Άπληστοι Αλγόριθμοι

- Οι διαφάνειες βασίζονται στις ακόλουθες πηγές:
  - Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms. MIT press
  - Lecture Slides for Algorithm Design (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/)

Ένα πρόβλημα επιλογής δραστηριοτήτων

- Δίνεται κάποιο σύνολο $S = a_1, a_2, \ldots, a_n$ από n ***δραστηριότητες*** που απαιτούν τη χρήση ενός πόρου ο οποίος μπορεί να χρησιμοποιείται μόνο από μία δραστηριότητα κάθε φορά.
- Η κάθε δραστηριότητα a_i έχει ***χρόνο έναρξης*** s_i και ***χρόνο λήξης*** f_i, όπου $0 < s_i < f_i < \infty$.
- Οι δραστηριότητες a_i και a_j χαρακτηρίζονται ***συμβατές*** εάν τα διαστήματα $[s_i, f_i)$ και $[s_j , f_j)$ δεν επικαλύπτονται
- Το ***πρόβλημα της επιλογής δραστηριοτήτων:*** η εύρεση ενός μέγιστου συνόλου αμοιβαία συμβατών δραστηριοτήτων.

Ένα πρόβλημα επιλογής δραστηριοτήτων

Οι δραστηριότητες είναι διατεταγμένες κατά μονότονα αύξουσα σειρά ως προς τον χρόνο λήξης:

$$f_1 \leq f_2 \leq f_3 \leq \cdots \leq f_{n-1} \leq f_n$$

Π.χ.:

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

Αμοιβαία συμβατές δραστηριοτήτες: $\{a_3, a_9, a_{11}\}$
Μέγιστα σύνολα: $\{a_1, a_4, a_8, a_{11}\}$, $\{a_2, a_4, a_9, a_{11}\}$.

Ορίζουμε

$$S_{ij} = \{a_k \in S : f_i \leq s_k < f_k < s_j\}$$

c[i,j] = το πλήθος των δραστηριοτήτων σε ένα μέγιστο υποσύνολο αμοιβαία συμβατών δραστηριοτήτων στο S_{ij}.

Προσθέτουμε τις φανταστικές δραστηριότητες a_0 με χρόνο λήξης 0 και a_{n+1} με χρόνο έναρξης ∞.

Η απαιτούμενη λύση: c[0, n+1].

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \varnothing \\ \max_{\substack{i<k<j \\ a_k \in S_{ij}}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \varnothing \end{cases}$$

Η αναδρομική εξίσωση οδηγεί σε λύση δυναμικού προγραμματισμού.

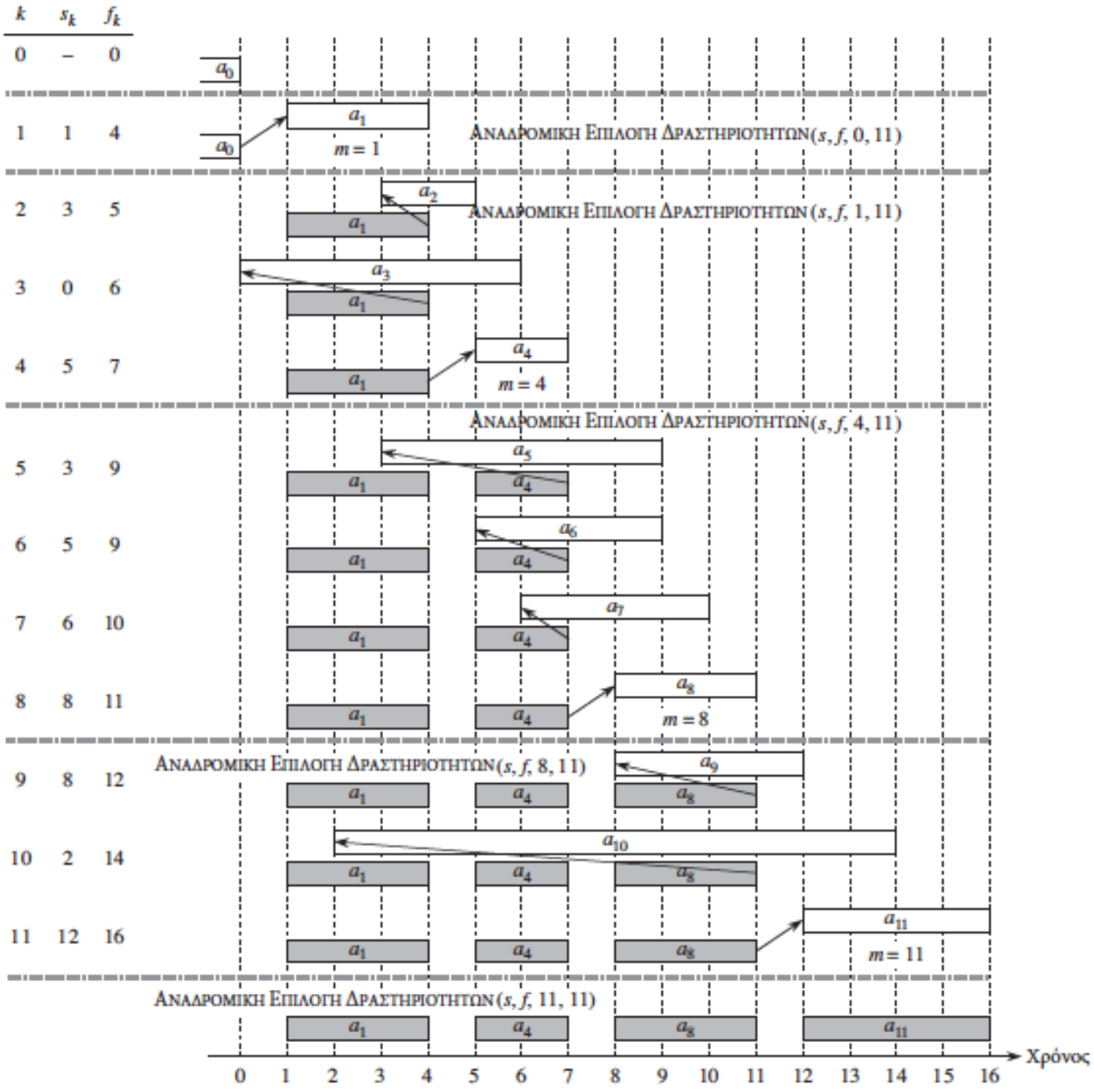
Όμως υπάρχει και γρηγορότερη λύση:

