Σχεδίαση και Ανάλυση Αλγορίθμων

Ενότητα 8.0

Αλγοριθμική Τεχνική Απληστίας

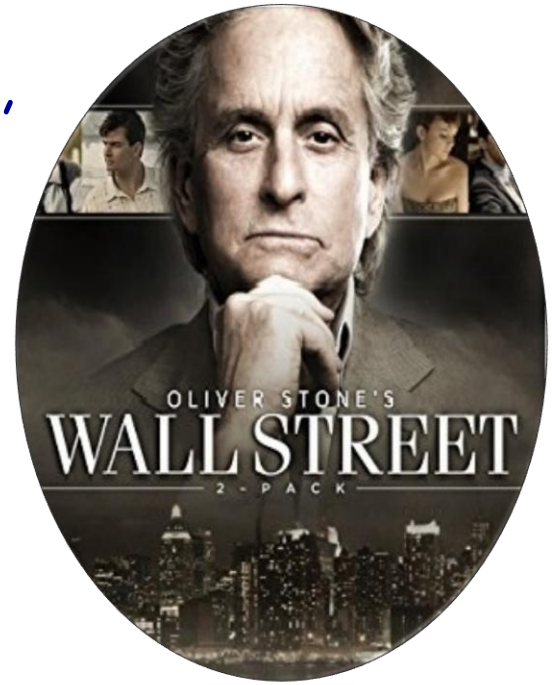
Ελάχιστες Διαδρομές – Dijkstra, Bellman-Ford

Ελάχιστα Συνδετικά Δένδρα – Kruskal, Prim

Σταύρος Δ. Νικολόπουλος | 2016-17
Τμήμα Μηχανικών Η/Υ & Πληροφορικής
Πανεπιστήμιο Ιωαννίνων
Webpage: www.cs.uoi.gr/~stavros

Τεχνική Απληστίας

Στην εμβληματική ταινία Wall Street του 1980, ο Michael Douglas σηκώνεται μπροστά σε ένα πλήθος μετόχων και διακηρύσσει...



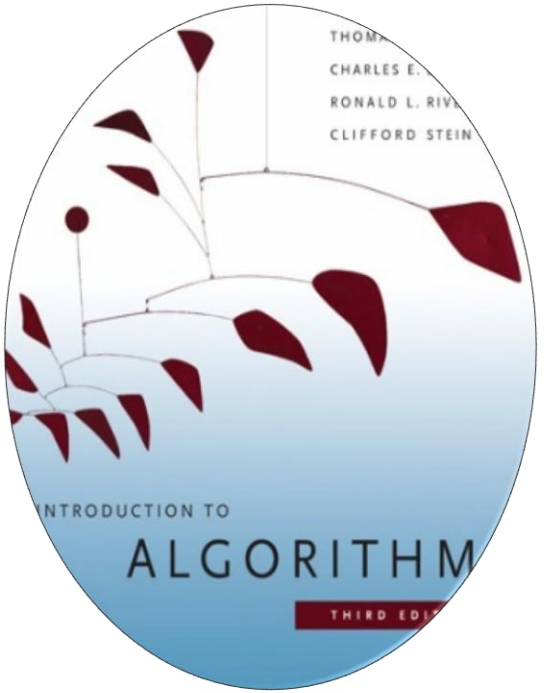
Η απληστία... είναι καλή !

Η απληστία είναι σωστή !!

Η απληστία αποδίδει !!!

Τεχνική Απληστίας

Στη σχεδίαση αλγορίθμων…



Η απληστία… είναι καλή ?

Η απληστία είναι σωστή ??

Η απληστία αποδίδει ???

Τεχνική Απληστίας

Δεν είναι εύκολο να ορίσουμε με ακρίβεια τι εννοούμε με τον όρο άπληστος αλγόριθμος (greedy algorithm).

Ένας αλγόριθμος είναι άπληστος

- ✓ όταν δομεί μια λύση με μικρά βήματα,
- ✓ παίρνοντας «μυωπικές αποφάσεις» σε κάθε βήμα,
- ✓ με στόχο τη βελτιστοποίηση κάποιου κριτηρίου.

Τεχνική Απληστίας

Ένας άπληστος αλγόριθμος ακολουθεί την ευρετική (heuristic) επίλυση προβλημάτων…

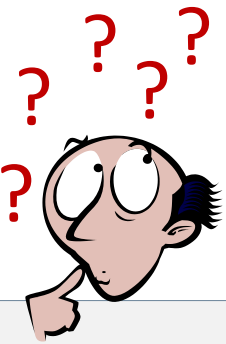
- ✓ κάνοντας σε κάθε βήμα μια τοπική βέλτιστη επιλογή,
- ✓ με την «ελπίδα» τελικά να βρει μια ολική βέλτιστη λύση του προβλήματος !!!

Θεμελιώδες χαρακτηριστικό :

- Σε κάθε βήμα ένας άπληστος αλγόριθμος κάνει μια τοπικά βέλτιστη επιλογή και δεν «ξανασκέφτεται» αυτή την επιλογή του !!!

Τεχνική Απληστίας

Εύλογο Ερώτημα !!!



Κάνοντας μια τοπική βέλτιστη επιλογή σε κάθε βήμα, μπορούμε να επιτύχουμε ένα ολικό βέλτιστο ?

δηλ. μια βέλτιστη λύση του προβλήματός μας ?

δεδομένου ότι η άπληστη τεχνική δεν επιτρέπει να αναθεωρήσουμε επιλογές που έχουμε ήδη κάνει !!!

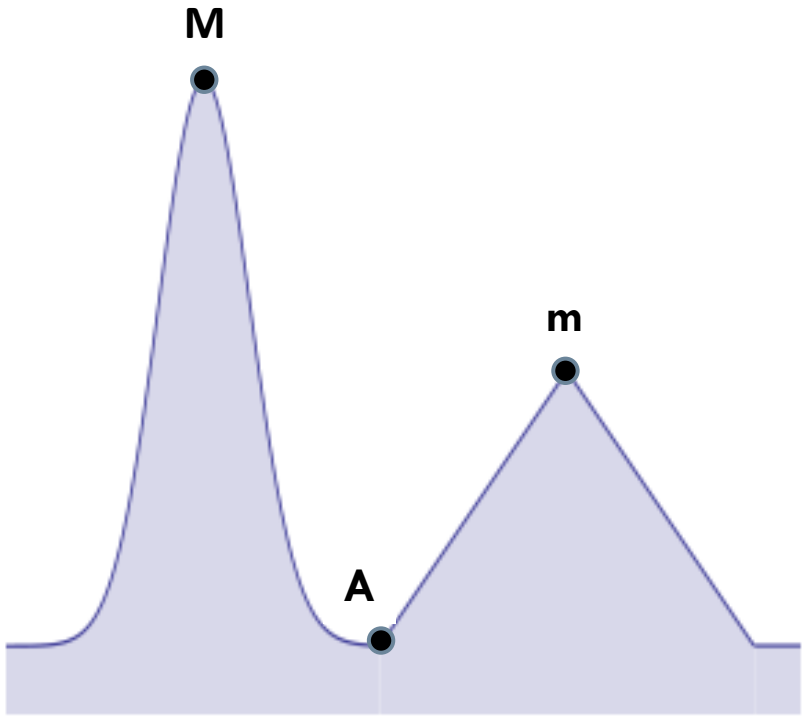
Τεχνική Απληστίας

Σε πολλά προβλήματα, η άπληστη στρατηγική αποτυγχάνει να δώσει βέλτιστη λύση !!!

Για παράδειγμα, στην κατάκτηση του υψηλότερου σημείου μιας οροσειράς!

Τοπική βέλτιστη επιλογή :

«σε κάθε επανάληψη επίλεξε να μεταβαίνεις σε υψηλότερη θέση, ξεκινώντας από το σημείο Α» !!!



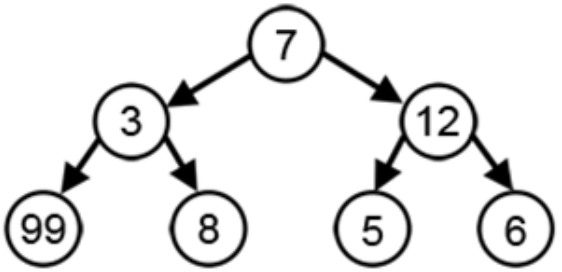
Απληστία : σημείο οροσειράς **m**

Βέλτιστο: σημείο οροσειράς **M**

Τεχνική Απληστίας

Σε πολλά προβλήματα, η άπληστη στρατηγική αποτυγχάνει να δώσει βέλτιστη λύση !!!

Για παράδειγμα, στην εύρεση μιας max διαδρομής σε ένα δένδρο !



Τοπική βέλτιστη επιλογή :

«σε κάθε επανάληψη επίλεξε τον κόμβο με το μεγαλύτερο κόστος και πρόσθεσέ τον στην διαδρομή» !!!

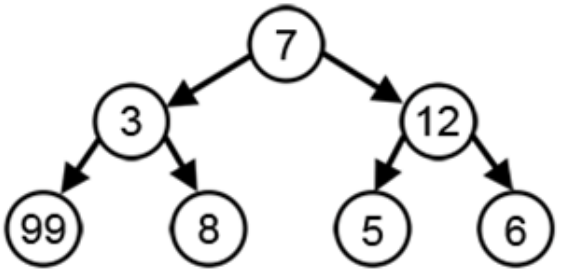
Απληστία : κόστος διαδρομής **25**

Βέλτιστο: κόστος διαδρομής **109**

Τεχνική Απληστίας

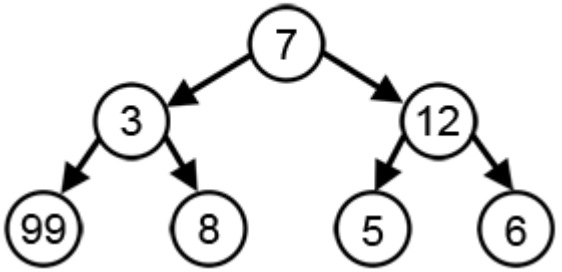
Σε πολλά προβλήματα, η άπληστη στρατηγική αποτυγχάνει να δώσει βέλτιστη λύση !!!

Για παράδειγμα, στην εύρεση μιας max διαδρομής σε ένα δένδρο !



Τοπική βέλτιστη επιλογή :

«σε κάθε επανάληψη επίλεξε τον κόμβο με το μεγαλύτερο κόστος και πρόσθεσέ τον στην διαδρομή» !!!



Απληστία : κόστος διαδρομής **25**

Βέλτιστο: κόστος διαδρομής **109**

Τεχνική Απληστίας

Σε άλλα προβλήματα, η άπληστη τεχνική δίδει βέλτιστη λύση !!!

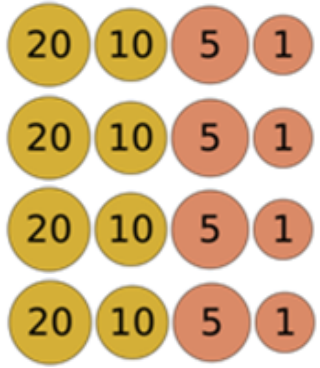
Όπως, για παράδειγμα, στην ανταλλαγή ενός νομίσματος αξίας 38 (Euro, USD, …) με το ελάχιστο πλήθος κερμάτων αξίας 1, 5, 10, 20.

Τοπική βέλτιστη επιλογή :

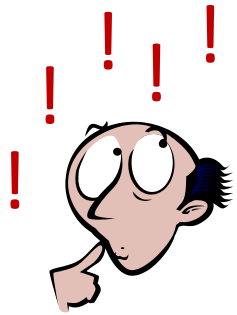
«σε κάθε επανάληψη επίλεξε το κέρμα με την μεγαλύτερη αξία που δεν ξεπερνάει το ποσό που θα πληρωθεί» !

Απληστία = Βέλτιστο : 20 10 5 1 1 1

38 – 20 = 18
18 – 10 = 8
8 – 5 = 3
3 – 1 = 2
2 – 1 = 1
1 – 1 = 0



Τεχνική Απληστίας



Σε άλλα προβλήματα, η άπληστη τεχνική δίδει βέλτιστη λύση !!!

Τέλεια !!!... Η απληστία δουλεύει καλά !!!

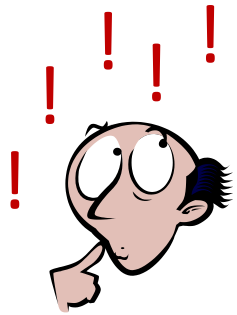
Νόμισμα αξίας: 34 ¢

Κέρματα : 1, 5, 10, 25, 100

Απληστία : 25, 5, 1, 1, 1, 1

Βέλτιστο: 25, 5, 1, 1, 1, 1

Τεχνική Απληστίας



Σε άλλα προβλήματα, η άπληστη τεχνική δίδει βέλτιστη λύση !!!

Τέλεια !!!... Η απληστία δουλεύει καλά !!!

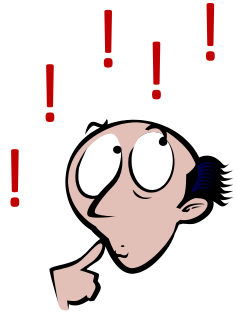
Νόμισμα αξίας: 2.89 $

Κέρματα : 1, 5, 10, 25, 100

Απληστία : 2x100, 3x25, 10, 4x1

Βέλτιστο: 2x100, 3x25, 10, 4x1

Τεχνική Απληστίας



Σε άλλα προβλήματα, η άπληστη τεχνική δίδει βέλτιστη λύση !!!

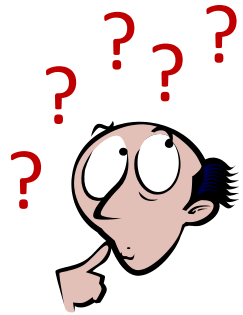
Τέλεια !!!... Η απληστία δουλεύει καλά !!!

Γραμματόσημα αξίας: 140 $

Γραμματόσημα : 1, 10, 21, 34, 70, 100, 350, 1225, 1500

Απληστία : 100, 34, 1, 1, 1, 1, 1, 1

Τεχνική Απληστίας



Σε άλλα προβλήματα, η άπληστη τεχνική δίδει βέλτιστη λύση !!!

Τέλεια !!!... Η απληστία δουλεύει καλά !!!

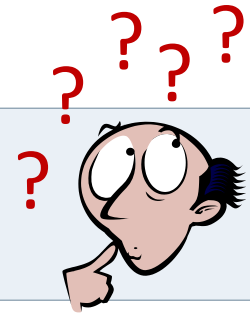
Γραμματόσημα αξίας: 140 $

Γραμματόσημα : 1, 10, 21, 34, 70, 100, 350, 1225, 1500

Απληστία : 100, 34, 1, 1, 1, 1, 1, 1

Βέλτιστο: 70, 70

Τεχνική Απληστίας



Η απληστία… θέλει μεγάλη προσοχή !!!

- ✓ Μεταφορικά, με την άπληστη τεχνική είναι σαν να «περπατάμε» σε «ναρκοθετημένο» πεδίο !!!
- ✓ Όλα φαίνονται καλά στην επιφάνεια, αλλά εάν λίγο δεν προσέξουμε οι κρυμμένες λεπτομέρειες μπορεί να πυροδοτήσουν την «έκρηξη» !!!...

δηλ. την αποτυχία!!!... που είναι η κατάρρευση της ιδέας του αλγορίθμου μας !!!

Τεχνική Απληστίας

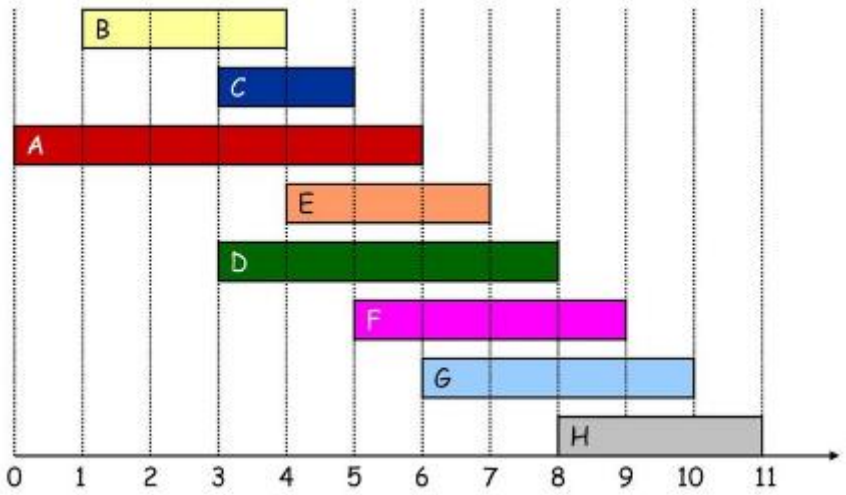
- Είναι εύκολο να επινοήσουμε άπληστους αλγορίθμους για σχεδόν οποιοδήποτε πρόβλημα !!!

Η πρόκληση είναι :

- Η εύρεση περιπτώσεων στις οποίες οι άπληστοι αλγόριθμοι λειτουργούν καλά, και
- η απόδειξη ότι πράγματι αποδίδουν !!

Ας δούμε με ένα απλό παράδειγμα πόσο εύκολα μπορεί κάποιος να παρασυρθεί σε λάθος άπληστες προσεγγίσεις !!!

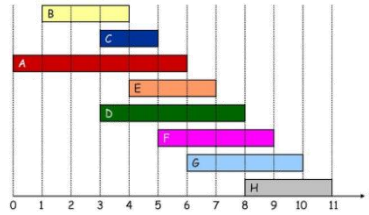
Χρονοπρογραμματισμός Διαστημάτων



Παράδειγμα

- Ας εξετάσουμε το πρόβλημα του Χρονοπρογραμματισμού Διαστημάτων (Interval Scheduling Problem).

Χρονοπρογραμματισμός Διαστημάτων



- Στο πρόβλημα του Χρονοπρογραμματισμού Διαστημάτων έχουμε :

Ένα πόρο : Για παράδειγμα,

- ✓ έναν υπερυπολογιστή, ένα ηλεκτρονικό μικροσκόπιο, ή
- ✓ μια αίθουσα διδασκαλίας,

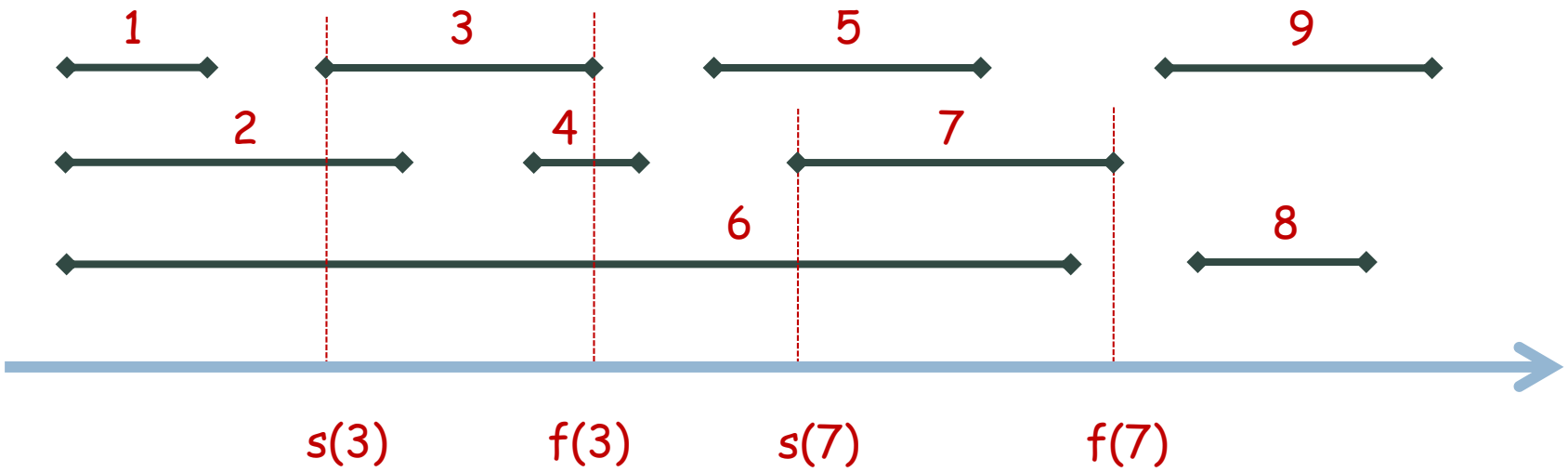
και πολλοί ζητούν να τον χρησιμοποιήσουν για κάποιες χρονικές περιόδους (ο πόρος μπορεί να χρησιμοποιηθεί το πολύ από ένα άτομο την φορά).

Αιτήματα : Θέλω να χρησιμοποιήσω τον πόρο από την ώρα s_i έως την ώρα f_i.

Στόχος : Έχοντας ένα σύνολο n αιτημάτων $A = \{1, 2, \ldots, n\}$, θέλουμε να μεγιστοποιήσουμε το πλήθος των αιτημάτων που θα γίνουν αποδεκτά !

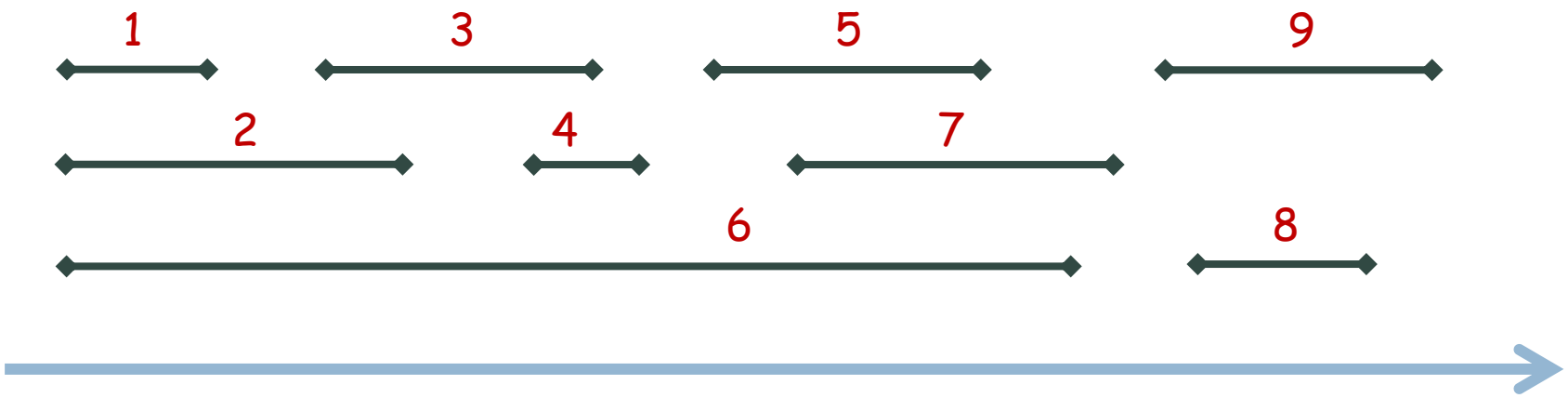
Χρονοπρογραμματισμός Διαστημάτων

- Έστω ένα σύνολο αιτημάτων $A = \{1, 2, \ldots, n\}$.
- Το αίτημα i αντιστοιχεί σε ένα χρονικό διάστημα $[s(i), f(i)]$.
- Μοντελοποίηση:



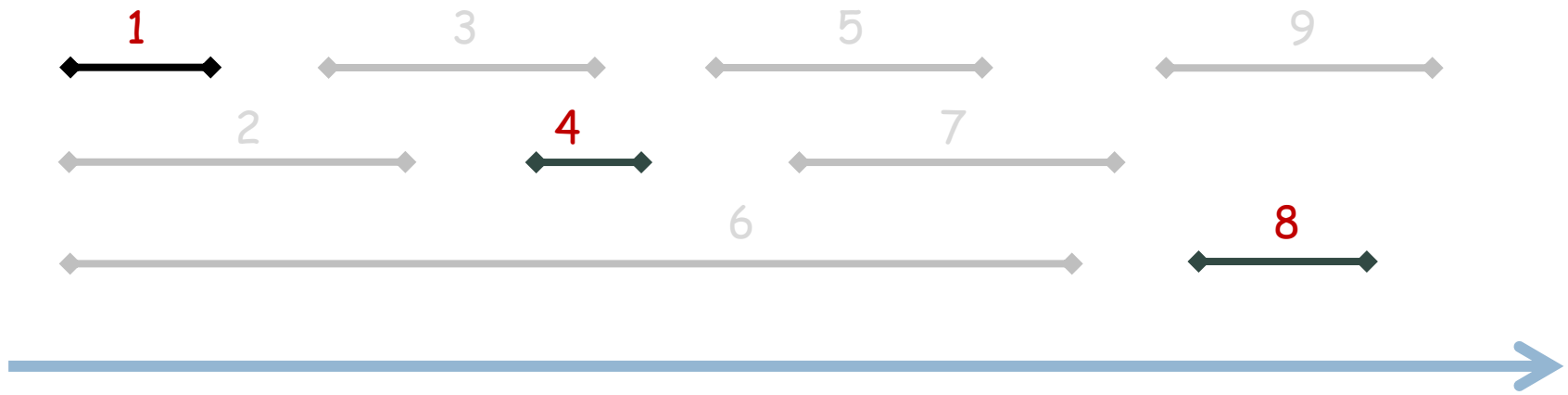
Χρονοπρογραμματισμός Διαστημάτων

- Ένα σύνολο αιτημάτων S λέγεται *συμβατό* (compatible) εάν *ανά δύο τα αιτήματά του δεν επικαλύπτονται χρονικά*.
- Στόχος είναι να υπολογίσουμε ένα *συμβατό υποσύνολο με το μέγιστο πλήθος διαστημάτων*, το οποίο ονομάζουμε *βέλτιστο* (optimal).



Χρονοπρογραμματισμός Διαστημάτων

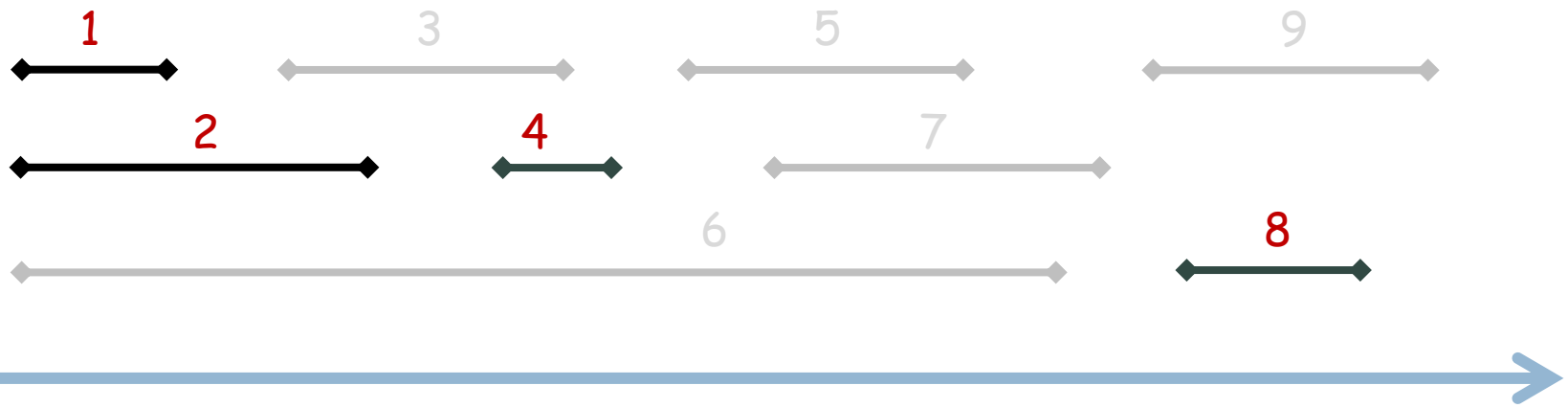
- Ένα σύνολο αιτημάτων S λέγεται *συμβατό* (compatible) εάν *ανά δύο τα αιτήματά του δεν επικαλύπτονται χρονικά*.
- Στόχος είναι να υπολογίσουμε ένα *συμβατό υποσύνολο με το μέγιστο πλήθος διαστημάτων*, το οποίο ονομάζουμε *βέλτιστο* (optimal).



✓ Συμβατό: $S = \{1, 4, 8\}$

Χρονοπρογραμματισμός Διαστημάτων

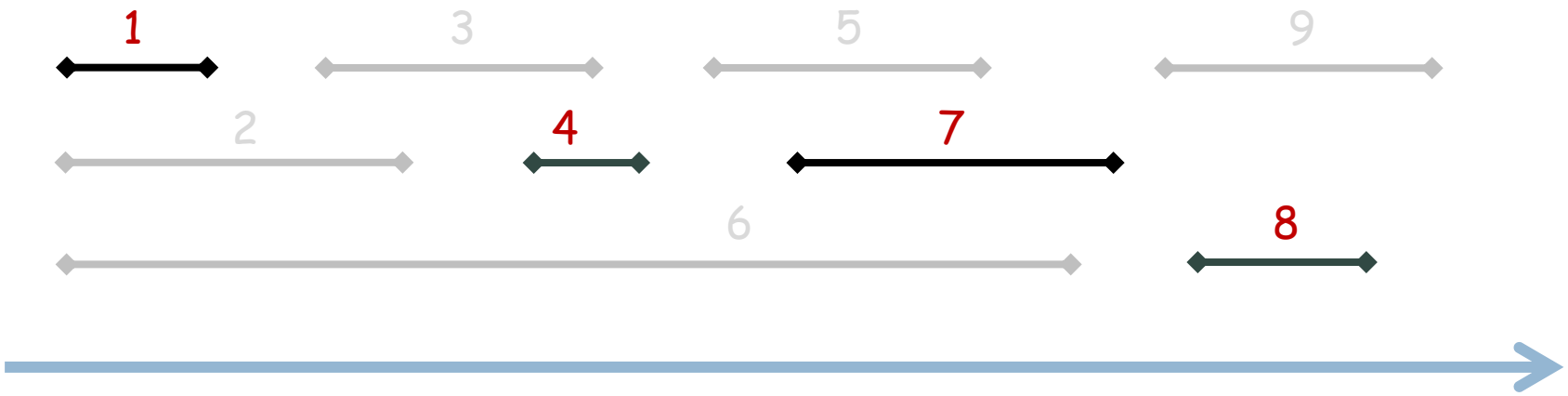
- Ένα σύνολο αιτημάτων S λέγεται *συμβατό* (compatible) εάν *ανά δύο τα αιτήματά του δεν επικαλύπτονται χρονικά*.
- Στόχος είναι να υπολογίσουμε ένα *συμβατό υποσύνολο με το μέγιστο πλήθος διαστημάτων*, το οποίο ονομάζουμε *βέλτιστο* (optimal).



✓ Μη-συμβατό : $S = \{1, 2, 4, 8\}$

Χρονοπρογραμματισμός Διαστημάτων

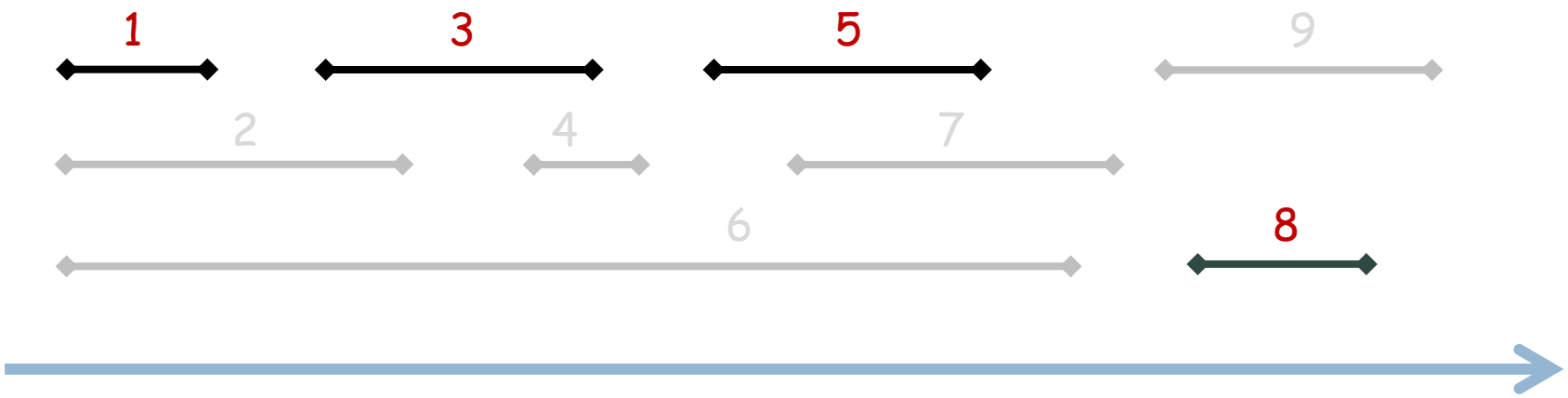
- Ένα σύνολο αιτημάτων S λέγεται *συμβατό* (compatible) εάν *ανά δύο τα αιτήματά του δεν επικαλύπτονται χρονικά*.
- Στόχος είναι να υπολογίσουμε ένα *συμβατό υποσύνολο με το μέγιστο πλήθος διαστημάτων*, το οποίο ονομάζουμε *βέλτιστο* (optimal).



✓ Βέλτιστο : $S = \{1, 4, 7, 8\}$

Χρονοπρογραμματισμός Διαστημάτων

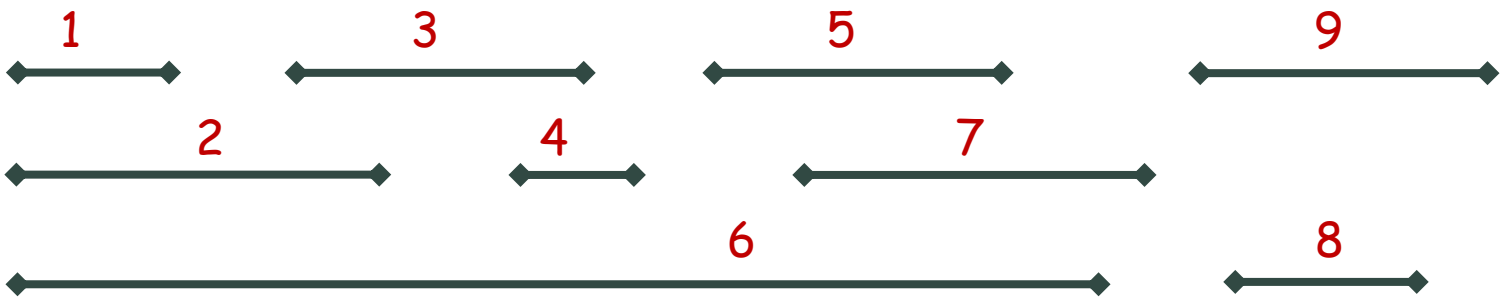
- Ένα σύνολο αιτημάτων S λέγεται *συμβατό* (compatible) εάν *ανά δύο τα αιτήματά του δεν επικαλύπτονται χρονικά*.
- Στόχος είναι να υπολογίσουμε ένα *συμβατό υποσύνολο με το μέγιστο πλήθος διαστημάτων*, το οποίο ονομάζουμε *βέλτιστο* (optimal).



✓ Βέλτιστο : $S = \{1, 3, 5, 8\}$

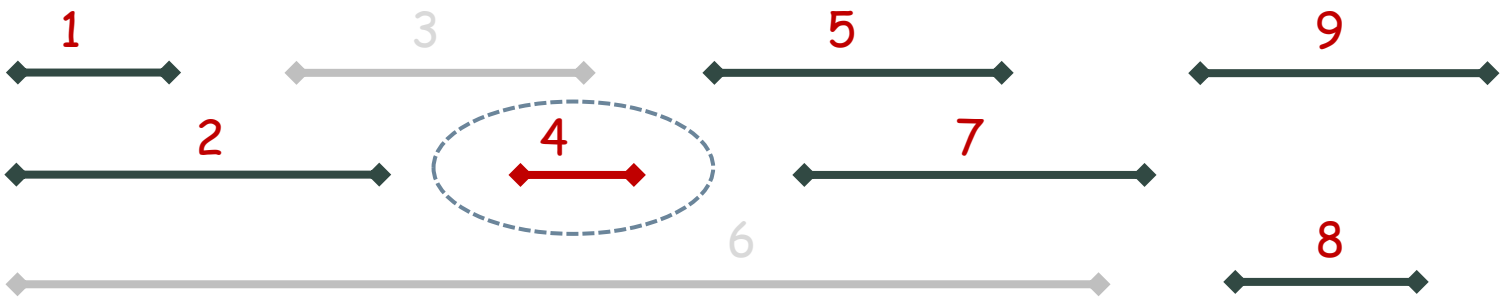
Χρονοπρογραμματισμός Διαστημάτων

- **Βασική ιδέα** ενός **άπληστου αλγόριθμου**
  - Χρησιμοποίησε έναν απλό **κανόνα** για την ***τοπική βέλτιστη επιλογή*** του πρώτου αιτήματος i_1 (άπληστη επιλογή), απορρίπτοντας τα μη-συμβατά αιτήματα με αυτό.
  - Επίλεξε με βάση τον κανόνα το επόμενο αίτημα i_2, απορρίπτοντας τα μη-συμβατά αιτήματα με το i_2, και συνέχισε έως να εξαντληθούν όλα τα αιτήματα.



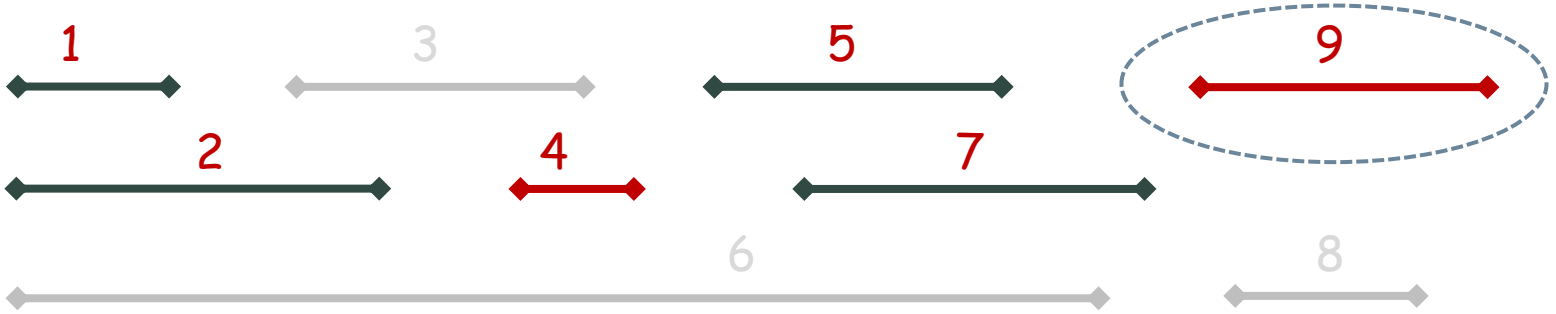
Χρονοπρογραμματισμός Διαστημάτων

- **Βασική ιδέα** ενός **άπληστου αλγόριθμου**
  - Χρησιμοποίησε έναν απλό **κανόνα** για την ***τοπική βέλτιστη επιλογή*** του πρώτου αιτήματος i_1 (άπληστη επιλογή), απορρίπτοντας τα μη-συμβατά αιτήματα με αυτό.
  - Επίλεξε με βάση τον κανόνα το επόμενο αίτημα i_2, απορρίπτοντας τα μη-συμβατά αιτήματα με το i_2, και συνέχισε έως να εξαντληθούν όλα τα αιτήματα.



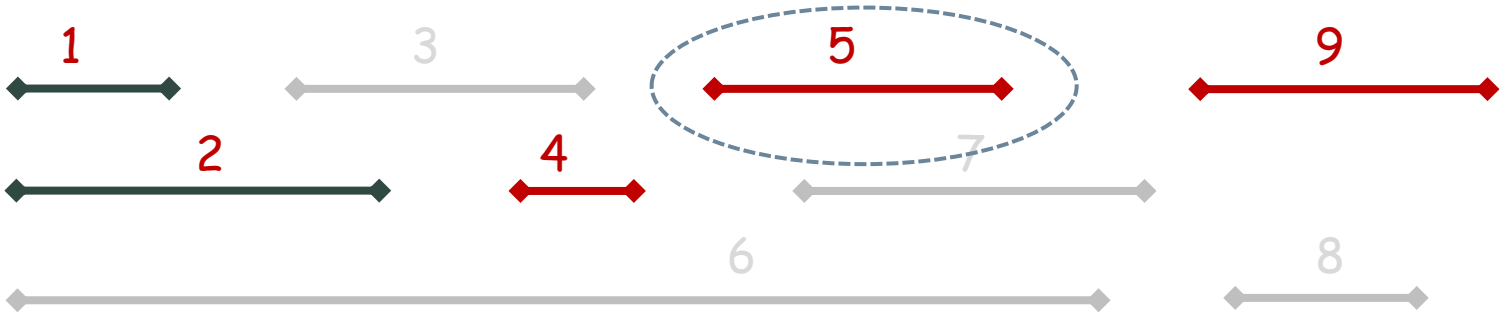
Χρονοπρογραμματισμός Διαστημάτων

- **Βασική ιδέα** ενός **άπληστου αλγόριθμου**
  - Χρησιμοποίησε έναν απλό **κανόνα** για την ***τοπική βέλτιστη επιλογή*** του πρώτου αιτήματος i_1 (άπληστη επιλογή), απορρίπτοντας τα μη-συμβατά αιτήματα με αυτό.
  - Επίλεξε με βάση τον κανόνα το επόμενο αίτημα i_2, απορρίπτοντας τα μη-συμβατά αιτήματα με το i_2, και συνέχισε έως να εξαντληθούν όλα τα αιτήματα.



