Dynamic Programming
CSCI 432

Rod Cutting

length	1	2	3	4
profit	$1	$5	$8	$9

Given a rod of length n and a table of prices,
determine the maximum profit obtainable
by cutting the rod up and selling the pieces.

Rod Cutting

length	1	2	3	4
profit	\$1	\$5	\$8	\$9

profit = \$9

Given a rod of length n and a table of prices, determine the maximum profit obtainable by cutting the rod up and selling the pieces.

Rod Cutting

length	1	2	3	4
profit	$1	$5	$8	$9

profit = $9

profit = $9

Given a rod of length n and a table of prices, determine the maximum profit obtainable by cutting the rod up and selling the pieces.

Rod Cutting

length	1	2	3	4
profit	$1	$5	$8	$9

profit = $9

profit = $9

profit = $9

Given a rod of length n and a table of prices, determine the maximum profit obtainable by cutting the rod up and selling the pieces.

Rod Cutting

length	1	2	3	4
profit	$1	$5	$8	$9

profit = $9

profit = $9

profit = $10

profit = $7

profit = $4

Given a rod of length n and a table of prices,
determine the maximum profit obtainable
by cutting the rod up and selling the pieces.

Rod Cutting

Assume there is a profit ≥ 0 for each possible length.

length	1	2	3	4
profit	$1	$5	$8	$9

profit = $9

profit = $9

profit = $10

profit = $7

profit = $4

Given a rod of length n and a table of prices, determine the maximum profit obtainable by cutting the rod up and selling the pieces.

Rod Cutting

Can we use dynamic programming?

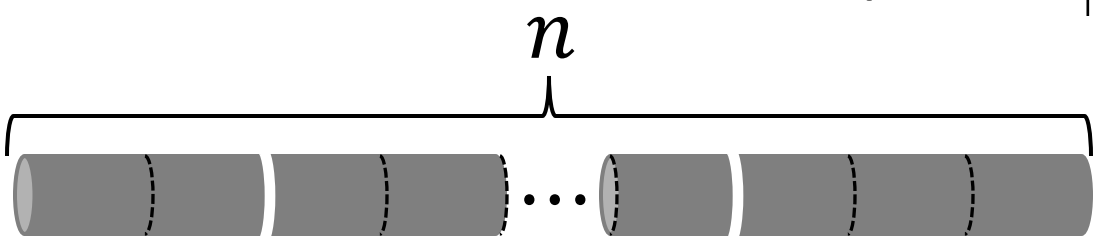
Rod Cutting

Does the rod cutting problem have optimal substructure?

Rod Cutting

Assume there is a profit ≥ 0
for each possible length.

length	1	2	3	4	…	n
profit	\$1	\$5	\$8	\$9	…	\$?

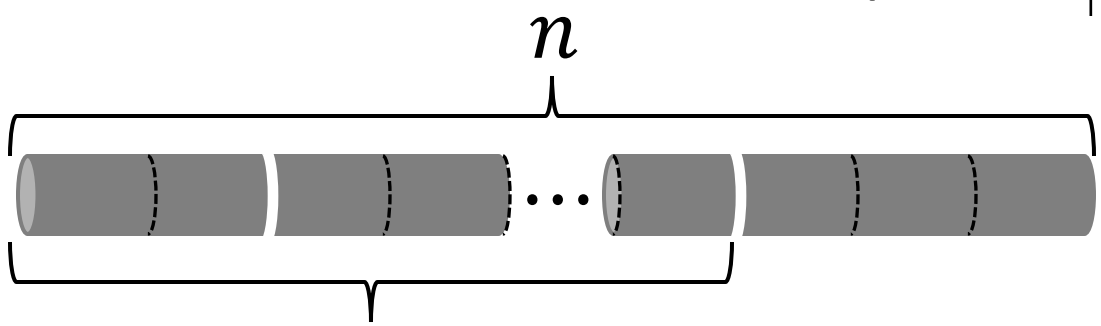


Does the rod cutting problem have optimal substructure?

I.e. Does an optimal solution to an instance
imply an optimal solution to a smaller instance?