Θεώρημα: Έστω a_m η δραστηριότητα του S_{ij} με το μικρότερο χρόνο λήξης δηλ., $f_m = \min\{f_k : a_k \in S_{ij}\}$. Τότε η δραστηριότητα a_m χρησιμοποιείται σε κάποιο μεγίστου πλήθους υποσύνολο αμοιβαία συμβατών δραστηριοτήτων του S_{ij}.

Το υποπρόβλημα S_{im} κενό, επομένως απομένει μόνο ένα υποπρόβλημα S_{mj} μη κενό.

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1  m = k + 1
2  while m ≤ n and s[m] < f[k]        // find the first activity in S_k to finish
3      m = m + 1
4  if m ≤ n
5      return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6  else return ∅
```

```
GREEDY-ACTIVITY-SELECTOR(s, f)
1  n = s.length
2  A = {a_1}
3  k = 1
4  for m = 2 to n
5      if s[m] ≥ f[k]
6          A = A ∪ {a_m}
7          k = m
8  return A
```


k
s_k
f_k
0 – 0
1 1 4
2 3 5
3 0 6
4 5 7
5 3 9
6 5 9
7 6 10
8 8 11
9 8 12
10 2 14
11 12 16
a_0
a_1
a_2
a_3
a_4
a_5
a_6
a_7
a_8
a_9
a_{10}
a_{11}
$m = 1$
$m = 4$
$m = 8$
$m = 11$
ΑΝΑΔΡΟΜΙΚΗ ΕΠΙΛΟΓΗ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ$(s, f, 0, 11)$
ΑΝΑΔΡΟΜΙΚΗ ΕΠΙΛΟΓΗ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ$(s, f, 1, 11)$
ΑΝΑΔΡΟΜΙΚΗ ΕΠΙΛΟΓΗ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ$(s, f, 4, 11)$
ΑΝΑΔΡΟΜΙΚΗ ΕΠΙΛΟΓΗ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ$(s, f, 8, 11)$
ΑΝΑΔΡΟΜΙΚΗ ΕΠΙΛΟΓΗ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ$(s, f, 11, 11)$
Χρόνος
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Άπληστη στρατηγική έναντι δυναμικού προγραμματισμού

Το ***ακέραιο πρόβλημα του σακιδίου:***

- Ένας διαρρήκτης παραβιάζει ένα κατάστημα και βρίσκει n αντικείμενα.
- Το i-οστό αντικείμενο έχει αξία v_i ευρώ και βάρος w_i κιλά, όπου τα v_i και w_i είναι ακέραιοι αριθμοί.
