Rod Cutting

We'll cover the following

Problem Statement

Given a rod of length ‘n’, we are asked to cut the rod and sell the pieces in a way that will maximize the profit. We are also given the price of every piece of length ‘i’ where ‘1 <= i <= n’.

Example:

Lengths: [1, 2, 3, 4, 5]
Prices: [2, 6, 7, 10, 13]
Rod Length: 5

Let’s try different combinations of cutting the rod:

Five pieces of length 1 => 10 price
Two pieces of length 2 and one piece of length 1 => 14 price
One piece of length 3 and two pieces of length 1 => 11 price
One piece of length 3 and one piece of length 2 => 13 price
One piece of length 4 and one piece of length 1 => 12 price
One piece of length 5 => 13 price

This shows that we get the maximum price (14) by cutting the rod into two pieces of length ‘2’ and one piece of length ‘1’.

Basic Solution

This problem can be mapped to the Unbounded Knapsack (https://www.educative.io/collection/page/5668639101419520/5633779737559040/5745865499082752/) pattern. The Weights array of the Unbounded Knapsack problem is equivalent to Lengths array, and Profits is equivalent to Prices.

A basic brute-force solution could be to try all combinations of the given rod lengths to choose the one with the maximum sale price. This is what our algorithm will look like:

```
for each rod length 'i'
  create a new set which includes one quantity of length 'i', and recursively process
      all rod lengths for the remaining length
  create a new set without rod length 'i', and recursively process for remaining rod lengt
return the set from the above two sets with a higher sales price
```

Since this problem is quite similar to Unbounded Knapsack (https://www.educative.io/collection/page/5668639101419520/5633779737559040/5745865499082752/), let's jump directly to the bottom-up dynamic solution.

Bottom-up Dynamic Programming

Let's try to populate our `dp[][]` array in a bottom-up fashion. Essentially, what we want to achieve is: "Find the maximum sales price for every rod length and for every possible sales price".

So for every possible rod length 'len' (0<= len <= n), we have two options:

1. Exclude the piece. In this case, we will take whatever price we get from the rod length excluding this piece => `dp[index-1][len]`
2. Include the piece if its length is not more than 'len'. In this case, we include its price plus whatever price we get from the remaining rod length => `prices[index] + dp[index][len-lengths[index]]`

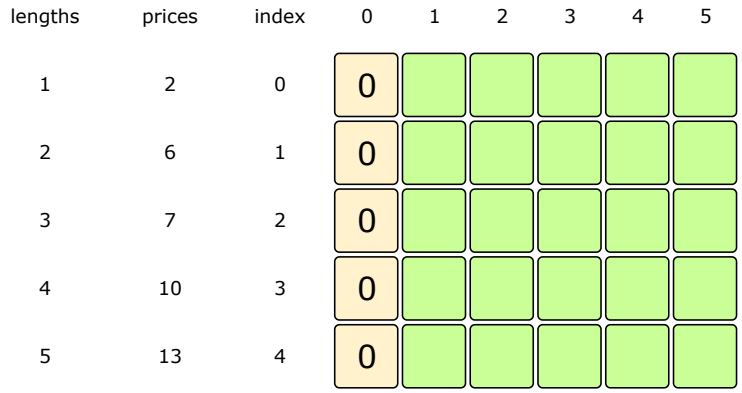
Finally, we have to take the maximum of the above two values:

```
    dp[index][len] = max (dp[index-1][len], prices[index] + dp[index][len-lengths[
index]])
```