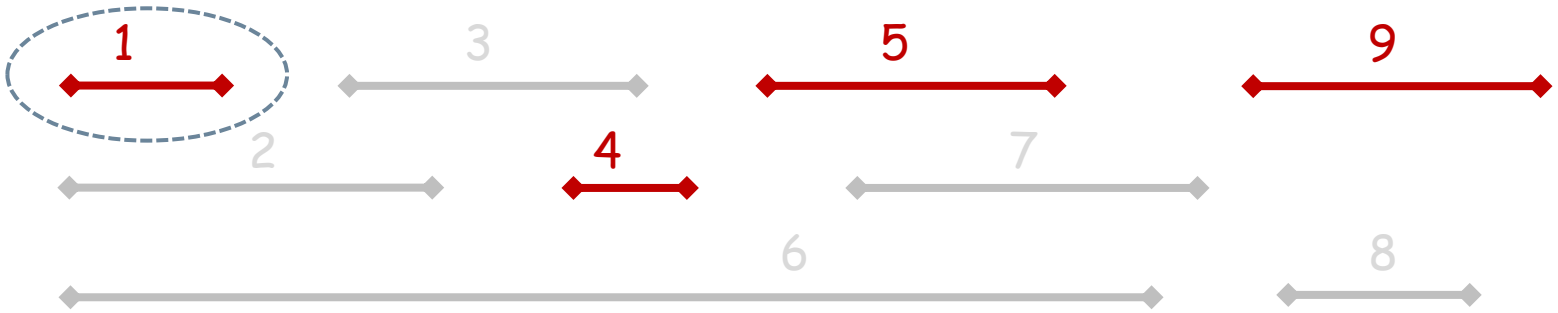
Χρονοπρογραμματισμός Διαστημάτων

- **Βασική ιδέα** ενός **άπληστου αλγόριθμου**
  - Χρησιμοποίησε έναν απλό **κανόνα** για την ***τοπική βέλτιστη επιλογή*** του πρώτου αιτήματος i_1 (άπληστη επιλογή), απορρίπτοντας τα μη-συμβατά αιτήματα με αυτό.
  - Επίλεξε με βάση τον κανόνα το επόμενο αίτημα i_2, απορρίπτοντας τα μη-συμβατά αιτήματα με το i_2, και συνέχισε έως να εξαντληθούν όλα τα αιτήματα.



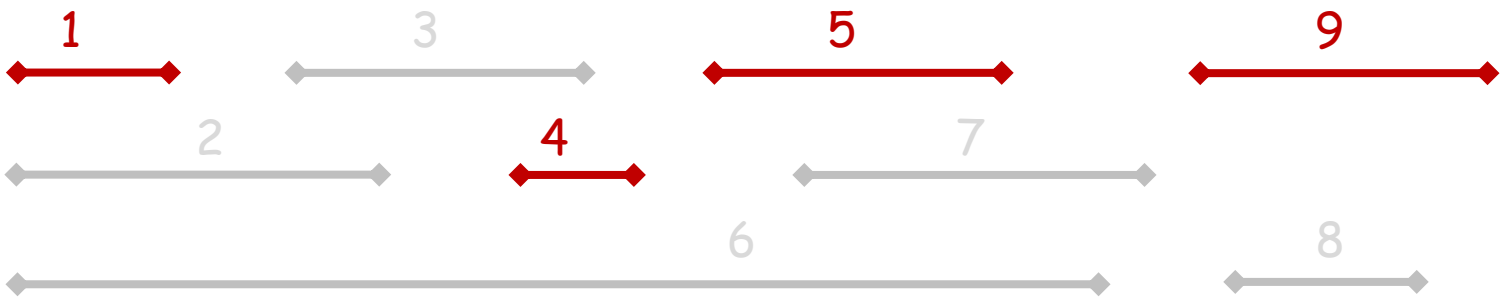
Χρονοπρογραμματισμός Διαστημάτων

- **Βασική ιδέα** ενός **άπληστου αλγόριθμου**
  - Χρησιμοποίησε έναν απλό **κανόνα** για την ***τοπική βέλτιστη επιλογή*** του πρώτου αιτήματος i_1 (άπληστη επιλογή), απορρίπτοντας τα μη-συμβατά αιτήματα με αυτό.
  - Επίλεξε με βάση τον κανόνα το επόμενο αίτημα i_2, απορρίπτοντας τα μη-συμβατά αιτήματα με το i_2, και συνέχισε έως να εξαντληθούν όλα τα αιτήματα.



Χρονοπρογραμματισμός Διαστημάτων

- **Βασική ιδέα** ενός **άπληστου αλγόριθμου**
  - Χρησιμοποίησε έναν απλό **κανόνα** για την ***τοπική βέλτιστη επιλογή*** του πρώτου αιτήματος i_1 (άπληστη επιλογή), απορρίπτοντας τα μη-συμβατά αιτήματα με αυτό.
  - Επίλεξε με βάση τον κανόνα το επόμενο αίτημα i_2, απορρίπτοντας τα μη-συμβατά αιτήματα με το i_2, και συνέχισε έως να εξαντληθούν όλα τα αιτήματα.



✓ Άπληστη λύση : $S = \{1, 4, 5, 9\}$

Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής

Η πρόκληση για το σχεδιασμό ενός καλού άπληστου αλγόριθμου *βρίσκεται στην απόφαση* για τον απλό κανόνα για την επιλογή !!!

- Για το πρόβλημα του Χρονοπρογραμματισμού Διαστημάτων, όπως και για πολλά άλλα, υπάρχουν πολλοί φυσικοί (καλοί και κακοί) κανόνες !
- Για παράδειγμα, απλοί κανόνες επιλογής θα μπορούσαν να είναι:
  - ✓ Επίλεξε πάντα το διαθέσιμο διάστημα που ξεκινάει νωρίτερα, ή
  - ✓ Αυτό που απαιτεί το μικρότερο χρονικό διάστημα ολοκλήρωσης, ή ακόμα
  - ✓ Το διάστημα που απορρίπτει το μικρότερο πλήθος μη-συμβατών αιτημάτων, ή
  - ✓ Το διαθέσιμο διάστημα που τελειώνει νωρίτερα.

Χρονοπρογραμματισμός Διαστημάτων

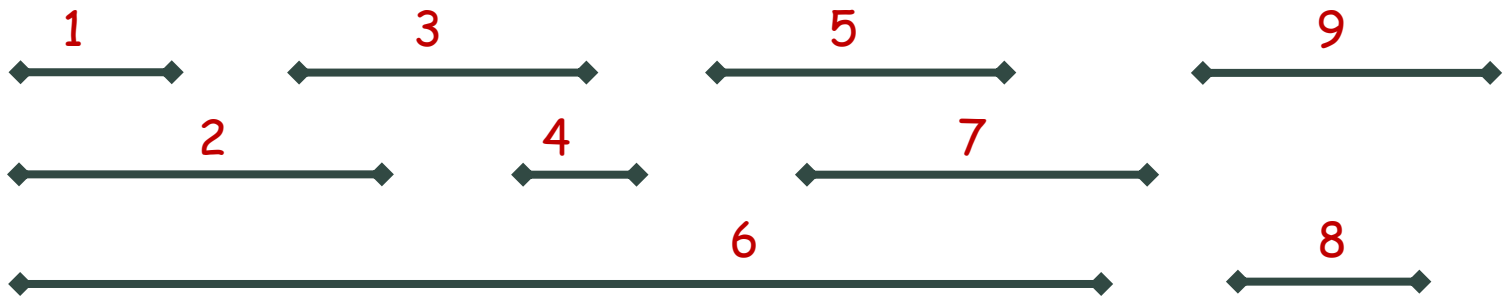
Κανόνας Επιλογής 1: Επίλεξε πάντα το διαθέσιμο διάστημα που ξεκινάει νωρίτερα – δηλ. το αίτημα με το μικρότερο $s(i)$!!!

- Ιδέα της άπληστης επιλογής :

  Ο πόρος μας θα ξεκινά να χρησιμοποιείται όσο το δυνατόν νωρίτερα και έτσι θα μείνει χρόνος για να ικανοποιηθούν περισσότερα αιτήματα !

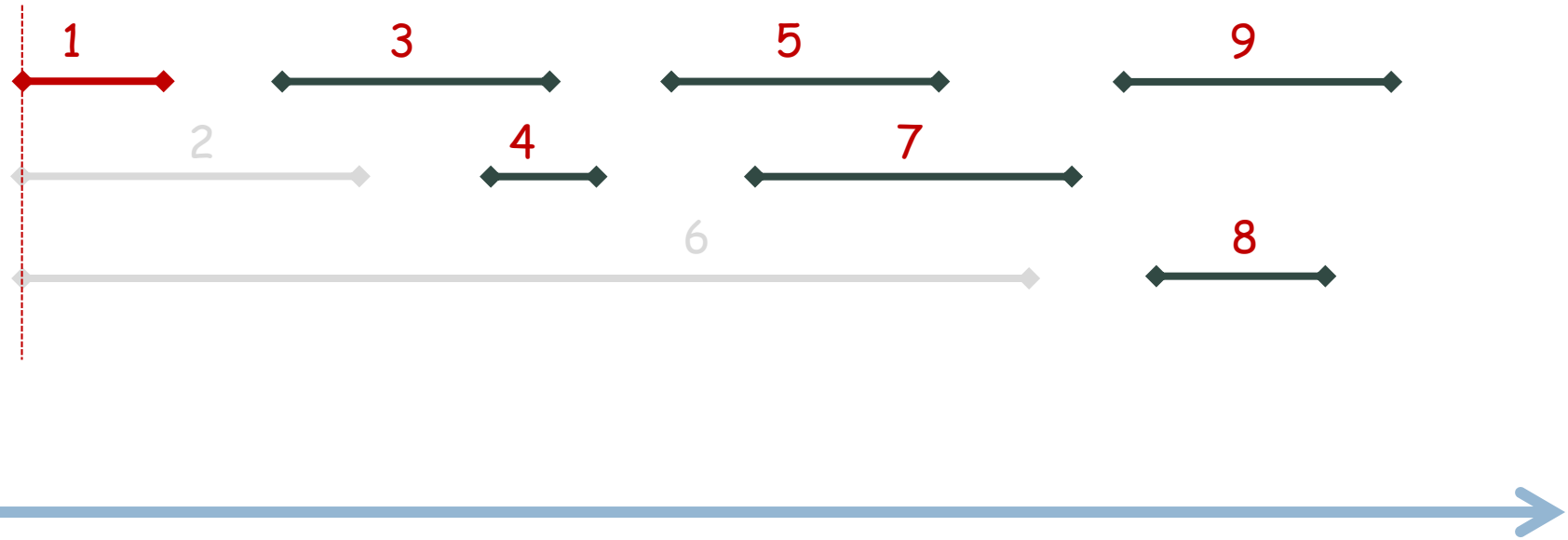
Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 1: Επίλεξε πάντα το διαθέσιμο διάστημα που ξεκινάει νωρίτερα – δηλ. το αίτημα με το μικρότερο s(i) !!!



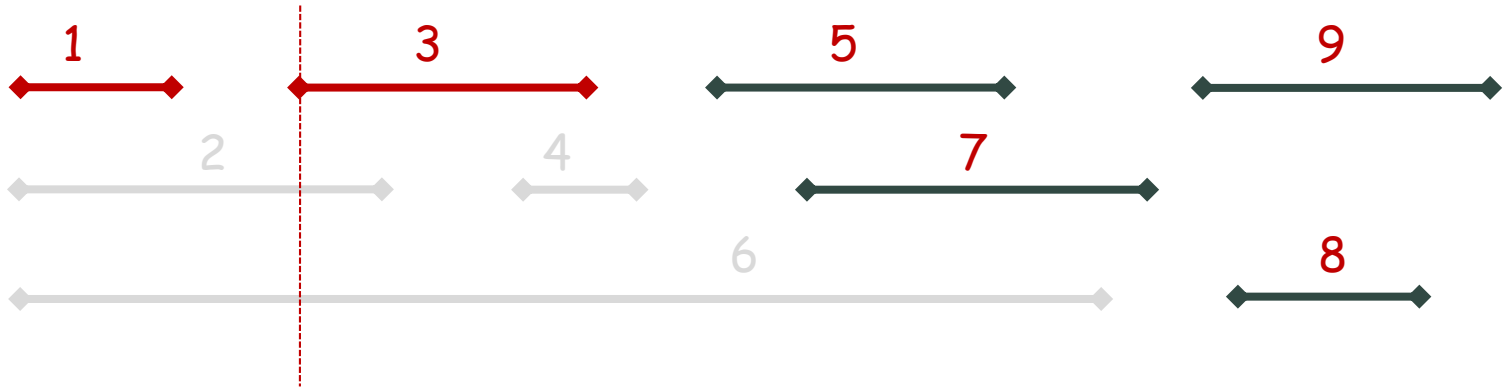
Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 1: Επίλεξε πάντα το διαθέσιμο διάστημα που ξεκινάει νωρίτερα – δηλ. το αίτημα με το μικρότερο s(i) !!!



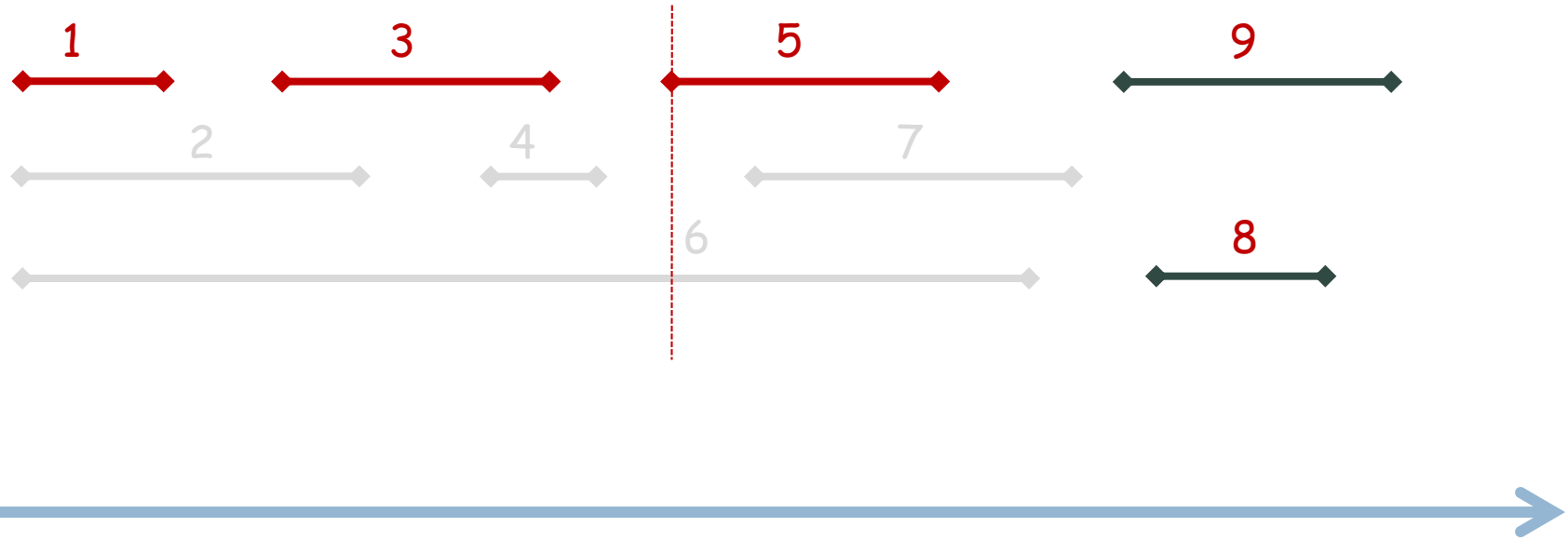
Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 1: Επίλεξε πάντα το διαθέσιμο διάστημα που ξεκινάει νωρίτερα – δηλ. το αίτημα με το μικρότερο s(i) !!!



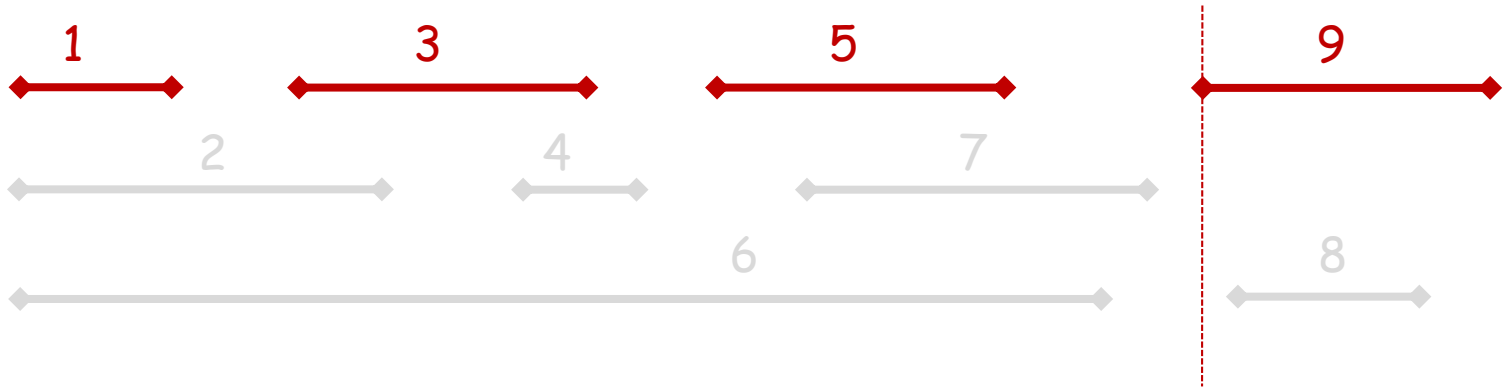
Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 1: Επίλεξε πάντα το διαθέσιμο διάστημα που ξεκινάει νωρίτερα – δηλ. το αίτημα με το μικρότερο s(i) !!!



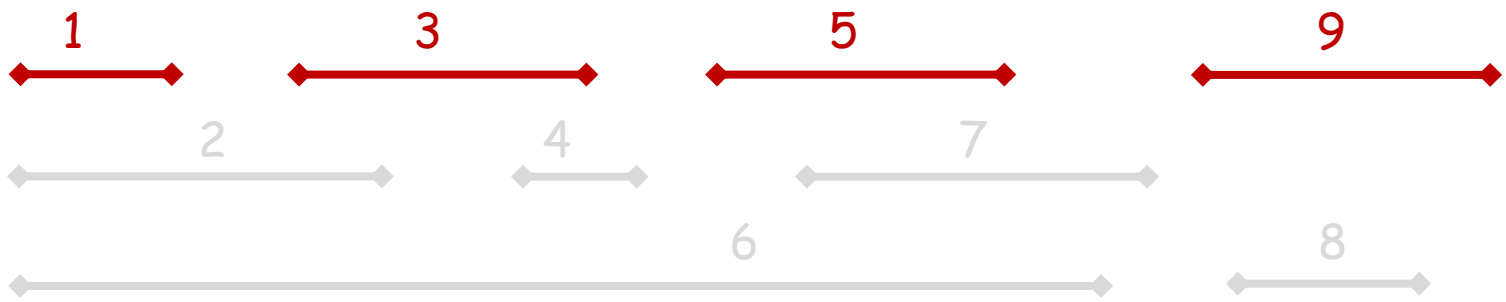
Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 1: Επίλεξε πάντα το διαθέσιμο διάστημα που ξεκινάει νωρίτερα – δηλ. το αίτημα με το μικρότερο s(i) !!!



Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 1: Επίλεξε πάντα το διαθέσιμο διάστημα που ξεκινάει νωρίτερα – δηλ. το αίτημα με το μικρότερο s(i) !!!



Άπληστη λύση : S = {1, 3, 5, 9} ⇒ Βέλτιστη

Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 1: Επίλεξε πάντα το διαθέσιμο διάστημα που ξεκινάει νωρίτερα – δηλ. το αίτημα με το μικρότερο s(i) !!!

Ερώτηση:

Ο **Κανόνας Τοπικής Επιλογής 1**
δίνει πάντα βέλτιστη λύση ?

Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 1: Επίλεξε πάντα το διαθέσιμο διάστημα που ξεκινάει νωρίτερα – δηλ. το αίτημα με το μικρότερο s(i) !!!

Αντιπαράδειγμα

Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 1: Επίλεξε πάντα το διαθέσιμο διάστημα που ξεκινάει νωρίτερα – δηλ. το αίτημα με το μικρότερο s(i) !!!

Αντιπαράδειγμα

Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 1: Επίλεξε πάντα το διαθέσιμο διάστημα που ξεκινάει νωρίτερα – δηλ. το αίτημα με το μικρότερο s(i) !!!

Αντιπαράδειγμα

Ο κανόνας αποτυγχάνει !!!

Χρονοπρογραμματισμός Διαστημάτων

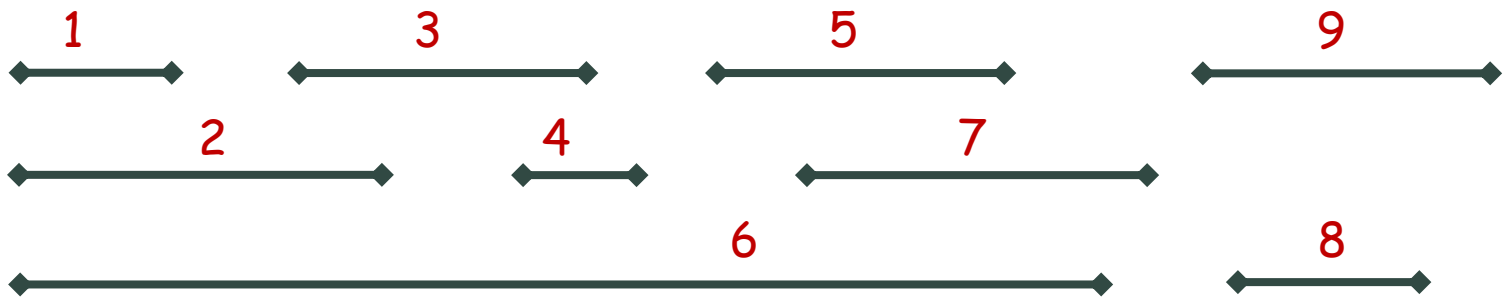
Κανόνας Επιλογής 2: Επίλεξε πάντα το διαθέσιμο διάστημα που απαιτεί το μικρότερο χρονικό διάστημα – δηλ. το αίτημα για το οποίο το $\delta(i) = f(i)-s(i)$ είναι όσο το δυνατό μικρότερο !!!

- Ιδέα της άπληστης επιλογής :

  Το επιλεγόμενο αίτημα έχοντας με μικρό χρόνο εξυπηρέτησης εκτιμάται ότι θα απορρίπτει το λιγότερο δυνατό πλήθος μη-συμβατών αιτημάτων κάθε φορά !

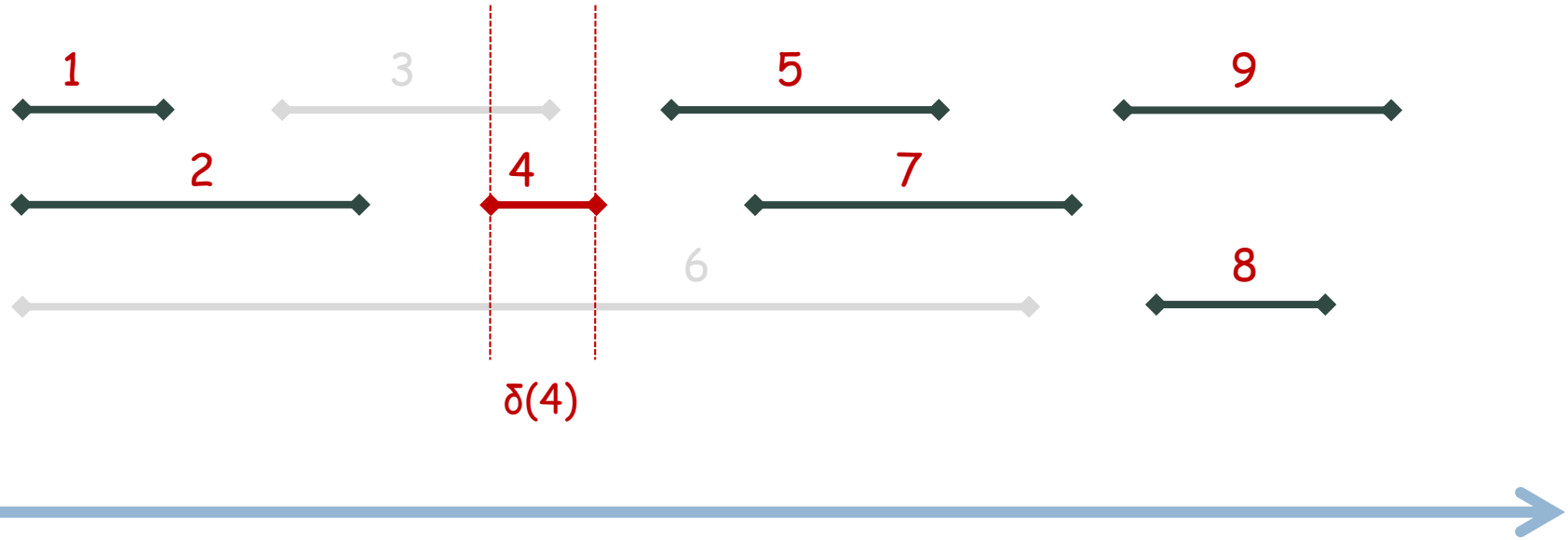
Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 2: Επίλεξε πάντα το διαθέσιμο διάστημα που απαιτεί το μικρότερο χρονικό διάστημα – δηλ. το αίτημα για το οποίο το $\delta(i) = f(i)-s(i)$ είναι όσο το δυνατό μικρότερο !!!



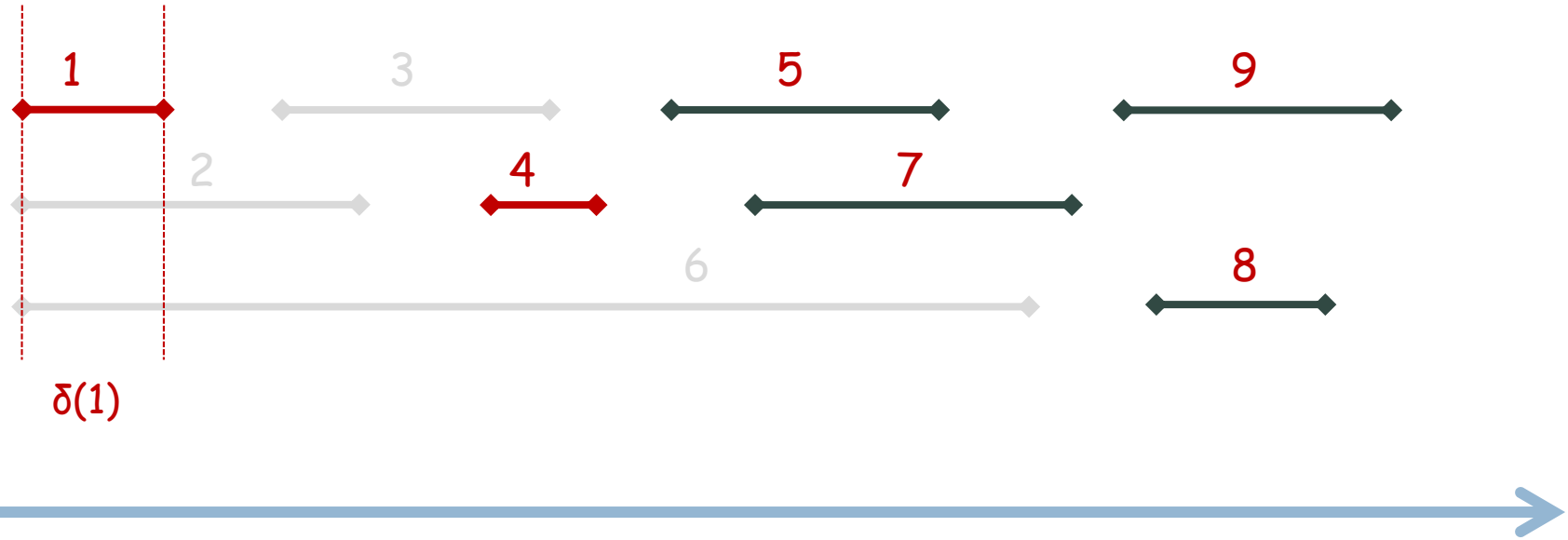
Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 2: Επίλεξε πάντα το διαθέσιμο διάστημα που απαιτεί το μικρότερο χρονικό διάστημα – δηλ. το αίτημα για το οποίο το δ(i) = f(i)-s(i) είναι όσο το δυνατό μικρότερο !!!



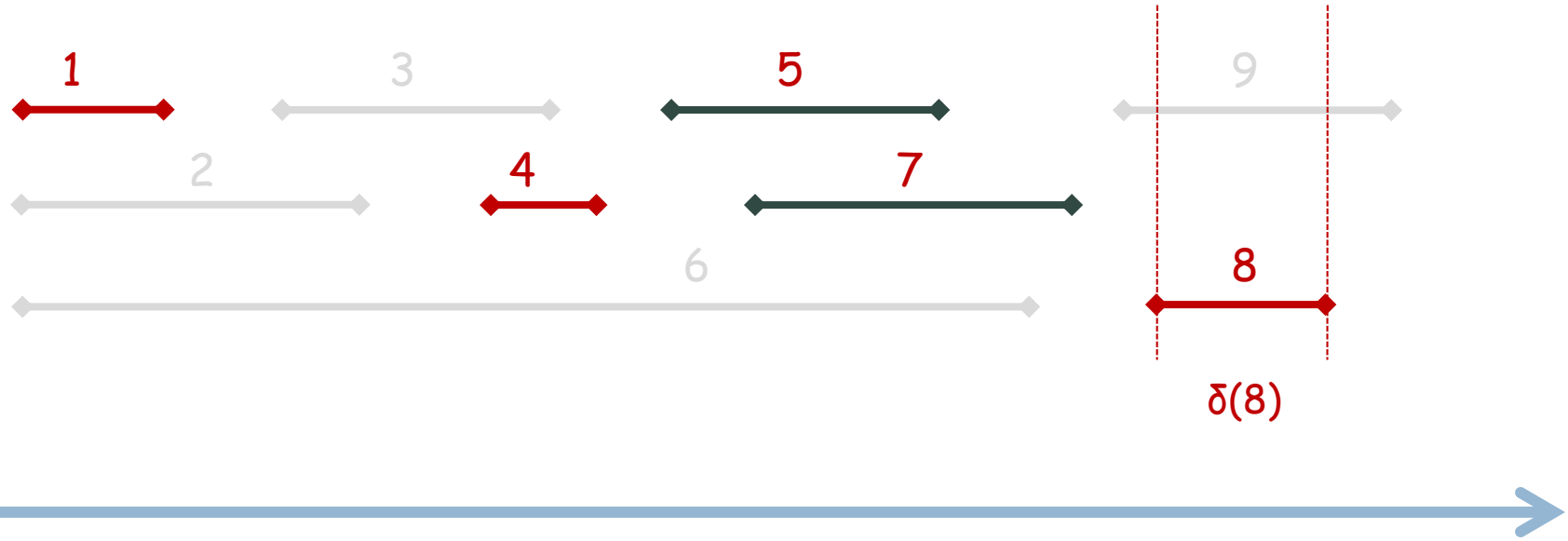
Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 2: Επίλεξε πάντα το διαθέσιμο διάστημα που απαιτεί το μικρότερο χρονικό διάστημα – δηλ. το αίτημα για το οποίο το $\delta(i) = f(i)-s(i)$ είναι όσο το δυνατό μικρότερο !!!



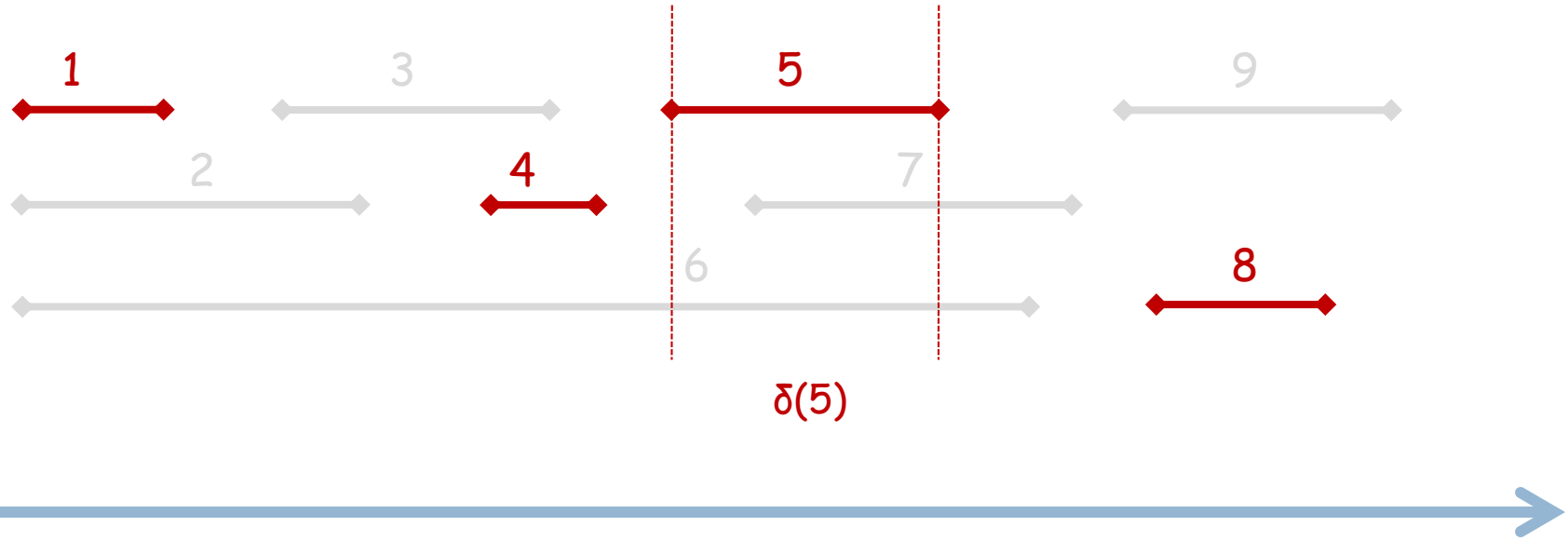
Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 2: Επίλεξε πάντα το διαθέσιμο διάστημα που απαιτεί το μικρότερο χρονικό διάστημα – δηλ. το αίτημα για το οποίο το δ(i) = f(i)-s(i) είναι όσο το δυνατό μικρότερο !!!



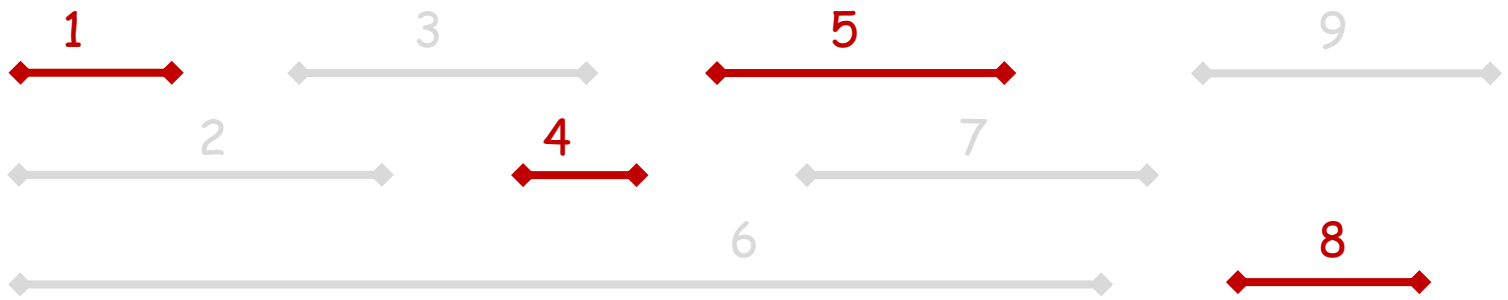
Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 2: Επίλεξε πάντα το διαθέσιμο διάστημα που απαιτεί το μικρότερο χρονικό διάστημα – δηλ. το αίτημα για το οποίο το δ(i) = f(i)-s(i) είναι όσο το δυνατό μικρότερο !!!



Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 2: Επίλεξε πάντα το διαθέσιμο διάστημα που απαιτεί το μικρότερο χρονικό διάστημα – δηλ. το αίτημα για το οποίο το $\delta(i) = f(i)-s(i)$ είναι όσο το δυνατό μικρότερο !!!



Άπληστη λύση : $S = \{1, 4, 5, 8\}$ $\Rightarrow$ Βέλτιστη

Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 2: Επίλεξε πάντα το διαθέσιμο διάστημα που απαιτεί το μικρότερο χρονικό διάστημα – δηλ. το αίτημα για το οποίο το $\delta(i) = f(i)-s(i)$ είναι όσο το δυνατό μικρότερο !!!

Ερώτηση:

Ο **Κανόνας Τοπικής Επιλογής 2**

δίνει πάντα βέλτιστη λύση ?

Χρονοπρογραμματισμός Διαστημάτων

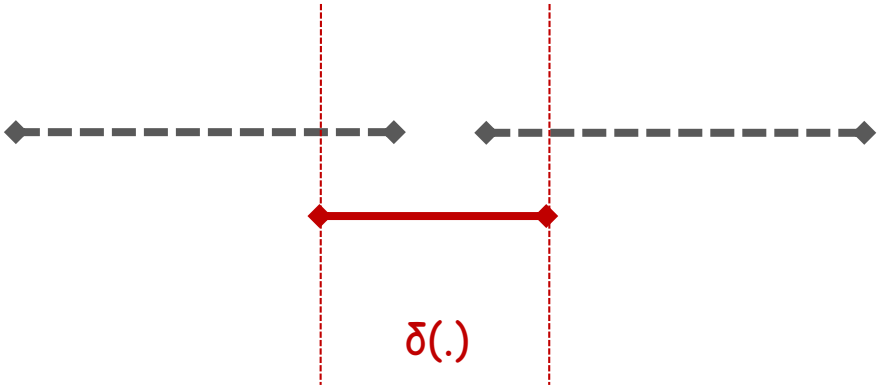
Κανόνας Επιλογής 2: Επίλεξε πάντα το διαθέσιμο διάστημα που απαιτεί το μικρότερο χρονικό διάστημα – δηλ. το αίτημα για το οποίο το $\delta(i) = f(i)-s(i)$ είναι όσο το δυνατό μικρότερο !!!

Αντιπαράδειγμα

Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 2: Επίλεξε πάντα το διαθέσιμο διάστημα που απαιτεί το μικρότερο χρονικό διάστημα – δηλ. το αίτημα για το οποίο το δ(i) = f(i)-s(i) είναι όσο το δυνατό μικρότερο !!!

Αντιπαράδειγμα



Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 2: Επίλεξε πάντα το διαθέσιμο διάστημα που απαιτεί το μικρότερο χρονικό διάστημα – δηλ. το αίτημα για το οποίο το $\delta(i) = f(i)-s(i)$ είναι όσο το δυνατό μικρότερο !!!

Αντιπαράδειγμα

Αυτός ο κανόνας αποτυγχάνει !!!

Χρονοπρογραμματισμός Διαστημάτων

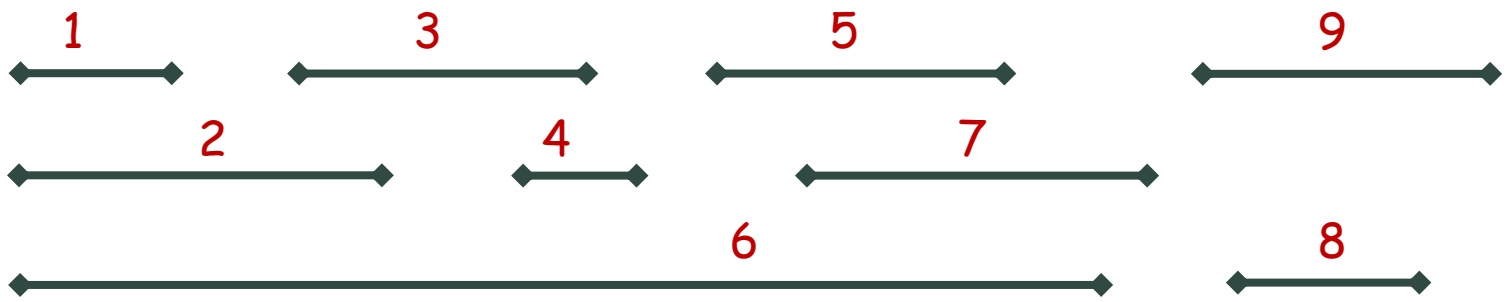
Κανόνας Επιλογής 3: Επίλεξε πάντα το διαθέσιμο διάστημα που απορρίπτει το μικρότερο πλήθος μη-συμβατών αιτημάτων – δηλ. το αίτημα με τις λιγότερες «διενέξεις» !!!

- Ιδέα της άπληστης επιλογής :

  Το αίτημα να επιλέγεται έτσι ώστε πράγματι να απορρίπτει το λιγότερο δυνατό πλήθος μη-συμβατών αιτημάτων κάθε φορά !

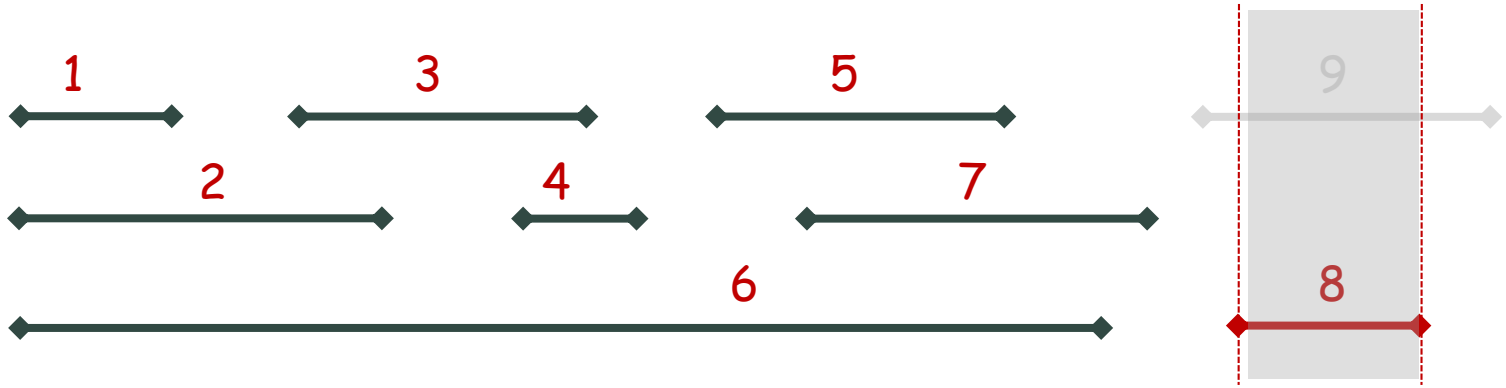
Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 3: Επίλεξε πάντα το διαθέσιμο διάστημα που απορρίπτει το μικρότερο πλήθος μη-συμβατών αιτημάτων – δηλ. το αίτημα με τις λιγότερες «διενέξεις» !!!



