ZİNCİR MATRİS ÇARPIMI

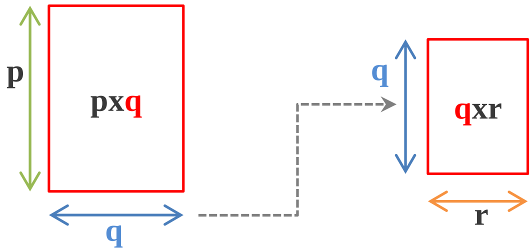
A1 x A2 x A3 matrisleri

Sadece hesaplama kısmı ile ilgileniyoruz!

$$\begin{aligned}
&count = M[i][k] + M[k+1][j] + p[i-1]\cdot p[k]\cdot p[j] \\
&\qquad \text{if } (count < M[i][j]) \\
&\qquad\qquad M[i][j] = count \\
&\text{return } M[1][n-1];
\end{aligned}$$

Özyinelemeyi dizi üzerinden yapıyoruz

pxq qxr
3x5 5x2
A x B = C



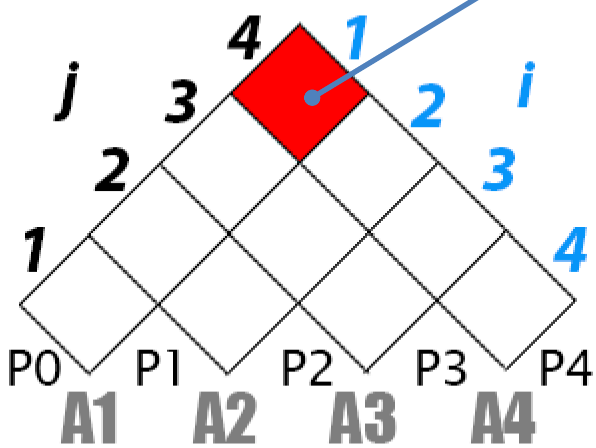
Örnek

A1	**A2**	**A3**	**A4**
5x4	4x6	6x2	2x7

Minimum elde edebileceğimiz çarpma sayısı nedir?

P0	**P1**	**P2**	**P3**	**P4**
5	4	6	2	7

$M[1][4] = ?$



Burayı bulduğumuzda min sayı bulunmuş olacak

Çözüm

Formül : $M[i][j] = \min(M[i][k] + M[k+1][j] + p[i-1] \cdot p[k] \cdot p[j])$

$$M[1][1] = 0$$
$$M[2][2] = 0$$
$$M[3][3] = 0$$
$$M[4][4] = 0$$

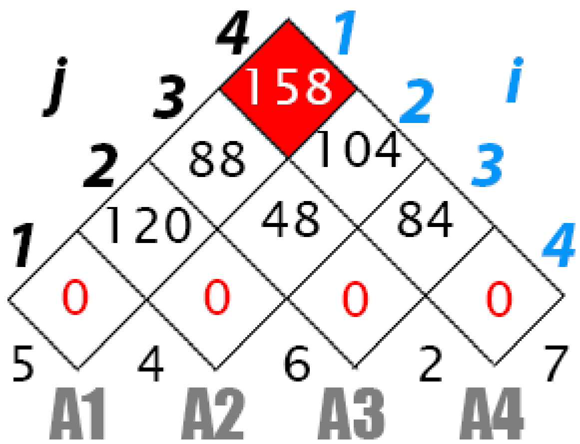
$$M[1][2] = \min(M[1][1] + M[2][2] + p[0] \cdot p[1] \cdot p[2]) = 120$$

$$M[2][3] = \min(M[2][k] + M[k+1][3] + p[1] \cdot p[2] \cdot p[3]) = 48$$

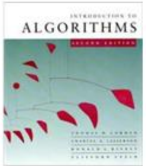
$$M[1][3] = \min \begin{cases} k=1 & \min(M[1][1] + M[2][3] + p[0] \cdot p[1] \cdot p[3]) = 0 + 48 + 5 \cdot 4 \cdot 2 = 88 \quad \lhd \min \\ k=2 & \min(M[1][2] + M[3][3] + p[0] \cdot p[2] \cdot p[3]) = 120 + 0 + 5 \cdot 6 \cdot 2 = 180 \end{cases}$$

