Project 3

Dynamic Programming

Presented by:

BRANDON JANG JIN TIAN

KEITH LIM EN KAI

GOH ANNA, NINETTE

TEE QIN TONG BETTINA

Table Of Contents

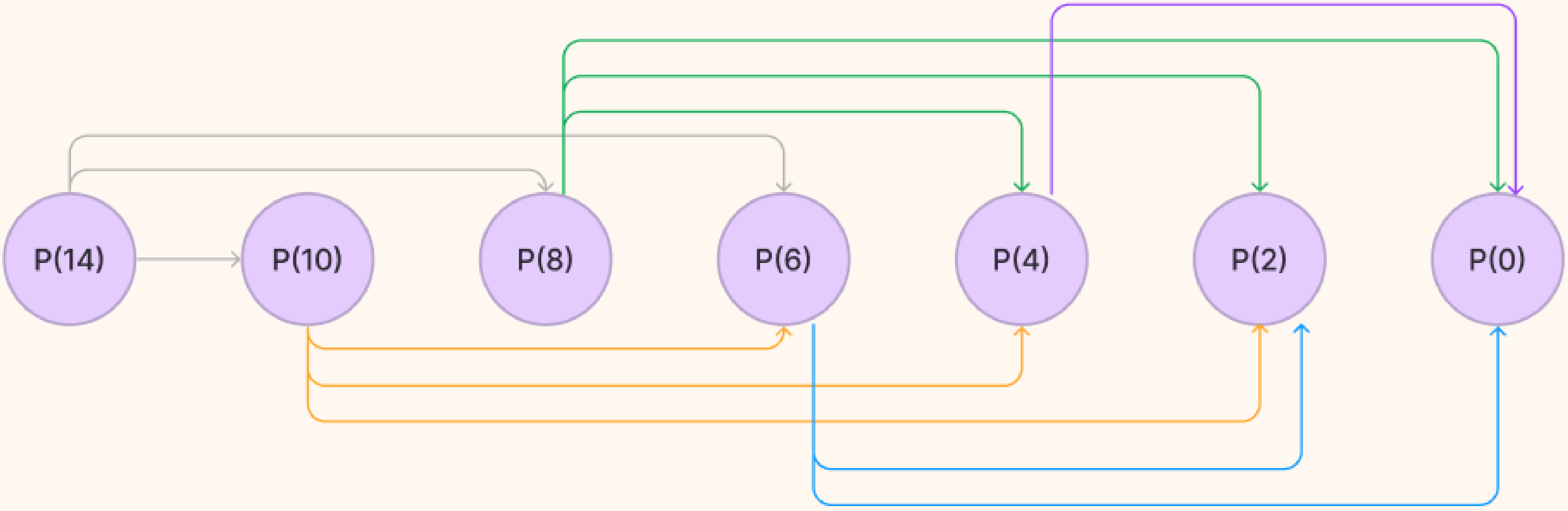
1 Recursive Definition

$P(C) = 0$, if $C = 0$
$P(C) = \max(P(C), P(C - w_i) + p_i)$ for all i, where $0 <= i < n$ and $w_i <= C$

The maximum profit for a knapsack of capacity C is either:

1. 0 if there's no capacity, or
2. The maximum of two options:
   a. either we include the i^{th} object, which contributes p_i to the profit, and reduce the capacity by w_i
   b. or we exclude the i^{th} object.

2 Subproblem Graph



3 Dynamic Programming Algorithm

Goal: Using Dynamic Programming, craft a bottom up approach algorithm to calculate P(C).

3 Dynamic Programming Algorithm

Pseudocode:

Algorithm 1: Dynamic Programming Algorithm for 0-1 Knapsack Problem

Data: $W, v_1, v_2, v_3, ..., v_n\ , w_1, w_2, w_3, ..., w_n$

Result: $M[n, W]$

$M[0, w] \leftarrow 0,\ \forall w\ 0\ to\ W$

$M[i, 0] \leftarrow 0,\ \forall i\ 0\ to\ n$

```
for i ← 1 to n do
    for w ← 1 to W do
        if w_i ≤ w then
            M[i, w] = max(M[i − 1, w − w_i] + v_i, M[i − 1, w])
        else
            M[i, w] = M[i − 1, w]
        end
    end
end
return M[n, W]
```

3 Dynamic Programming Algorithm

Implementation in Python:

```python
def knapsack_max_profit(weights, profits, C):
    n = len(weights)
    dp = [0] * (C + 1)

    for i in range(C + 1):
        for j in range(n):
            if weights[j] <= i:
                dp[i] = max(dp[i], dp[i - weights[j]] + profits[j])

    return dp[C]
```

3 Dynamic Programming Algorithm

Exploring Further...

