Οι διαφάνειες βασίζονται σε αυτές του ακόλουθου μαθήματος:

Introduction to Algorithms (6-046J), MIT

https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/

Δυναμικός Προγραμματισμός

Τεχνική Σχεδιασμού, όπως η διαίρει και βασίλευε.

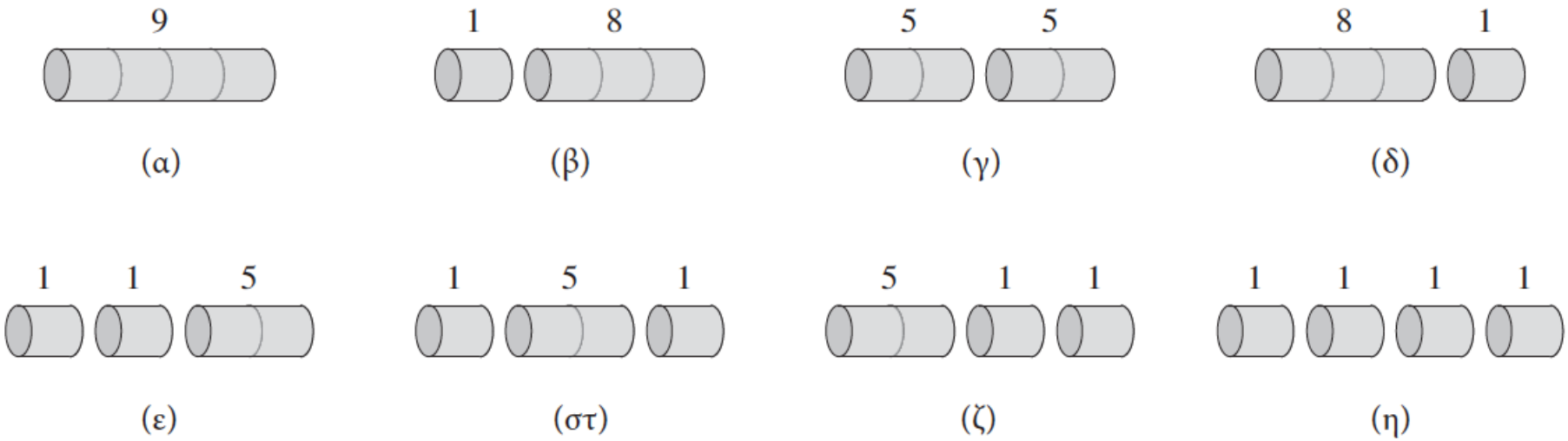
Το πρόβλημα της κοπής ράβδου

Το ***πρόβλημα της κοπής ράβδου:***
Είσοδος: ράβδος μήκους n ιντσών και ένας πίνακας τιμών p_i για $i = 1, 2, \ldots, n$

Ζητείται να προσδιορίσουμε το μέγιστο έσοδο r_n αν κόψουμε τη ράβδο και πουλήσουμε τα διάφορα κομμάτια

μήκος i	1	2	3	4	5	6	7	8	9	10
τιμή p_i	1	5	8	9	10	17	17	20	24	30

Σχήμα 15.1 Ενδεικτικός πίνακας τιμών για ράβδους. Κάθε ράβδος μήκους i ιντσών αποφέρει στη βιοτεχνία έσοδο p_i ευρώ.



Σχήμα 15.2 Οι 8 δυνατοί τρόποι κοπής μιας ράβδου μήκους 4. Πάνω από κάθε κομμάτι αναγράφεται η τιμή του, σύμφωνα με τον ενδεικτικό πίνακα τιμών του Σχήματος 15.1. Η βέλτιστη λύση είναι αυτή του μέρους (γ) –κοπή της ράβδου σε δύο κομμάτια μήκους 2– η οποία δίνει συνολικό έσοδο 10.

Το πρόβλημα της κοπής ράβδου

Γενικότερα, μπορούμε να εκφράσουμε τις τιμές r_n για $n \geq 1$ συναρτήσει βέλτιστων εσόδων από μικρότερες ράβδους:

$$r_n = \max (p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1) .$$

Ισοδύναμα:

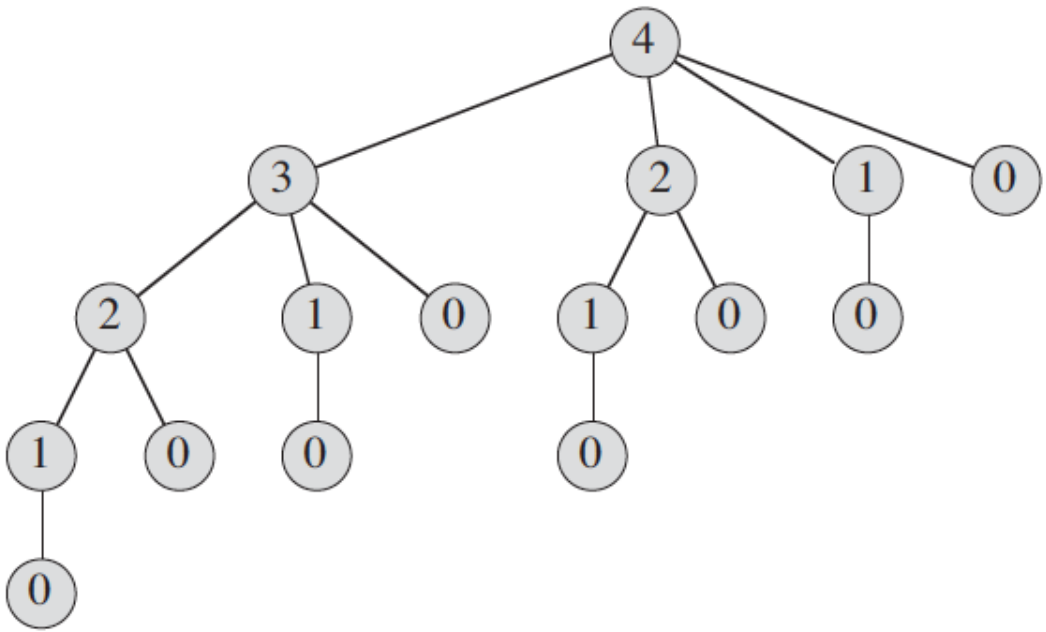
$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})$$