$$M[2][4] = \min \begin{cases} k=2 & \min(M[2][2] + M[3][4] + p[1] \cdot p[2] \cdot p[4]) = 0 + 84 + 4 \cdot 6 \cdot 7 = 252 \\ k=3 & \min(M[2][3] + M[4][4] + p[1] \cdot p[3] \cdot p[4]) = 48 + 0 + 4 \cdot 2 \cdot 7 = 104 \quad \lhd \min \end{cases}$$

$$M[1][4] = \min \begin{cases} k=1 & \min(M[1][1] + M[2][4] + p[0] \cdot p[1] \cdot p[4]) = 0 + 104 + 5 \cdot 4 \cdot 7 = 244 \\ k=2 & \min(M[1][2] + M[3][4] + p[0] \cdot p[2] \cdot p[4]) = 120 + 84 + 5 \cdot 6 \cdot 7 = 414 \\ k=3 & \min(M[1][3] + M[4][4] + p[0] \cdot p[3] \cdot p[4]) = 88 + 0 + 5 \cdot 2 \cdot 7 = 158 \quad \lhd \min \end{cases}$$



Bulduğumuz değerler piramitte yerine yazılarak aşağıdan yukarıya doğru çıkılır
Cevap:158

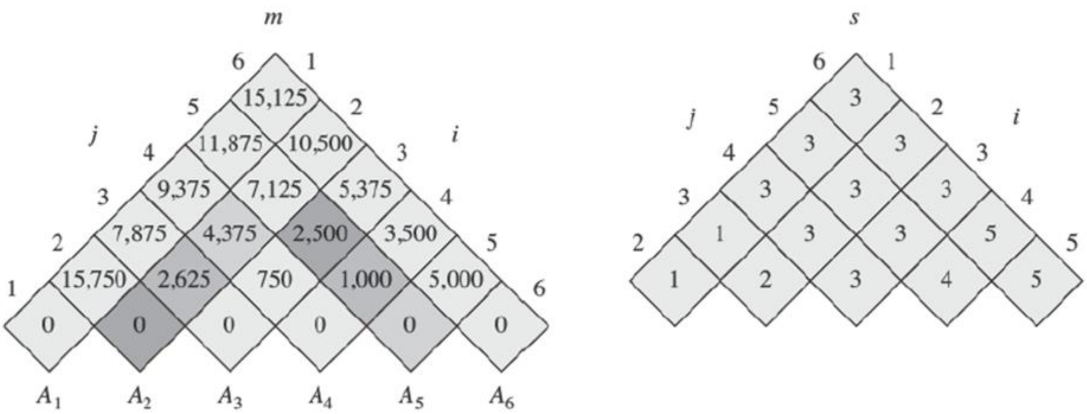


Zincir Matris Çarpımı
(Dinamik Programlama ile Çözümü)

Örnek:

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13{,}000\,, \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125\,, \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11{,}375 \end{cases}$$
$$= 7125\,.$$



Örnek Kaynak [iKaynak için tıklayınız!]

