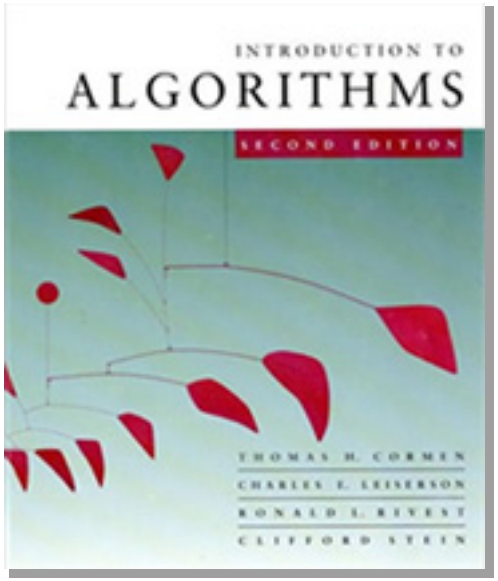
Algoritmalara Giriş

6.046J/18.401J

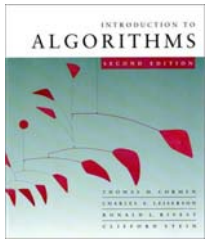


Ders 15

Dinamik Programlama

- En uzun ortak altdizi
- En uygun altyapı
- Altproblemlerin çakışması

Prof. Charles E. Leiserson

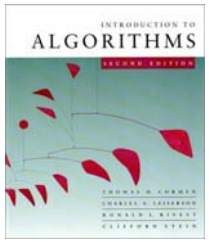


Dinamik Programlama

Böl-ve-fethet gibi bir tasarım tekniği.

Örnek: *En uzun ortak altdizi (LCS)*

- İki tane, $x[1 \ldots m]$ve $y[1 \ldots n]$ dizisi verilmiş, ikisinde de ortak olan en uzun altdiziyi bulun.



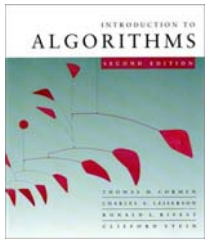
Dinamik Programlama

Böl-ve-fethet gibi bir tasarım tekniği.

Örnek: *En uzun ortak altdizi (LCS)*

- İki tane, $x[1 \,..\, m]$ve $y[1 \,..\, n]$ dizisi verilmiş, ikisinde de ortak olan en uzun altdiziyi bulun.

En uzun altdizi tek değildir.



Dinamik Programlama

Böl-ve-fethet gibi bir tasarım tekniği.

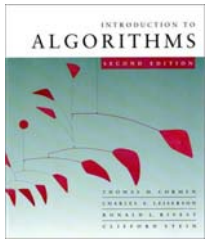
Örnek: *En uzun ortak altdizi (LCS)*

- İki tane, $x[1 \ldots m]$ve $y[1 \ldots n]$ dizisi verilmiş, ikisinde de ortak olan en uzun altdiziyi bulun.

En uzun altdizi tek değildir.

x: A B C B D A B

y: B D C A B A



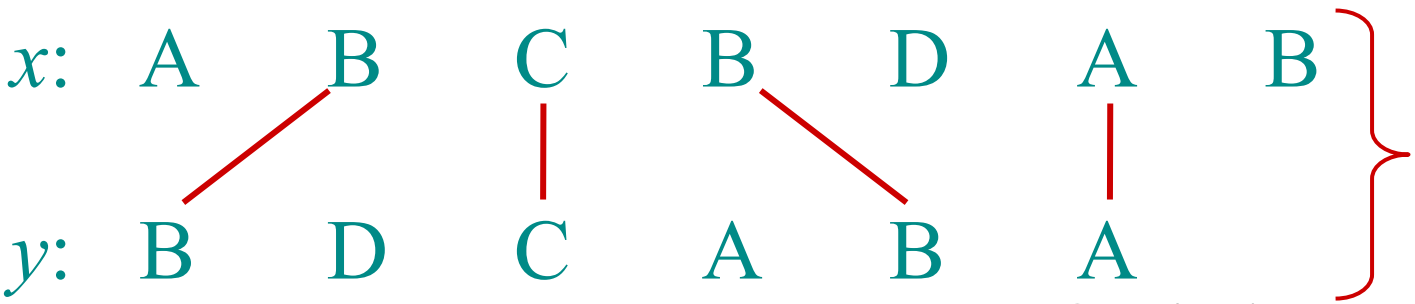
Dinamik Programlama

Böl-ve-fethet gibi bir tasarım tekniği.

