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.

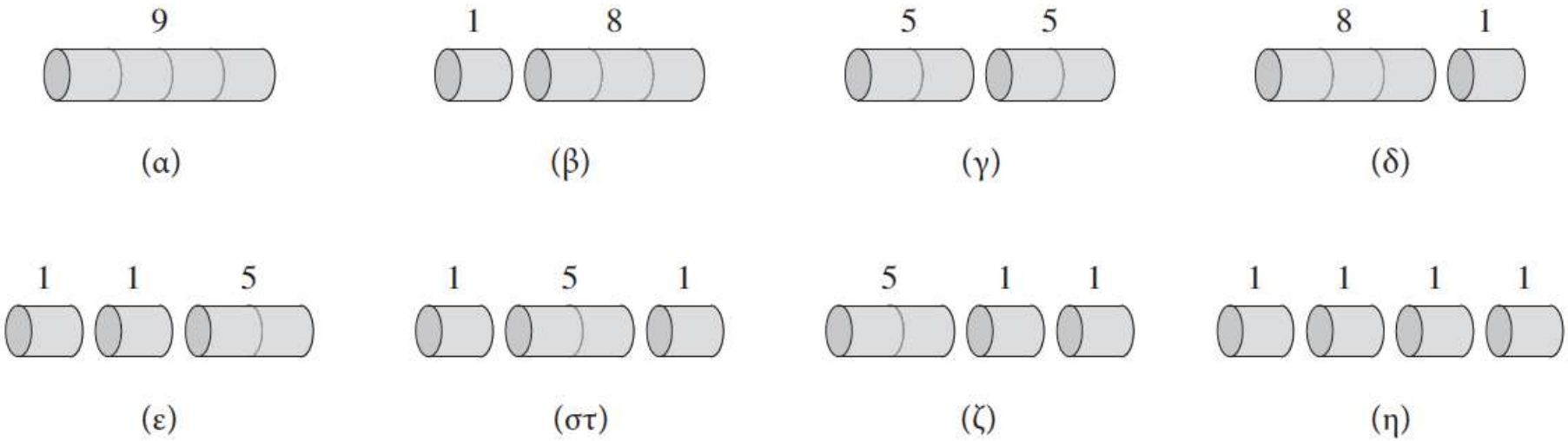
Το πρόβλημα της κοπής ράβδου

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

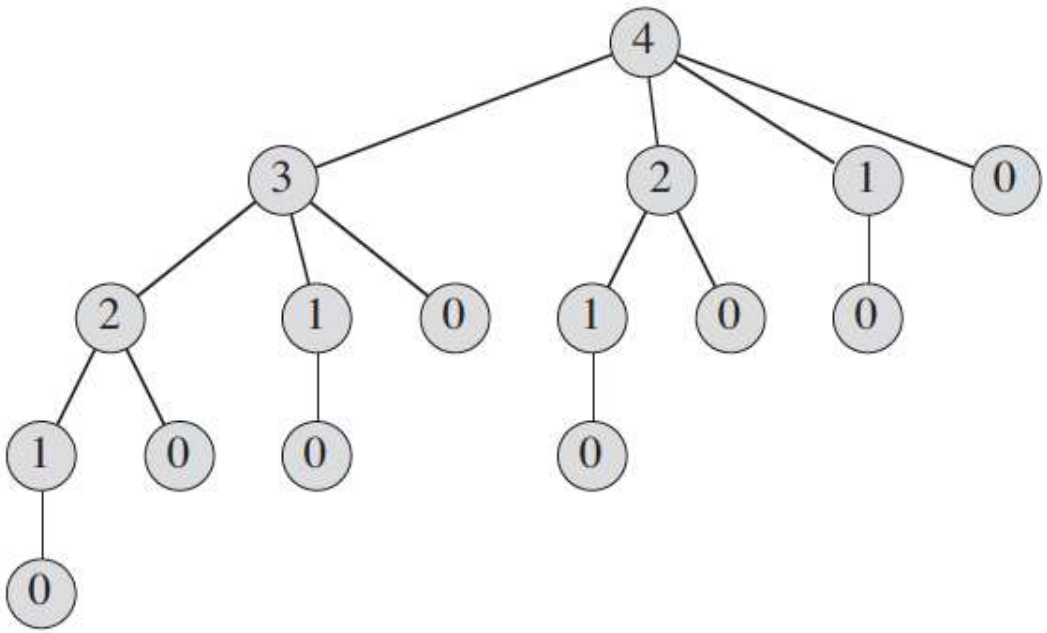
$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})$$

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

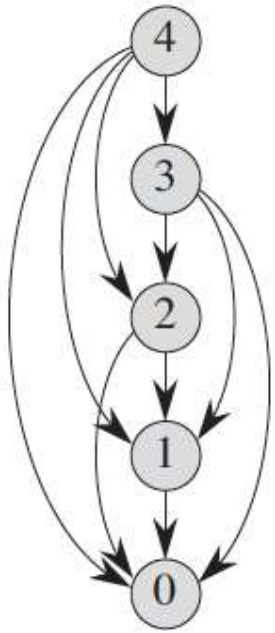
Βέλτιστη κοπή ράβδου μέσω δυναμικού προγραμματισμού

```
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)

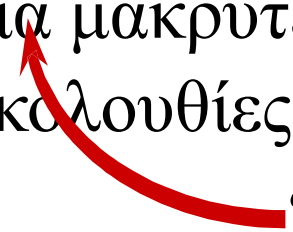
- Δοθεισών δύο ακολουθιών $x[1 \,..\, m]$ και $y[1 \,..\, n]$, βρες μία μακρύτερη υπακολουθία κοινή και στις δύο ακολουθίες.

Δυναμικός Προγραμματισμός

Τεχνική Σχεδιασμού, όπως η διαίρει και βασίλευε.

Παράδειγμα: Μακρύτερη Κοινή Υπακολουθία (ΜΚΥ)

- Δοθεισών δύο ακολουθιών $x[1 \ldots m]$ και $y[1 \ldots n]$, 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

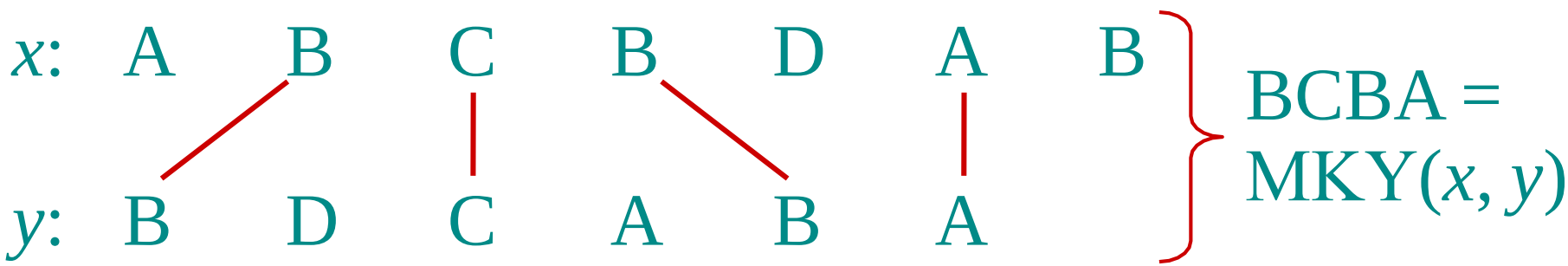
Δυναμικός Προγραμματισμός

Τεχνική Σχεδιασμού, όπως η διαίρει και βασίλευε.

Παράδειγμα: Μακρύτερη Κοινή Υπακολουθία (ΜΚΥ)

- Δοθεισών δύο ακολουθιών $x[1 \ldots m]$ και $y[1 \ldots n]$, 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.

