_i <= C$
(i) For each item i, if its weight w_i fits into C, take the item and fill the rest optimally → $P(C - w_i)$
(ii) Add its profit, p_i

$$P(C) = max(P(C - w[0]) + p[0], P(C - w[1]) + p[1], P(C - w[2]) + p[2], .., P(C - w[n - 1]) + p[n - 1])$$

(2)



---

(3) Give a dynamic programming algorithm to compute the maximum profit, given a knapsack of capacity *C*, *n* types of objects with weights w_i and profits p_i using the bottom up approach.

Answer

(3)

1. Create a list “dp” of size C+1 to store the maximum profit for every capacity from every capacity from 0 to C.
2. Set dp[0]=0 because if the knapsack capacity is 0, the maximum profit is 0.
3. For each capacity “c” from 1 to X:
    a. Set “current_best_profit” to 0 (this will track the best profit for the current capacity).
    b. For each item “i” (with weight w[i] and profit p[i]):
        i. If the item fits (w[i]<=c:):
            - Calculate the remaining capacity if this item is taken: remaining_capacity = c- w[i]
            - Update “current_best_profit” to the maximum of:
            - Not taking the item: current_best_profit
            -Taking the item: p[i] +dp[remaining_capacity] which is the profit of the item + best profit of the remaining capacity
    c. Set dp[c]= current_best_profit (this stores the best profit for the current capacity “c”).
4. The final maximum profit will be in dp[C] (this is the maximum profit for the given knapsack of capacity C).
5. Return dp[c].

---

(4) Code your algorithm in a programming language

a. show the running result of P(14) with weights and profits given in (2).
b. Show the running result of P(14) with weights and profits given below.

	0	1	2
w_i	5	6	8
p_i	7	6	9

Answers

Code:

```python
def knapsack_bottom_up(weights, profits, capacity):
    # initialize a dp table with size capacity + 1, all set to 0
    dp = [0] * (capacity + 1)

    # iterate through each capacity from 1 to the given capacity
    for c in range(1, capacity + 1):
        current_best_profit = 0  # initialize the best profit for this capacity
        # check each item to see if it can fit into the current capacity
        for i in range(len(weights)):
            item_weight = weights[i]
            item_profit = profits[i]
            if item_weight <= c:
                remaining_capacity = c - item_weight
                # update the best profit for this capacity
                current_best_profit = max(current_best_profit, item_profit + dp[remaining_capacity])
        dp[c] = current_best_profit  # store the best profit for this capacity

    # return the maximum profit for the given knapsack capacity
    return dp[capacity]
```

(a)
Code for testing:

```python
#testing inputs for part a

weights=[4,6,8] #weight of each item
profits=[7,6,9] #profit of each item
capacity=14 #capacity of the knapsack

max_profit = knapsack_bottom_up(weights, profits, capacity)

# print output in table format
print("\n" + "="*50)
print("KNAPSACK PROBLEM - BOTTOM-UP APPROACH")
print("="*50)
print("\nITEMS:")
print("-"*50)
print(f"{'Item':<10} {'Weight':<15} {'Profit':<15}")
print("-"*50)
for i in range(len(weights)):
    print(f"{'Item ' + str(i+1):<10} {weights[i]:<15} {profits[i]:<15}")
print("-"*50)
print(f"\n{'Knapsack Capacity:':<30} {capacity}")
print(f"{'Maximum Profit:':<30} {max_profit}")
print("="*50 + "\n")
```

Output:

```
==================================================
KNAPSACK PROBLEM - BOTTOM-UP APPROACH
==================================================

ITEMS:
--------------------------------------------------
Item       Weight          Profit
--------------------------------------------------
Item 1     4               7
Item 2     6               6
Item 3     8               9
--------------------------------------------------

Knapsack Capacity:              14
Maximum Profit:                 21
==================================================
```

(b)

Code for testing:

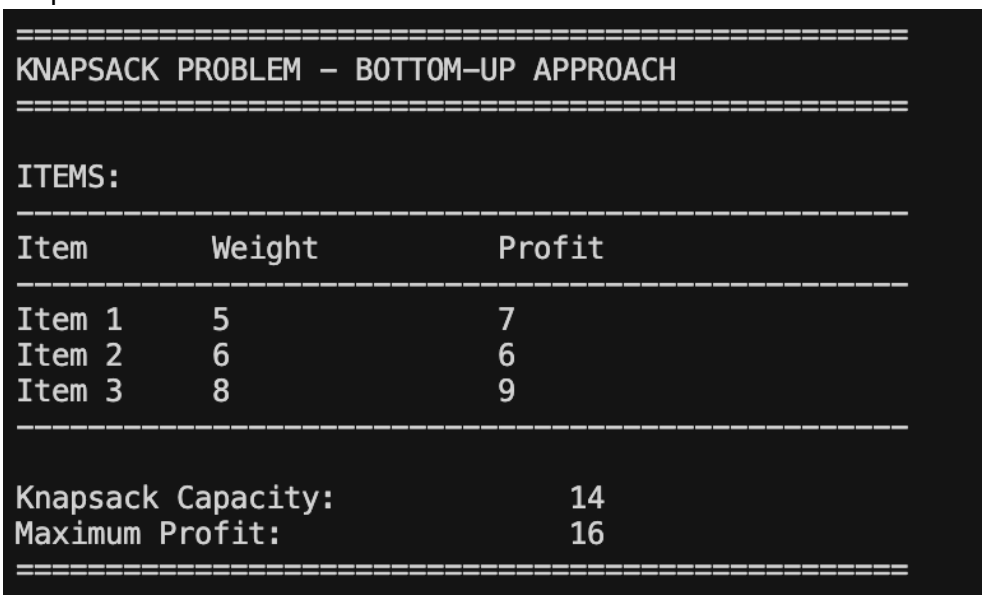
```
#testing inputs for part b

weights=[5,6,8] #weight of each item
profits=[7,6,9] #profit of each item
capacity=14 #capacity of the knapsack

max_profit = knapsack_bottom_up(weights, profits, capacity)

# print output in table format
print("\n" + "="*50)
print("KNAPSACK PROBLEM - BOTTOM-UP APPROACH")
print("="*50)
print("\nITEMS:")
print("-"*50)
print(f"{'Item':<10} {'Weight':<15} {'Profit':<15}")
print("-"*50)
for i in range(len(weights)):
    print(f"{'Item ' + str(i+1):<10} {weights[i]:<15} {profits[i]:<15}")
print("-"*50)
print(f"\n{'Knapsack Capacity:':<30} {capacity}")
print(f"{'Maximum Profit:':<30} {max_profit}")
print("="*50 + "\n")
```

Output:


KNAPSACK PROBLEM - BOTTOM-UP APPROACH
ITEMS:
Item Weight Profit
Item 1 5 7
Item 2 6 6
Item 3 8 9
Knapsack Capacity: 14
Maximum Profit: 16

—----------