Time Complexity: 2 for-loops

= O(C*N + 1)

= O(C*N)

Space Complexity: **O(C)**

4 Running Results

	0	1	2
w_i	4	6	8
p_i	7	6	9

```python
weights1 = [4, 6, 8]
profits1 = [7, 6, 9]
capacity1 = 14

result1 = knapsack_max_profit(weights1, profits1, capacity1)
print("Maximum profit for P(14) with weights [4, 6, 8] and profits [7, 6, 9]:", result1)
```

```
Maximum profit for P(14) with weights [4, 6, 8] and profits [7, 6, 9]: 21
```

4 Running Results

	0	1	2
w_i	5	6	8
p_i	7	6	9

```python
weights2 = [5, 6, 8]
profits2 = [7, 6, 9]
capacity2 = 14

result2 = knapsack_max_profit(weights2, profits2, capacity2)
print("Maximum profit for P(14) with weights [5, 6, 8] and profits [7, 6, 9]:", result2)
```

```
Maximum profit for P(14) with weights [5, 6, 8] and profits [7, 6, 9]: 16
```

4 Running Results – Knapsack Table

```python
def knapsack_max_profit_steps_table(weights, profits, C):
    n = len(weights)
    dp = [[0] * (C + 1) for _ in range(n + 1)]

    for i in range(n + 1):
        for w in range(C + 1):
            if i == 0 or w == 0:
                dp[i][w] = 0
            elif weights[i - 1] <= w:
                dp[i][w] = max(dp[i - 1][w], dp[i][w - weights[i - 1]] + profits[i - 1])
            else:
                dp[i][w] = dp[i - 1][w]

    dp_table = pd.DataFrame(dp, columns=range(C + 1), index=range(n + 1))
    dp_table = dp_table.style.set_table_styles([{
        'selector': 'td',
        'props': [
            ('padding', '10px')
        ]
    }])

    return dp_table
```

4 Running Results – Knapsack Table

	0	1	2
w_i	4	6	8
p_i	7	6	9

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	7	7	7	7	14	14	14	14	21	21	21
2	0	0	0	0	7	7	7	7	14	14	14	14	21	21	21
3	0	0	0	0	7	7	7	7	14	14	14	14	21	21	21