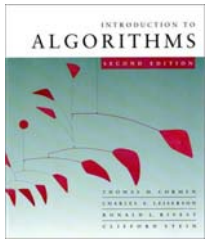
Örnek: *En uzun ortak altdizi (LCS)*

- İki tane, $x[1 \ldots m]$ve $y[1 \ldots n]$ dizisi verilmiş, ikisinde de ortak olan en uzun altdiziyi bulun.

En uzun altdizi tek değildir.

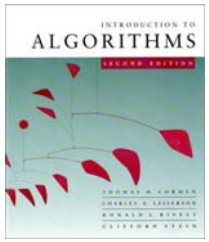


fonksiyonel gösterim, ama fonksiyon değil.



Kaba-kuvvet algoritması

$x[1 \,..\, m]$' nin her altdizisini
$y[1 \,..\, n]$' nin de altdizisi mi diye kontrol edin.



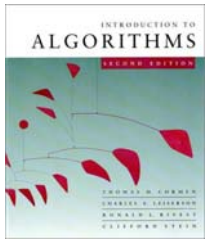
Kaba-kuvvet algoritması

$x[1\ .\ .\ m]$' nin her altdizisini
$y[1\ .\ .\ n]$' nin de altdizisi mi diye kontrol edin.

Analiz

- Kontrol etme = her altdizi için $O(n)$ zamanı.
- x' in 2^m sayıda altdizisi var.(m uzunluğunda her bir bit-vektörü, x'in farklı bir altdizisini belirler.)

  En kötü durum koşma süresi = $O(n2^m)$
  = üstel zaman.

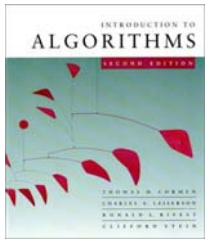


Daha iyi bir algoritmaya doğru

Basitleştirme:

1. En uzun ortak bir altdizinin uzunluğuna bakalım.

2. Algoritmayı LCS' yi kendisi bulacak şekilde genişletin.



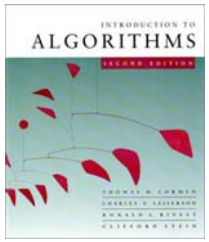
Daha iyi bir algoritmaya doğru

Basitleştirme:

1. En uzun ortak bir altdizinin uzunluğuna bakalım.

2. Algoritmayı LCS' yi kendisi bulacak şekilde genişletin.

Simgelem: s dizisinin uzunluğunu $|s|$ ile belirtin.



Daha iyi bir algoritmaya doğru

Basitleştirme:

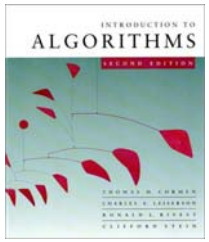
1. En uzun ortak bir altdizinin uzunluğuna bakalım.

2. Algoritmayı LCS' yi kendisi bulacak şekilde genişletin.

Simgelem: s dizisinin uzunluğunu $|s|$ ile belirtin.

Strateji: x ve y'nin öneklerini düşünün.

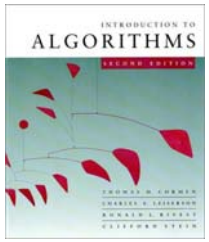
- Define(tanımlama) $c[i, j] = |\mathrm{LCS}(x[1\,..\,i], y[1\,..\,j])|$.
- Then(sonra), $c[m, n] = |\mathrm{LCS}(x, y)|$.



Özyinelemeli formülleme

Teorem.

$$c[i,j] = \begin{cases} c[i-1,j-1]+1 & \text{(eğer) if } x[i]=y[j] \text{ ise,} \\ \max\{c[i-1,j], c[i,j-1]\} & \text{aksi takdirde.} \end{cases}$$