- Ο διαρρήκτης θέλει να αποκομίσει όσο το δυνατόν πολυτιμότερη λεία
- Μπορεί να μεταφέρει στο σακίδιό του φορτίο βάρους το πολύ ίσου με W κιλά, όπου W ακέραιος αριθμός.
- Ποια αντικείμενα θα πρέπει να πάρει;

Άπληστη στρατηγική έναντι δυναμικού προγραμματισμού

Κλασματικό πρόβλημα του σακιδίου:

Έχουμε το ίδιο σκηνικό μόνο που ο διαρρήκτης δεν είναι υποχρεωμένος να επιλέξει «διαζευκτικά» για το κάθε αντικείμενο, αλλά μπορεί να πάρει και κλάσματα αντικειμένων.

Δυναμικός Προγραμματισμός για το ακέραιο πρόβλημα

- Αν *k* είναι όλα τα αντικείμενα και *V*[*k*,*w*] είναι η μέγιστη συνολική αξία που μπορούν να χωρέσουν στο σάκο χωρίς το συνολικό βάρος να υπερβεί το *w*, ισχύει η ακόλουθη αναδρομική σχέση:

$$V[k,w] = \begin{cases} V[k-1,w] & \text{if } w_k > w \\ \max\{V[k-1,w], V[k-1,w-w_k]+v_k\} & \text{αλλιώς} \end{cases}$$

Το πρόβλημα του σακιδίου: bottom-up τεχνική

i	v_i	w_i
1	\$1	1 kg
2	\$6	2 kg
3	\$18	5 kg
4	\$22	6 kg
5	\$28	7 kg

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{OPT(i-1,w),\ v_i + OPT(i-1,w-w_i)\} & \text{otherwise} \end{cases}$$

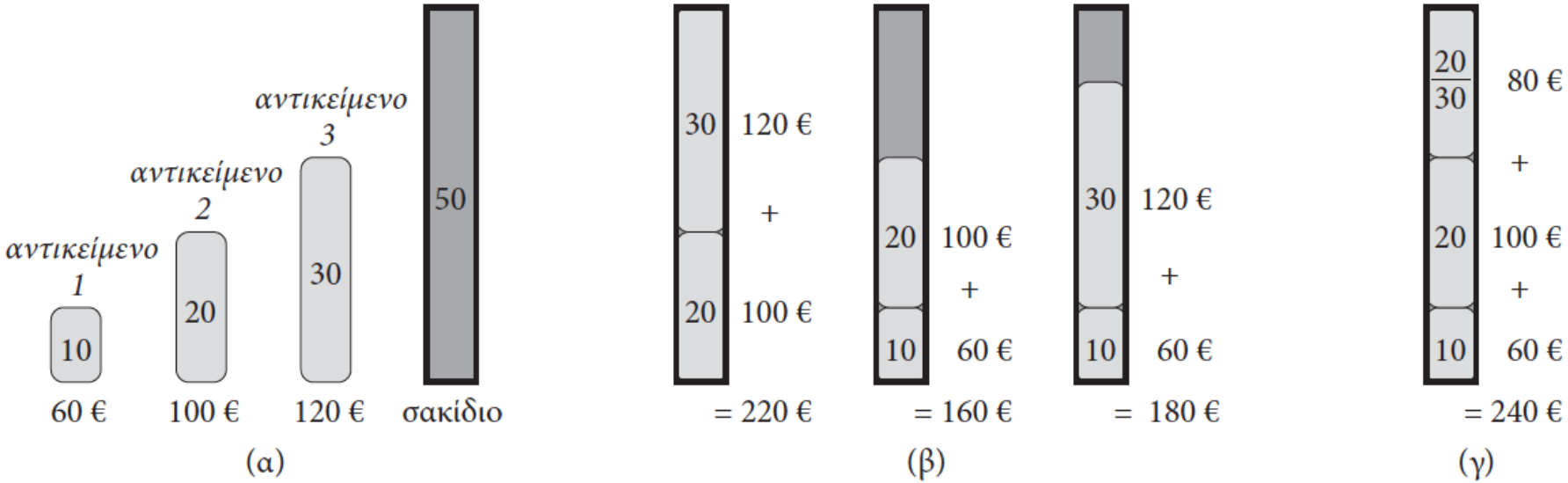
weight limit w

subset of items 1, …, i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
{ }	0	0	0	0	0	0	0	0	0	0	0	0
{ 1 }	0	1	1	1	1	1	1	1	1	1	1	1
{ 1, 2 }	0	1	6	7	7	7	7	7	7	7	7	7
{ 1, 2, 3 }	0	1	6	7	7	18	19	24	25	25	25	25
{ 1, 2, 3, 4 }	0	1	6	7	7	18	22	24	28	29	29	40
{ 1, 2, 3, 4, 5 }	0	1	6	7	7	18	22	28	29	34	35	40

OPT(i, w) = βέλτιστη αξία με τα αντικείμενα 1, …, i, με τον περιορισμό του μέγιστου βάρους w

Κλασματικό πρόβλημα σακιδίου

- Ταξινομούμε τα αντικείμενα ως προς το λόγο v_i/w_i κατά φθίνουσα.
- Με αυτή τη σειρά εισάγουμε τα αντικείμενα στο σάκο.
