Project 3

Members : Yi Heng, Potala, Kiefer

What is recursion?

Repeated application of the same procedure on subproblems of the same type of the problem

Recursion in function P(C)

P(C) finds the largest total profit of the set by trying different combinations with N types of objects, each object with different profit P and weight W given a knapsack of capacity C

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) Recursive Definition

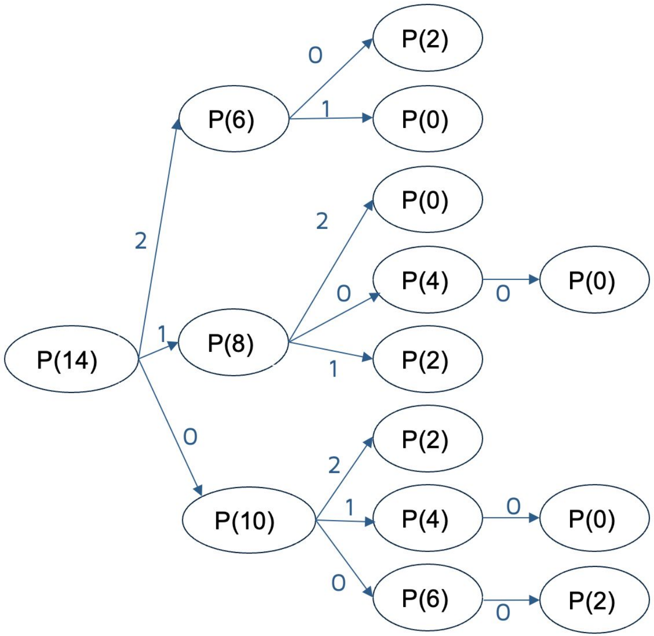
$P(C, 0) = P(0, i) = 0$

$P(C, i) = \max(P(C, i-1), p_i + P(C-w_i, i))$

C = Capacity, w = weight, p = profit, i = object index

2) Subproblem Graph for P(14), n = 3

	0	1	2
w_i	4	6	8
p_i	7	6	9



Profit = 9 + 7 = 16

Profit = 6 + 9 = 15

Profit = 6 + 9 = 15

Profit = 6 + 7 + 7 = 20

Profit = 6 + 6 = 12

Profit = 9 + 7 = 16

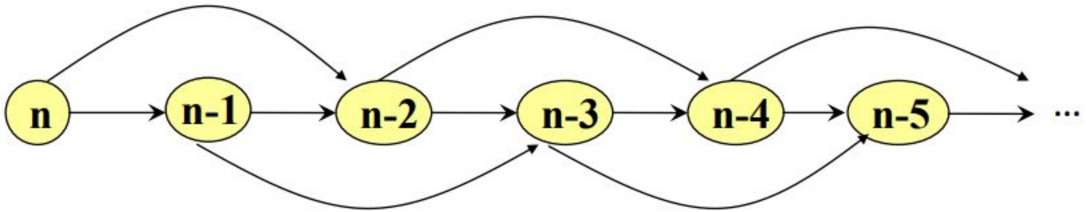
Profit = 7 + 6 + 7 = 20

$\boxed{\text{Profit} = 7 + 7 + 7 = 21}$

(3) Algorithm Implementation (Bottom-up Approach)

- Formulate knapsack problem in smaller capacities, C_1, C_2, C_3, C_4, …
- Instantiate weights in a list w
- Instantiate profit in a list p with identical indexes
- Use a 2D Array to store solution to subproblems
-

The subproblem graph of fib(n)



(3) Algorithm Implementation
(Bottom-up Approach) - Example Case

With C = 11,

Comparing significant possible combinations :

- Item 0 (x2) : Profit 14 (Soln for C = 10) ( )
- Item 0 + Item 1 : Profit 13

With C = 13,

Comparing significant possible combinations :

- Item 0 + Item 1 : Profit 13 (X because previously solved)
- Item 1 (x2) : Profit 12 (X because solved in C = 12)
- Item 0 (x2) : Profit 14 (Soln for C = 11 && C = 12)
- Item 0 + Item 2 : Profit 16 ( )

	0	1	2
w_i	5	6	8
p_i	7	6	9

(3) Algorithm Implementation (Bottom-up Approach) - Time Complexity

Without DP : $O(2^n)$

- Iterate through all possible combinations each time

With DP : $O(n)$

- Linear runtime due to non-repeated solutions

for r = 1 to C
    for c = 1 to n
        profit[r][c] = profit[r][c-1
        if (w[c] <= r)
            if (profit[r][c] <
        profit[r-w[c]][~~c-1~~] + p[c])
                profit[r][c] =
                    profit[r-w[c]][c-
                    + p[c]

(3) Algorithm Implementation (Bottom-up Approach) - Space Complexity

Without DP :

- Don't store any solution

With DP : (C + 1) * n

- 2D Array to store profit for each capacity

```
profit = [[0] * (n + 1) for _ in range(C + 1)]
```

(4) Code Algorithm

```
def knapsack(C, w, p):
    n = len(w)
    profit = [[0] * (n + 1) for _ in range(C + 1)]

    for rC in range(1, C+1): #iterate through capacities
        for cP in range(1, n+1): #iterate through objects
            if w[cP-1] <= rC: #if current object can fit
                profit[rC][cP] = max(profit[rC][cP-1], profit[rC-w[cP-1]][cP]+p[cP-1])
            else: #go next
                profit[rC][cP] = profit[rC][cP-1]

    #print the profit table
    for row in profit:
        print(row)

    return profit[C][n]
```

Initialise 2D Array with row = capacity, column = object, first row & col = 0

Find the max of (don’t include object, include object)

Difference between original and lab’s
Instead of comparing with n-1, compare with same n.

(4) Code Algorithm

```
#Main
C = 14
w = [5, 6, 8]
p = [7, 6, 9]
max_profit = knapsack(C, w, p)
print("The maximum profit is:", max_profit)
```

C -> capacity
w -> weight array
p -> profit array

(4a) Code Algorithm; P(14)

	0	1	2
w_i	4	6	8
p_i	7	6	9

n

C

```
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 7, 7, 7]
[0, 7, 7, 7]
[0, 7, 7, 7]
[0, 7, 7, 7]
[0, 14, 14, 14]
[0, 14, 14, 14]
[0, 14, 14, 14]
[0, 14, 14, 14]
[0, 21, 21, 21]
[0, 21, 21, 21]
[0, 21, 21, 21]
The maximum profit is: 21
```

(4b) Code Algorithm; P(14)

	0	1	2
w_i	5	6	8
p_i	7	6	9

n

C

```
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 7, 7, 7]
[0, 7, 7, 7]
[0, 7, 7, 7]
[0, 7, 7, 9]
[0, 7, 7, 9]
[0, 14, 14, 14]
[0, 14, 14, 14]
[0, 14, 14, 14]
[0, 14, 14, 16]
[0, 14, 14, 16]
The maximum profit is: 16
```