SIRT ÇANTASI PROBLEMİ (Dinamik programlama ile çözümü)

Eşyaların değerleri ve ağırlıkları var ama en değerli eşyaları çantaya koymaya çalışıyoruz

Problem
n adet eleman ***ağırlıkları*** ve ***değerleri***
Kapasite : w

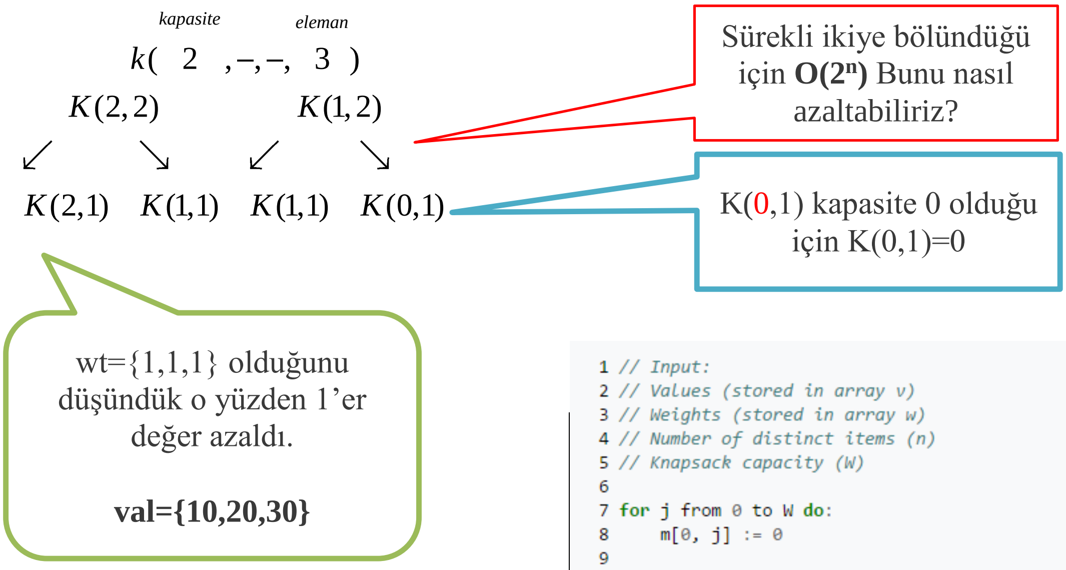
En basit çözüm bütün alt kümeleri bul
Değeri en yüksek olacak şekilde çantaya doldur. // Özyineleme ile yazılabilir

Knapsak (Sırtçantası) // Özyineleme

int Knapsack $(int\ w,\ int\ wt[\ ],\ int\ val,\ int\ n)$
// Temel durum
if $(\underbrace{n==0}_{\text{ürün yok}},\ \underbrace{w==0}_{\text{çantada yer yok}})$ return 0;
if $(wt[n-1] > w)$
    return Knapsak $(w, wt, val, n-1)$
else

$$\text{return } \textbf{max}\,(val[n-1] + \underbrace{Knapsack(\overbrace{w-wt[n-1]}^{\text{çanta'nın kapasitesi düşecek}}, wt, val, n-1)}_{1.değer}, \underbrace{Knapsack(w, wt, val, n-1)}_{2.değer})$$

Analiz




```
// Input:
// Values (stored in array v)
// Weights (stored in array w)
// Number of distinct items (n)
// Knapsack capacity (W)

for j from 0 to W do:
    m[0, j] := 0

for i from 1 to n do:
    for j from 0 to W do:
        if w[i] > j then:
            m[i, j] := m[i-1, j]
        else:
            m[i, j] := max(m[i-1, j], m[i-1, j-w[i]] + v[i])
```

Sırt Çantası // Dinamik Yaklaşımla, Dizi üzerinde tutarsak

```
int Knapsack (int w, int wt[ ], int val, int n)
{
int i,W;
int K[n+1][W+1]
for (i=0, i<n, i++)
        for (w=0, w<W, w++)
        {
                if (i==0 || w==0)
                 K[i][w]=0;
        else if (wt[i-1] ≤ w)
                 K[i][w]=max(val[i-1], K[i-1][w-wt[i-1]], K[i-1][w])
        else
                 K[i][w]=K[i-1][w-wt[i-1]], K[i-1][w])
        }
}
return K[n][w]
```

