

ROD CUTTING PROBLEMI

Dinamik Programlama

İÇERİK

- Problem Tanımı
- DP Yaklaşımı ve Recurrence Relation
- Kod ve Açıklamalar
- Örnek Uygulama (tablo dahil)
- Zaman / Bellek Analizi
- Sonuç

PROBLEM TANIMI

Rod Cutting (Çubuk Kesme) problemi, belirli uzunluktaki bir çubuğun farklı boyutlarda kesilerek satılabildeği bir optimizasyon problemidir.

Her bir uzunluk için birim fiyatlar verilmiştir. Amaç, çubuğu uygun uzunluklarda kesip satarak maksimum toplam geliri elde etmektir.

◆ Problem Özeti:

Bir uzunluk **n** ve fiyat dizisi **price[]** veriliyor.

- **price[i]**: uzunluğu **i+1** olan bir çubuğun satış fiyatı
- Her bir kesimde farklı uzunlıklar elde edilebilir.
- Çubuk, hiç kesilmenden de satılabilir.

DP YAKLAŞIMI

Rod Cutting Problemi, optimal alt yapı (optimal substructure) özelliğine sahip bir problemdir.

Yani, problemin en iyi çözümü, alt problemlerin en iyi çözümlerinden oluşturulabilir.

- ◆ DP Yaklaşımı:
 - Her olası kesim noktası denenir.
 - Kesilen parçanın fiyatı + geriye kalan parçanın en yüksek getirisi hesaplanır.
 - Bu işlemlerle n uzunluğu için maksimum gelir bulunur.

RECURRENCE RELATION

Formül: $R(n) = \max(\text{price}[i-1] + R(n - i)), 1 \leq i \leq n$

Taban durum(Base Case): $R(0) = 0$

Burada:

- **$R(n)$** = uzunluğu n olan çubuğun maksimum geliri
- **$\text{price}[i-1]$** = uzunluğu i olan parçanın fiyatı
- **$R(n - i)$** = kalan kısmın maksimum getirisi

PYTHON KODU

```
def rod_cutting(prices, n):  
    dp = [0] * (n + 1)  
  
    # dp[i]: uzunluğu i olan çubuk için maksimum gelir  
    for i in range(1, n + 1):  
        max_val = float('-inf')  
        for j in range(1, i + 1):  
            max_val = max(max_val, prices[j - 1] + dp[i - j])  
        dp[i] = max_val  
  
    return dp[n], dp  
  
# Örnek kullanım  
prices = [1, 5, 8, 9, 10, 17, 17, 20]  
n = 8  
result, table = rod_cutting(prices, n)  
print("Maksimum gelir:", result)  
print("DP Tablosu:", table)
```

- ◆ Açıklama:
 - **dp[i]**: uzunluğu i olan çubuğun kazancını tutar.
 - İç döngü, her olası kesim uzunluğu j için fiyatı hesaplar.
 - En büyük değer seçilerek **dp[i]** güncellenir.