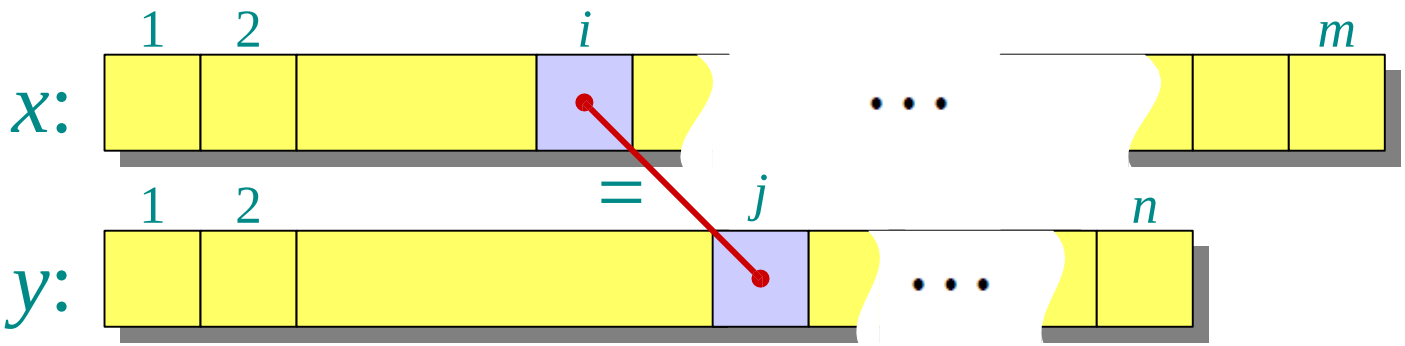
- Ορίζουμε $c[i, j] = |\text{MKY}(x[1\,..\,i], y[1\,..\,j])|$.
- Τότε, $c[m, n] = |\text{MKY}(x, y)|$.

Αναδρομική διατύπωση

Θεώρημα.

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

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

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

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

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.

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.

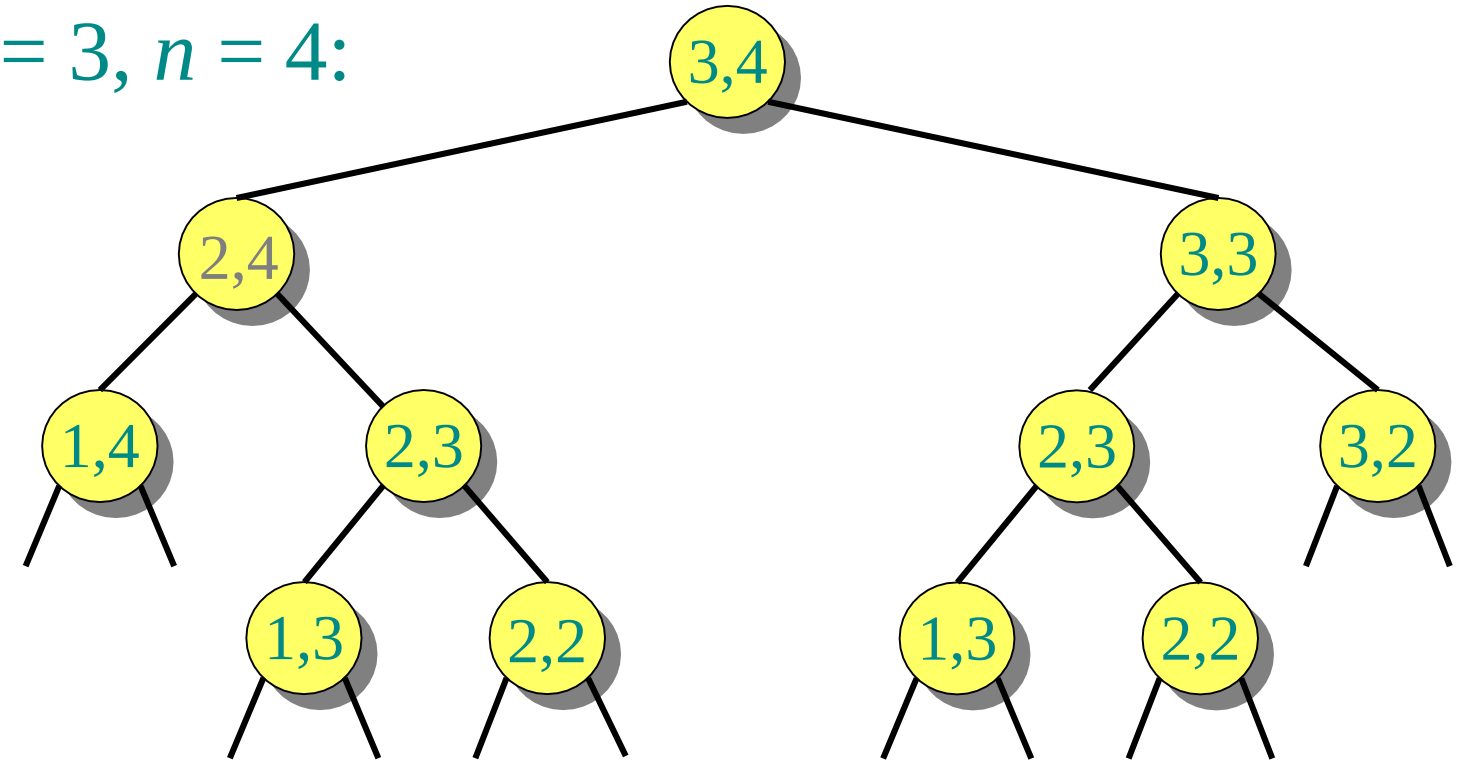
Αναδρομικός αλγόριθμος για MKY

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.