Bottomup çözüm
Zincir matrisi çözümüne benziyor

T(n)=O(n.W)
Karesel ifade

VİZE'de buraya kadar sorumluyuz // çıkabilecek soru tarzları

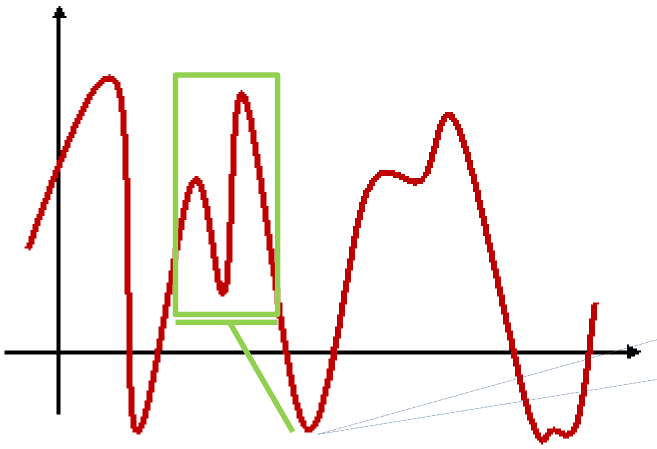
- ☐ Belki bir tanım sorulabilir // Düşük bir ihtimal
- ☐ Kod yada algoritma verilip analizini yapın T(n)=? // Quizdeki gibi
- ☐ Asimptotik yaklaşımla ne olur? O(n)=?
- ☐ Derste adı geçen ve üzerinde sıkça durulmuş olan algoritmalar bkz (Örn. 2li arama , mergesort)
- ☐ Zincir matrisi hesaplama (Boyutları şu olan matrisler ile kaç çarpma yapılır?)

AÇ GÖZLÜ (GREEDY) YAKLAŞIM

Mevcut çözümler içerisinde o anki min ve max'ları bulmak için yöntemler sunar.

Burada neyi garanti edebiliriz?

En temel düzeyde lokal optimizasyon (min yada max) garanti edilir.



Local olarak baktığımızda bir min. Noktası buluruz fakat genele baktığımızda farklı min. Noktaları olduğunu görüyoruz

Aç gözlü yaklaşım hangi algoritmalarda kullanılıyor

- Kruskol algoritması
- Shortest pat
- Huffman Algoritması
- Sırt çantası problemi

⋮

Dinamik yaklaşımla karşılaştırdığımızda

Greedy daha hızlı bir performans eğilimi gösteriyor.

Greedy Algortiması 2 ön koşula ihtiyaç duyar

1. Aç gözlü seçenek belirlenecek
2. Optimal alt yapı özelliği

SIRT ÇANTASI PROBLEMİ // Greedy Yaklaşımla Çözümü

Çantanın Kapasitesi....C
Elemanlar.................... $x_1, x_2 \cdots x_n$
Ağırlıklar.................... $s_1, s_2 \cdots s_n$
Değerler...................... $V_1, V_2 \cdots V_n$

➔ Çarp, Topla

Burada hangi elemanlar alınacak // Maximize edeceğiz

$$\underbrace{\sum_{i=1}^{n} x_i V_i}_{\substack{Hangi\\ x'ler\\ alınacak}} \quad \sum_{i=1}^{n} x_i S_i \le C$$

$$y_i = \frac{V_i}{x_i}$$

Kesirli ifade edilebilirse
Greedy yaklaşımı
kullanılabilir

Örnek
Kapasite 5kg

Elemanlar	1	2	3	4
Değeri	12	10	20	15
Ağırlık kg	2	1	3	2

Çözüm

Elemanlar x	1	2	3	4
Değeri V	12	10	20	15
Ağırlık w	2	1	3	2
$\frac{V}{w}$	6	10	6,66	7,5
Azalan Sırada Sırala	X_2	X_4	X_3	X_1
Değeri	10	15	20	12
$\frac{V}{w}$ sıralı	10	7,5	6,66	6
Çantaya Ekle	1	2	2	
Çantaya ekle	1x10=10	2x7,5=15	2x6,66=13,2	
TOPLAM	10+15+13,2=**38,2**			

