Group 1

EXAMPLE CLASS 3

Afreen, Arun, Athena

6 Nov, 2024

CONTENT

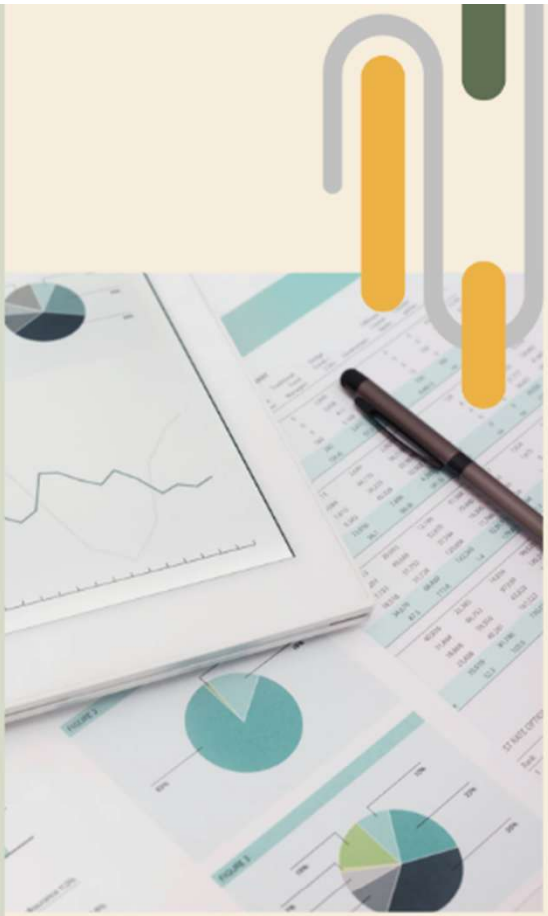
Example Class 3

ABSTRACT

A knapsack with a **capacity C** and **n types** of objects, where each object type i has:

- Weight wi (positive integer)
- Profit pi (positive integer)
- Unbounded Knapsack problem.

The objective is to **maximize the profit** that can be achieved by **packing the knapsack with any combination of objects**, provided the **total weight does not exceed C**. We define **P(C) as the maximum profit achievable** with the given capacity C.