- Αν κάποιο δεν χωράει ολόκληρο παίρνουμε ακριβώς το κλάσμα από αυτό το αντικείμενο μέχρι να το συνολικό βάρος να φθάσει στο όριο w.
- Άπληστη λογική.

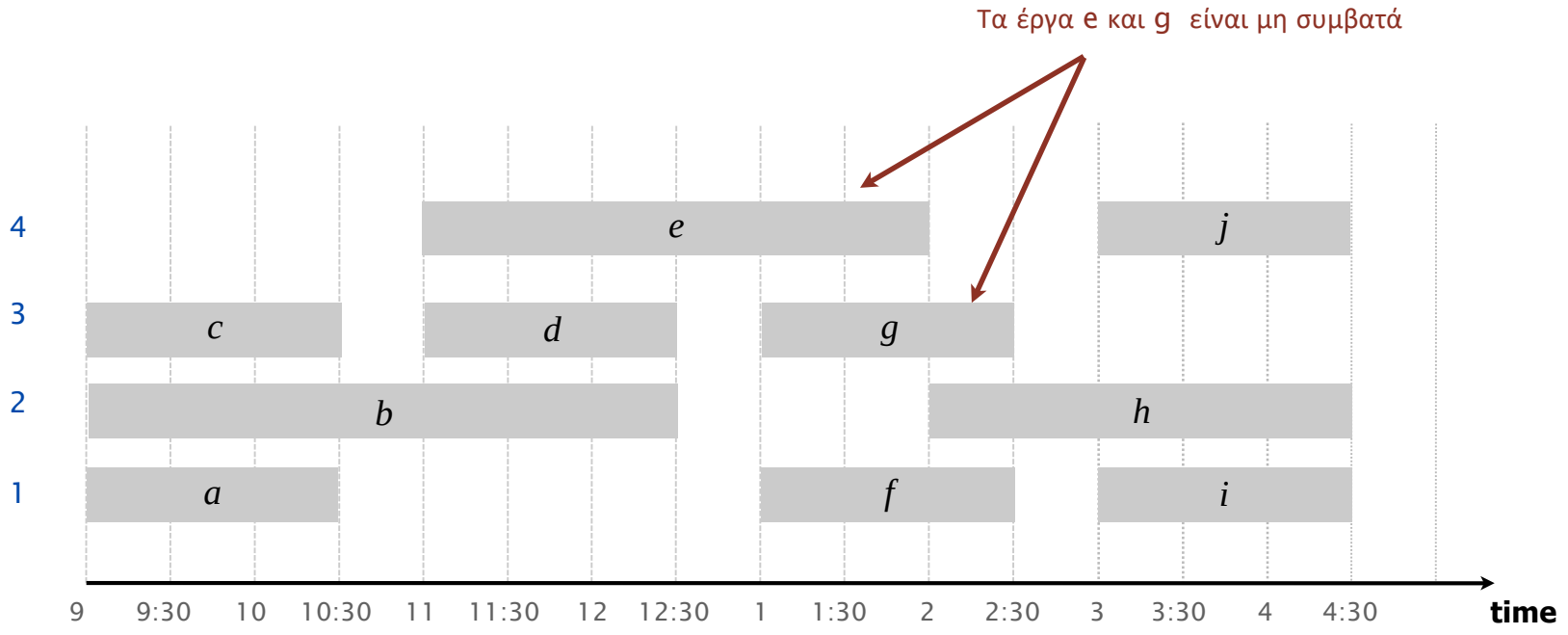


Σχήμα 16.2 Ένα παράδειγμα που δείχνει ότι η άπληστη στρατηγική δεν δίνει σωστά αποτελέσματα για το ακέραιο πρόβλημα του σακιδίου. (α) Ο διαρρήκτης θα πρέπει να επιλέξει ένα υποσύνολο των τριών εικονιζόμενων αντικειμένων το βάρος του οποίου να μην υπερβαίνει τα 50 κιλά. (β) Το βέλτιστο υποσύνολο αποτελείται από τα αντικείμενα 2 και 3. Οποιαδήποτε λύση που περιλαμβάνει το αντικείμενο 1 είναι χειρότερη της βέλτιστης, παρ' όλο που το αντικείμενο αυτό έχει τη μέγιστη αξία ανά μονάδα βάρους. (γ) Για το κλασματικό πρόβλημα του σακιδίου, η επιλογή των αντικειμένων κατά φθίνουσα σειρά αξίας ανά μονάδα βάρους δίνει μια βέλτιστη λύση.

Διαμέριση Διαστηματων (Interval partitioning)

- Η διάλεξη j ξεκινάει τη χρονική στιγμή s_j και ολοκληρώνεται τη χρονική στιγμή f_j.
- Στόχος: εύρεση του ελαχίστου πλήθους αιθουσών που επαρκούν για τον προγραμματισμό όλων των διαλέξεων έτσι ώστε να μην υπάρχουν διαλέξεις που γίνονται την ίδια ώρα στην ίδια αίθουσα

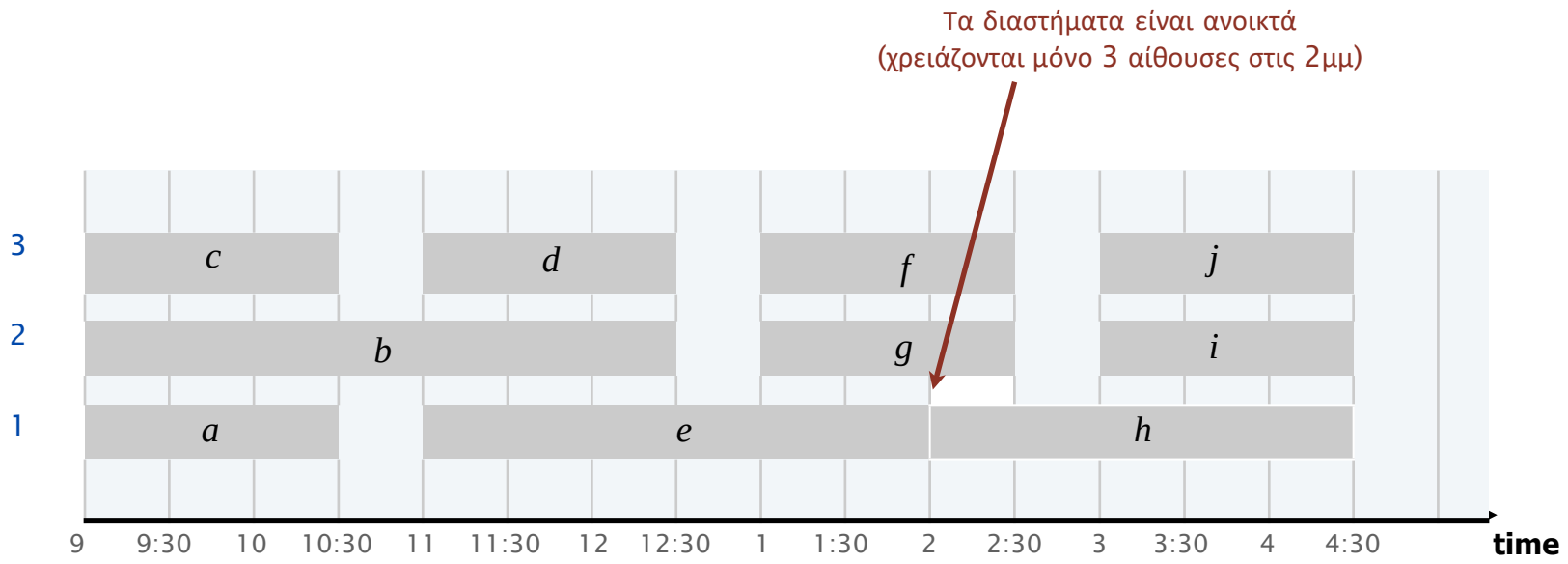
Παρ. Αυτός ο προγραμματισμός χρησιμοποιεί 4 αίθουσες για τη δρομολόγηση 10 διαλέξεων.



Διαμέριση Διαστηματων (Interval partitioning)

- Η διάλεξη j ξεκινάει τη χρονική στιγμή s_j και ολοκληρώνεται τη χρονική στιγμή f_j.
- Στόχος: εύρεση του ελαχίστου πλήθους αιθουσών που επαρκούν για τον προγραμματισμό όλων των διαλέξεων έτσι ώστε να μην υπάρχουν διαλέξεις που γίνονται την ίδια ώρα στην ίδια αίθουσα

Παρ. Αυτός ο προγραμματισμός χρησιμοποιεί 3 αίθουσες για τη δρομολόγηση 10 διαλέξεων.