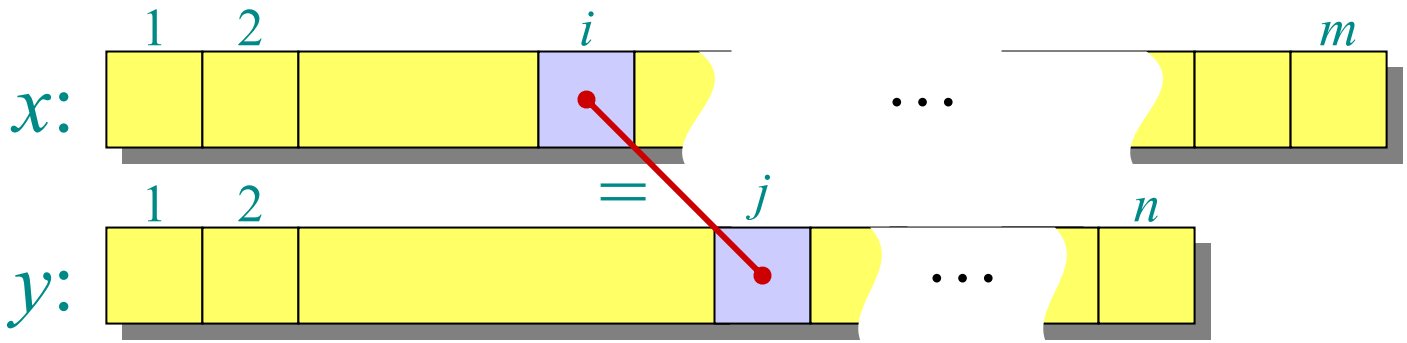


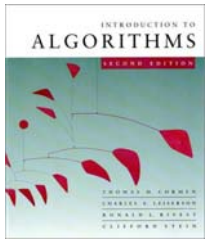
Özyinelemeli formülleme

Teorem.

$$c[i,j] = \begin{cases} c[i-1,j-1]+1 & \text{(eğer) if } x[i]=y[j] \text{ ise,} \\ \max\{c[i-1,j],\, c[i,j-1]\} & \text{aksi takdirde.} \end{cases}$$

Kanıt. Durum $x[i] = y[j]$:



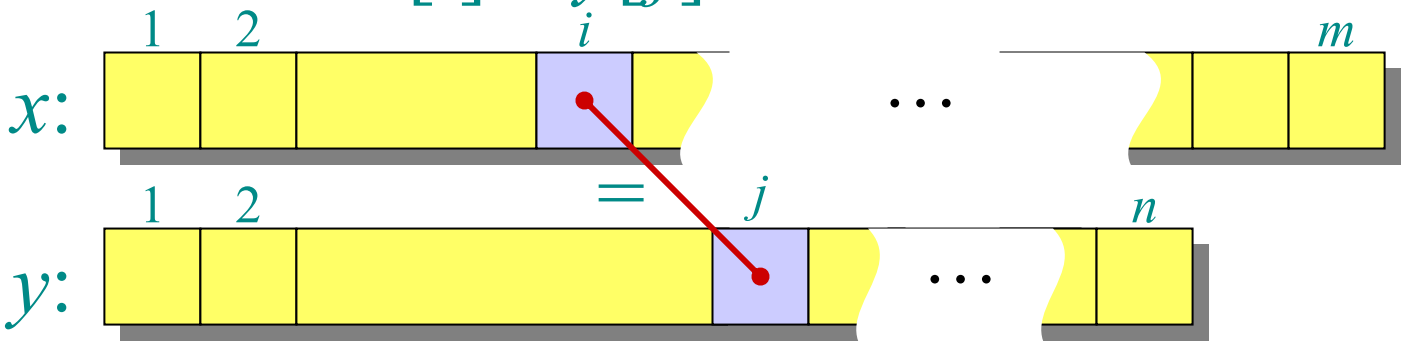


Özyinelemeli formülleme

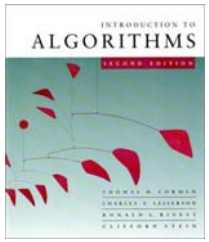
Teorem.

$$c[i,j] = \begin{cases} c[i-1,j-1]+1 & \text{(eğer) if } x[i]=y[j] \text{ ise,} \\ \max\{c[i-1,j],\, c[i,j-1]\} & \text{aksi takdirde.} \end{cases}$$

Kanıt. Durum $x[i] = y[j]$:



$c[i, j] = k$ iken $z[1\,..\,k] = \text{LCS}(x[1\,..\,i], y[1\,..\,j])$ olsun.
Sonra, $z[k] = x[i]$, ya da başka z genişletilebilir.
Böylece, $z[1\,..\,k-1]$, $x[1\,..\,i-1]$ ve $y[1\,..\,j-1]$'nin CS'sidir.