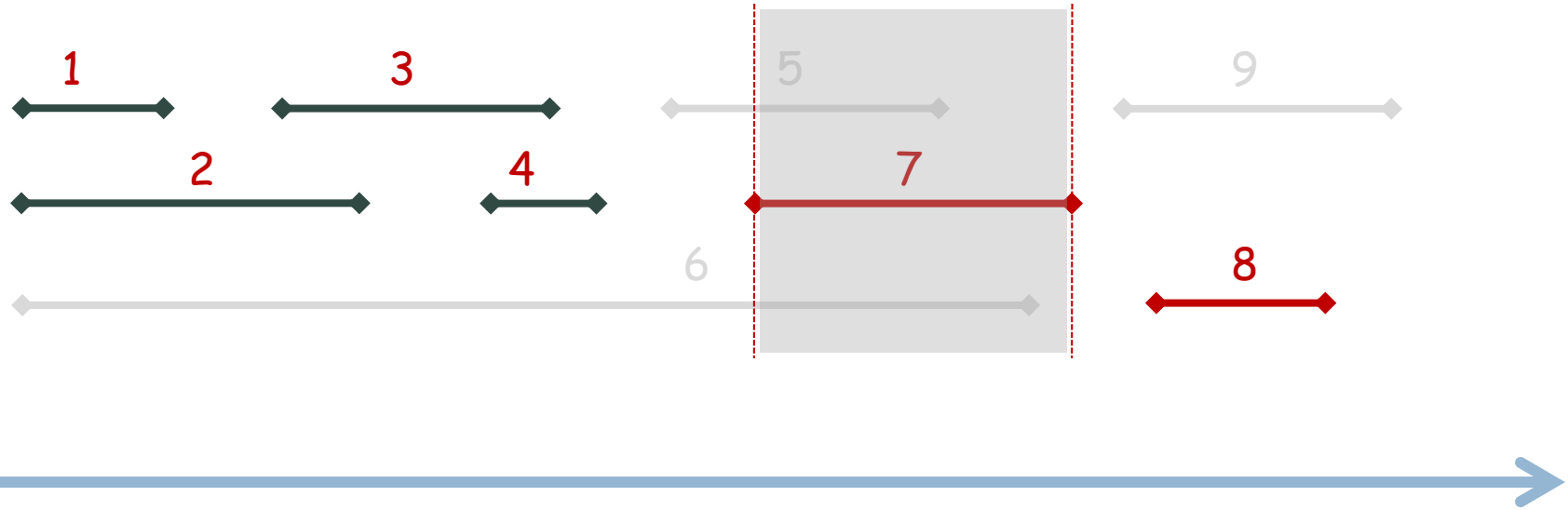
Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 3: Επίλεξε πάντα το διαθέσιμο διάστημα που απορρίπτει το μικρότερο πλήθος μη-συμβατών αιτημάτων – δηλ. το αίτημα με τις λιγότερες «διενέξεις» !!!



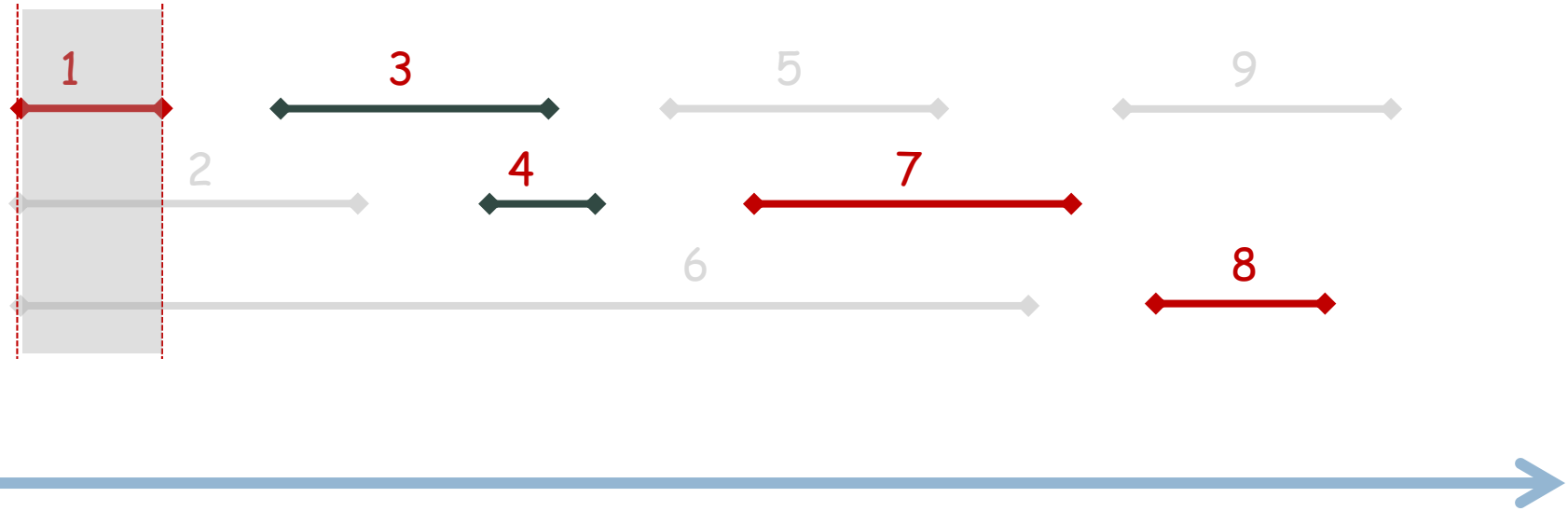
Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 3: Επίλεξε πάντα το διαθέσιμο διάστημα που απορρίπτει το μικρότερο πλήθος μη-συμβατών αιτημάτων – δηλ. το αίτημα με τις λιγότερες «διενέξεις» !!!



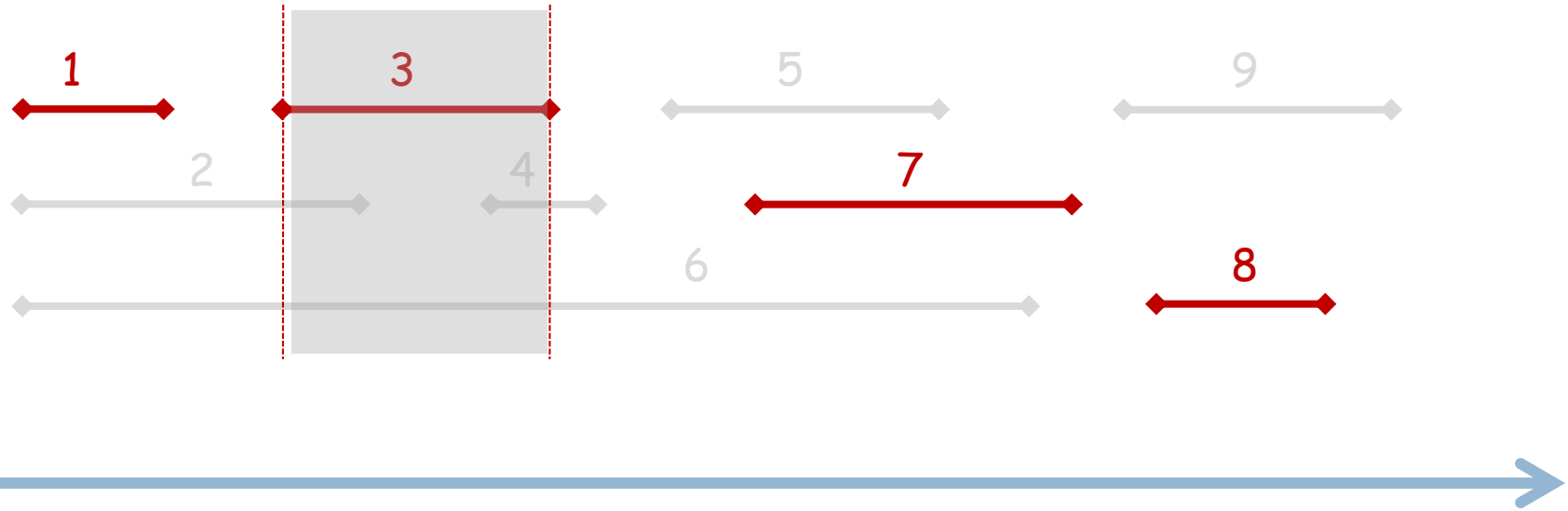
Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 3: Επίλεξε πάντα το διαθέσιμο διάστημα που απορρίπτει το μικρότερο πλήθος μη-συμβατών αιτημάτων – δηλ. το αίτημα με τις λιγότερες «διενέξεις» !!!



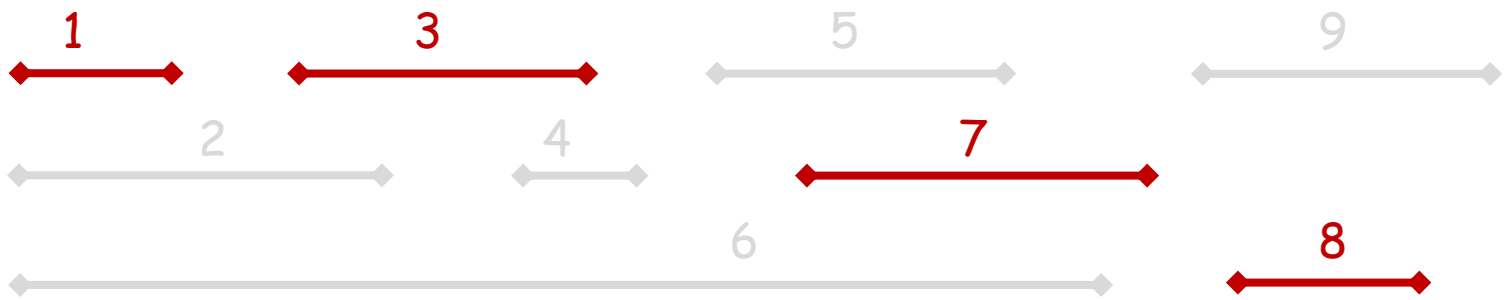
Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 3: Επίλεξε πάντα το διαθέσιμο διάστημα που απορρίπτει το μικρότερο πλήθος μη-συμβατών αιτημάτων – δηλ. το αίτημα με τις λιγότερες «διενέξεις» !!!



Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 3: Επίλεξε πάντα το διαθέσιμο διάστημα που απορρίπτει το μικρότερο πλήθος μη-συμβατών αιτημάτων – δηλ. το αίτημα με τις λιγότερες «διενέξεις» !!!



Άπληστη λύση : $S = \{1, 3, 7, 8\}$ $\Rightarrow$ Βέλτιστη

Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 3: Επίλεξε πάντα το διαθέσιμο διάστημα που απορρίπτει το μικρότερο πλήθος μη-συμβατών αιτημάτων – δηλ. το αίτημα με τις λιγότερες «διενέξεις» !!!

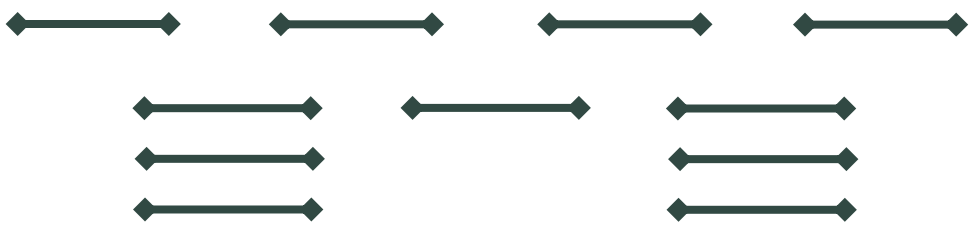
Ερώτηση:

Ο **Κανόνας Τοπικής Επιλογής 3** δίνει πάντα βέλτιστη λύση ?

Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 3: Επίλεξε πάντα το διαθέσιμο διάστημα που απορρίπτει το μικρότερο πλήθος μη-συμβατών αιτημάτων – δηλ. το αίτημα με τις λιγότερες «διενέξεις» !!!

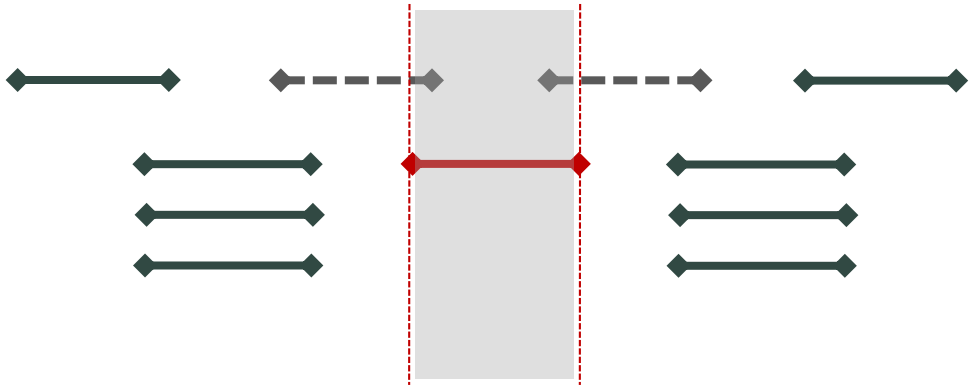
Αντιπαράδειγμα

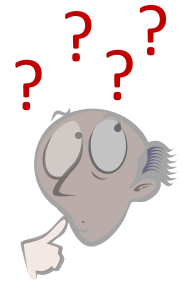


Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 3: Επίλεξε πάντα το διαθέσιμο διάστημα που απορρίπτει το μικρότερο πλήθος μη-συμβατών αιτημάτων – δηλ. το αίτημα με τις λιγότερες «διενέξεις» !!!

Αντιπαράδειγμα

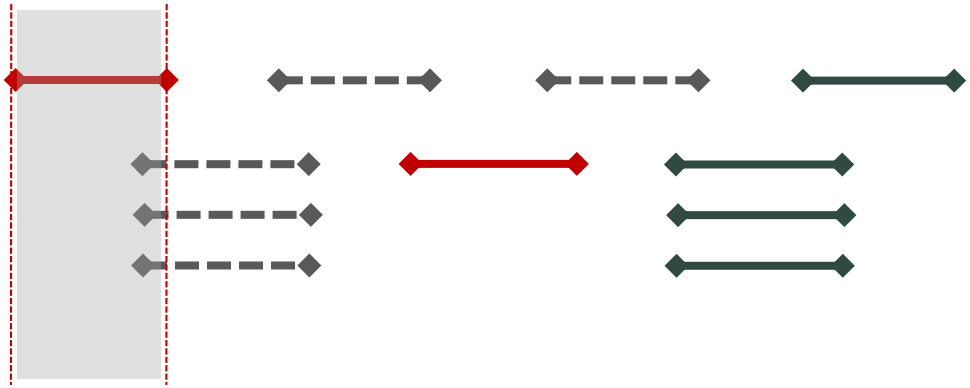


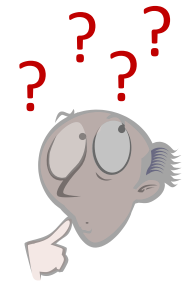


Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 3: Επίλεξε πάντα το διαθέσιμο διάστημα που απορρίπτει το μικρότερο πλήθος μη-συμβατών αιτημάτων – δηλ. το αίτημα με τις λιγότερες «διενέξεις» !!!

Αντιπαράδειγμα

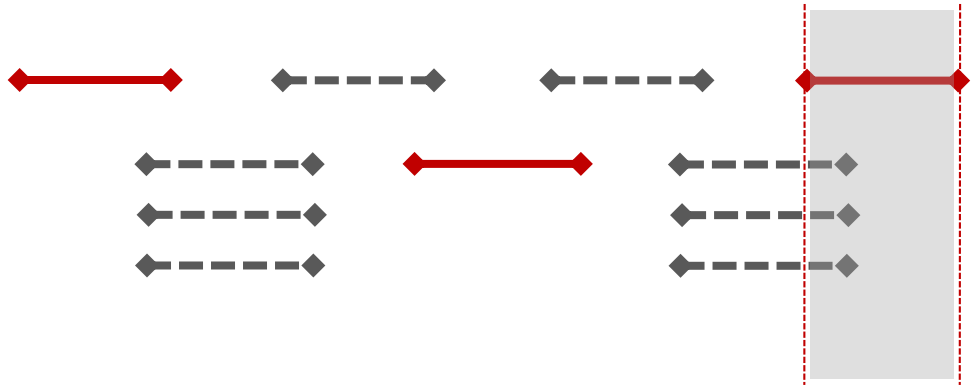


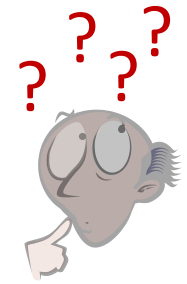


Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 3: Επίλεξε πάντα το διαθέσιμο διάστημα που απορρίπτει το μικρότερο πλήθος μη-συμβατών αιτημάτων – δηλ. το αίτημα με τις λιγότερες «διενέξεις» !!!

Αντιπαράδειγμα

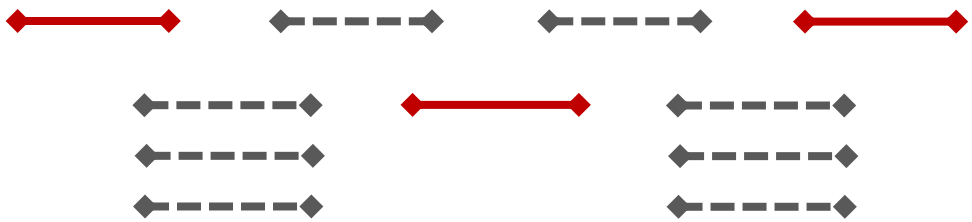




Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 3: Επίλεξε πάντα το διαθέσιμο διάστημα που απορρίπτει το μικρότερο πλήθος μη-συμβατών αιτημάτων – δηλ. το αίτημα με τις λιγότερες «διενέξεις» !!!

Αντιπαράδειγμα



Και αυτός ο κανόνας αποτυγχάνει !!!

Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 4: Επίλεξε πάντα το διαθέσιμο διάστημα που τελειώνει νωρίτερα – δηλ. το αίτημα με το μικρότερο f(i) !!!

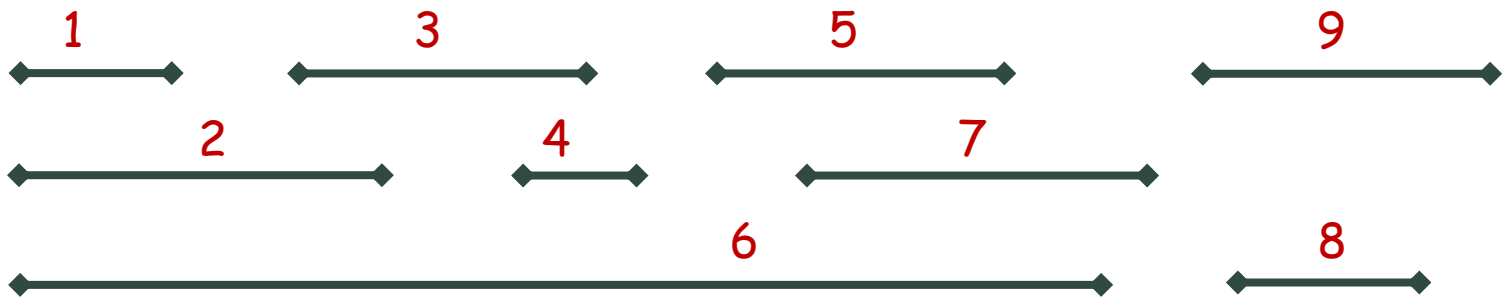
Αυτή είναι επίσης μια αρκετά φυσική ιδέα:

- ✓ διασφαλίσουμε ότι οι πόροι μας ***απελευθερώνονται όσο το δυνατόν πιο σύντομα***, και έτσι
- ✓ ***μεγιστοποιούμε το χρόνο που απομένει*** για την ικανοποίηση των άλλων αιτημάτων !!!

Αυτός ο **άπληστος κανόνας** οδηγεί σε βέλτιστη λύση !!!

Χρονοπρογραμματισμός Διαστημάτων

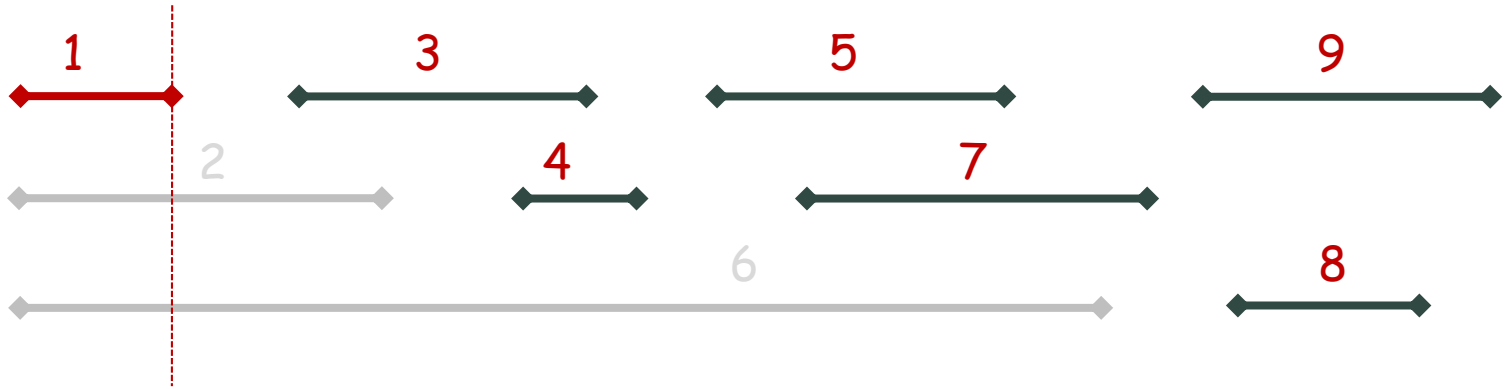
Κανόνας Επιλογής 4: Επίλεξε πάντα το διαθέσιμο διάστημα που τελειώνει νωρίτερα – δηλ. το αίτημα με το μικρότερο f(i) !!!



Αυτός ο **άπληστος κανόνας** οδηγεί σε βέλτιστη λύση !!!

Χρονοπρογραμματισμός Διαστημάτων

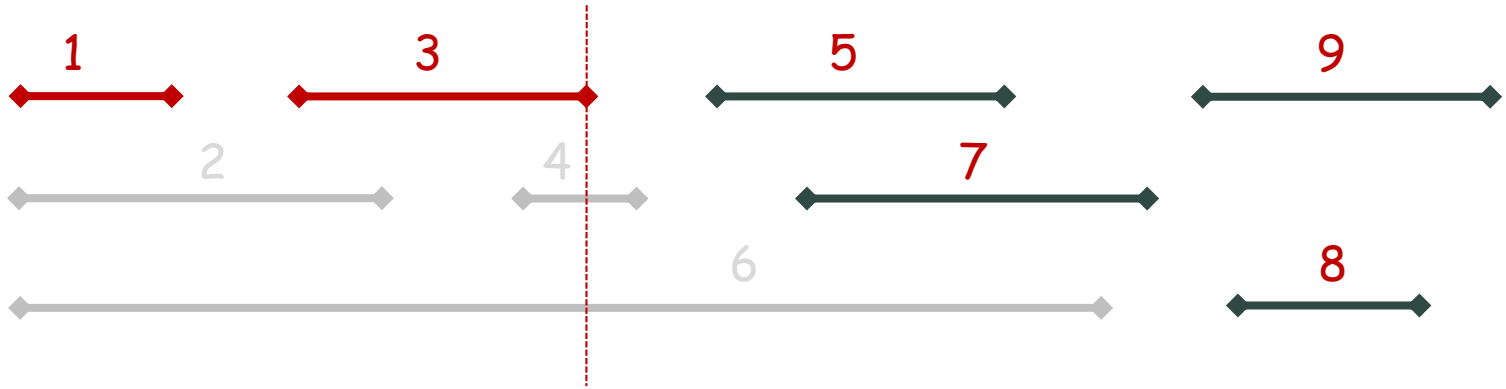
Κανόνας Επιλογής 4: Επίλεξε πάντα το διαθέσιμο διάστημα που τελειώνει νωρίτερα – δηλ. το αίτημα με το μικρότερο f(i) !!!



Αυτός ο **άπληστος κανόνας** οδηγεί σε βέλτιστη λύση !!!

Χρονοπρογραμματισμός Διαστημάτων

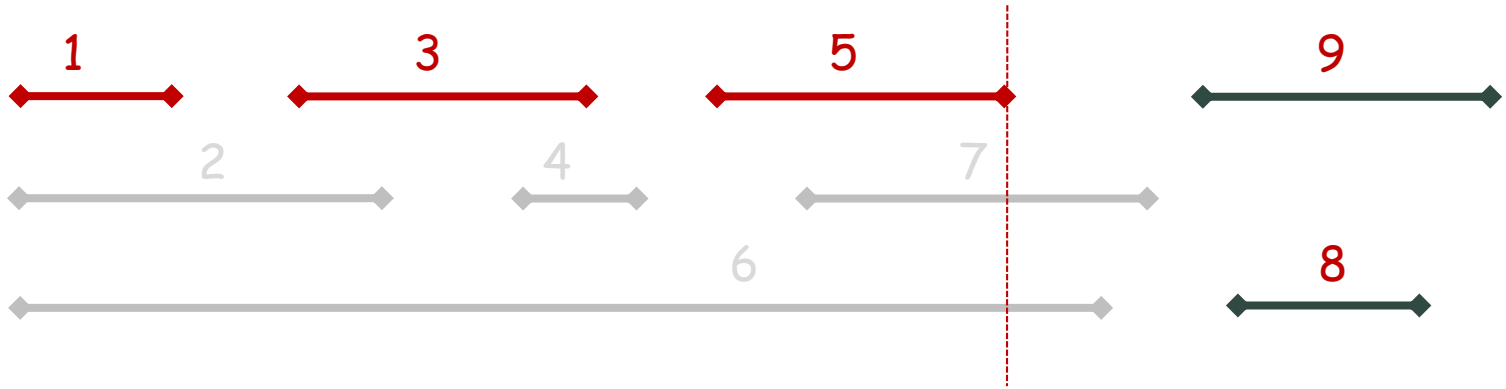
Κανόνας Επιλογής 4: Επίλεξε πάντα το διαθέσιμο διάστημα που τελειώνει νωρίτερα – δηλ. το αίτημα με το μικρότερο f(i) !!!



Αυτός ο **άπληστος κανόνας** οδηγεί σε βέλτιστη λύση !!!

Χρονοπρογραμματισμός Διαστημάτων

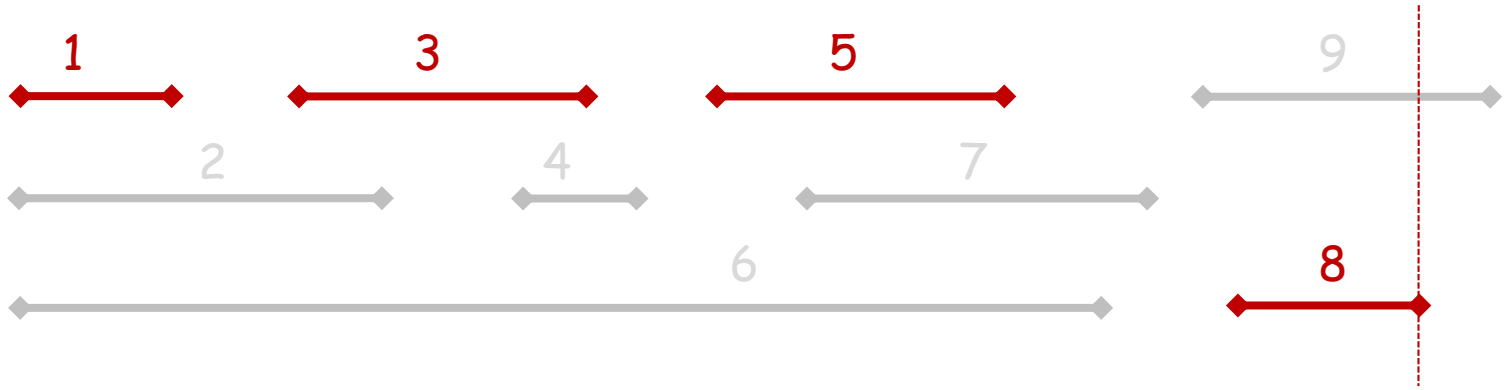
Κανόνας Επιλογής 4: Επίλεξε πάντα το διαθέσιμο διάστημα που τελειώνει νωρίτερα – δηλ. το αίτημα με το μικρότερο f(i) !!!



Αυτός ο **άπληστος κανόνας** οδηγεί σε βέλτιστη λύση !!!

Χρονοπρογραμματισμός Διαστημάτων

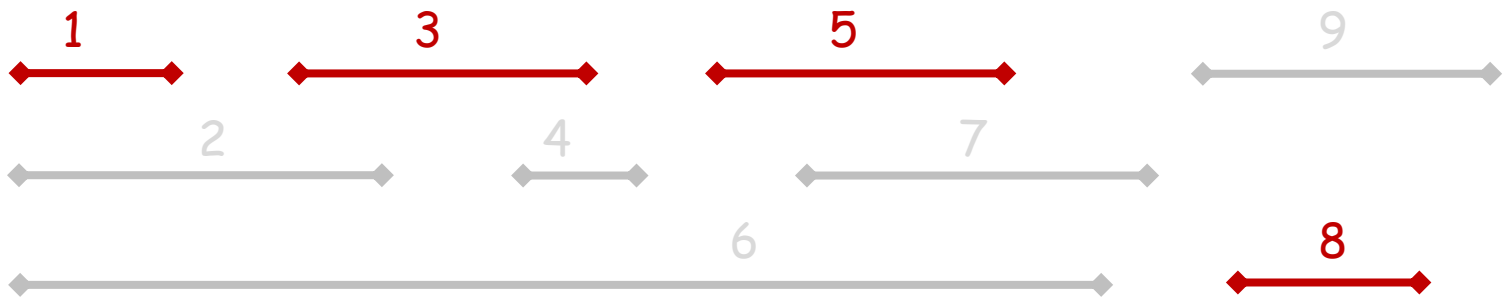
Κανόνας Επιλογής 4: Επίλεξε πάντα το διαθέσιμο διάστημα που τελειώνει νωρίτερα – δηλ. το αίτημα με το μικρότερο f(i) !!!



Αυτός ο **άπληστος κανόνας** οδηγεί σε βέλτιστη λύση !!!

Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 4: Επίλεξε πάντα το διαθέσιμο διάστημα που τελειώνει νωρίτερα – δηλ. το αίτημα με το μικρότερο f(i) !!!



Αυτός ο **άπληστος κανόνας** οδηγεί σε βέλτιστη λύση !!!

Χρονοπρογραμματισμός Διαστημάτων

Κανόνας Επιλογής 4: Επίλεξε πάντα το διαθέσιμο διάστημα που τελειώνει νωρίτερα – δηλ. το αίτημα με το μικρότερο f(i) !!!

Προσοχή !

- ✓ Αυτό είναι ένα παράδειγμα και όχι απόδειξη !
- ✓ ***Απαιτείται απόδειξη ότι αυτός ο άπληστος κανόνας δίδει πάντα βέλτιστη λύση*** !!!

Αυτός ο **άπληστος κανόνας** οδηγεί σε βέλτιστη λύση !!!

Χαρακτηριστικά της Τεχνικής

- Τα προβλήματα βελτιστοποίησης όπου η άπληστη τεχνική λειτουργεί καλά έχουν δύο κοινά χαρακτηριστικά:

  - **Ιδιότητα άπληστης επιλογής** (Greedy choice property):

    Από ένα τοπικό βέλτιστο μπορούμε να επιτύχουμε ένα ολικό βέλτιστο, δηλ. μια βέλτιστη λύση του προβλήματος, χωρίς να αναθεωρούμε αποφάσεις που έχουμε ήδη πάρει !

  - **Ιδιότητα Βέλτιστης υποδομής** (Optimal substructure property):

    Μια βέλτιστη λύση του προβλήματος μπορεί να κατασκευαστεί από βέλτιστες λύσεις των υποπροβλημάτων του !!

Χαρακτηριστικά της Τεχνικής

- Τα υπέρ και τα κατά της απληστίας :

1) Οι άπληστοι αλγόριθμοι κυρίως (αλλά όχι πάντα) *αποτυγχάνουν να βρουν μια βέλτιστη λύση*, επειδή συνήθως δεν λειτουργούν εξαντλητικά σε όλα τα δεδομένα.

2) Αναλαμβάνουν δεσμεύσεις πολύ νωρίς για ορισμένες επιλογές που τους εμποδίζουν αργότερα να βρουν την καλύτερη συνολική λύση.

   - Για παράδειγμα, όλοι οι γνωστοί άπληστοι αλγόριθμοι για το πρόβλημα του χρωματισμού γραφημάτων, και για όλα τα άλλα NP-πλήρη προβλήματα, αποτυγχάνουν να δώσουν βέλτιστες λύσεις.

3) Παρόλα αυτά, είναι χρήσιμοι επειδή είναι γρήγοροι και συχνά δίνουν καλές προσεγγίσεις στο βέλτιστο !!!

Χαρακτηριστικά της Τεχνικής

- Τα υπέρ και τα κατά της απληστίας :

4) Όταν ένας άπληστος αλγόριθμος επιλύει βέλτιστα ένα πρόβλημα, αυτό μας δείχνει κάτι ενδιαφέρον και χρήσιμο για τη δομή του ίδιου του προβλήματος:
   - ✓ *υπάρχει ένας τοπικός κανόνας αποφάσεων τον οποίο μπορεί κανείς να χρησιμοποιήσει για να κατασκευάσει βέλτιστες λύσεις* !

5) Η *τοπικά βέλτιστη επιλογή* που κάνει ένας άπληστος αλγόριθμος σε ένα βήμα δεν βασίζεται στις μελλοντικές επιλογές!

6) Οι άπληστοι αλγόριθμοι μπορούν να χαρακτηριστούν ως «κοντόφθαλμοι» και είναι ιδανικοί μόνο για προβλήματα που έχουν την *ιδιότητα της βέλτιστης υποδομής*.

Χαρακτηριστικά της Τεχνικής

- Τα υπέρ και τα κατά της απληστίας :

7) Σε ορισμένες περιπτώσεις *υπάρχουν πολλές άπληστες παραδοχές*, αλλά μόνο λίγες από αυτές δίνουν βέλτιστη λύση (χρονοπρογραμματισμός διαστημάτων).

8) Γενικά οι άπληστοι αλγόριθμοι είναι απλοί, γρήγοροι και εύκολα υλοποιήσιμοι καθώς βασίζονται μόνο σε τοπικές επιλογές και όχι σε μελλοντικές επιλογές ούτε σε λύσεις υποπροβλημάτων.

9) Η *απόδειξη της ορθότητας* τους μπορεί να απαιτήσει αυστηρές μαθηματικές αποδείξεις και μερικές φορές είναι εξαιρετικά δύσκολη!!

10) Ωστόσο, όταν δεν έχουμε τίποτα άλλο στη διάθεσή μας η άπληστη τεχνική μπορεί να είναι η μόνη «σωτηρία» μας !!!

Χαρακτηριστικά της Τεχνικής

- Τα υπέρ και τα κατά της απληστίας :

11) Δεν υπάρχει γενικός κανόνας εφαρμογής της άπληστης μεθόδου σε ένα δεδομένο πρόβλημα, ωστόσο οι ίδιες οι προδιαγραφές του προβλήματος μπορεί να σας δώσουν μια καλή ιδέα !

12) Οι προχωρημένες μαθηματικές έννοιες μπορεί να μας βοηθήσουν να αποδείξουμε ότι ένα πρόβλημα μπορεί να λυθεί βέλτιστα με την άπληστη τεχνική, αλλά τελικά όλα καταλήγουν στην *διαίσθηση*, την *εμπειρία* και την *βαθιά γνώση* του επιστήμονα !

13) Παρά την αυστηρότητα της απληστίας, θα πρέπει να «κοιτάμε» τις άπληστες προσεγγίσεις με το μάτι ενός «ντετέκτιβ» και όχι ενός «επιστήμονα» !!!

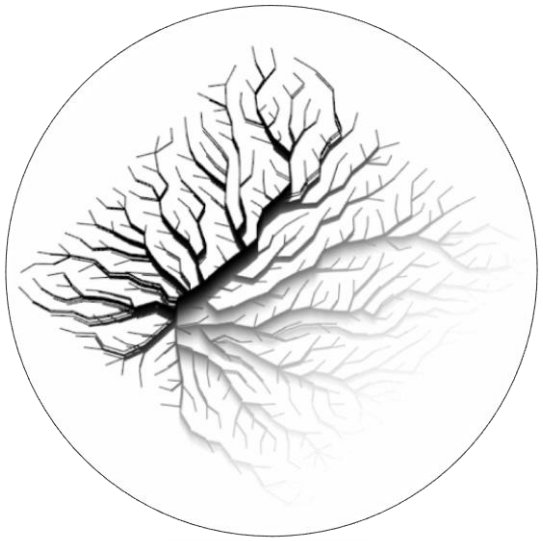
Χαρακτηριστικά της Τεχνικής

- Τα υπέρ και τα κατά της απληστίας :

14) Η άπληστη τεχνική ***προτιμάται*** έναντι άλλων μεθόδων, όπως είναι ο *δυναμικός προγραμματισμός*, όταν αποδειχθεί ότι δίδει βέλτιστη λύση σε μια κλάση προβλημάτων !!!

15) Παραδείγματα προβλημάτων και άπληστων αλγορίθμων βέλτιστης λύσης είναι:
   - οι γνωστοί μας αλγόριθμοι συγχώνευσης και τοπολογικής ταξινόμησης,
   - ο αλγόριθμος του Dijkstra για την εύρεση ελάχιστων διαδρομών,
   - οι αλγόριθμοι των Kruskal και Prim για την εύρεση ελάχιστων συνδετικών δέντρων, και
   - ο αλγόριθμος για την εύρεση βέλτιστων δέντρων Huffman.