Διαμέριση διαστημάτων: ο αλγόριθμος earliest-start-time-first

EARLIEST-START-TIME-FIRST (n, s_1, s_2, …, s_n , f_1, f_2, …, f_n)

SORT lectures by start times and renumber so that $s_1 \leq s_2 \leq \ldots \leq s_n$.

$d \leftarrow 0$. ← number of allocated classrooms

FOR $j = 1$ TO n

  IF (lecture j is compatible with some classroom)

    Schedule lecture j in any such classroom k.

  ELSE

    Allocate a new classroom $d + 1$.

    Schedule lecture j in classroom $d + 1$.

    $d \leftarrow d + 1$.

RETURN schedule.

Διαμέριση Διαστημάτων: ο αλγόριθμος earliest-start-time-first

Λήμμα. Ο αλγόριθμος earliest-start-time-first εκτελείται σε χρόνο $O(n \log n)$.

Απόδειξη.

- Η ταξινόμηση ως προς το χρόνο έναρξης απαιτεί χρόνο $O(n \log n)$.
- Αποθήκευση των αιθουσών σε μία ουρά προτεραιότητας (κλειδί = ο χρόνος λήξης της τελευταίας διάλεξης που έχει δρομολογηθεί στην αίθουσα.
- Για να δεσμεύεις μία νέα αίθουσα, INSERT αίθουσα στην ουρά προτεραιότητας.
- Για να δρομολογήσεις μία διάλεξη j σε μία αίθουσα k, INCREASE-KEY της αίθουσας k στο f_j.
- Για να καθορίσεις εάν η διάλεξη j είναι συμβατή με μία αίθουσα, σύγκρινε s_j με το FIND-MIN
- Συνολικό πλήθος των λειτουργιών της ουράς προτεραιότητας είναι $O(n)$. Κάθε λειτουργία χρειάζεται χρόνο $O(\log n)$.

Παρατήρηση. Αυτή η υλοποίηση διαλέγει μία αίθουσα k της οποίας η τελευταία της διάλεξη τελειώνει νωρίτερα από αυτές των άλλων αιθουσών.

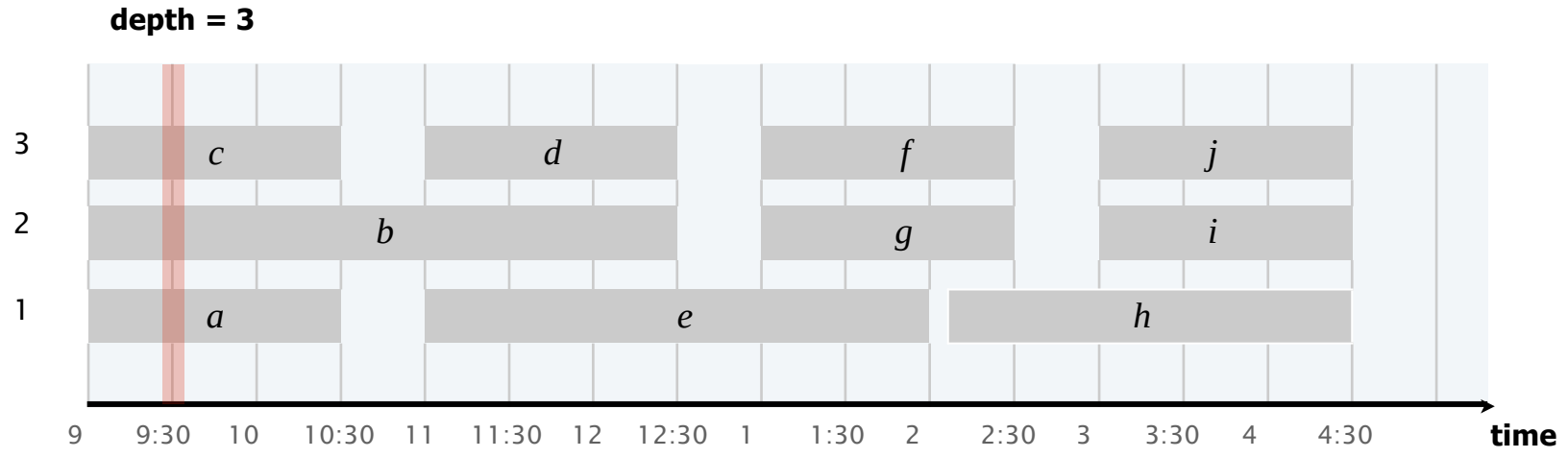
Διαμέριση Διαστημάτων: κάτω όριο της βέλτιστης λύσης

Ορισμός. Το βάθος ενός συνόλου ανοικτών διαστημάτων είναι το μέγιστο πλήθος ανοικτών διαστημάτων τα οποία παρουσιάζουν χρονική επικάλυψη.