Rod Cutting

length	1	2	3	4	…	n
profit	\$1	\$5	\$8	\$9	…	\$?

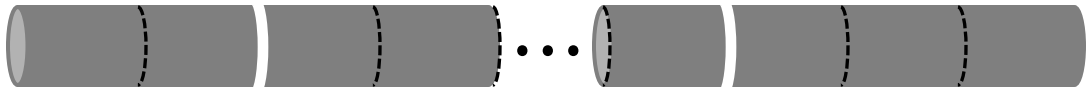


Does the rod cutting problem have optimal substructure?

Yes! Given an optimal partition for n, every subset of that partition must also be optimal for its size!

Rod Cutting

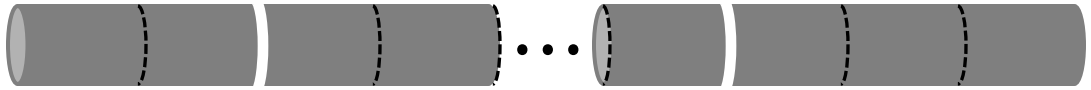
length	1	2	3	4	…	n
profit	\$1	\$5	\$8	\$9	…	\$?



O_n = optimal profit from partitioning rod of length n.
p_i = profit for rod of length i.

Rod Cutting

length	1	2	3	4	…	n
profit	\$1	\$5	\$8	\$9	…	\$?



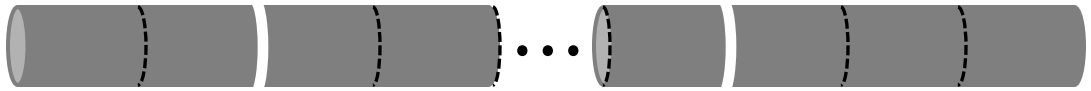
O_n = optimal profit from partitioning rod of length n.
p_i = profit for rod of length i.

$$O_n = ?$$

How can we recursively calculate O_n from smaller solutions, leveraging the optimal substructure?

Rod Cutting

length	1	2	3	4	…	n
profit	\$1	\$5	\$8	\$9	…	\$?



O_n = optimal profit from partitioning rod of length n.
p_i = profit for rod of length i.

$$O_n = \max_{1 \le i \le n} (p_i + O_{n-i})$$

Rod Cutting

Algorithm Plan:

?

Rod Cutting

Algorithm Plan:

1. Initialize array to hold what we care about.

Rod Cutting

Algorithm Plan:

1. Initialize array to hold what we care about.
2. Fill out array from small to large.

Rod Cutting

Algorithm Plan:

1. Initialize array to hold what we care about.
2. Fill out array from small to large.
3. Fill out array using:

$$O_n = \max_{1 \le i \le n} (p_i + O_{n-i})$$

Rod Cutting

Algorithm?

Algorithm Plan:

1. Initialize array to hold what we care about.
2. Fill out array from small to large.
3. Fill out array using:

$$O_n = \max_{1 \le i \le n} (p_i + O_{n-i})$$

Rod Cutting

```
partition(n, p)
   r[0,...,n] = [0,...,0]
   for m = 1 to n
      max = 0
      for length l ≤ m
         if r[m – l] + p_l > max
            max = r[m – l] + p_l
      r[m] = max
   return r[n]
```

Algorithm Plan:

1. Initialize array to hold what we care about.
2. Fill out array from small to large.
3. Fill out array using:

$$O_n = \max_{1 \le i \le n} (p_i + O_{n-i})$$

Rod Cutting

```
partition(n, p)
   r[0,...,n] = [0,...,0]
   for m = 1 to n
      max = 0
      for length l ≤ m
         if r[m – l] + p_l > max
            max = r[m – l] + p_l
      r[m] = max
   return r[n]
```

Running Time?