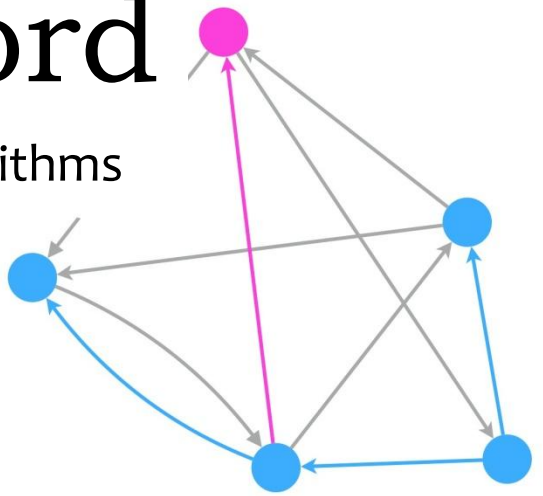
Ελάχιστες Διαδρομές



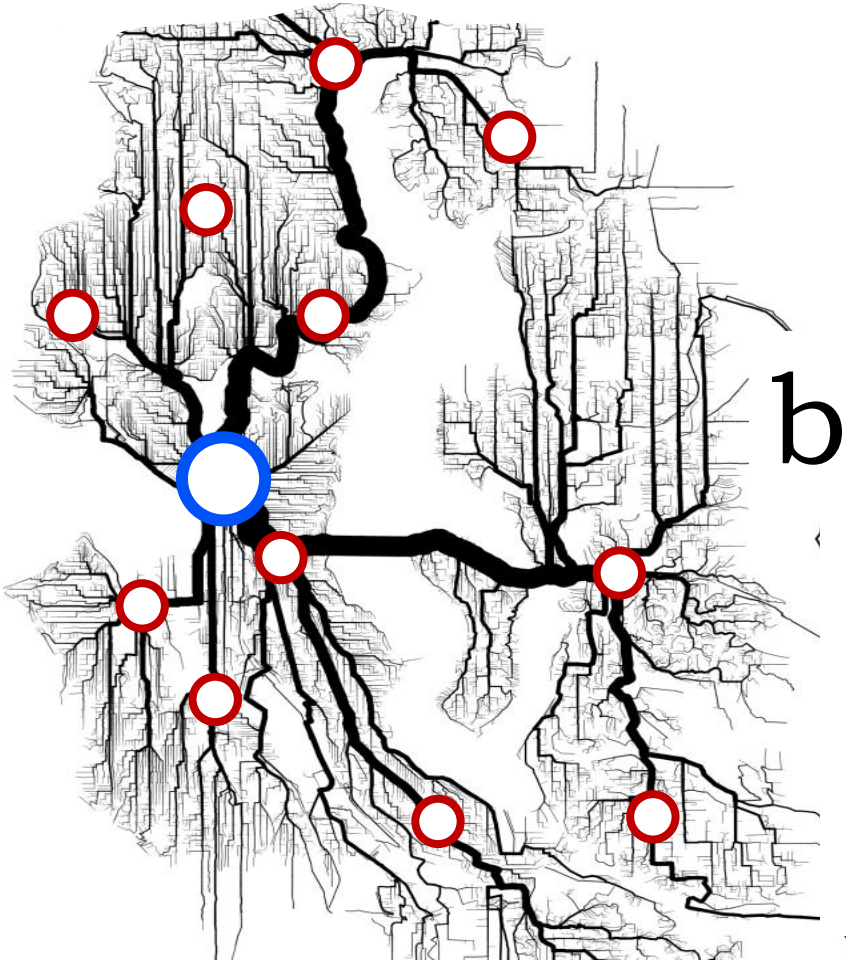
dijkstra

bellman-ford

shortest path algorithms



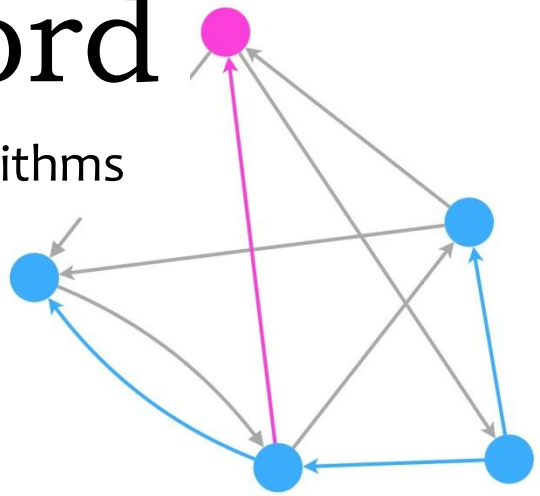
Ελάχιστες Διαδρομές



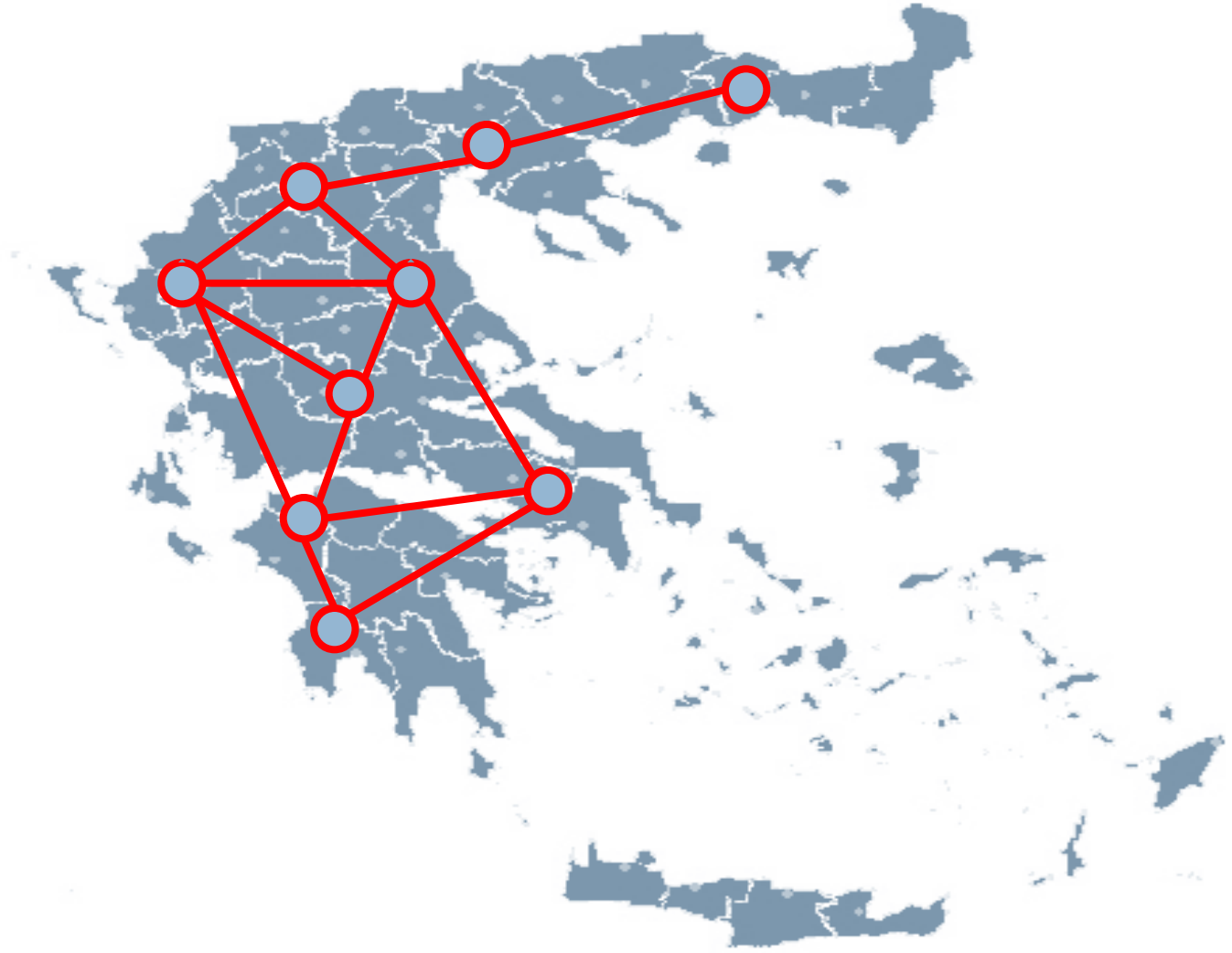
dijkstra

bellman-ford

shortest path algorithms

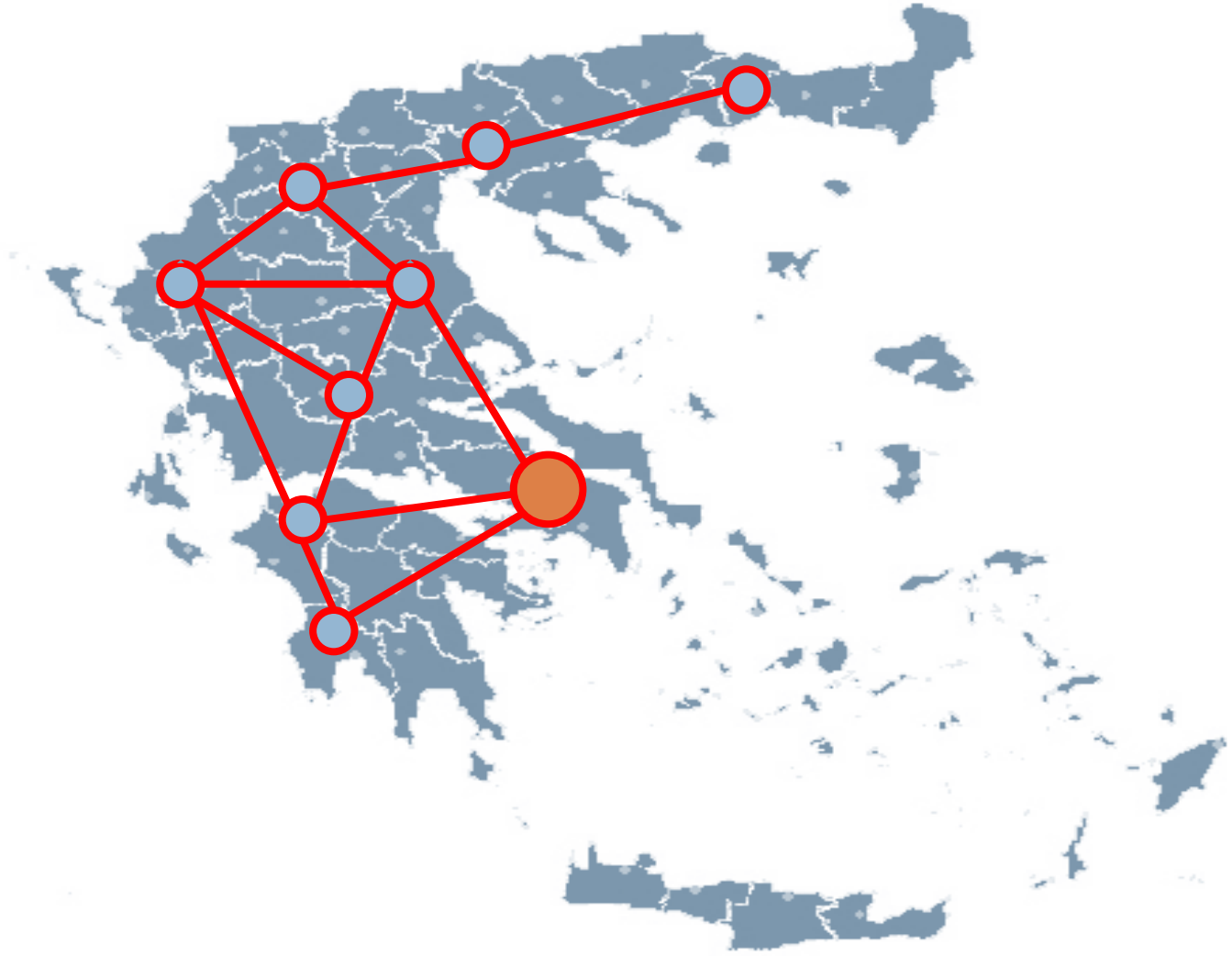


Πρόβλημα



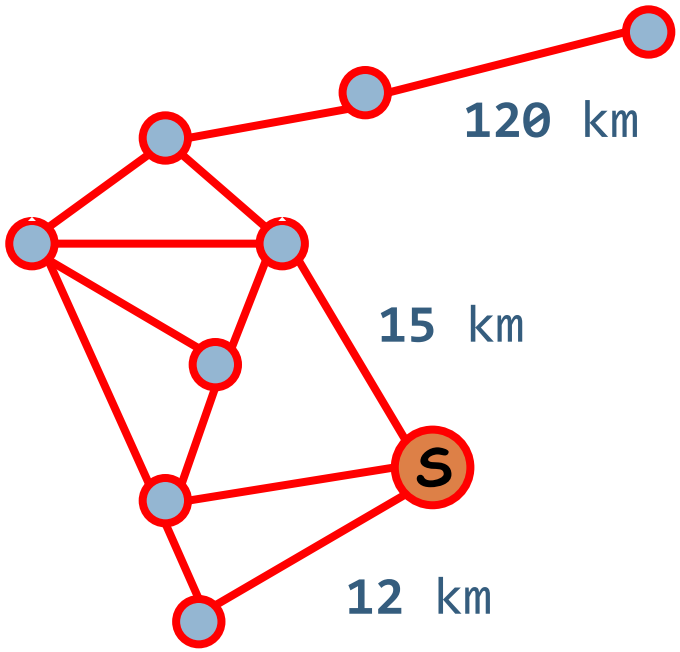
Οδικό Δίκτυο

Πρόβλημα



Οδικό Δίκτυο

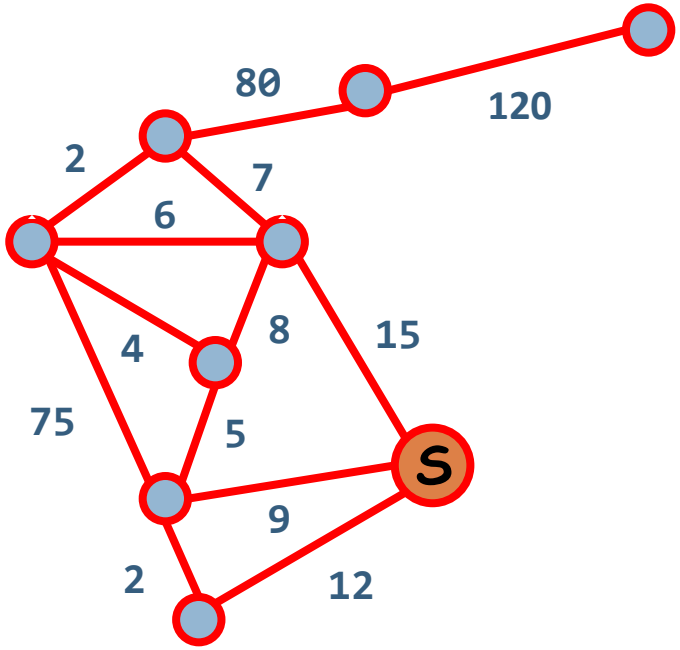
Πρόβλημα



Ελάχιστες Διαδρομές

από την πόλη S

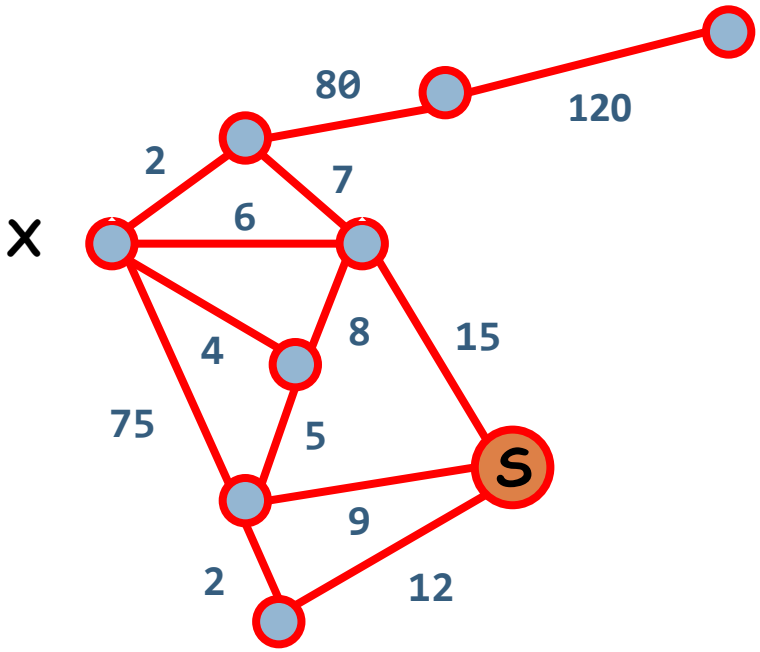
Πρόβλημα



Ελάχιστες Διαδρομές

από την πόλη S

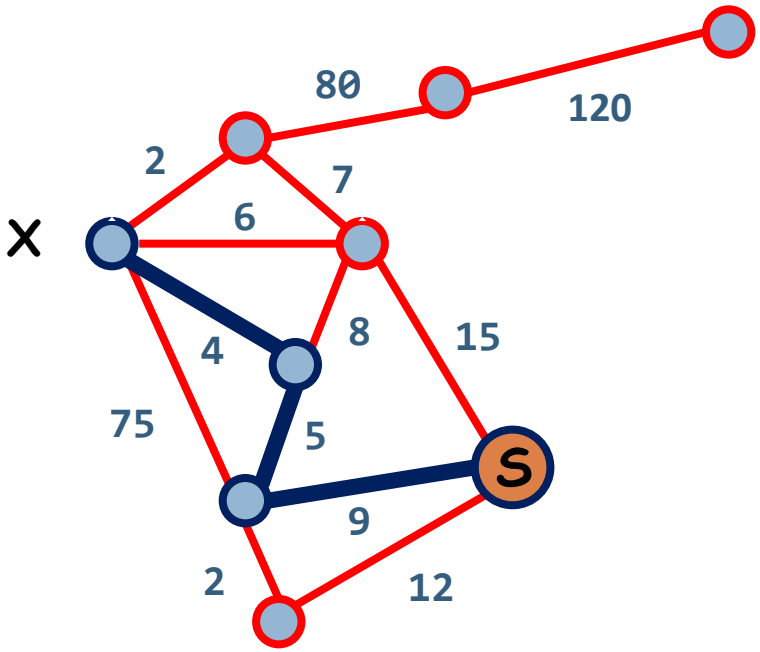
Πρόβλημα



Ελάχιστες Διαδρομές

από την πόλη S

Πρόβλημα

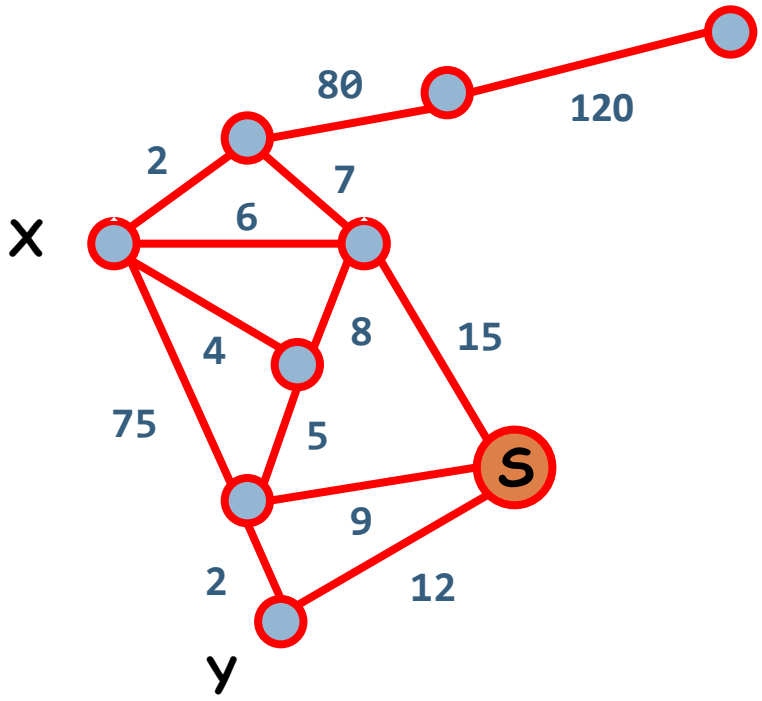


Ελάχιστες Διαδρομές

από την πόλη S

d(S, X) = 18

Πρόβλημα

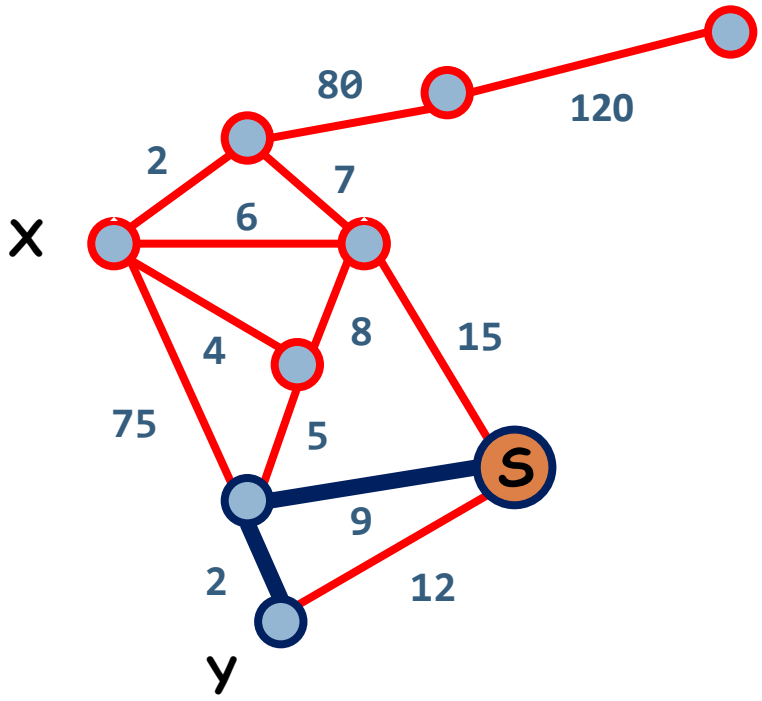


Ελάχιστες Διαδρομές

από την πόλη S

d(S, X) = 18

Πρόβλημα



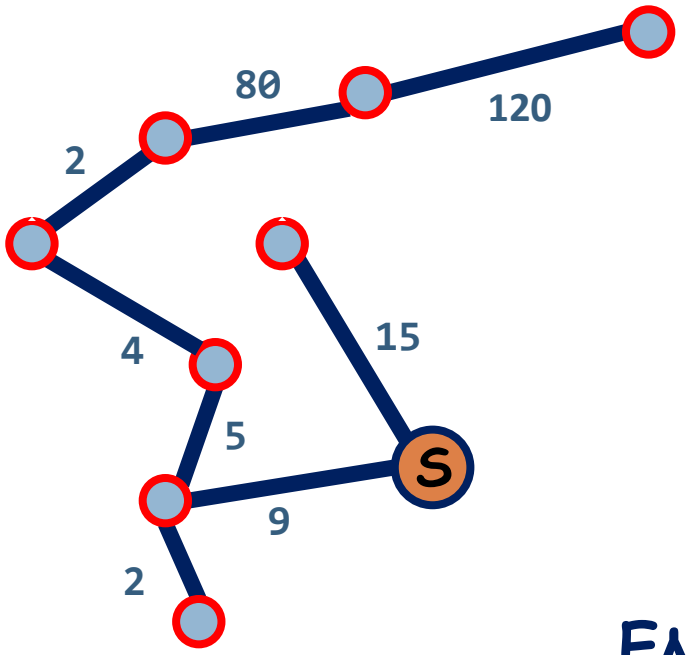
Ελάχιστες Διαδρομές

από την πόλη S

d(S, X) = 18

d(S, Y) = 11

Πρόβλημα



Ελάχιστες Διαδρομές

από την πόλη S

Δένδρο Ελαχίστων Διαδρομών

Βάρος ή Κόστος Διαδρομής

Έστω $G=(V,E)$ ένα γράφημα (*κατευθυνόμενο ή μη*), *έμβαρο* (δηλ. γράφημα στο οποίο κάθε ακμή φέρει κάποιο βάρος ή κόστος που κατά κανόνα δίδεται από μια *συνάρτηση βάρους* $w : E \rightarrow R$), και έστω

$$P = (v_1, v_2, \ldots, v_k)$$

μια διαδρομή του G από τον κόμβο v_1 στον v_k. Το *βάρος ή κόστος* της διαδρομής P ορίζεται ως εξής:

$$w(P) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Όροι: Διαδρομή = Μονοπάτι

Βάρος ή Κόστος Ελάχιστης Διαδρομής

- Ορίζουμε το βάρος ή κόστος μιας *ελάχιστης διαδρομής* (shortest path)

$$P = (v_1, v_2, \ldots, v_k)$$

ενός έμβαρου γραφήματος G από τον κόμβο v_1 στον v_k ως εξής:

$$d(v_1, v_k) \begin{cases} \min\{w(P): v_1 \rightsquigarrow v_k\} & \\ \infty & \text{σε κάθε άλλη περίπτωση} \end{cases}$$

Όροι: Ελάχιστη Διαδρομή = Συντομότατη Διαδρομή

Βασικές Παρατηρήσεις

Είδαμε ότι τα προβλήματα όπου η άπληστη τεχνική λειτουργεί καλά έχουν δύο κοινά χαρακτηριστικά:

- **Ιδιότητα άπληστης επιλογής** (Greedy choice property):

  Από ένα τοπικό βέλτιστο μπορούμε να επιτύχουμε ένα ολικό βέλτιστο χωρίς να αναθεωρούμε αποφάσεις που έχουμε ήδη πάρει !

- **Ιδιότητα Βέλτιστης υποδομής** (Optimal substructure property):

  Μια βέλτιστη λύση του προβλήματος μπορεί να κατασκευαστεί από βέλτιστες λύσεις των υποπροβλημάτων του !!

Εύλογο Ερώτημα

Τι ισχύει για το πρόβλημα των Ελαχίστων Διαδρομών ?

Βασικές Παρατηρήσεις

- **Ιδιότητα Βέλτιστης υποδομής** (Optimal substructure property):

  Οι αλγόριθμοι υπολογισμού ελάχιστων διαδρομών βασίζονται κατά κανόνα στην ιδιότητα ότι μία ελάχιστη διαδρομή `P : x ~> y` ενός γραφήματος εμπεριέχει άλλες ελάχιστες διαδρομές !!!

  Η Ιδιότητα της Βέλτιστης Υποδομής των ελάχιστων διαδρομών διατυπώνεται ακριβέστερα στο Λήμμα 1.

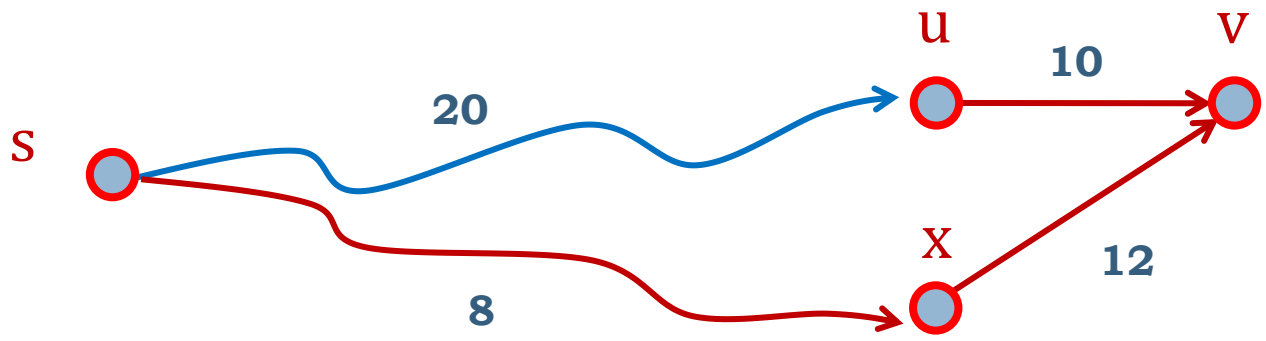
  Υπενθυμίζουμε ότι η Ιδιότητα της Βέλτιστης Υποδομής αποτελεί μία από τις βασικές «ενδείξεις» για την δυνατότητα εφαρμογής τόσο της *άπληστης τεχνικής* όσο και του *δυναμικού προγραμματισμού*.

  Παρατήρηση: Ο αλγόριθμος του Dijkstra ανήκει στην κατηγορία των άπληστων αλγορίθμων, ενώ ο αλγόριθμος των Floyd-Watshall, τον οποίο θα εξετάσουμε σε επόμενη ενότητα, εμπίπτει στον δυναμικό προγραμματισμό !!!

Βασικές Παρατηρήσεις

Έστω $G=(V,E)$ ένα έμβαρο γράφημα, $s \in V$ ένας κόμβος αφετηρίας και έστω $d(s,u)$ το κόστος της ελάχιστης διαδρομής $s \rightsquigarrow u$.
Τότε, για κάθε ακμή $(u,v) \in E$ ισχύει:

$$d(s,v) \leq d(s,u) + w(u,v)$$



Τριγωνική Ιδιότητα (Triangle Property)

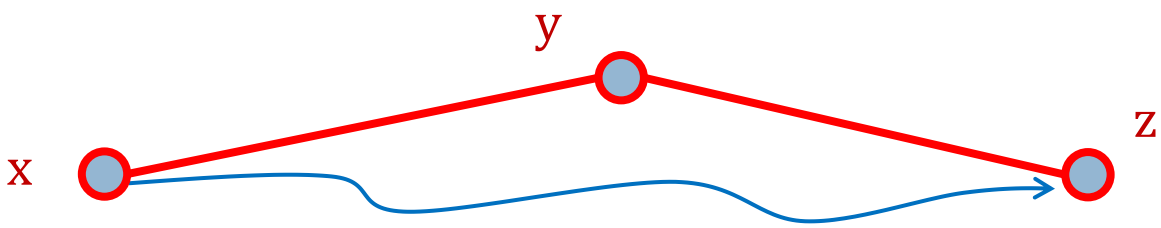
Βασικές Παρατηρήσεις

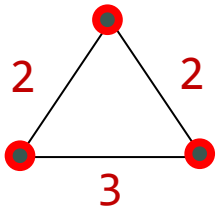
Έστω δύο **ελάχιστες** διαδρομές **P** και **Q** ενός γραφήματος.

P ελάχιστη διαδρομή $x \rightsquigarrow y$

Q ελάχιστη διαδρομή $y \rightsquigarrow z$

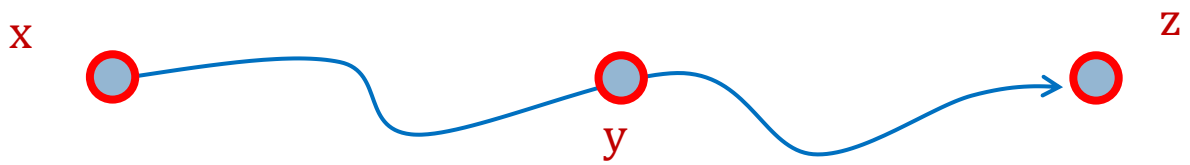
$\Rightarrow^{?}$ $P \cup Q$ ελάχιστη $x \rightsquigarrow z$





Βασικές Παρατηρήσεις

Έστω ότι η **ελάχιστη** διαδρομή `x ~> z`



η οποία αποτελείται από την διαδρομή `P : x ~> y` ακολουθούμενη από την `Q : y ~> z`

Τότε: `P` είναι **ελάχιστη** διαδρομή `x ~> y`

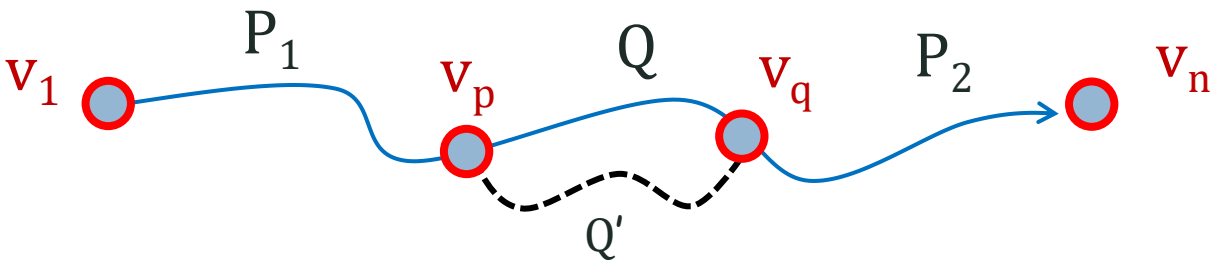
`Q` είναι **ελάχιστη** διαδρομή `y ~> z`

Ιδιότητα Βέλτιστης Υποδομής

Λήμμα 1 (Οι υποδιαδρομές από ε.δ αποτελούν ε.δ)

Έστω G έμβαρο, $P = (v_1, v_2, \ldots, v_n)$ μια ε.δ $v_1 \rightsquigarrow v_n$ και $Q = (v_p, v_{p+1}, \ldots, v_q)$ μια υποδιαδρομή $v_P \rightsquigarrow v_q$ της P .

Τότε, η υποδιαδρομή Q είναι **ελάχιστη** διαδρομή $v_P \rightsquigarrow v_q$



Εάν υπήρχε Q′ συντομότερη της Q, τότε P δεν θα ήταν ε.δ.

Ιδιότητα Βέλτιστης Υποδομής

Πόρισμα 1 (Οι υποδιαδρομές από ε.δ αποτελούν ε.δ)

Έστω ότι η ελάχιστη διαδρομή $P = (s, \ldots, v)$ διαμερίζεται

$$P: \; s \overset{P'}{\rightsquigarrow} u \rightarrow v$$

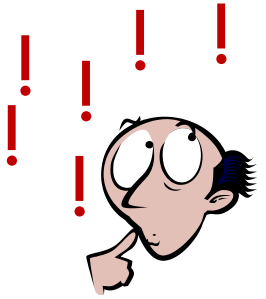
Τότε,

$$d(s,v) = d(s,u) + w(u,v)$$

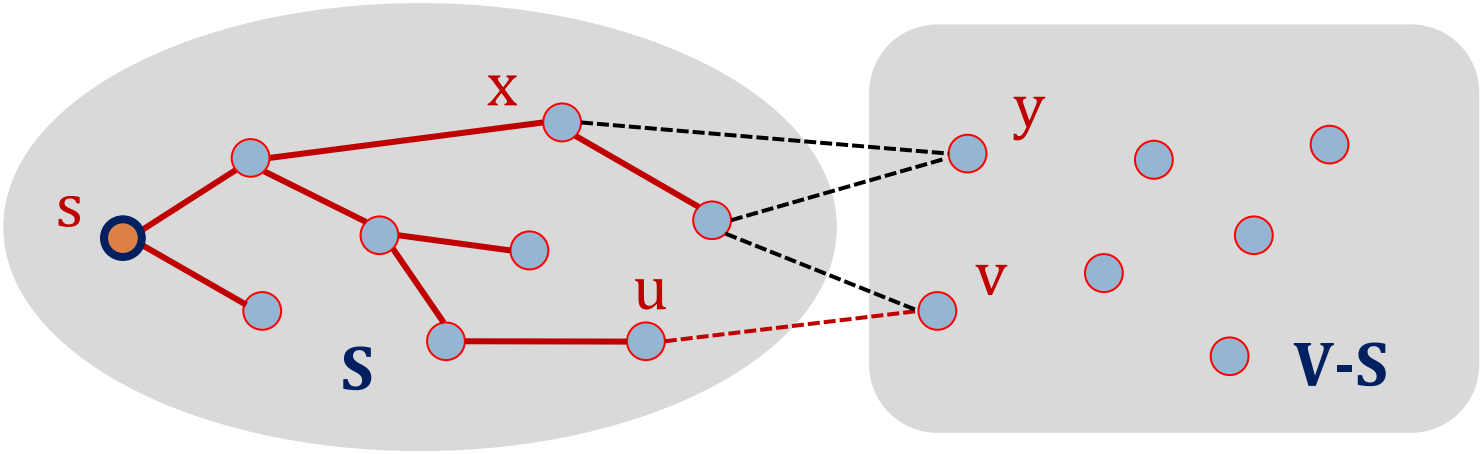
Απόδειξη: Από Λήμμα 1, η διαδρομή P' είναι ελάχιστη από το κόμβο s στο κόμβο u. Επομένως,

$$d(s,v) = w(P) = w(P') + w(u,v) = d(s,u) + w(u,v)$$ ■

1. Αλγόριθμος Dijkstra



Βασική Ιδέα !!!



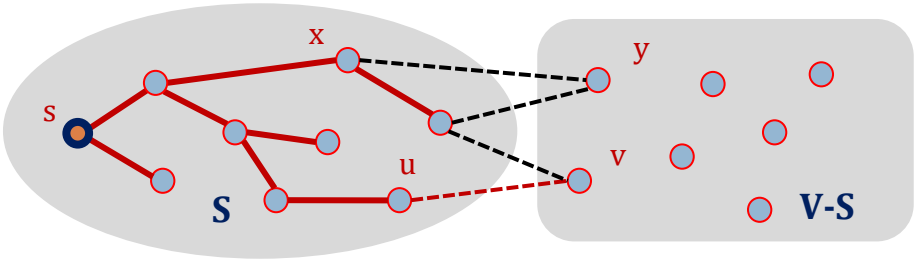
Επέκταση του Δένδρου Ελάχιστων Αποστάσεων κατά ένα κόμβο σε κάθε επανάληψη !... Άπληστη προσέγγιση !!!

Αλγόριθμος Dijkstra

Βασική Ιδέα !!!

Άπληστη προσέγγιση !!!

- Ο αλγόριθμος διατηρεί ένα σύνολο **S** από κόμβους u για τους οποίους έχει υπολογίσει το κόστος d(s,u)=d(u) των ε.δ από τον κόμβο s.

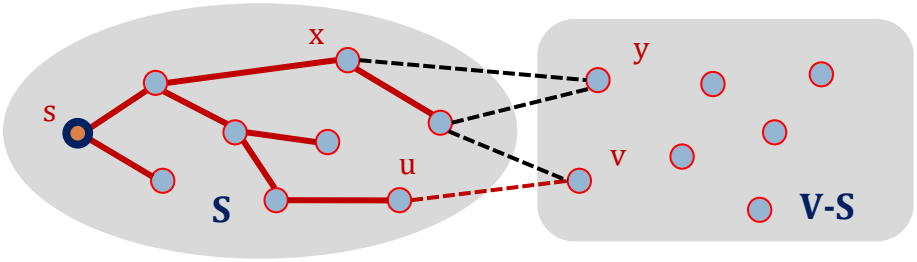


Δένδρο Ελαχίστων Διαδρομών με κόμβους του S

- Σε κάθε βήμα, ακλουθώντας ένα **κριτήριο άπληστης επιλογής**, κάνει τα εξής:
  - ελέγχει τους κόμβους v∈V-S για τους οποίους υπάρχει διαδρομή της μορφής (s, …,u,v), όπου (s, …,u) η ε.δ s ~> u στο S και (u,v)∈E, και επιλέγει αυτόν που έχει την *ελάχιστη τέτοια διαδρομή* και τον προσθέτει στο S.

Αλγόριθμος Dijkstra

Βασική Ιδέα !!!



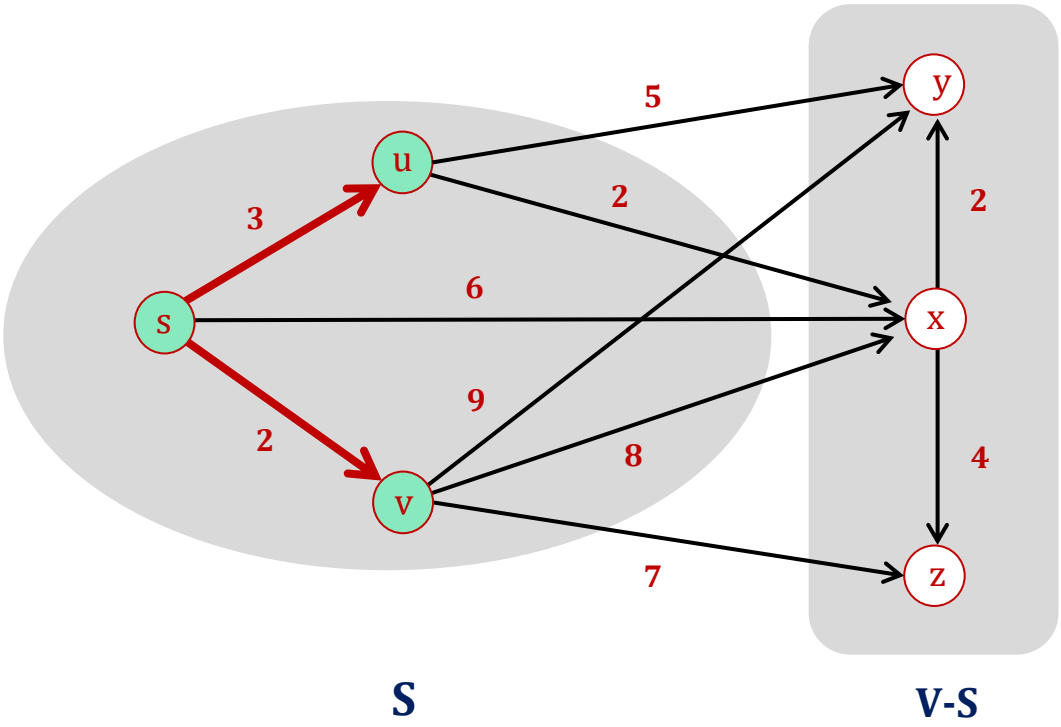
Άπληστη προσέγγιση !!!

- Αρχικά $S=\{s\}$ και του $d(s)=0$
- Στο βήμα i ($2 \le i \le n$), κάνει τα εξής:

  (1) για κάθε κόμβο $v \in V-S$ για τον οποίο $\exists$ διαδρομή της μορφής $(s, \ldots, u, v)$, υπολογίζει το κόστος $d'(v)$ της συντομότερης διαδρομής μεταξύ όλων των διαδρομών αυτής της μορφής, δηλ. $d'(v) = \min\{d(u) + w(u,v)\}$ $\forall\, u \in S : (u,v) \in E$, και

  (2) επιλέγει τον κόμβο $v \in V-S$ που ελαχιστοποιεί την ποσότητα $d'(v)$ μεταξύ όλων των κόμβων $v \in V-S$ για τους οποίους $\exists$ διαδρομή $(s, \ldots, u, v)$ και τον προσθέτει στο S θέτοντας $d(v) = d'(v)$.

Παράδειγμα



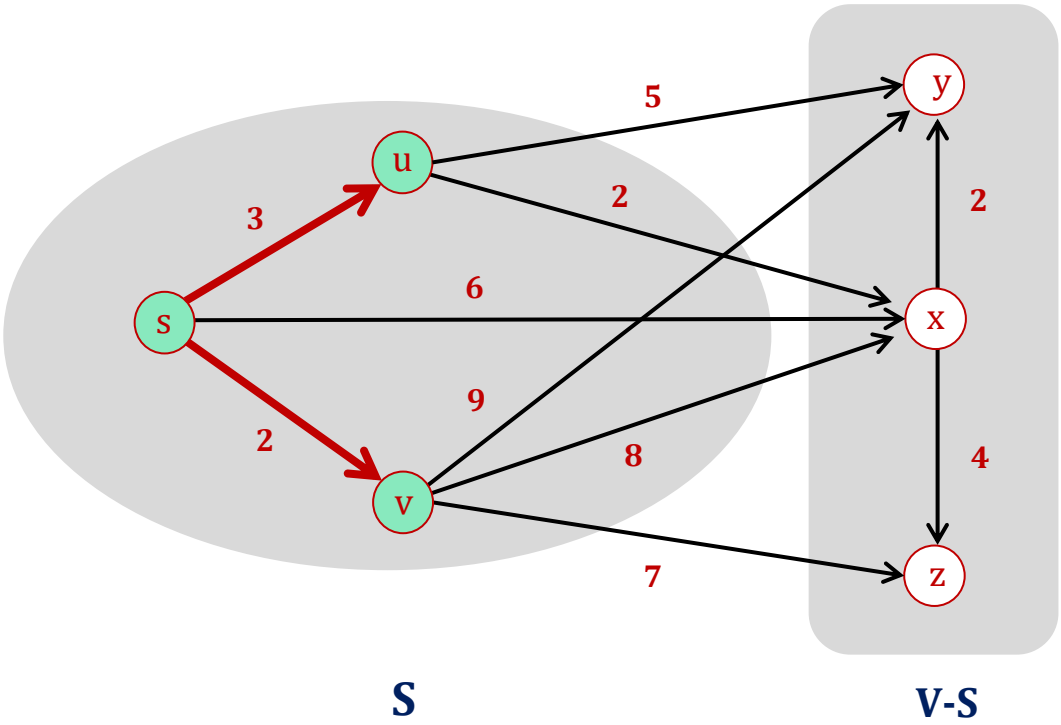
Μια ενδιάμεση κατάσταση εκτέλεσης του Αλγόριθμου

Στο Σχήμα έχουν εκτελεστεί 3 Βήματα:

- Το αρχικό (αρχικοποίηση), και δύο επαναλήψεις.
- Στο σύνολο S έχουν προστεθεί 3 κόμβοι: ο κόμβος εκκίνησης s, και οι κόμβοι v και u.