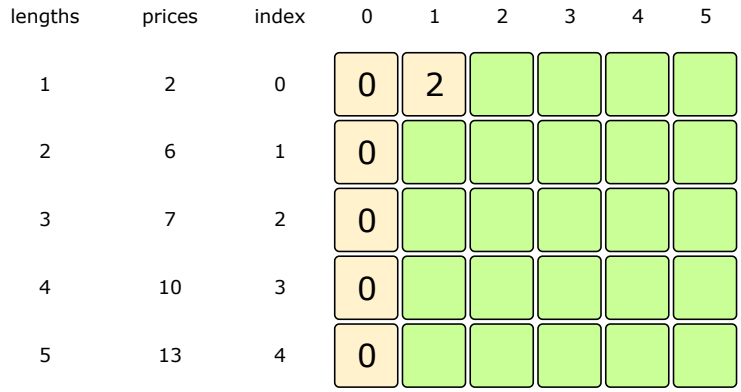
Let's draw this visually, with the example:

```
Lengths: [1, 2, 3, 4, 5]
Prices: [2, 6, 7, 10, 13]
Rod Length: 5
```

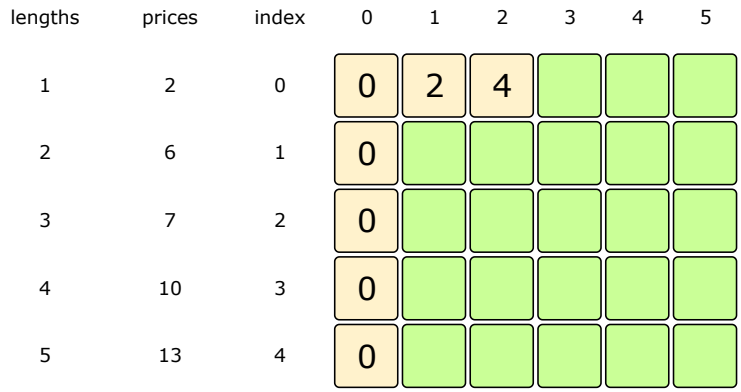
Let's start with our base case of zero size:



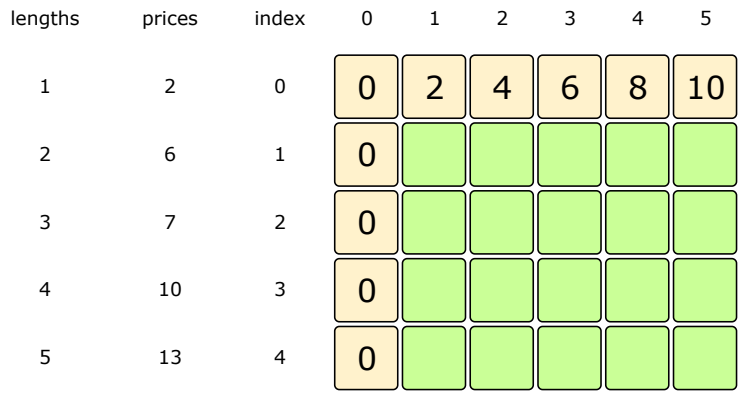
With '0' capacity, maximum profit we can have for every subarray is '0'



Length = 1, Index = 0, i.e., if we consider the sub-array till index '0', maximum price will be '2' from the rod of length '1'



Length = 2, Index = 0, => prices[Index] + dp[Index][1], we are not considering dp[Index-1][Length] as Index is not bigger than '0'



Length = 3-5, Index = 0, => prices[Index] + dp[Index][1]

lengths	prices	index	0	1	2	3	4	5
1	2	0	0	2	4	6	8	10
2	6	1	0	2				
3	7	2	0					
4	10	3	0					
5	13	4	0					

Length = 1, Index =1, since item at index '1' has length '2', which is greater than the maximum length '1', so we will take the dp[Index-1][Length]

lengths	prices	index	0	1	2	3	4	5
1	2	0	0	2	4	6	8	10
2	6	1	0	2	6			
3	7	2	0					
4	10	3	0					
5	13	4	0					

Length = 2, Index =1, from the formula discussed above: max(dp[Index-1][Length], prices[Index] + dp[Index][Length-lengths[index]])

lengths	prices	index	0	1	2	3	4	5
1	2	0	0	2	4	6	8	10
2	6	1	0	2	6	8		
3	7	2	0					
4	10	3	0					
5	13	4	0					