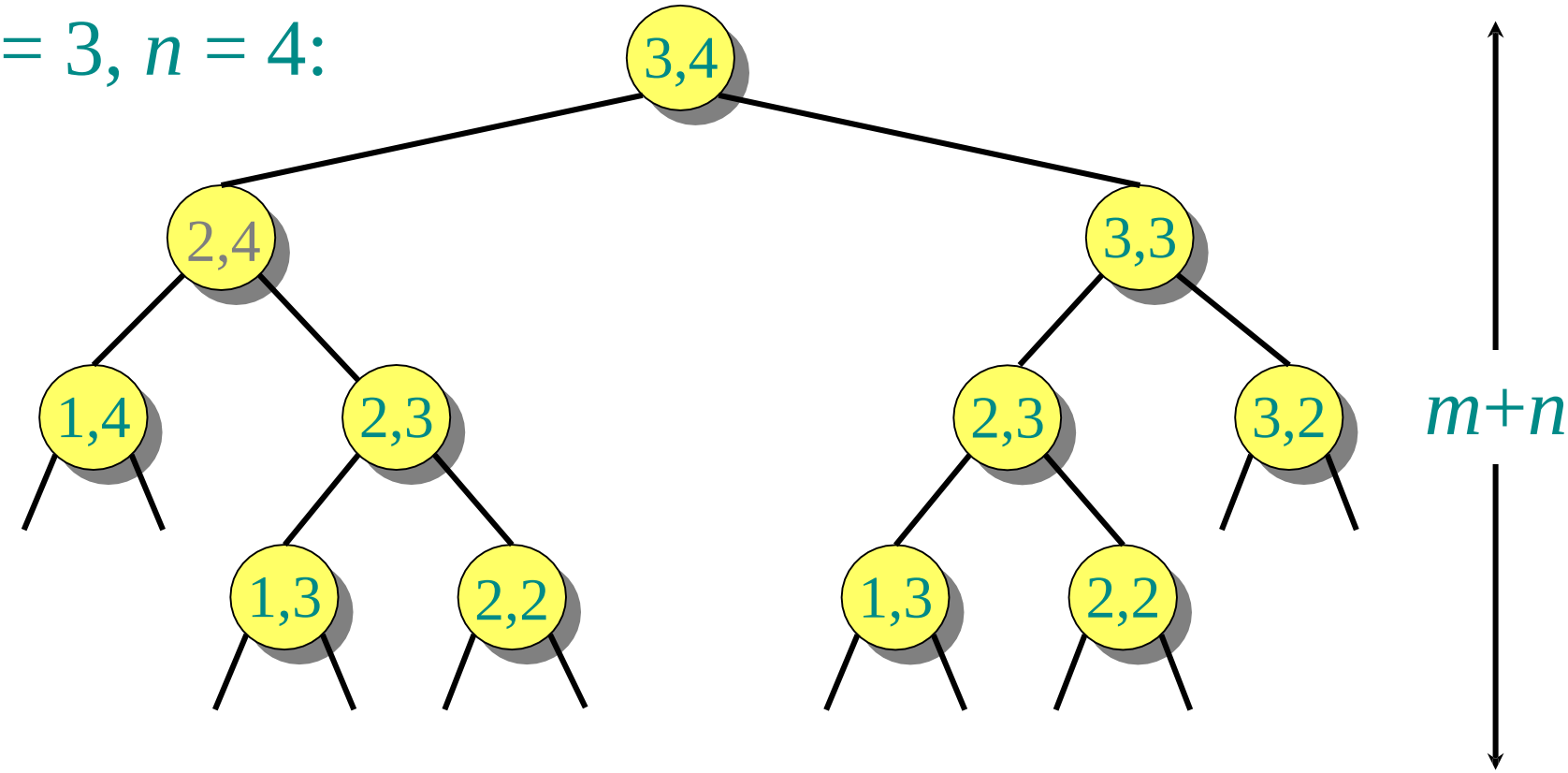
Δένδρο Αναδρομής

$m = 3, n = 4$:



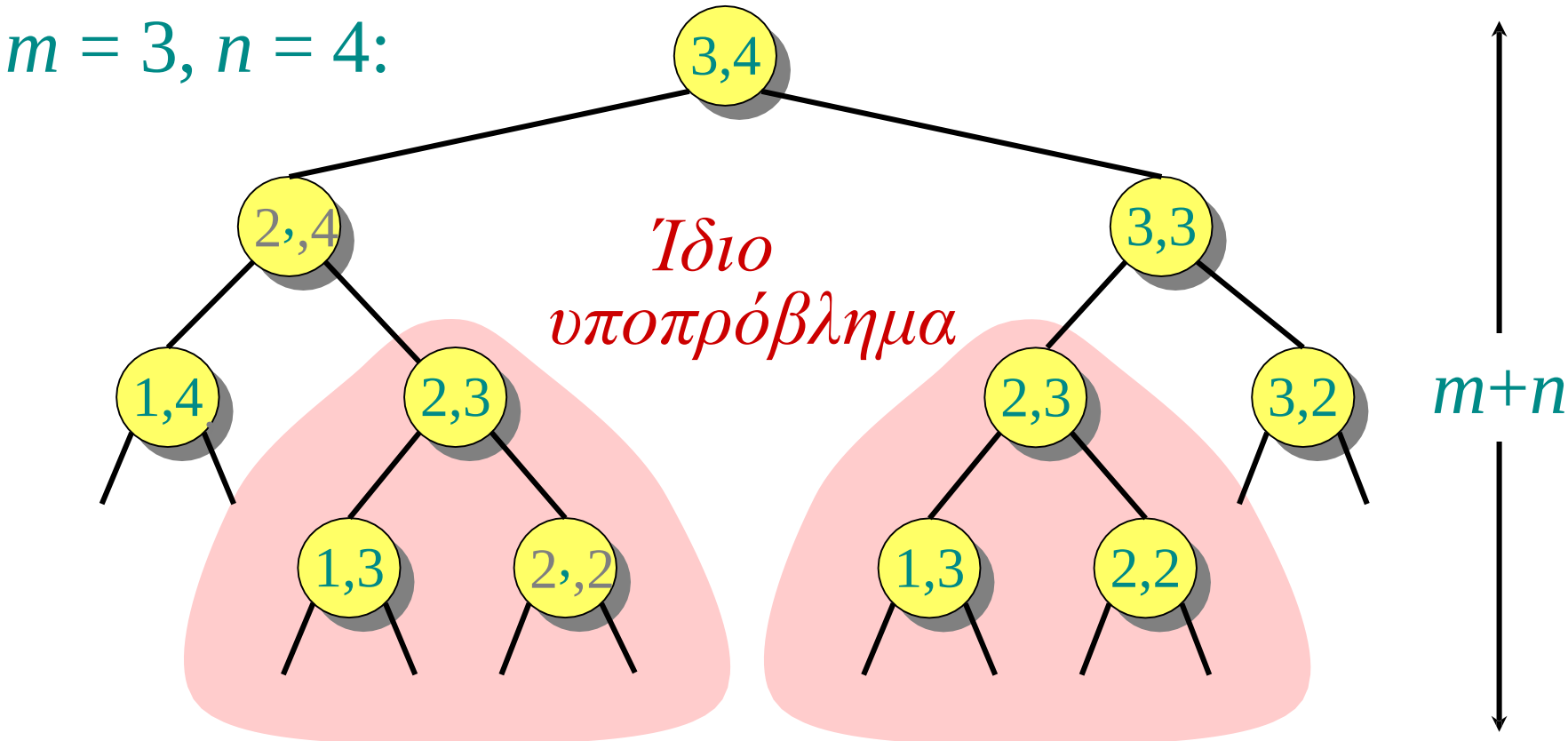
Δένδρο Αναδρομής

$m = 3, n = 4$:



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

Δένδρο Αναδρομής



Ύψος = $m + n$ $\Rightarrow$ 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 $c[i, j] =$ NIL
    then if $x[i] = y[j]$
      then $c[i, j] \leftarrow$ LCS$(x, y, i–1, j–1) + 1$
      else $c[i, j] \leftarrow \max\{LCS(x, y, i–1, j),$
        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

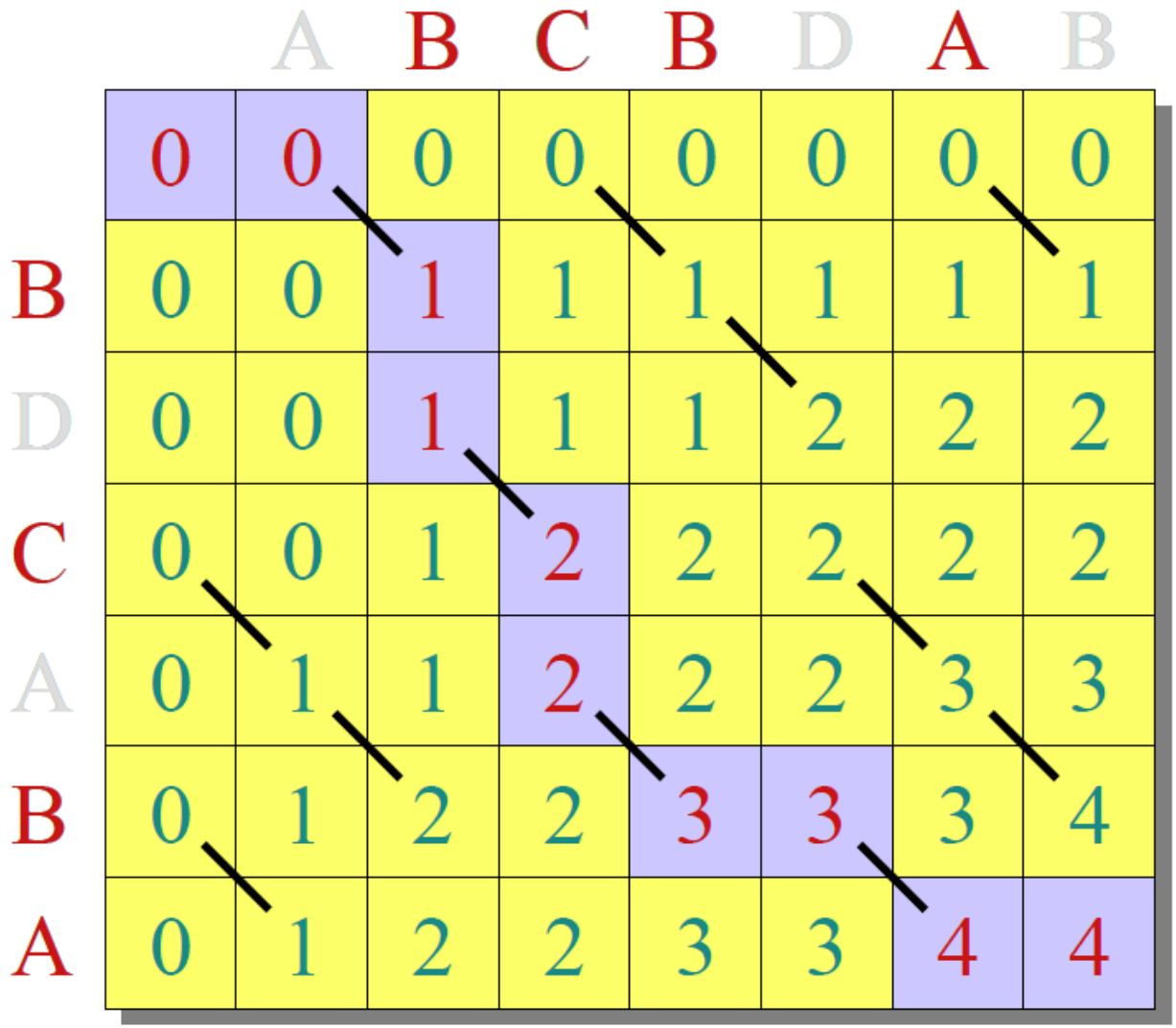
Ιδέα:

Υπολόγισε τον πίνακα από πάνω προς τα κάτω.

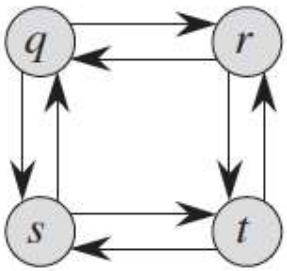
Χρόνος = $\Theta(mn)$.

Ανακατασκεύασε την ΜΚΥ με ανίχνευση προς τα πίσω.

Χώρος= $\Theta(mn)$.



Πρόβλημα με μη βέλτιστη υποδομή