ÖRNEK UYGULAMA

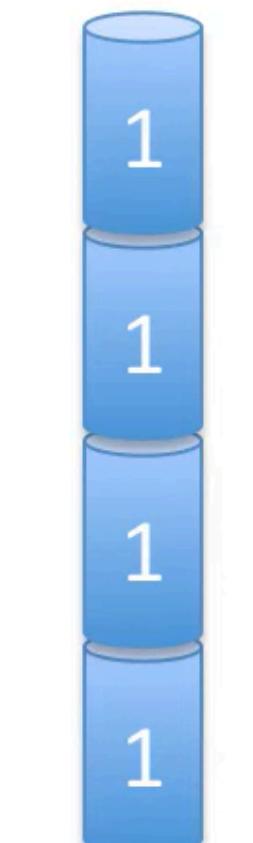


The Rod

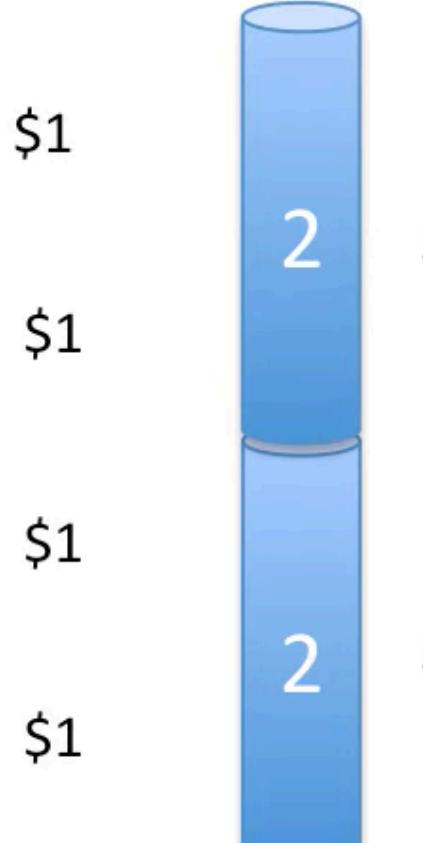
Length	1	2	3	4	5	6	7	8
Price \$	1	5	8	9	10	17	17	20

4

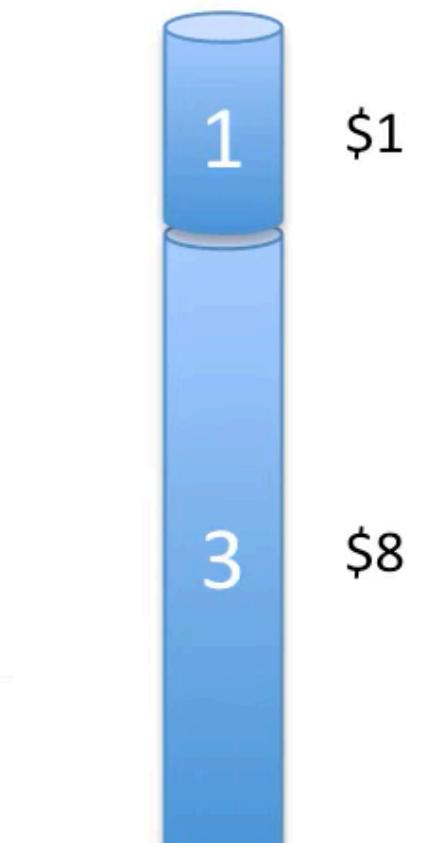
Possible Combinations:



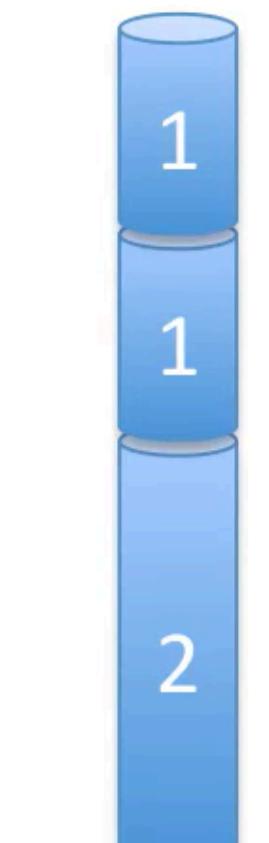
Total: \$4



Total: \$10



Total: \$9



Total: \$7



Total: \$9

Rod Cutting DP

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

Length	1	2	3	4	5	6	7	8
Price \$	1	5	8	9	10	17	17	20
Len (i)	1	2	3	4	5	6	7	8

C(i)

Opt

$$C(1) = 1$$

Rod Cutting DP

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

Length	1	2	3	4	5	6	7	8
Price \$	1	5	8	9	10	17	17	20
Len (i)	1	2	3	4	5	6	7	8

$C(i)$

Opt

$$C(2) = \max \left[\begin{array}{l} V_1 + C(1) = 1 + 1 = 2 \\ V_2 = 5 \end{array} \right]$$


Rod Cutting DP

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

Length	1	2	3	4	5	6	7	8
Price \$	1	5	8	9	10	17	17	20
Len (i)	1	2	3	4	5	6	7	8

C(i)

