641: Εισαγωγή στη Θεωρία και Ανάλυση Αλγορίθμων

Χάρης Παπαδόπουλος

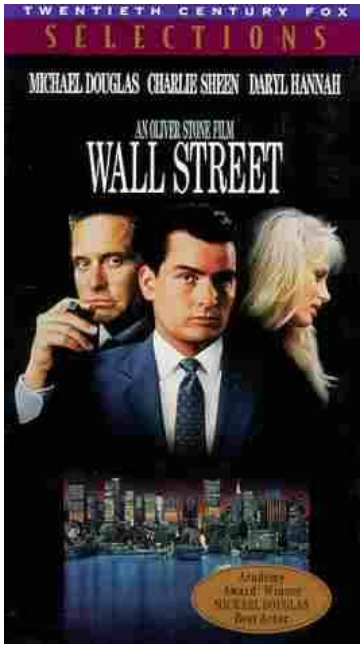
Υλη του μαθήματος

- Βασικά στοιχεία σχεδίασης & ανάλυσης αλγορίθμων
- Ανάλυση αλγορίθμων, αποδοτικότητα, ασυμπτωτικός συμβολισμός
- Συνηθισμένοι χρόνοι εκτέλεσης και βασικές δομές δεδομένων
  - πίνακες, λίστες, στοίβες, ουρές
- Ευσταθές ταίριασμα, ορθότητα, σωρός και ουρά προτεραιότητας
- Μέθοδος «Διαίρει και Βασίλευε»
  - Εφαρμογές σε ταξινόμηση στοιχείων
  - Επίλυση αναδρομικών σχέσεων
- Γραφήματα και αλγόριθμοι γραφημάτων
  - Διάτρεξη γραφημάτων (BFS, DFS)
  - Συνεκτικότητα
  - Τοπολογική διάταξη
- Μέθοδοι «Απληστείας» και «Δυναμικού Προγραμματισμού»
  - Ελάχιστα σκελετικά δένδρα (αλγόριθμος Prim, αλγόριθμος Kruskal)
  - Συντομότερες διαδρομές (αλγόριθμος Dijkstra, Ροή δικτύου)
  - Χρονοπρογραμματισμός
- Επιλεγμένα θέματα
  - Υπολογιστική πολυπλοκότητα, NP-πληρότητα

Η μέθοδος της απληστίας

Greed is good. Greed is right. Greed works. Greed clarifies, cuts through, and captures the essence of the evolutionary spirit.

- *Gordon Gecko (Michael Douglas)*



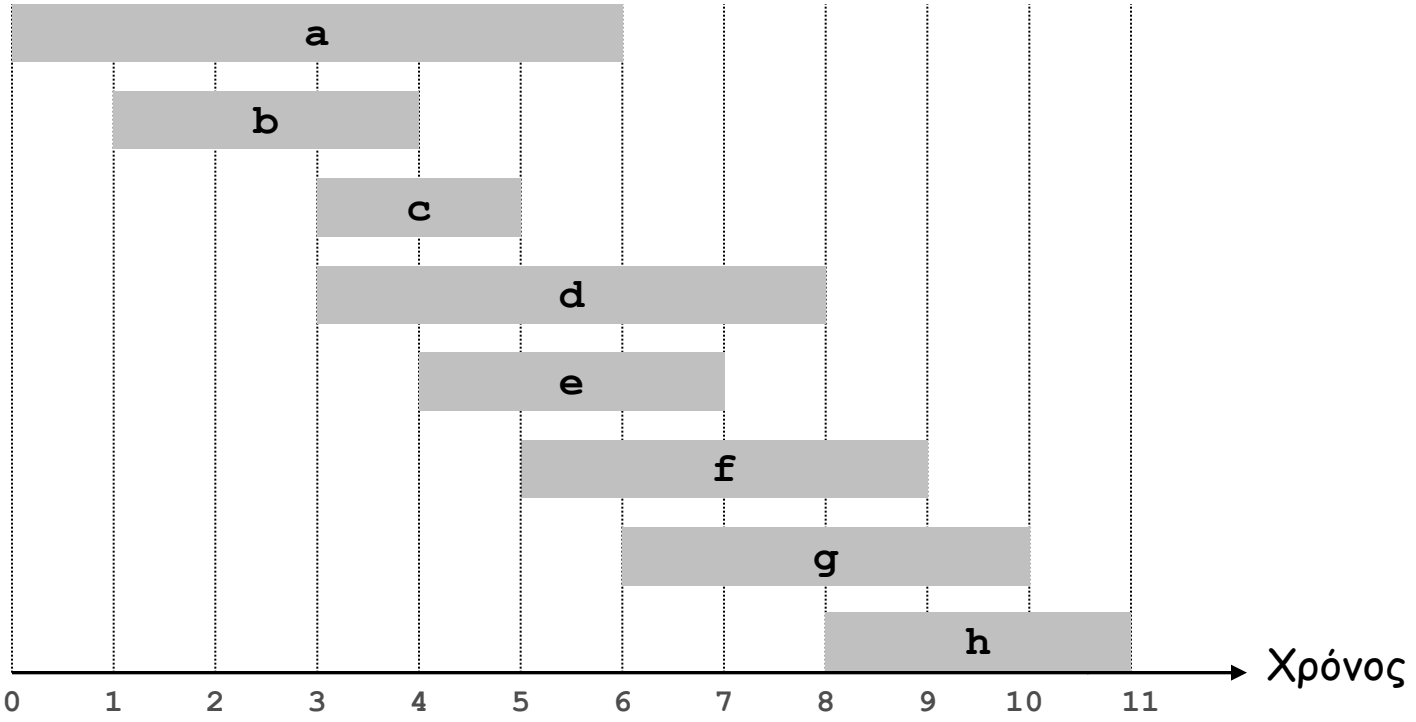
Απληστία: Δόμηση της λύσης σταδιακά, βελτιστοποιώντας κάποιο τοπικό κριτήριο σε κάθε βήμα

Χρονοπρογραμματισμός Διαστημάτων

Χρονοπρογραμματισμός Διαστημάτων

Χρονοπρογραμματισμός.

- Η εργασία j ξεκινάει τη στιγμή s_j και τελειώνει τη στιγμή f_j.
- Δυο εργασίες είναι συμβατές αν δεν επικαλύπτονται.
- Στόχος: εύρεση του μεγαλύτερου υποσυνόλου από συμβατές εργασίες.



Χρονοπρογραμματισμός Διαστημάτων: Άπληστοι Αλγόριθμοι

Άπληστο Πρότυπο. Θεωρήστε τις εργασίες ως προς κάποια διάταξη.
Επιλέξτε κάποια εργασία με δεδομένο ότι είναι συμβατή με αυτές που έχουν ήδη επιλεγεί.

- [Μικρότερος χρόνος έναρξης] εργασίες $\rightarrow$ σε αύξουσα τάξη ως προς s_j.
- [Μικρότερος χρόνος λήξης] εργασίες $\rightarrow$ σε αύξουσα τάξη ως προς f_j.
- [Μικρότερο χρον. διάστημα] εργασίες $\rightarrow$ σε αύξουσα τάξη ως προς $f_j - s_j$.
- [Ελάχιστες διενέξεις] Για κάθε εργασία j, μέτρησε το πλήθος των μη-συμβατών εργασιών c_j. Χρονοπρογραμμάτισε κατά αύξουσα τάξη ως προς c_j.

Χρονοπρογραμματισμός Διαστημάτων: Άπληστοι Αλγόριθμοι

Άπληστο Πρότυπο. Θεωρήστε τις εργασίες ως προς κάποια διάταξη.
Επιλέξτε κάποια εργασία με δεδομένο ότι είναι συμβατή με αυτές που έχουν ήδη επιλεγεί.

αντιπαράδειγμα για μικρότερο χρόνο έναρξης

αντιπαράδειγμα για μικρότερο χρονικό διάστημα

αντιπαράδειγμα για ελάχιστες διενέξεις

Χρονοπρογραμματισμός Διαστημάτων: Άπληστος Αλγόριθμος

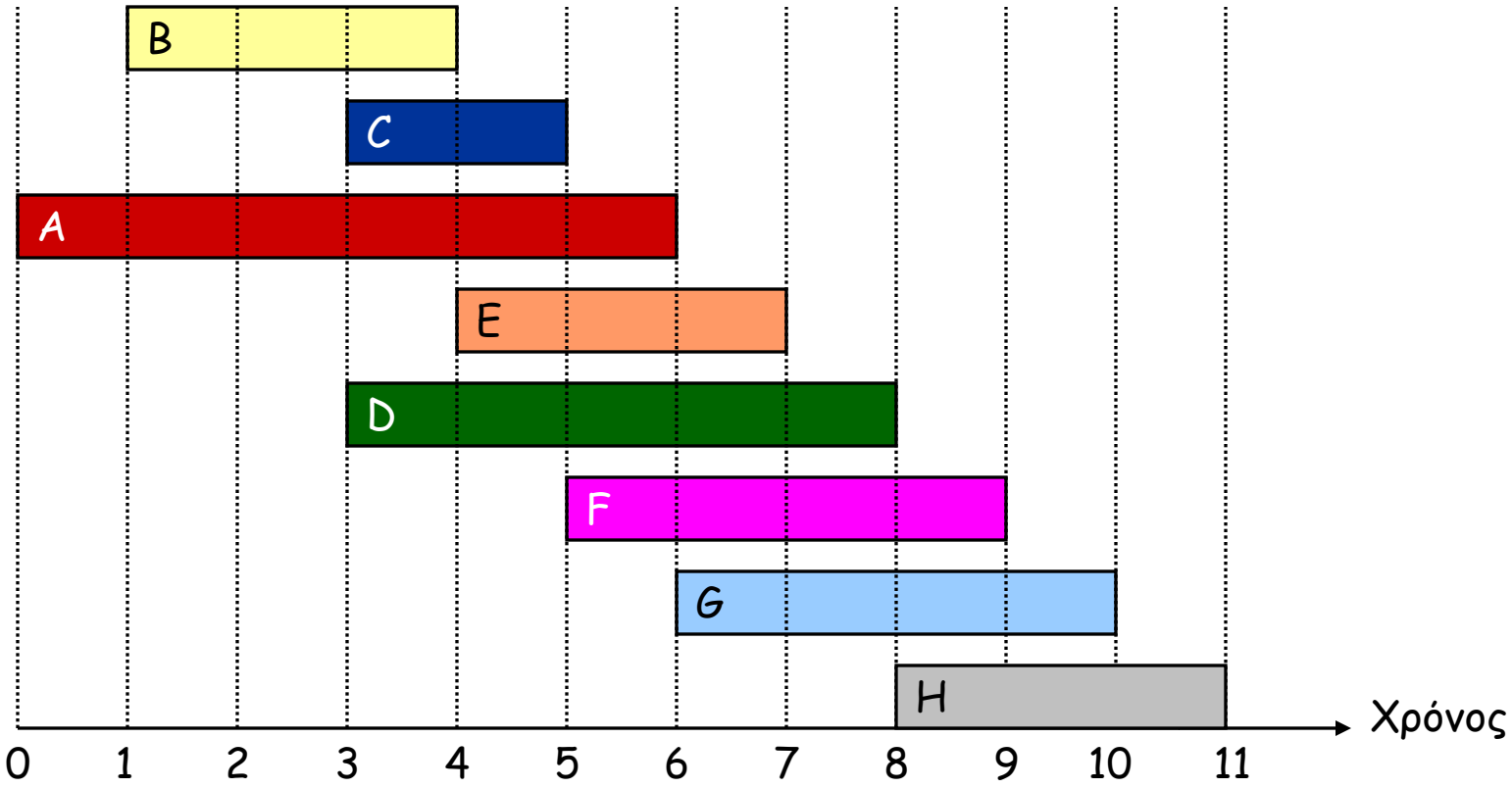
Άπληστος Αλγόριθμος. Θεωρούμε τις εργασίες κατά αύξουσα τάξη ως προς το χρόνο λήξης. Επιλέγουμε κάποια εργασία εφόσον αυτή είναι συμβατή με τις υπόλοιπες που έχουμε ήδη επιλέξει.

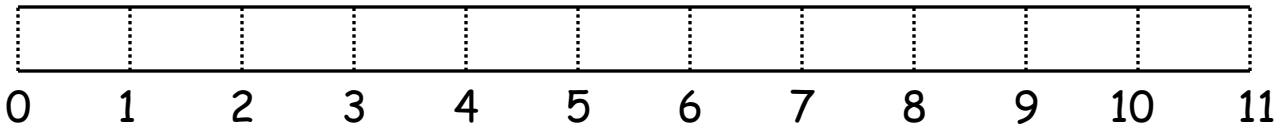
```
Ταξινόμησε τις εργασίες ως προς χρόνο λήξης έτσι ώστε
                f1 ≤ f2 ≤ ... ≤ fn.

   σύνολο εργασιών που θα επιλέγει
A ← ∅
for j = 1 to n {
   if (η εργασία j είναι συμβατή με το A)
      A ← A ∪ {j}
}
return A
```

Υλοποίηση. $O(n \log n)$.

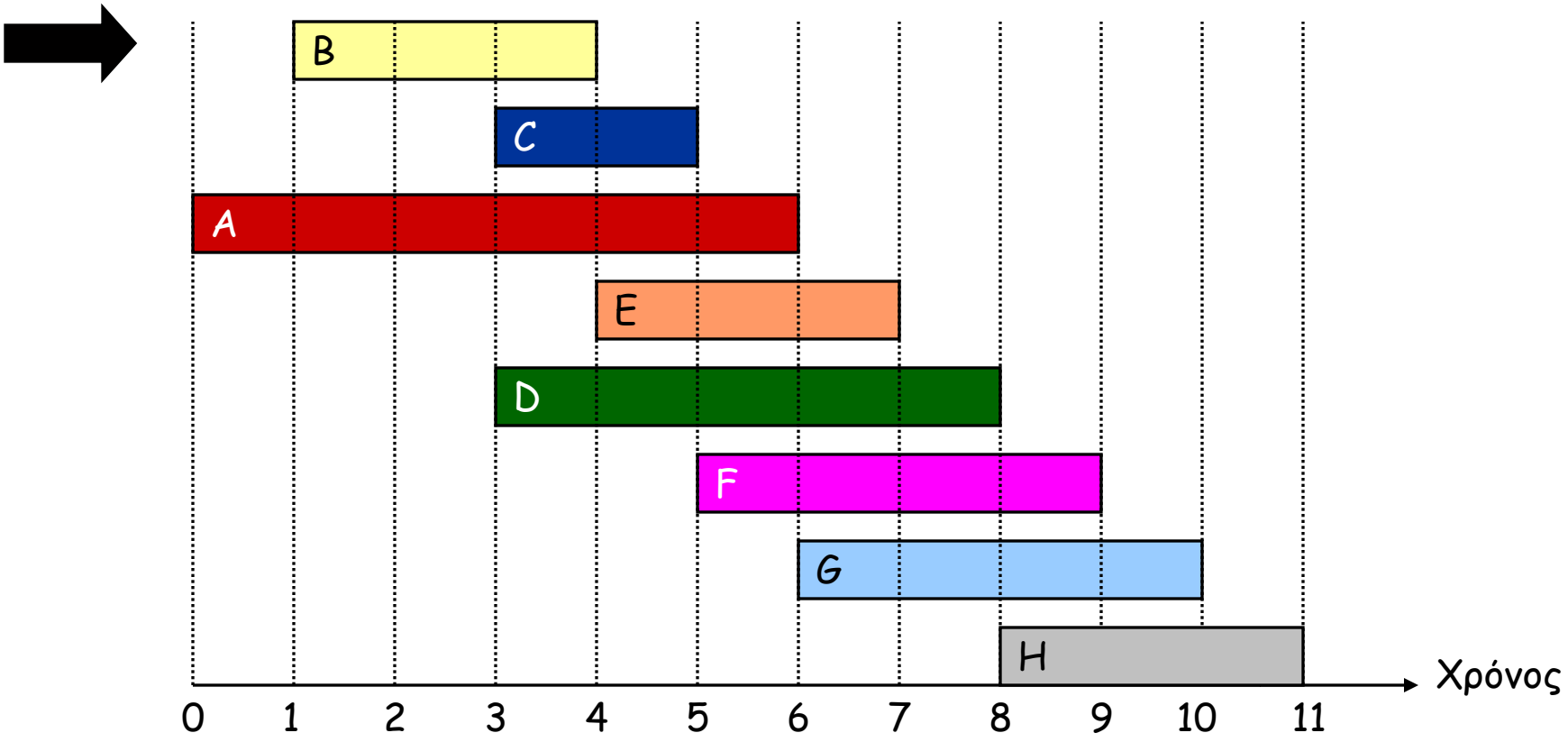
- Θυμάται την εργασία j* που τοποθετήθηκε τελευταία στο A.
- Η εργασία j είναι συμβατή με (κάθε εργασία του) A αν $s_j \geq f_{j^*}$.

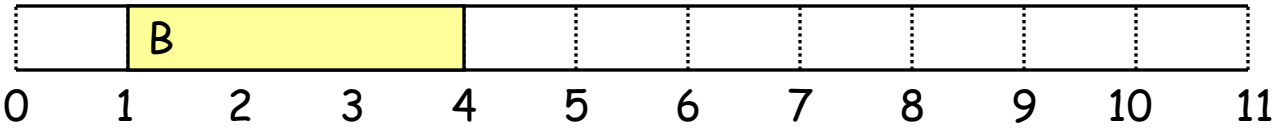
Χρονοπρογραμματισμός Διαστημάτων: Παράδειγμα



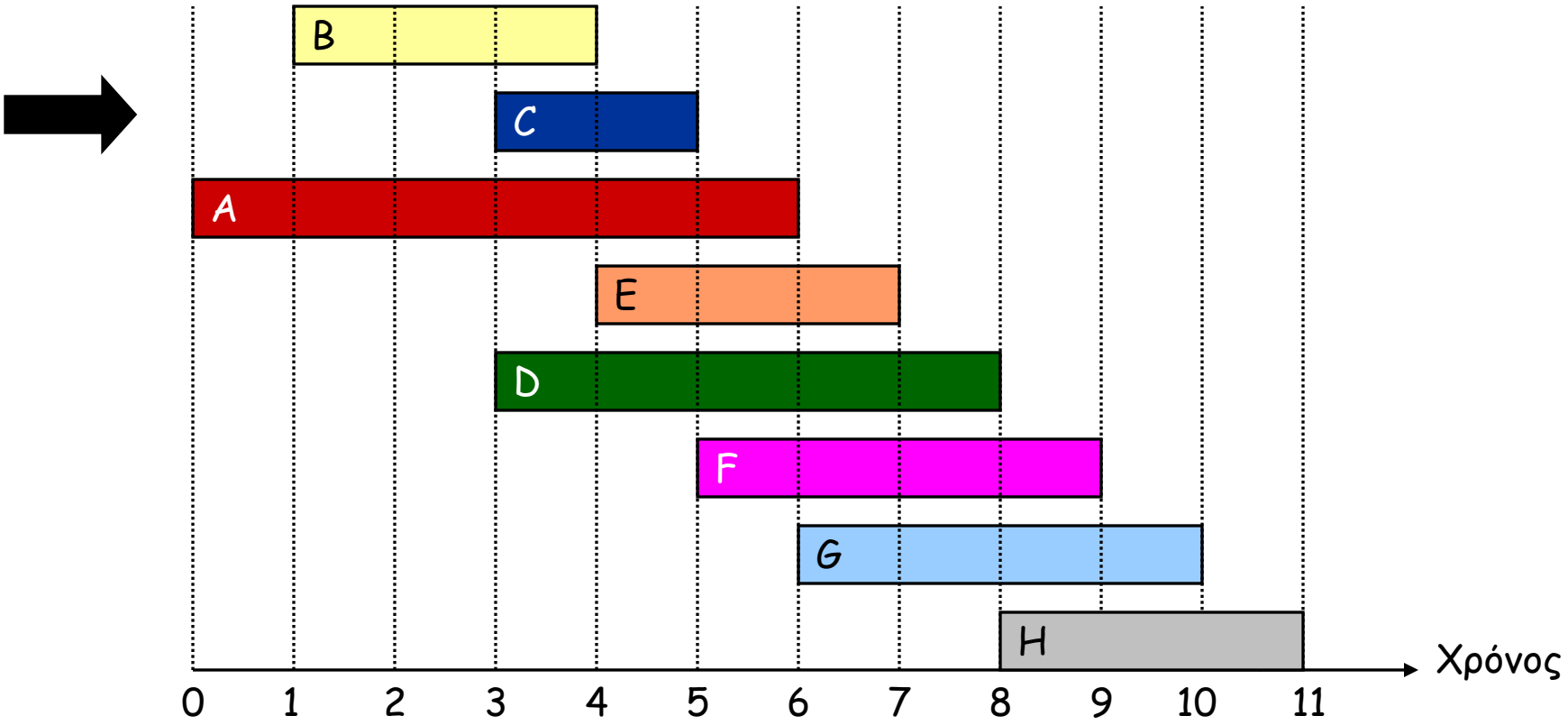


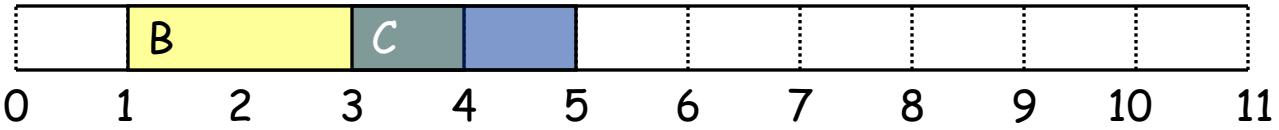
Χρονοπρογραμματισμός Διαστημάτων: Παράδειγμα



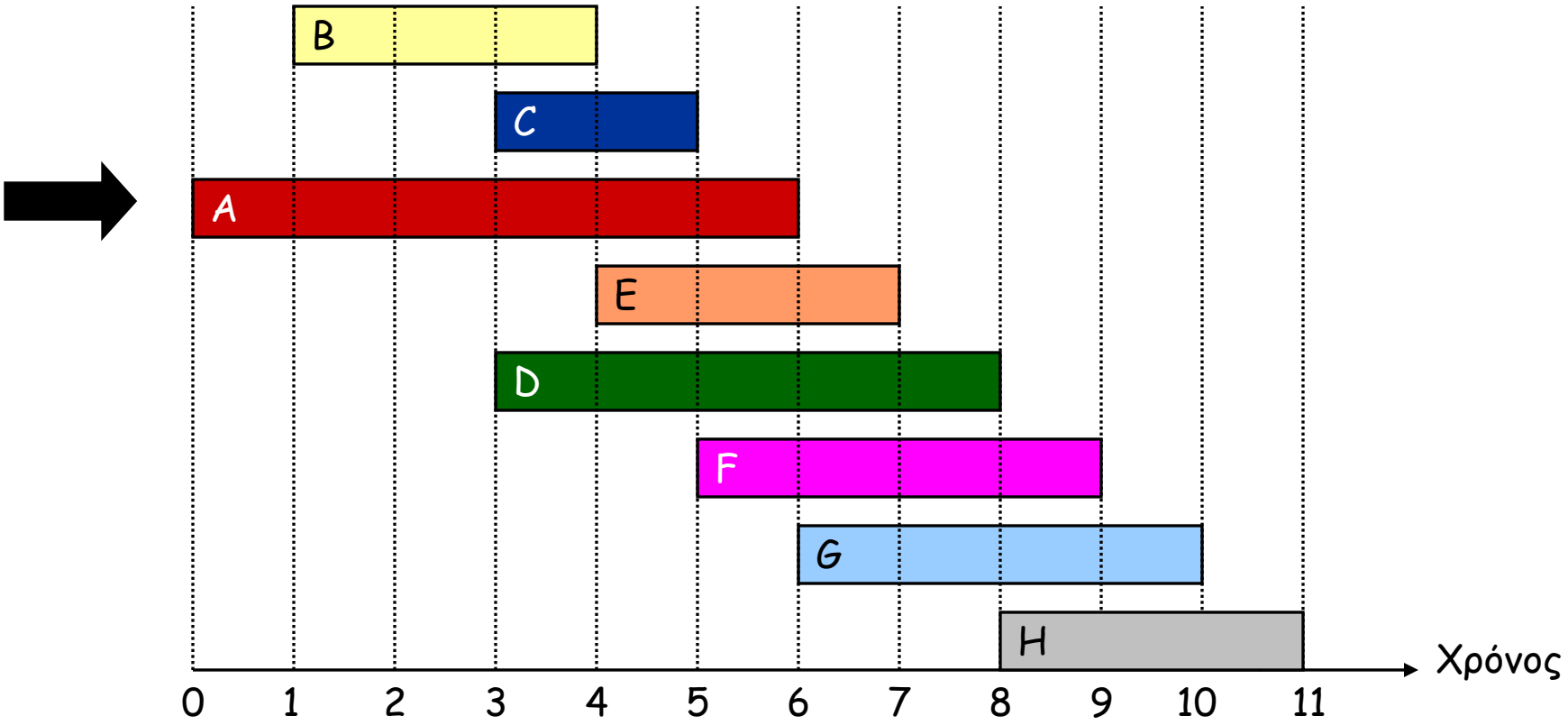


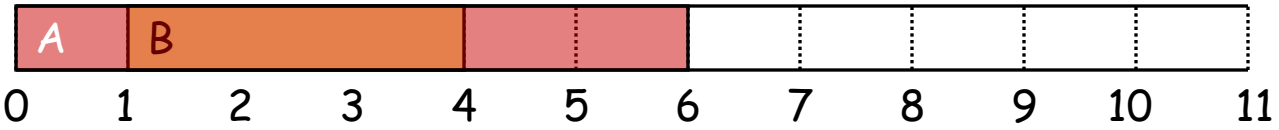
Χρονοπρογραμματισμός Διαστημάτων: Παράδειγμα