Σχήμα 15.6 Ένα κατευθυντό γράφημα το οποίο καταδεικνύει ότι το πρόβλημα της εύρεσης μιας μακρύτατης απλής διαδρομής σε ένα αβαρές κατευθυντό γράφημα δεν έχει βέλτιστη υποδομή. Αν και η διαδρομή $q \to r \to t$ είναι μια μακρύτατη απλή διαδρομή από τον κόμβο q μέχρι τον κόμβο t, οι υποδιαδρομές $q \to r$ και $r \to t$ δεν είναι μακρύτατες απλές διαδρομές από τον q μέχρι τον r και από τον r μέχρι τον t, αντίστοιχα.

Βέλτιστος Πολλαπλασιασμός Πινάκων

- Δίνεται μία ακολουθία πινάκων:

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

  να βρεθεί ο πιο γρήγορος τρόπος υπολογισμού του γινομένου:

  $A_0 \cdot A_1 \cdot \cdots \cdot A_{n-1}$.
- 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:

$$d_i \left\{ \underbrace{[\ ][\ ]}_{d_{i+1}} [\ ] \right\} d_{i+1} \qquad d_{i+2}$$

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

$$C(n) = \frac{1}{n+1} \binom{2n}{n}$$

- Η σωστή αντιμετώπιση:

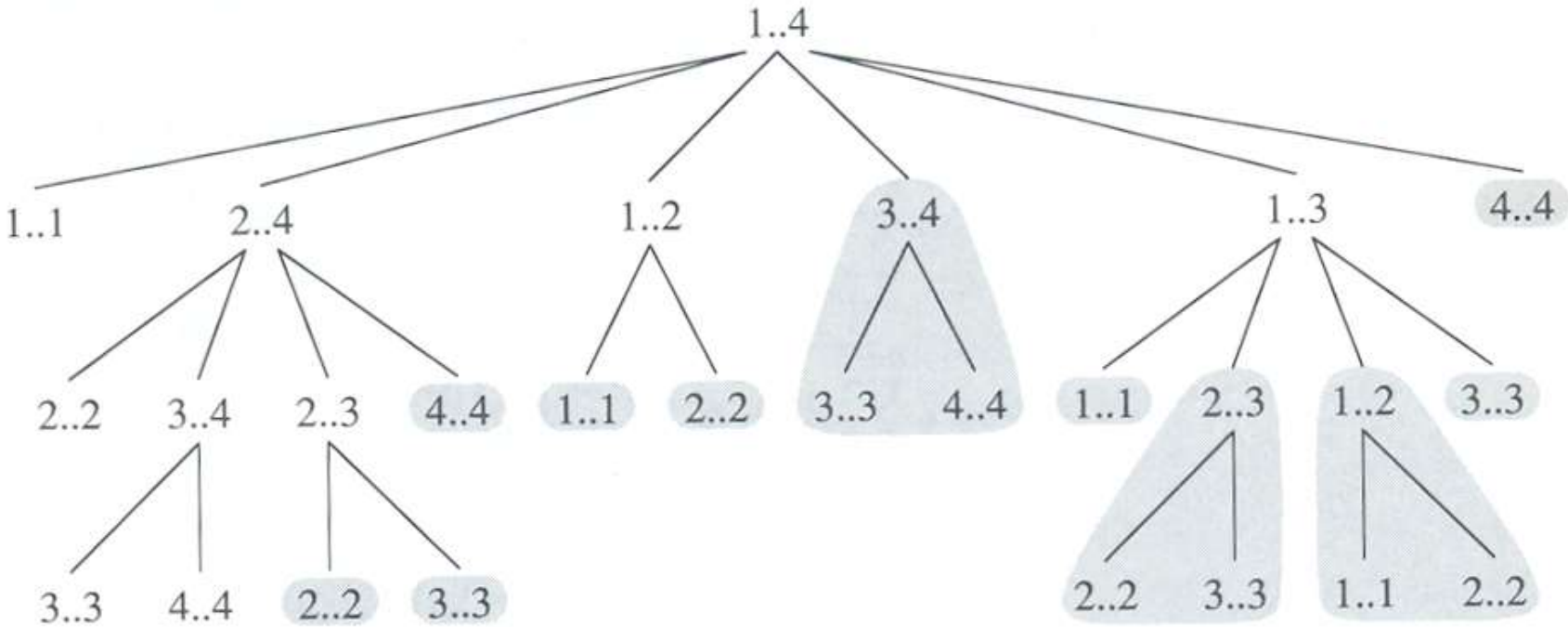
  Έστω $A_{i,j} = A_i \cdot A_{i+1} \cdot \cdot \cdot A_j$, με κόστος $M_{i,j}$.