Βασική παρατήρηση. Πλήθος των αιθουσών που χρειάζονται $\geq$ βάθος.

Ερ. Το ελάχιστο πλήθος των αιθουσών που χρειάζονται είναι πάντα ίσος με το βάθος;

Απ. Ναι. Επιπλέον, ο αλγόριθμος earliest-start-time-first βρίσκει ένα πρόγραμμα του οποίου το πλήθος των αιθουσών είναι ίσος με το βάθος.



Διαμέριση Διαστημάτων: ανάλυση του αλγορίθμου earliest-start-time-first

Παρατήρηση. Ο αλγόριθμος earliest-start-time first ποτέ δεν δρομολογεί δύο μη συμβατές διαλέξεις στην ίδια αίθουσα.

Θεώρημα. Ο αλγόριθμος Earliest-start-time-first είναι βέλτιστος.

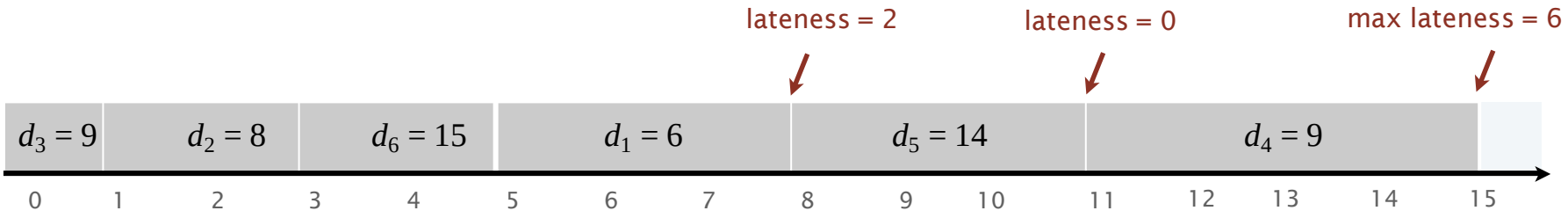
Απόδειξη.

- Έστω d το πλήθος των αιθουσών τις οποίες ο αλγόριθμος δεσμεύει.
- Η αίθουσα d δεσμεύεται επειδή χρειαζόμαστε να δρομολογήσουμε μία διάλεξη, έστω j, η οποία είναι ασύμβατη με κάθε μία από $d-1$ άλλες διαλέξεις.
- Έτσι, κάθε μία από τις d διαλέξεις ολοκληρώνεται μετά τη χρονική στιγμή s_j.
- Αφού ταξινομήσαμε ως προς το χρόνο έναρξης, κάθε μία από τις ασύμβατες διαλέξεις εκκινούν όχι αργότερα της χρονικής στιγμής s_j.
- Έτσι, έχουμε d διαλέξεις που επικαλύπτονται τη χρονική στιγμή $s_j + \varepsilon$.
- Βασική παρατήρηση $\Rightarrow$ όλα τα προγράμματα μαθημάτων χρησιμοποιούν $\geq d$ αίθουσες. ▪

Δρομολόγηση με στόχο την ελαχιστοποίηση της αργοπορίας (lateness)

- Ένας πόρος επεξεργάζεται ένα έργο τη φορά.
- Το έργο j απαιτεί t_j χρονικές μονάδες επεξεργασίας και λήγει τη χρονική στιγμή d_j.
- Αν το έργο j αρχίζει τη χρονική στιγμή s_j, ολοκληρώνεται τη χρονική στιγμή $f_j = s_j + t_j$.
- Αργοπορία : $\ell_j = \max \{ 0, f_j - d_j \}$.
- Στόχος: δρομολόγησε όλα τα έργα για την ελαχιστοποίηση της μέγιστής αργοπορίας $L = \max_j \ell_j$.