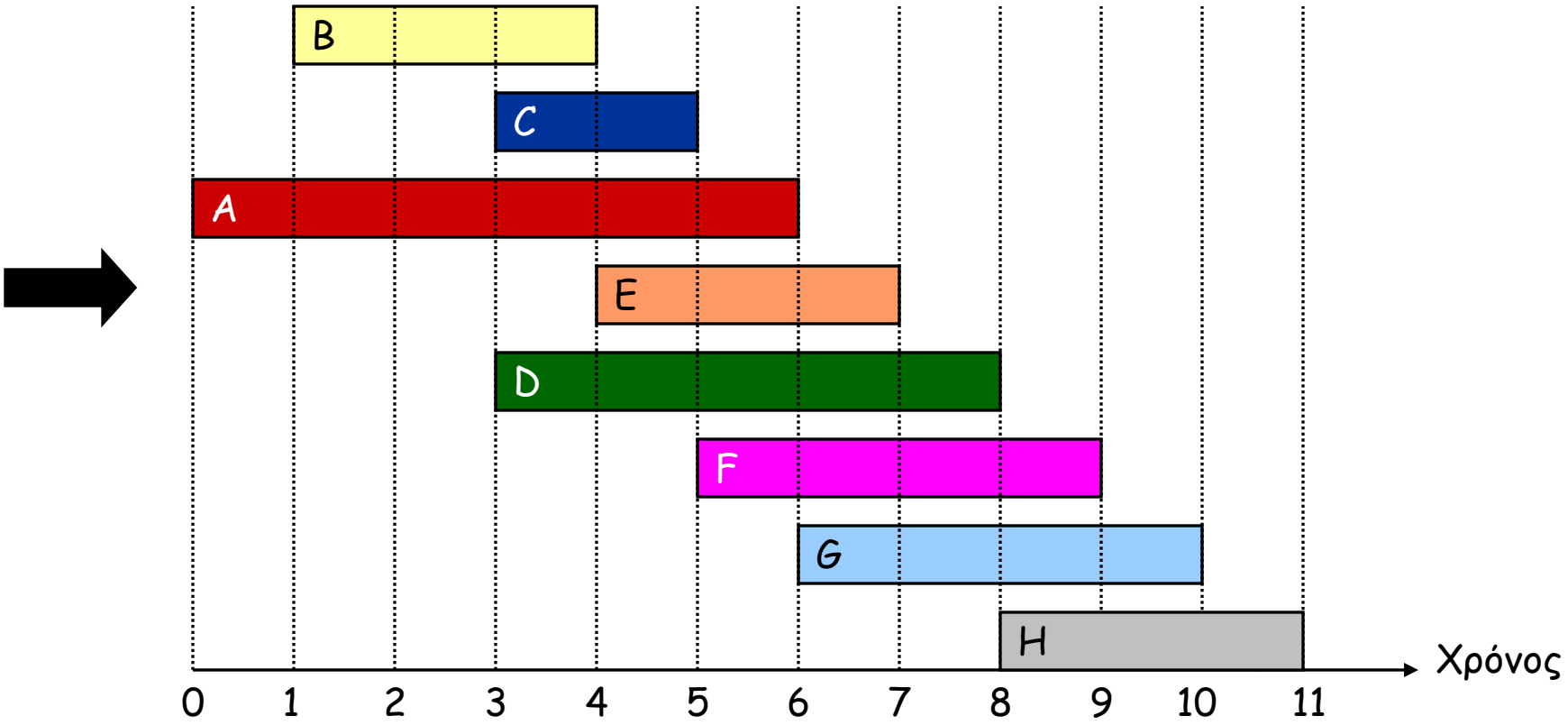


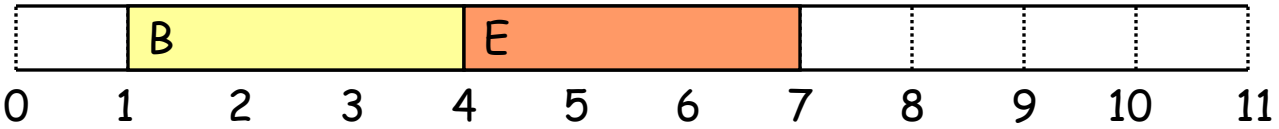
Χρονοπρογραμματισμός Διαστημάτων: Παράδειγμα



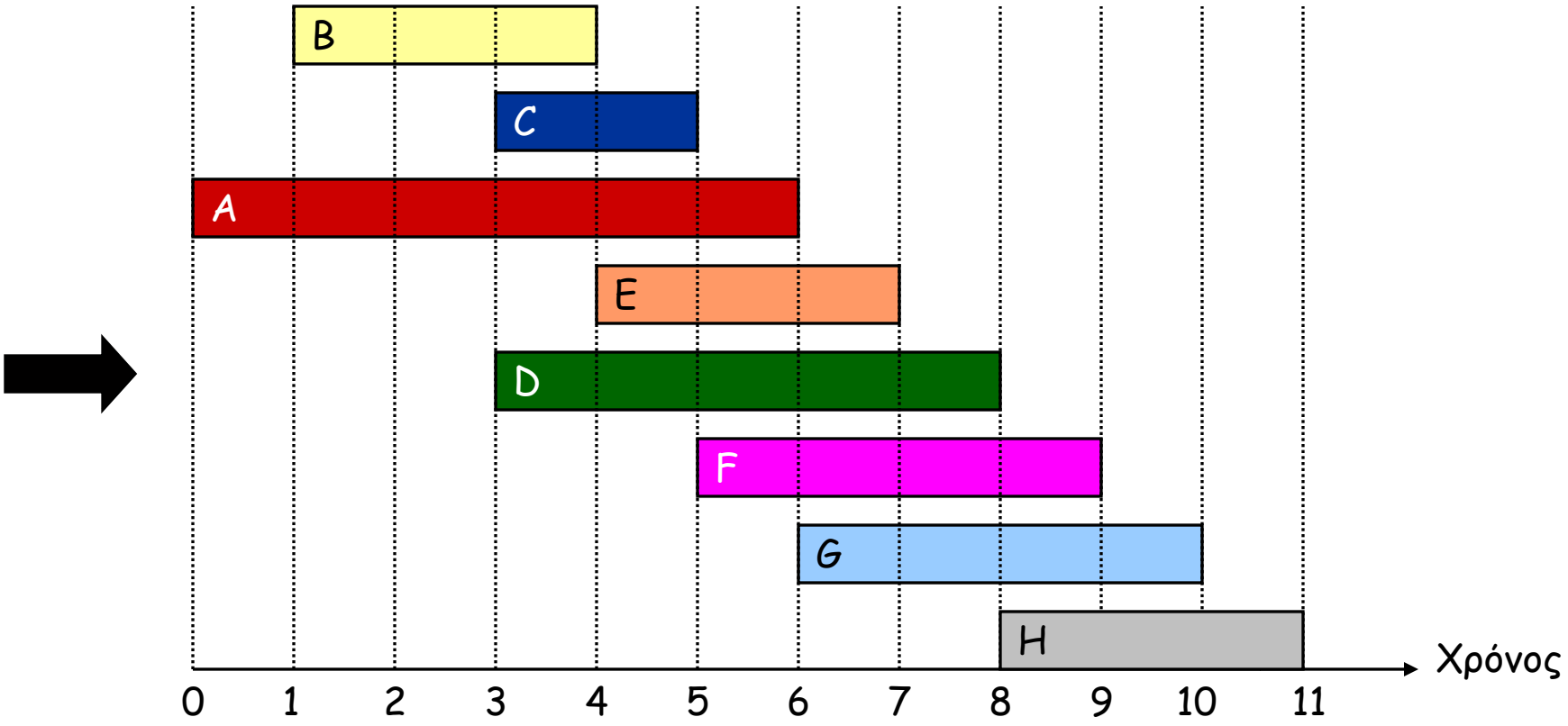


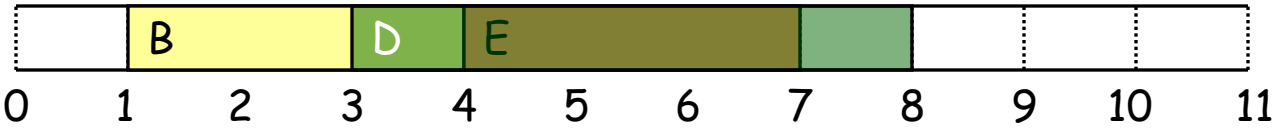
Χρονοπρογραμματισμός Διαστημάτων: Παράδειγμα



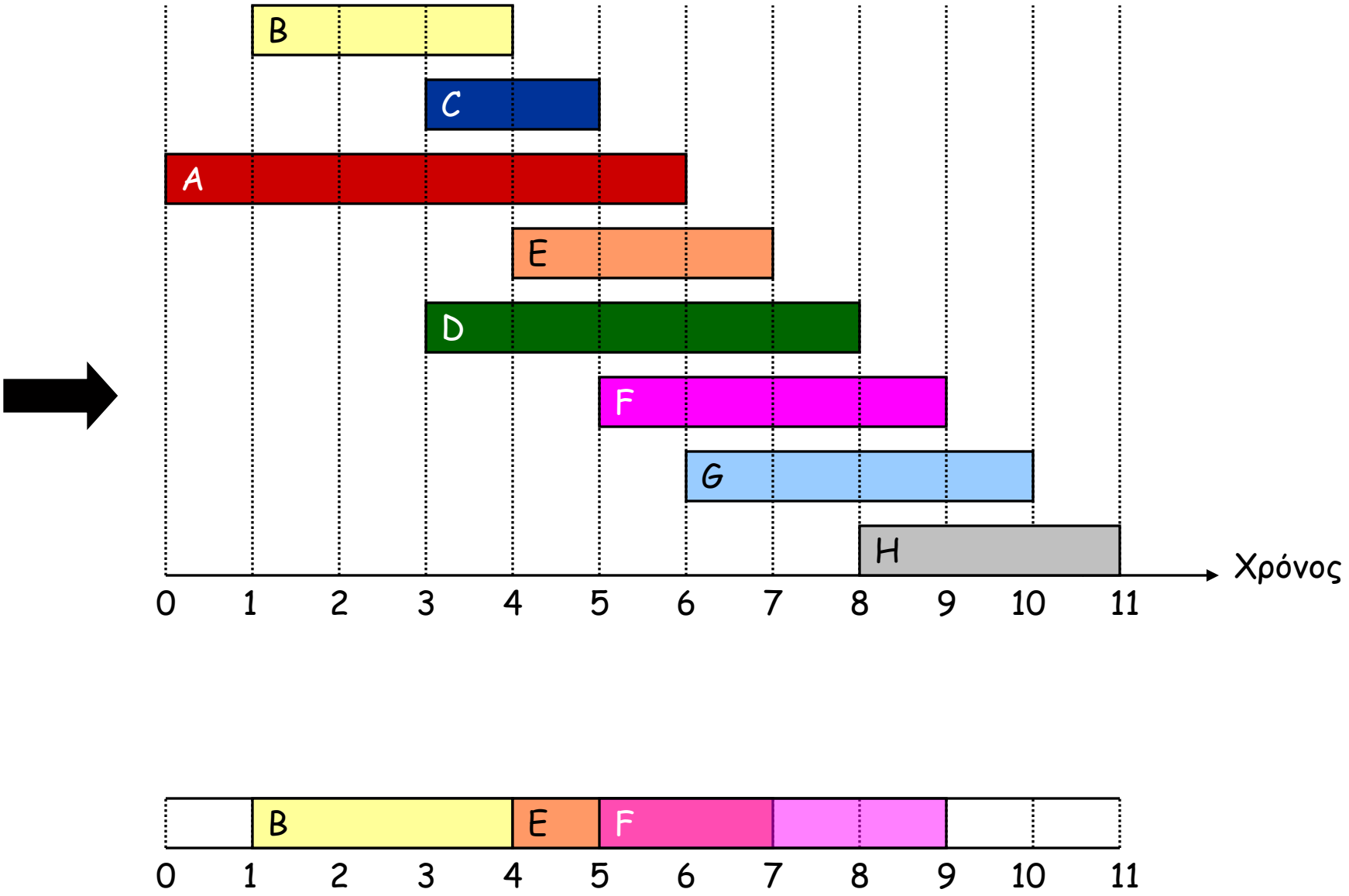


Χρονοπρογραμματισμός Διαστημάτων: Παράδειγμα

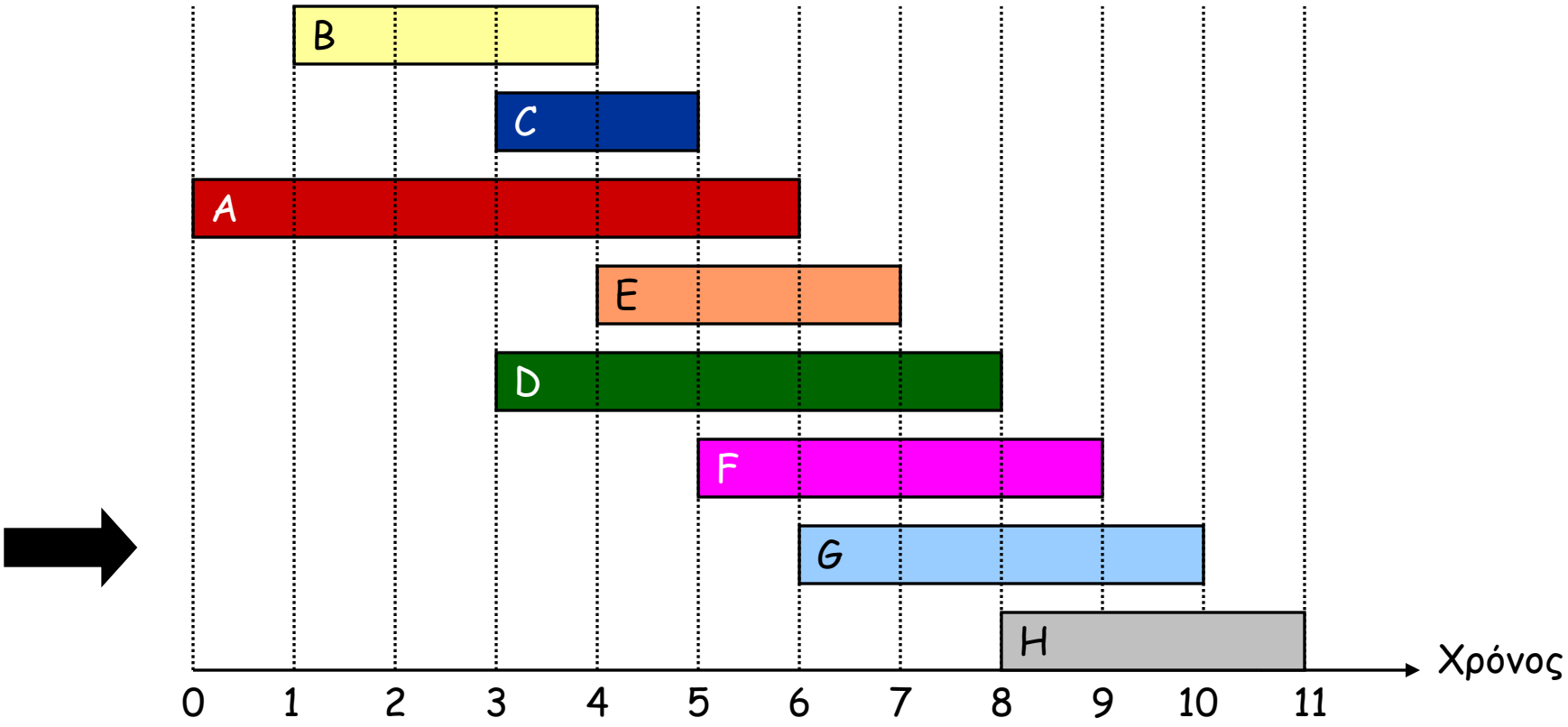


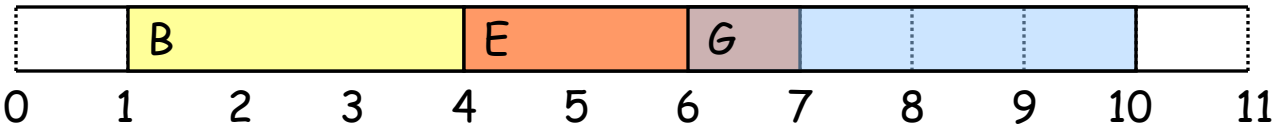


Χρονοπρογραμματισμός Διαστημάτων: Παράδειγμα

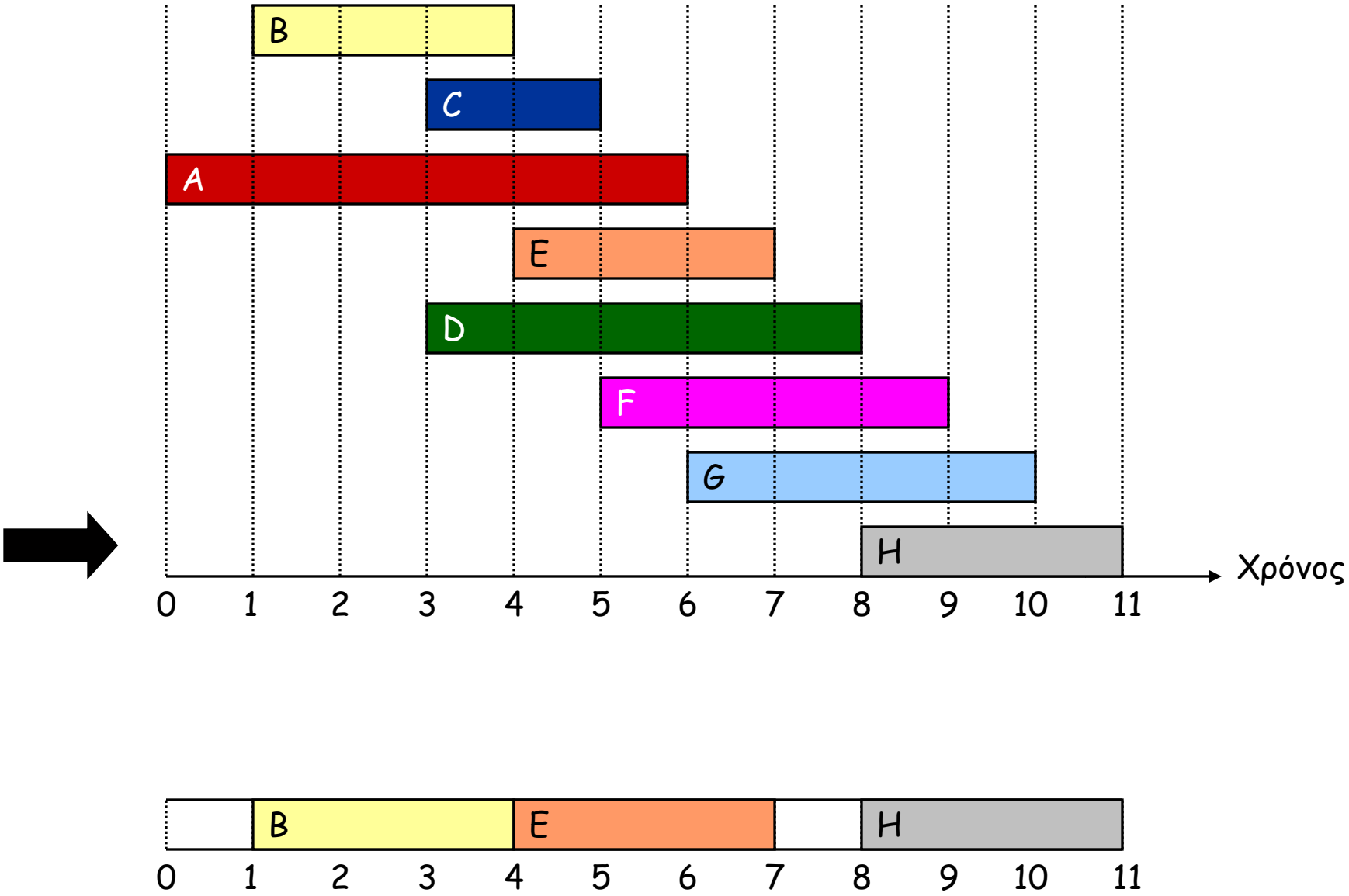


Χρονοπρογραμματισμός Διαστημάτων: Παράδειγμα





Χρονοπρογραμματισμός Διαστημάτων: Παράδειγμα



Χρονοπρογραμματισμός Διαστημάτων: Άπληστος Αλγόριθμος

Άπληστος Αλγόριθμος. Θεωρούμε τις εργασίες κατά αύξουσα τάξη ως προς το χρόνο λήξης. Επιλέγουμε κάποια εργασία εφόσον αυτή είναι συμβατή με τις υπόλοιπες που έχουμε ήδη επιλέξει.

Ταξινόμησε τις εργασίες ως προς χρόνο λήξης έτσι ώστε $f_1 \le f_2 \le \ldots \le f_n$.

σύνολο εργασιών που θα επιλέγει

```
A ← ∅
for j = 1 to n {
   if (η εργασία j είναι συμβατή με το A)
      A ← A ∪ {j}
}
return A
```

Υλοποίηση. $O(n \log n)$.

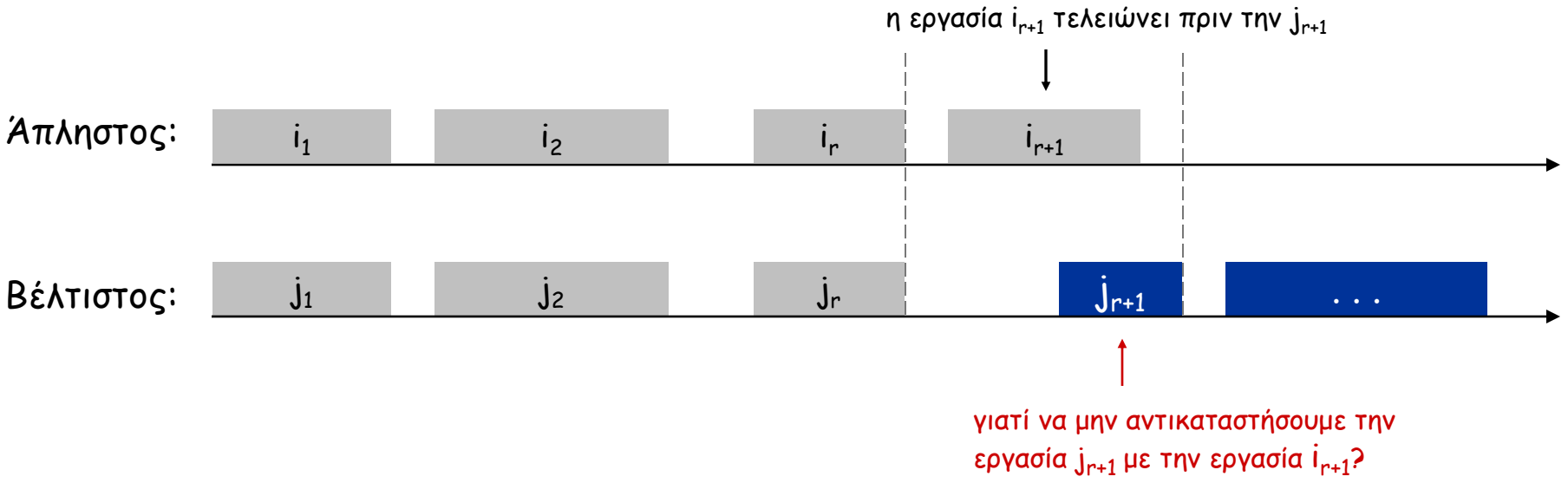
- Θυμάται την εργασία j* που τοποθετήθηκε τελευταία στο A.
- Η εργασία j είναι συμβατή με (κάθε εργασία του) A αν $s_j \ge f_{j^*}$.