  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

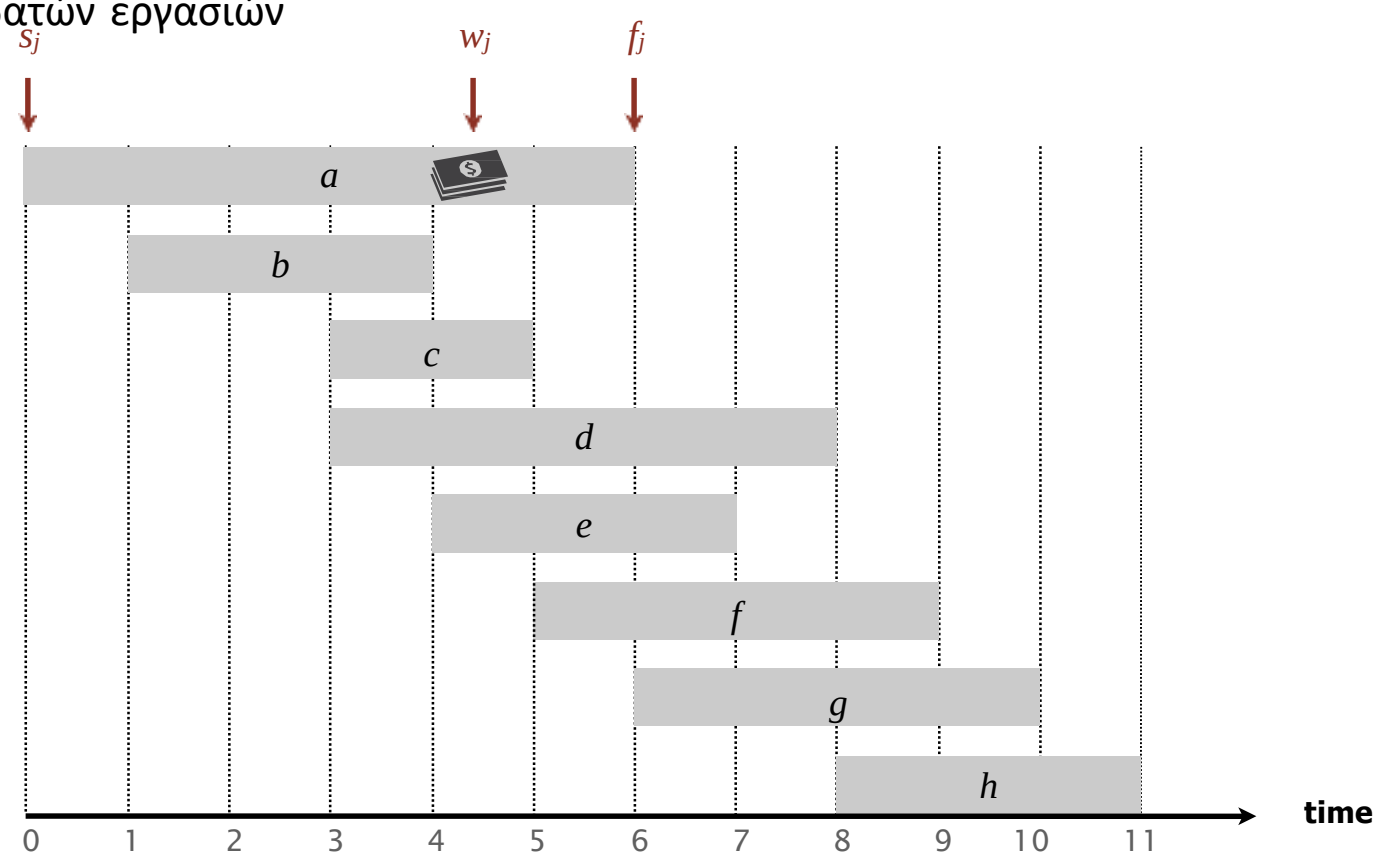
$(A_0 \cdot A_1) \cdot A_2$, 10+0+5×1×2=20

	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**

Βαροζυγισμένη δρομολόγηση διαστημάτων

- Η εργασία j εκκινεί την χρονική στιγμή s_j, τελειώνει την χρονική στιγμή f_j, και έχει βάρος $w_j > 0$.
- Δύο έργα είναι συμβατά αν δεν επικαλύπτονται.
- Ο στόχος: βρες το μεγίστου βάρους υποσύνολο των αμοιβαία συμβατών εργασιών

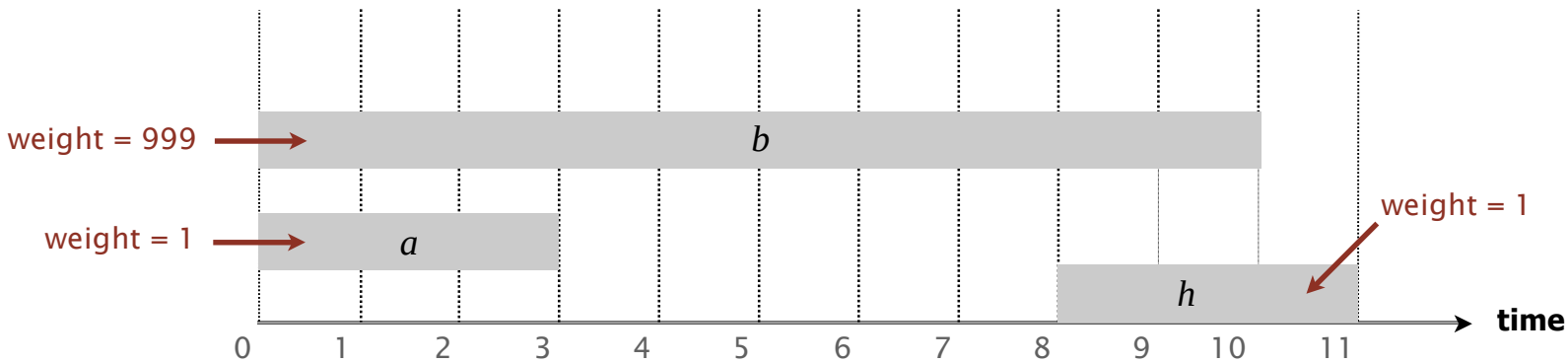


Αλγόριθμος Earliest-finish-time first

Earliest finish-time first.

- Εξέτασε τα έργα σε αύξουσα σειρά ως προς το χρόνο ολοκλήρωσης.
- Πρόσθεσε το έργο στο υποσύνολο αν αυτό είναι συμβατό με τα προηγούμενα επιλεχθέντα έργα.
- Ο άπληστος αλγόριθμος είναι ορθός αν τα βάρη είναι 1.

Παρατήρηση. Ο άπληστος αλγόριθμος αποτυγχάνει για τη βαροζυγισμένη εκδοχή.