Αναδρομική καταβιβαστική υλοποίηση

```
CUT-ROD(p, n)
1  if n == 0
2       return 0
3  q = -∞
4  for i = 1 to n
5       q = max(q, p[i] + CUT-ROD(p, n - i))
6  return q
```

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j)$$

$T(n) = 2^n$



Δένδρο αναδρομής

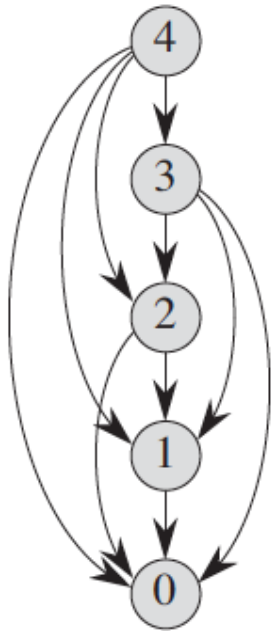
Βέλτιστη κοπή ράβδου μέσω δυναμικού προγραμματισμού

```
MEMOIZED-CUT-ROD(p, n)
1  let r[0..n] be a new array
2  for i = 0 to n
3      r[i] = -∞
4  return MEMOIZED-CUT-ROD-AUX(p, n, r)
```

```
MEMOIZED-CUT-ROD-AUX(p, n, r)
1  if r[n] ≥ 0
2      return r[n]
3  if n == 0
4      q = 0
5  else q = -∞
6      for i = 1 to n
7          q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n - i, r))
8  r[n] = q
9  return q
```

Βέλτιστη κοπή ράβδου μέσω δυναμικού προγραμματισμού

```
BOTTOM-UP-CUT-ROD(p, n)
1  let r[0..n] be a new array
2  r[0] = 0
3  for j = 1 to n
4      q = -∞
5      for i = 1 to j
6          q = max(q, p[i] + r[j - i])
7      r[j] = q
8  return r[n]
```



Το γράφημα υποπροβλημάτων για το πρόβλημα της κοπής ράβδου με $n = 4$

Ανακατασκευή λύσης

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
 1  let r[0..n] and s[0..n] be new arrays
 2  r[0] = 0
 3  for j = 1 to n
 4      q = -∞
 5      for i = 1 to j
 6          if q < p[i] + r[j - i]
 7              q = p[i] + r[j - i]
 8              s[j] = i
 9      r[j] = q
10  return r and s
```