RECURSIVE DEFINITION: NAIVE APPROACH

$$P(C)=\max(0,\max_{0\leq i<n}\{p_i+P(C-w_i)\})$$

Remaining Capacity of the Knapsack

profit from including object i

weight of object i

COMPLEXITY

TIME COMPLEXITY

$O(nC^2)$

SPACE COMPLEXITY

$O(C)$

RECURSIVE DEFINITION: OPTIMISED APPROACH

$$dp[c]=\max(dp[c],p_i+dp[c-w_i])$$

for each i and $c \geqslant wi$

COMPLEXITY

TIME COMPLEXITY

$O(nC)$

SPACE COMPLEXITY

$O(C)$

COMPLEXITY

Naive Approach

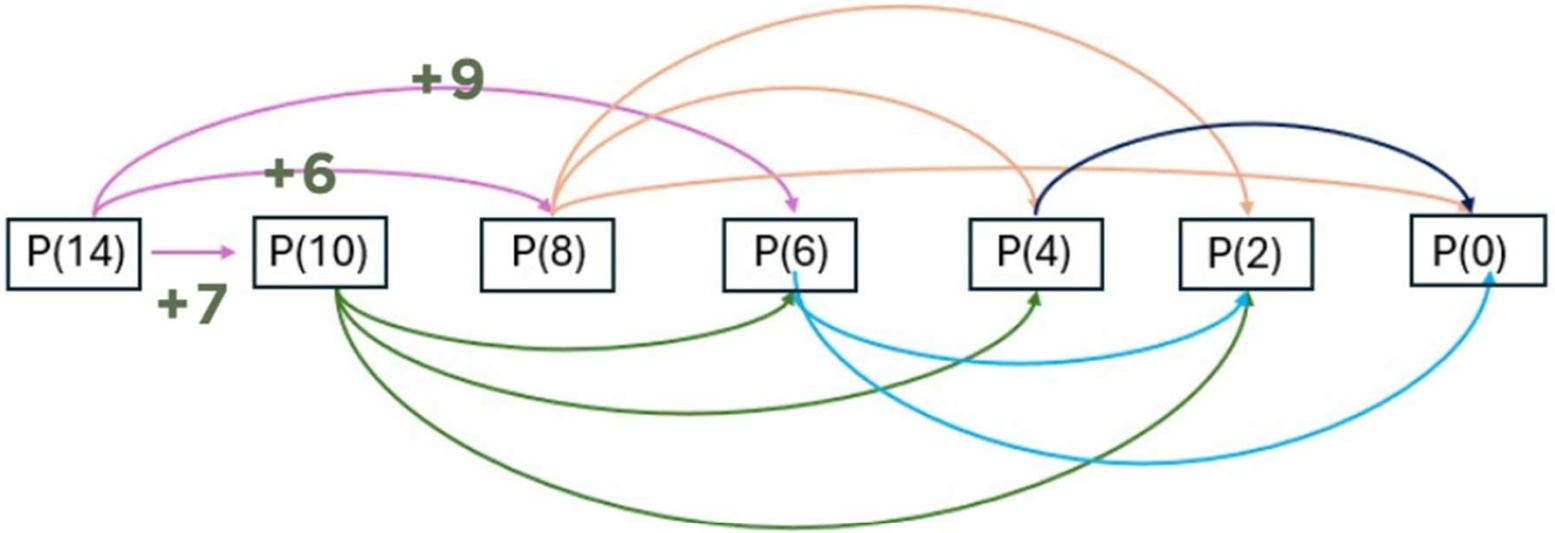
$O(nC^2)$

>

Optimised Approach

$O(nC)$

SUBPROBLEM GRAPH



P(14)
n = 3

	0	1	2
w_i	4	6	8
p_i	7	6	9

PSEUDOCODE OF KNAPSACK

```
function unboundedKnapsack(C, weights, profits, n):
    # Step 1: Initialize array P of size C + 1 with all zeros
    P = array of size (C + 1) initialized to 0

    # Step 2: Loop through each possible capacity from 1 to C
    for c from 1 to C:
        # Step 3: Loop through each object type
        for i from 0 to n - 1:
            if weights[i] <= c:
                # Step 4: Calculate maximum profit including this object
                P[c] = max(P[c], profits[i] + P[c - weights[i]])

    # Step 5: Return the maximum profit for capacity C
    return P[C]
```

PYTHON CODE OF KNAPSACK

```python
def unbounded_knapsack(C, weights, profits):
    # Step 1: Initialize the DP array P with zero values
    P = [0] * (C + 1)

    # Step 2: Process each capacity from 1 to C
    for c in range(1, C + 1):
        # Step 3: Check each item type
        for i in range(len(weights)):
            if weights[i] <= c:
                # Step 4: Calculate maximum profit by including this item
                P[c] = max(P[c], profits[i] + P[c - weights[i]])

    # Step 5: Return the maximum profit for the full capacity
    return P[C]
```

SUBPROBLEM GRAPH FOR TABLE(2)