Χρονοπρογραμματισμός Διαστημάτων: Ορθότητα

Θεώρημα. Ο άπληστος αλγόριθμος είναι βέλτιστος.

Απόδειξη. (με άτοπο)

- Έστω ότι ο άπληστος δεν είναι βέλτιστος.
- Έστω $i_1, i_2, \ldots i_k$ οι εργασίες που επιλέγει ο άπληστος.
- Έστω $j_1, j_2, \ldots j_m$ οι εργασίες σε μια βέλτιστη λύση τέτοια ώστε $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ για τη μέγιστη δυνατή τιμή του r.

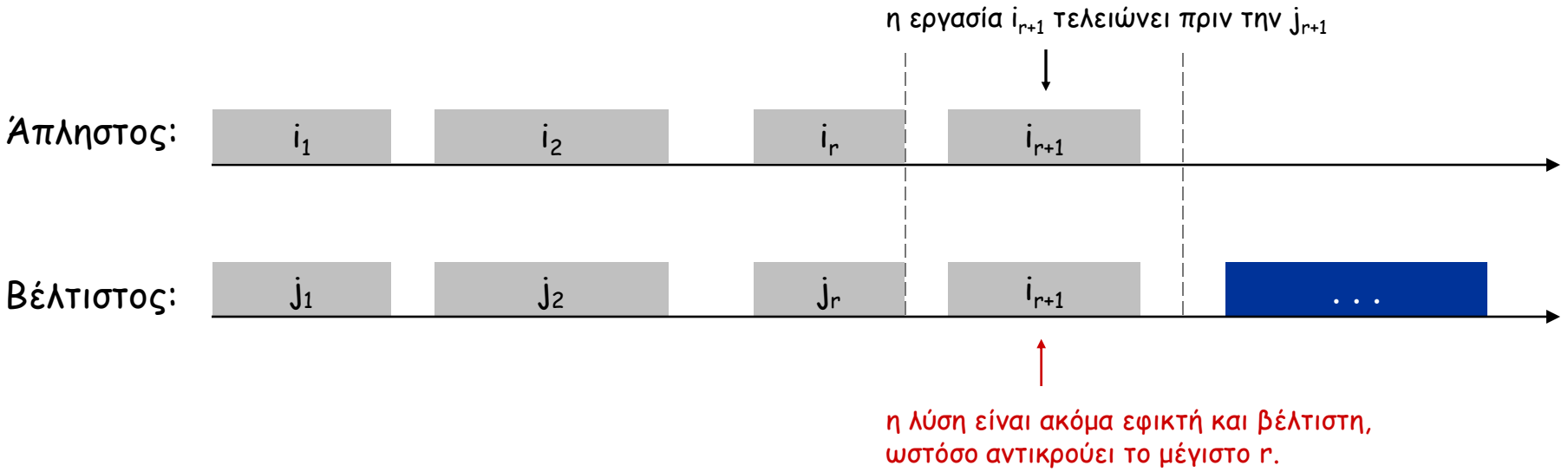


Χρονοπρογραμματισμός Διαστημάτων: Ορθότητα

Θεώρημα. Ο άπληστος αλγόριθμος είναι βέλτιστος.

Απόδειξη. (με άτοπο)

- Έστω ότι ο άπληστος δεν είναι βέλτιστος.
- Έστω $i_1, i_2, \ldots i_k$ οι εργασίες που επιλέγει ο άπληστος.
- Έστω $j_1, j_2, \ldots j_m$ οι εργασίες σε μια βέλτιστη λύση τέτοια ώστε $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ για τη μέγιστη δυνατή τιμή του r.

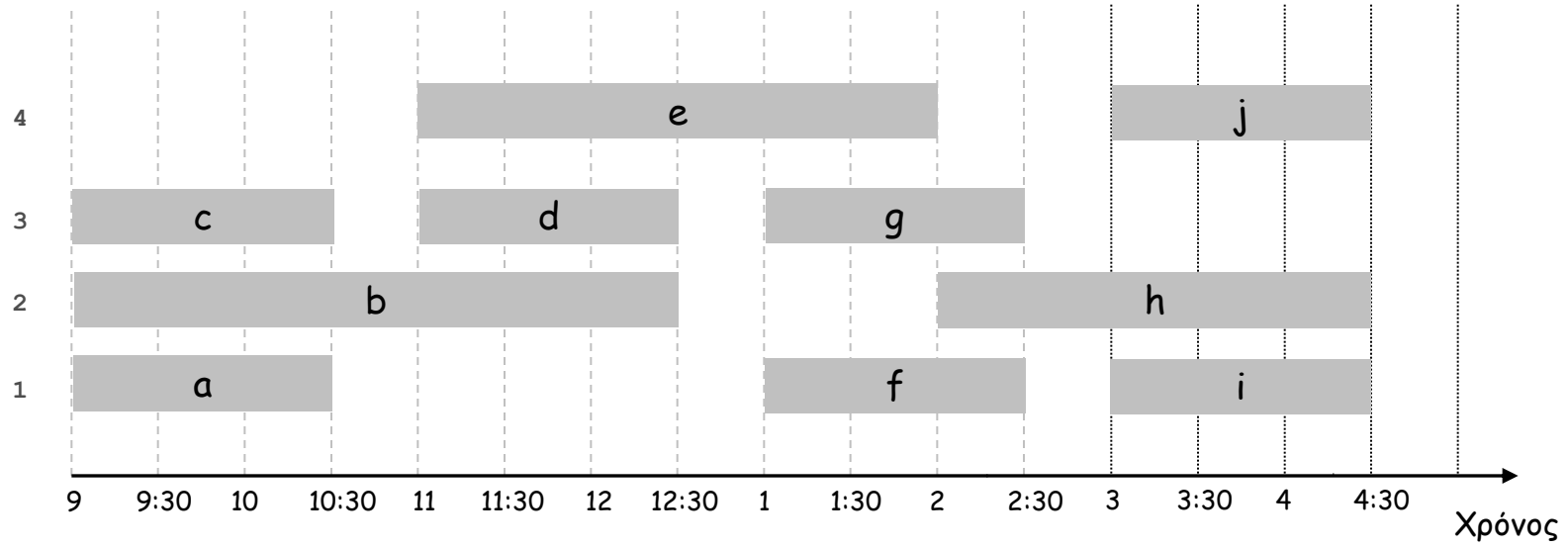


Διαμέριση Χρονικών Διαστημάτων

Διαμέριση Χρονικών Διαστημάτων

- Η διάλεξη j ξεκινάει τη στιγμή s_j και τελειώνει τη στιγμή f_j.
- Στόχος: εύρεση του ελάχιστου πλήθους αιθουσών για να χρονοπρογραμματιστούν όλες οι διαλέξεις έτσι ώστε να μην υπάρχουν δυο διαλέξεις στην ίδια αίθουσα.

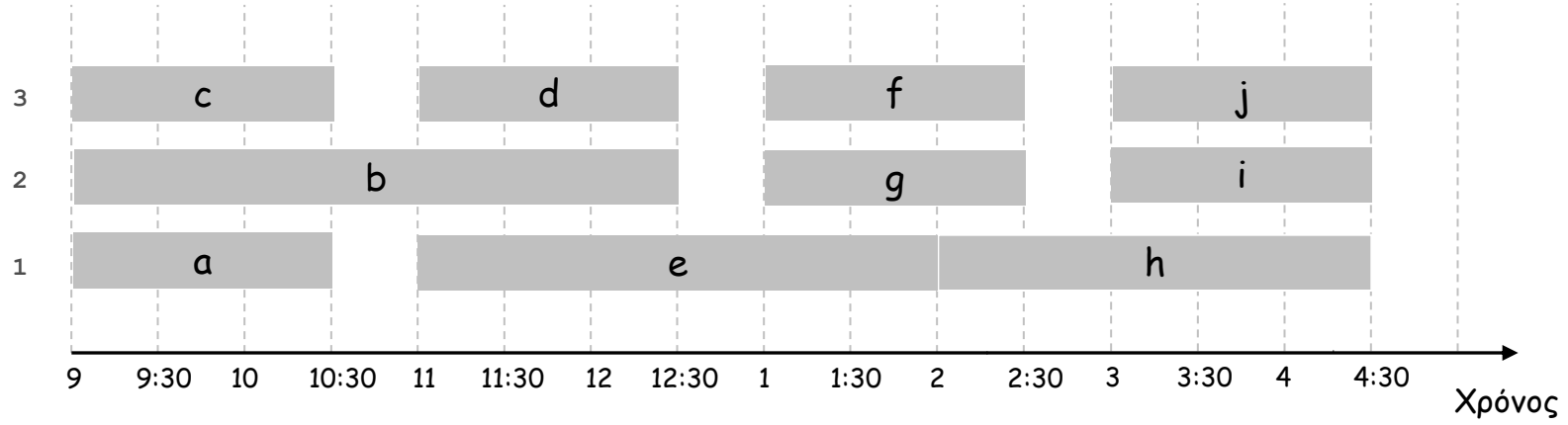
Π.χ.: Έχουμε 4 αίθουσες για την προγραμματισμό 10 διαλέξεων.



Διαμέριση Χρονικών Διαστημάτων

- Η διάλεξη j ξεκινάει τη στιγμή s_j και τελειώνει τη στιγμή f_j.
- Στόχος: εύρεση του ελάχιστου πλήθους αιθουσών για να χρονοπρογραμματιστούν όλες οι διαλέξεις έτσι ώστε να μην υπάρχουν δυο διαλέξεις στην ίδια αίθουσα.

Π.χ.: Το ακόλουθο χρησιμοποιεί μόνο 3 αίθουσες.



Διαμέριση Χρονικών Διαστημάτων: Κάτω Φράγμα στη Βέλτιστη Λύση

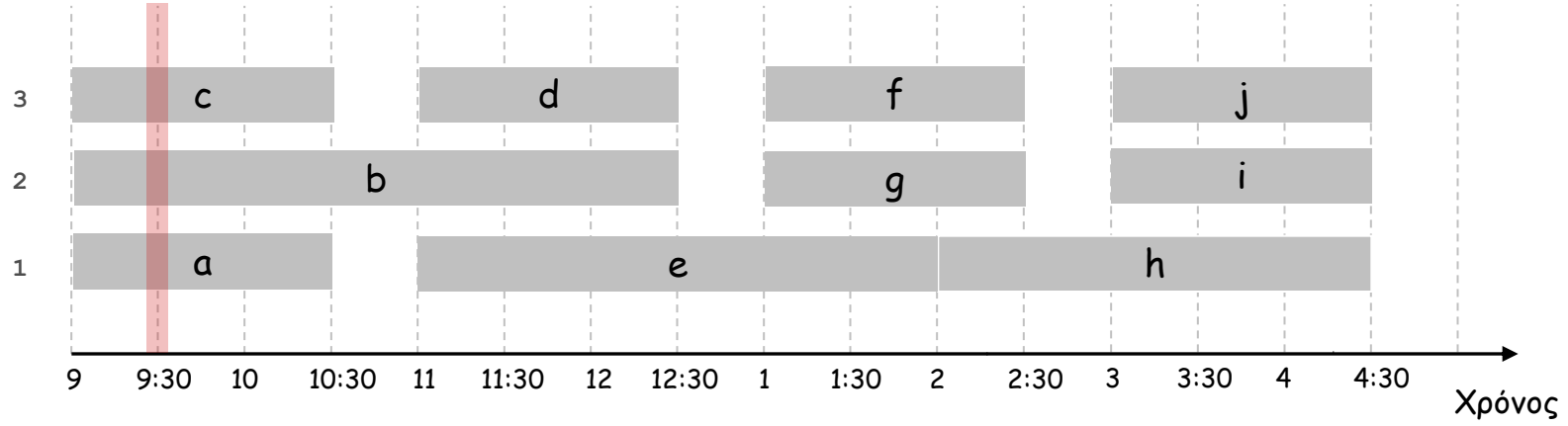
Ορισμός. Το βάθος ενός συνόλου διαστημάτων είναι το μέγιστο πλήθος διαστημάτων που περιέχουν ένα (κοινό) χρονικό σημείο.

Σημαντική Παρατήρηση. Πλήθος των αναγκαίων αιθουσών ≥ βάθος.

Π.χ.: Βάθος του προγράμματος= 3 ⇒ βέλτιστο χρονοδιάγραμμα.

τα a, b, c περιέχουν τη χρονική στιγμή 9:30

Ερώτηση. *Πάντα υπάρχει πρόγραμμα ίσο με το βάθος των διαστημάτων ;*



Διαμέριση Χρονικών Διαστημάτων: Άπληστος Αλγόριθμος

Άπληστος αλγόριθμος. Θεωρήστε τις διαλέξεις σε αύξουσα σειρά ως προς την στιγμή έναρξης: αναθέστε τη διάλεξη σε κάποια συμβατή αίθουσα.

```
Ταξινόμησε τις διαλέξεις (τα διαστήματα) ως προς το χρόνο
έναρξης έτι ώστε        s_1 ≤ s_2 ≤ ... ≤ s_n.

d ← 0   ⟵  πλήθος δεσμευμένων αιθουσών

for j = 1 to n {
   if (η διάλεξη j είναι συμβατή με κάποια αίθουσα k)
      προγραμμάτισε την διάλεξη j για την αίθουσα k
   else
      δέσμευσε μια νέα αίθουσα d + 1
      προγραμμάτισε την διάλεξη j για την αίθουσα d + 1
      d ← d + 1
}
```

Υλοποίηση. $O(n \log n)$.

- Για κάθε αίθουσα k, διατήρησε τη στιγμή λήξης της τελευταίας διάλεξης.
- Διατήρησε τις αίθουσες σε μια ουρά προτεραιότητας.

Διαμέριση Χρονικών Διαστημάτων: Ανάλυση Άπληστου Αλγορίθμου

Παρατήρηση. Ο άπληστος αλγόριθμος δεν προγραμματίζει ποτέ δύο ασύμβατες διαλέξεις στην ίδια αίθουσα.

Θεώρημα. Ο άπληστος αλγόριθμος είναι βέλτιστος.
Απόδειξη.

- Έστω d = πλήθος αιθουσών που δεσμεύει ο άπληστος αλγόριθμος.
- Μια αίθουσα d δεσμεύεται επειδή πρέπει να προγραμματίσουμε μια διάλεξη, έστω j, που είναι ασύμβατη με όλες τις υπόλοιπες d-1 αίθουσες.
- Οι d αυτές διαλέξεις τελειώνουν κάποια στιγμή μετά τη s_j.
- Επειδή ταξινομούμε κατά χρόνο έναρξης, όλες αυτές οι ασυμβατότητες προκαλούνται από διαλέξεις που ξεκινούν πριν τη στιγμή s_j.
- Άρα, έχουμε d διαλέξεις που επικαλύπτονται τη στιγμή $s_j + \varepsilon$ (για μικρό ε).
- Από Σημαντική Παρατήρηση $\Rightarrow$ κάθε χρονοπρογραμματισμός χρειάζεται $\geq$ d αίθουσες. ▪

Χρονοπρογραμματισμός για Ελαχιστοποίηση Καθυστέρησης

Χρονοπρογραμματισμός για Ελαχιστοποίηση Καθυστέρησης

Πρόβλημα ελαχιστοποίησης καθυστέρησης

- Μια μηχανή επεξεργάζεται μια εργασία τη φορά.
- Η εργασία j απαιτεί t_j μονάδες χρόνου και έχει προθεσμία χρόνου d_j.
- Αν η j ξεκινά τη στιγμή s_j, θα τελειώσει τη στιγμή $f_j = s_j + t_j$.
- Καθυστέρηση: $\ell_j = \max \{ 0, f_j - d_j \}$. ($f_j - d_j > 0$ «χάνει» τη προθεσμία)
- Στόχος: χρονοπρογραμματισμός των εργασιών για ελαχιστοποίηση της μέγιστης καθυστέρησης $L = \max \ell_j$.

Π.χ.:

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

Ελαχιστοποίηση Καθυστέρησης: Άπληστοι Αλγόριθμοι

Άπληστο πρότυπο. Θεωρήστε τις εργασίες με κάποια σειρά.

- [Πρώτα ο συντομότερος χρόνος επεξεργασίας] εργασίες $\rightarrow$ σε αύξουσα τάξη ως προς το χρόνο επεξεργασίας t_j.

- [Πρώτα η μικρότερη προθεσμία] εργασίες $\rightarrow$ σε αύξουσα τάξη ως προς την προθεσμία d_j.

- [Ελάχιστο χρονικό περιθώριο] εργασίες $\rightarrow$ σε αύξουσα τάξη ως προς το χρονικό περιθώριο $d_j - t_j$.

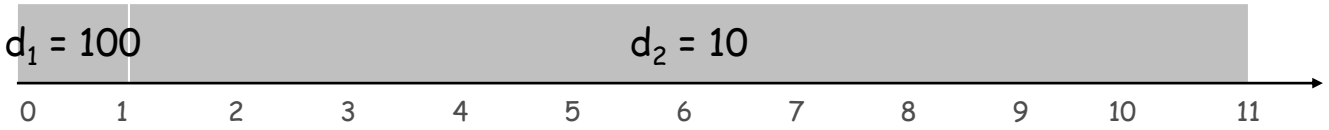
Ελαχιστοποίηση Καθυστέρησης: Άπληστοι Αλγόριθμοι

Άπληστο πρότυπο. Θεωρήστε τις εργασίες με κάποια σειρά.

- [Πρώτα ο συντομότερος χρόνος επεξεργασίας] εργασίες → σε αύξουσα τάξη ως προς το χρόνο επεξεργασίας t_j.

	1	2
t_j	1	10
d_j	100	10

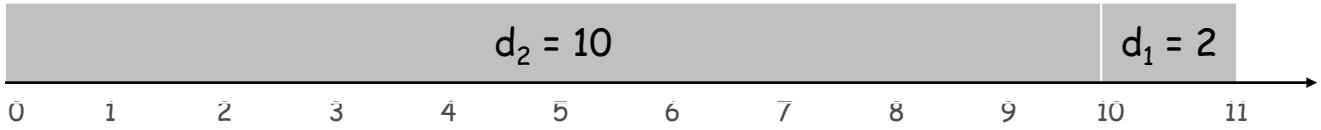
Αντιπαράδειγμα, L=1 αλλά L_{opt}=0



- [Ελάχιστο χρονικό περιθώριο] εργασίες → σε αύξουσα τάξη ως προς το χρονικό περιθώριο $d_j - t_j$.

	1	2
t_j	1	10
d_j	2	10

Αντιπαράδειγμα, L=9 αλλά L_{opt}=1

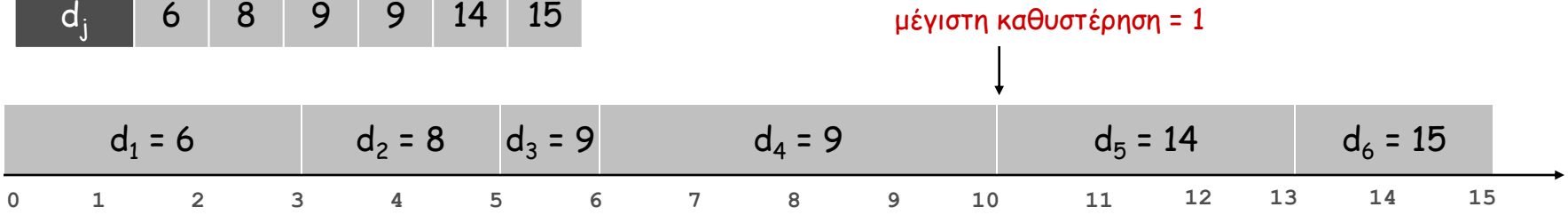


Ελαχιστοποίηση Καθυστέρησης: Άπληστος Αλγόριθμος

Άπληστος αλγόριθμος. Πρώτα η μικρότερη προθεσμία.

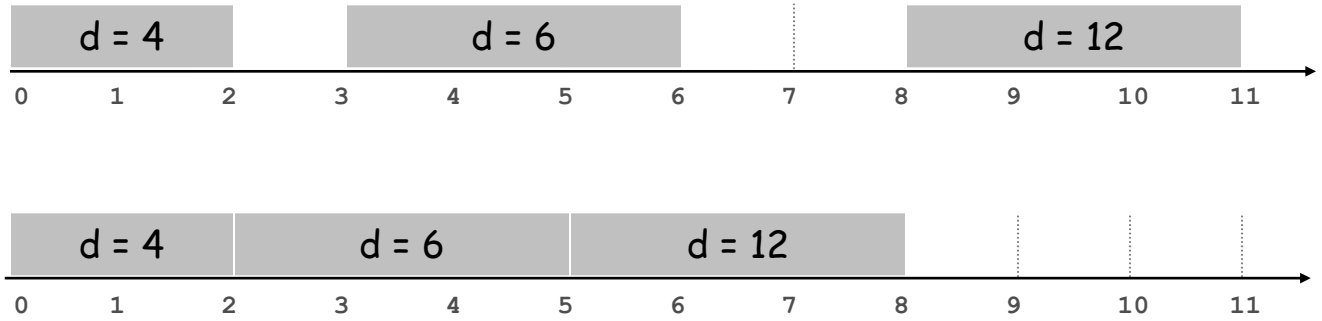
```
Ταξινόμησε n εργασίες ως προς τις προθεσμίες έτσι
ώστε:    d_1 ≤ d_2 ≤ … ≤ d_n

t ← 0
for j = 1 to n
   Ανάθεσε στην εργασία j το διάστημα [t, t + t_j]
   s_j ← t, f_j ← t + t_j
   t ← t + t_j
εκτύπωσε τα διαστήματα [s_j, f_j]
```

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



Ελαχιστοποίηση Καθυστέρησης: Δεν υπάρχει Χρόνος Αδράνειας

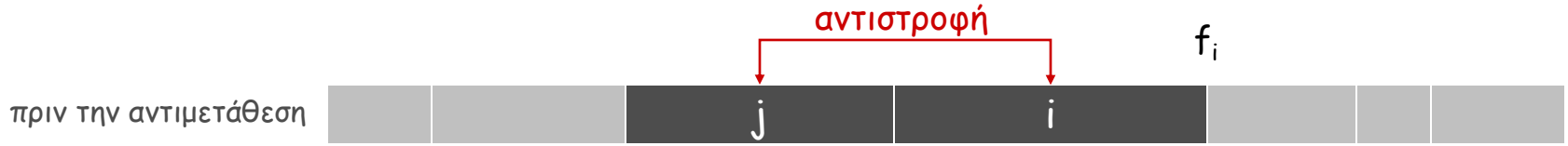
Παρατήρηση. Υπάρχει μια βέλτιστη λύση χωρίς χρόνο αδράνειας.



Παρατήρηση. Ο άπληστος αλγόριθμος δεν έχει χρόνο αδράνειας.

Ελαχιστοποίηση Καθυστέρησης: Αντιστροφές

Ορισμός. Μια **αντιστροφή** σε ένα χρονοδιάγραμμα S είναι ένα ζευγάρι εργασιών i και j με $d_i < d_j$ αλλά η j χρονοπρογραμματίζεται πριν την i.