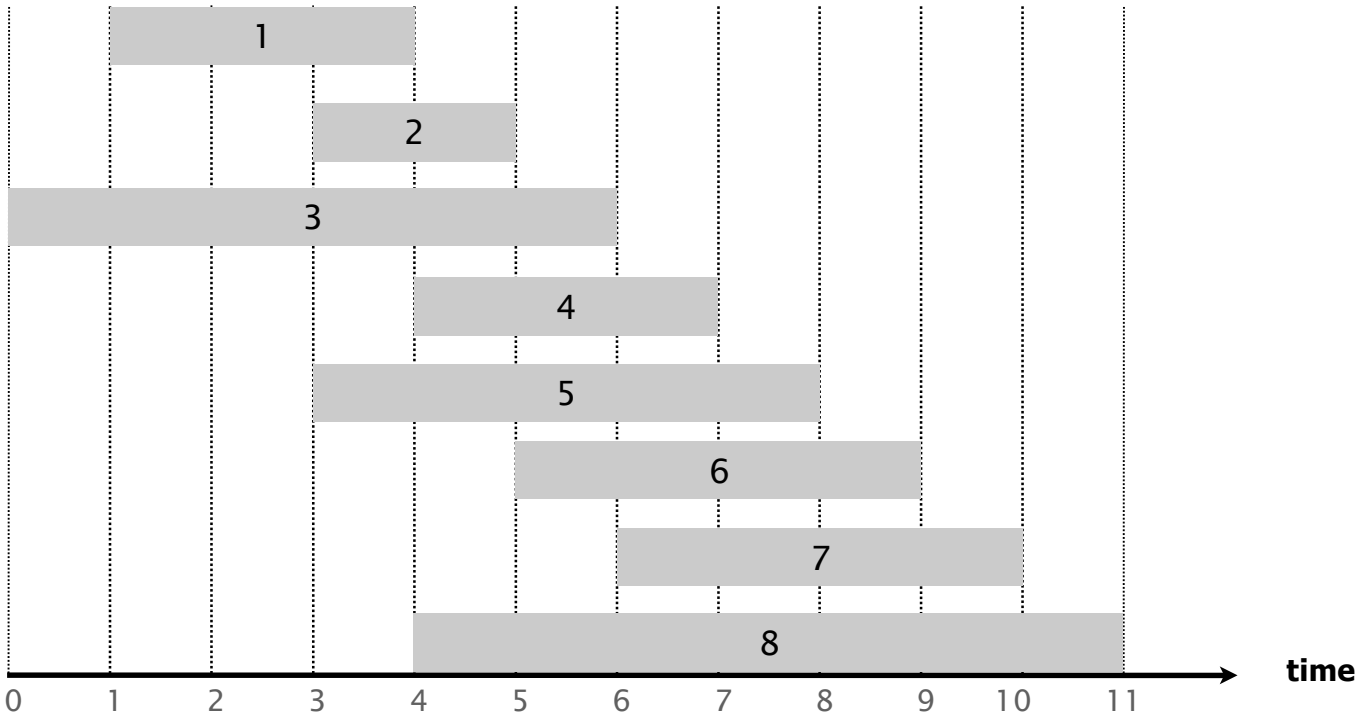
Βαροζυγισμένη δρομολόγηση διαστημάτων

Σύμβαση. Τα έργα είναι σε αύξουσα σειρά του χρόνου ολοκλήρωσης: $f_1 \leq f_2 \leq \ldots \leq f_n$.

Ορι. $p(j)$ = ο μεγαλύτερος δείκτης $i < j$ τέτοιο ώστε το έργο i είναι συμβατό με j.

Παρ. $p(8) = 1, p(7) = 3, p(2) = 0.$

i είναι το πιο δεξιό διάστημα το οποίο ολοκληρώνεται πριν το j αρχίσει



Δυναμικός προγραμματισμός: δυαδική επιλογή

Ορι. $OPT(j)$ = μέγιστο βάρος οποιουδήποτε υποσυνόλου αμοιβαίων συμβατών εργασιών για το υποπρόβλημα που αποτελείται μόνο από τα έργα 1, 2, …, j.

Στόχος. $OPT(n)$ = μέγιστο βάρος οποιουδήποτε υποσυνόλου αμοιβαία συμβατών έργων

Περίπτωση 1. $OPT(j)$ δεν επιλέγει το έργο j.

- Πρέπει να είναι μία βέλτιστη λύση στο πρόβλημα που αποτελείται από τα υπόλοιπα 1, 2, ..., $j-1$.

ιδιότητα βέλτιστης υποδομής

Περίπτωση 2. $OPT(j)$ επιλέγει το έργο j.

- Συλλογή κερδών w_j.
- Δεν χρησιμοποιούνται μη συμβατά έργα{ $p(j)+1, p(j)+2, \ldots, j-1$ }.
- Πρέπει να περιλαμβάνει τη βέλτιστη λύση στο πρόβλημα που αποτελείται από τα υπόλοιπα έργα 1, 2, ..., $p(j)$.

Bellman equation.

$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max\{OPT(j-1),\ w_j + OPT(p(j))\} & \text{if } j > 0 \end{cases}$$

Βαροζυγισμένη δρομολόγηση διαστημάτων: ωμή βια

BRUTE-FORCE (n, s_1, …, s_n, f_1, …, f_n, w_1, …, w_n)

Sort jobs by finish time and renumber so that $f_1 \leq f_2 \leq \ldots \leq f_n$.

Compute $p[1], p[2], \ldots, p[n]$ via binary search.

RETURN COMPUTE-OPT(n).

COMPUTE-OPT(j)

IF ($j = 0$)

  RETURN 0.

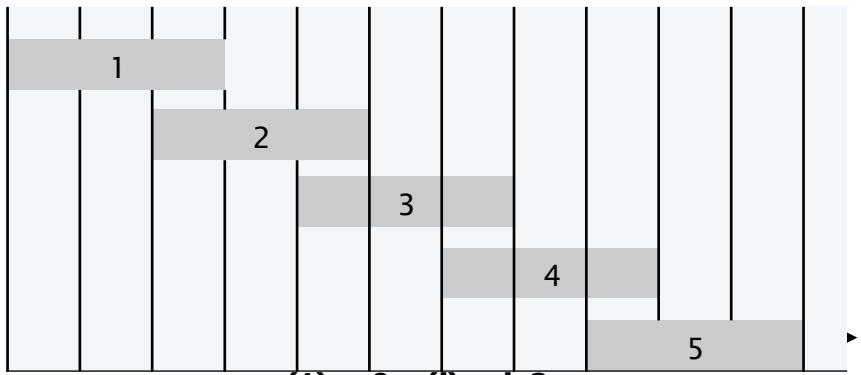
ELSE

  RETURN max {COMPUTE-OPT($j-1$), w_j + COMPUTE-OPT($p[j]$) }.

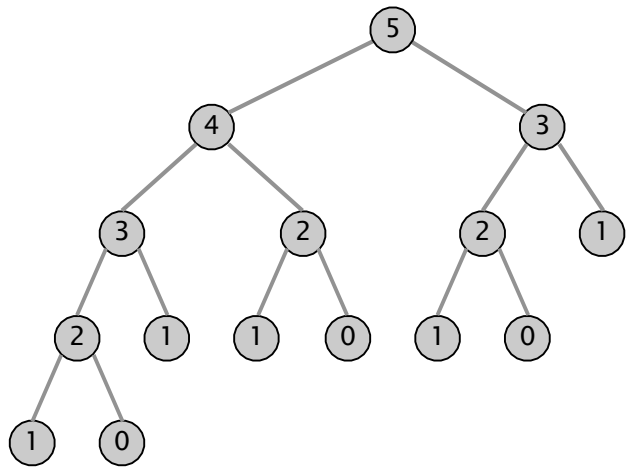
Βαροζυγισμένη δρομολόγηση διαστημάτων: ωμή βια

Παρατήρηση. Οι αναδρομικοί αλγόριθμοι είναι εξαιρετικά αργοί εξαιτίας των επικαλυπτόμενων προβλημάτων $\Rightarrow$ εκθετικού χρόνου αλγόριθμος.

Παρ. Το πλήθος των αναδρομικών κλήσεων για μία οικογένεια «διαστρωμένων» στιγμιοτύπων αυξάνεται όπως η ακολουθία Fibonacci.