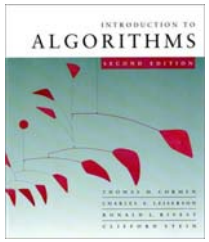


Kanıt (devamı)

İddia: $z[1\,..\,k\text{–}1] = \text{LCS}(x[1\,..\,i\text{–}1], y[1\,..\,j\text{–}1])$.

Diyelim ki w, $x[1\,..\,i\text{–}1]$ ve $y[1\,..\,j\text{–}1]$' nin daha uzun bir CS'si,yani $|w| > k\text{–}1$. Öyleyse, ***kes ve yapıştır***: $w \,\|\, z[k]$ (w, $z[k]$ile bitiştirilmiş.)

$x[1\,..\,i]$ ve $y[1\,..\,j]$' nin ortak altdizisidir ve $|w\|\, z[k]| > k$' dır. Çelişki, iddiayı kanıtlar.



Kanıt (devamı)

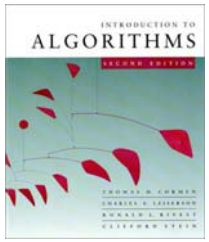
İddia: $z[1\,..\,k–1] = \text{LCS}(x[1\,..\,i–1], y[1\,..\,j–1])$.

Diyelim ki w, $x[1\,..\,i–1]$ ve $y[1\,..\,j–1]$' nin daha uzun bir CS'si,yani $|w| > k–1$. Sonra, ***kes ve yapıştır***: $w \,\|\, z[k]$ (w, $z[k]$ile bitiştirilmiş.)

$x[1\,..\,i]$ ve $y[1\,..\,j]$' nin ortak altdizisidir ve $|w\| z[k]| > k$' dır.Çelişki, iddiayı kanıtlar.

Böylece, $c[i–1, j–1] = k–1$'dir ki bu $c[i, j] = c[i–1, j–1] + 1$ anlamına gelir.

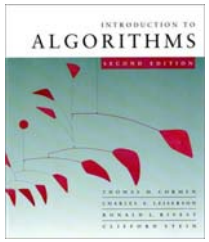
Diğer durumlar benzerdir. □



Dinamik-programlama kalite işareti #1

Optimal altyapı

Bir probleme optimal çözüm (örnek) alt-problemlere optimal çözümler içerir.

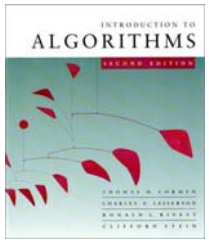


Dinamik-programlama kalite işareti #1

Optimal altyapı

Bir probleme optimal çözüm (örnek) alt-problemlere optimal çözümler içerir.

Eğer $z = \text{LCS}(x, y)$ ise, z nin her öneki ‚x' in ve y' nin öneklerinin LCS' sidir.



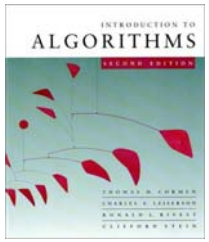
LCS için özyinelemeli algoritma

$$\text{LCS}(x, y, i, j)$$

(eğer) **if** $x[i] = y[j]$

(sonra) **then** $c[i, j] \leftarrow \text{LCS}(x, y, i{-}1, j{-}1) + 1$

(başka) **else** $c[i, j] \leftarrow \max\{\text{LCS}(x, y, i{-}1, j),$
$\text{LCS}(x, y, i, j{-}1)\}$