Αλγόριθμος Dijkstra

Παράδειγμα



d(s) = 0
d(v) = 2
d(u) = 3 d(x) = ∞, d(y) = ∞, d(z) = ∞

Βήμα 1

Αρχικοποίηση: S={s} και d(s)=0

Βήμα 2

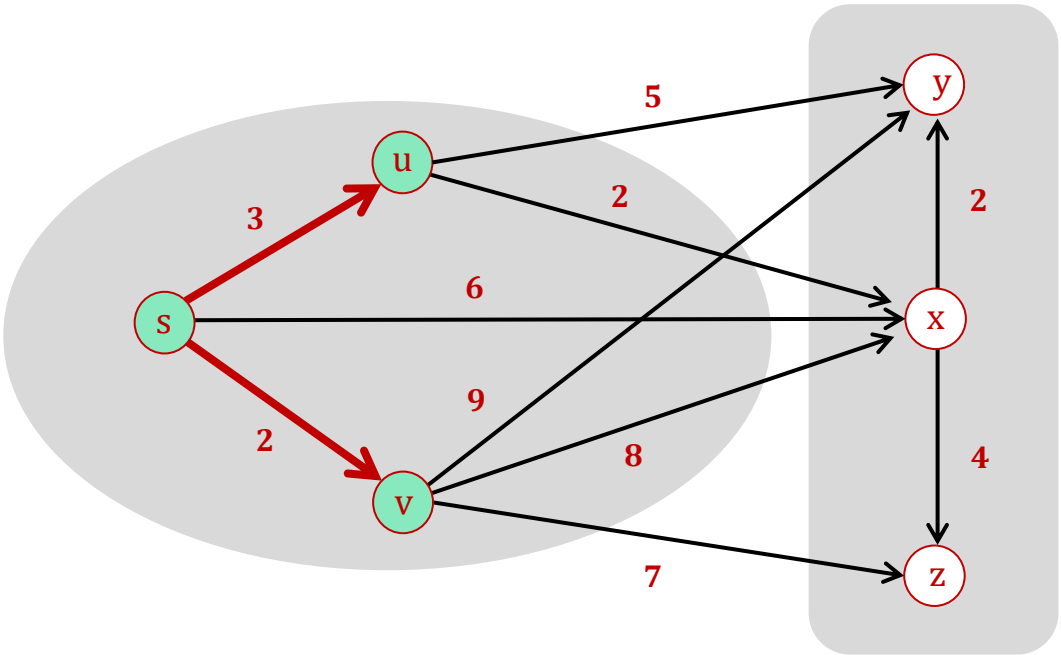
Επέλεξε τον κόμβο v∈V-S,
και τον πρόσθεσε στο S θέτοντας
d(v) = 2

Βήμα 3

Επέλεξε τον κόμβο u∈V-S,
και τον πρόσθεσε στο S θέτοντας
d(u) = 3

Αλγόριθμος Dijkstra

Παράδειγμα



d(s) = 0
d(v) = 2
d(u) = 3 d(x) = ∞, d(y) = ∞, d(z) = ∞

Βήμα 4

d'(v) = min{d(u) + w(u,v)}

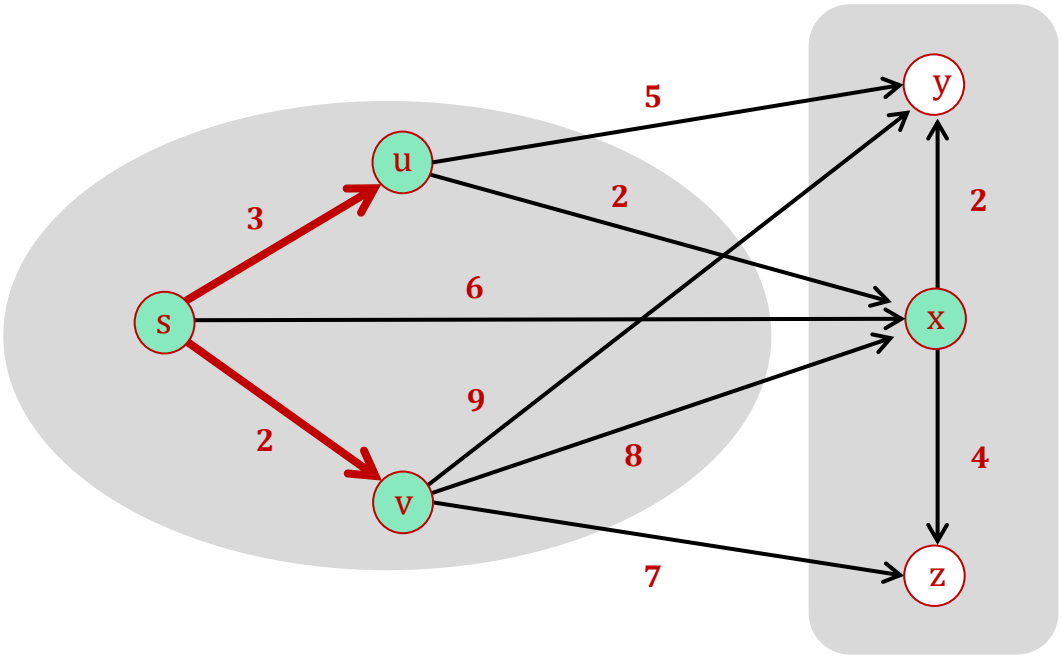
d'(y) = min{3+5,2+9} = 8

d'(x) = min{3+2,0+6,2+8} = 5

d'(z) = min{2+7} = 9

Αλγόριθμος Dijkstra

Παράδειγμα



d(s) = 0
d(v) = 2
d(u) = 3 d(x) = ∞, d(y) = ∞, d(z) = ∞

Βήμα 4

d′(v) = min{d(u) + w(u,v)}

d′(y) = min{3+5,2+9} = 8

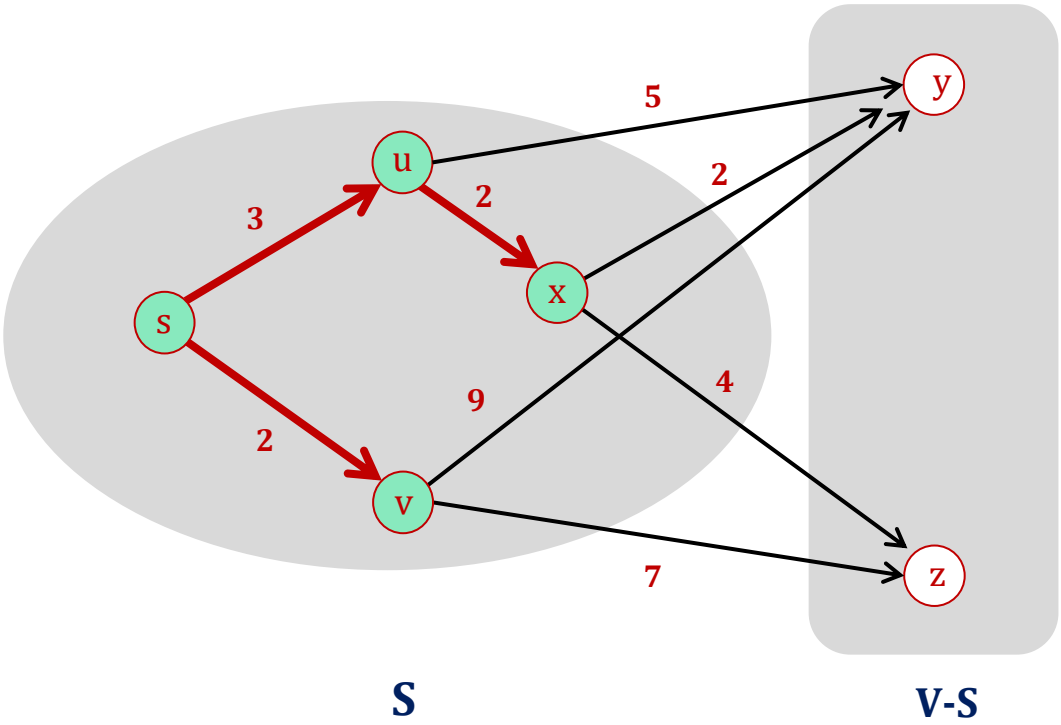
d′(x) = min{3+2,0+6,2+8} = 5

d′(z) = min{2+7} = 9

Επιλέγει τον κόμβο x∈V-S διότι ελαχιστοποιεί την ποσότητα d′(x) μεταξύ των κόμβων {y,x,z}.

Αλγόριθμος Dijkstra

Παράδειγμα



d(s) = 0
d(v) = 2
d(u) = 3 d(x) = 5, d(y) = ∞, d(z) = ∞

Βήμα 4

d'(v) = min{d(u) + w(u,v)}

d'(y) = min{3+5,2+9} = 8

d'(x) = min{3+2,0+6,2+8} = 5

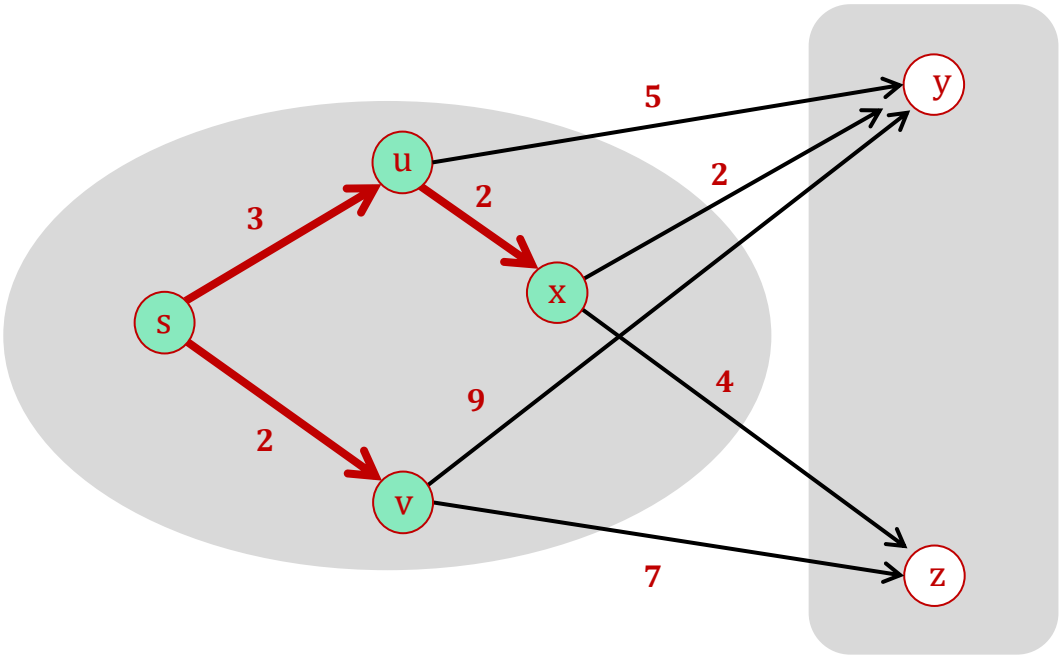
d'(z) = min{2+7} = 9

Προσθέτει τον κόμβο x στο S θέτοντας d(v) = 5

Το Δένδρο Ελαχίστων Διαδρομών επεκτείνεται με τον κόμβο x και με την ακμή (u,x)… **γιατί ?**

Αλγόριθμος Dijkstra

Παράδειγμα



d(s) = 0
d(v) = 2
d(u) = 3 d(x) = 5, d(y) = ∞, d(z) = ∞

Βήμα 5

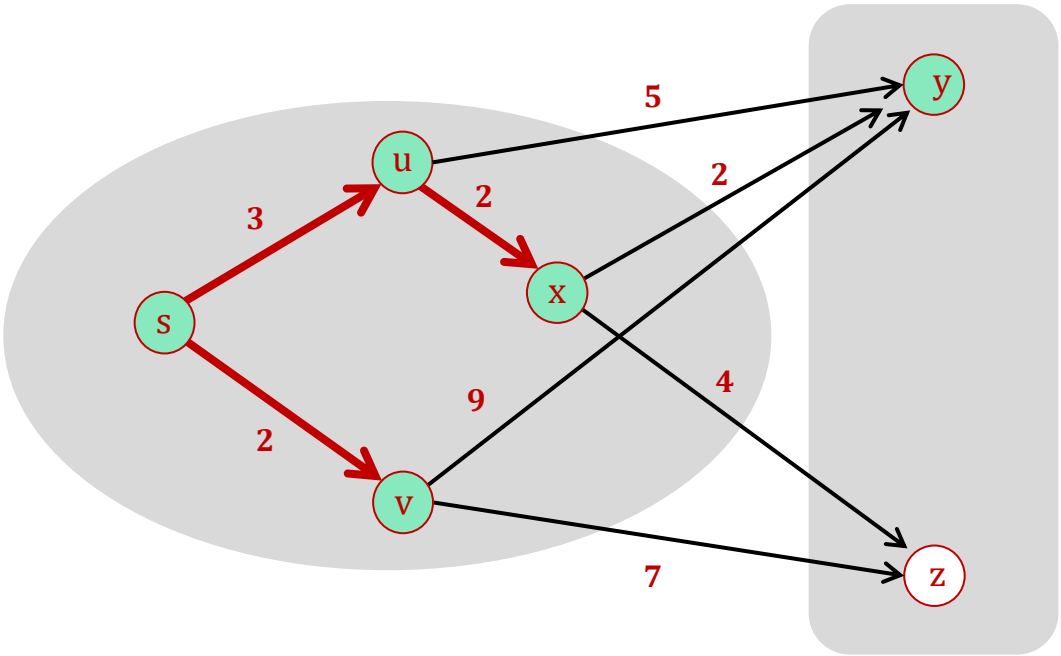
d'(v) = min{d(u) + w(u,v)}

d'(y) = min{3+5,5+2,2+9} = 7

d'(z) = min{5+4,2+7} = 9

Αλγόριθμος Dijkstra

Παράδειγμα



d(s) = 0
d(v) = 2
d(u) = 3 d(x) = 5, d(y) = ∞, d(z) = ∞

Βήμα 5

d'(v) = min{d(u) + w(u,v)}

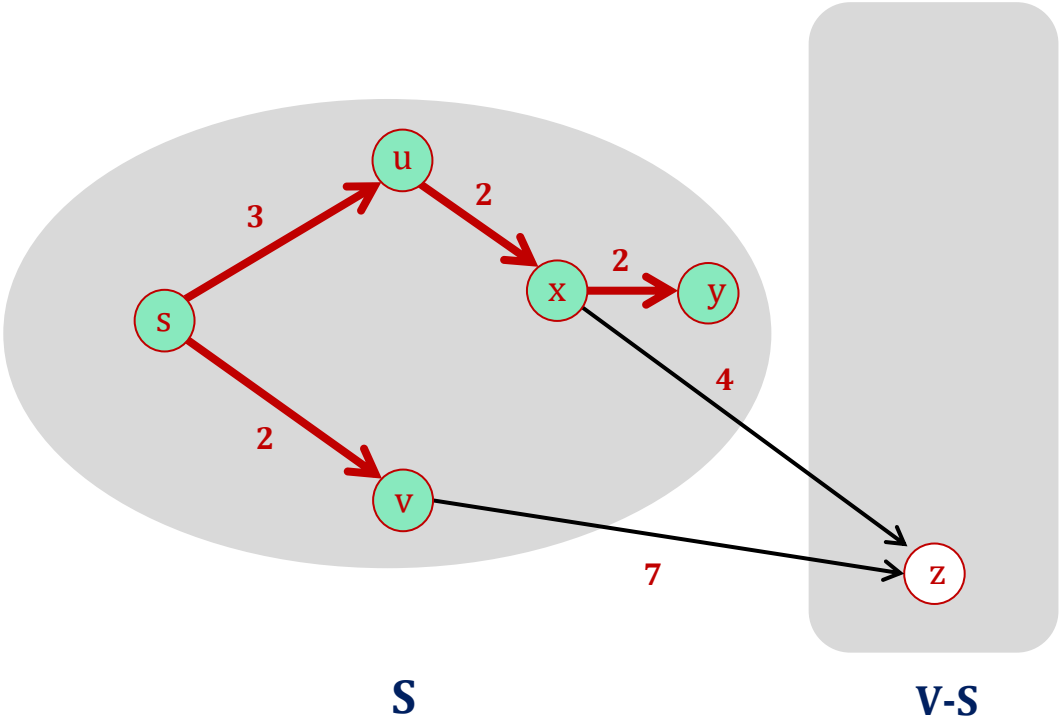
d'(y) = min{3+5,5+2,2+9} = 7

d'(z) = min{5+4,2+7} = 9

Επιλέγει τον κόμβο y∈V-S διότι ελαχιστοποιεί την ποσότητα d'(y) μεταξύ των κόμβων {y,z}.

Αλγόριθμος Dijkstra

Παράδειγμα



d(s) = 0
d(v) = 2
d(u) = 3 d(x) = 5, d(y) = 7, d(z) = ∞

Βήμα 5

d'(v) = min{d(u) + w(u,v)}

d'(y) = min{3+5,5+2,2+9} = 7

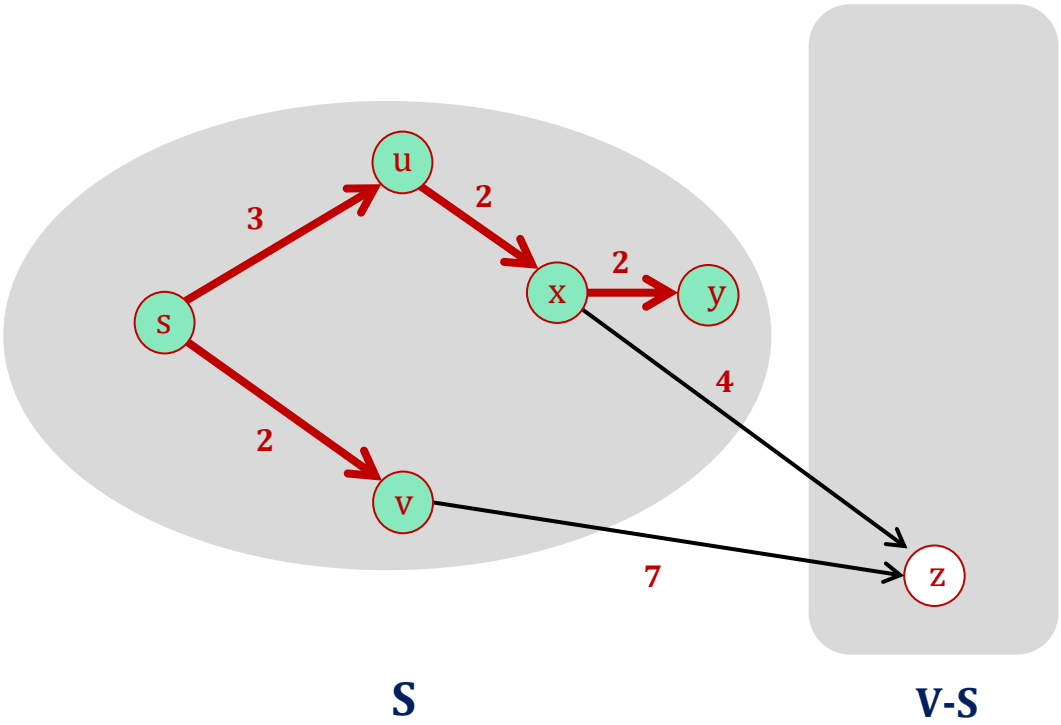
d'(z) = min{5+4,2+7} = 9

Προσθέτει τον κόμβο y στο S θέτοντας d(y) = 7

Το Δένδρο Ελαχίστων Διαδρομών επεκτείνεται με τον κόμβο y και με την ακμή (x,y) !!!

Αλγόριθμος Dijkstra

Παράδειγμα



d(s) = 0
d(v) = 2
d(u) = 3 d(x) = 5, d(y) = 7, d(z) = ∞

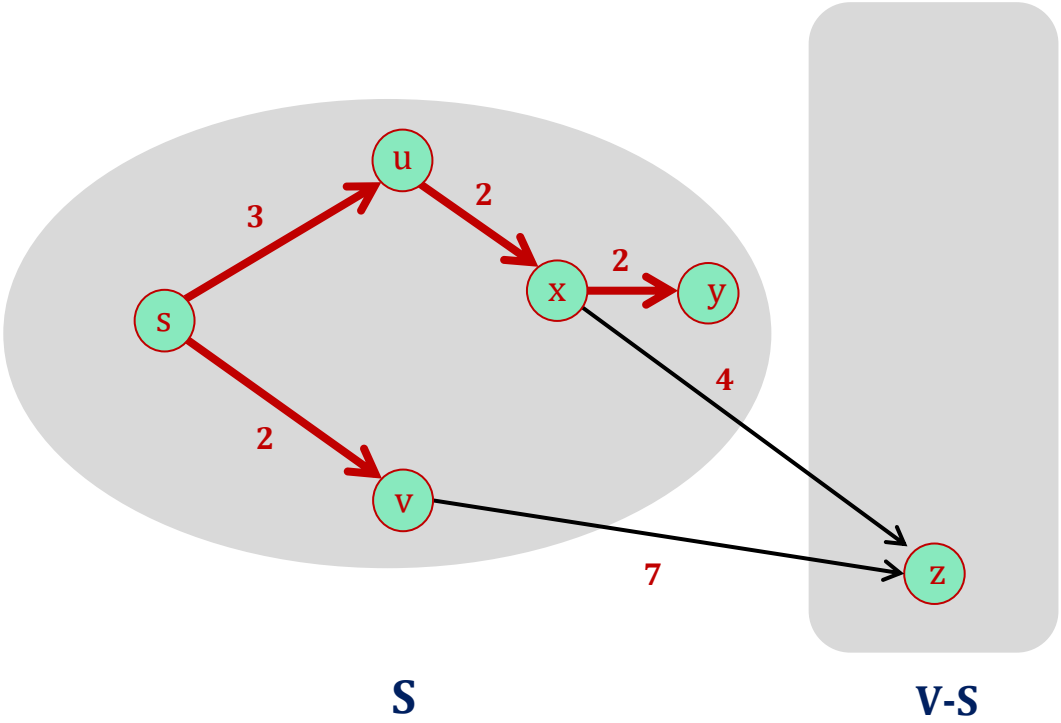
Βήμα 6

d'(v) = min{d(u) + w(u,v)}

d'(z) = min{5+4,2+7} = 9

Αλγόριθμος Dijkstra

Παράδειγμα



d(s) = 0
d(v) = 2
d(u) = 3 d(x) = 5, d(y) = 7, d(z) = ∞

Βήμα 6

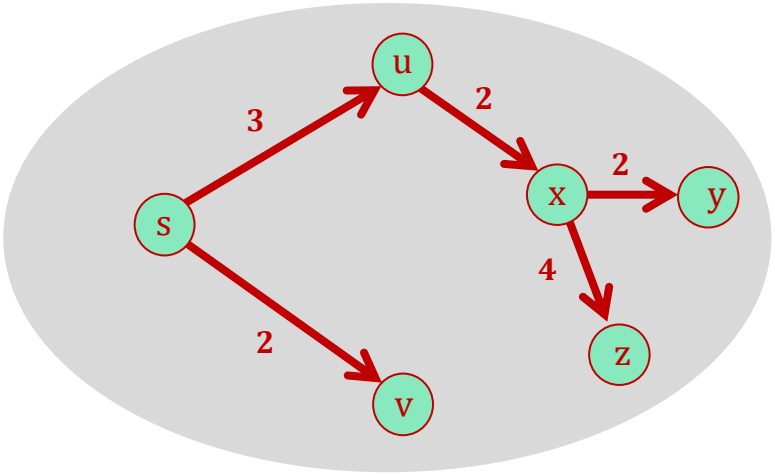
d'(v) = min{d(u) + w(u,v)}

d'(z) = min{5+4,2+7} = 9

Επιλέγει τον κόμβο z∈V-S διότι ελαχιστοποιεί την ποσότητα d'(z) μεταξύ των κόμβων {z}.

Αλγόριθμος Dijkstra

Παράδειγμα



S **V-S**

d(s) = 0
d(v) = 2
d(u) = 3 d(x) = 5, d(y) = 7, d(z) = 9

Βήμα 6

d'(v) = min{d(u) + w(u,v)}

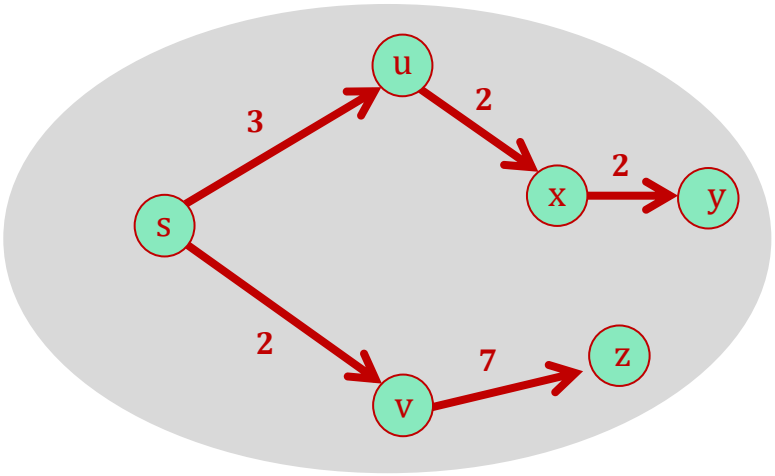
d'(z) = min{5+4,2+7} = 9

Προσθέτει τον κόμβο z στο S θέτοντας d(z) = 9

Το Δένδρο Ελαχίστων Διαδρομών επεκτείνεται με τον κόμβο z και με την ακμή (x,z) !!!

Αλγόριθμος Dijkstra

Παράδειγμα



S **V-S**

d(s) = 0
d(v) = 2
d(u) = 3 d(x) = 5, d(y) = 7, d(z) = 9

Βήμα 6

d'(v) = min{d(u) + w(u,v)}

d'(z) = min{5+4,2+7} = 9

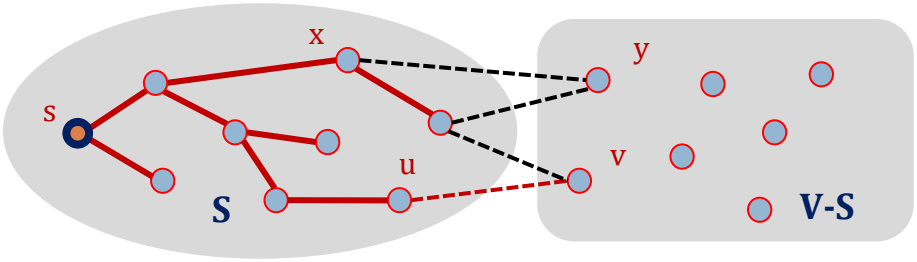
Προσθέτει τον κόμβο z στο S θέτοντας d(z) = 9

Το Δένδρο Ελαχίστων Διαδρομών επεκτείνεται με τον κόμβο z και με την ακμή (x,z) !!!

Ισοδύναμη επιλογή

Κόμβος z και ακμή (v,z) !!!

Ορθότητα !!!



- Στο παράδειγμα είδαμε ότι ο αλγόριθμος του Dijjstra κάνει το σωστό και αποφεύγει τις «παγίδες»!!!

- Η επαύξηση του συνόλου S με λανθασμένο κόμβο v μπορεί να οδηγήσει σε *υπερεκτίμηση* του κόστους της ε.δ από τον s προς τον v !!!

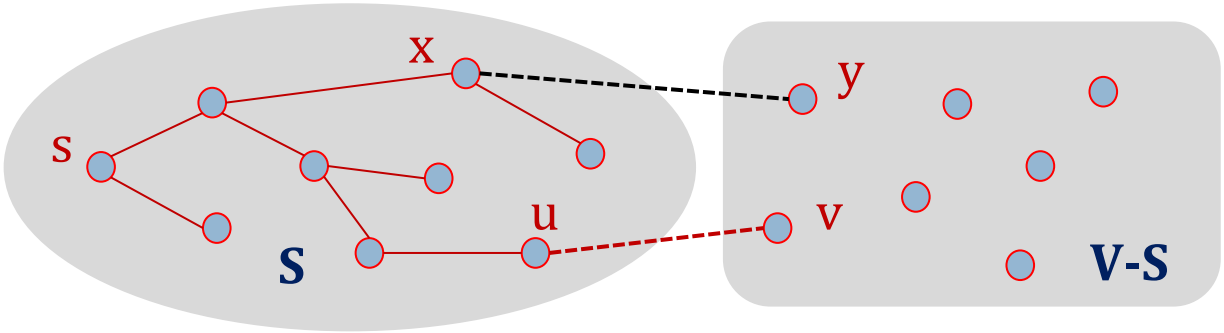
Εύλογο Ερώτημα:

Είναι πάντοτε αληθές ότι, όταν ο αλγόριθμος του Dijjstra προσθέτει έναν κόμβο v στο σύνολο S, *παίρνουμε πραγματικά το κόστος της ελάχιστης διαδρομής* από τον s προς τον v ?

Ορθότητα Αλγόριθμου Dijkstra

Θεώρημα 1

Έστω G=(V,E) έμβαρο γράφημα με μη-αρνητικά βάρη, έστω s∈V ένας κόμβος αφετηρίας και (S,V-S) μια διαμέριση του V : s∈S και d(s,u) ∀u∈S.



Εάν (u,v) είναι η ακμή που ελαχιστοποιεί την ποσότητα

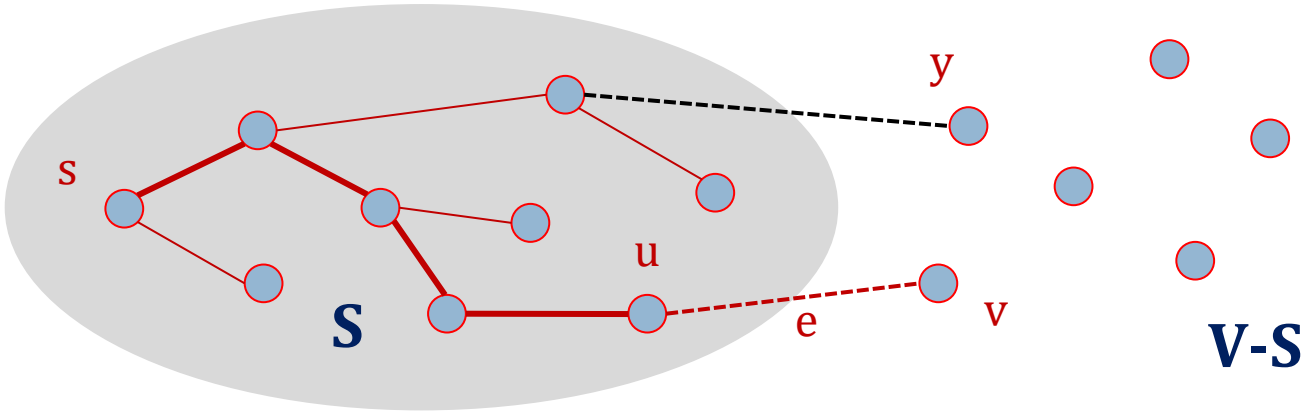
d(s,u) + w(u,v)

μεταξύ όλων των ακμών (x, y) με x∈S και y∈V-S, τότε

P = (s, …,u,v) είναι ε.δ. s ~> v.

Ορθότητα Αλγόριθμου Dijkstra

Απόδειξη



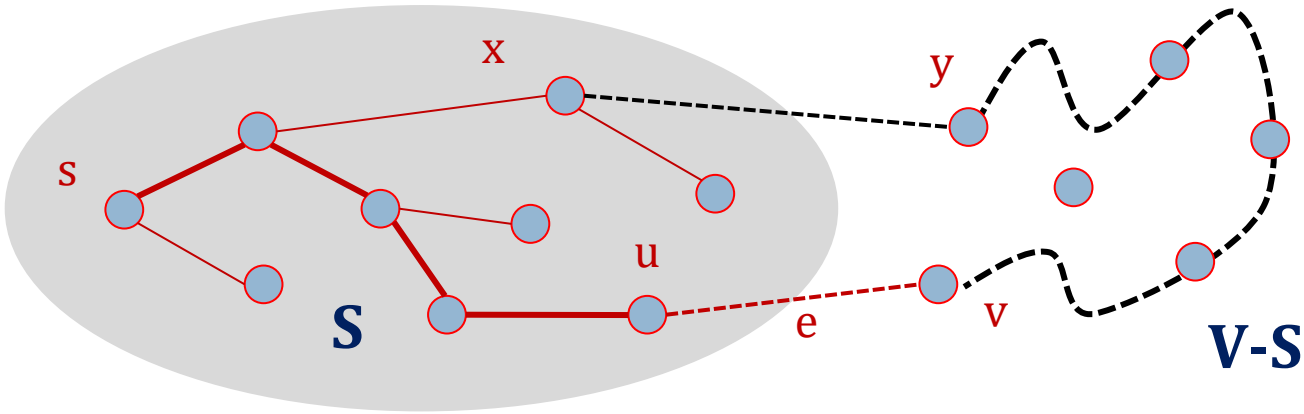
Έστω e = (u,v) και έστω (s, …,u) η ελάχιστη διαδρομή s ~> u.

Για την **διαδρομή P = (s, …,u,v)** από τον s στον v ισχύει :

$$w(P) = d(s,u) + w(e) \qquad (1)$$

Ορθότητα Αλγόριθμου Dijkstra

Απόδειξη

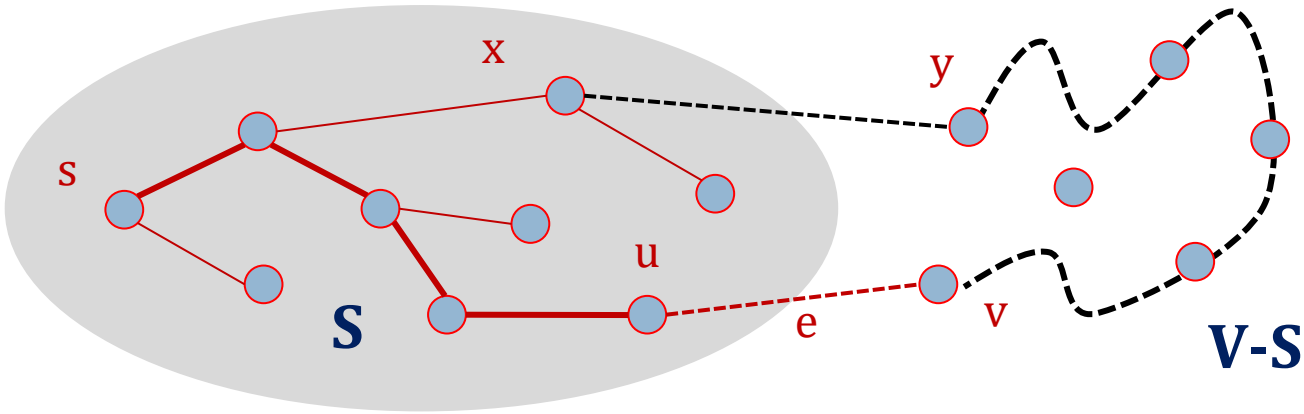


Έστω Q = (s, …,x,y, …,v) μια ελάχιστη διαδρομή s ~> v στο G, και έστω y ο πρώτος κόμβος της διαδρομής Q : y∈V-S.

Θα δείξουμε ότι **w(P) ≤ w(Q).**

Ορθότητα Αλγόριθμου Dijkstra

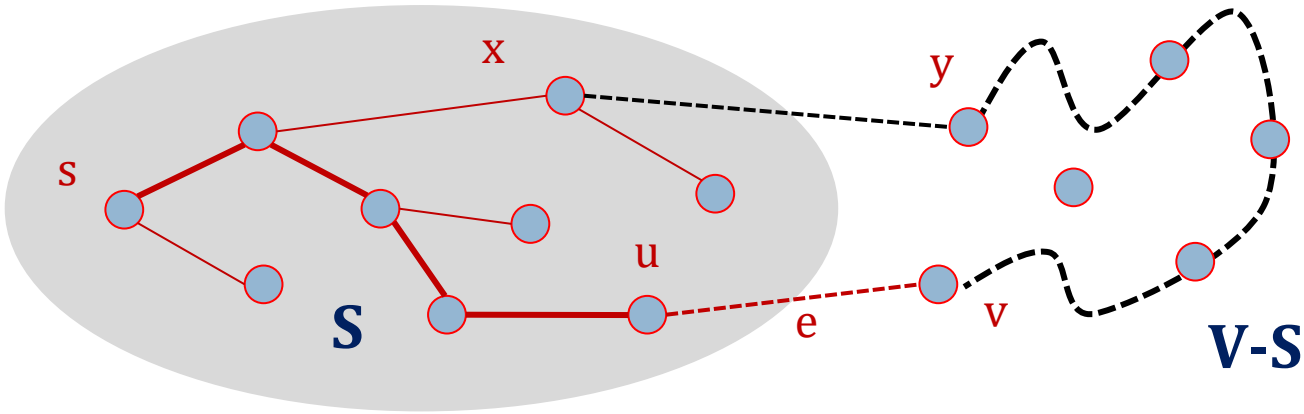
Απόδειξη



Από την Eq.(1) και από το κριτήριο επιλογής της ακμής e = (u,v), έχουμε:

$$w(P) = d(s,u) + w(e) \leq d(s,x) + w(x,y) \quad (2)$$

Απόδειξη

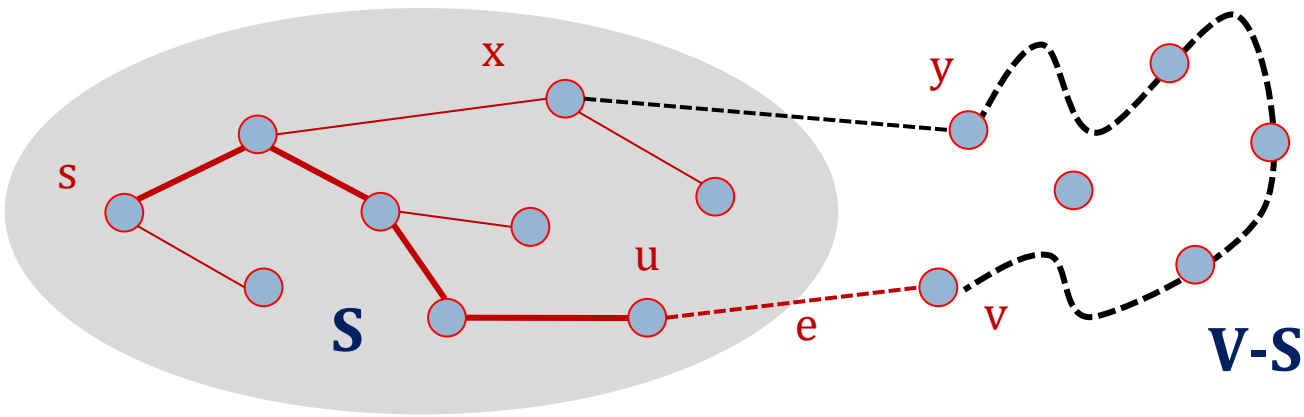


Η διαδρομή (s, …,x) είναι ελάχιστη με τιμή d(s,x).

Από την Ιδιότητα Βέλτιστης Υποδομής, επειδή (s, …,x,y) είναι υποδιαδρομή της ε.δ Q = (s, …,x,y, …,v) και οι ακμές έχουν μη-αρνητικά βάρη, ισχύει:

$$d(s,x) + w(x,y) \leq w(Q) \qquad (3)$$

Απόδειξη



Από Eq.(2) και Eq.(3), δηλ.

$$\text{Eq.(2)} : w(P) = d(s,u) + w(e) \leq d(s,x) + w(x,y)$$

$$\text{Eq.(3)} : d(s, x) + w(x, y) \leq w(Q)$$

παίρνουμε:

$$\mathbf{w(P) \leq w(Q)}$$

■

Ορθότητα Αλγόριθμου Dijkstra

Θεώρημα 2 (Ορθότητα Αλγορίθμου Dijkstra)

Έστω $G=(V,E)$ έμβαρο γράφημα με μη-αρνητικά βάρη και $s \in V$. Ο αλγόριθμος του Dijkstra υπολογίζει τις ε.δ από τον κόμβο s προς κάθε άλλο κόμβο $v \in V$ του G που είναι προσπελάσιμος από τον s.

Απόδειξη

- ✓ Με επαγωγή στην ακολουθία των κόμβων όπως αυτή δημιουργείται από την πρόσθεσή τους στο δένδρο ελάχιστων-διαδρομών.
- ✓ Χρήση του Θεωρήματος 1.

Δύο Σημαντικές Παρατηρήσεις !!!

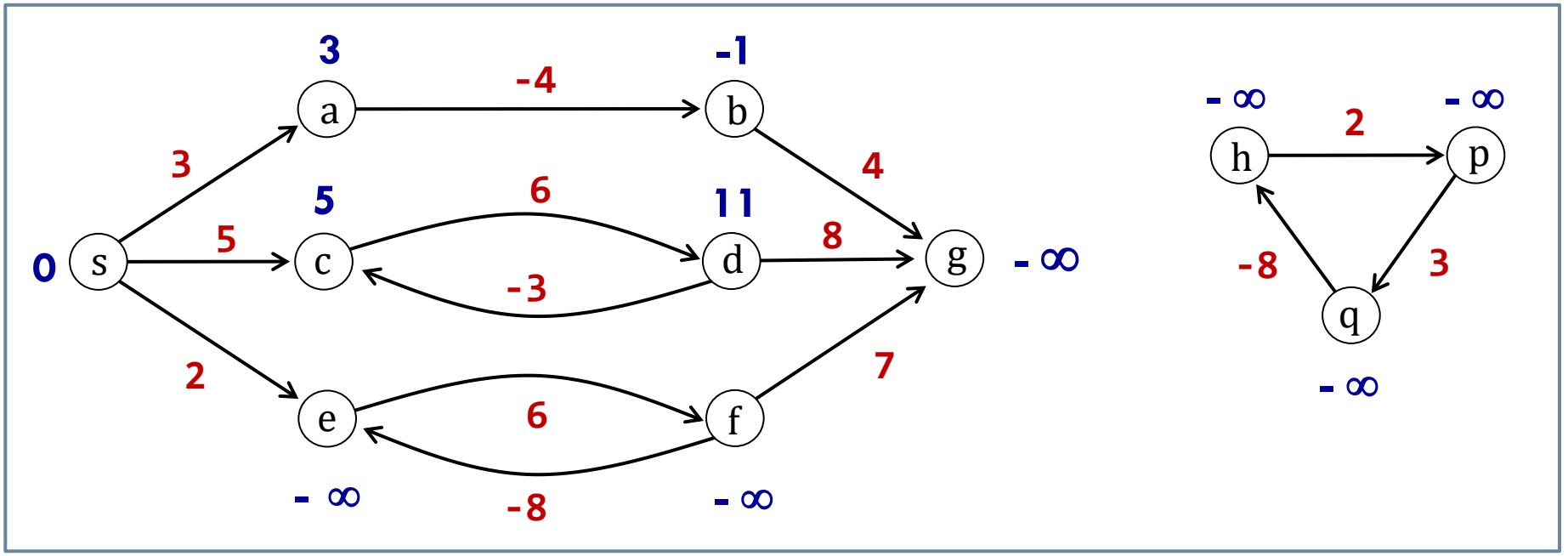
Παρατήρηση 1. Ο αλγόριθμος του Dijkstra δεν δίνει πάντα σωστό αποτέλεσμα όταν το γράφημα περιέχει ακμές με αρνητικά βάρη !!!