Length = 3, Index =1, from the formula discussed above: max(dp[Index-1][Length], prices[Index] + dp[Index][Length-lengths[index]])

lengths	prices	index	0	1	2	3	4	5
1	2	0	0	2	4	6	8	10
2	6	1	0	2	6	8	12	
3	7	2	0					
4	10	3	0					
5	13	4	0					

Length = 4, Index =1, from the formula discussed above: max(dp[Index-1][Length], prices[Index] + dp[Index][Length-lengths[index]])

lengths	prices	index	0	1	2	3	4	5
1	2	0	0	2	4	6	8	10
2	6	1	0	2	6	8	12	14
3	7	2	0					
4	10	3	0					
5	13	4	0					

Length = 5, Index =1, from the formula discussed above: max(dp[Index-1][Length], prices[Index] + dp[Index][Length-lengths[index]])

lengths	prices	index	0	1	2	3	4	5
1	2	0	0	2	4	6	8	10
2	6	1	0	2	6	8	12	14
3	7	2	0	2				
4	10	3	0					
5	13	4	0					

Length = 1, Index =2, since item at index '2' has length '3', which is greater than the maximum length '1', so we will take the dp[Index-1][Length]

lengths	prices	index	0	1	2	3	4	5
1	2	0	0	2	4	6	8	10
2	6	1	0	2	6	8	12	14
3	7	2	0	2	6			
4	10	3	0					
5	13	4	0					

Length = 2, Index =2, since item at index '2' has length '3', which is greater than the maximum length '2', so we will take the dp[Index-1][Length]

lengths	prices	index	0	1	2	3	4	5
1	2	0	0	2	4	6	8	10
2	6	1	0	2	6	8	12	14
3	7	2	0	2	6	8		
4	10	3	0					
5	13	4	0					

Length = 3, Index =2, from the formula discussed above: max(dp[Index-1][Length], prices[Index] + dp[Index][Length-lengths[index]])

lengths	prices	index	0	1	2	3	4	5
1	2	0	0	2	4	6	8	10
2	6	1	0	2	6	8	12	14
3	7	2	0	2	6	8	12	
4	10	3	0					
5	13	4	0					

Length = 4, Index =2, from the formula discussed above: max(dp[Index-1][Length], prices[Index] + dp[Index][Length-lengths[index]])

lengths	prices	index	0	1	2	3	4	5
1	2	0	0	2	4	6	8	10
2	6	1	0	2	6	8	12	14
3	7	2	0	2	6	8	12	14
4	10	3	0					
5	13	4	0					

Length = 5, Index =2, from the formula discussed above: max(dp[Index-1][Length], prices[Index] + dp[Index][Length-lengths[index]])

lengths	prices	index	0	1	2	3	4	5
1	2	0	0	2	4	6	8	10
2	6	1	0	2	6	8	12	14
3	7	2	0	2	6	8	12	14
4	10	3	0	2	6	8		
5	13	4	0					

Length = 1-3, Index =3, since item at index '3' has length '4', which is greater than the maximum length '1-3', so we will take the dp[Index-1][Length]

lengths	prices	index	0	1	2	3	4	5
1	2	0	0	2	4	6	8	10
2	6	1	0	2	6	8	12	14
3	7	2	0	2	6	8	12	14
4	10	3	0	2	6	8	12	
5	13	4	0					

Length = 4, Index =3, from the formula discussed above: max(dp[Index-1][Length], prices[Index] + dp[Index][Length-lengths[index]])

lengths	prices	index	0	1	2	3	4	5
1	2	0	0	2	4	6	8	10
2	6	1	0	2	6	8	12	14
3	7	2	0	2	6	8	12	14
4	10	3	0	2	6	8	12	14
5	13	4	0					

Length = 4, Index =3, from the formula discussed above: max(dp[Index-1][Length], prices[Index] + dp[Index][Length-lengths[index]])