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Ελαχιστοποίηση αργοπορίας: earliest deadline first

EARLIEST-DEADLINE-FIRST $(n, t_1, t_2, \ldots, t_n, d_1, d_2, \ldots, d_n)$

SORT jobs by due times and renumber so that $d_1 \leq d_2 \leq \ldots \leq d_n$.

$t \leftarrow 0$.

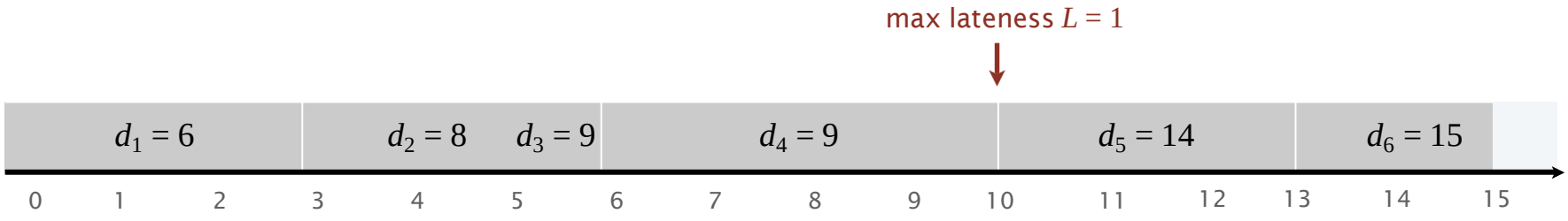
FOR $j = 1$ TO n

  Assign job j to interval $[t, t + t_j]$.

  $s_j \leftarrow t$; $f_j \leftarrow t + t_j$.

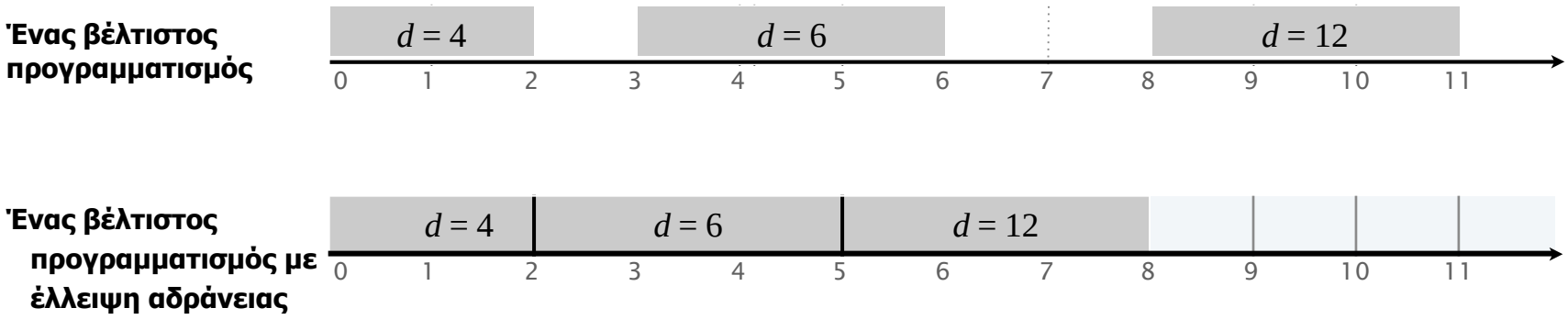
  $t \leftarrow t + t_j$.

RETURN intervals $[s_1, f_1]$, $[s_2, f_2]$, $\ldots$, $[s_n, f_n]$.



Ελαχιστοποίηση αργοπορίας: έλλειψη αδρανούς διαστήματος

Παράτηρηση 1. Υπάρχει ένας βέλτιστος προγραμματισμός με μη αδρανή διαστήματα.



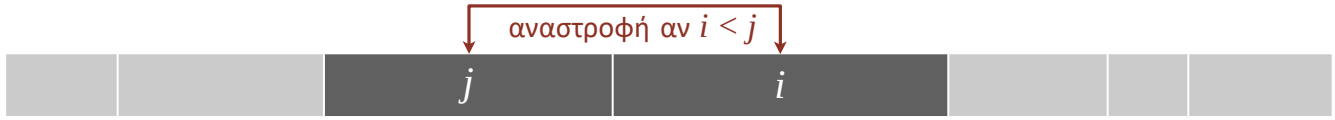
Παρατήρηση 2. Ο προγραμματισμός earliest-deadline-first δεν έχει αδρανή διαστήματα.

Ελαχιστοποίηση αργοπορίας: αναστροφές

Ορισμός. Δοθέντος ενός προγράμματος S, μία αναστροφή είναι ένα ζεύγος έργων i και j τέτοιο ώστε:

$i < j$ αλλά το έργο j δρομολογείται πριν το έργο i.

Ένα πρόγραμμα με αναστροφή



Σημείωση: υποθέτουμε ότι τα έργα αριθμούνται έτσι ώστε $d_1 \leq d_2 \leq \ldots \leq d_n$

Ορισμός 3. Η δρομολόγηση earliest-deadline-first είναι η μοναδική χρονο-δρομολόγηση χωρίς αδρανή διαστήματα και αναστροφές.

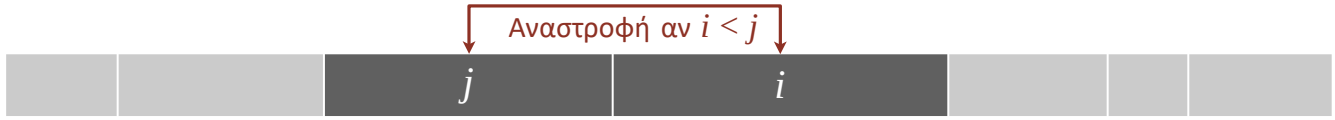
1	2	3	4	5	6	…	n

Ελαχιστοποίηση αργοπορίας: αναστροφές

Ορ. Δοθέντος ενός προγράμματος S, μία αναστροφή είναι ένα ζεύγος έργων i και j τέτοιο ώστε:

$i < j$ αλλά το j δρομολογείται πριν το i.

Ένα πρόγραμμα με αναστροφή



Σημείωση: υποθέτουμε ότι τα έργα αριθμούνται έτσι ώστε $d_1 \leq d_2 \leq \ldots \leq d_n$

Παρατήρηση 4. Αν ένα πρόγραμμα χωρίς αδρανή διαστήματα έχει μία αναστροφή, τότε έχει μία γειτονική αναστροφή.