Opt	1	5						
-----	---	---	--	--	--	--	--	--

$$C(3) = \max \begin{cases} V_1 + C(2) = 1 + 5 = 6 \\ V_2 + C(1) = 5 + 1 = 6 \\ V_3 = 8 \end{cases}$$

Rod Cutting DP

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

Length	1	2	3	4	5	6	7	8
Price \$	1	5	8	9	10	17	17	20
Len (i)	1	2	3	4	5	6	7	8

C(i)

Opt	1	5	8					
-----	---	---	---	--	--	--	--	--

$$C(4) = \max \left\{ \begin{array}{l} V_1 + C(3) = 1 + 8 = 9 \\ V_2 + C(2) = 5 + 5 = 10 \\ V_3 + C(1) = 8 + 1 = 9 \\ V_4 = 9 \end{array} \right.$$

Rod Cutting DP

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

Length	1	2	3	4	5	6	7	8
Price \$	1	5	8	9	10	17	17	20
Len (i)	1	2	3	4	5	6	7	8

$C(i)$

Opt	1	5	8	10				
-----	---	---	---	----	--	--	--	--

$$C(5) = \max \left[\begin{array}{l} V_1 + C(4) = 1 + 10 = 11 \\ V_2 + C(3) = 5 + 8 = 13 \\ V_3 + C(2) = 8 + 5 = 13 \\ V_4 + C(1) = 9 + 1 = 10 \\ V_5 = 10 \end{array} \right]$$


Rod Cutting DP

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

Length	1	2	3	4	5	6	7	8
Price \$	1	5	8	9	10	17	17	20
Len (i)	1	2	3	4	5	6	7	8

C(i)

Opt	1	5	8	10	13			
-----	---	---	---	----	----	--	--	--

$$C(6) = \max \left\{ \begin{array}{l} V_1 + C(5) = 1 + 13 = 14 \\ V_2 + C(4) = 5 + 10 = 15 \\ V_3 + C(3) = 8 + 8 = 16 \\ V_4 + C(2) = 9 + 5 = 14 \\ V_5 + C(1) = 10 + 1 = 11 \\ V_6 = 17 \end{array} \right.$$

Rod Cutting DP

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

Length	1	2	3	4	5	6	7	8
Price \$	1	5	8	9	10	17	17	20
Len (i)	1	2	3	4	5	6	7	8

C(i)

Opt	1	5	8	10	13	17		
-----	---	---	---	----	----	----	--	--

$$C(7) = \max \left[\begin{array}{l} V_1 + C(6) = 1 + 17 = 18 \\ V_2 + C(5) = 5 + 13 = 18 \\ V_3 + C(4) = 8 + 10 = 18 \\ V_4 + C(3) = 9 + 8 = 17 \\ V_5 + C(2) = 10 + 5 = 15 \\ V_6 + C(1) = 17 + 1 = 18 \\ V_7 = 17 \end{array} \right]$$

Rod Cutting DP

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

Length	1	2	3	4	5	6	7	8
Price \$	1	5	8	9	10	17	17	20
Len (i)	1	2	3	4	5	6	7	8

C(i)

Opt	1	5	8	10	13	17	18	
-----	---	---	---	----	----	----	----	--

$$C(8) = \max \left[\begin{array}{l} V_1 + C(7) = 1 + 18 = 19 \\ V_2 + C(6) = 5 + 17 = 22 \\ V_3 + C(5) = 8 + 13 = 21 \\ V_4 + C(4) = 9 + 10 = 19 \\ V_5 + C(3) = 10 + 8 = 18 \\ V_6 + C(2) = 17 + 5 = 22 \\ V_7 + C(1) = 17 + 1 = 18 \\ V_8 = 20 \end{array} \right]$$