```
PRINT-CUT-ROD-SOLUTION(p, n)
1  (r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
2  while n > 0
3      print s[n]
4      n = n - s[n]
```

Η βέλτιστη κοπή και κέρδος για το αρχικό παράδειγμα:

i	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	8	10	13	17	18	22	25	30
$s[i]$	0	1	2	3	2	2	6	1	2	3	10

Δυναμικός Προγραμματισμός

Τεχνική Σχεδιασμού, όπως η διαίρει και βασίλευε.

Παράδειγμα: Μακρύτερη Κοινή Υπακολουθία (ΜΚΥ)

- Δοθεισών δύο ακολουθιών $x[1 \,..\, m]$ και $y[1 \,..\, n]$, βρες μία μακρύτερη υπακολουθία κοινή και στις δύο ακολουθίες.

Δυναμικός Προγραμματισμός

Τεχνική Σχεδιασμού, όπως η διαίρει και βασίλευε.

Παράδειγμα: Μακρύτερη Κοινή Υπακολουθία (ΜΚΥ)

- Δοθεισών δύο ακολουθιών $x[1 \,..\, m]$ και $y[1 \,..\, n]$, βρες μία μακρύτερη υπακολουθία κοινή και στις δύο ακολουθίες. ■

"μία" *και όχι* "τη"

x: A B C B D A B

y: B D C A B A

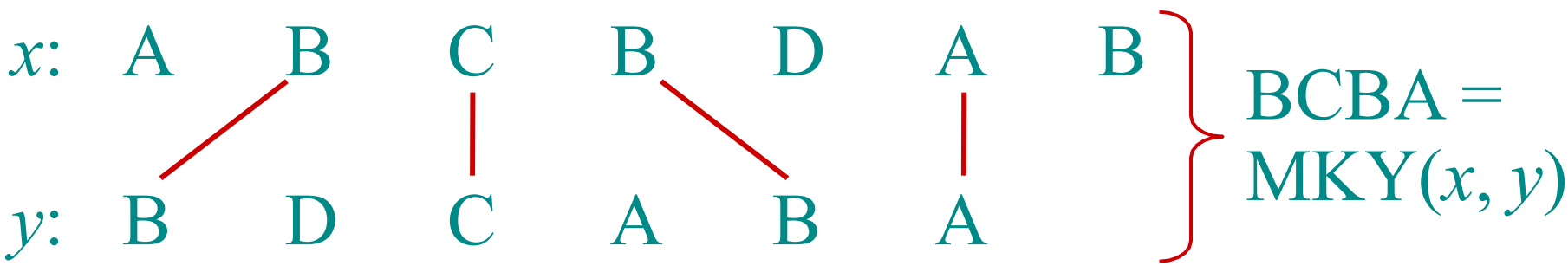
Δυναμικός Προγραμματισμός

Τεχνική Σχεδιασμού, όπως η διαίρει και βασίλευε.

Παράδειγμα: Μακρύτερη Κοινή Υπακολουθία (ΜΚΥ)

- Δοθεισών δύο ακολουθιών $x[1 \,..\, m]$ και $y[1 \,..\, n]$, βρες μία μακρύτερη υπακολουθία κοινή και στις δύο ακολουθίες. ■

"μία" *και όχι* "τη"



Ο ΜΚΥ αλγόριθμος – Προφανής αλγόριθμος