Δύο αναστραμμένα έργα δρομολογούνται διαδοχικά

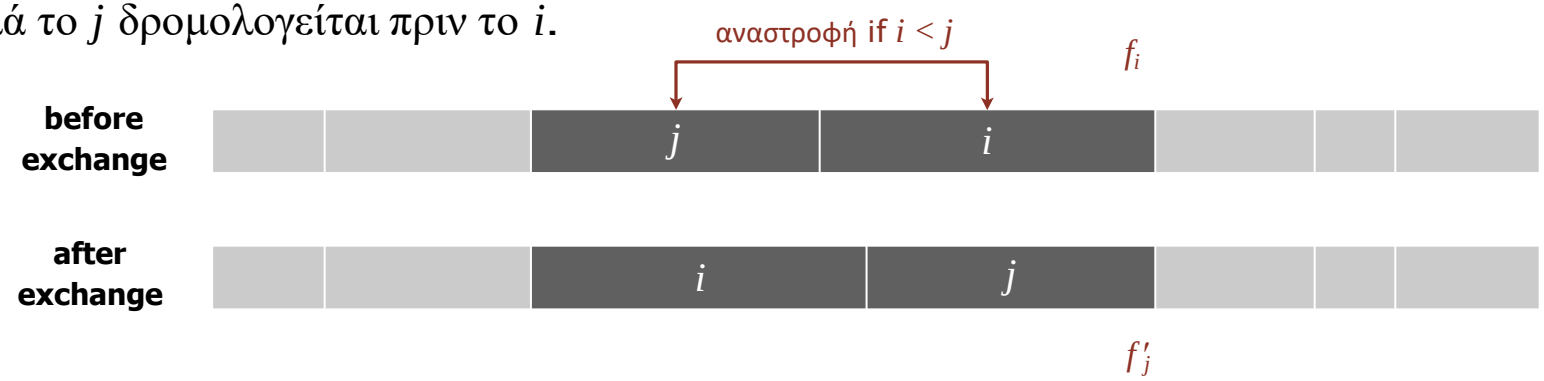
Απόδειξη.

- Έστω $i– j$ η πλησιέστερη αναστροφή.
- Έστω k το έργο αμέσως δεξιά του j.
- Περίπτωση 1. [$j > k$] Τότε $j–k$ είναι μία γειτονική αναστροφή.
- Περίπτωση 2. [$j < k$] Τότε $i–k$ είναι πλησιέστερη αναστροφή αφού $i < j < k$.

Minimizing lateness: inversions

Ορ. Δοθέντος ενός προγράμματος S, μία αναστροφή είναι ένα ζεύγος έργων i και j τέτοιο ώστε:

$i < j$ αλλά το j δρομολογείται πριν το i.



Βασικός ισχυρισμός. Ανταλλάσσοντας δύο γειτονικά, αναστραμμένα έργα i και j μειώνει το πλήθος των αναστροφών κατά 1 και δεν προκαλεί αύξηση στη μέγιστη αργοπορία.

Έστω ℓ η αργοπορία πριν την ανταλλαγή, και ℓ' η αργοπορία μετά την ενέργεια αυτή.

$\ell'_k = \ell_k$ for all $k \neq i, j$.

$\ell'_i \leq \ell_i$.

Αν το έργο j είναι αργοπορημένο, $\ell'_j = f'_j - d_j$ ← ορισμός

$= f_i - d_j$ ← j τώρα τελειώνει τη χρονική στιγμή f_i

$\leq f_i - d_i$ ← $i < j \Rightarrow d_i \leq d_j$

$\leq \ell_i$. ← ορισμός

Ελαχιστοποίηση αργοπορίας: ανάλυση του αλγορίθμου earliest-deadline-first algorithm

Θεώρημα. Η χρόνο-δρομολόγηση earliest-deadline-first S είναι βέλτιστη.

Απ. [με εις Άτοπο Απαγωγή]

Έστω S^* μία βέλτιστη χρόνο-δρομολόγηση με το λιγότερο πλήθος αναστροφών.

- S^* δεν έχει διαστήματα αδράνειας. ← Παρατήρηση 1
- Περίπτωση 1. [S^* δεν έχει αναστροφές] Τοτε $S = S^*$. ← Παρατήρηση 3
- Περίπτωση 2. [S^* έχει αναστροφή]
  - Έστω i–j μία γειτονική αναστροφή ← Παρατήρηση 4
  - Η ανταλλαγή των έργων i και j μειώνει το πλήθος των αναστροφών κατά 1 χωρίς αύξηση της μέγιστης αργοπορίας ← key claim
  - Άτοπο γιατί η S^* έχει το ελάχιστο πλήθος αναστροφών.