p(1) = 0, p(j) = j-2



recursion tree

Βαροζυγισμένη δρομολόγηση διαστημάτων: απομνημόνευση

Top-down δυναμικός προγραμματισμός (απομνημόνευση).

Αποθήκευσε το αποτέλεσμα του υποπροβλήματος j στη θέση $M[j]$.

Χρησιμοποίησε το $M[j]$ για να αποφύγεις να επιλύσεις περισσότερες από μία φορά.

TOP-DOWN(n, s_1, …, s_n, f_1, …, f_n, w_1, …, w_n)

Sort jobs by finish time and renumber so that $f_1 \leq f_2 \leq \ldots \leq f_n$.

Compute $p[1], p[2], \ldots, p[n]$ via binary search.

$M[0] \leftarrow 0$. ⟵ Σφαιρική (global) μεταβλητή

RETURN M-COMPUTE-OPT(n).

M-COMPUTE-OPT(j)

IF ($M[j]$ is uninitialized)

$M[j] \leftarrow \max \{$ M-COMPUTE-OPT $(j-1)$, w_j + M-COMPUTE-OPT($p[j]$) $\}$.

RETURN $M[j]$.

Βαροζυγισμένη δρομολόγηση διαστημάτων: ο χρόνος εκτέλεσης

Ισχυρισμός. Η τεχνική απομνημόνευσης παίρνει χρόνο $O(n \log n)$ time.

Απόδειξη.

- Sort by finish time: $O(n \log n)$ με mergesort.
- Compute $p[j]$ for each j : $O(n \log n)$ με δυαδική αναζήτηση.

- M-COMPUTE-OPT(j): κάθε κλήση θέλει χρόνο $O(1)$ και είτε
  - (1) επιστρέφει μία αρχικοποιημένη τιμή $M[j]$
  - (2) Αρχικοποιεί το $M[j]$ και κάνει δύο αναδρομικές κλήσεις

- Μέτρο προόδου Φ = # αρχικοποιημένων στοιχείων εντός του $M[1..n]$.
  - αρχικά $\Phi = 0$; $\Phi \leq n$ καθόλη τη διάρκεια της εκτέλεσης.
  - (2) αυξάνεται Φ κατά $1 \Rightarrow \leq 2n$ αναδρομικές κλήσεις.

- Συνολικός χρόνος εκτέλεσης της M-COMPUTE-OPT(n) είναι $O(n)$.

▪

Το πρόβλημα του μέγιστου υποπίνακα

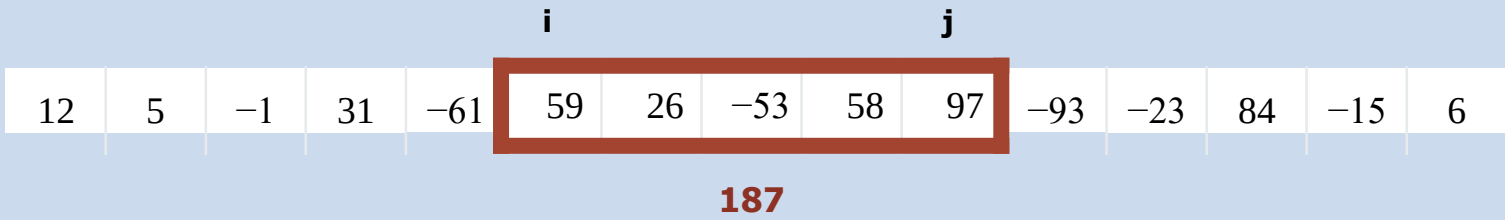
Στόχος. Δοθέντος ενός πίνακα x n ακεραίων (θετικοί ή αρνητικοί), βρες ένα συνεχόμενο υποπίνακα του οποίου το άθροισμα είναι μέγιστο.

12	5	−1	31	−61	59	26	−53	58	97	−93	−23	84	−15	6

187

MAXIMUM SUBARRAY PROBLEM

Στόχος. Δοθέντος ενός πίνακα $x\ n$ ακεραίων (θετικοί ή αρνητικοί), βρες ένα συνεχόμενο υποπίνακα του οποίου το άθροισμα είναι μέγιστο.



Αλγόριθμος ωμής βίας.

- Για κάθε i και j : υπολόγισε $a[i] + a[i+1] + \ldots + a[j]$.
- Θέλει χρόνο $\Theta(n^3)$.

Εφάρμοσε την τεχνική του "σωρευτικού αθροίσματος"

- Υπολόγισε από πριν τα σωρευτικά αθροίσματα: $S[i] = a[0] + a[1] + \ldots + a[i]$.

Τώρα $a[i] + a[i+1] + \ldots + a[j] = S[j] - S[i-1]$.

- Βελτιώνει το χρόνο εκτέλεσης σε $\Theta(n^2)$.

KADANE'S ALGORITHM

Ορ. $OPT(i)$ = το μέγιστο άθροισμα οποιουδήποτε υποπίνακα του x του οποίου ο πιο δεξιός δείκτης είναι ο i.

Στόχος. $\max_i OPT(i)$

Εξίσωση Bellman.

$$OPT(i) = \begin{cases} x_1 & \text{if } i = 1 \\ \max\{x_i,\ x_i + OPT(i-1)\} & \text{if } i > 1 \end{cases}$$

Πάρε μόνο το στοιχείο i

Πάρε το στοιχείο i μαζί με το καλύτερο υποπίνακα που τελειώνει στον δείκτη $i-1$

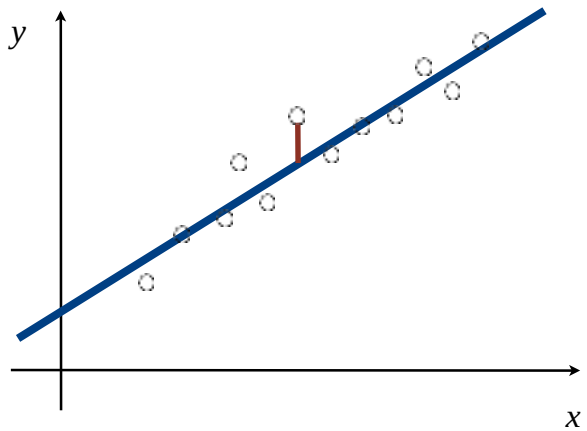
Χρόνος εκτέλεσης. $O(n)$.

Ελάχιστα Τετράγωνα

Ελάχιστα τετράγωνα. Βασικό πρόβλημα στη Στατιστική.

- Δοθέντων n σημείων στο επίπεδο: (x_1, y_1), (x_2, y_2) , …, (x_n, y_n).