Έλεγξε κάθε υποακολουθία της $x[1 \,..\, m]$ για να δεις αν είναι επίσης υπακολουθία της $y[1 \,..\, n]$.

Ανάλυση

2^m υπακολουθίες της x (κάθε διάνυσμα διφύων μήκους m προσδιορίζει μία διαφορετική υπακολουθία της x).

Άρα εκθετικός ο χρόνος εκτέλεσης.

Προς ένα καλύτερο αλγόριθμο

Απλοποίηση:

1. Εστιάζουμε στον υπολογισμό του *μήκους* της μακρύτερης κοινής υπακολουθίας.
2. Επεκτείνουμε τον αλγόριθμο για να βρούμε την MKY. ■

Συμβολισμός: Συμβολίζουμε το μήκος της ακολουθίας s με $|s|$.

Στρατηγική: Θεωρούμε τα ***προθέματα*** των x και y.

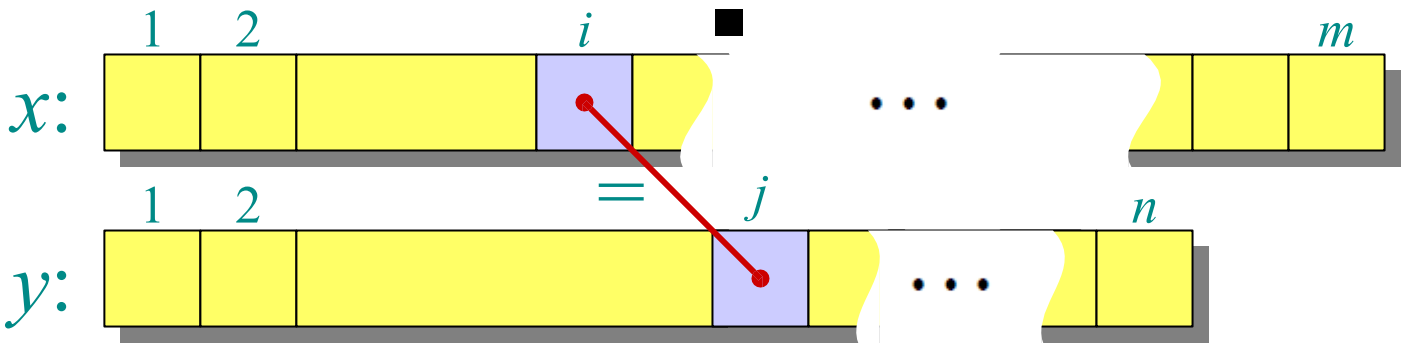
- Ορίζουμε $c[i, j] = |\text{MKY}(x[1\,..\,i], y[1\,..\,j])|$.
- Τότε, $c[m, n] = |\text{MKY}(x, y)|$.

Αναδρομική διατύπωση

Θεώρημα.

$$c[i,j] = \begin{cases} c[i-1,j-1]+1 & \text{αν } x[i]=y[j], \\ \max\{c[i-1,j],\, c[i,j-1]\} & \text{αλλιώς.} \end{cases}$$

Proof. Περίπτωση $x[i] = y[j]$:



Αν $z[1\,..\,k] = \text{MKY}(x[1\,..\,i], y[1\,..\,j])$, όπου $c[i,j] = k$. Τότε, $z[k] = x[i]$=y[j], αλλιώς το z μπορεί να επεκταθεί. Έτσι, η $z[1\,..\,k-1]$ είναι ΚΥ των $x[1\,..\,i-1]$ and $y[1\,..\,j-1]$.

Απόδειξη (συν.)

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

Υποθέτουμε ότι η w είναι μακρύτερη κοινή υπακολουθία των $x[1 \,..\, i–1]$ και $y[1 \,..\, j–1]$, δηλ., $|w| > k–1$. ■