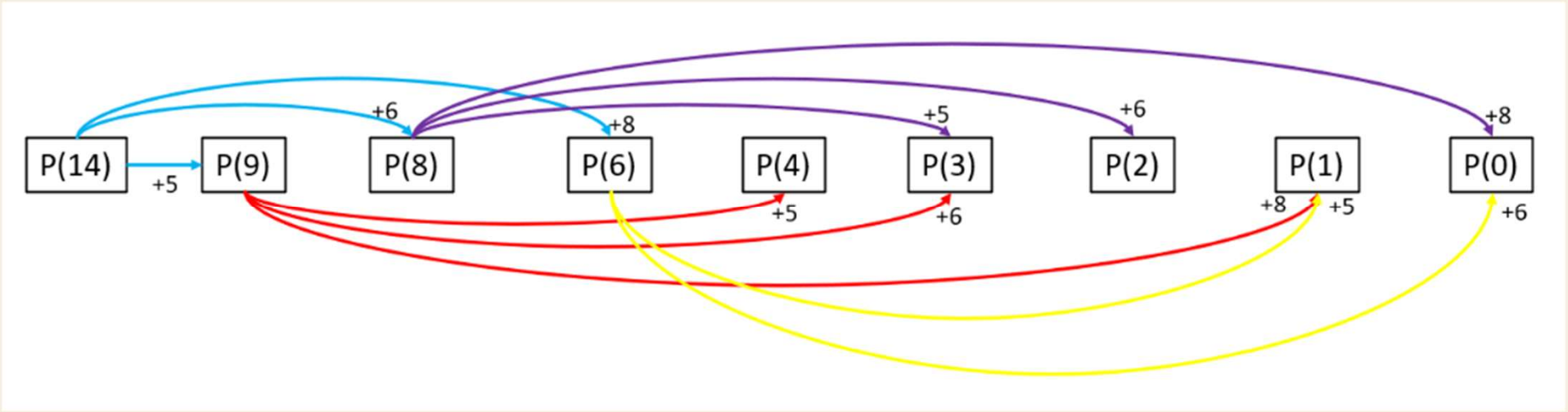
knapsack of capacity weight

c = 14

Number of objects

n = 3

	0	1	2
w_i	5	6	8
p_i	7	6	9



RUNNING RESULT OF P(14) OF TABLE(1)

```
# Table1
weights = [4, 6, 8]       # weights of items
profits = [7, 6, 9]       # profits of items
capacity = 14             # maximum capacity of the knapsack
n = 3                     # number of item types

# Call the function
max_profit = knapsack(weights, profits, capacity, n)
```

PROBLEMS 22 OUTPUT DEBUG CONSOLE TERMINAL PORTS Code

```
PS C:\Users\arunk\Desktop\NTU-SC2-HMS> python -u "c:\Users\arunk\Desktop\NTU-SC2-HMS\
tempCodeRunnerFile.python"
Profit arrays after each capacity increment:
After capacity 1, profit array: [0, 0]
After capacity 2, profit array: [0, 0, 0]
After capacity 3, profit array: [0, 0, 0, 0]
After capacity 4, profit array: [0, 0, 0, 0, 7]
After capacity 5, profit array: [0, 0, 0, 0, 7, 7]
After capacity 6, profit array: [0, 0, 0, 0, 7, 7, 7]
After capacity 7, profit array: [0, 0, 0, 0, 7, 7, 7, 7]
After capacity 8, profit array: [0, 0, 0, 0, 7, 7, 7, 7, 14]
After capacity 9, profit array: [0, 0, 0, 0, 7, 7, 7, 7, 14, 14]
After capacity 10, profit array: [0, 0, 0, 0, 7, 7, 7, 7, 14, 14, 14]
After capacity 11, profit array: [0, 0, 0, 0, 7, 7, 7, 7, 14, 14, 14, 14]
After capacity 12, profit array: [0, 0, 0, 0, 7, 7, 7, 7, 14, 14, 14, 14, 21]
After capacity 13, profit array: [0, 0, 0, 0, 7, 7, 7, 7, 14, 14, 14, 14, 21, 21]
After capacity 14, profit array: [0, 0, 0, 0, 7, 7, 7, 7, 14, 14, 14, 14, 21, 21, 21]
Maximum Profit: 21
PS C:\Users\arunk\Desktop\NTU-SC2-HMS>
```