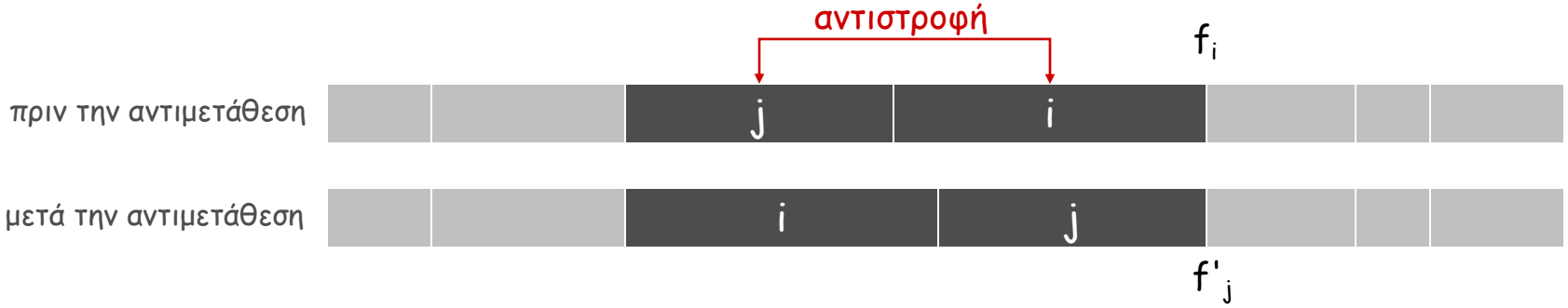
[υποθέτουμε ότι οι εργασίες αριθμούνται έτσι ώστε $d_1 \leq d_2 \leq \ldots \leq d_n$]

Παρατήρηση. Το άπληστο χρονοδιάγραμμα δεν έχει αντιστροφές.

Παρατήρηση. Αν ένα χρονοδιάγραμμα (χωρίς χρόνο αδράνειας) έχει μια αντιστροφή, τότε έχει ένα ζευγάρι αντεστραμμένων εργασιών που προγραμματίζονται σε διαδοχικές θέσεις.

Ελαχιστοποίηση Καθυστέρησης: Αντιστροφές

Ορισμός. Μια **αντιστροφή** σε ένα χρονοδιάγραμμα S είναι ένα ζευγάρι εργασιών i και j με $d_i < d_j$ αλλά η j χρονοπρογραμματίζεται πριν την i.



Ισχυρισμός. Η αλλαγή δυο διαδοχικών, αντεστραμμένων εργασιών μειώνει το πλήθος των αντιστροφών κατά 1 και δεν αυξάνει την μέγιστη καθυστέρηση.

Απόδειξη. ℓ : καθυστέρηση πριν την αλλαγή, ℓ' : μετά την αλλαγή.

- $\ell'_k = \ell_k$ για κάθε $k \neq i, j$
- $\ell'_i \leq \ell_i$
- Αν η εργασία j καθυστερήσει:

$$\ell_j = \max \{ 0,\ f_j - d_j \}$$

$$\begin{aligned} \ell'_j &= f'_j - d_j && (\text{ορισμος}) \\ &= f_i - d_j && (\eta\ \text{j}\ \tau\epsilon\lambda\epsilon\iota\omega\nu\epsilon\iota\ \tau\eta\ \sigma\tau\iota\gamma\mu\eta\ \text{f}_\text{i}) \\ &\leq f_i - d_i && (i < j) \\ &\leq \ell_i && (\text{ορισμος}) \end{aligned}$$

Ελαχιστοποίηση Καθυστέρησης: Ανάλυση Άπληστου Αλγορίθμου

Θεώρημα. Το χρονοδιάγραμμα S του άπληστου αλγορίθμου είναι βέλτιστο.

Απόδειξη. (με άτοπο)

S* : βέλτιστο χρονοδιάγραμμα με το μικρότερο πλήθος αντιστροφών.

- Γνωρίζουμε ότι το S* δεν έχει χρόνο αδράνειας.
- Αν το S* δεν έχει αντιστροφές, τότε S = S*.
- Αν το S* έχει μια αντιστροφή, τότε έστω i-j μια γειτονική αντιστροφή.
  - αντιμεταθέτοντας i με j δεν αυξάνει τη μέγιστη καθυστέρηση και μειώνει αυστηρά το πλήθος των αντιστροφών
  - αυτό έρχεται σε αντίθεση με τον ορισμό του S* ▪

Στρατηγικές Ανάλυσης Άπληστων Αλγορίθμων

Ο άπληστος αλγόριθμος υπερτερεί. Δείχνουμε ότι μετά από κάθε βήμα του άπληστου αλγορίθμου, η λύση του είναι τουλάχιστον τόσο καλή όσο οποιουδήποτε άλλου αλγορίθμου.

Δομική ανάλυση. Ανακαλύπτουμε ένα απλό «δομικό» όριο που να επιβεβαιώνει ότι κάθε πιθανή λύση πρέπει να έχει μια συγκεκριμένη τιμή. Στη συνέχεια δείχνουμε ότι ο άπληστος αλγόριθμος πάντα πετυχαίνει αυτό το όριο.

Επιχείρημα ανταλλαγής. Μετατρέπουμε σταδιακά οποιαδήποτε λύση σε αυτή που ανακαλύπτει ο άπληστος αλγόριθμος χωρίς να επηρεάζεται η ποιότητα της λύσης.

Άλλοι άπληστοι αλγόριθμοι. Kruskal, Prim, Dijkstra, Huffman, …

Ρέστα σε Κέρματα

Ρέστα σε Κέρματα

Στόχος. Δεδομένου μιας ονομασίας νομισμάτων: 1, 2, 5, 10, 20, 50, 100, 200 βρες μια μέθοδο για να πληρώσεις τον πελάτη χρησιμοποιώντας το ελάχιστο πλήθος κερμάτων.

Π.χ.: 34λεπτά.

Ο αλγόριθμος του ταμία. Σε κάθε επανάληψη, πρόσθεσε το κέρμα με την μεγαλύτερη αξία που δεν ξεπερνά το συνολικό ποσό.

Π.χ.: 2.89€

Ρέστα σε Κέρματα: Άπληστος Αλγόριθμος

Ο αλγόριθμος του ταμία. Σε κάθε επανάληψη, πρόσθεσε το κέρμα με την μεγαλύτερη αξία που δεν ξεπερνά το συνολικό ποσό.

Ταξινόμησε τα κέρματα ως προς την αξία: $c_1 < c_2 < \ldots < c_n$.

επιλεγμένα κέρματα

```
S ← ∅
while (x ≠ 0) {
    έστω k ο μέγιστος ακέραιος για τον οποίο c_k ≤ x
    if (k = 0)
        return "δεν υπάρχει λύση"
    x ← x - c_k
    S ← S ∪ {k}
}
return S
```

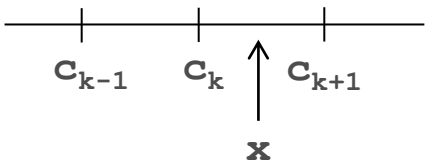
Ερώτημα. *Είναι ο αλγόριθμος του ταμία βέλτιστος;*

Ρέστα σε Κέρματα: Ανάλυση του Άπληστου Αλγορίθμου

Θεώρημα. Ο άπληστος αλγόριθμος είναι βέλτιστος για : 1, 5, 10, 25, 100.

Απόδειξη. (με επαγωγή στο x)

- Έστω ένας βέλτιστος τρόπος για ποσό $c_k \le x < c_{k+1}$: ο άπληστος αλγόριθμος επιλέγει το κέρμα k.
- Ισχυριζόμαστε ότι κάθε βέλτιστη λύση πρέπει να συμπεριλάβει το c_k.
  - διαφορετικά, χρειάζεται κέρματα $c_1, \ldots, c_{k-1}$ που αθροίζουν στο x
  - ο κάτω πίνακας δείχνει ότι καμία βέλτιστη λύση μπορεί να το πετύχει
- Το πρόβλημα ανάγεται στα κέρματα για το ποσό των $x - c_k$, που, από την επαγωγική υπόθεση είναι βέλτιστο από τον άπληστο αλγόριθμο. ▪



$sol(x) = sol(x - c_k) + 1$

k	c_k	Οι βέλτιστες λύσεις πρέπει να ικανοποιούν	Max τιμή νομισμάτων 1, 2, …, k-1 σε κάθε OPT
1	1	$1\Lambda \le 4$	-
2	5	$5\Lambda \le 1$	4
3	10	$10\Lambda \le 2$	4 + 5 = 9
4	25	$25\Lambda \le 3$	20 + 4 = 24
5	100	χωρίς όριο	75 + 24 = 99

Ρέστα σε Κέρματα: Ανάλυση του Άπληστου Αλγορίθμου

Παρατήρηση. Ο άπληστος αλγόριθμος δεν δουλεύει για κάθε είδους νομισμάτων: Π.χ. 1, 10, 21, 34, 70, 100

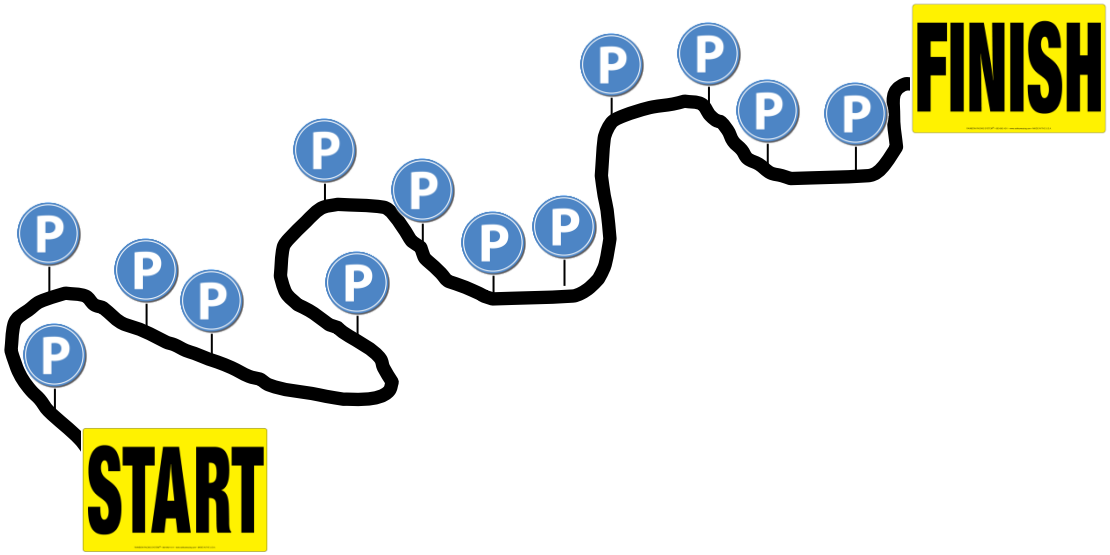
Αντιπαράδειγμα. 140¢.

- Άπληστος: 100, 34, 1, 1, 1, 1, 1, 1.
- Βέλτιστος: 70, 70.

Άσκηση σε Άπληστους Αλγορίθμους

Γρήγορο, ξεκούραστο και ήσυχο ταξίδι: Θέλουμε να ταξιδέψουμε στην ακόλουθη ευθεία μήκους L διανύοντας με το φως της ημέρας το πολύ d χλμ.

Όταν βραδιάζει θέλουμε να ξεκουραζόμαστε στα σημεία P που βρίσκονται σε αποστάσεις $x_1, x_2, \ldots, x_n$ από την αρχή.



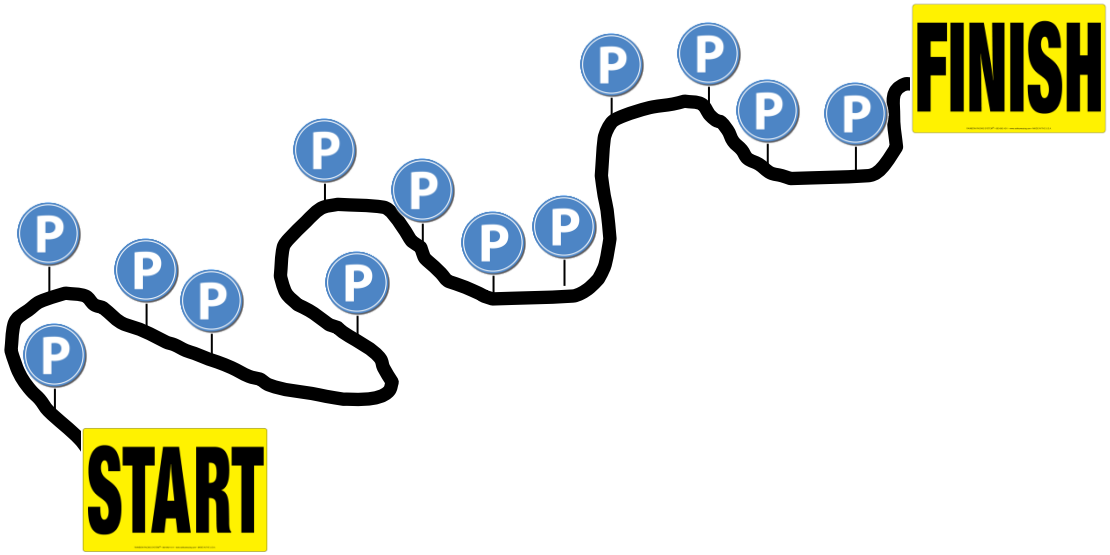
Θέλουμε να βρούμε όσο το δυνατό λιγότερα σημεία ξεκούρασης έτσι ώστε να πραγματοποιήσουμε όσο γίνεται γρηγορότερα το ταξίδι.

Άσκηση σε Άπληστους Αλγορίθμους

Άπληστος Αλγόριθμος: Κάθε φορά που φτάνουμε σε ένα σημείο P , x_j αποφασίζουμε αν προλαβαίνουμε να πάμε μέχρι το επόμενο P , x_{j+1} πριν βραδιάσει. Αν μπορούμε συνεχίζουμε, διαφορετικά σταματάμε.

↑

αν $(x_{j+1} - x_j) < d$



Θέλουμε να βρούμε όσο το δυνατό λιγότερα σημεία ξεκούρασης έτσι ώστε να πραγματοποιήσουμε όσο γίνεται γρηγορότερα το ταξίδι.

Άσκηση σε Άπληστους Αλγορίθμους

Άπληστος Αλγόριθμος: Κάθε φορά που φτάνουμε σε ένα σημείο P , x_j αποφασίζουμε αν προλαβαίνουμε να πάμε μέχρι το επόμενο P , x_{j+1} πριν βραδιάσει. Αν μπορούμε συνεχίζουμε, διαφορετικά σταματάμε.

Έστω $R = \{x_1, \ldots, x_m, \ldots, x_k\}$ τα σημεία που επιλέγει ο άπληστος
Έστω $S = \{y_1, \ldots, y_m\}$ τα σημεία που επιλέγει ο βέλτιστος

Υποθέτουμε ότι $m < k$ για να φτάσουμε σε άτοπο.

Ισχυρισμός. Για κάθε μέρα $j=1,\ldots,m$ έχουμε $x_j \geq y_j$.

Απόδειξη (με επαγωγή στο j)
Βάση: j=1 ισχύει καθώς προχωράμε όσο γίνεται μακρύτερα
Επαγωγική Υπόθεση: Ισχύει για όλα τα $i < j$.
Επαγωγικό βήμα: Στο S θα πρέπει: $y_j - y_{j-1} \leq d$
Από Επαγωγική Υπόθεση θα πρέπει: $x_{j-1} \geq y_{j-1} \Leftrightarrow y_j - x_{j-1} \leq y_j - y_{j-1} \leq d$
Δηλαδή: $y_j - x_{j-1} \leq d \Leftrightarrow x_j \geq y_j$ (διότι η άπληστη επιλογή $x_j - x_{j-1} \leq d$)

Άσκηση σε Άπληστους Αλγορίθμους

Άπληστος Αλγόριθμος: Κάθε φορά που φτάνουμε σε ένα σημείο P , x_j αποφασίζουμε αν προλαβαίνουμε να πάμε μέχρι το επόμενο P , x_{j+1} πριν βραδιάσει. Αν μπορούμε συνεχίζουμε, διαφορετικά σταματάμε.

Έστω R = $\{x_1, \ldots, x_m, \ldots, x_k\}$ τα σημεία που επιλέγει ο άπληστος
Έστω S = $\{y_1, \ldots, y_m\}$ τα σημεία που επιλέγει ο βέλτιστος

Υποθέτουμε ότι $m < k$ για να φτάσουμε σε άτοπο.

Ισχυρισμός. Για κάθε μέρα $j=1,\ldots,m$ έχουμε $x_j \geq y_j$.

Επομένως $x_m \geq y_m$

Στο x_{m+1} ξεκουραζόμαστε, άρα θα πρέπει $x_m \leq L - d$

Δηλαδή $y_m \leq x_m \leq L - d$ και επειδή $m < k \Rightarrow y_m < L - d$. Άτοπο διότι το S θα έπρεπε τότε να είχε επιλέξει και άλλο σημείο για να τερματίσει.

Άδειες Λογισμικών (Άσκηση)

Άδειες λογισμικών. Κάθε άδεια πωλείται για 100€ και κάθε μήνα η εταιρία σας μπορεί να αγοράσει το πολύ μια άδεια.

Το κόστος μιας άδειας αυξάνει κάθε μήνα κατά ένα συντελεστή $r_j > 1$, δηλαδή αν μια άδεια αγοραστεί μετά από k μήνες θα κοστίζει $100 \cdot (r_j)^k$.

Οι συντελεστές μεταξύ τους είναι όλοι διαφορετικοί: r_i ,r_j $r_i \neq r_j$.

Στόχος: Με ποια σειρά πρέπει να αγοράσει η εταιρεία n άδειες με συντελεστές $r_1, r_2, \ldots, r_n$ έτσι ώστε να ελαχιστοποιήσει το συνολικό κόστος;

Π.χ. $r_1 = 2, r_2 = 3, r_3 = 4$:

Κατά αύξουσα σειρά: $100 \cdot (2 + 3^2 + 4^3) = 7500$

Κατά φθίνουσα σειρά: $100 \cdot (4 + 3^2 + 2^3) = 2100$

Άδειες Λογισμικών (Άσκηση)

Ισχυρισμός. Η ταξινόμηση κατά φθίνουσα τάξη δίνει την βέλτιστη λύση.

Απόδειξη (με άτοπο και επιχείρημα ανταλλαγής)

- S = η λύση του άπληστου αλγορίθμου κατά φθίνουσα τάξη.
- Φ = η βέλτιστη λύση
- Υποθέτουμε ότι $S \neq \Phi$
- Η Φ περιέχει μια τουλάχιστον αντιστροφή ως προς την S
- Δηλαδή, σε δυο διαδοχικούς μήνες k και k+1 της Φ θα πρέπει: $r_k <_\Phi r_{k+1}$
- Αν ανταλλάξουμε τα r_k και r_{k+1} θα πρέπει να κερδίσουμε κάτι:

$$\Phi: \quad 100 \cdot ((r_k)^k + (r_{k+1})^{k+1})$$

$$\Phi': \quad 100 \cdot ((r_{k+1})^k + (r_k)^{k+1}) \quad \text{(δηλαδή θέλουμε } \Phi' < \Phi)$$

- Τα υπόλοιπα κομμάτια των Φ και Φ′ παραμένουν αναλλοίωτα.
- Γνωρίζουμε ότι $r_k < r_{k+1}$ και επειδή $r_i > 1$:

$$r_k (r_k - 1) < r_{k+1} (r_{k+1} - 1) \Leftrightarrow$$

$$(r_k)^k (r_k - 1) < (r_{k+1})^k (r_{k+1} - 1) \Leftrightarrow$$

$$(r_k)^{k+1} - (r_k)^k < (r_{k+1})^{k+1} - (r_{k+1})^k \Leftrightarrow$$

$(r_k)^{k+1} + (r_{k+1})^k < (r_{k+1})^{k+1} + (r_k)^k \Leftrightarrow \Phi' < \Phi$. Άρα Φ δεν είναι βέλτιστη. ▪

Άδειες Λογισμικών (Άσκηση)

Ισχυρισμός. Η ταξινόμηση κατά φθίνουσα τάξη δίνει την βέλτιστη λύση.

Άπληστος Αλγόριθμος:

1. Ταξινόμησε τις άδειες κατά φθίνουσα τάξη $r_1 > r_2 > \ldots > r_n$
2. Κάθε μήνα επέλεξε το r_i

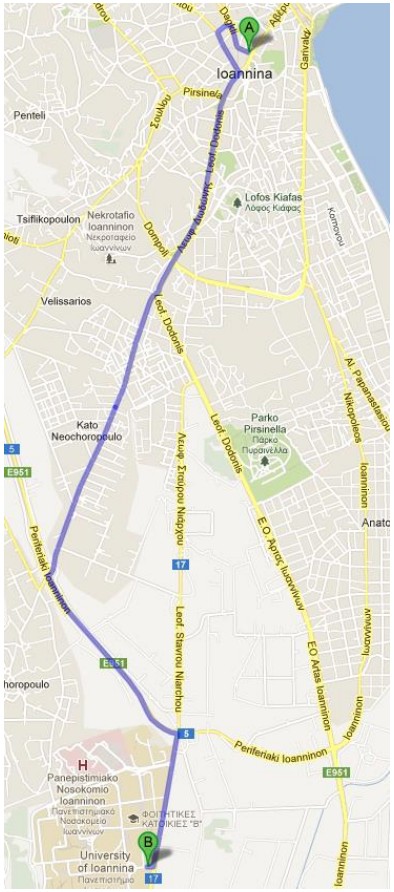
Χρόνος $O(n \log n)$: λόγω ταξινόμησης

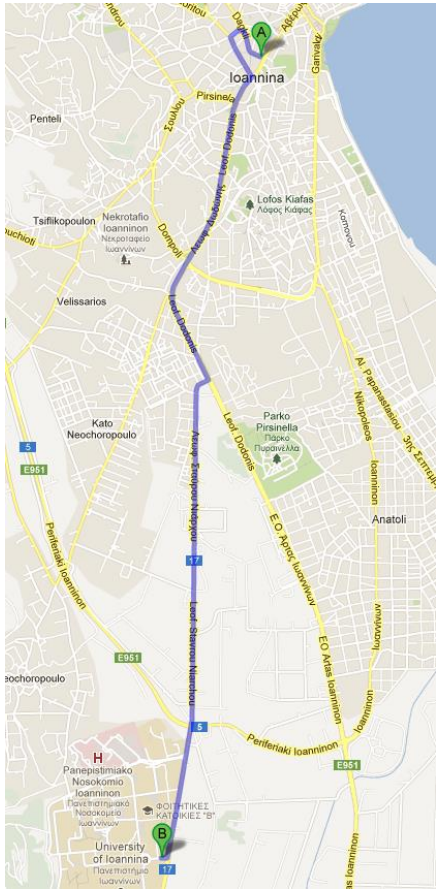
Παραπλήσια εκδοχή:

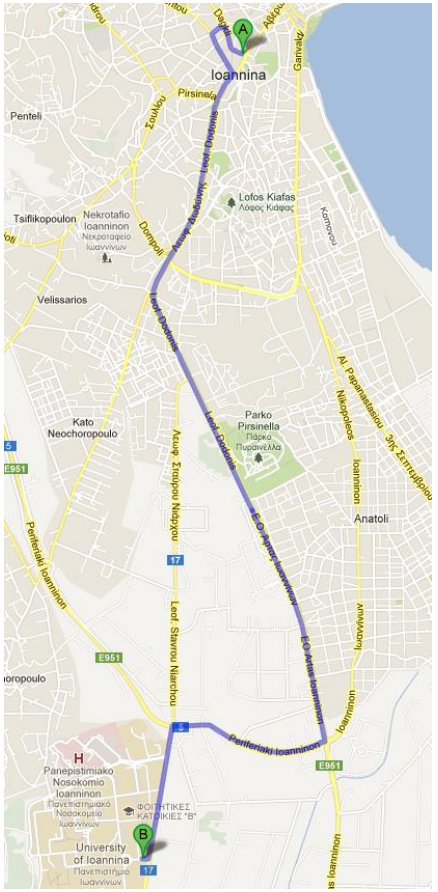
Θέλουμε να πουλήσουμε Η/Υ που το κόστος τους μειώνεται κάθε μήνα κατά ένα συντελεστή $r_j < 1$: μετά από k μήνες κοστίζει $100 \cdot (r_j)^k$.
Ποια είναι η βέλτιστη σειρά πώλησης;

Π.χ. $r_1 = 3/4$, $r_2 = 1/2$, $r_3 = 1/100$

Συντομότερες Διαδρομές σε Γραφήματα







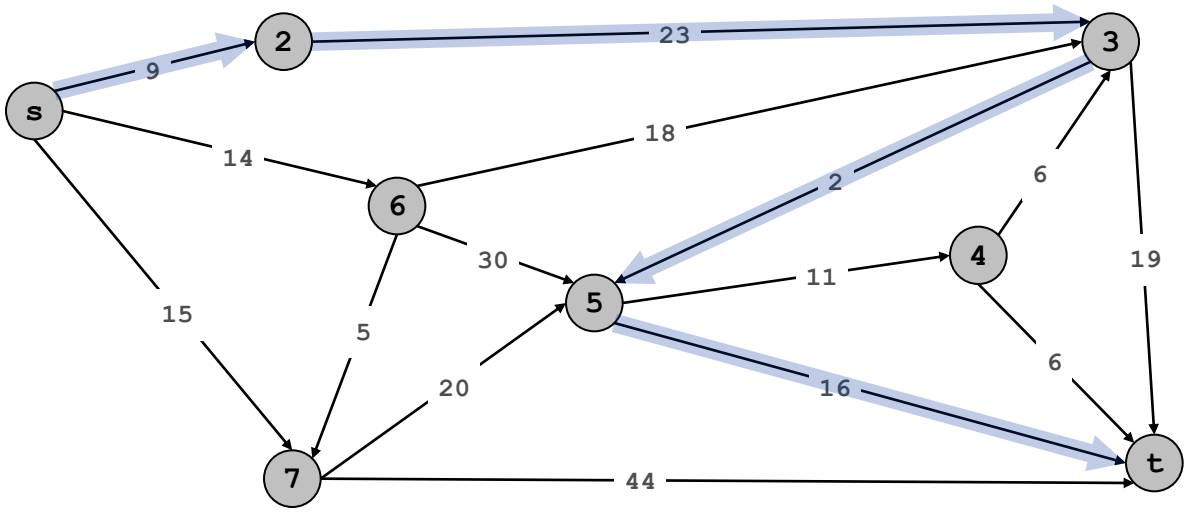
Πρόβλημα Συντομότερης Διαδρομής

Δίκτυο συντομότερων διαδρομών

- Κατευθυνόμενο γράφημα $G = (V, E)$.
- Αφετηρία s, προορισμός t.
- Κόστος ℓ_e = μήκος της ακμής e.