Τότε, ***αποκοπή και επικόλληση*** : $w \,\|\, z[k]$ (w ακολουθούμενη από τη $z[k]$) είναι μία κοινή υπακολουθία των $x[1 \,..\, i]$ και $y[1 \,..\, j]$ με $|w \,\|\, z[k]| > k$. Άτοπο, αποδεικνύοντας τον ισχυρισμό.

Απόδειξη (συν.)

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

Υποθέτουμε ότι η w είναι μακρύτερη κοινή υπακολουθία των $x[1\,..\,i–1]$ και $y[1\,..\,j–1]$, δηλ., $|w| > k–1$.

Τότε, ***αποκοπή και επικόλληση*** : $w \parallel z[k]$ (w ακολουθούμενη από τη $z[k]$) είναι μία κοινή υπακολουθία των $x[1\,..\,i]$ και $y[1\,..\,j]$ με $|w \parallel z[k]| > k$. Άτοπο, αποδεικνύοντας τον ισχυρισμό.

Έτσι, $c[i–1, j–1] = k–1$ που σημαίνει ότι $c[i, j] = c[i–1, j–1] + 1$.

Οι άλλες περιπτώσεις είναι παρόμοιες.

1η Βασική Προϋπόθεση για την Εφαρμογή του Δυναμικού Προγραμματισμού

Βέλτιστη Υποδομή

Μία βέλτιστη λύση σε ένα πρόβλημα (στιγμιότυπο) περιέχει βέλτιστες λύσεις σε υποπροβλήματα.

1η Βασική Προϋπόθεση εφαρμογής Δυναμικού Προγραμματισμού

Βέλτιστη Υποδομή

Μία βέλτιστη λύση σε ένα πρόβλημα (στιγμιότυπο) περιέχει βέλτιστες λύσεις σε υποπροβλήματα.

Αν $z = \text{ΜΚΥ}(x, y)$, τότε οποιοδήποτε πρόθεμα της z είναι μία ΜΚΥ ενός προθέματος της x και ενός προθέματος της y.

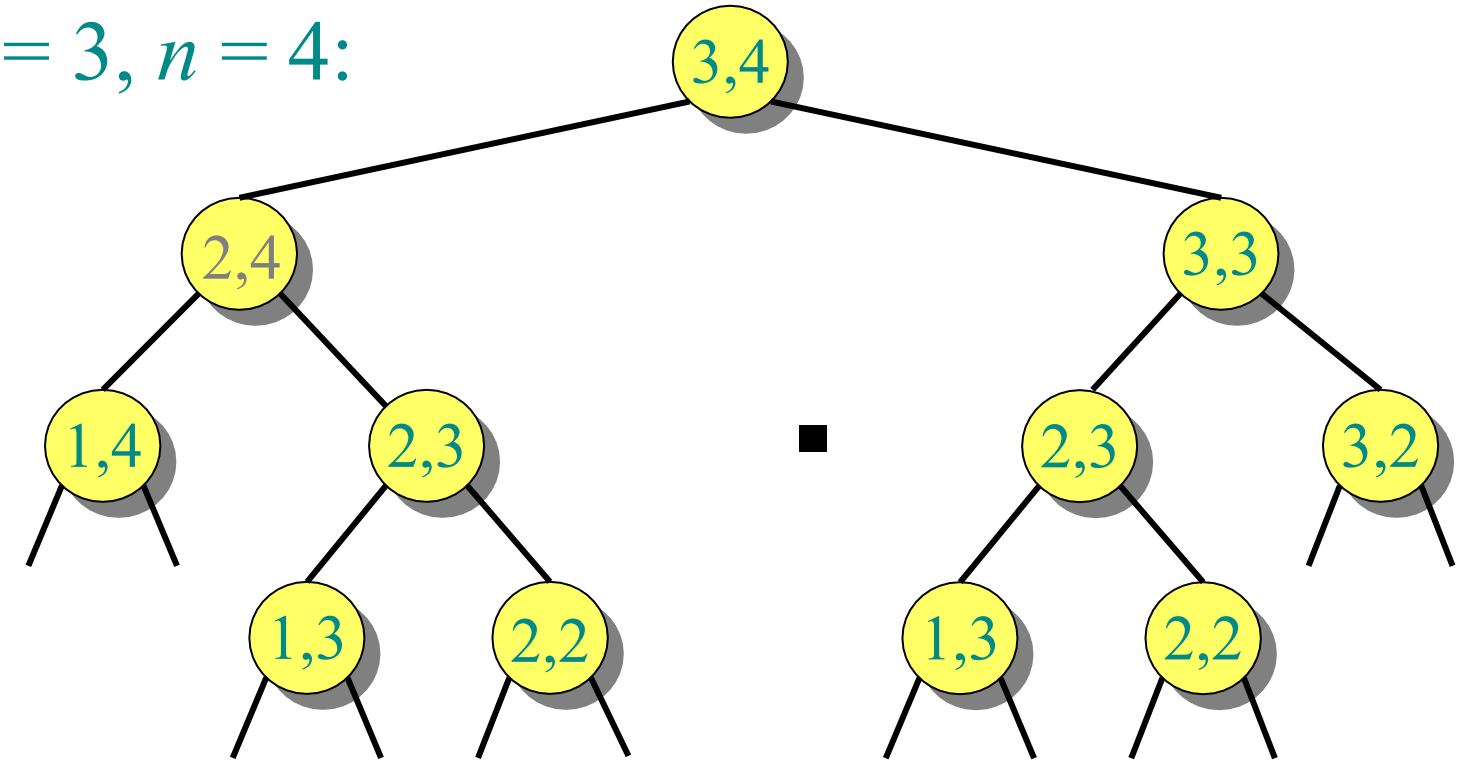
Αναδρομικός αλγόριθμος για MKY

MKY(x, y, i, j)
  if $x[i] = y[j]$