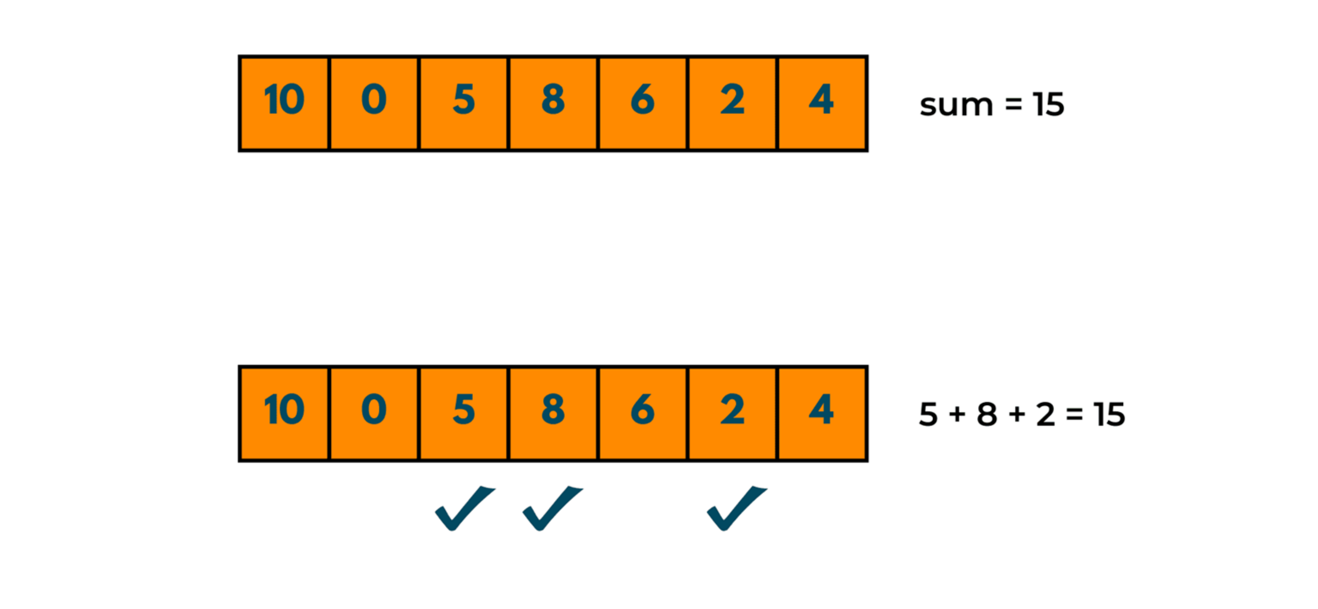
4 Running Results – Knapsack Table

	0	1	2
w_i	5	6	8
p_i	7	6	9

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	7	7	7	7	7	14	14	14	14	14
2	0	0	0	0	0	7	7	7	7	7	14	14	14	14	14
3	0	0	0	0	0	7	7	7	9	9	14	14	14	16	16

5 Food for Thought (Extra)

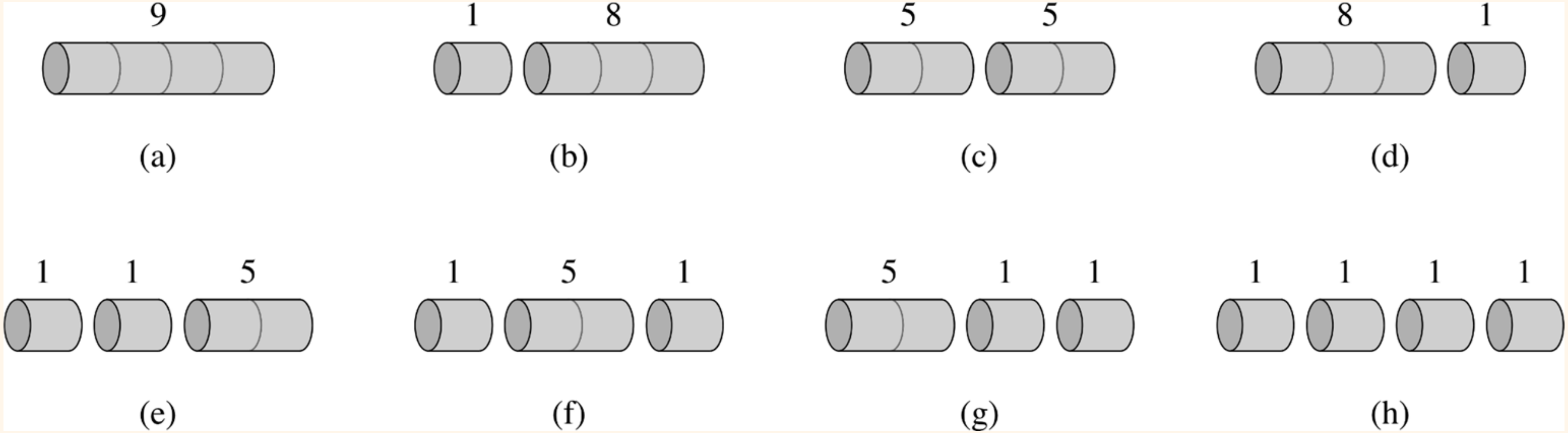
Subset Sum Problem (Dynamic Programming)



Given a set of positive integers and a target sum, determine if there is a subset of the integers that adds up to the target sum

5 Food for Thought (Extra)

Rod Cutting Problem (Dynamic Programming)



Given a rod of a certain length and a list of prices for different lengths at which the rod can be cut. The goal is to determine the optimal way to cut the rod to maximize the total revenue.

5 Food for Thought (Extra)

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.

What if supplies are limited?

5 Food for Thought (Extra)

Dynamic Programming Problems Allowing Reusing (Repetitions)

- Items/types can be used multiple times in the solution.
- More relaxed constraints in the dynamic programming equations
- Typically easier to solve due to flexibility of reusing the same type
- Example: Unbounded Knapsack

Dynamic Programming Problems Not Allowing Reusing (No Repetitions)

- Each item/type can be used only once in the solution
- Additional complexity due to tracking used and available items
- Example: 0/1 Knapsack