Project 3
Dynamic Programming

presented by

Aditi
Mabel
Ming Kai

timeline

- Recursion Definition
- Graph Subproblem for P(14)
- Pseudocode for 0/1 Knapsack
- Python Implement for 0/1 Knapsack

1. Recursion Function

P(C) = Maximum Profit
C = Capacity of the Knapsack
n = Number of items
W[n] = Weight of the n'th item
P[n] = Profit of the n'th item

Base Case: P(0) = 0

If C > 0 and n > 0

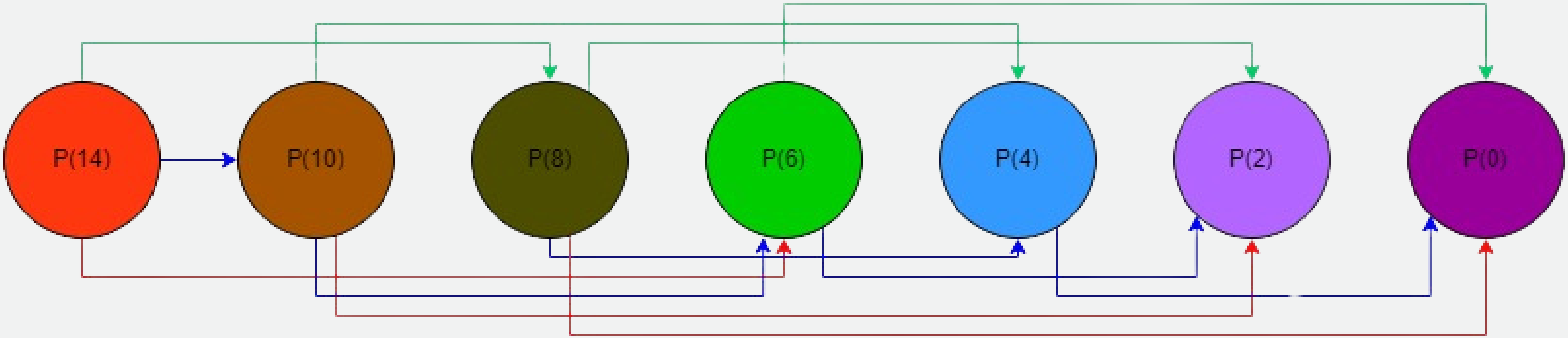
$$P(C) = \max\,[P(C)\ ,\ P(C\text{-}wi)+pi]$$