Πολλές εφαρμογές περιλαμβάνουν αρνητικά βάρη ακμών, και σε αυτή την περίπτωση απαιτείται ένας πιο πολύπλοκος αλγόριθμος – όπως αυτός των Bellman-Ford !!!

Παρατήρηση 2. Ο αλγόριθμος του Dijkstra είναι, κατά μία έννοια, πολύ πιο απλούστερος απ' ότι τον περιγράψαμε !!!

Ο αλγόριθμος στην πραγματικότητα είναι μια «έξυπνη» εκδοχή του αλγορίθμου διερεύνησης BFS, και το κίνητρό του προέρχεται από μια φυσική διαισθητική ιδέα που θα περιγράψουμε στη συνέχεια !!!

1. Αρνητικά Βάρη !!!



- ✓ Κόστος ελάχιστης διαδρομής: `d(s,d) = w(s,c) + w(c,d) = 11`
- ✓ Διαδρομές: `(s,e,f), (s,e,f,e,f), (s,e,f,e,f,e,f), (s,e,f, …, e,f)`
- ✓ Κύκλος αρνητικού κόστους: `(e,f,e) = w(e,f) + w(f,e) = -2 < 0`

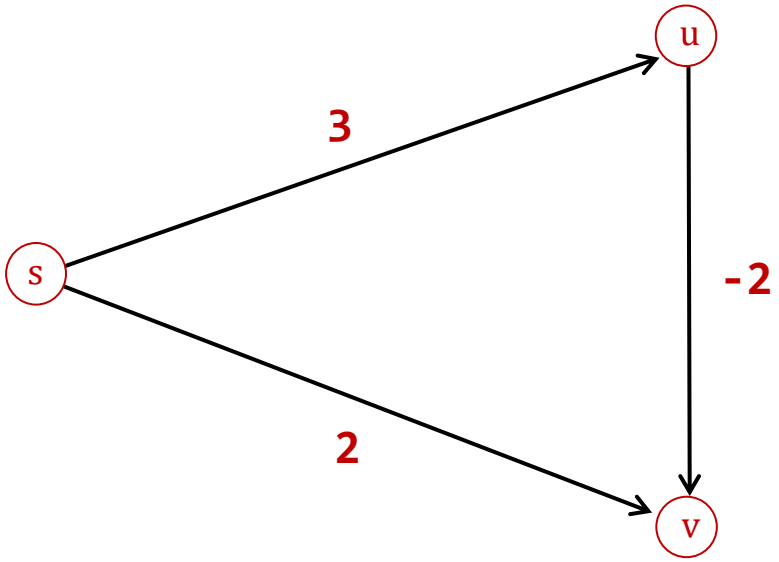
1. Αρνητικά Βάρη !!!

Αντιπαράδειγμα

Ελάχιστη διαδρομή P από τον s στον v:

P = (s,u,v)

d(v) = 1



Ο αλγόριθμος του Dijkstra δεν δίνει πάντα σωστό αποτέλεσμα όταν το γράφημα περιέχει ακμές με αρνητικά βάρη

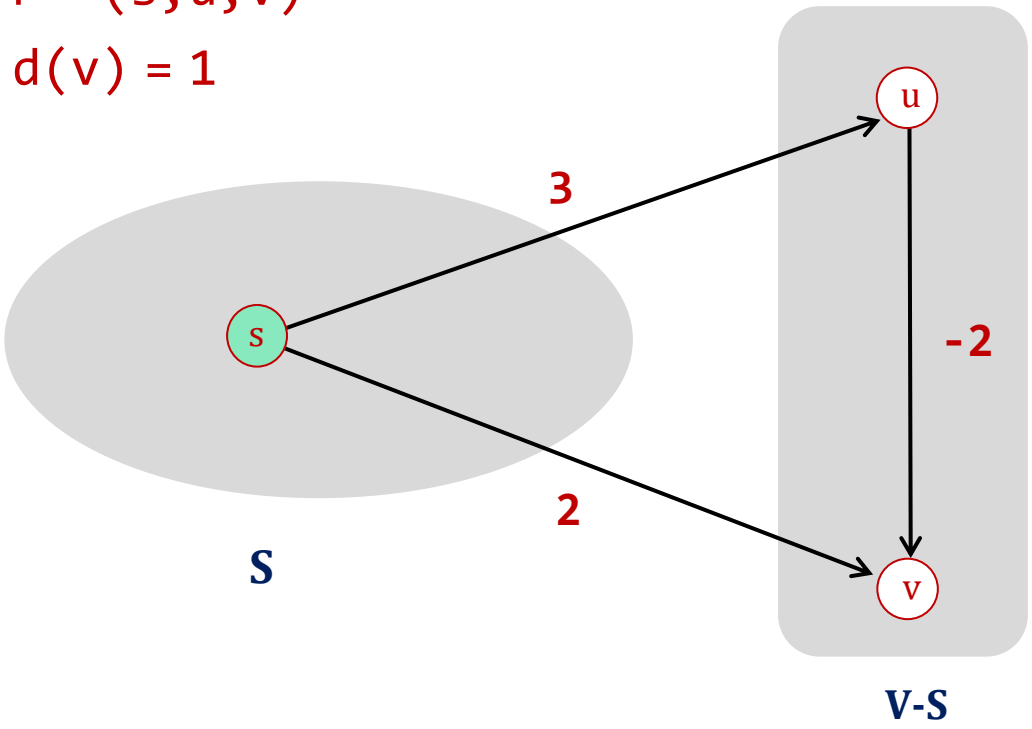
1. Αρνητικά Βάρη !!!

Αντιπαράδειγμα

Ελάχιστη διαδρομή P από τον s στον v:

P = (s,u,v)

d(v) = 1



Βήμα 1

Αρχικοποίηση: S={s} και d(s)=0

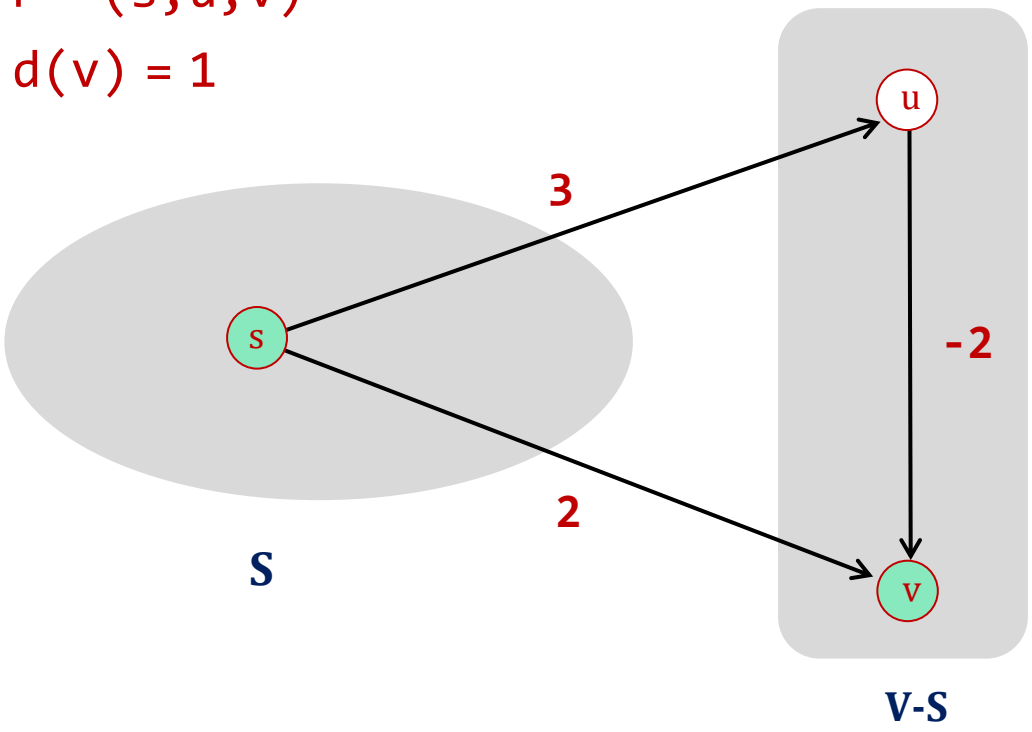
1. Αρνητικά Βάρη !!!

Αντιπαράδειγμα

Ελάχιστη διαδρομή P από τον s στον v:

P = (s,u,v)

d(v) = 1



Βήμα 1

Αρχικοποίηση: S={s} και d(s)=0

Βήμα 2

Επιλέγει τον κόμβο v∈V-S, διότι

d'(u) = min{0+3} = 3

d'(v) = min{0+2} = 2

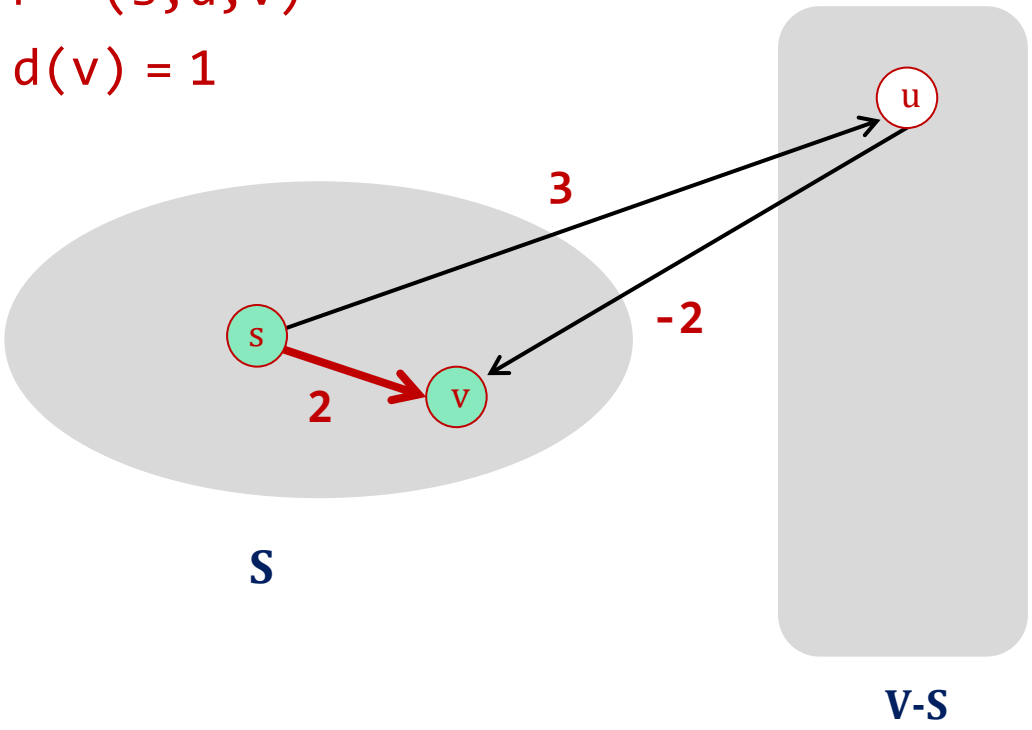
1. Αρνητικά Βάρη !!!

Αντιπαράδειγμα

Ελάχιστη διαδρομή P από τον s στον v:

P = (s,u,v)

d(v) = 1



Βήμα 1

Αρχικοποίηση: S={s} και d(s)=0

Βήμα 2

Επιλέγει τον κόμβο v∈V-S, διότι

d'(u) = min{0+3} = 3

d'(v) = min{0+2} = 2

και τον προσθέτει στο S θέτοντας d(v) = 2 !!!

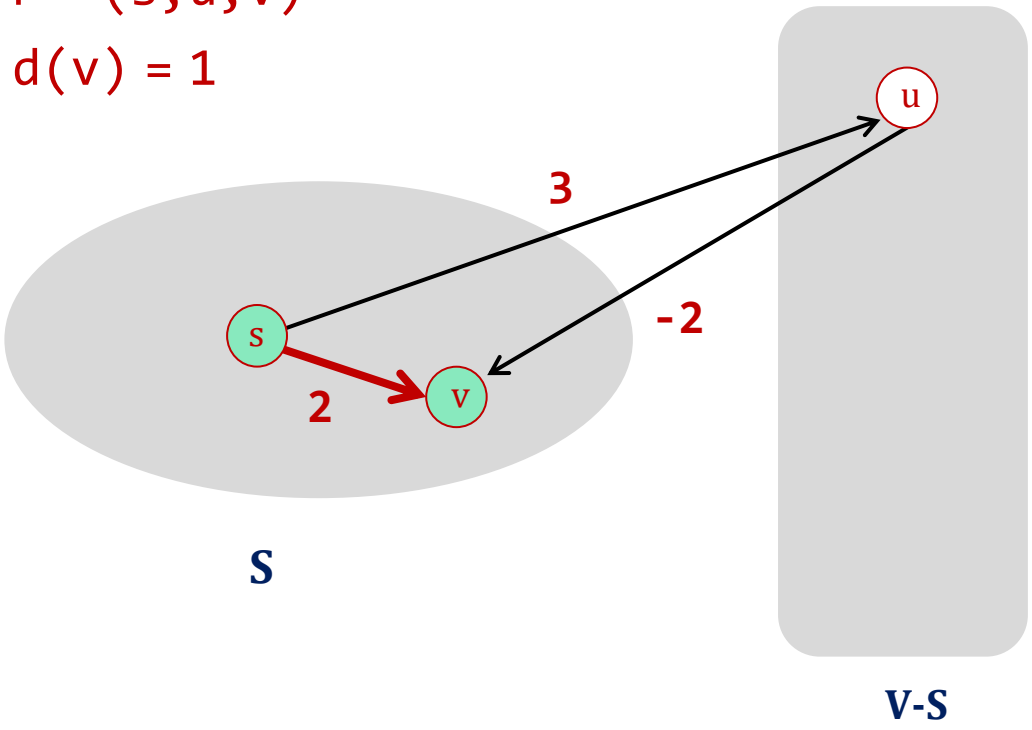
1. Αρνητικά Βάρη !!!

Αντιπαράδειγμα

Ελάχιστη διαδρομή P από τον s στον v:

P = (s,u,v)

d(v) = 1



Βήμα 1

Αρχικοποίηση: S={s} και d(s)=0

Βήμα 2

Επιλέγει τον κόμβο v∈V-S, διότι

d'(u) = min{0+3} = 3

d'(v) = min{0+2} = 2

και τον προσθέτει στο S θέτοντας d(v) = 2 !!!

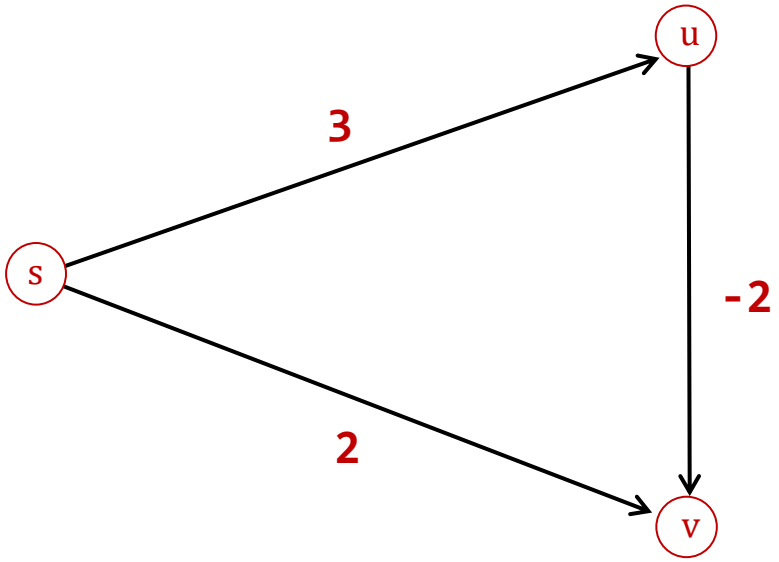
Λάθος, διότι d(v) = 1 και από την στιγμή που ο v προστίθεται στο S το d(v) δεν αλλάζει !!!

1. Αρνητικά Βάρη !!!

Ελάχιστη διαδρομή P από τον s στον v:

P = (s,u,v)

d(v) = 1



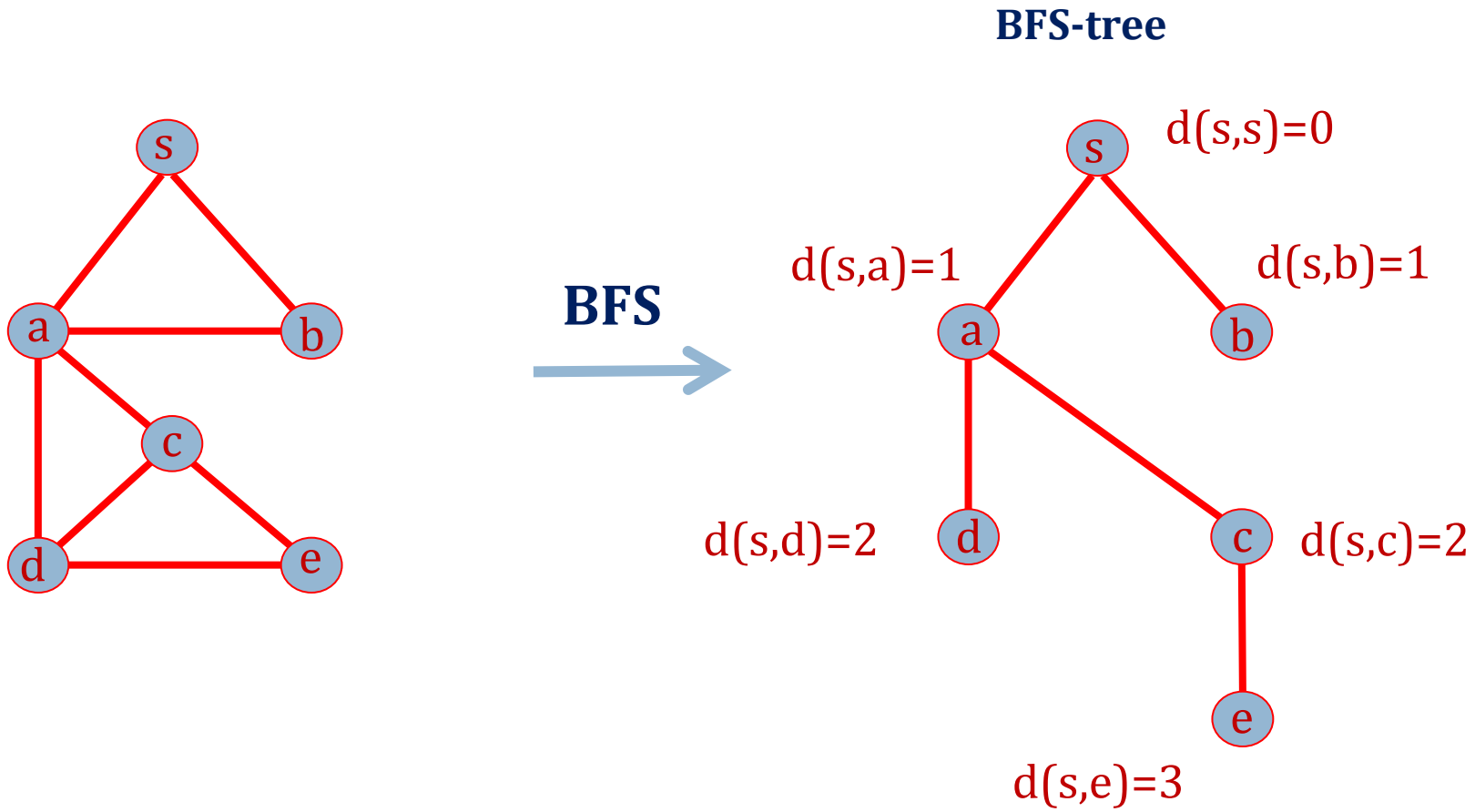
Σε αυτή την περίπτωση προσφεύγουμε στον Αλγόριθμο των

Bellman-Ford

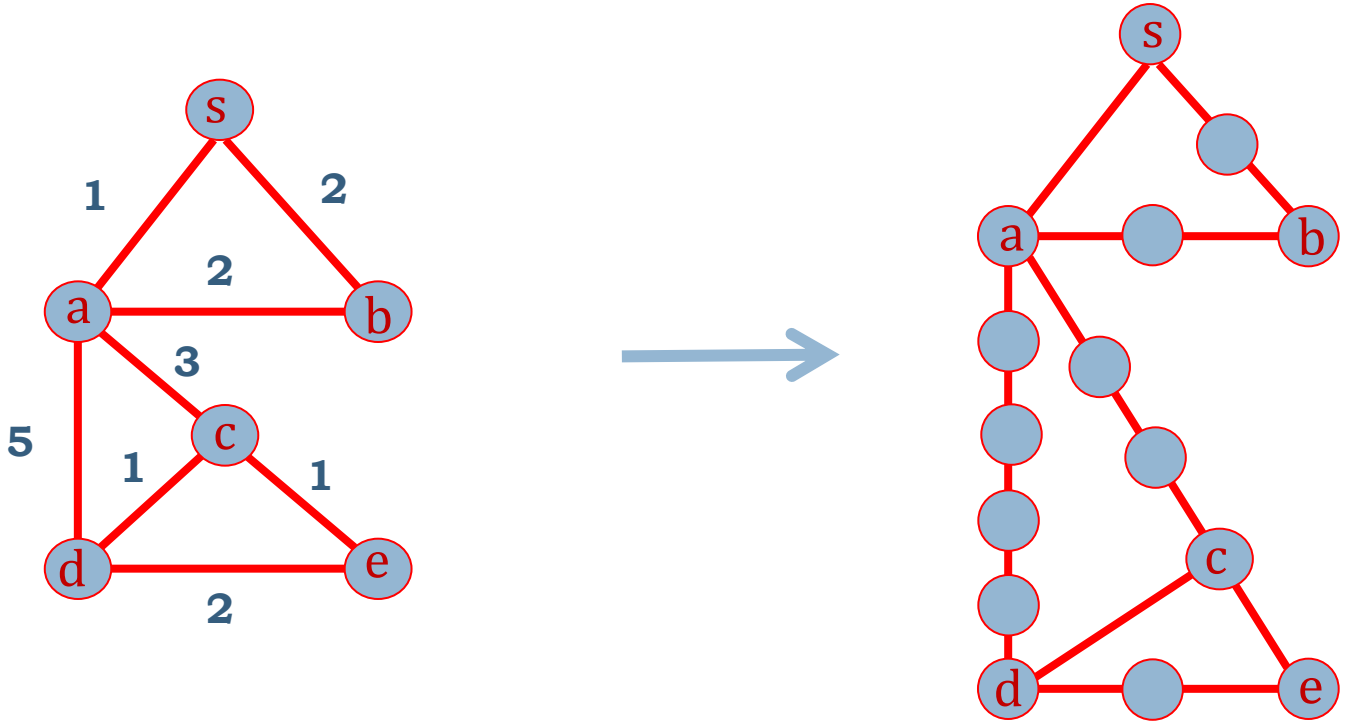
Επιτρέπει **αρνητικά βάρη** ακμών

Εντοπίζει κύκλους αρνητικού βάρους (κόστους)

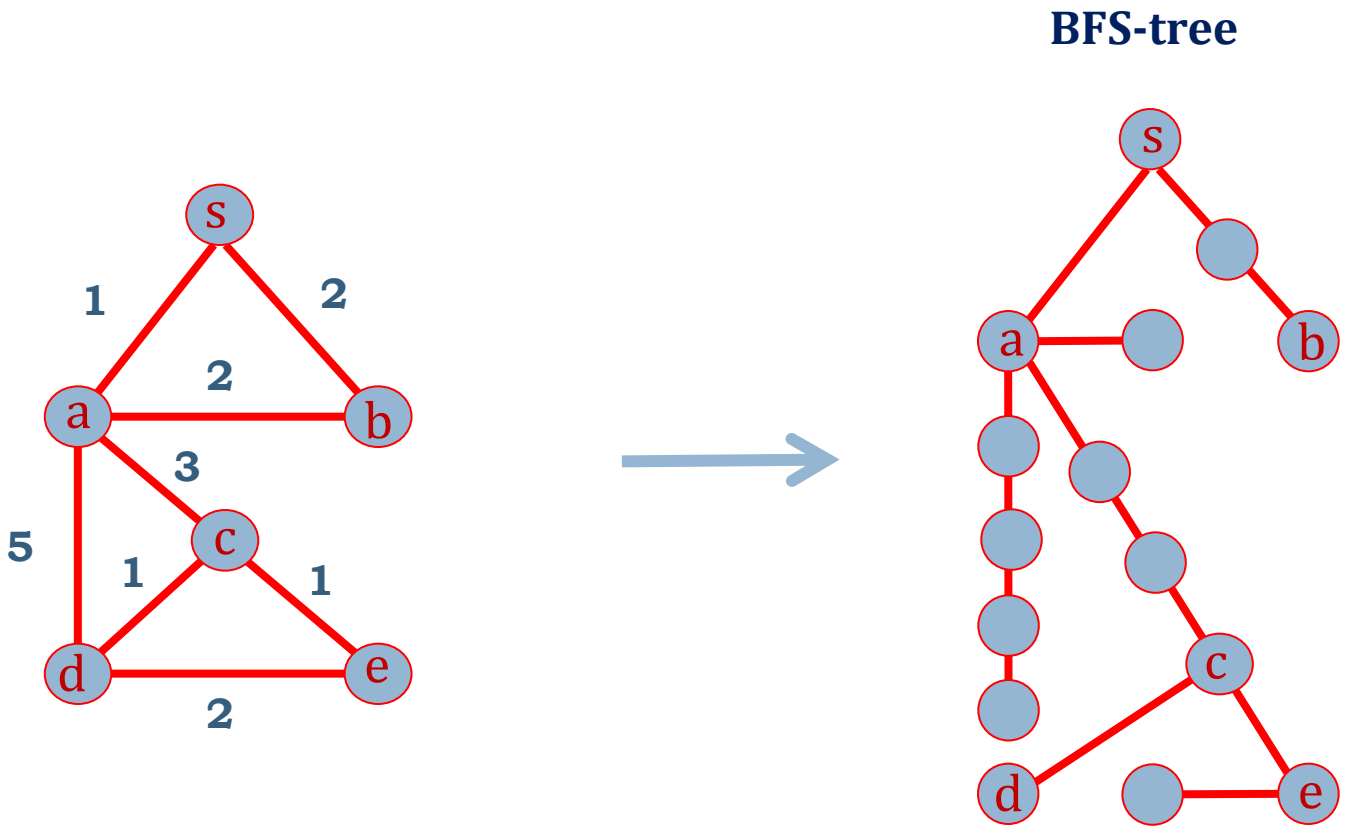
- 2. Αλγοριθμικές Σκέψεις !!!



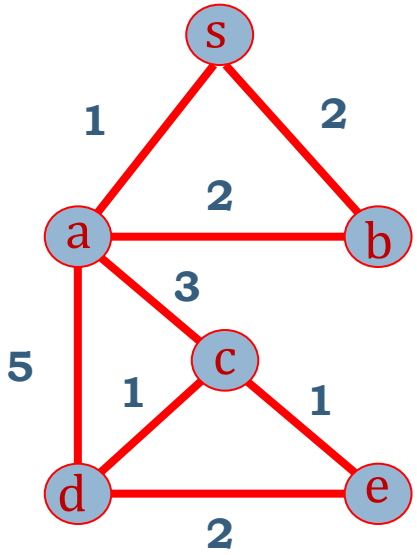
- 2. Αλγοριθμικές Σκέψεις !!!

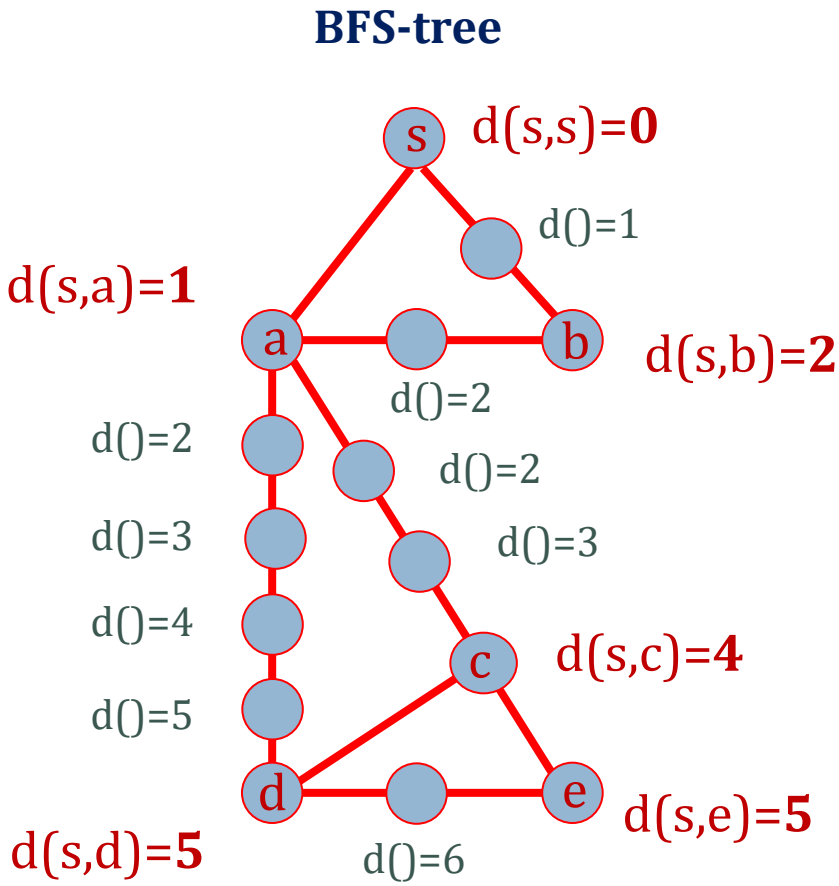


- 2. Αλγοριθμικές Σκέψεις !!!

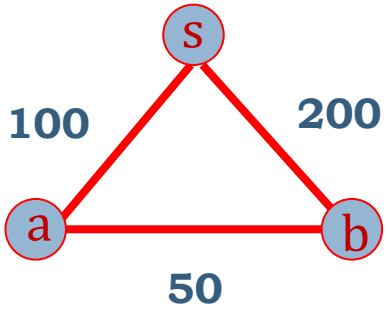


- 2. Αλγοριθμικές Σκέψεις !!!

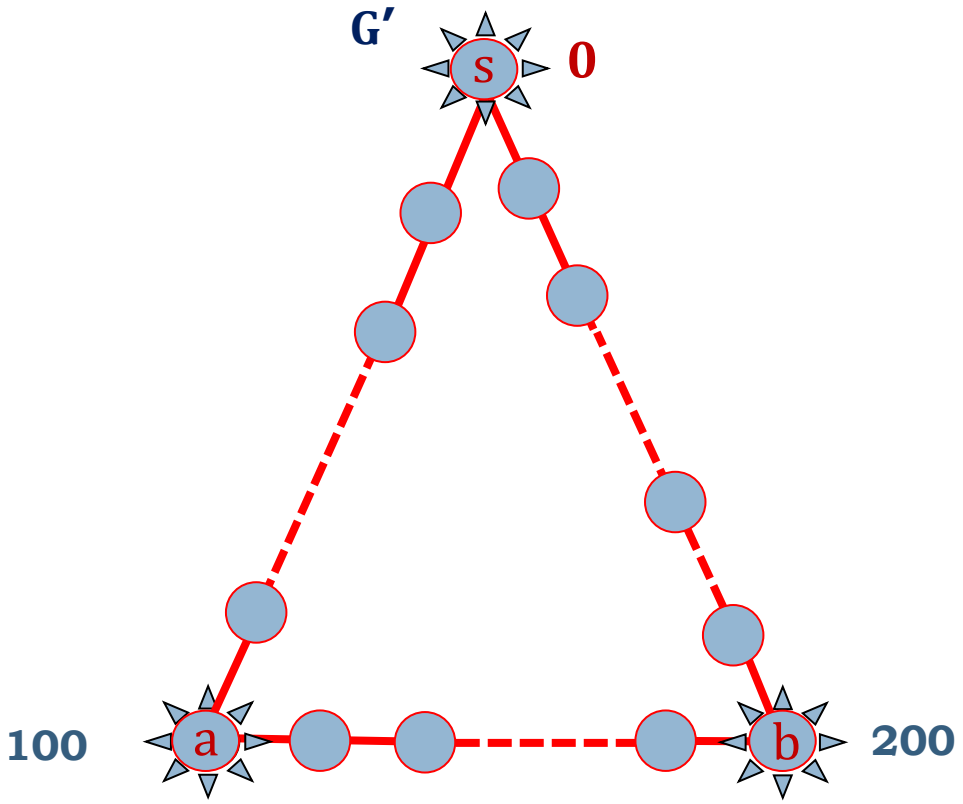




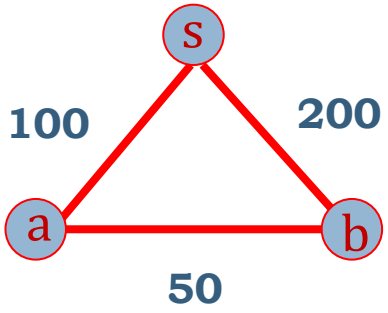
- 2. Αλγοριθμικές Σκέψεις !!!



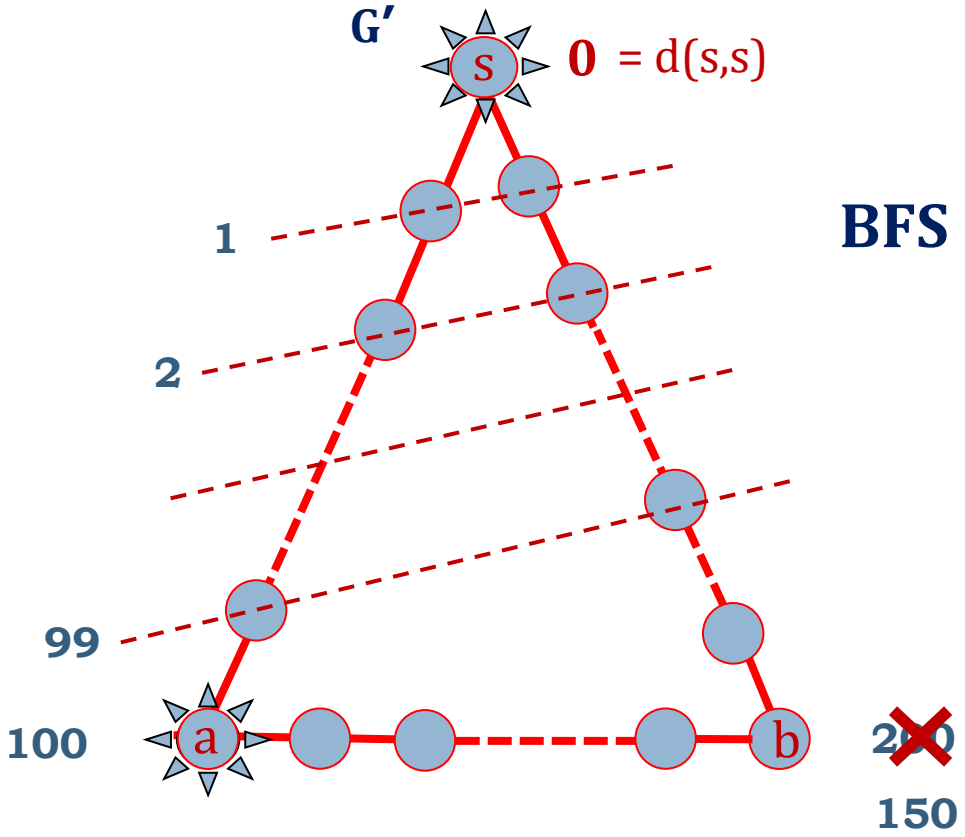
Έμβαρο γράφημα **G**



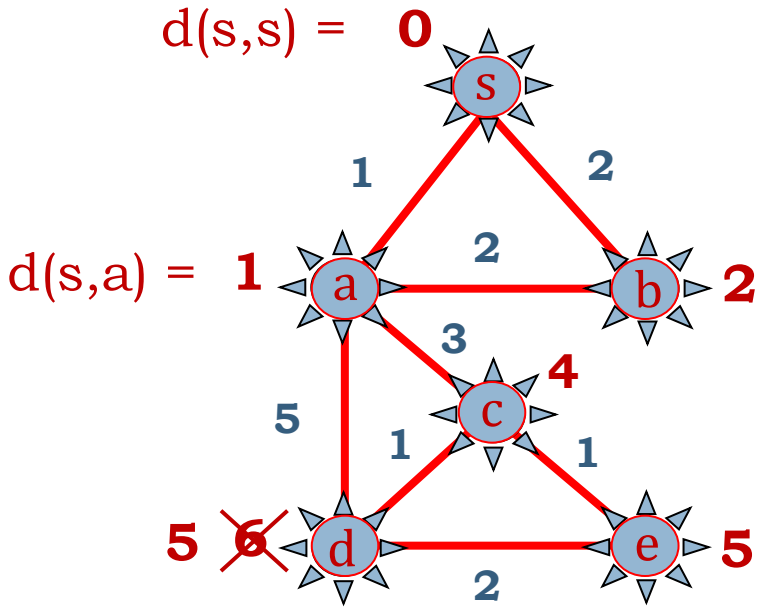
2. Αλγοριθμικές Σκέψεις !!!



Έμβαρο γράφημα **G**



Αλγόριθμος «Αφύπνισης»



Ρύθμισε ένα ρολόι για τον κόμβο **s** να χτυπήσει την χρονική στιγμή **0**

Επανάλαβε έως ότου χτυπήσουν όλα τα ρολόγια:

Έστω ότι το επόμενο ρολόι χτυπά την χρονική στιγμή **t** για τον κόμβο **u**, τότε:

Η απόσταση του **u** από τον **s** είναι **d(s,u) = t**

Για κάθε γείτονα **x** του **u** στο G:

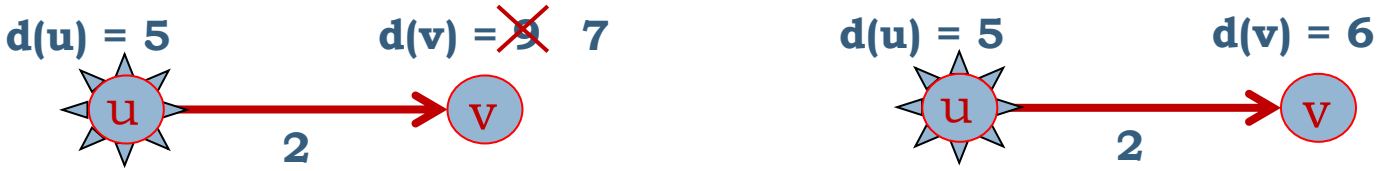
Ρύθμισε την αφύπνιση του **x** ίση με **t + w(u,x)** εάν δεν ∃ αφύπνιση για τον **x** ή ∃ κάποια τη χρονική στιγμή **t' > t + w(u,x)**, άλλως μη κάνεις τίποτε!

Ελάχιστες Διαδρομές – Διαδικασία Update()

- Αλγοριθμική Ρύθμιση της «Αφύπνισης»

d(s, v) = d(v) **Εκτιμητής Ελάχιστη Απόσταση**

```
Update(u,v,w):
        if d(v) > d(u) + w(u,v) then
                d(v) ← d(u) + w(u,v)
                prev(v) ← u
```



Χρήση της διαδικασίας `Update(u,v,w)` !!!

Θα παρουσιάσουμε τον αλγόριθμο του Dijkstra και τον αλγόριθμο των Bellman-Ford για τον υπολογισμό των ελάχιστων διαδρομών από ένα κόμβο **s** σε ένα έμβαρο γράφημα, βασιζόμενοι στην διαδικασία `Update(u,v,w)`

1. **Αλγόριθμος Dijkstra**
2. **Αλγόριθμος Bellman-Ford**

Χαρακτηριστικά των Δύο Αλγορίθμων !!!

1. Αλγόριθμος Dijkstra

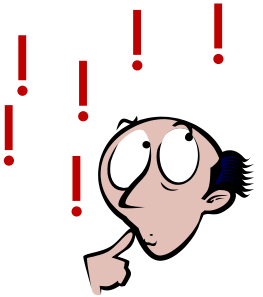
Επιτρέπει μόνο **μη-αρνητικά βάρη** ακμών.

2. Αλγόριθμος Bellman-Ford

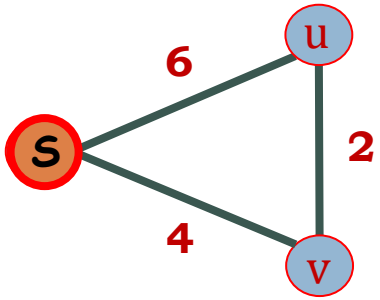
Επιτρέπει **αρνητικά βάρη** ακμών.

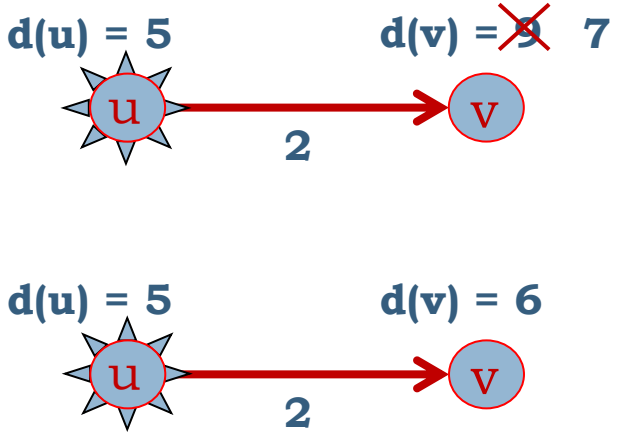
Εντοπίζει κύκλους αρνητικού βάρους (κόστους).

1. Αλγόριθμος Dijkstra



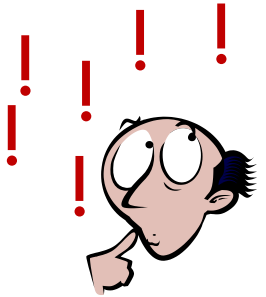
Επέκταση Βασικής Ιδέας !!!