Πρόβλημα συντομότερης διαδρομής: εύρεση της συντομότερης κατευθυνόμενης διαδρομής από τον s στον t. Κόστος s-t διαδρομής: δ(s,t)

κόστος διαδρομής = άθροισμα των ακμών (τα μήκη τους) της διαδρομής



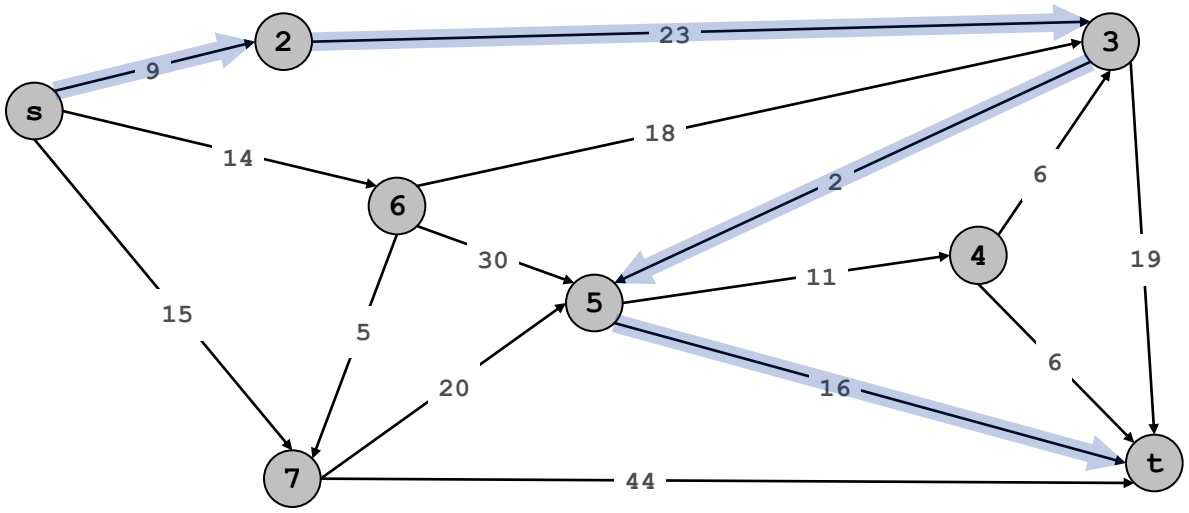
Κόστος διαδρομής (s-2-3-5-t)
= 9 + 23 + 2 + 16
= 50.

Ιδιότητα Συντομότερης Διαδρομής

Ιδιότητα υποδιαδρομής

- Έστω $P = (v_1, \ldots, v_k)$ μια συντομότερη διαδρομή.
- Τότε οποιαδήποτε υποδιαδρομή $(v_i, \ldots, v_j)$ της P είναι βέλτιστη, $1 \le i,j \le k$
- Δηλαδή:

  Η υποδιαδρομή $(v_i, \ldots, v_j)$ είναι η συντομότερη $v_i - v_j$ διαδρομή.



Κόστος διαδρομής (s-2-3-5-t)
= 9 + 23 + 2 + 16
= 50.

Αλγόριθμος του Dijkstra

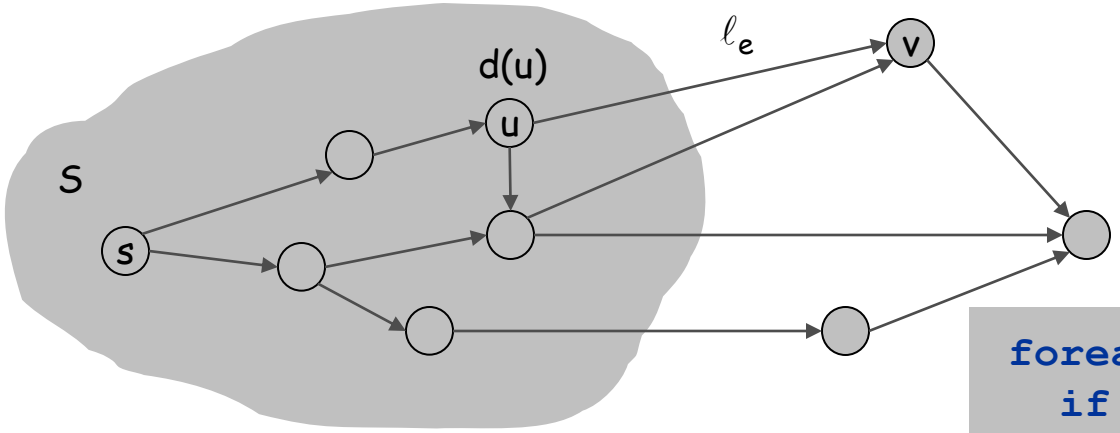
Ο αλγόριθμος του Dijkstra.

- Διατήρηση συνόλου κόμβων S που έχουν ανακαλυφθεί: για κάθε κόμβο u του S γνωρίζουμε τη συντομότερη διαδρομή d(u) από την αφετηρία s.
- Αρχικοποιούμε $S = \{ s \}$, $d(s) = 0$, $\forall v \in V-\{s\}$: $d(v) = \infty$.
- Επαναληπτικά επιλέγουμε κόμβο v που δεν έχει ανακαλυφθεί που ελαχιστοποιεί

$$\pi(v) = \min_{e=(u,v) : u \in S} d(u) + \ell_e ,$$

προσθέτουμε v στο S, και θέτουμε $d(v) = \pi(v)$.

συντομότερη διαδρομή σε κάποιο u που έχει ανακαλυφθεί, ακολουθούμενο από μια μοναδική ακμή (u, v)




```
foreach edge e = (u,v) do
  if d(v) > d(u) + ℓe
     d(v) = d(u) + ℓe
```

Αλγόριθμος του Dijkstra

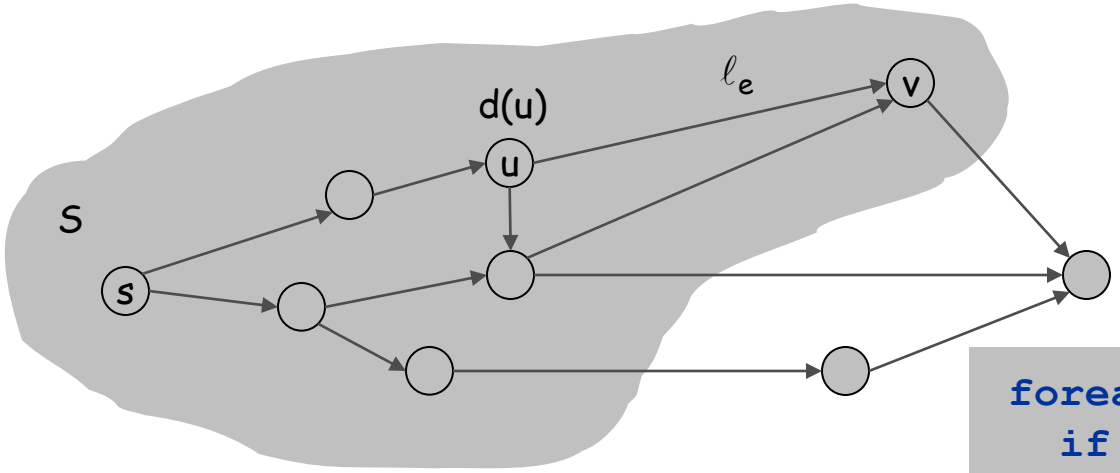
Ο αλγόριθμος του Dijkstra.

- Διατήρηση συνόλου κόμβων S που έχουν ανακαλυφθεί: για κάθε κόμβο u του S γνωρίζουμε τη συντομότερη διαδρομή d(u) από την αφετηρία s.
- Αρχικοποιούμε $S = \{ s \}$, $d(s) = 0$, $\forall v \in V-\{s\}$: $d(v) = \infty$.
- Επαναληπτικά επιλέγουμε κόμβο v που δεν έχει ανακαλυφθεί που ελαχιστοποιεί

$$\pi(v) = \min_{e=(u,v) : u \in S} d(u) + \ell_e ,$$

προσθέτουμε v στο S, και θέτουμε $d(v) = \pi(v)$.

συντομότερη διαδρομή σε κάποιο u που έχει ανακαλυφθεί, ακολουθούμενο από μια μοναδική ακμή (u, v)




```
foreach edge e = (u,v) do
  if d(v) > d(u) + ℓe
     d(v) = d(u) + ℓe
```

Αλγόριθμος του Dijkstra: Απόδειξη Ορθότητας

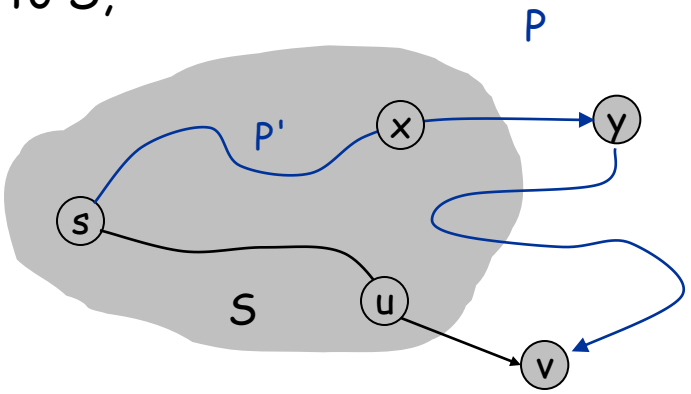
Ιδιότητα. Για κάθε κόμβο $u \in S$, $d(u)$ είναι το μήκος συντομότερης s-u διαδρομής.

Απόδειξη. (με επαγωγή στο $|S|$)

Βάση: Για $|S| = 1$ ισχύει.

Επαγωγική Υπόθεση: Υποθέτουμε ότι ισχύει για $|S| = k \geq 1$.

- Έστω v ο επόμενος κόμβος για το S, και έστω (u,v) η επιλεγμένη ακμή.
- Η συντομότερη s-u διαδρομή + (u, v) : s-v διαδρομή μήκους $\pi(v)$.
- Έστω τυχαία s-v διαδρομή P. Θα δείξουμε ότι δεν μπορεί να είναι < $\pi(v)$.
- Έστω (x,y) η πρώτη ακμή του P που φεύγει από το S, και έστω P' η υποδιαδρομή προς τον x.
- Η P είναι ήδη πολύ μεγάλη από τη στιγμή που φεύγει από το S.



$$\ell(P) \geq \ell(P') + \ell(x,y) \geq d(x) + \ell(x,y) \geq \pi(y) \geq \pi(v)$$

μη-αρνητικά βάρη	επαγωγική υπόθεση	ορισμός του $\pi(y)$	Ο αλγόριθμος του Dijkstra επιλέγει τον v αντί του y

Αλγόριθμος του Dijkstra: Υλοποίηση

Για κάθε κόμβο v που δεν έχει ανακαλυφθεί διατηρεί $\pi(v) = \min_{e=(u,v) : u \in S} d(u) + \ell_e$.

- Επόμενος κόμβος = κόμβος με το ελάχιστο π(v).
- Όταν ανακαλύπτουμε τον v, για κάθε προσκείμενη ακμή e = (v, w), ενημερώνουμε

$$\pi(w) = \min \{ \pi(w), \pi(v) + \ell_e \}.$$

```
foreach edge e = (u,v) do
  if d(v) > d(u) + ℓe
     d(v) = d(u) + ℓe
```

Αποδοτική υλοποίηση. Διατήρησε τους κόμβους που δεν έχουν ανακαλυφθεί σε μια ουρά προτεραιότητας (PQ) με προτεραιότητα π(v).

Λειτουργία PQ	Dijkstra	Πίνακας	Δυαδ. Σωρός	Ουρά Fib †
Insert	n	n	log n	1
ExtractMin	n	n	log n	log n
ChangeKey	m	1	log n	1
IsEmpty	n	1	1	1
Σύνολο		n^2	m log n	m + n log n

† Ουρά Fibonacci: τα όρια των διακριτών λειτουργιών είναι επιμερισμένοι χρόνοι

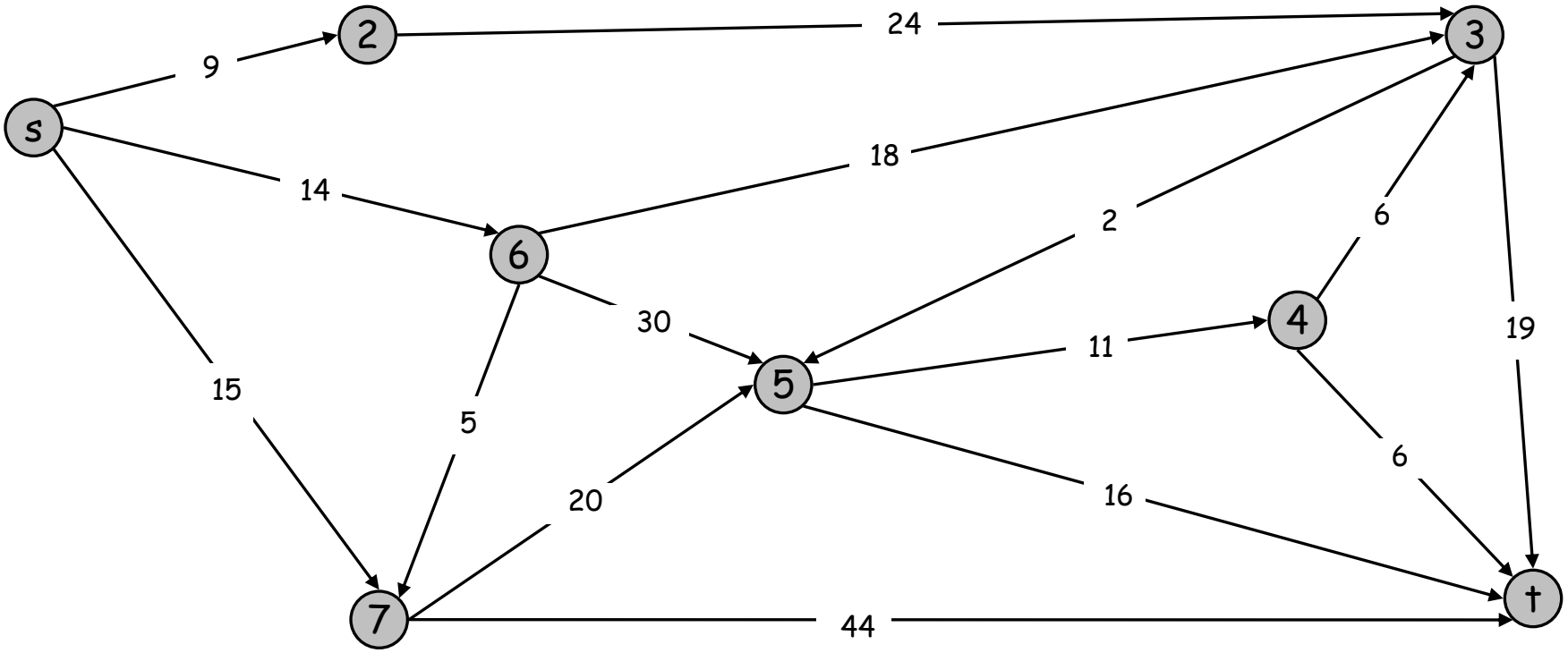
Αλγόριθμος του Dijkstra: Υλοποίηση

```
Dijkstra(G,s) {
foreach v ∈ V do  {  d[v] = ∞  }
Αρχικοποίησε μια ουρά προτεραιότητας Q με n στοιχεία:
Q = V
d[s] = 0

while(Q δεν είναι άδεια)  {
        v = ExtractMin(Q)
        foreach e = (v,w)∈ E do
                if d(w) > d(v) + ℓe {
                     d(w) = d(v) + ℓe  // μείωση προτεραιότητας
                     ChangeKey(Q,w,d[w])  // μείωση κλειδιού
                }
        }
}
```

- Διατηρούμε το V-S σε μια ουρά προτεραιότητας Q
- Στην ουσία βρίσκουμε τις συντομότερες διαδρομές (αποστάσεις d[v]) για κάθε κόμβο v από την αφετηρία s
- Μπορούμε να επεκτείνουμε τον αλγόριθμο έτσι ώστε να αποθηκεύει τον προηγούμενο κόμβο για την διαδρομή από το s στο t (p[w]=v)

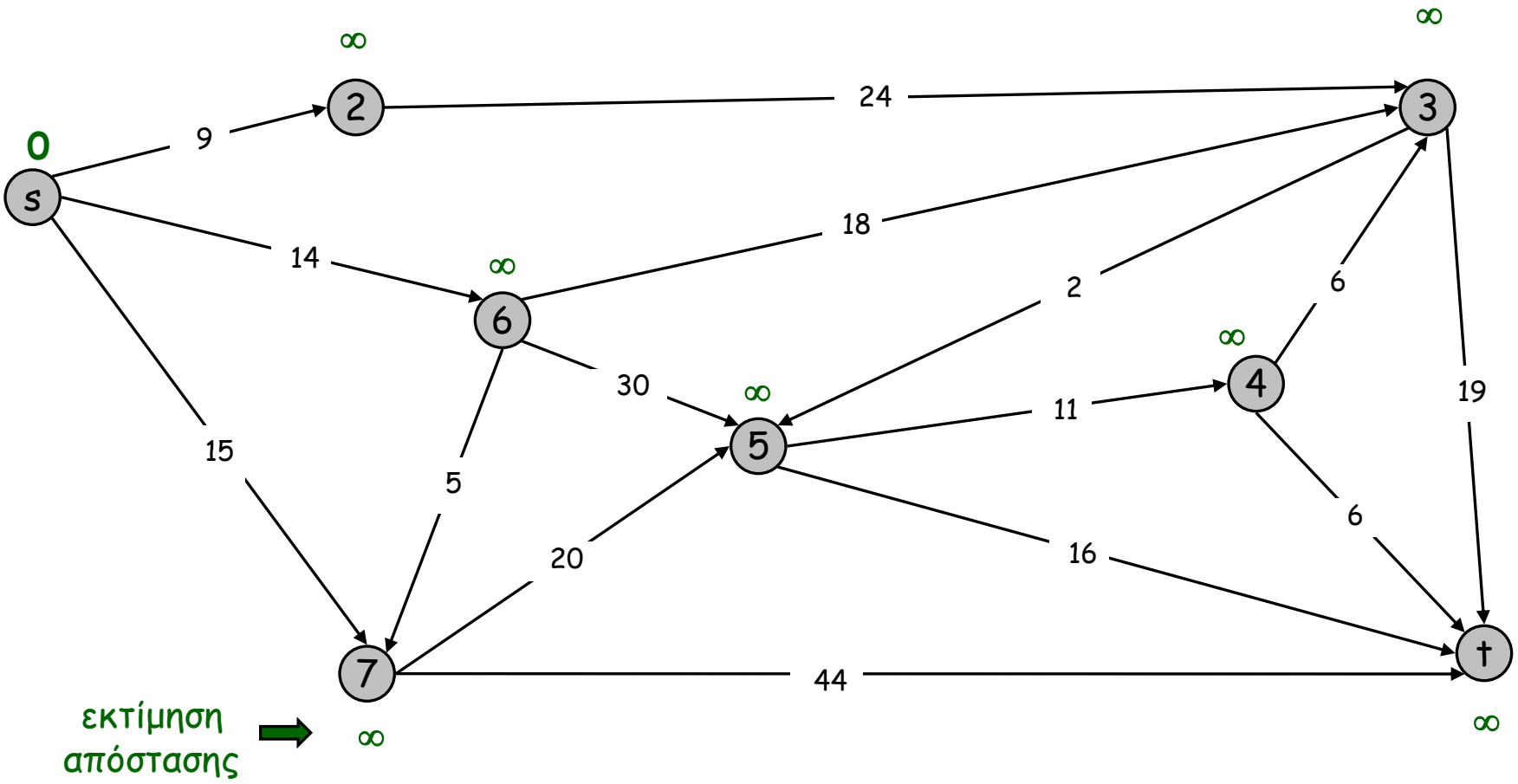
Αλγόριθμος του Dijkstra: Παράδειγμα

Εύρεση της συντομότερης διαδρομής από το s στο t.