    then $c[i, j] \leftarrow$ MKY$(x, y, i–1, j–1) + 1$
    else $c[i, j] \leftarrow \max\{MKY(x, y, i–1, j)$,
      MKY$(x, y, i, j–1)\}$

Χειρότερη περίπτωση: $x[i] \neq y[j]$. Σε αυτή την περίπτωση, ο αλγόριθμος εκτελείται σε δύο υποπροβλήματα στα οποία μόνο μία παράμετρος ελαττώνεται.

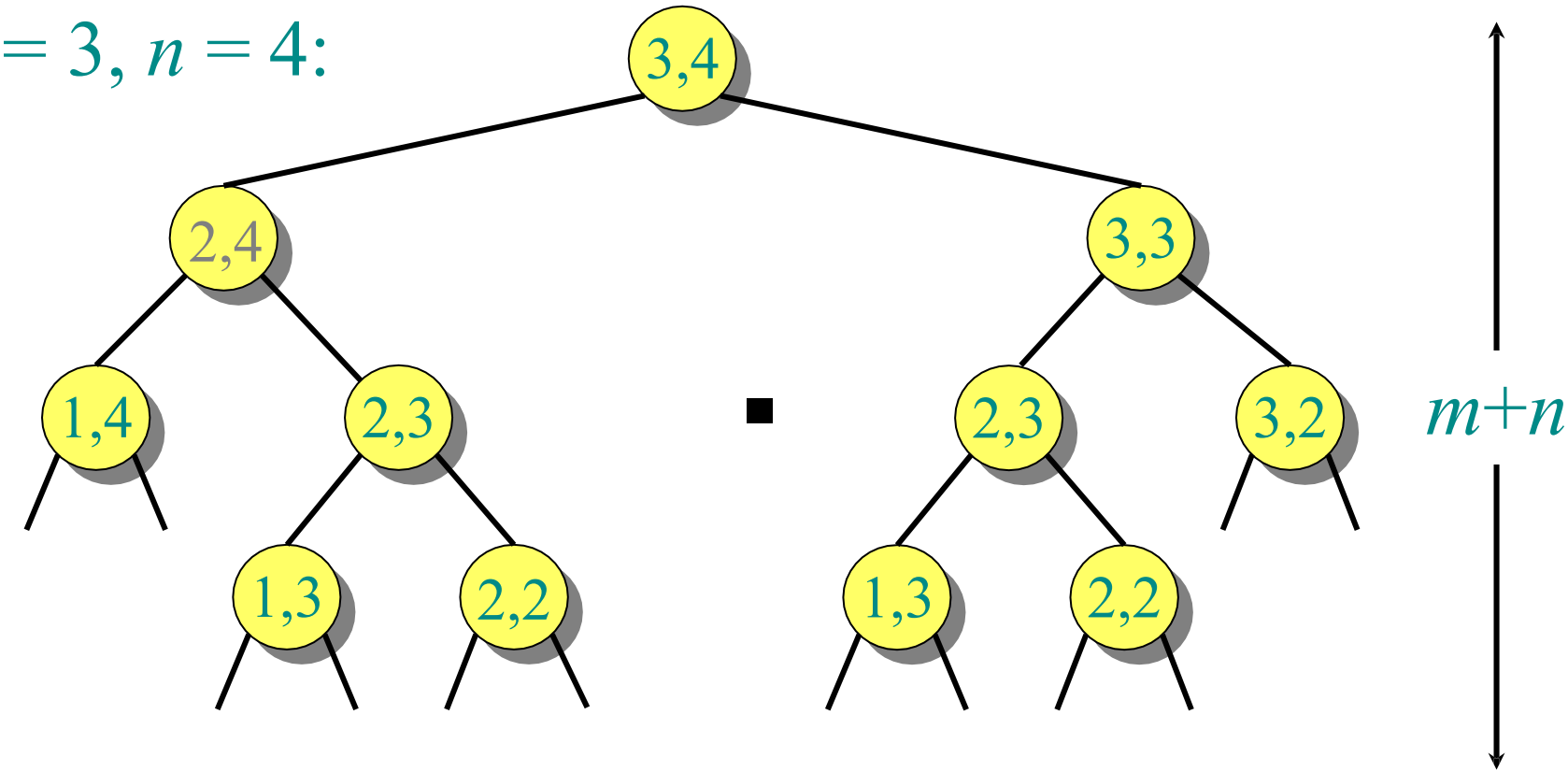
Δένδρο Αναδρομής

$m = 3, n = 4$:



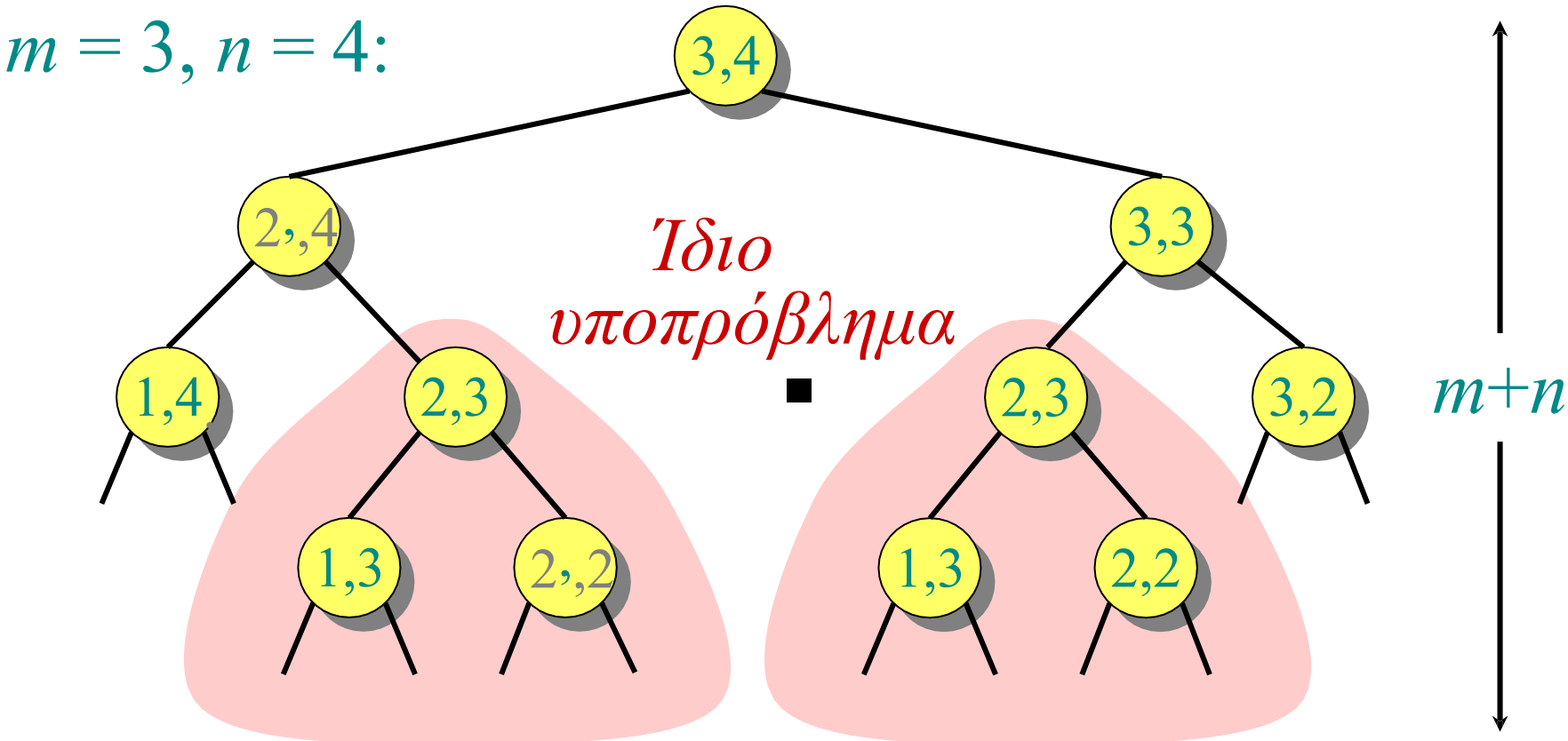
Δένδρο Αναδρομής

$m = 3, n = 4$:



Ύψος $= m + n \Rightarrow$ Εκθετικός χρόνος δυνητικά

Δένδρο Αναδρομής



Ύψος = $m + n$ ⇒ ο χρόνος δυνητικά εκθετικός. Αλλά, επιλύουμε συχνά υποπροβλήματα που ήδη έχουν λυθεί.

2η Βασική Προϋπόθεση για την Εφαρμογή του Δυναμικού Προγραμματισμού

Επικαλυπτόμενα Προβλήματα

Μία αναδρομική λύση περιέχει ένα "μικρό" αριθμό διαφορετικών υποπροβλημάτων που επαναλαμβάνονται πολλές φορές.

Το πλήθος των διαφορετικών υποπροβλημάτων στο πρόβλημα ΜΚΥ για δύο συμβολοσειρές μήκους m και n είναι μόνο mn.

Αλγόριθμος με Υπομνηματισμό

Υπομνηματισμός: Μετά από τον υπολογισμό μίας λύσης σε ένα υποπρόβλημα, αποθήκευσε την στο πίνακα. Στις επόμενες κλήσεις, έλεγξε το πίνακα για να αποφύγεις να επιλύσεις το ίδιο υποπρόβλημα.

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

Ίδιο όπως πριν

Χρόνος = $\Theta(mn)$ = σταθερό έργο ανά στοιχείο πίνακα.