$\frac{V}{w}$ değerine göre sıralandı

Algoritması // Kesirli Knapsack problemi

Knapsack (V, w, C)
$\quad yük = 0$
$\quad i = 1$
$\quad$ while $(yük < C \quad \&\& \quad i <= n)$
$\quad\quad$ if $w_i <= C - yük$ then
$\quad\quad\quad yük += eleman\imath n\ hepsi$
$\quad\quad$ else
$\quad\quad\quad yük += (C - yük) / w_i$ kadar // Ağırlığı geçmeyecek kadar
$\quad\quad$ end if
$\quad$ end while
return $yük$ **;**

EN KISA YOL PROBLEMİ (SHORTEST PATH)

- ✓ İki düğüm arasındaki en kısa yolu buluyor
- ✓ Ağırlıklı bir graf olması lazım

Düğümler........V
Kenarlar.........E

$G = (V, E)$

$V = \{1, 2, \cdots, n\}$
$x = \{1\} \qquad y = \{2, 3, \cdots, n\}$

Algortiması // Shortest Path

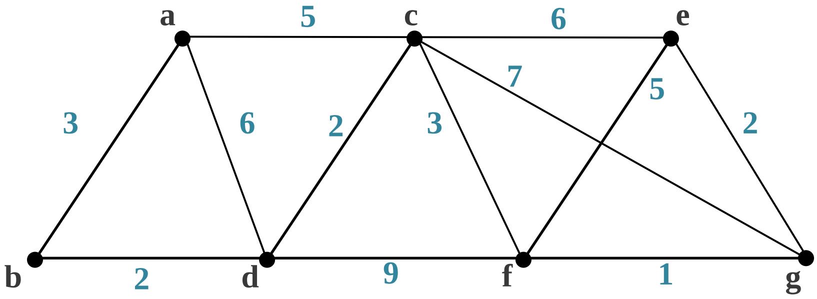
$x = \{1\} \qquad y = V - \{1\}$
$\forall y \in Y \quad için \qquad \underbrace{\lambda(V)}_{1\ ile\ aras\imath ndaki\ uzakl\imath k} \qquad hesaplan\imath r$
while $Y != \{\}$
$\quad y \in Y \quad \underbrace{\lambda(y) = \min}_{1'den\ y'ye\ \min}$

heap olduğu için
logn
tepedeki değeri (root) al

$\quad y'yi \quad Y'den \quad x'e \quad ta\c{s}\imath$

$\quad Y'ye$ komşu olan bütün düğümlerin etiketlerini güncelle
end while

Örnek



x \ y	a	b	c	d	e	f	g
a	0a	3a	5a	6a	∞ a	∞ a	∞ a
b	0a	3a	5a	5b	∞ a	∞ a	∞ a
c	0a	3a	5a	5b	11c	8c	12c
d	0a	3a	5a	5b	11c	8c	12c
e	0a	3a	5a	5b	11c	8c	12c
f	0a	3a	5a	5b	11c	8c	9f
			↑			↑	↑

En kısa yol
a-c-f-g
9

Sütundaki min değere bakıyoruz

Algoritma // Shortes Path

Ekleme (logn) // Heap kullanılıyorsa

Ekleme *(logn)* + minSilme *O(logn)* + Anahtarların 1 azalması *O(logn)*

O(m.logn) // Greedy yaklaşımı kullanılırsa

Greedy Yaklaşımı
O(mlogn)

2^n Maliyet // BruteForce, *Hepsini kontrol ederse her yolu dene*

ÖZETLE

- ✓ Dinamik programlama optimizasyonu (iyileştirmeyi) garanti eder
- ✓ Greedy yaklaşımı garanti etmez (Sezgisel bir yaklaşım olduğu için)
- ✓ Greedy Dinamik programlamanın özel bir çözümüdür.