LCS için özyinelemeli algoritma

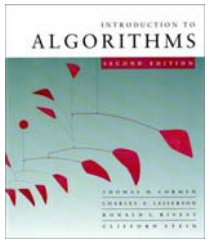
$\text{LCS}(x, y, i, j)$

(eğer) **if** $x[i] = y[j]$

(sonra) **then** $c[i, j] \leftarrow \text{LCS}(x, y, i–1, j–1) + 1$

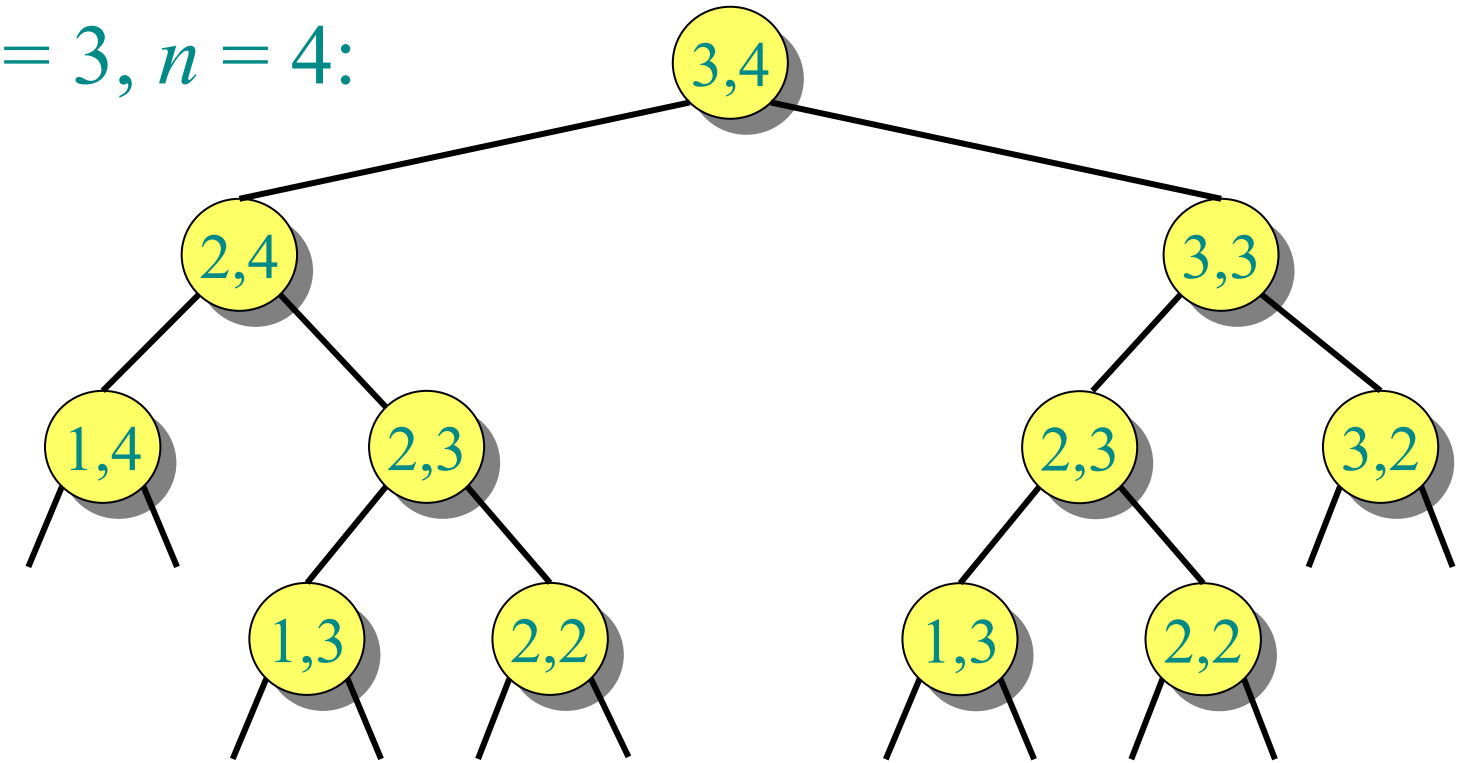
(başka) **else** $c[i, j] \leftarrow \max\{\text{LCS}(x, y, i–1, j),$
$\text{LCS}(x, y, i, j–1)\}$

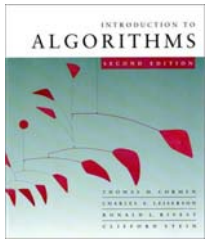
En kötü durum: $x[i] \neq y[j]$, bu durumda algoritma, tek parametrenin azaltıldığı iki alt-problemi değerlendirir.