17 of 19

lengths	prices	index	0	1	2	3	4	5
1	2	0	0	2	4	6	8	10
2	6	1	0	2	6	8	12	14
3	7	2	0	2	6	8	12	14
4	10	3	0	2	6	8	12	14
5	13	4	0	2	6	8	12	

Length = 1-4, Index =4, since item at index '4' has length '5', which is greater than the maximum length '1-4', so we will take the dp[Index-1][Length]

18 of 19

lengths	prices	index	0	1	2	3	4	5
1	2	0	0	2	4	6	8	10
2	6	1	0	2	6	8	12	14
3	7	2	0	2	6	8	12	14
4	10	3	0	2	6	8	12	14
5	13	4	0	2	6	8	12	14

Length = 5, Index =4, from the formula discussed above: max(dp[Index-1][Length], prices[Index] + dp[Index][Length-lengths[index]])

19 of 19

Code #

Here is the code for our bottom-up dynamic programming approach:

Java JS Python3 C++

```python
def solve_rod_cutting(lengths, prices, n):
  lengthCount = len(lengths)
  # base checks
  if n <= 0 or lengthCount == 0 or len(prices) != lengthCount:
    return 0

  dp = [[0 for _ in range(n+1)] for _ in range(lengthCount)]

  # process all rod lengths for all prices
  for i in range(lengthCount):
    for length in range(1, n+1):
      p1, p2 = 0, 0
      if lengths[i] <= length:
        p1 = prices[i] + dp[i][length - lengths[i]]
      if i > 0:
        p2 = dp[i - 1][length]
      dp[i][length] = max(p1, p2)

  # maximum price will be at the bottom-right corner.
  return dp[lengthCount - 1][n]


def main():
  print(solve_rod_cutting([1, 2, 3, 4, 5], [2, 6, 7, 10, 13], 5))


main()

```

The above solution has time and space complexity of $O(N * C)$, where 'N' represents total items and 'C' is the maximum capacity.

Find the selected items #

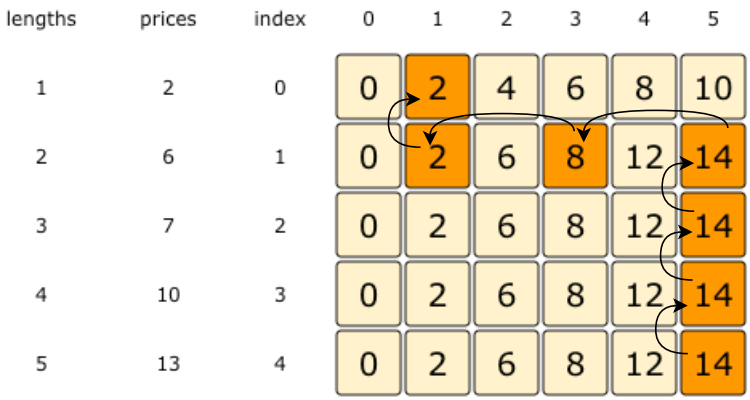
As we know, the final price is at the right-bottom corner; hence we will start from there to find the rod lengths.

As you remember, at every step we had two options: include a rod piece or skip it. If we skip it, then we take the price from the cell right above it; if we include it, then we jump to the remaining length to find more pieces.

Let's understand this from the above example:

1. '14' did come from the top cell, so we jump to the fourth row.
2. '14' came from the top cell, so we jump to the third row.
3. Again, '14' came from the top cell, so we jump to the second row.

4. Now ‘14’ is different from the top cell, so we must include rod of length ‘2’. After this, we subtract the price of the rod of length ‘2’ from ‘14’ and jump to that cell.
5. ‘8’ is different than the top cell, so we must include rod of length ‘2’ again. After this, we subtract the price of the rod of length ‘2’ from ‘8’ and jump to that cell.
6. ‘2’ did come from the top cell, so we jump to the first row.
7. Now we must include a piece of length ‘1’. So the desired rod lengths are {2, 2, 1}.



Back

Unbounded Knapsack

Next

Coin Change

Mark as Completed