Χώρος = $\Theta(mn)$.

Αλγόριθμος με Υπομνηματισμό

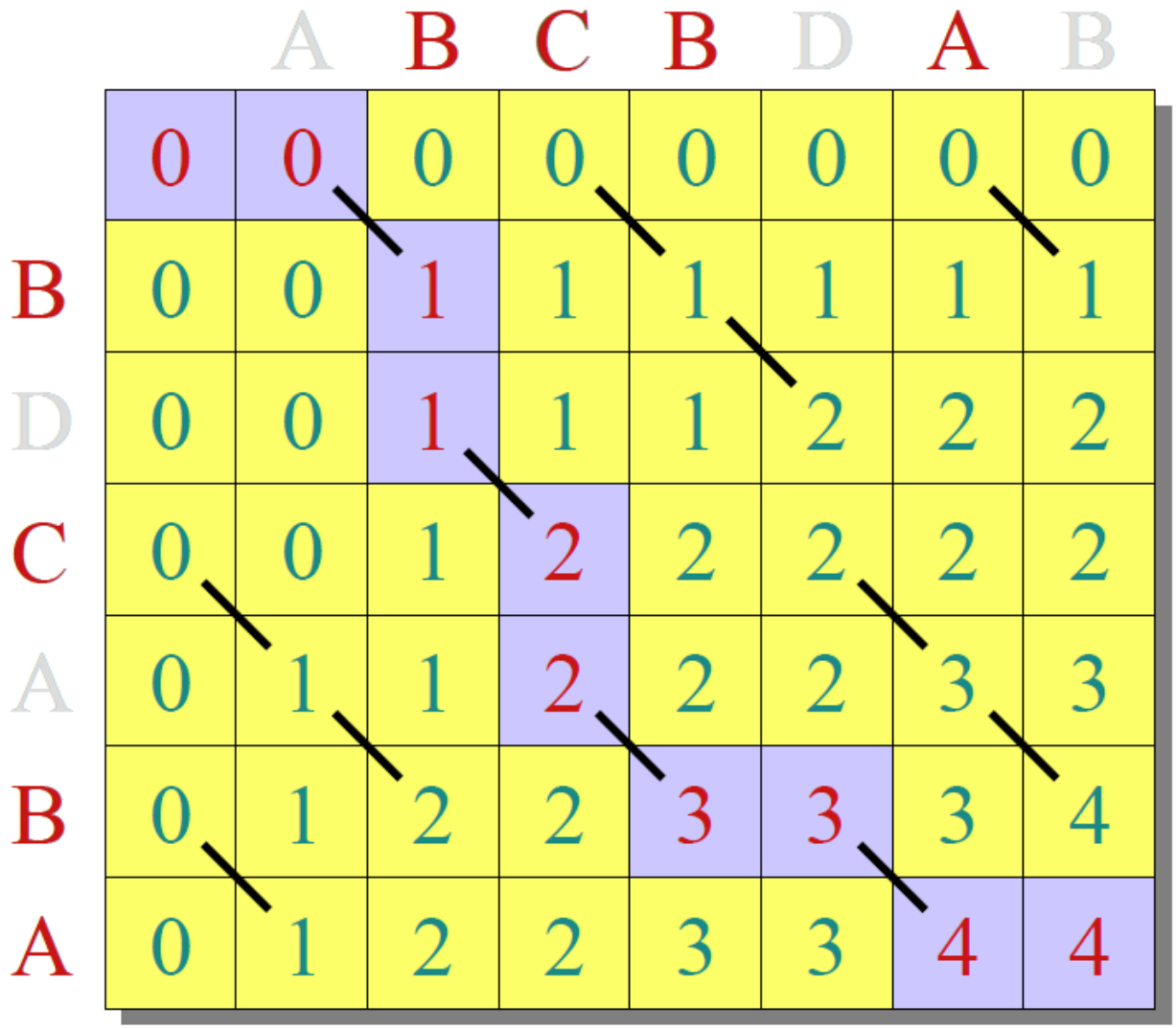
Ιδέα:

Υπολόγισε τον πίνακα από πάνω προς τα κάτω.

Χρόνος = $\Theta(mn)$.

Ανακατασκεύασε την ΜΚΥ με ανίχνευση προς τα πίσω.

Χώρος= $\Theta(mn)$.



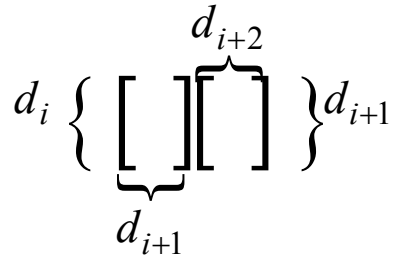
Βέλτιστος Πολλαπλασιασμός Πινάκων

- Δίνεται μία ακολουθία πινάκων:

  $A_0, A_1, \ldots, A_{n-1},$

  να βρεθεί ο πιο γρήγορος τρόπος υπολογισμού του γινομένου:

  $A_0 \cdot A_1 \cdot \cdot \cdot A_{n-1}.$
- Εάν ο A_i είναι $d_i \times d_{i+1}$, οπότε το κόστος του γινομένου $A_i \cdot A_{i+1}$ είναι $d_i \cdot d_{i+1} \cdot d_{i+2}$ γιατί:



- Π.χ., για τρεις πίνακες διαστάσεων 5×4, 4×6, 6×2 έχουμε δύο διαφορετικούς τρόπους τοποθετήσεως των παρενθέσεων:

  $A_0 \cdot (A_1 \cdot A_2)$ κόστους $4 \cdot 6 \cdot 2 + 5 \cdot 4 \cdot 2 = 88$

  $(A_0 \cdot A_1) \cdot A_2$ κόστους $5 \cdot 4 \cdot 6 + 5 \cdot 6 \cdot 2 = 180$

- Αλγόριθμος brute force:
  - Θα εξετάσουμε όλες τις δυνατές τοποθετήσεις
  - Υπάρχουν n-1 θέσεις για να τοποθετήσουμε τις παρενθέσεις:

$$(A_0 \cdot A_1 \cdot \cdot \cdot A_{k-1}) \cdot (A_k A_{k+1} \cdot \cdot \cdot A_{n-1})$$

  - Το πλήθος των πιθανών τρόπων πολλαπλασιασμού:

$$\Pi(n) = \sum_{1 \le k \le n} \Pi(k) \cdot \Pi(n-k)$$