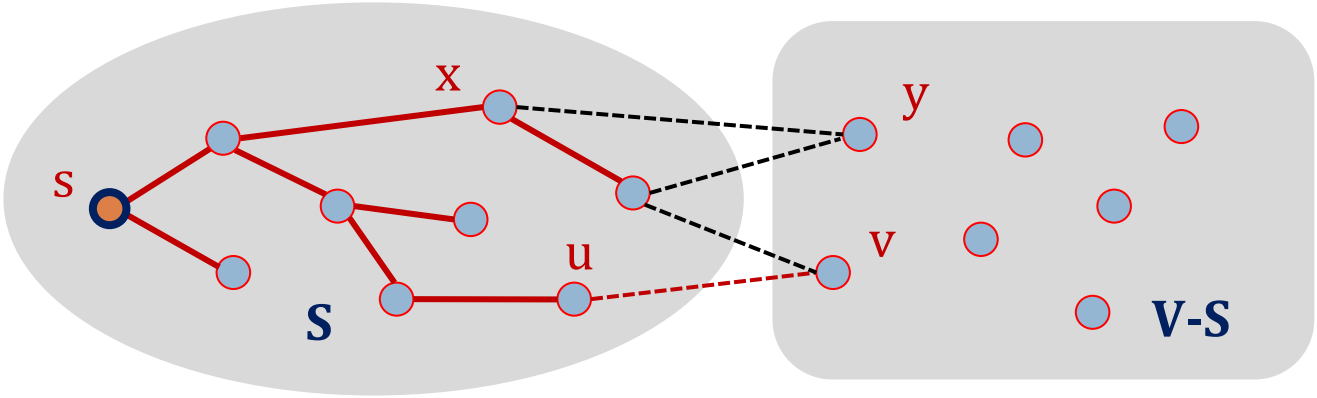


Μπορούμε να αποδώσουμε πιο απλά την ίδια ακριβώς λειτουργία του αλγορίθμου του Dijkstra που περιγράψαμε, χρησιμοποιώντας την διαδικασία `Update()` !

1. Αλγόριθμος Dijkstra



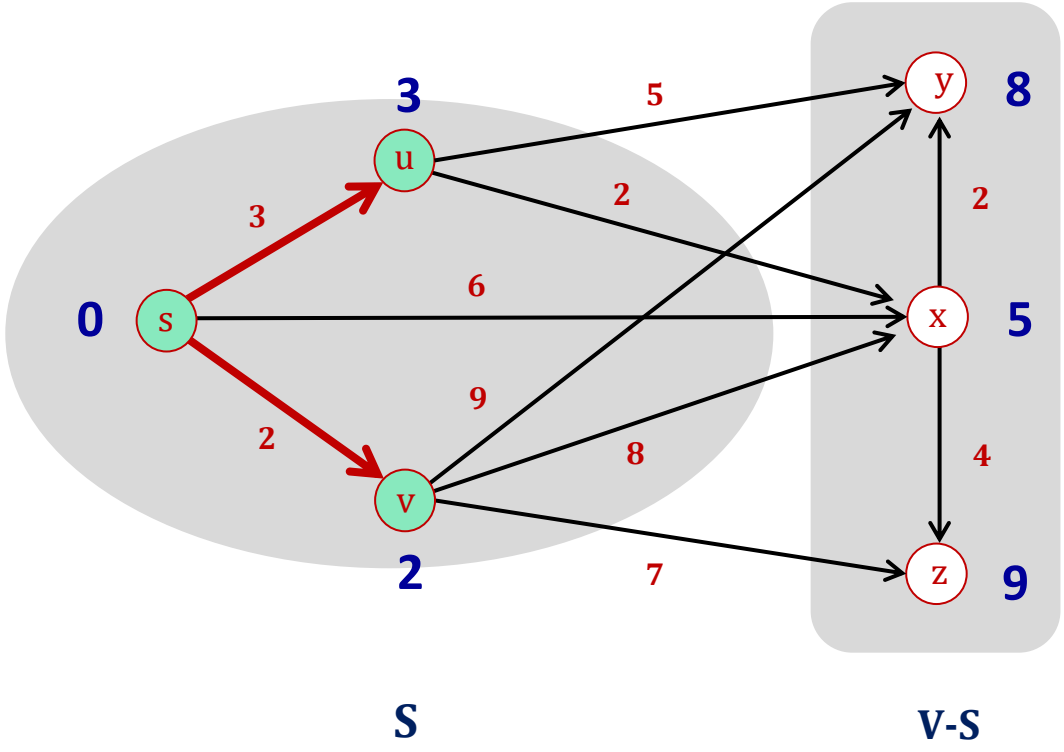
Επέκταση Βασικής Ιδέας !!!



Επιλέγει τον κόμβο `v` από το σύνολο `V-S` με τον `min` εκτιμητή `d(v)`, επεκτείνει το Δένδρο Ελαχίστων Διαδρομών προσθέτοντας τον `v` στο `S`), και ενημερώνει με `Update()` τους γείτονες του `v` στο σύνολο `V-S` !

Update() – Αλγόριθμος Dijkstra

Ενδιαφέρουσες Παρατηρήσεις



Βήμα 4

$d'(v) = \min\{d(u) + w(u,v)\}$

$d'(y) = \min\{3+5, 2+9\} = 8$

$d'(x) = \min\{3+2, 0+6, 2+8\} = 5$

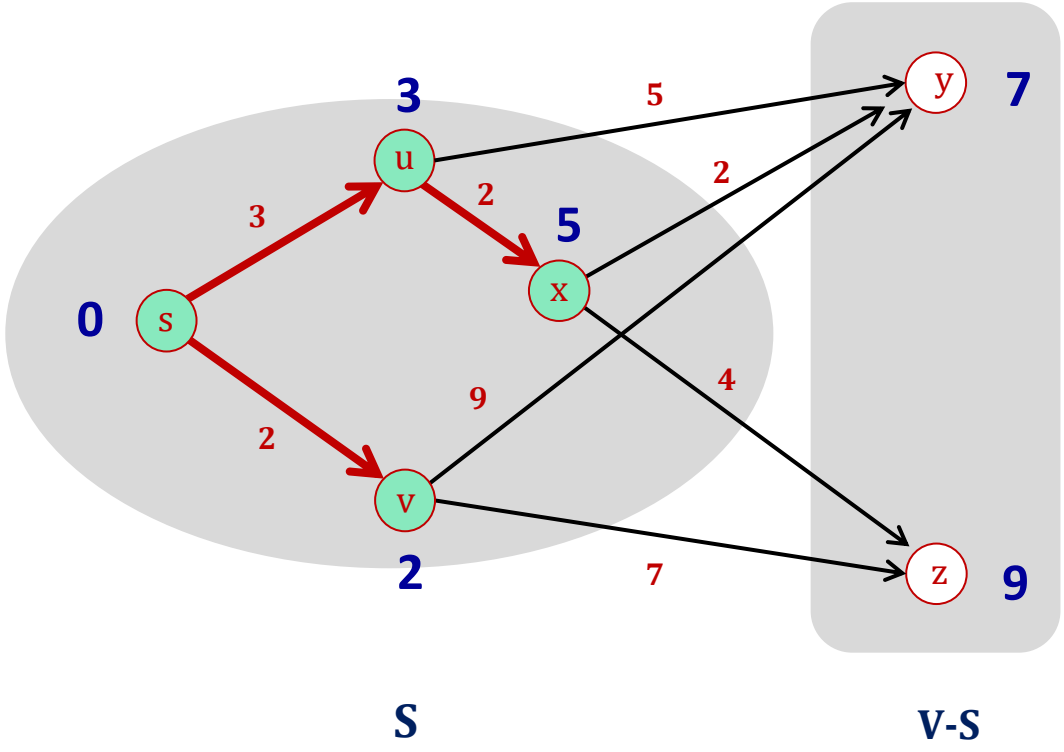
$d'(z) = \min\{2+7\} = 9$

Update(u,v,w)

```
if d(v) > d(u) + w(u,v) then
        d(v) ← d(u) + w(u,v)
```

Update() – Αλγόριθμος Dijkstra

Ενδιαφέρουσες Παρατηρήσεις



Βήμα 5

$d'(v) = \min\{d(u) + w(u,v)\}$

$d'(y) = \min\{3+5, 5+2, 2+9\} = 7$

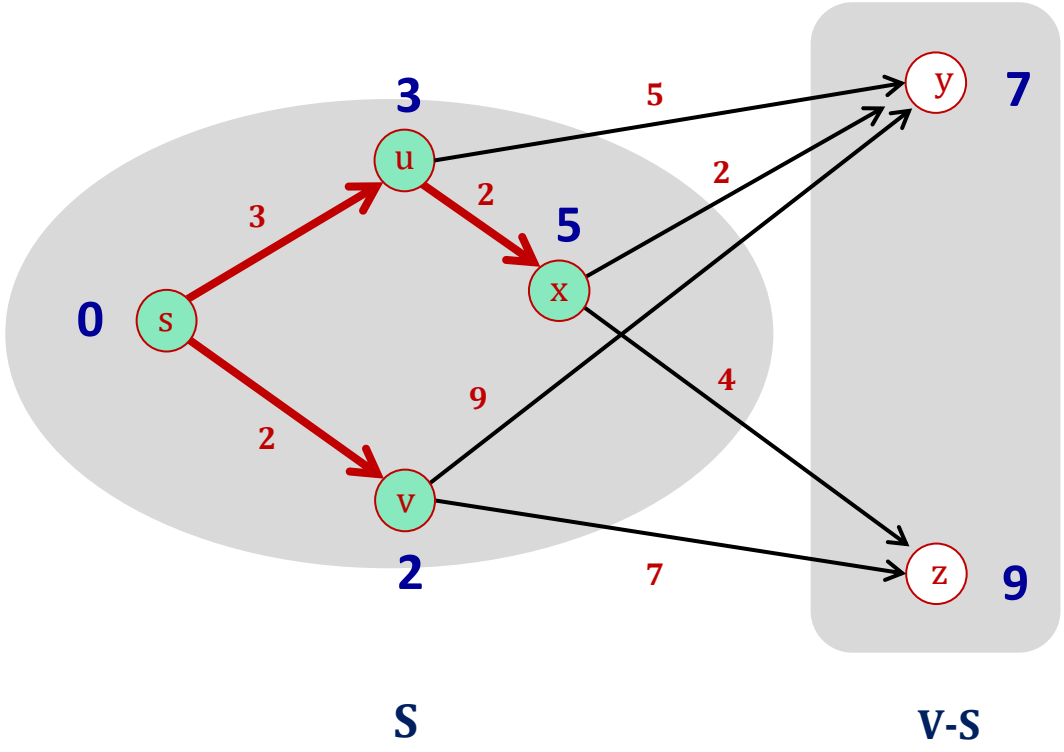
$d'(z) = \min\{5+4, 2+7\} = 9$

Update(u,v,w)

```
if d(v) > d(u) + w(u,v) then
        d(v) ← d(u) + w(u,v)
```

Update() – Αλγόριθμος Dijkstra

Ενδιαφέρουσες Παρατηρήσεις



Ισοδυναμία κριτηρίων:

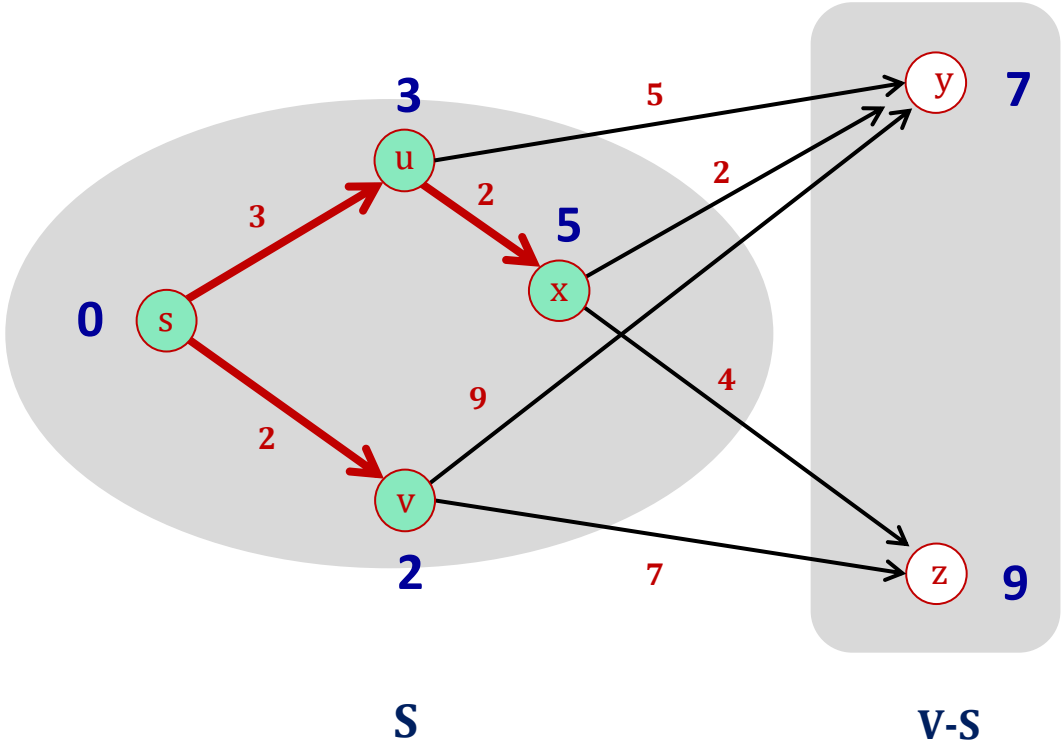
Το κόστος d'(v) της συντομότερης διαδρομής μεταξύ όλων των διαδρομών από τον s στον v της μορφής(s, …,u,v)

Ισούται

Με τον εκτιμητή ελάχιστης απόστασης d(v) του κόμβου v που υπολογίζεται με την συνάρτηση **Update(u,v,w)**.

Update() – Αλγόριθμος Dijkstra

Ενδιαφέρουσες Παρατηρήσεις



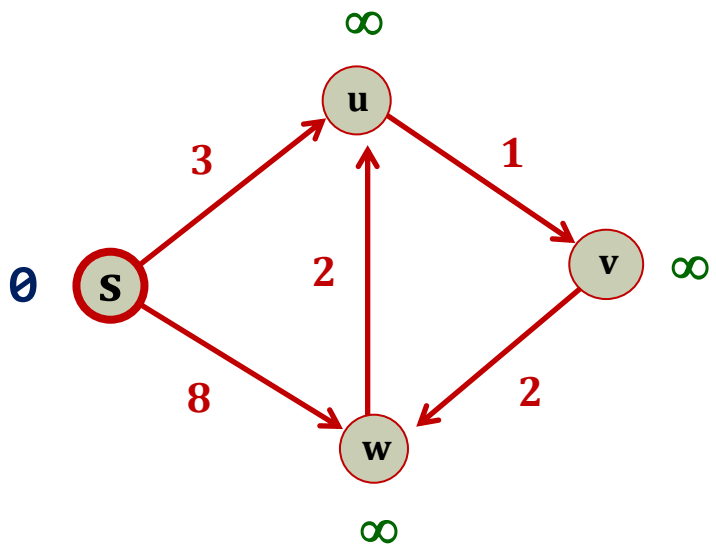
Κριτήριο άπληστης επιλογής:

Επίλεξε τον κόμβο `v` από το σύνολο `V-S` με τον `min` εκτιμητή `d(v)`, πρόσθεσέ τον στο `S`, και ενημερώνει με `Update()` τους γείτονες του `v` στο σύνολο `V-S` !

Update() – Αλγόριθμος Dijkstra

1. Αλγόριθμος Dijkstra

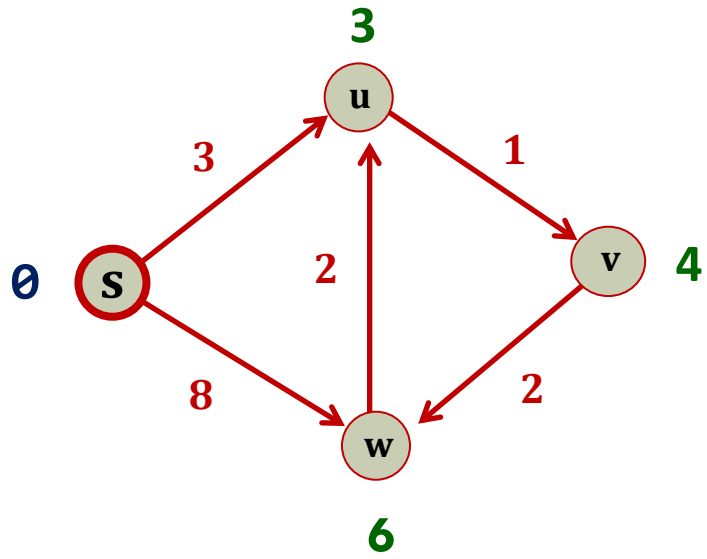
```
Initialize(G,s)
1.  για κάθε κόμβο v∈V:
        d(v) ← ∞
        prev(v) ← NIL
2.  d(s) ← 0
```



Update() – Αλγόριθμος Dijkstra

1. Αλγόριθμος Dijkstra

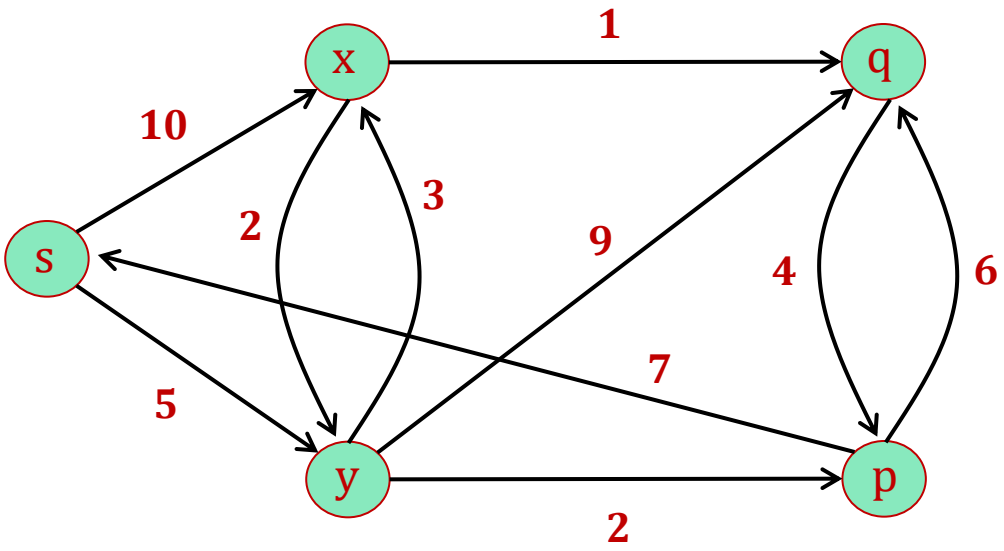
```
Dijkstra(G,w,s)
1.  Initialize(G,s)
2.  S ← ∅; Q ← V = V-S
3.  while Q ≠ ∅ :
        u ← extract-min(Q);
        S ← S ∪ {u}
        για κάθε ακμή (u,v)∈E:
                Update(u,v,w)   ⇒   if d(v) > d(u) + w(u,v) then
                                        d(v) ← d(u) + w(u,v)
                                        prev(v) ← u
4.  return d(), prev()
```



Update() – Αλγόριθμος Dijkstra

Παράδειγμα

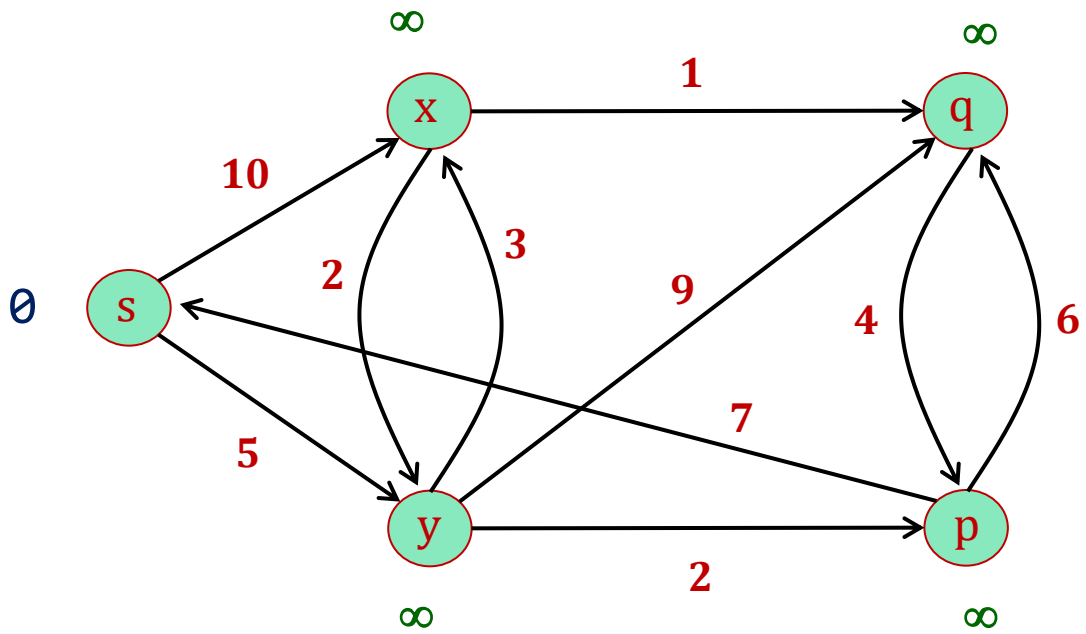
Είσοδος Αλγόριθμου



Update() – Αλγόριθμος Dijkstra

Παράδειγμα

```
Initialize(G,s)
    για κάθε κόμβο v∈V:
        d(v) ← ∞
        prev(v) ← NIL
    d(s) ← 0
```

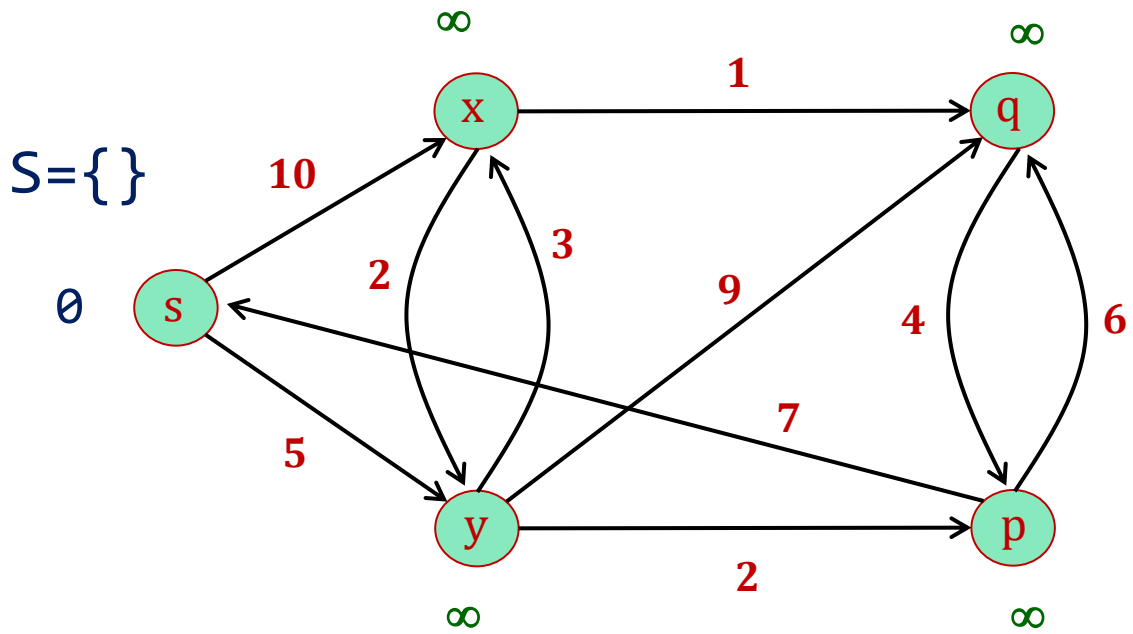


d(s)=0 prev(s)=NIL | d(x)=∞ prev(x)=Λ | d(y)=∞ prev(y)=Λ | d(p)=∞ prev(p)=Λ | d(q)=∞ prev(q)=Λ

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

Q = {s,x,y,p,q}



d(s)=0 d(x)=∞ d(y)=∞ d(p)=∞ d(q)=∞

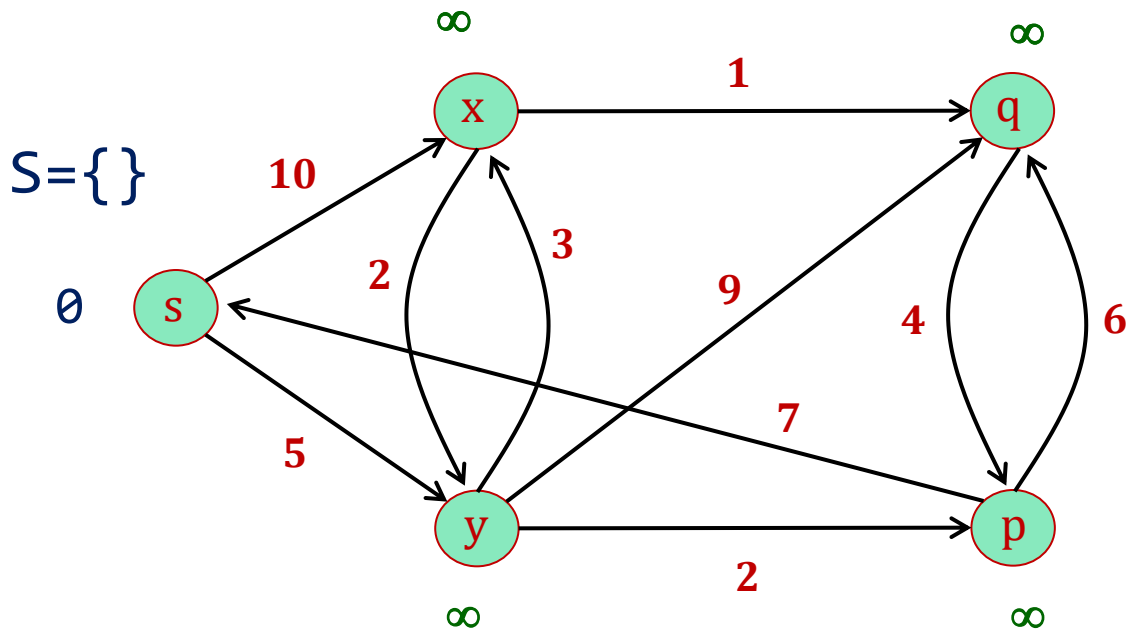
prev(s)=NIL prev(x)=Λ prev(y)=Λ prev(p)=Λ prev(q)=Λ

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

Q = {s,x,y,p,q}

while Q ≠ ∅ :



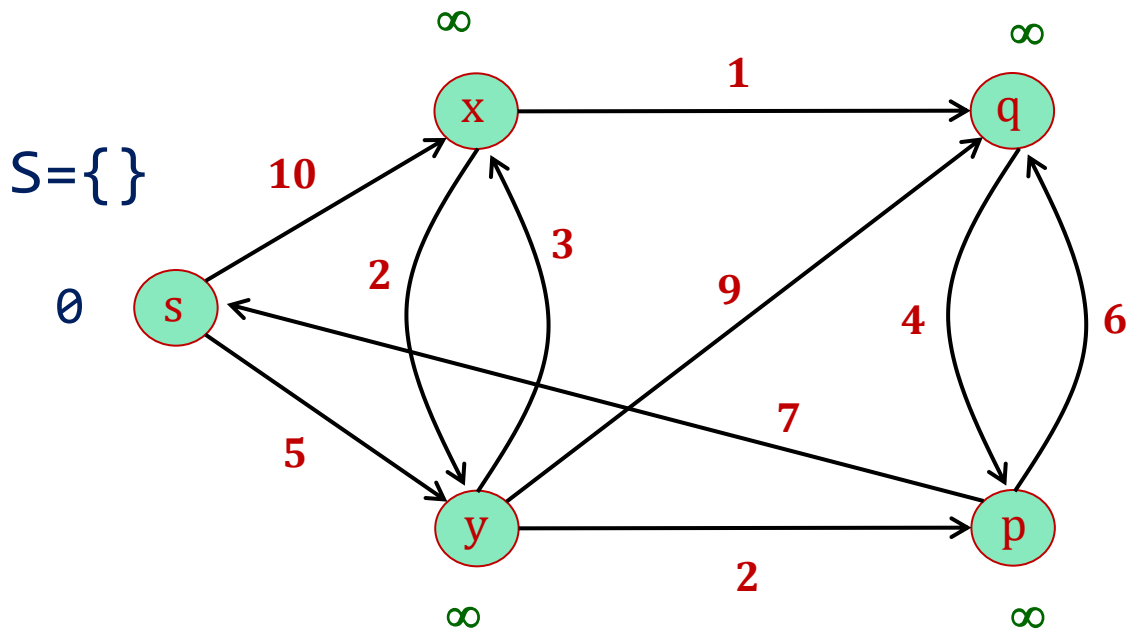
d(s)=0 d(x)=∞ d(y)=∞ d(p)=∞ d(q)=∞

prev(s)=NIL prev(x)=Λ prev(y)=Λ prev(p)=Λ prev(q)=Λ

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

```
Q = {s,x,y,p,q}
while Q ≠ ∅ :
    u ← extract-min(Q);
```



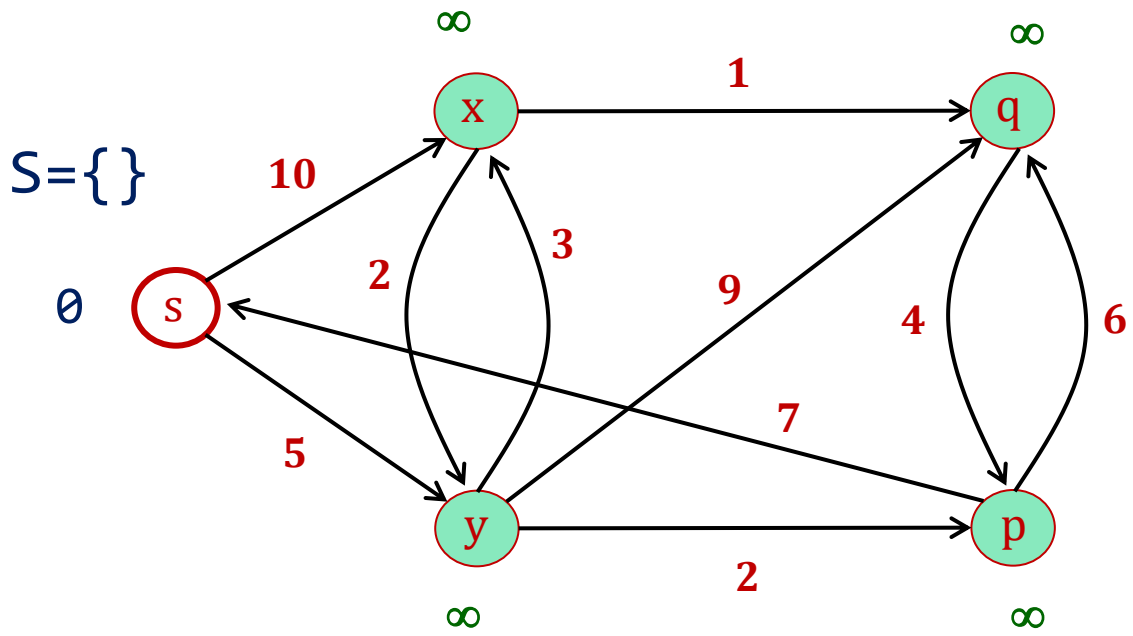
d(s)=0 d(x)=∞ d(y)=∞ d(p)=∞ d(q)=∞

prev(s)=NIL prev(x)=Λ prev(y)=Λ prev(p)=Λ prev(q)=Λ

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

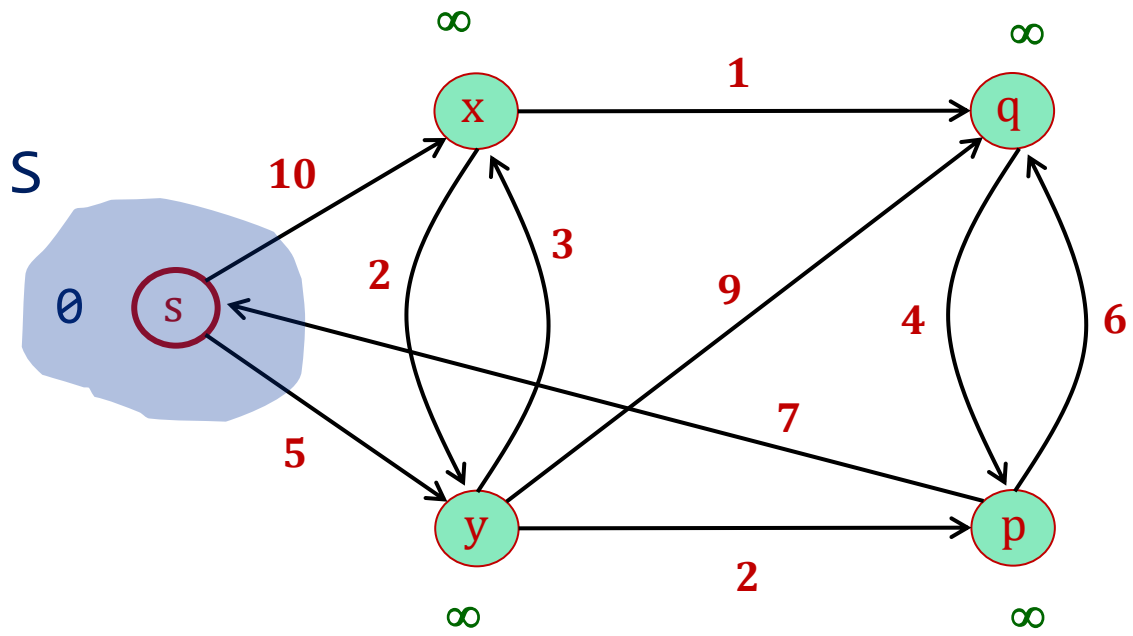
```
Q = {x,y,p,q}
while Q ≠ ∅ :
    u ← extract-min(Q);
```



d(s)=0
prev(s)=NIL

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

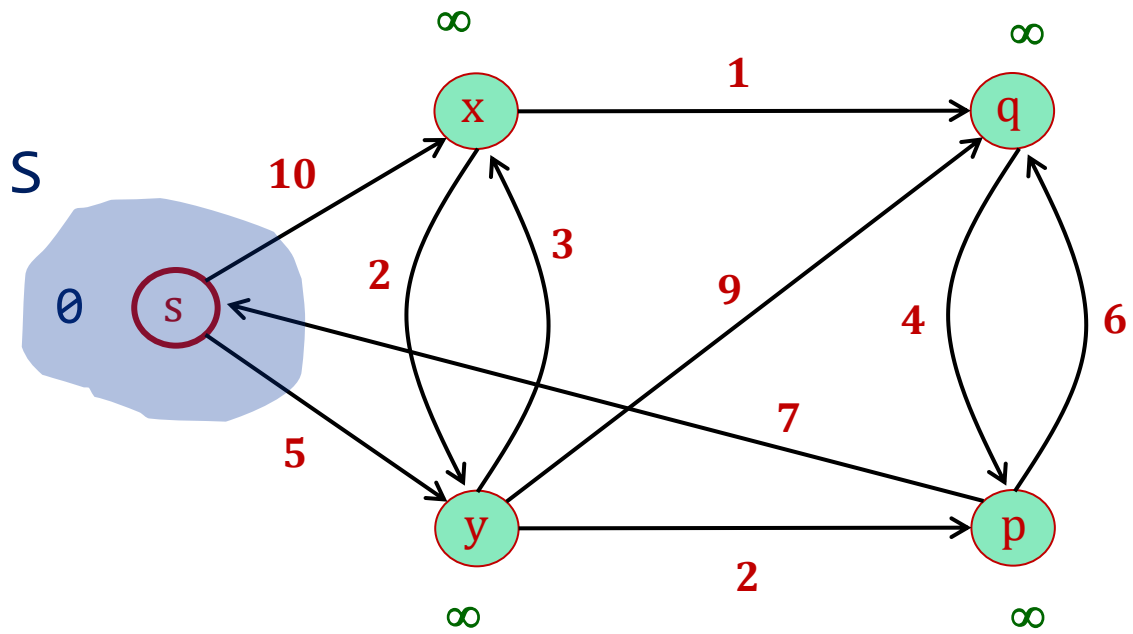



```
Q = {x,y,p,q}
while Q ≠ ∅ :
    u ← extract-min(Q);
    S ← S ∪ {u}
```

d(s)=0
prev(s)=NIL

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

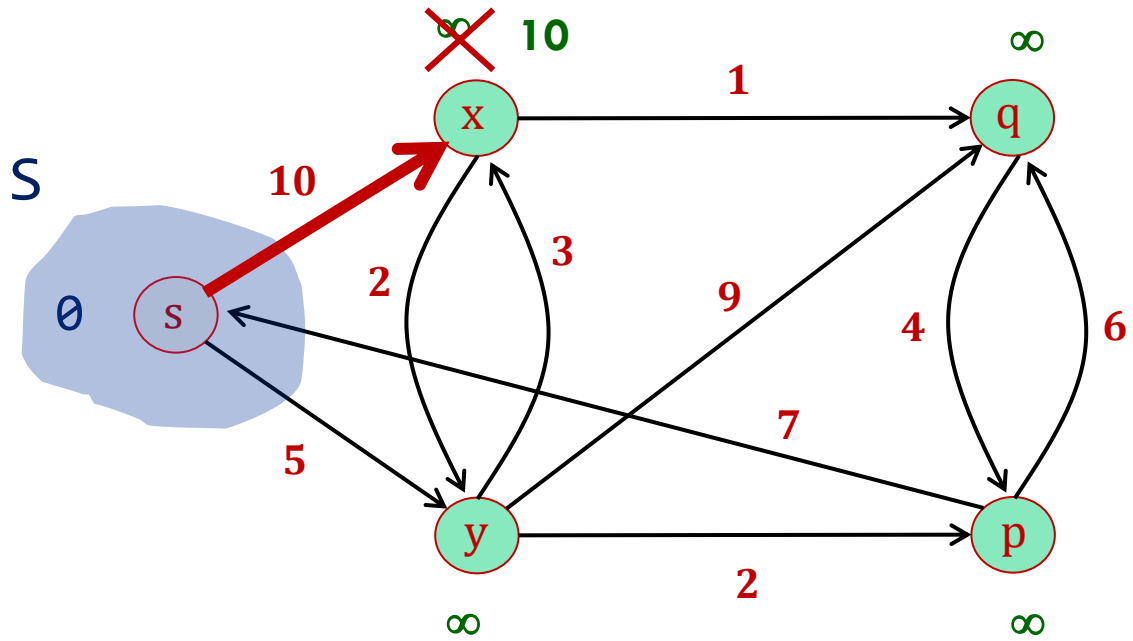



```
Q = {x,y,p,q}
while Q ≠ ∅ :
    u ← extract-min(Q);
    S ← S ∪ {u}
    Update(u,v,w)
```

d(s)=0
prev(s)=NIL

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

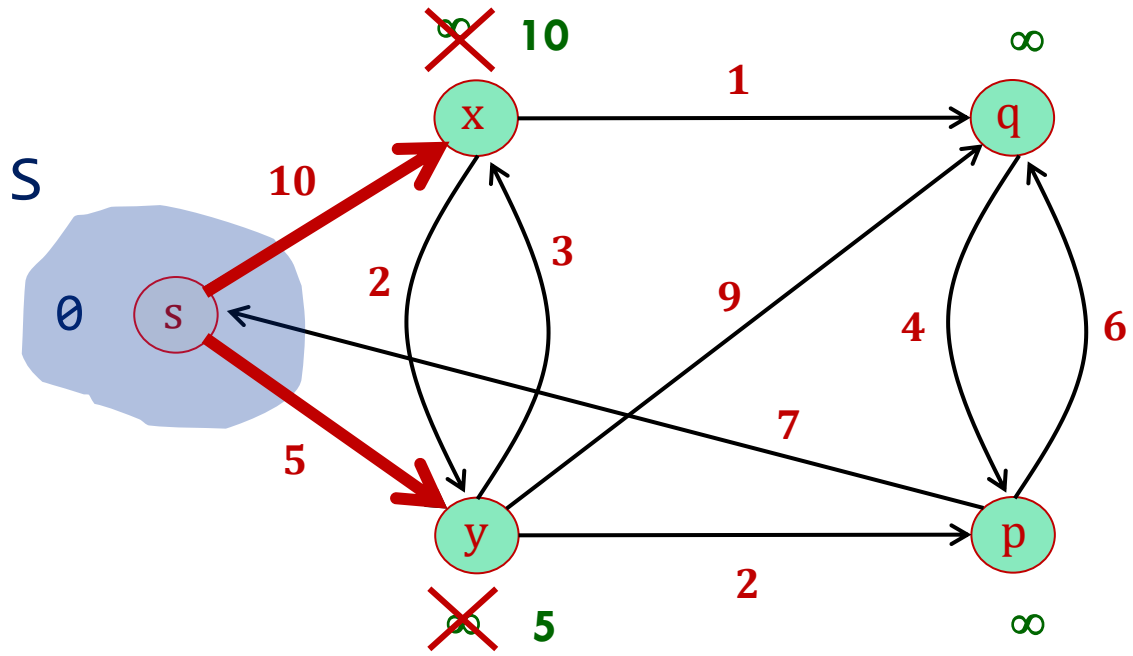



```
Q = {x,y,p,q}
while Q ≠ ∅ :
    u ← extract-min(Q);
    S ← S ∪ {u}
    Update(u,v,w)
```

d(s)=0
prev(s)=NIL

Update() – Αλγόριθμος Dijkstra

Παράδειγμα



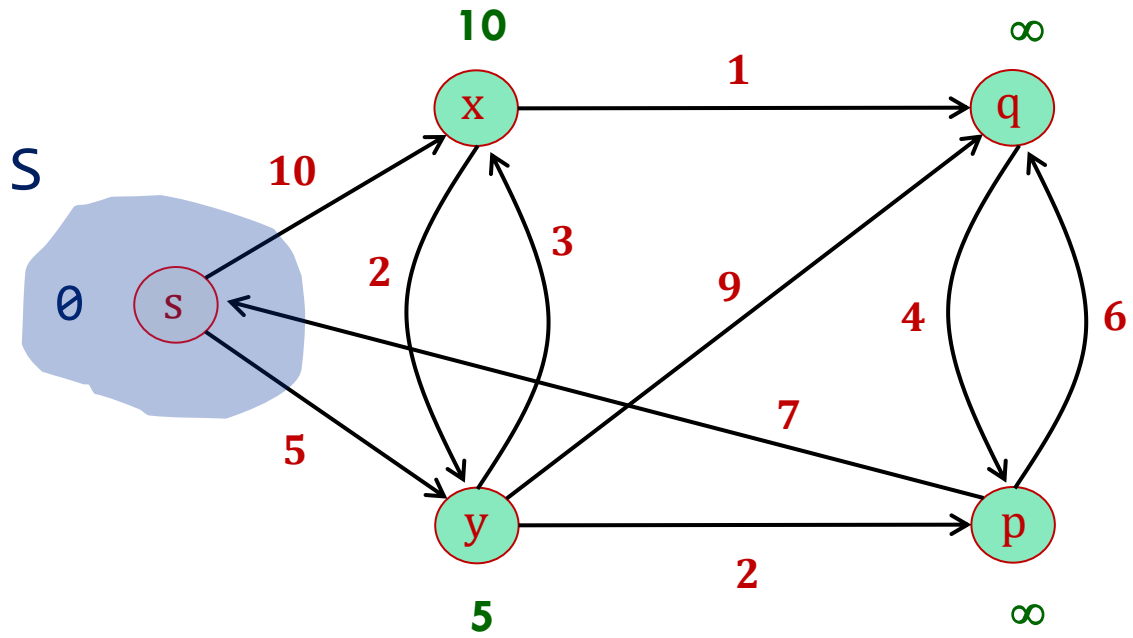

```
Q = {x,y,p,q}
while Q ≠ ∅ :
    u ← extract-min(Q);
    S ← S ∪ {u}
    Update(u,v,w)
```

d(s)=0
prev(s)=NIL

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

```
Q = {x,y,p,q}
while Q ≠ ∅ :
    u ← extract-min(Q);
```