- Βρες μία ευθεία $y = ax + b$ η οποία ελαχιστοποιεί το άθροισμα του τετραγωνικού σφάλματος:

$$SSE = \sum_{i=1}^{n} (y_i - ax_i - b)^2$$



Λύση. Το ελάχιστο λάθος επιτυγχάνεται όταν

$$a = \frac{n\sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{n\sum_i x_i^2 - (\sum_i x_i)^2}, \quad b = \frac{\sum_i y_i - a\sum_i x_i}{n}$$

Τμηματικά ελάχιστα τετράγωνα

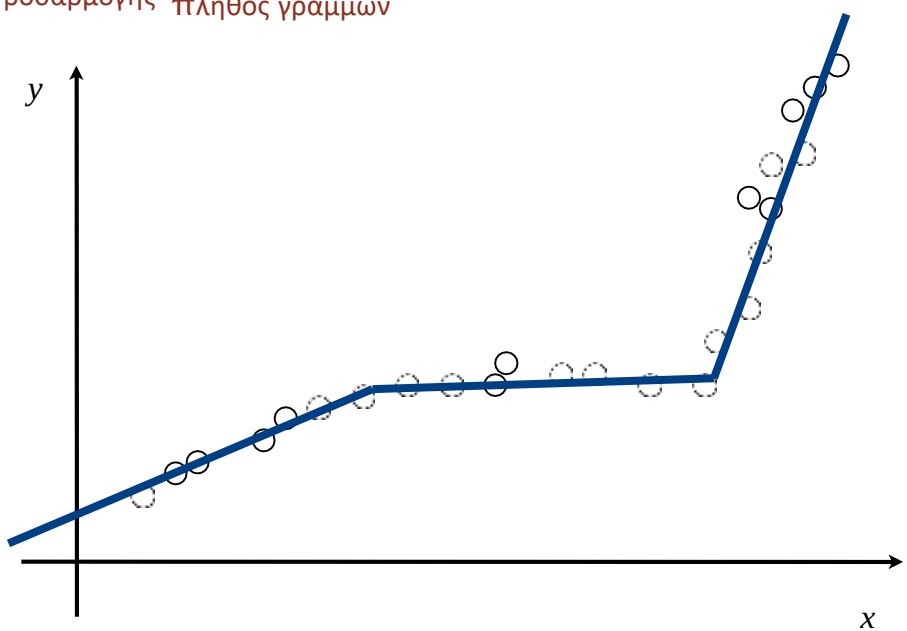
Τμηματικά ελάχιστα τετράγωνα.

- Τα σημεία είναι χονδρικά πάνω σε μία ακολουθία μερικών γραμμικών τμημάτων.
- Δοθέντων n σημείων σε ένα επίπεδο: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ με $x_1 < x_2 < \ldots < x_n$, βρες μία ακολουθία γραμμών που ελαχιστοποιούν την $f(x)$.

Ε. Ποια είναι η πιο αξιόπιστη επιλογή για την $f(x)$ για εξισορρόπηση μεταξύ ακρίβειας και οικονομίας;

επίπεδο προσαρμογής

πλήθος γραμμών



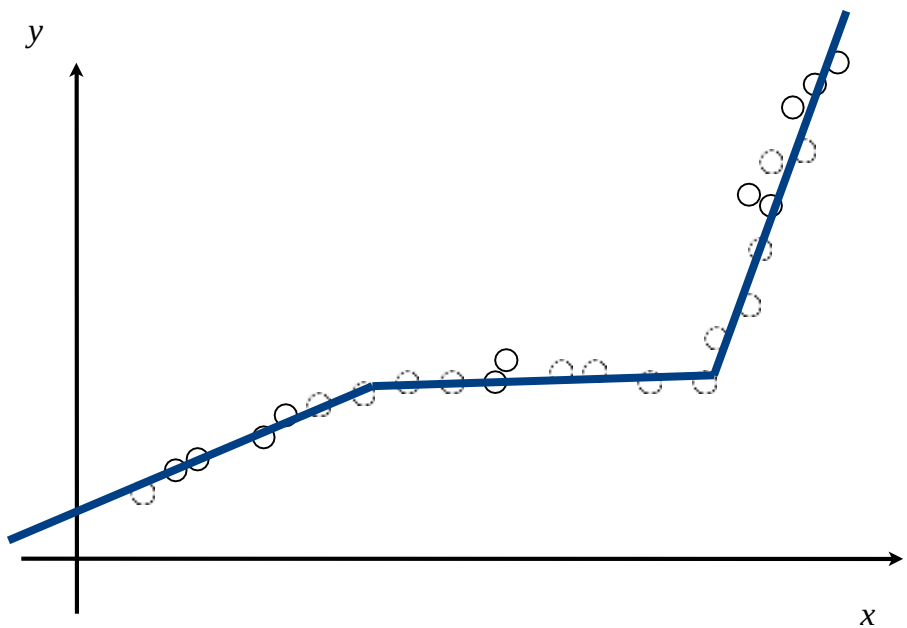
Τμηματικά ελάχιστα τετράγωνα

Τμηματικά ελάχιστα τετράγωνα.

- Τα σημεία είναι χονδρικά πάνω σε μία ακολουθία μερικών γραμμικών τμημάτων.
- Δοθέντων n σημείων σε ένα επίπεδο: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ με $x_1 < x_2 < \ldots < x_n$, βρες μία ακολουθία γραμμών που ελαχιστοποιούν την $f(x)$.

Στόχος. Ελαχιστοποίηση της $f(x) = E + c\,L$ για κάποια σταθερά $c > 0$, όπου

- E = το άθροισμα των αθροισμάτων των τετραγωνικών λαθών σε κάθε τμήμα.
- L = Πλήθος γραμμών.



Δυναμικός προγραμματισμός: επιλογή μεταξύ πολλάπλων εναλλακτικών

Συμβολισμός.

- $OPT(j)$ = ελάχιστο κόστος για τα σημεία $p_1, p_2, \ldots, p_j$.
- e_{ij} = SSE για τα σημεία $p_i, p_{i+1}, \ldots, p_j$.

Υπολογισμός του $OPT(j)$:

- Το τελευταίο τμήμα χρησιμοποιεί τα σημεία $p_i, p_{i+1}, \ldots, p_j$ για κάποια $i \leq j$.
- Cost = $e_{ij} + c + OPT(i-1)$. ← Ιδιότητα βέλτιστής υποδομής

Η εξίσωση του Bellman.

$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \min_{1 \leq i \leq j} \{ e_{ij} + c + OPT(i-1) \} & \text{if } j > 0 \end{cases}$$

Τμηματικά ελάχιστα τετράγωνα

SEGMENTED-LEAST-SQUARES(n, p_1, …, p_n, c)

FOR j = 1 TO n

  FOR i = 1 TO j

    Compute the SSE e_{ij} for the points p_i, p_{i+1}, …, p_j.

$M[0] \leftarrow 0$.

FOR $j = 1$ TO n

  $M[j] \leftarrow \min_{1 \le i \le j} \{ e_{ij} + c + M[i-1] \}$.

Προηγούμενη υπολογισμένη τιμή

RETURN $M[n]$.

Τμηματικά ελάχιστα τετράγωνα

Θεώρημα. [Bellman 1961] Ο αλγόριθμος δυναμικού προγραμματισμού επιλύει το πρόβλημα των τμηματικών ελαχίστων τετραγώνων σε $O(n^3)$ χρόνο και $O(n^2)$ χώρο.

Απ.

- Το πιο χρονοβόρο σημείο = υπολογισμός του SSE e_{ij} για κάθε i και j.

$$a_{ij} = \frac{n\sum_k x_k y_k - (\sum_k x_k)(\sum_k y_k)}{n\sum_k x_k^2 - (\sum_k x_k)^2}, \quad b_{ij} = \frac{\sum_k y_k - a_{ij}\sum_k x_k}{n}$$

- $O(n)$ για τον υπολογισμό του e_{ij}.

Σημείωση. Μπορεί να βελτιωθεί $O(n^2)$ χρόνο.

- Για κάθε i : υπολόγισε από πριν τα αθροίσματα

$$\sum_{k=1}^{i} x_k, \quad \sum_{k=1}^{i} y_k, \quad \sum_{k=1}^{i} x_k^2, \quad \sum_{k=1}^{i} x_k y_k$$

- Χρησιμοποιώντας αυτά τα αθροίσματα μπορούμε να υπολογίσουμε το e_{ij} σε χρόνο $O(1)$.