Özyineleme ağacı

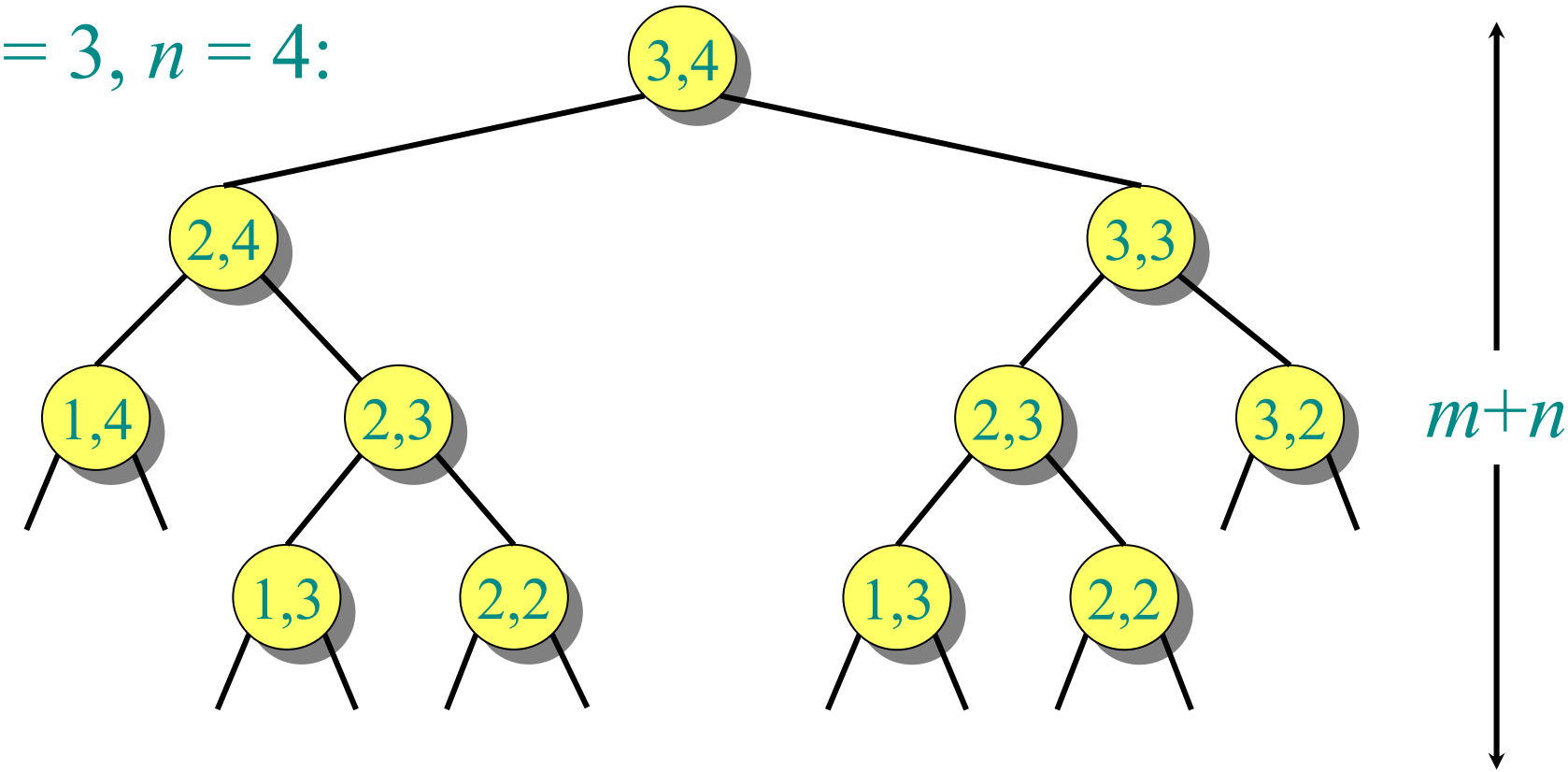
$m = 3, n = 4$:



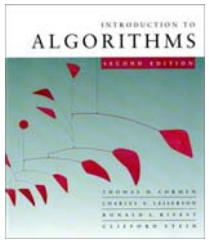


Özyineleme ağacı

$m = 3, n = 4$:

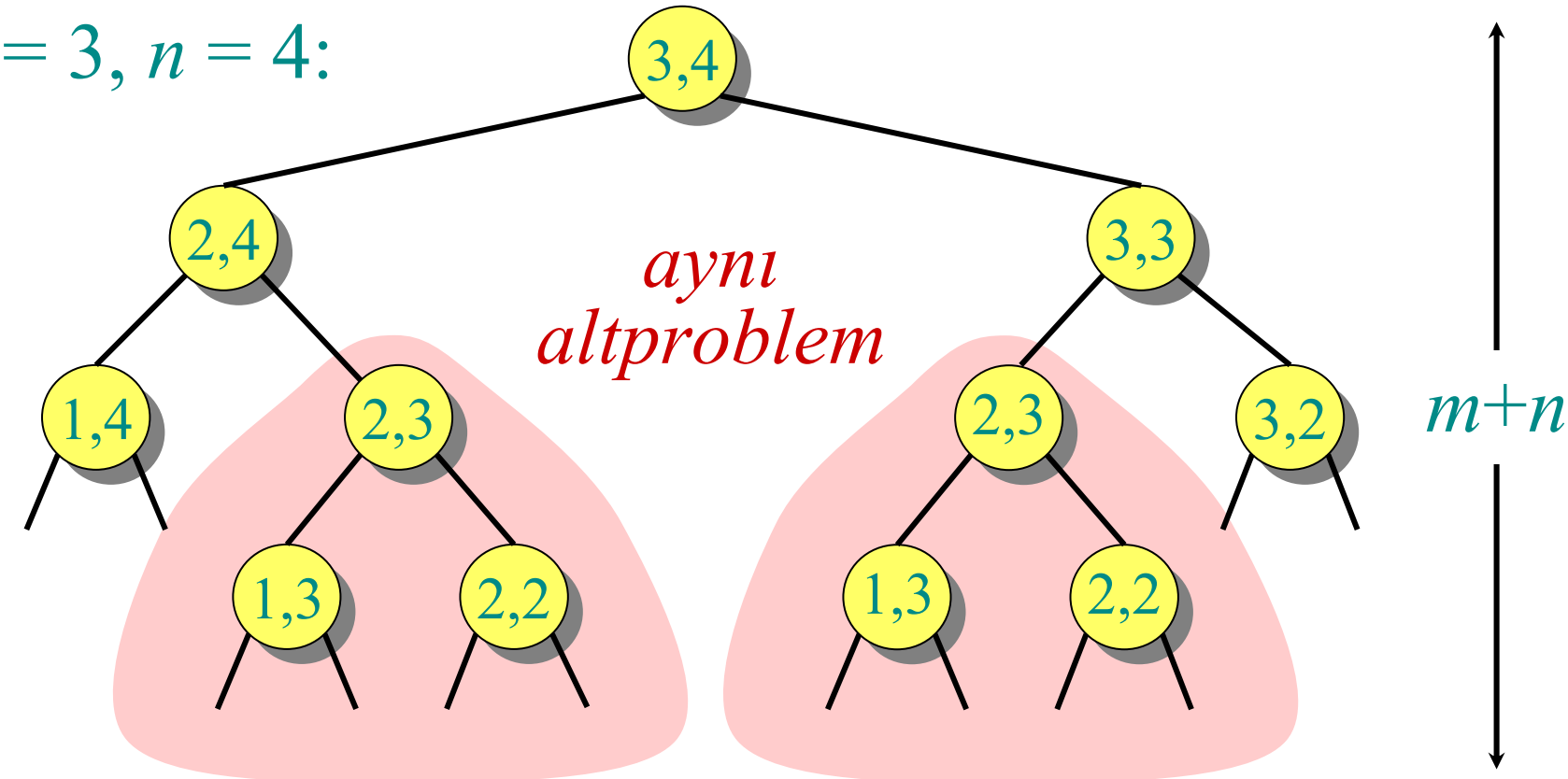


Yükseklik = $m + n$ ⇒iş potansiyel olarak üsteldir.

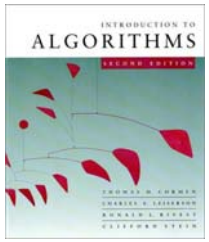


Özyineleme ağacı

$m = 3, n = 4$:



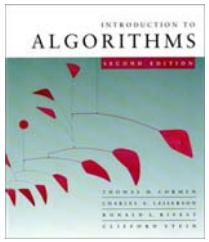
Yükseklik = $m + n$ ⇒iş potansiyel olarak üsteldir,
ama biz zaten çözülmüş olan altproblemleri çözüyoruz!



Dinamik-programlama kalite ikareti #2

Altproblemlerin çakışması

Özyinelemeli bir çözüm, az sayıda birbirinden farklı altproblemin birçok kere tekrarlanmasını içerir.