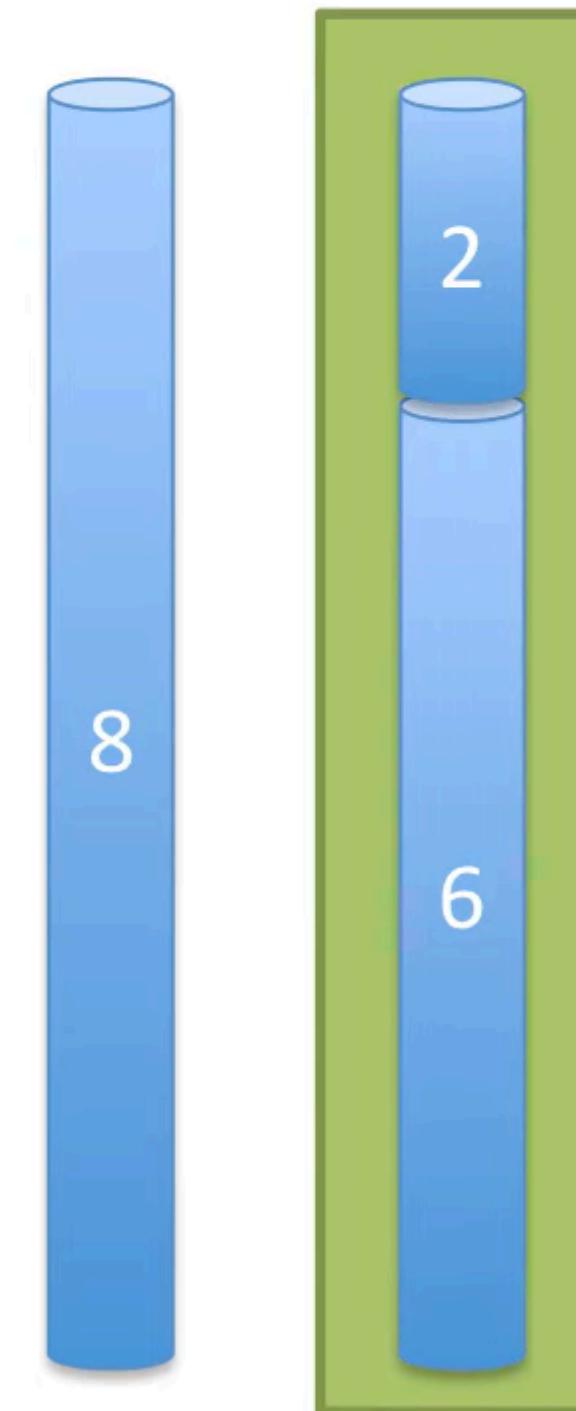


Rod Cutting DP

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

$$C(8) = V_2 + C(6) = 5 + 17 = 22$$

$$C(6) = V_6 = 17$$



Length	1	2	3	4	5	6	7	8
Price \$	1	5	8	9	10	17	17	20
Len (i)	1	2	3	4	5	6	7	8

C(i)

Opt	1	5	8	10	13	17	18	22
-----	---	---	---	----	----	----	----	----

8 birim uzunluğundaki çubuk, 2 + 6 şeklinde kesildiğinde maksimum gelir 22 elde edilir

ZAMAN VE BELLEK ANALİZİ

Analiz Türü

Zaman Karmaşıklığı

Bellek Karmaşıklığı

Karmaşıklık

$O(n^2)$

$O(n)$

Açıklama

Her uzunluk için tüm kesim noktaları denenir

Tek boyutlu DP dizisi ($dp[]$) kullanılır

Bu çözüm zaman açısından verimli ve bellek açısından optimize edilmiştir. Brute-force yöntemi ($O(2^n)$) yerine DP ($O(n^2)$) kullanmak büyük bir performans kazancı sağlar.

SONUÇ

Rod Cutting problemi, Dinamik Programlama'nın temel mantığını göstermesi açısından oldukça öğreticidir.

Bu problemde karmaşık bir optimizasyon, alt problemlere bölünerek verimli biçimde çözülür.

DP yaklaşımı sayesinde:

- Gereksiz tekrar hesaplamalar önlenir,
- Zaman verimliliği sağlanır,
- Alt problemlerin sonuçları kullanılarak optimal çözüm elde edilir.

**DİNLEDİĞİNİZ İÇİN
TEŞEKKÜRLER!**