Λύση: $C(n\text{-}1) = \Omega(4^n / n^{1.5})$

$$C(n) = \frac{1}{n+1} \begin{pmatrix} 2n \\ n \end{pmatrix}$$

- Η σωστή αντιμετώπιση:

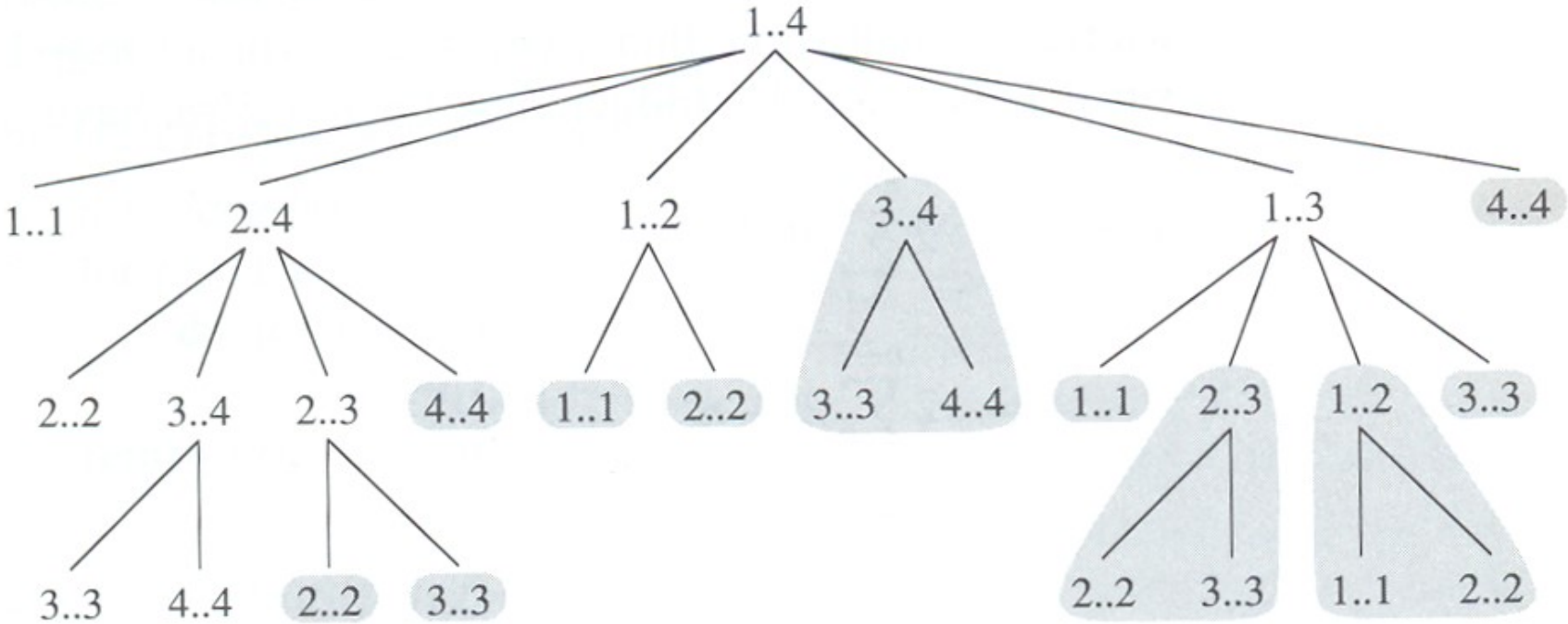
  Έστω $A_{i,j} = A_i \cdot A_{i+1} \cdot \cdot \cdot A_j$, με κόστος $M_{i,j}$.

  Τότε: $A_{0,n-1} = A_{0,k-1} \cdot A_{k,n-1}$

  και: $M_{0,n-1} = M_{0,k-1} + M_{k,n-1} + d_0 \cdot d_k \cdot d_n$
- Γενικό βήμα:

  Εύρεση του βέλτιστου *k, αναδρομικά,* αποθηκεύοντας τα ενδιάμεσα αποτελέσματα.
- Επίλυση του $A_{i,j}$ με κόστος

$$M_{i,j} = \begin{cases} \min_{i<k\leq j}\{M_{i,k-1} + M_{k,j} + d_i \cdot d_k \cdot d_j\} & i<j \\ 0 & i=j \end{cases}$$



Πρέπει να είναι διαθέσιμες οι βέλτιστες λύσεις $M_{i,k-1}$ και M_{kj} για να υπολογίσουμε το M*i,j*

Παράδειγμα για πίνακες διαστάσεως 5×2, 2×1, 1×2, 2×3, 3×2 ,

Λύση: $(A_0 \cdot A_1) \cdot ((A_2 \cdot A_3) \cdot A_4)$

$A_0 \cdot (A_1 \cdot A_2 \cdot A_3)$, 0+12+5×2×3=32

$(A_0 \cdot A_1) \cdot (A_2 \cdot A_3)$,10+6+5×1×3=31

$(A_0 \cdot A_1 \cdot A_2) \cdot A_3$, 20+0+5×2×3=50

$A_0 \cdot (A_1 \cdot A_2)$, 0+4+5×2×2=24

$(A_0 \cdot A_1) \cdot A_2$, 10+0+5×1×2=20

	0	1	2	3	4
0	0	10=5×2×1	20	31	32
1		0	4 =2×1×2	12	16
2			0	6 =1×2×3	12
3				0	12 =2×3×2
4					0

	0	1	2	3	4
0	**0**	**1**	2	2	**2**
1		**1**	2	2	2
2			**2**	**3**	**4**
3				**3**	4
4					**4**