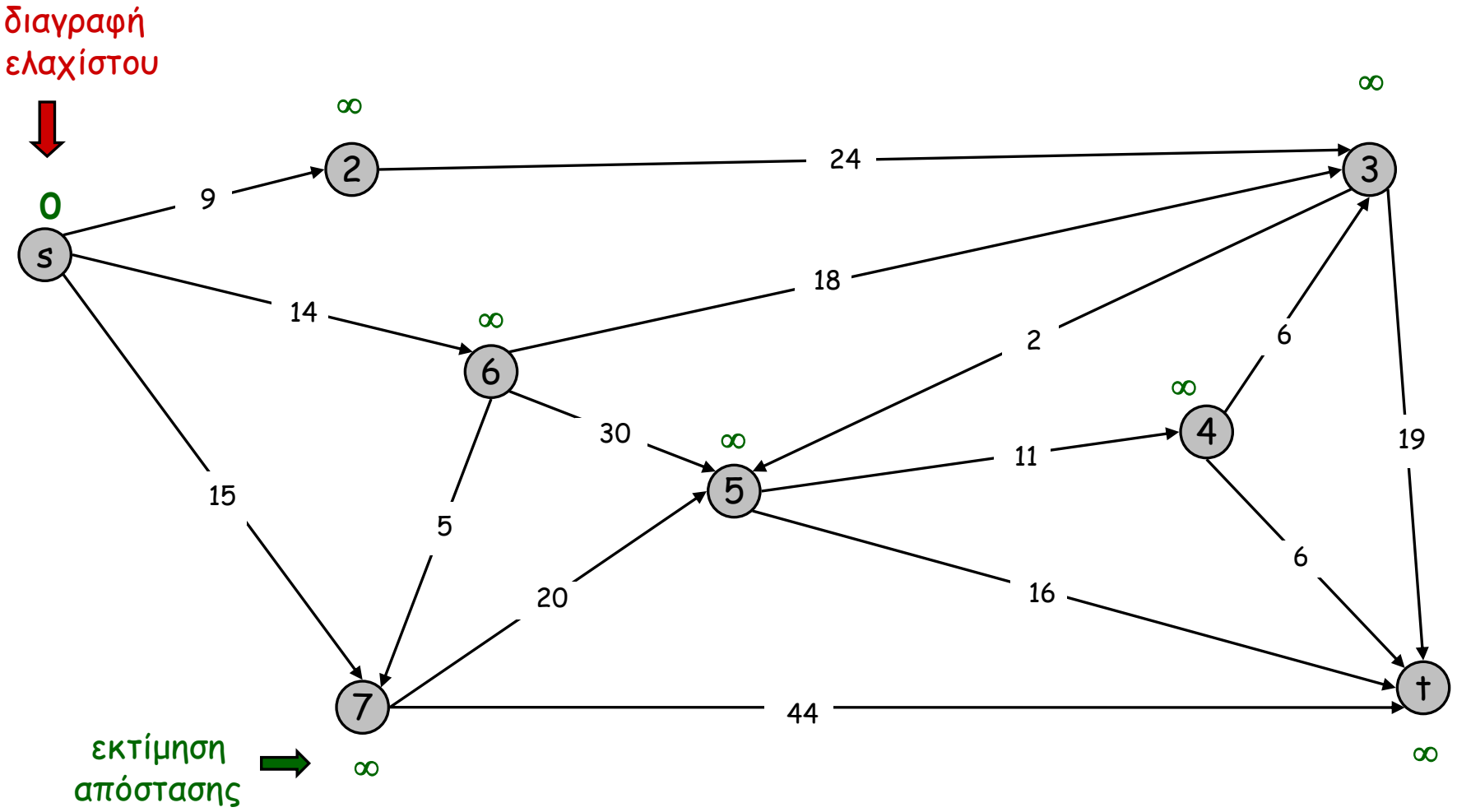
Αλγόριθμος του Dijkstra: Παράδειγμα

S = { }
PQ = { s, 2, 3, 4, 5, 6, 7, t }



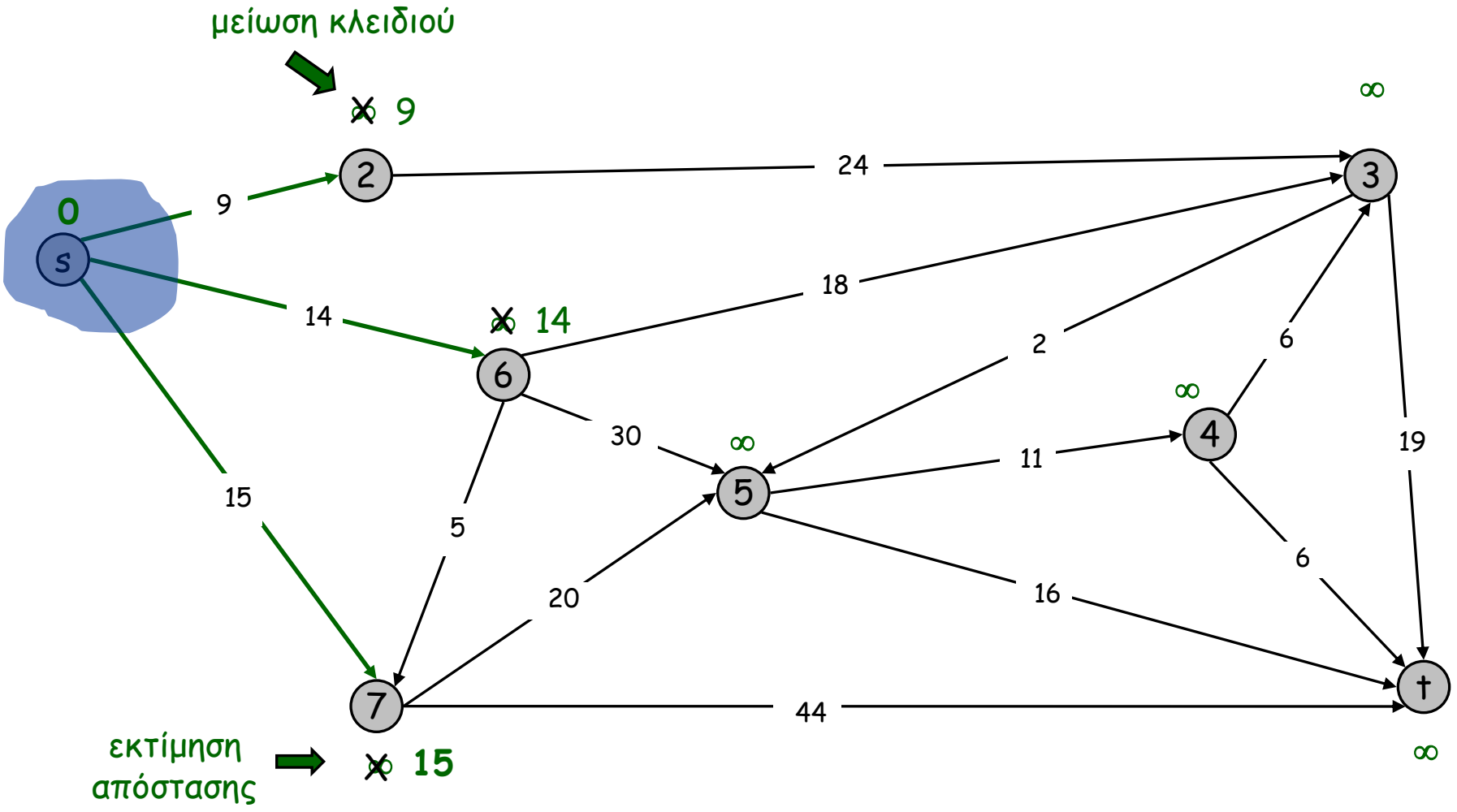
Αλγόριθμος του Dijkstra: Παράδειγμα

S = { }
PQ = { s, 2, 3, 4, 5, 6, 7, t }



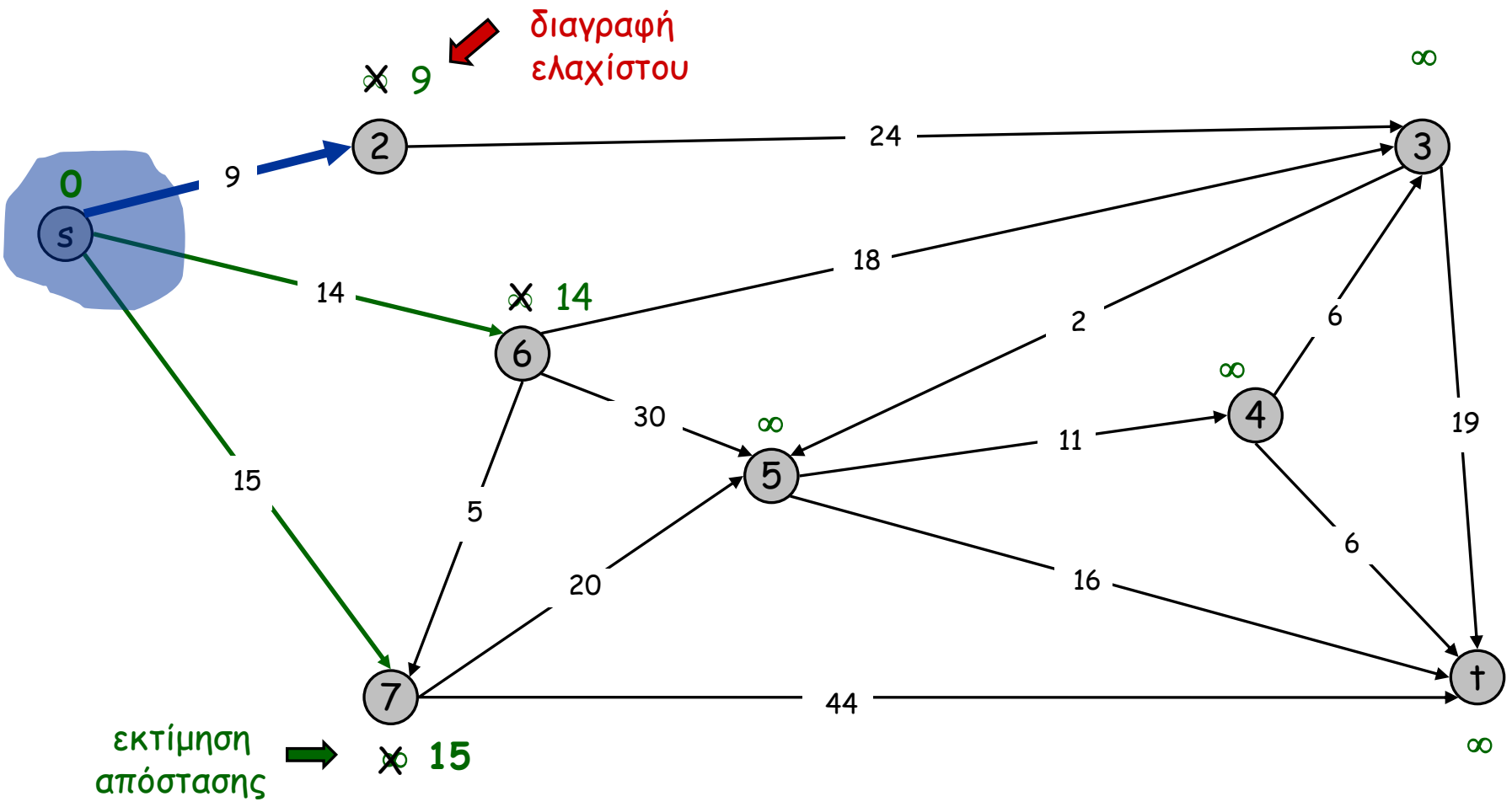
Αλγόριθμος του Dijkstra: Παράδειγμα

S = { s }
PQ = { 2, 3, 4, 5, 6, 7, t }



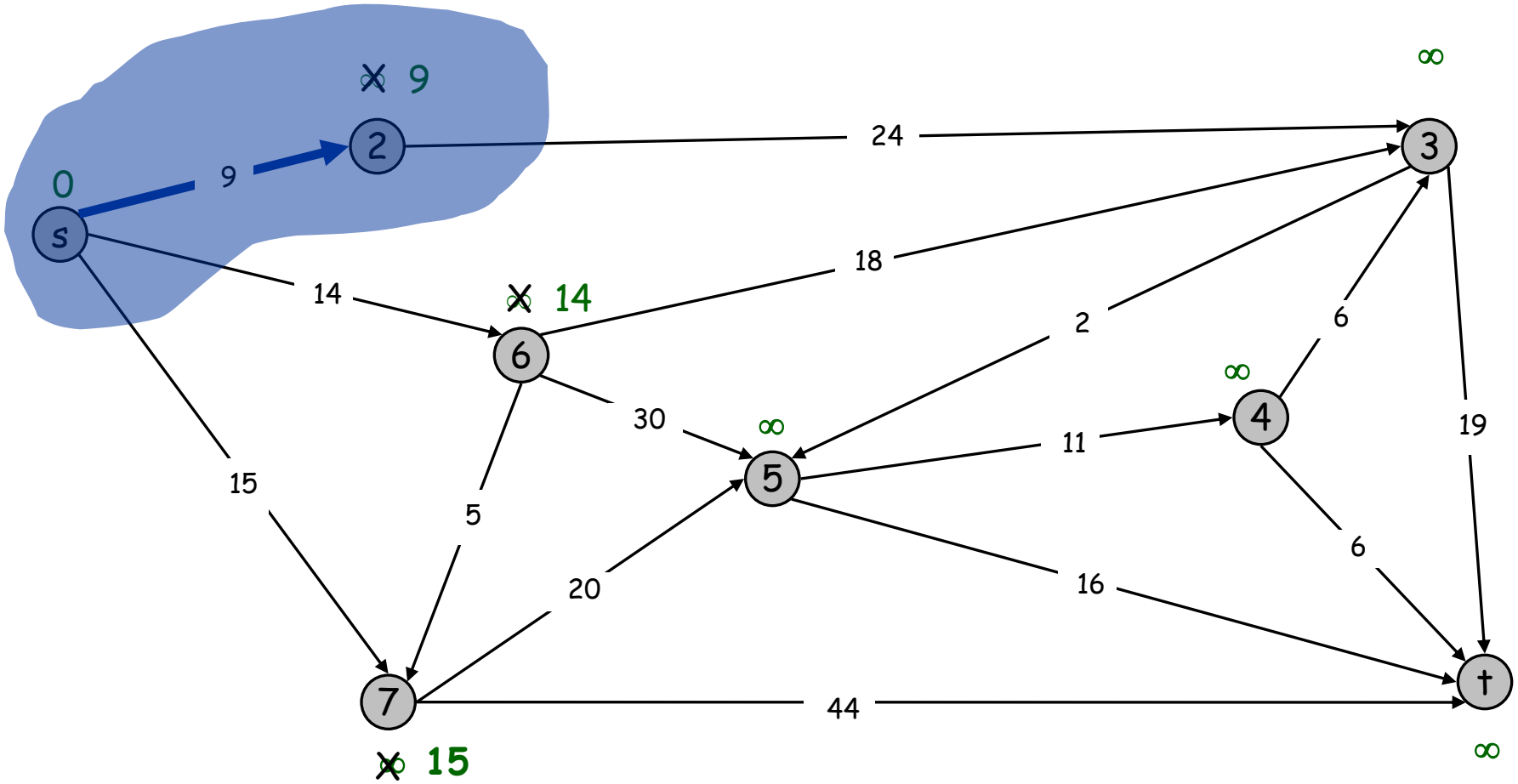
Αλγόριθμος του Dijkstra: Παράδειγμα

S = { s }
PQ = { 2, 3, 4, 5, 6, 7, t }



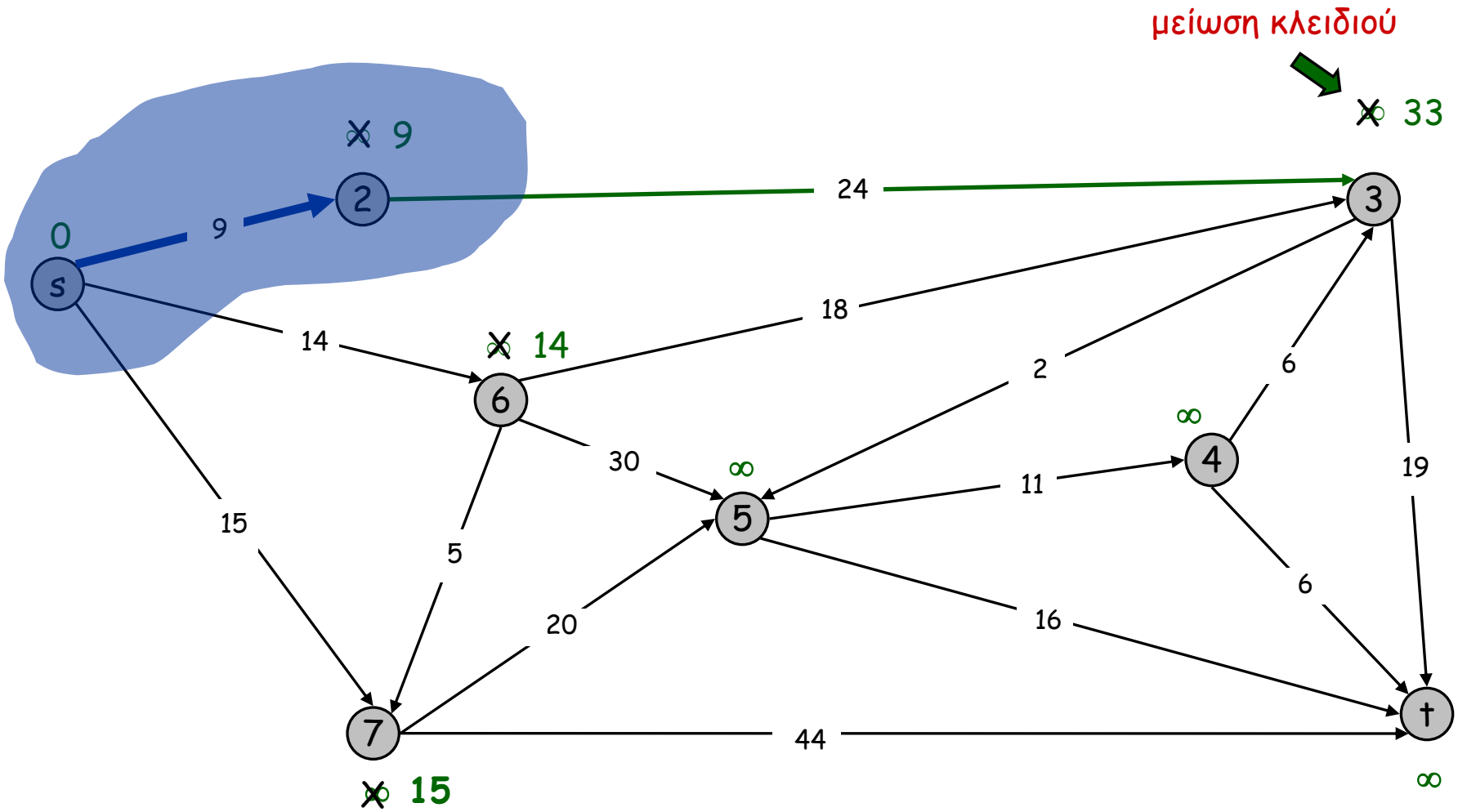
Αλγόριθμος του Dijkstra: Παράδειγμα

S = { s, 2 }
PQ = { 3, 4, 5, 6, 7, t }



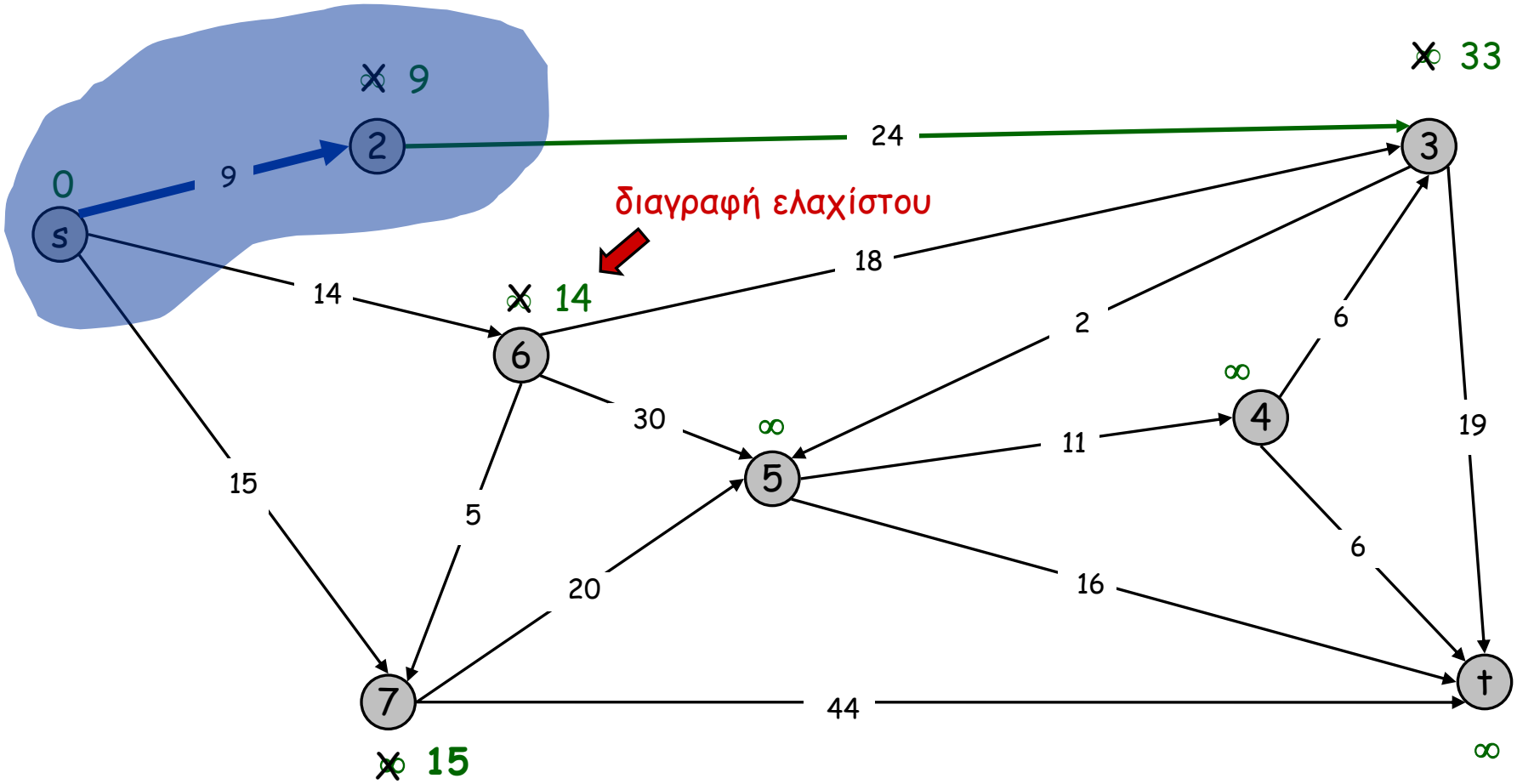
Αλγόριθμος του Dijkstra: Παράδειγμα

S = { s, 2 }
PQ = { 3, 4, 5, 6, 7, t }



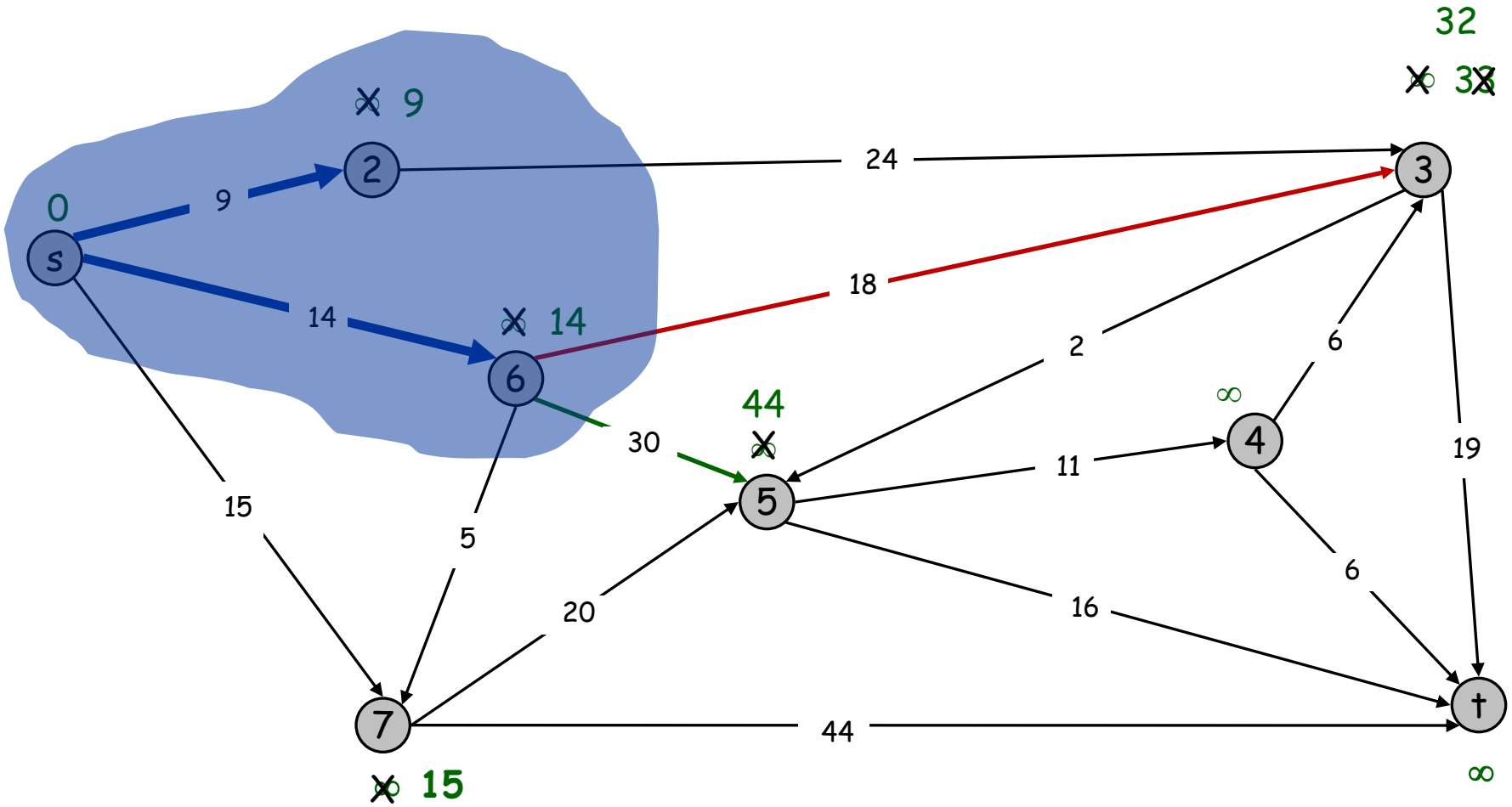
Αλγόριθμος του Dijkstra: Παράδειγμα

S = { s, 2 }
PQ = { 3, 4, 5, 6, 7, t }



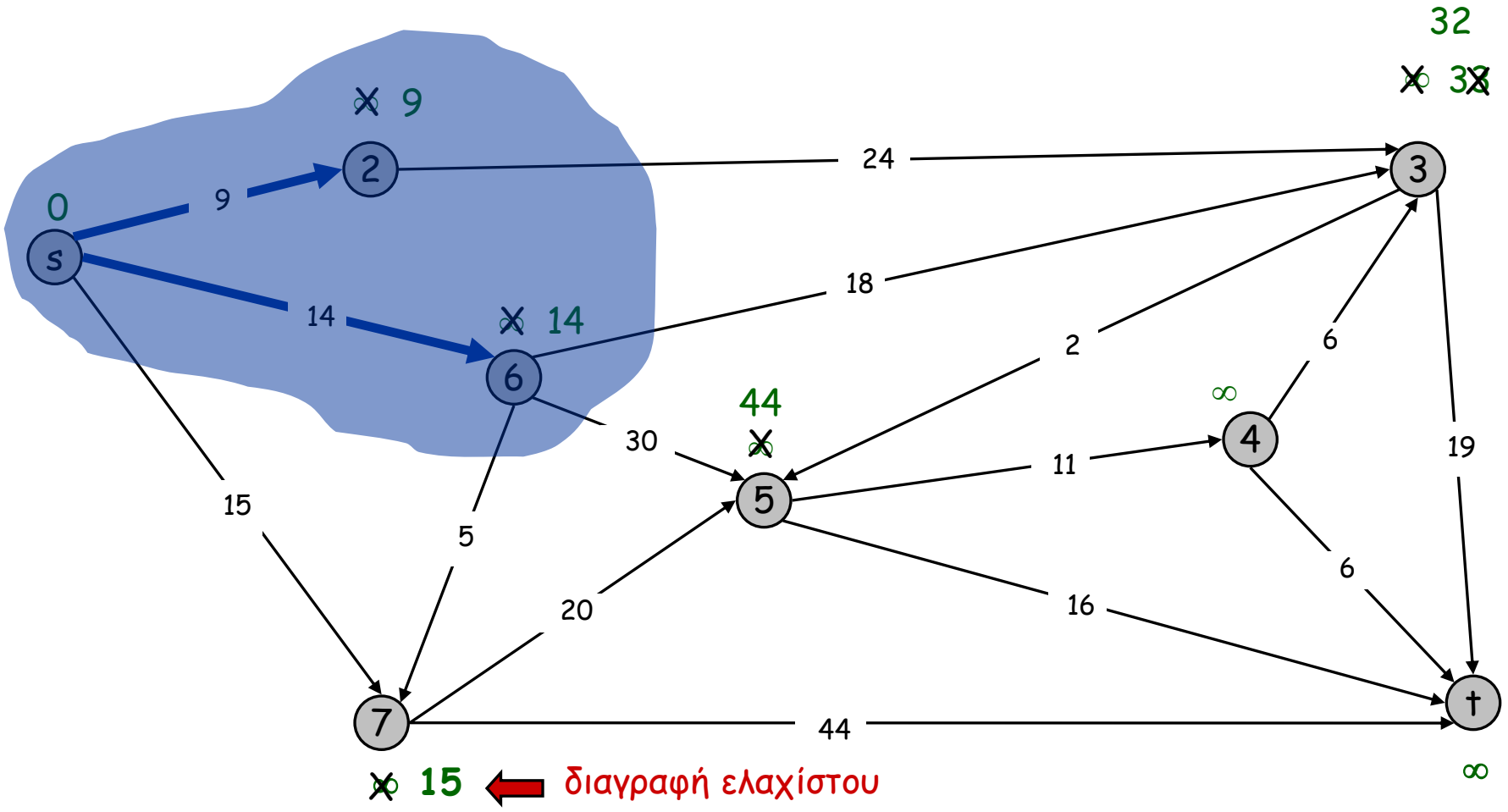
Αλγόριθμος του Dijkstra: Παράδειγμα

S = { s, 2, 6 }
PQ = { 3, 4, 5, 7, t }



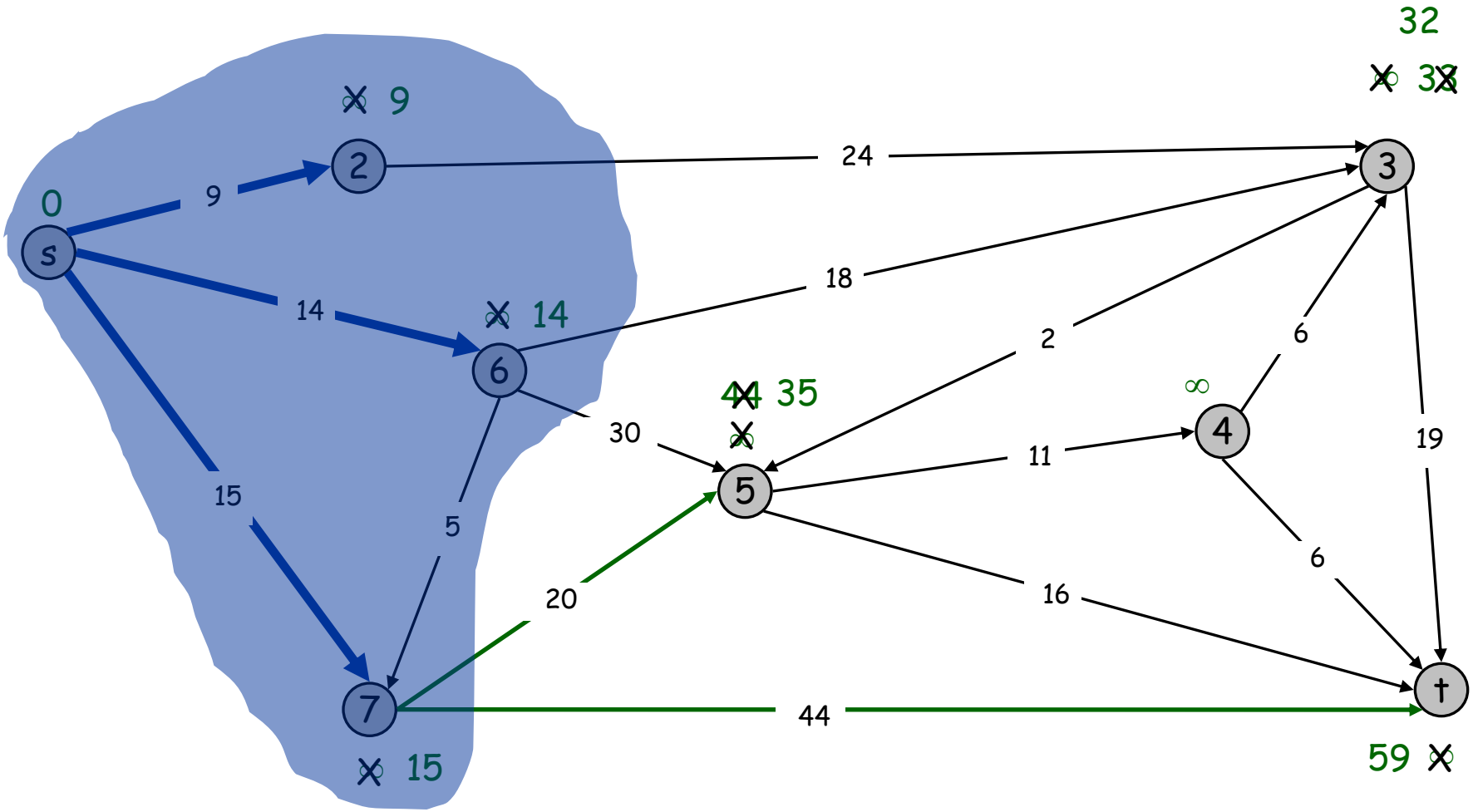
Αλγόριθμος του Dijkstra: Παράδειγμα

S = { s, 2, 6 }
PQ = { 3, 4, 5, 7, t }



Αλγόριθμος του Dijkstra: Παράδειγμα

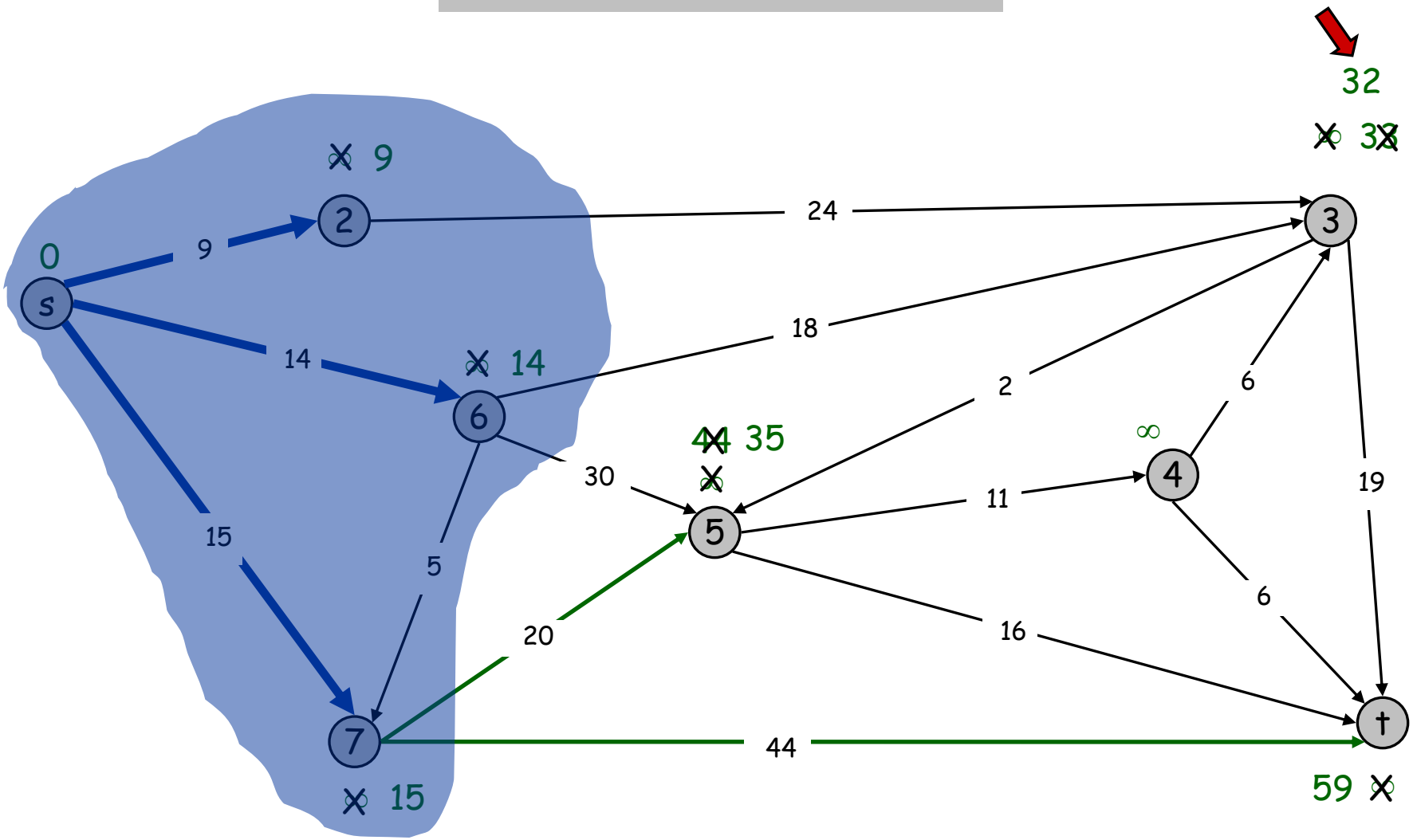
S = { s, 2, 6, 7 }
PQ = { 3, 4, 5, t }