1. Profits from capacity 0 to 3 = 0
2. Profits from capacity 4 to 7 = 7
3. Profits from capacity 8 to 11 = 14
4. Profits from capacity 12 to 14 =21

Max profit = 21

RUNNING RESULT OF P(14) OF TABLE(2)

```
# Table2
weights = [5, 6, 8]      # weights of items
profits = [7, 6, 9]      # profits of items
capacity = 14            # maximum capacity of the knapsack
n = 3                    # number of item types

# Call the function
max_profit = knapsack(weights, profits, capacity, n)
```

```
PS C:\Users\arunk\Desktop\NTU-SC2-HMS> python -u "c:\Users\arunk\Desktop\NTU-SC2-HMS
tempCodeRunnerFile.python"
Profit arrays after each capacity increment:
After capacity 1, profit array: [0, 0]
After capacity 2, profit array: [0, 0, 0]
After capacity 3, profit array: [0, 0, 0, 0]
After capacity 4, profit array: [0, 0, 0, 0, 0]
After capacity 5, profit array: [0, 0, 0, 0, 0, 7]
After capacity 6, profit array: [0, 0, 0, 0, 0, 7, 7]
After capacity 7, profit array: [0, 0, 0, 0, 0, 7, 7, 7]
After capacity 8, profit array: [0, 0, 0, 0, 0, 7, 7, 7, 9]
After capacity 9, profit array: [0, 0, 0, 0, 0, 7, 7, 7, 9, 9]
After capacity 10, profit array: [0, 0, 0, 0, 0, 7, 7, 7, 9, 9, 14]
After capacity 11, profit array: [0, 0, 0, 0, 0, 7, 7, 7, 9, 9, 14, 14]
After capacity 12, profit array: [0, 0, 0, 0, 0, 7, 7, 7, 9, 9, 14, 14, 14]
After capacity 13, profit array: [0, 0, 0, 0, 0, 7, 7, 7, 9, 9, 14, 14, 14, 16]
After capacity 14, profit array: [0, 0, 0, 0, 0, 7, 7, 7, 9, 9, 14, 14, 14, 16, 16]
Maximum Profit: 16
PS C:\Users\arunk\Desktop\NTU-SC2-HMS>
```

1. Profits from capacity 0 to 4 = 0
2. Profits from capacity 5 to 7 = 7
3. Profits from capacity 8 to 9 = 9
4. Profits from capacity 10 to 12 = 14
5. Profits from capacity 13 to 14 = 16

Max profit = 16

TABLE(1) VS TABLE(2)

```
# Table1
weights = [4, 6, 8]       # weights of items
profits = [7, 6, 9]       # profits of items
capacity = 14             # maximum capacity of the knapsack
n = 3                     # number of item types

# Call the function
max_profit = knapsack(weights, profits, capacity, n)
```

After capacity 14, profit array:
[0, 0, 0, 0, 7, 7, 7, 7, 14, 14, 14, 14, **21, 21, 21**]

1. Profits from capacity 0 to 3 = 0
2. Profits from capacity 4 to 7 = 7
3. Profits from capacity 8 to 11 = 7+7 = 14
4. Profits from capacity 12 to 14 =7+7+7 =21

```
# Table2
weights = [5, 6, 8]       # weights of items
profits = [7, 6, 9]       # profits of items
capacity = 14             # maximum capacity of the knapsack
n = 3                     # number of item types

# Call the function
max_profit = knapsack(weights, profits, capacity, n)
```

After capacity 14, profit array:
[0, 0, 0, 0, 0, 7, 7, 7, 9, 9, 14, 14, 14, **16, 16**]

1. Profits from capacity 0 to 4 = 0
2. Profits from capacity 5 to 7 = 7
3. Profits from capacity 8 to 9 = 9
4. Profits from capacity 10 to 12 = 7+7 = 14
5. Profits from capacity 13 to 14 = 7 + 9 = 16

Group 1

THANK YOU