2. Subproblem Graphs

Subproblem graph for P(14), where n=3

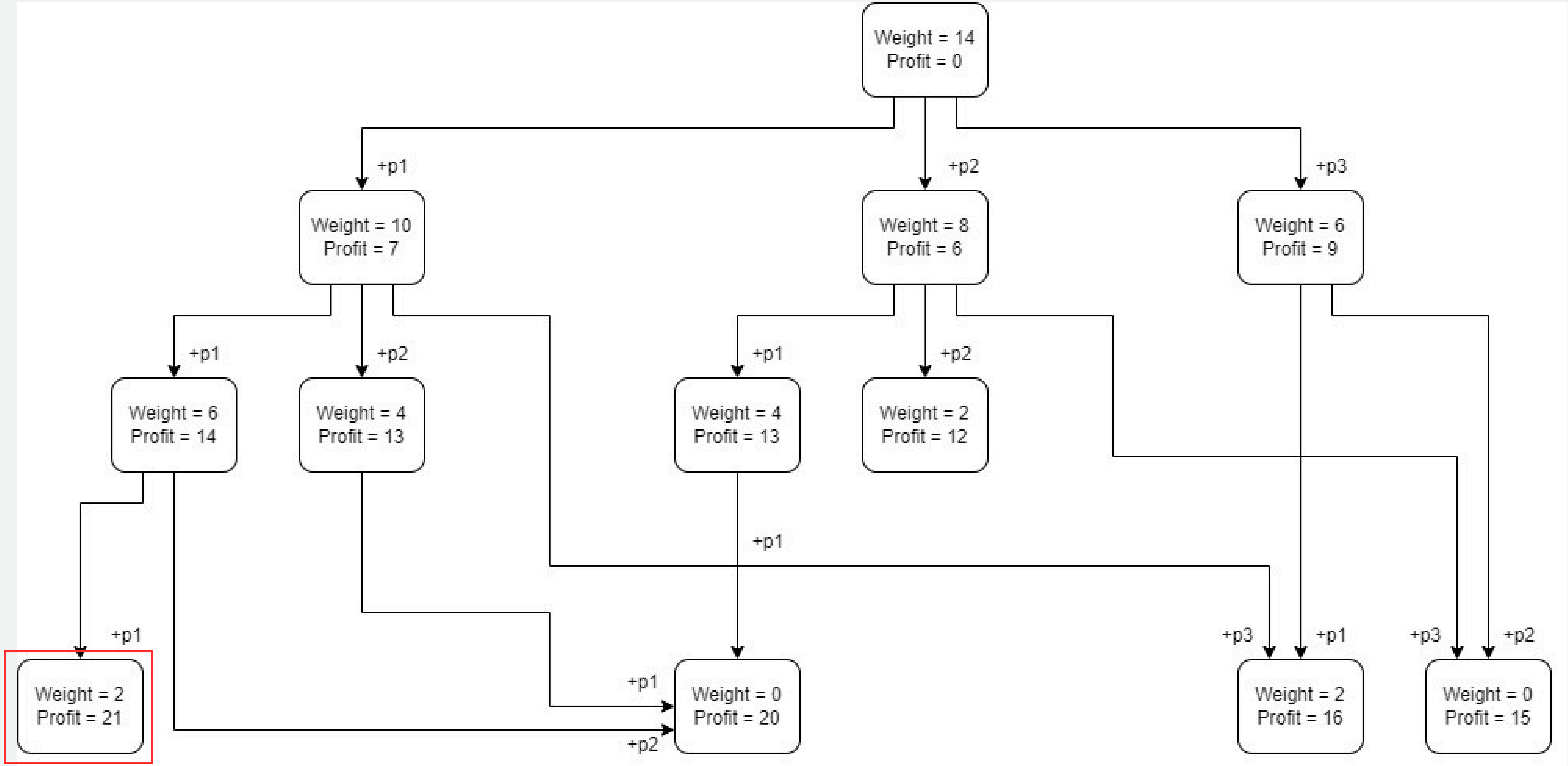
	0	1	2
w_i	4	6	8
p_i	7	6	9

2. Subproblem Graph 1



Red Arrow : P(8)
Green Arrow : P(6)
Blue Arrow : P(4)

2. Subproblem Graph 2



3. Pseudocode

```
Function BottomUpKnapSack(c,n,pi,wi):
        P = 1D of size (c+1), initialized to 0
        for I from 1 to c+1:
              for j from 1 to n:
                    if wi[j] <= i then
                          P[i] = max(P[i],P[i-wi[j]] + pi[j])
              end for
        end for
return P[c]
```

4. Implementation In Python

```python
def BottomUpKnapSack(c,n,p,w):
    P = [0 for i in range (c+1)]

    for i in range(1,c+1):
        for j in range(n):
            if w[j] <= i:
                P[i] = max(P[i], P[i-w[j]] + p[j])

    return P[c]
```

O(c)

O(n)

Time Complexity: O(c*n)

4. Test Cases

```
Part A
--------------------
Capacity is 14
Number of items is 3
Weights are [4, 6, 8]
Values are [7, 6, 9]
Maximum Profit is  21
```

Part A

- Capacity : 14
- No. of Items : 3
- Weights : [4,6,8]
- Values : [7,6,9]

Maximum Profit : 21 (Item 1, Item 1, Item 1)

```
Part B
--------------------
Capacity is 14
Number of items is 3
Weights are [5, 6, 8]
Values are [7, 6, 9]
Maximum Profit is 16
```

Part B

- Capacity : 14
- No. of Items : 3
- Weights : [5,6,8]
- Values : [7,6,9]

Maximum Profit : 16 (Item 2, Item 3)

0 to 3	0
4 to 7	7
8 - 11	14
12 - 14	21

0 to 4	0
5 to 7	7
8 to 9	9
10 to 12	14
13 to 14	16

Thank you for your attention