Αλγόριθμος του Dijkstra: Παράδειγμα

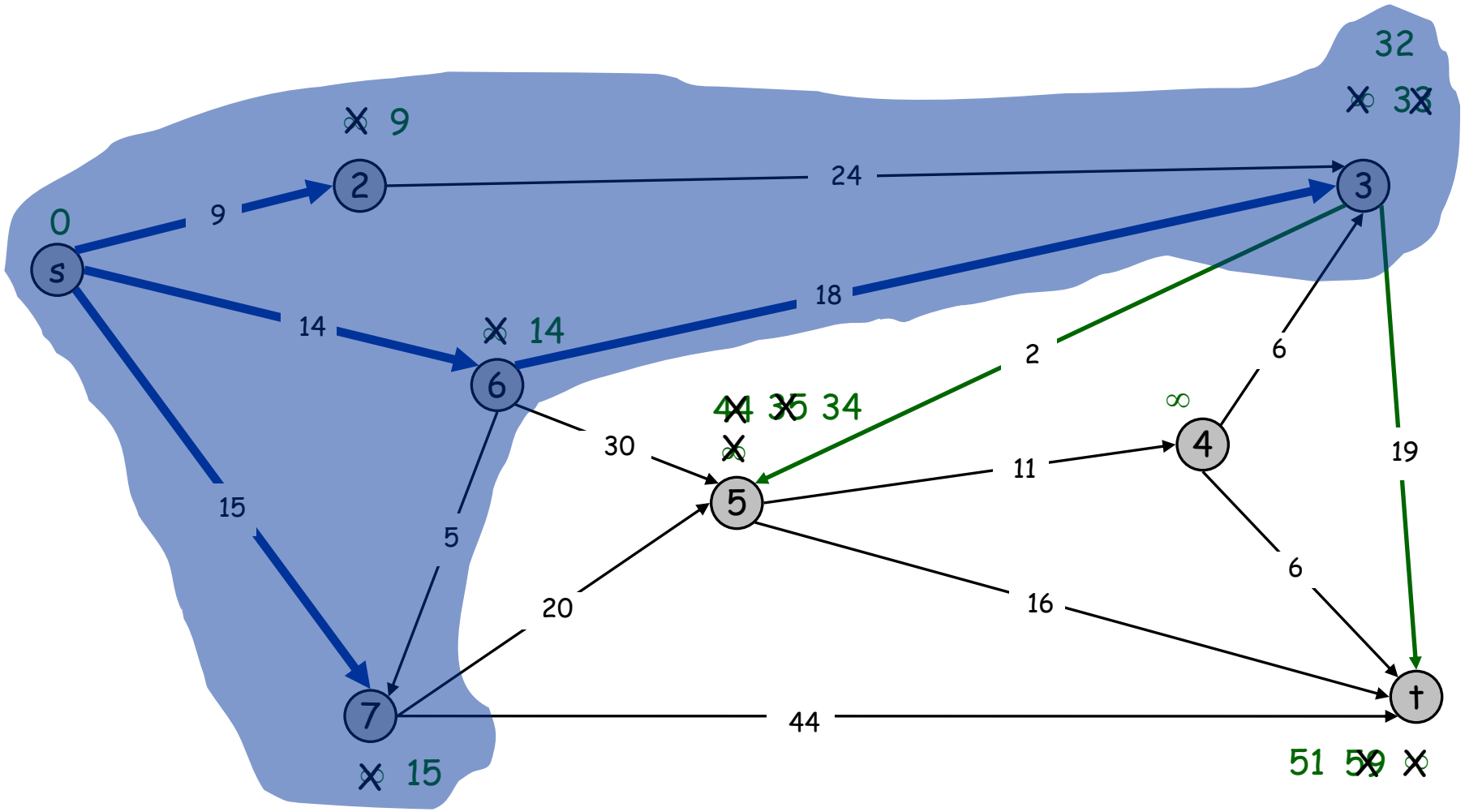
S = { s, 2, 6, 7 }
PQ = { 3, 4, 5, t }

διαγραφή ελαχίστου



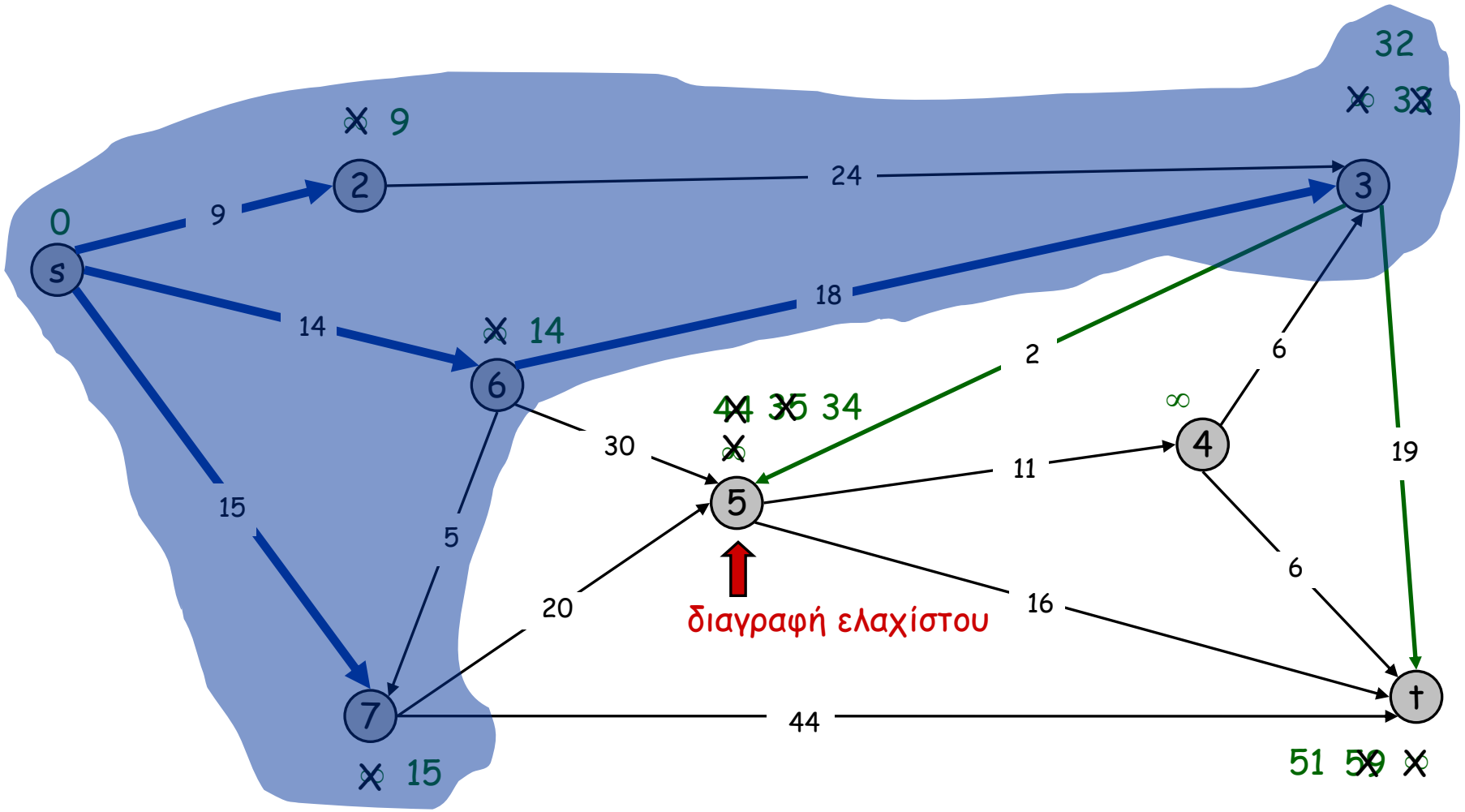
Αλγόριθμος του Dijkstra: Παράδειγμα

S = { s, 2, 3, 6, 7 }
PQ = { 4, 5, t }



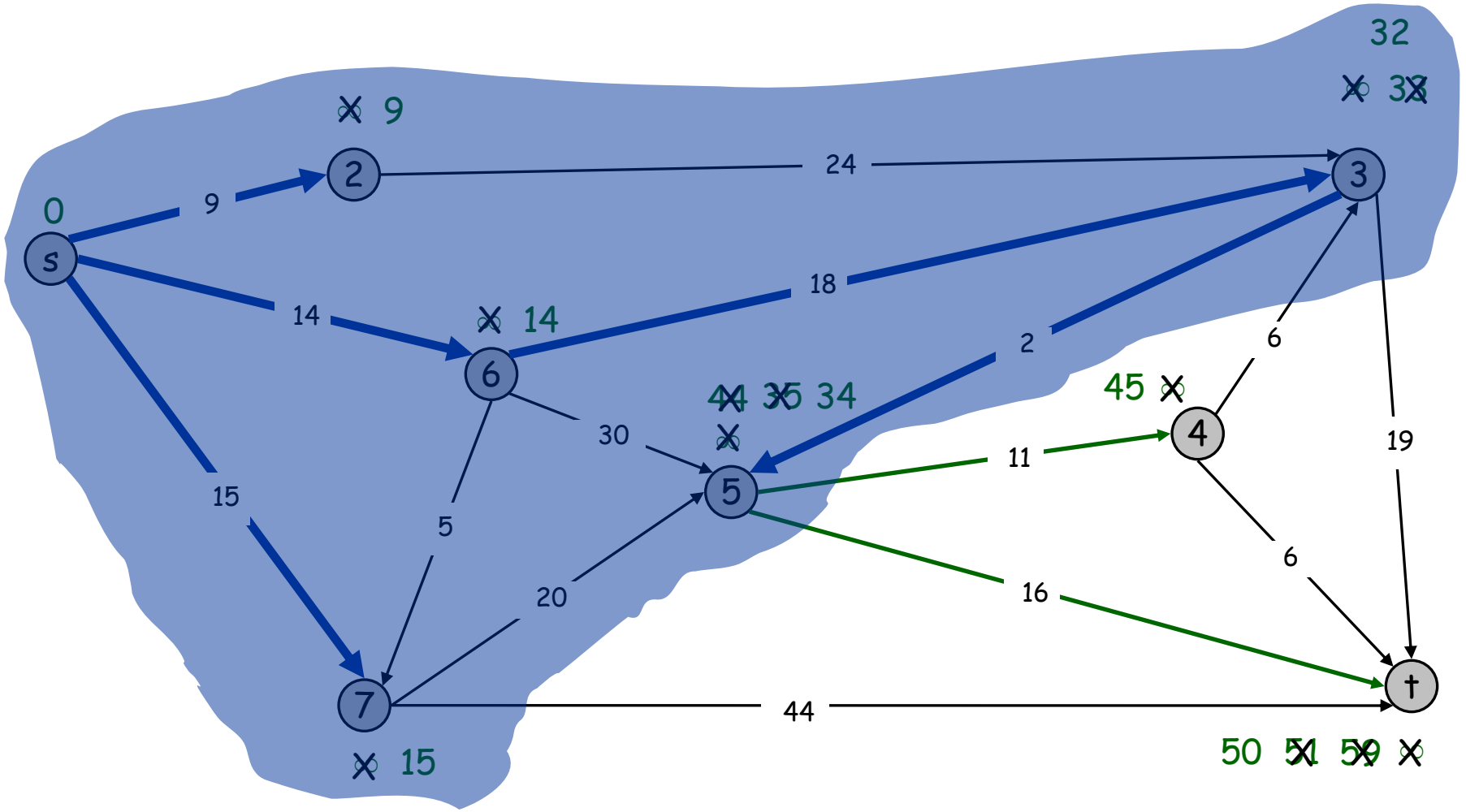
Αλγόριθμος του Dijkstra: Παράδειγμα

S = { s, 2, 3, 6, 7 }
PQ = { 4, 5, t }



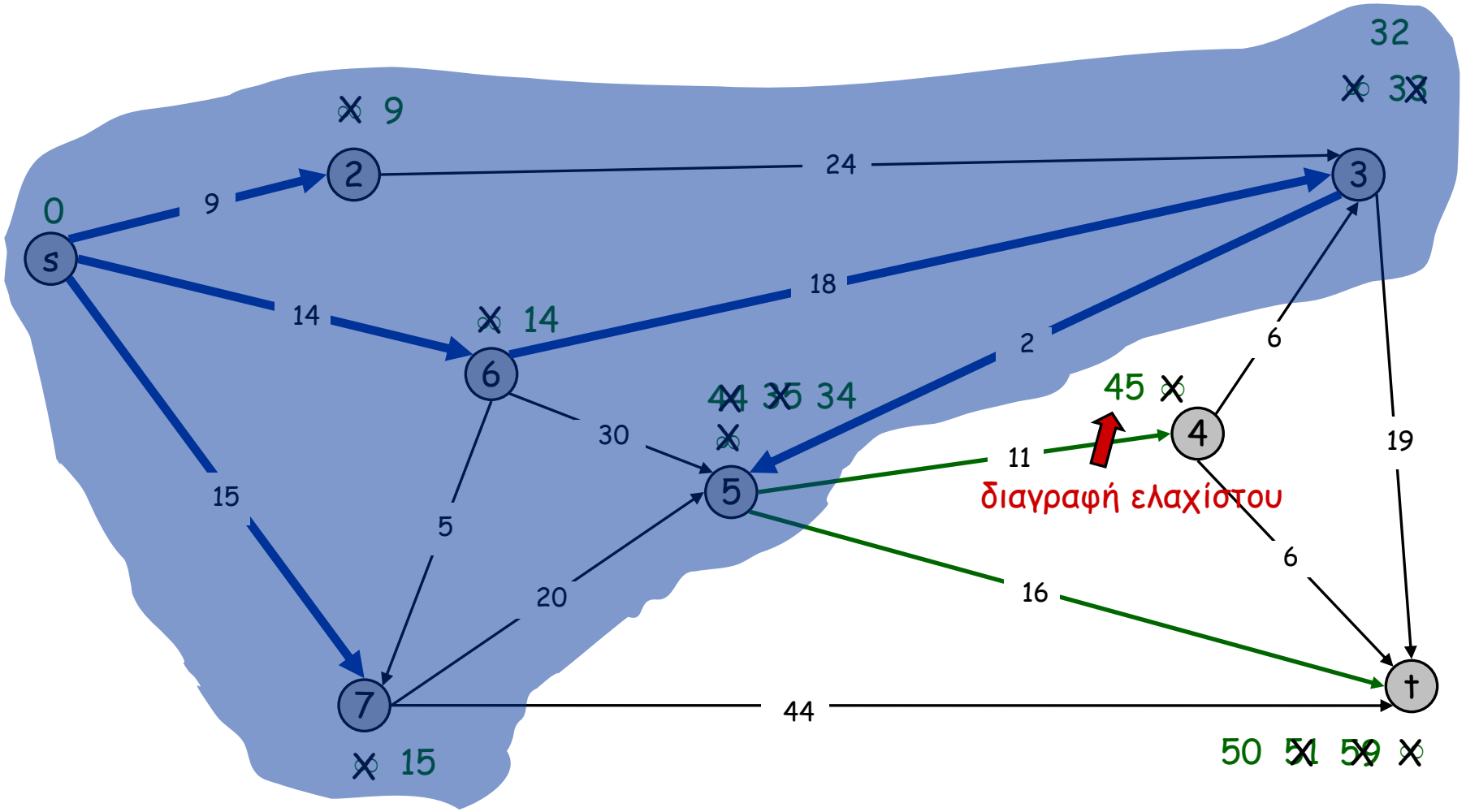
Αλγόριθμος του Dijkstra: Παράδειγμα

S = { s, 2, 3, 5, 6, 7 }
PQ = { 4, t }



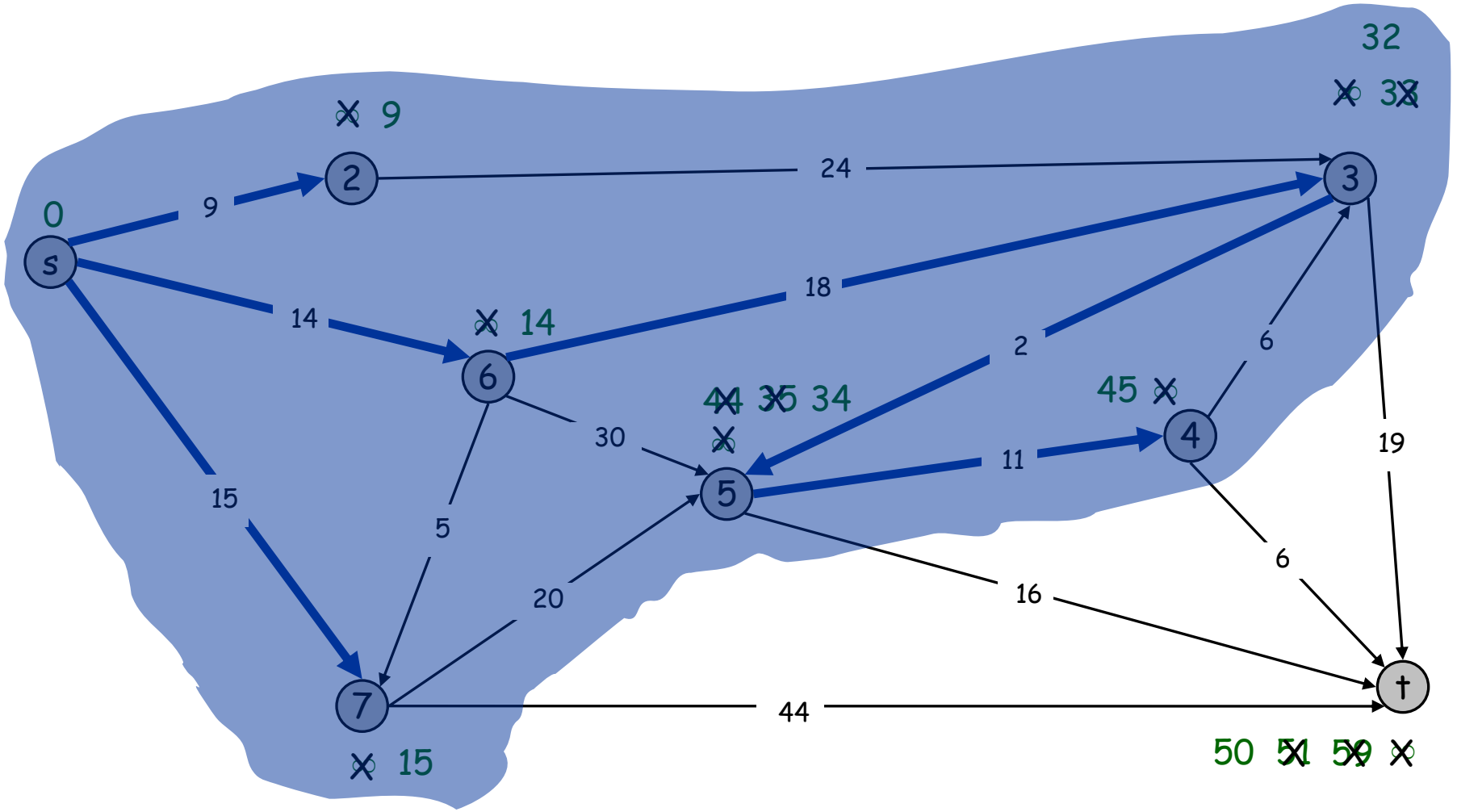
Αλγόριθμος του Dijkstra: Παράδειγμα

S = { s, 2, 3, 5, 6, 7 }
PQ = { 4, t }



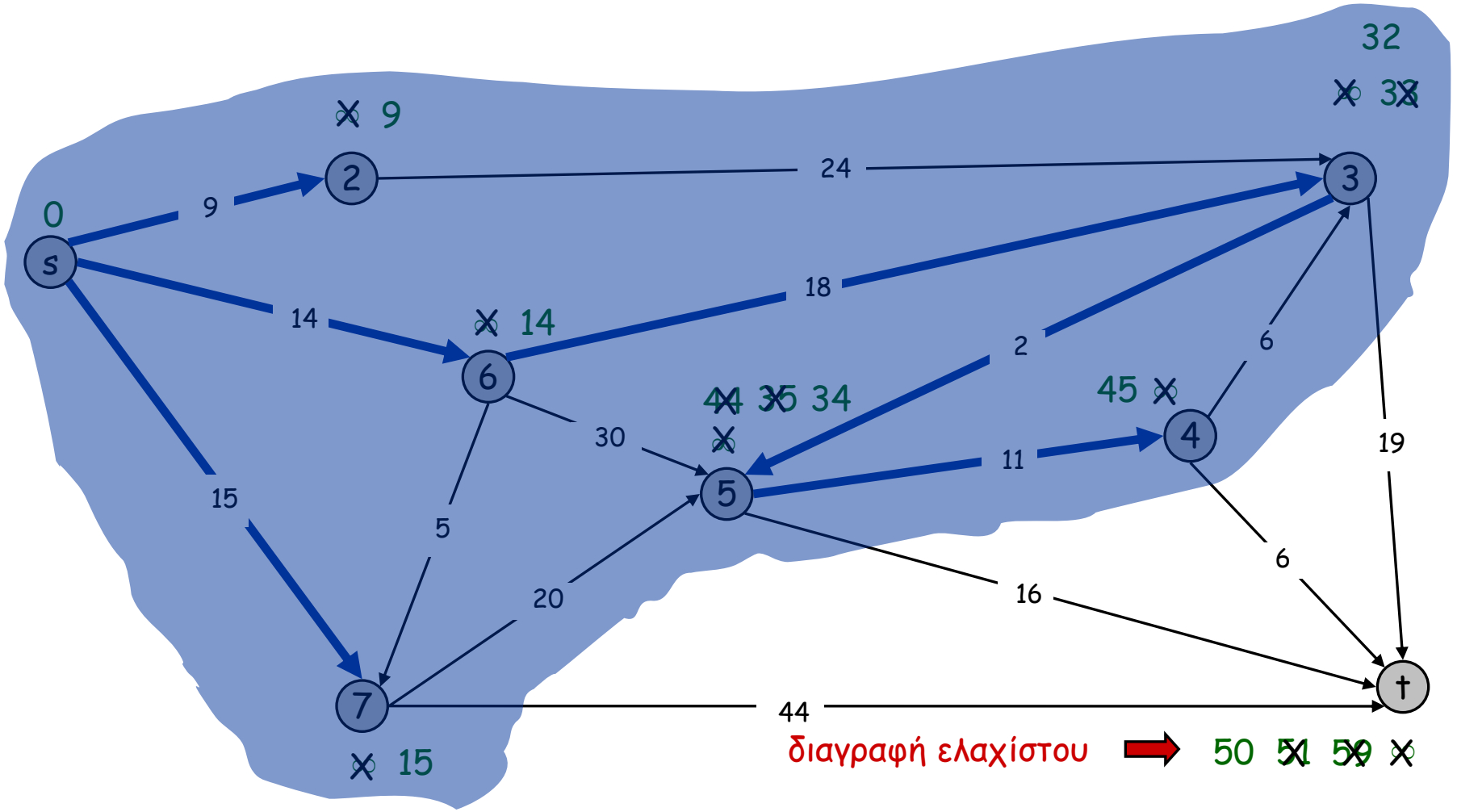
Αλγόριθμος του Dijkstra: Παράδειγμα

S = { s, 2, 3, 4, 5, 6, 7 }
PQ = { t }



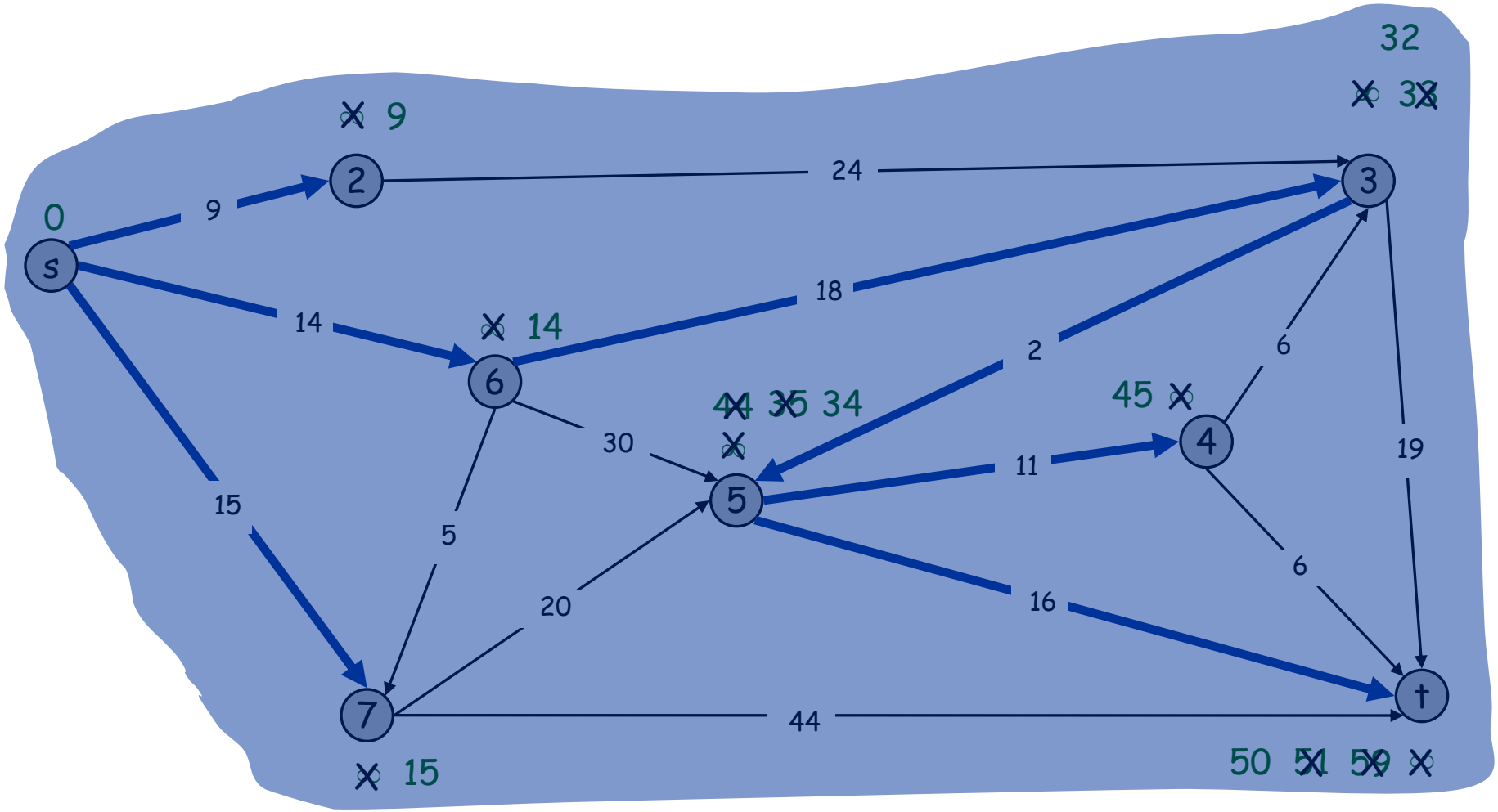
Αλγόριθμος του Dijkstra: Παράδειγμα

S = { s, 2, 3, 4, 5, 6, 7 }
PQ = { t }



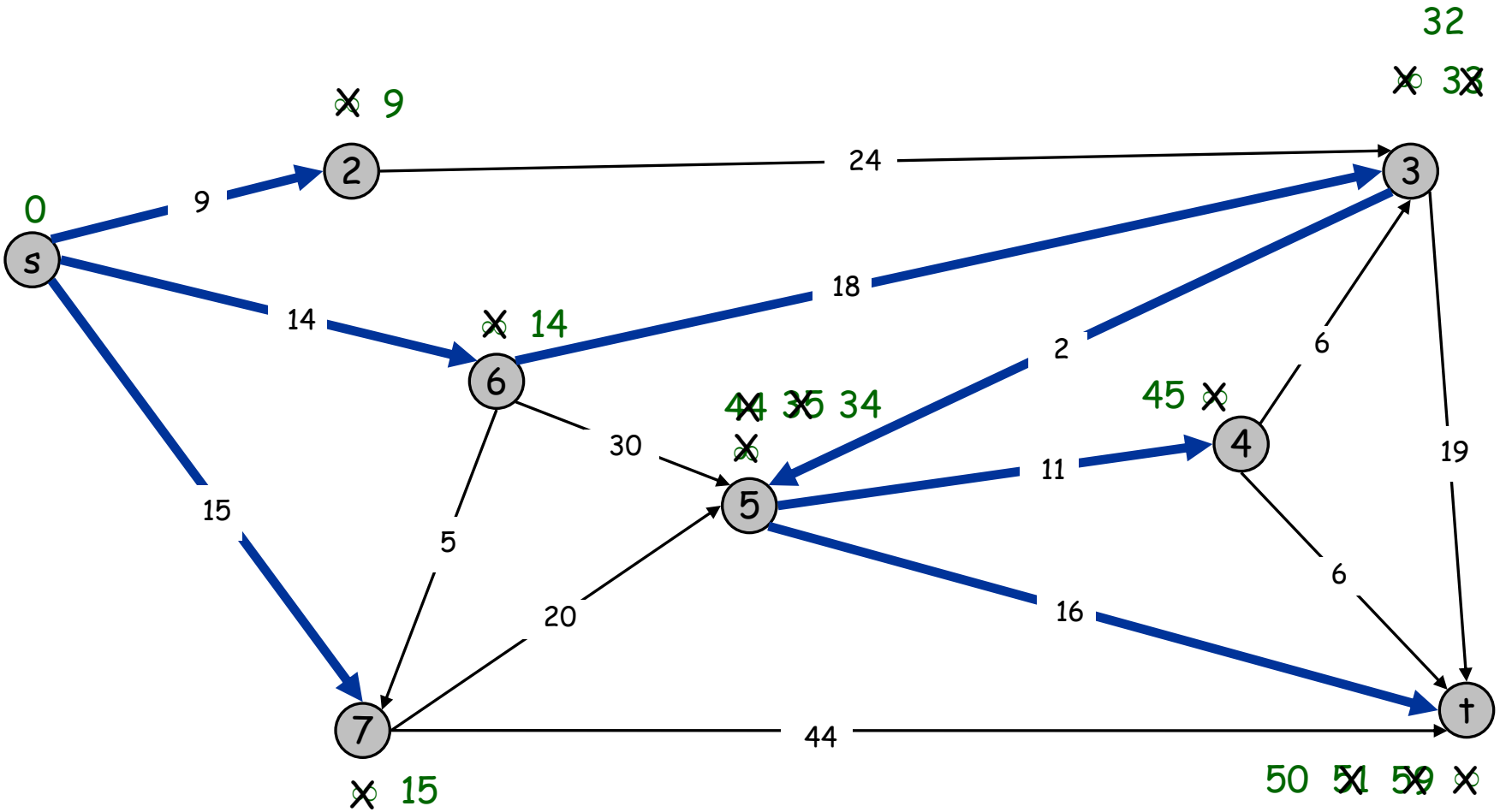
Αλγόριθμος του Dijkstra: Παράδειγμα

S = { s, 2, 3, 4, 5, 6, 7, t }
PQ = { }



Αλγόριθμος του Dijkstra: Παράδειγμα

S = { s, 2, 3, 4, 5, 6, 7, t }
PQ = { }



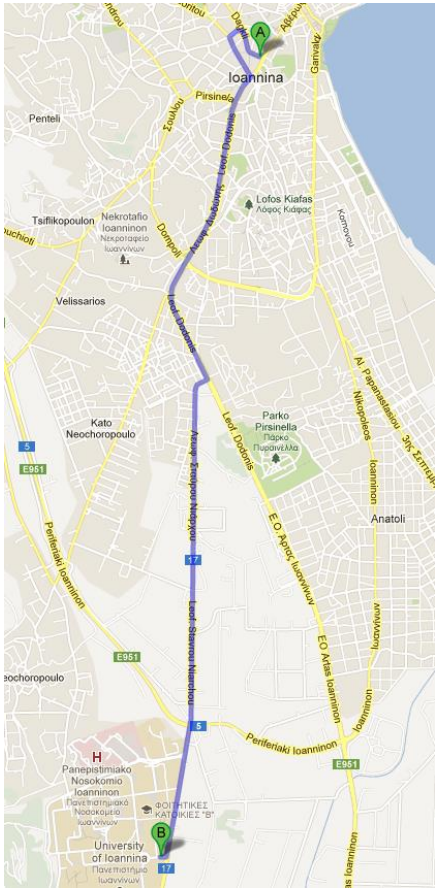
Αλγόριθμος του Dijkstra: Χρόνος Εκτέλεσης

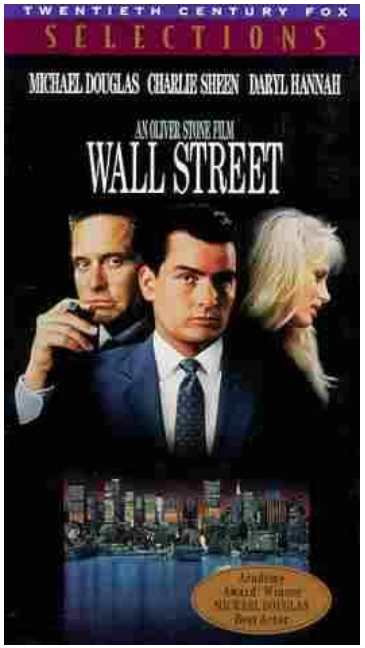
```
Q = V