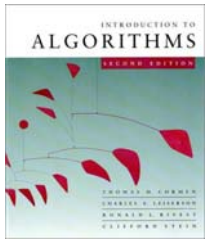


Dinamik-programlama kalite ikareti #2

Altproblemlerin çakışması

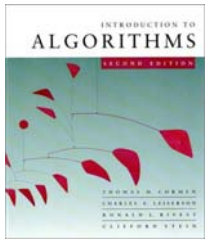
Özyinelemeli bir çözüm, az sayıda birbirinden farklı altproblemin birçok kere tekrarlanmasını içerir.

m ve n uzunluklarının 2 dizgisi için farklı LCS altproblemlerinin sayısı mn' dir.



Hatırlama algoritması

Memoization(Hatırlama): Bir altprobleme ait çözümü hesapladıktan sonra onu bir tabloda depolayın. Ardışık çağırmalar, işlemi tekrar yapmamak için tabloyu kontrol eder.

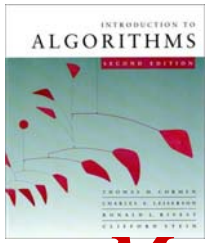


Hatırlama algoritması

Memoization(Hatırlama): Bir altprobleme ait çözümü hesapladıktan sonra onu bir tabloda depolayın. Ardışık çağırmalar, işlemi tekrar yapmamak için tabloyu kontrol eder.

LCS(x, y, i, j)
(eğer)**if** $c[i, j] = \text{NIL}$
(sonra eğer)**then if** $x[i] = y[j]$
(sonra) **then** $c[i, j] \leftarrow \text{LCS}(x, y, i–1, j–1) + 1$
(başka) **else** $c[i, j] \leftarrow \max\{\text{LCS}(x, y, i–1, j), \text{LCS}(x, y, i, j–1)\}$

} *önceki gibi*



Hatırlama algoritması

Memoization(Hatırlama): Bir altprobleme ait çözümü hesapladıktan sonra onu bir tabloda depolayın. Ardışık çağırmalar, işlemi tekrar yapmamak için tabloyu kontrol eder.

LCS(x, y, i, j)

(eğer) **if** $c[i, j] = \text{NIL}$

(sonra eğer) **then if** $x[i] = y[j]$

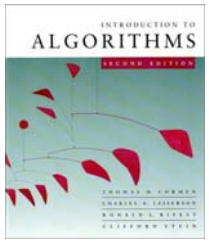
(sonra) **then** $c[i, j] \leftarrow \text{LCS}(x, y, i–1, j–1) + 1$

(başka) **else** $c[i, j] \leftarrow \max\{\text{LCS}(x, y, i–1, j), \text{LCS}(x, y, i, j–1)\}$

önceki gibi

Süre $= \Theta(mn)$ =her tablo girişi için sabit miktarda iş.

Yer(alan) $= \Theta(mn)$.

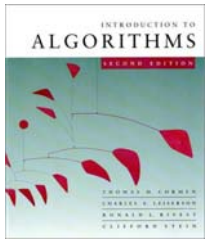


Dinamik-programlama algoritması

Fikir:

Aşağıdan yukarıya tabloyu hesaplayın.

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4



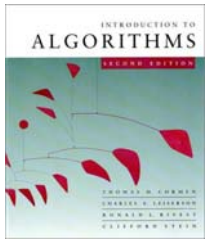
Dinamik-programlama algoritması

Fikir:

Aşağıdan yukarıya tabloyu hesaplayın.

Süre = $\Theta(mn)$.

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4



Dinamik-programlama algoritması

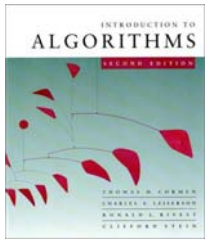
Fikir:

Aşağıdan yukarıya tabloyu hesaplayın.

Süre = $\Theta(mn)$.

LCS' yi, geriye giderek tekrar oluşturun.

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4



Dinamik-programlama algoritması

Fikir:

Aşağıdan yukarıya tabloyu hesaplayın

Süre = $\Theta(mn)$.

LCS' yi, geriye giderek tekrar oluşturun.

Alan = $\Theta(mn)$.

Alıştırma:

$O(\min\{m, n\})$.

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4