Rod Cutting

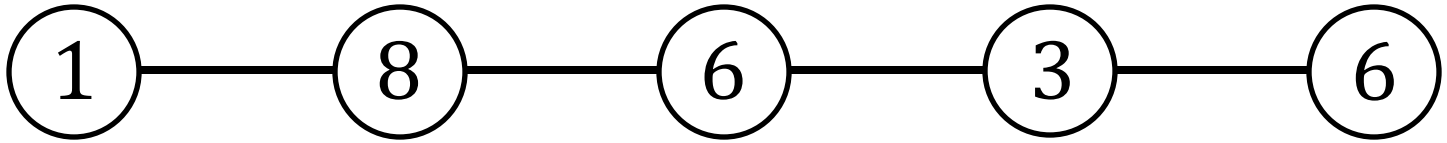
```
partition(n, p)
   r[0,...,n] = [0,...,0]
   for m = 1 to n
      max = 0
      for length l ≤ m
         if r[m – l] + p_l > max
            max = r[m – l] + p_l
      r[m] = max
   return r[n]
```

Running Time?

$O(n^2)$

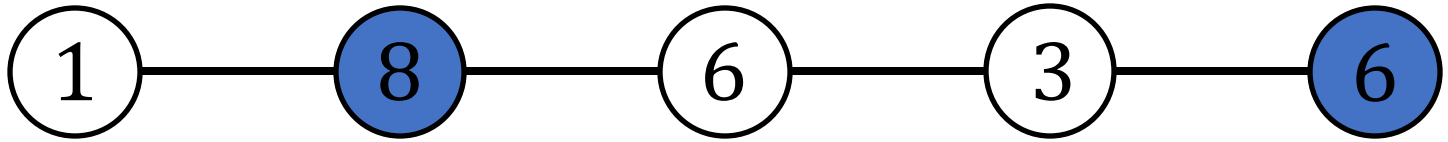
Problem Statement

Suppose we have a graph of this form:



Goal: Find a subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

Example:



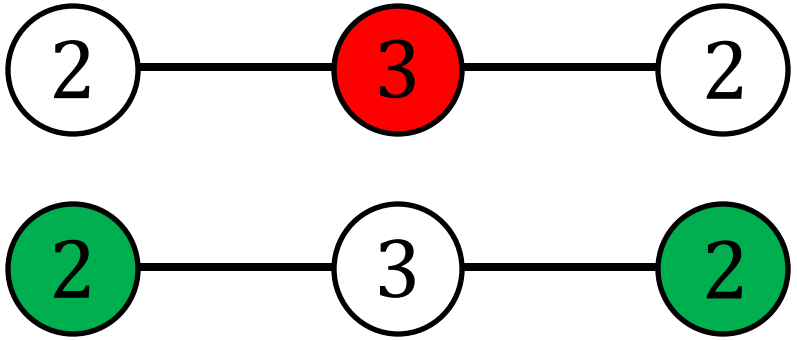
Find counter example for this algorithm:

```
maxWeight1(G = (V, E))
    S = Ø
    while V ≠ Ø
      Let v_i be node of max weight from V
      Add v_i to S
      Delete v_i and neighbors from G
    return S
```

Goal: Find a subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

Find counter example for this algorithm:

```
maxWeight1(G = (V, E))
    S = Ø
    while V ≠ Ø
      Let v_i be node of max weight from V
      Add v_i to S
      Delete v_i and neighbors from G
    return S
```



Goal: Find a subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

Find counter example for this algorithm:

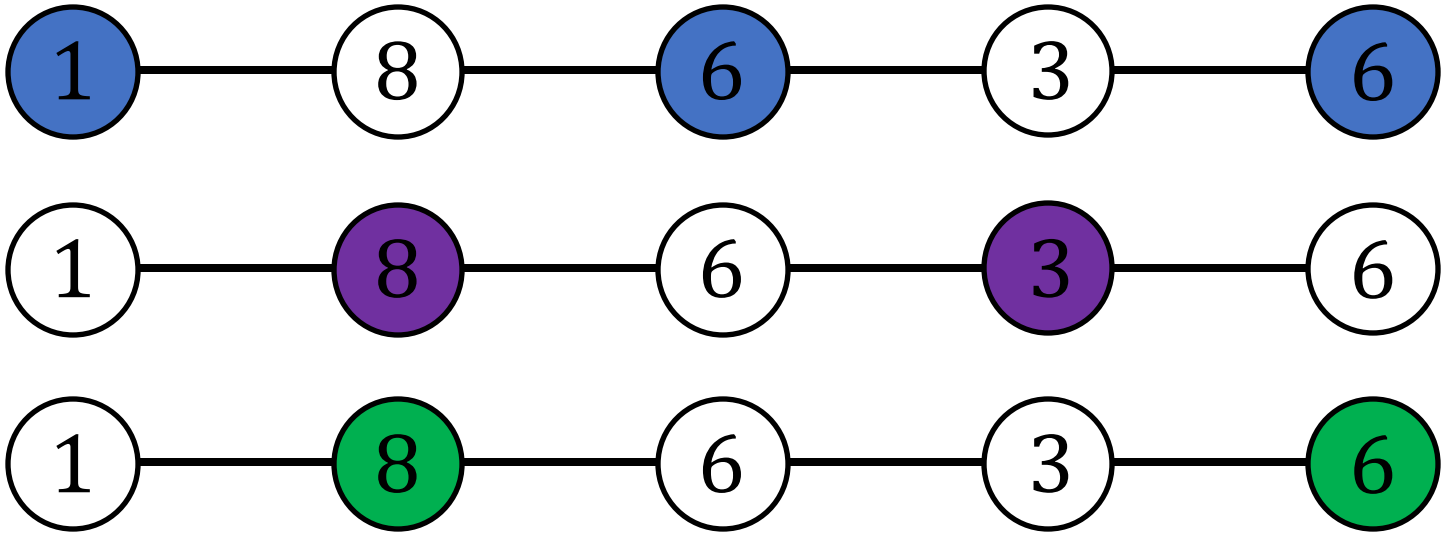
```
maxWeight2(G = (V, E))
    Let S1 = vi where i is odd
    Let S2 = vi where i is even
    return maxWeight(S1, S2)
```

Goal: Find a subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

Find counter example for this algorithm:

```
maxWeight2(G = (V, E))
    Let S1 = vi where i is odd
    Let S2 = vi where i is even
    return maxWeight(S1, S2)
```

Goal: Find a subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.



Make an algorithm:

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

Make an algorithm:

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

Is there an optimal substructure? Does an optimal solution to $V_{i-1} = \{v_1, \ldots, v_{i-1}\}$ translate to an optimal solution to $V_i = \{v_1, \ldots, v_i\}$?

Make an algorithm:

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

Is there an optimal substructure? Does an optimal solution to $V_{i-1} = \{v_1, \ldots, v_{i-1}\}$ translate to an optimal solution to $V_i = \{v_1, \ldots, v_i\}$?

Yes(ish). Optimal solution to V_i is either the optimal for V_{i-1} (with vertex i discarded), or the optimal for V_{i-2} (with vertex i included).

Make an algorithm:

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

Optimal solution to V_i is either the optimal for V_{i-1} (with vertex i discarded), or the optimal for V_{i-2} (with vertex i included).

Let $A[i]$ be the weight of the optimal solution for V_i.

$A[i] = ?$

Make an algorithm:

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

Optimal solution to V_i is either the optimal for V_{i-1} (with vertex i discarded), or the optimal for V_{i-2} (with vertex i included).

Let $A[i]$ be the weight of the optimal solution for V_i.

$$A[i] = \max \begin{pmatrix} A[i-1] \\ A[i-2] + w_i \end{pmatrix}$$

Make an algorithm:

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
```

?

$A[i]$ – weight of the optimal solution for V_i.

$$A[i] = \max \begin{pmatrix} A[i-1] \\ A[i-2] + w_i \end{pmatrix}$$

Running time ∈ ?

Make an algorithm:

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w_1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + w_i)
    return A[n]
```

$A[i]$ – weight of the optimal solution for V_i.

$$A[i] = \max\begin{pmatrix} A[i-1] \\ A[i-2] + w_i \end{pmatrix}$$

Running time $\in O(n)$