while(Q δεν είναι άδεια)  {
        v = ExtractMin(Q)
        foreach e = (v,w)∈ E do
                if d(w) > d(v) + ℓe {
                     d(w) = d(v) + ℓe  // μείωση προτεραιότητας
                     ChangeKey(Q,w,d[w])  // μείωση κλειδιού
                }
        }
```

- Βρόγχος while: n-1 επαναλήψεις, κάθε φορά αφαιρείται ένας κόμβος
- ExtractMin & ChangeKey: $O(\log n)$ λόγω ουράς προτεραιότητας
- Πόσο συχνά εκτελείται η ChangeKey για μια ακμή (v,w);
  - όταν επιλέγουμε το άκρο w στο S (δηλαδή, w = ExtractMin(Q)) τότε εκτελείται μια φορά για κάθε μια ακμή.

- Επομένως: $O(n)$ αρχικοποίηση + $O(n)$ επαναλήψεις
  + $O(n)$ Extactmin + $O(m)$ ChangeKey = $O(m \log n)$
  (διότι $m > n$)

Καλή Μελέτη!!


A
B
Ioannina
Pirsinea
Penteli
Lofos Kiafas
Λόφος Κιάφας
Tsiflikopoulon
Nekrotafio Ioanninon
Νεκροταφείο Ιωαννίνων
Velissarios
Kato Neochoropoulo
Parko Pirsinella
Πάρκο Πυρσινέλλα
Anatoli
Periferiaki Ioanninon
E951
17
5
H
Panepistimiako Nosokomio Ioanninon
Πανεπιστημιακό Νοσοκομείο Ιωαννίνων
ΦΟΙΤΗΤΙΚΕΣ ΚΑΤΟΙΚΙΕΣ "Β"
University of Ioannina
Πανεπιστήμιο Ιωαννίνων


TWENTIETH CENTURY FOX
SELECTIONS
MICHAEL DOUGLAS CHARLIE SHEEN DARYL HANNAH
AN OLIVER STONE FILM
WALL STREET