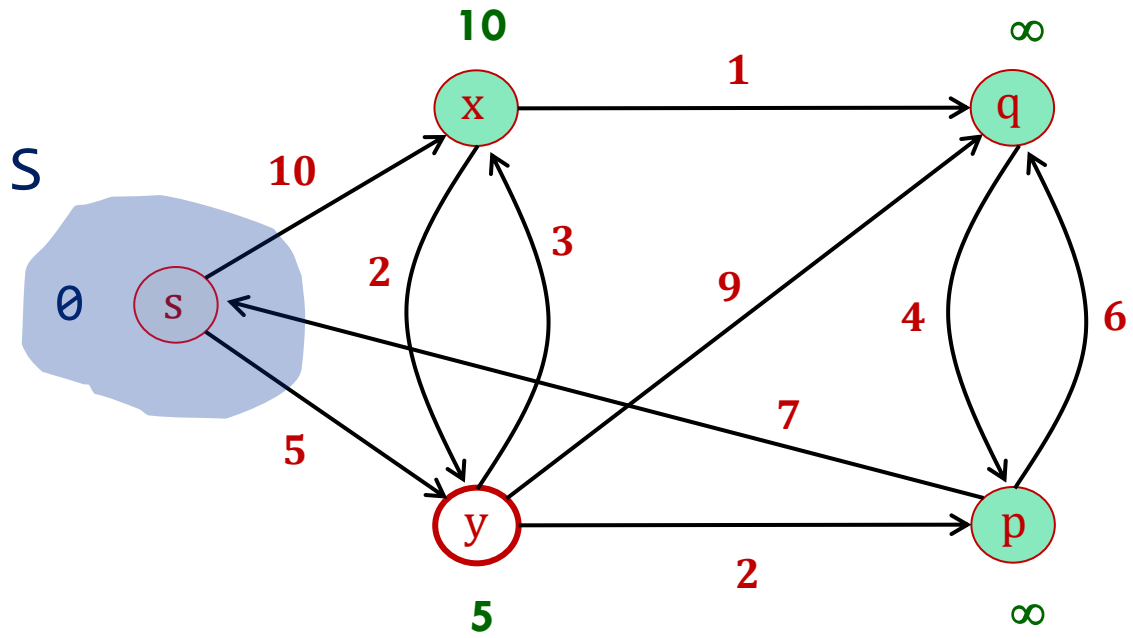


d(s)=0
prev(s)=NIL

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

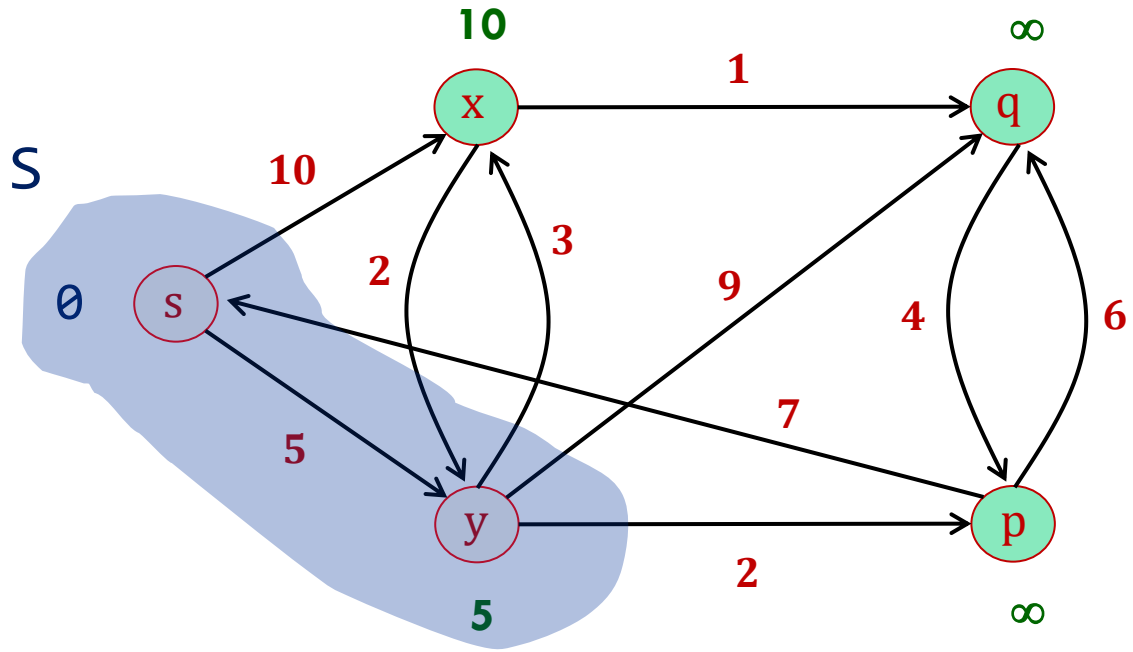
```
Q = {x,p,q}
while Q ≠ ∅ :
    u ← extract-min(Q);
```



d(s)=0 d(y)=5
prev(s)=NIL prev(y)=s

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

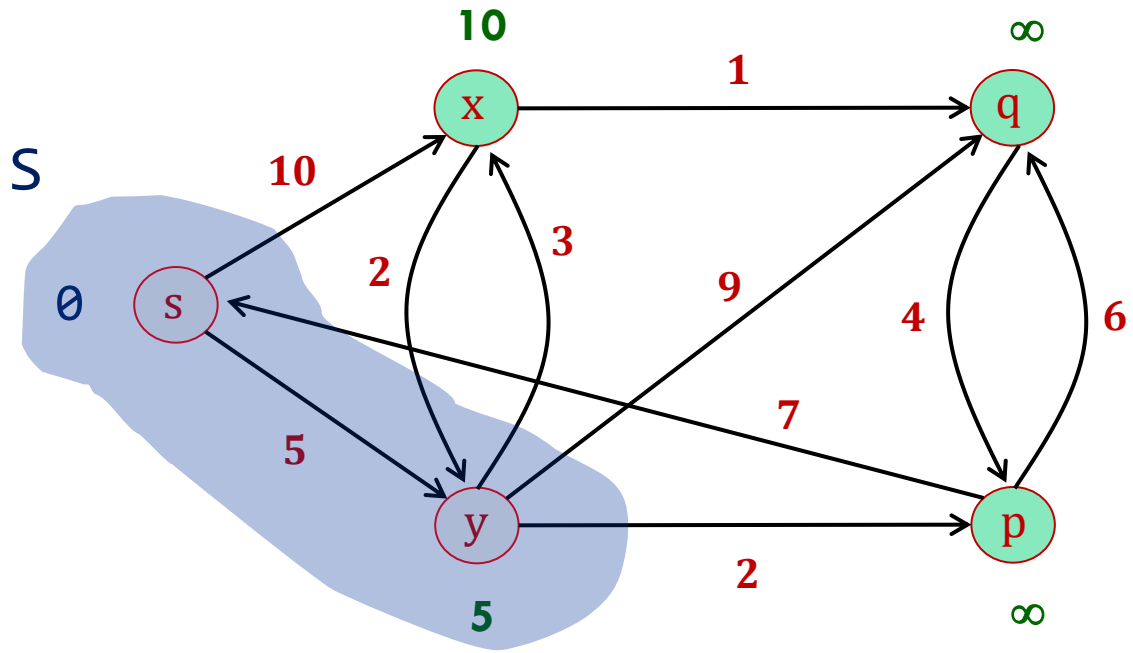



```
Q = {x,p,q}
while Q ≠ ∅ :
    u ← extract-min(Q);
    S ← S ∪ {u}
```

```
d(s)=0          d(y)=5
prev(s)=NIL     prev(y)=s
```

Update() – Αλγόριθμος Dijkstra

Παράδειγμα



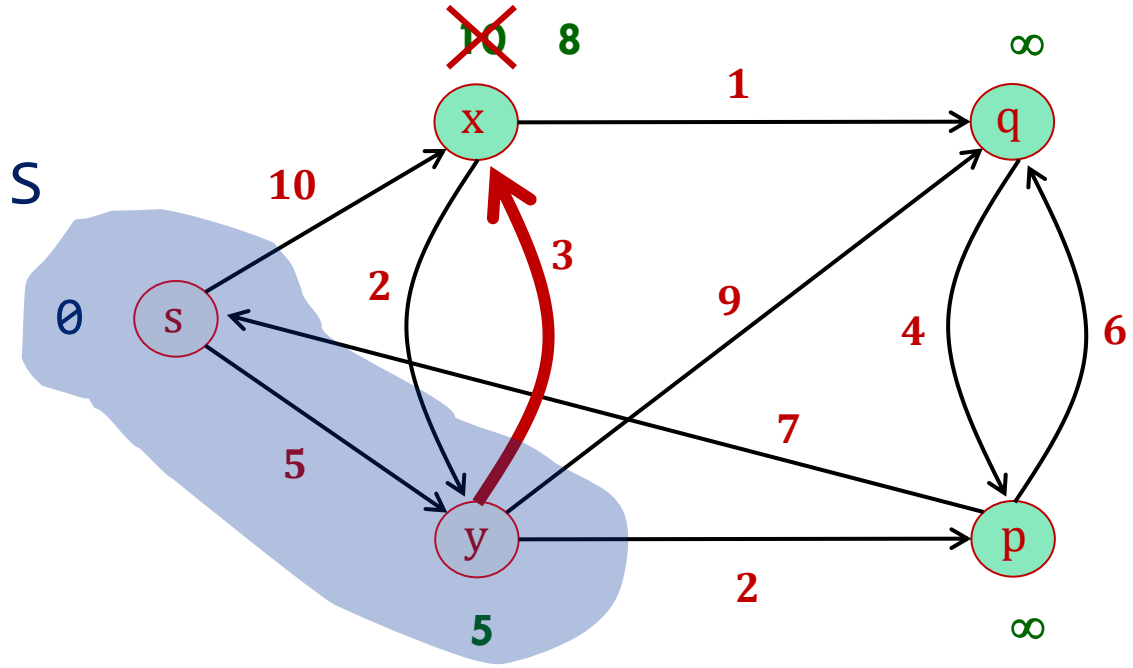

```
Q = {x,p,q}
while Q ≠ ∅ :
    u ← extract-min(Q);
    S ← S ∪ {u}
    Update(u,v,w)
```

d(s)=0 d(y)=5

prev(s)=NIL prev(y)=s

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

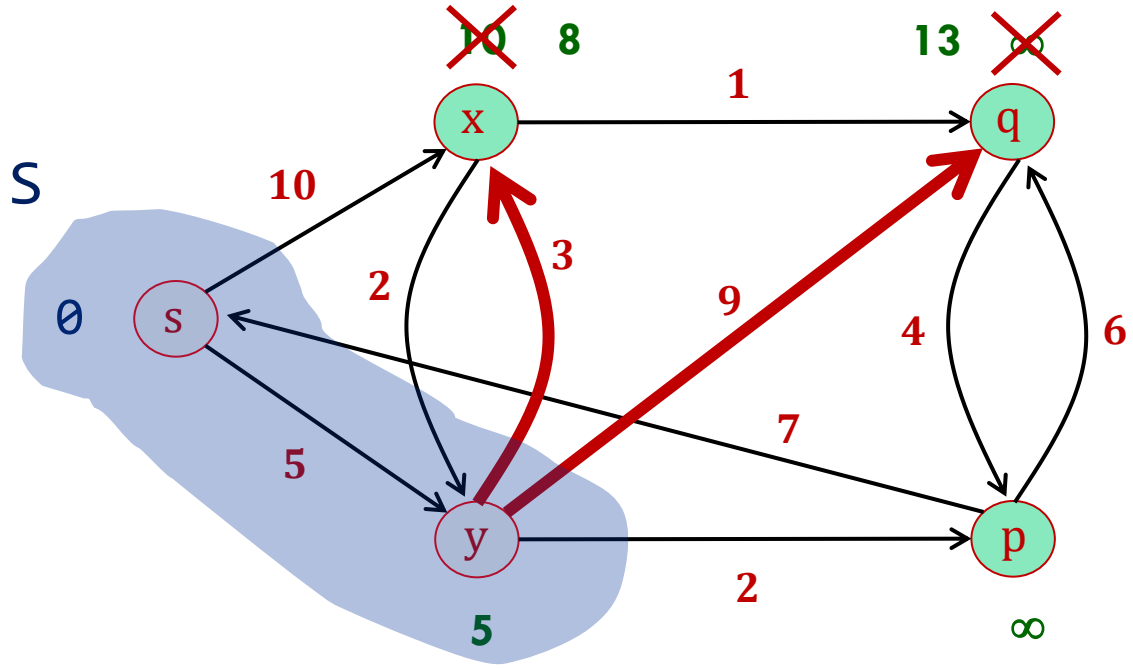



```
Q = {x,p,q}
while Q ≠ ∅ :
    u ← extract-min(Q);
    S ← S ∪ {u}
    Update(u,v,w)
```

d(s)=0 d(y)=5
prev(s)=NIL prev(y)=s

Update() – Αλγόριθμος Dijkstra

Παράδειγμα



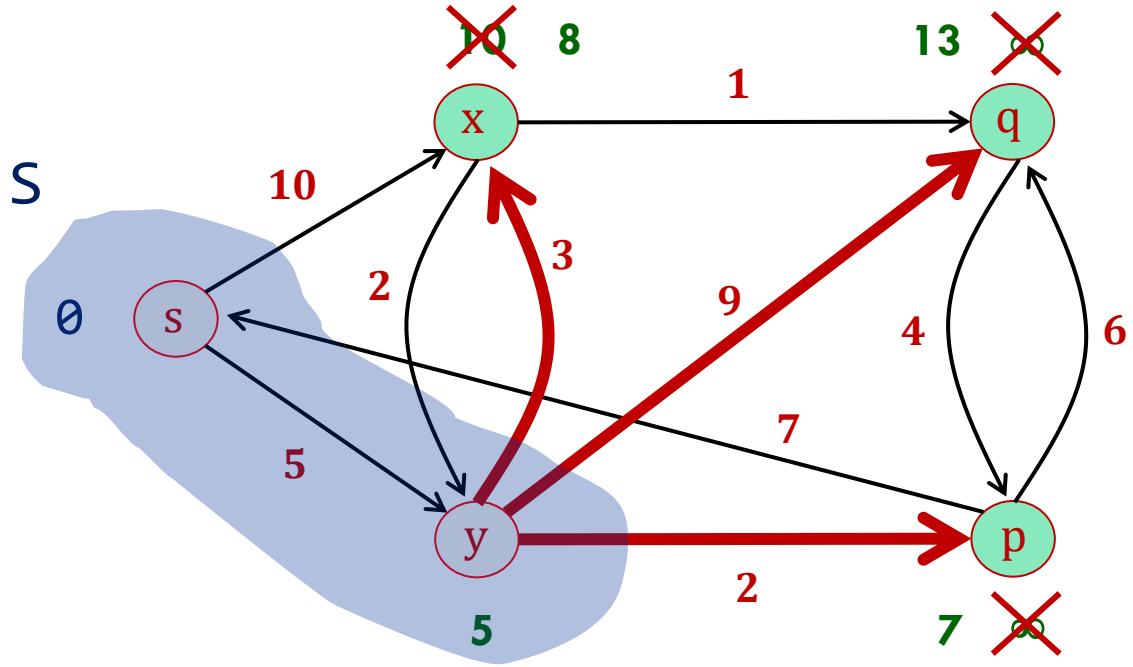

```
Q = {x,p,q}
while Q ≠ ∅ :
    u ← extract-min(Q);
    S ← S ∪ {u}
    Update(u,v,w)
```

d(s)=0 d(y)=5

prev(s)=NIL prev(y)=s

Update() – Αλγόριθμος Dijkstra

Παράδειγμα




```
Q = {x,p,q}
while Q ≠ ∅ :
    u ← extract-min(Q);
    S ← S ∪ {u}
    Update(u,v,w)
```

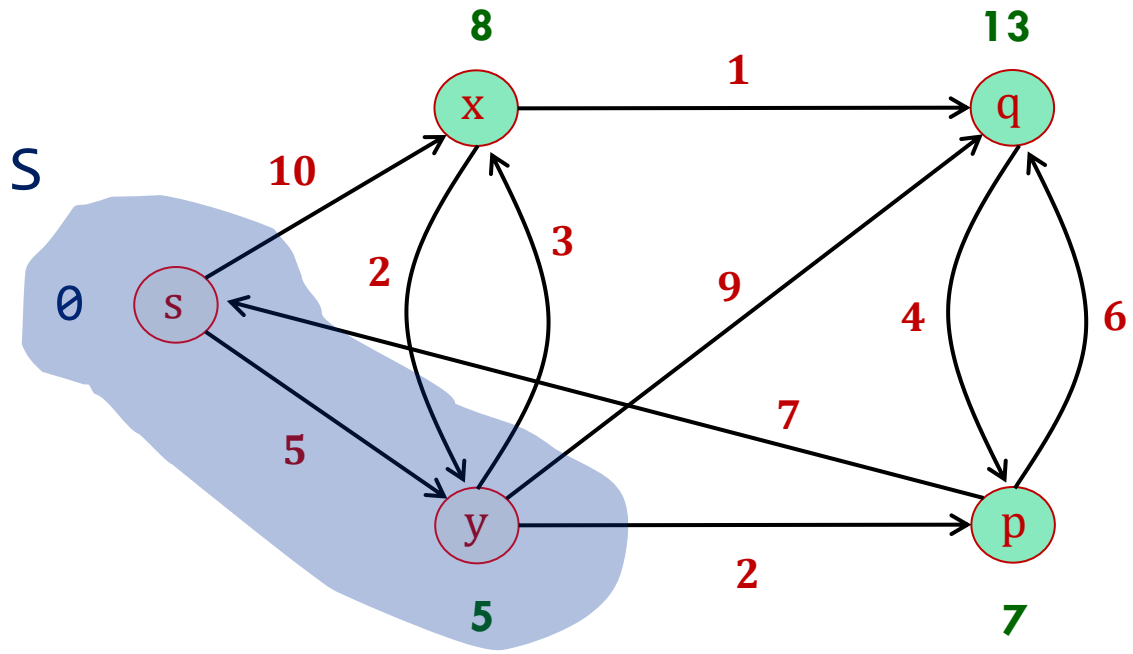
d(s)=0 d(y)=5

prev(s)=NIL prev(y)=s

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

```
Q = {x,p,q}
while Q ≠ ∅ :
    u ← extract-min(Q);
```

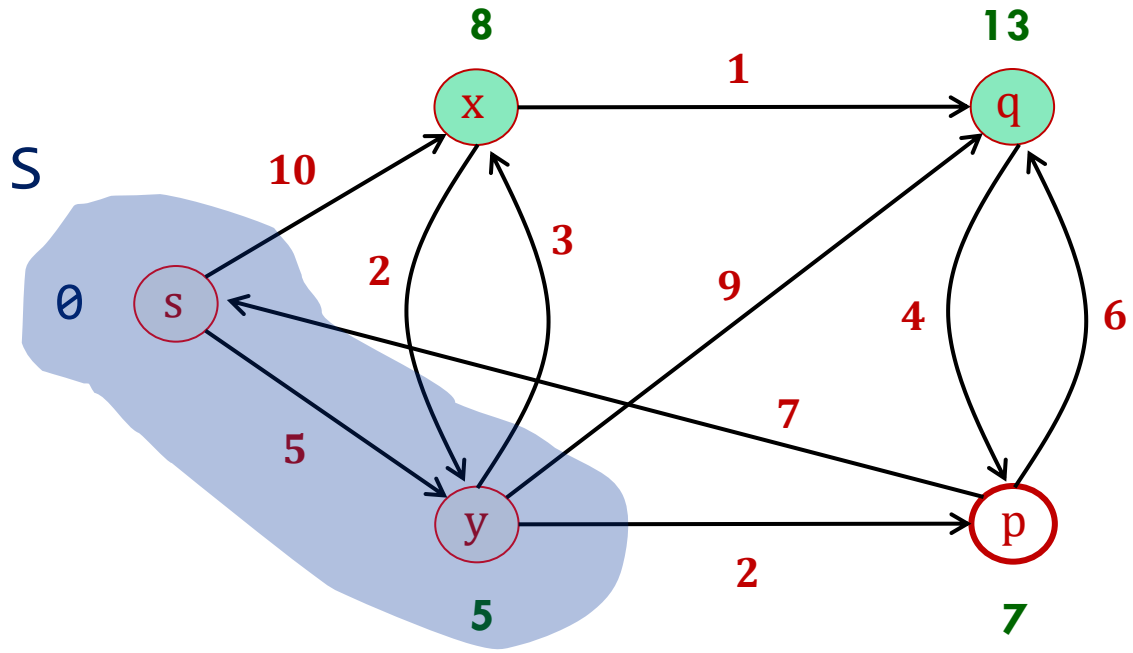


d(s)=0 d(y)=5

prev(s)=NIL prev(y)=s

Update() – Αλγόριθμος Dijkstra

Παράδειγμα



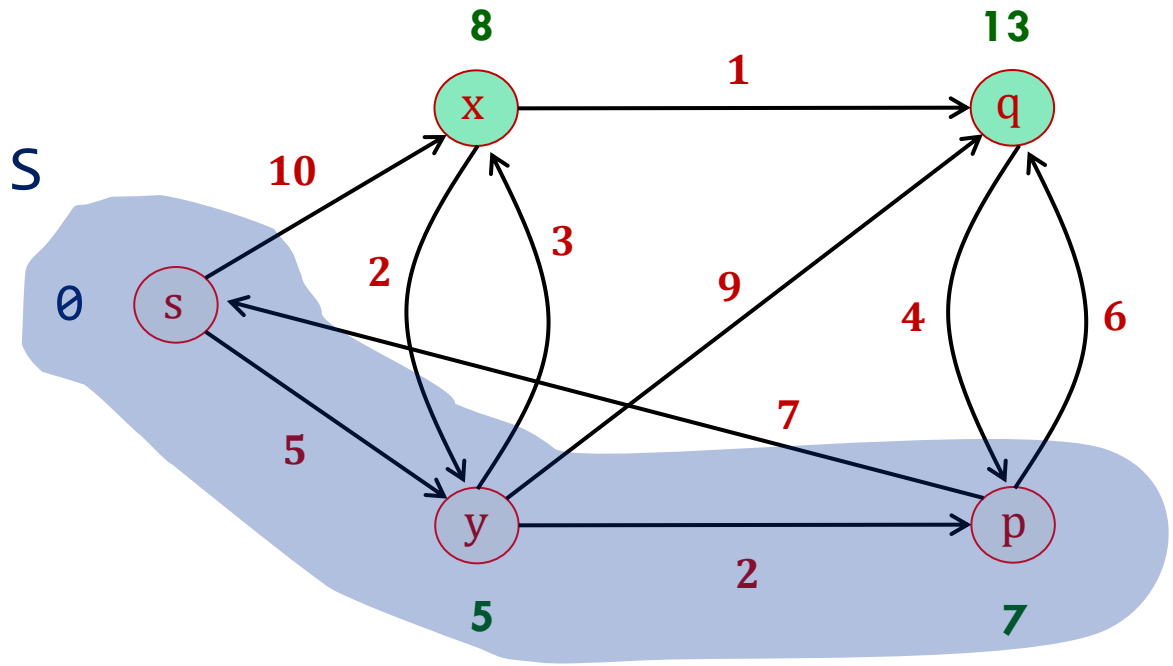

```
Q = {x,q}
while Q ≠ ∅ :
    u ← extract-min(Q);
```

```
d(s)=0        d(y)=5      d(p)=7
prev(s)=NIL   prev(y)=s   prev(p)=y
```

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

```
Q = {x,q}
while Q ≠ ∅ :
    u ← extract-min(Q);
    S ← S ∪ {u}
```

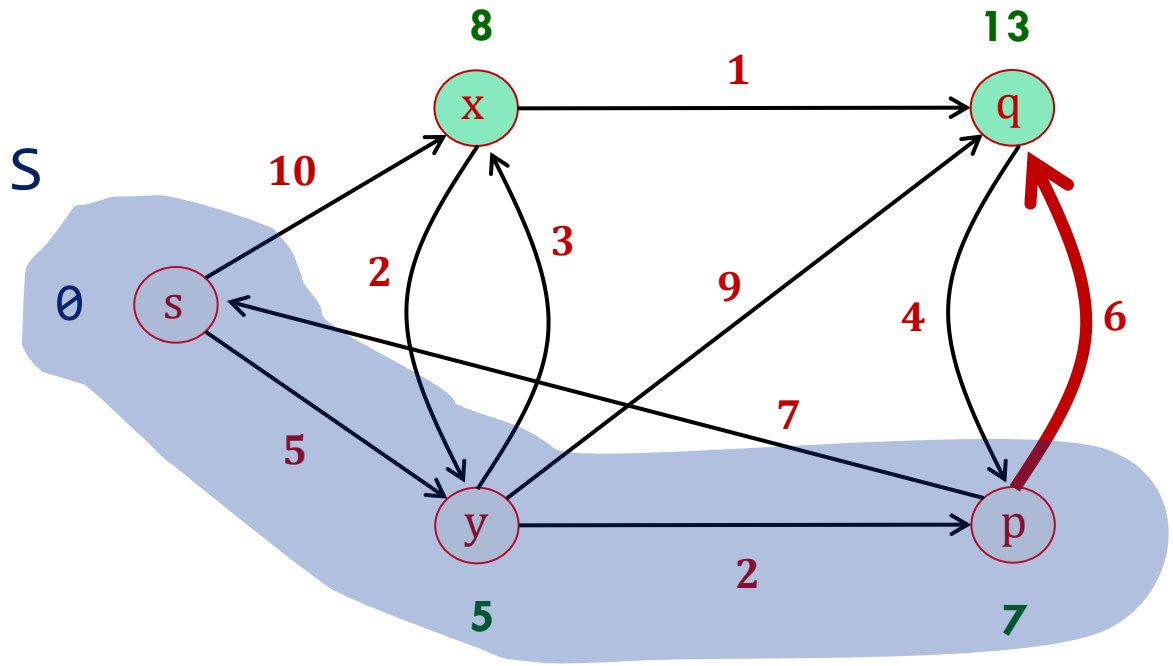


d(s)=0 d(y)=5 d(p)=7

prev(s)=NIL prev(y)=s prev(p)=y

Update() – Αλγόριθμος Dijkstra

Παράδειγμα




```
Q = {x,q}
while Q ≠ ∅ :
    u ← extract-min(Q);
    S ← S ∪ {u}
    Update(u,v,w)
```

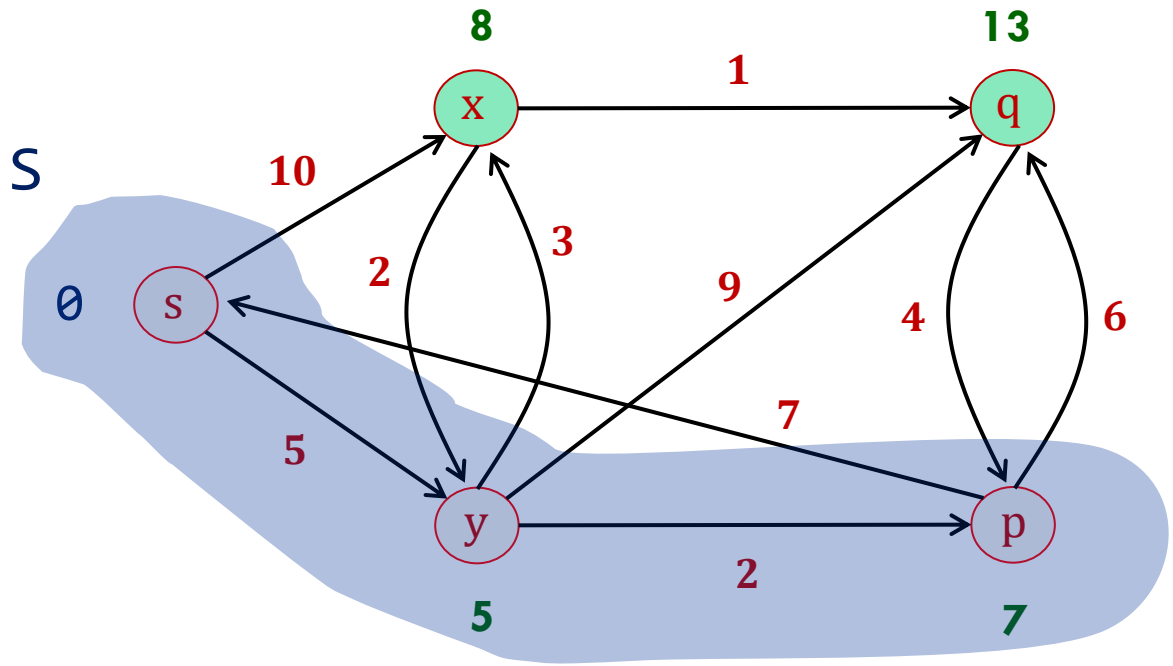
d(s)=0 d(y)=5 d(p)=7

prev(s)=NIL prev(y)=s prev(p)=y

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

```
Q = {x,q}
while Q ≠ ∅ :
    u ← extract-min(Q);
```

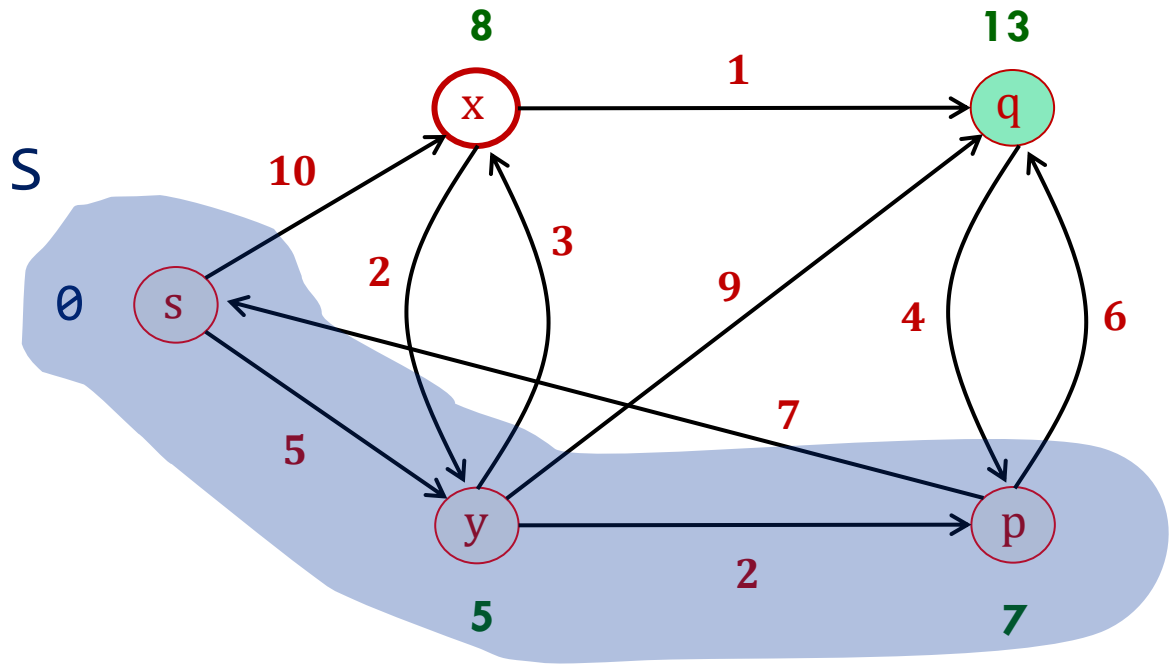



```
d(s)=0        d(y)=5      d(p)=7
prev(s)=NIL   prev(y)=s   prev(p)=y
```

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

```
Q = {q}
while Q ≠ ∅ :
    u ← extract-min(Q);
```

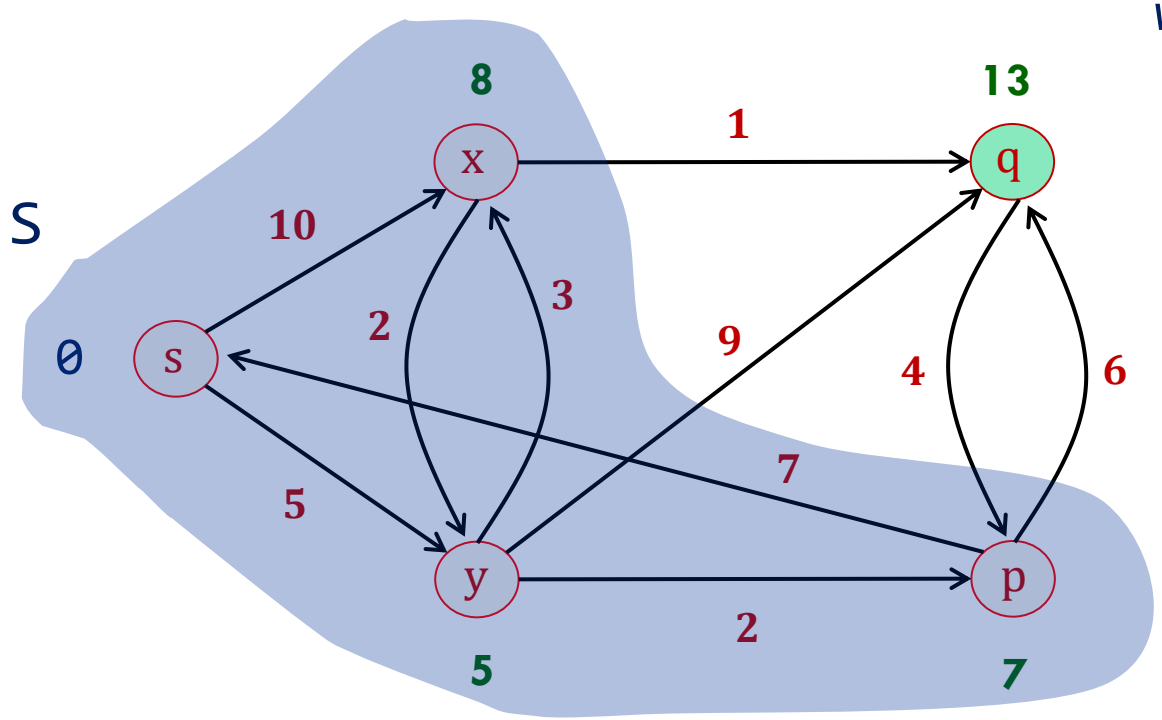


d(s)=0 d(y)=5 d(p)=7 d(x)=8

prev(s)=NIL prev(y)=s prev(p)=y prev(x)=y

Update() – Αλγόριθμος Dijkstra

Παράδειγμα



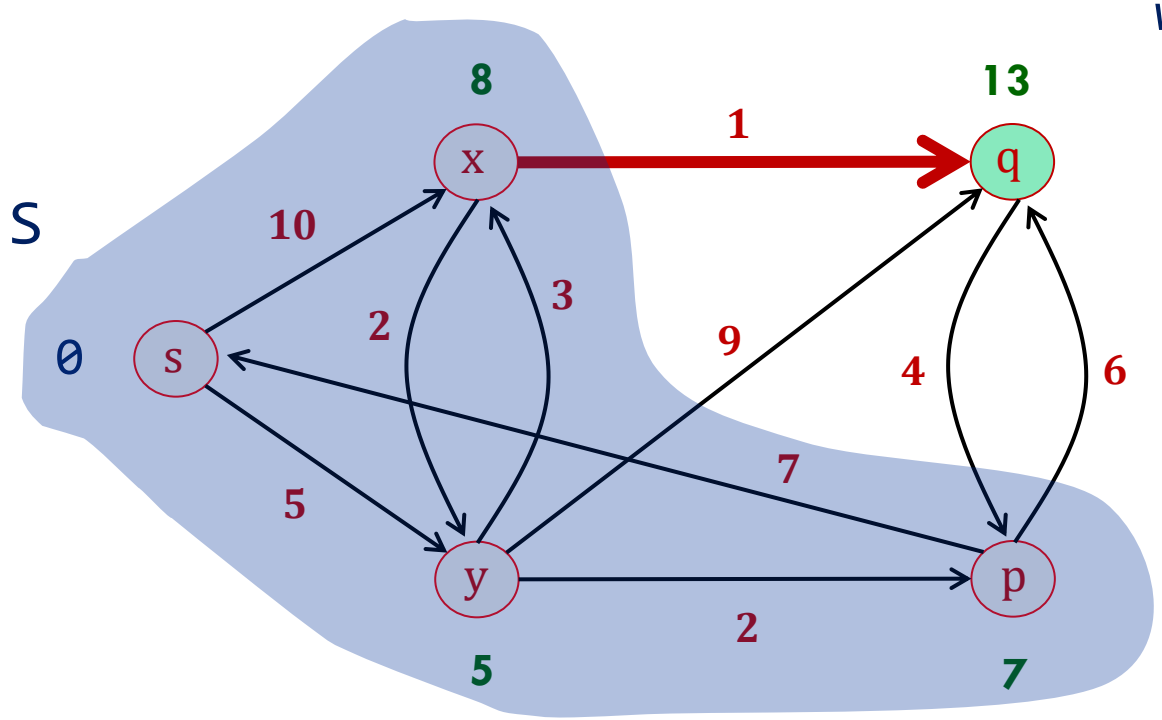

```
Q = {q}
while Q ≠ ∅ :
    u ← extract-min(Q);
    S ← S ∪ {u}
```

d(s)=0 d(y)=5 d(p)=7 d(x)=8

prev(s)=NIL prev(y)=s prev(p)=y prev(x)=y

Update() – Αλγόριθμος Dijkstra

Παράδειγμα




```
Q = {q}
while Q ≠ ∅ :
    u ← extract-min(Q);
    S ← S ∪ {u}
    Update(u,v,w)
```

d(s)=0 prev(s)=NIL

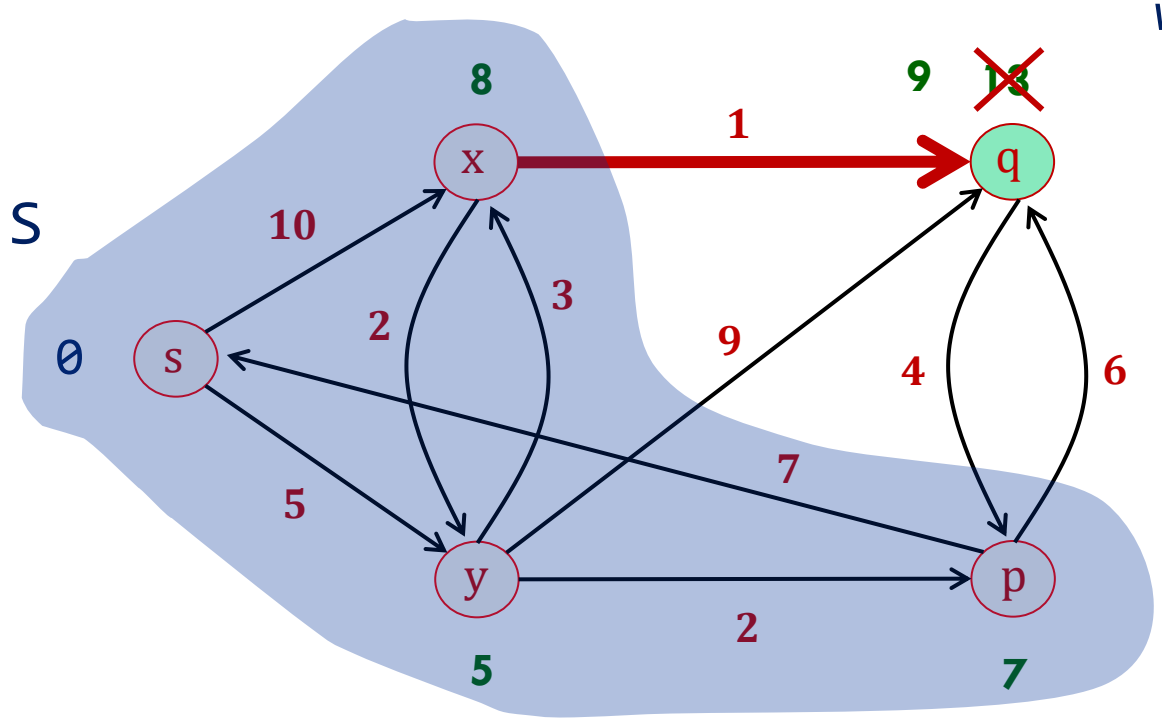
d(y)=5 prev(y)=s

d(p)=7 prev(p)=y

d(x)=8 prev(x)=y

Update() – Αλγόριθμος Dijkstra

Παράδειγμα




```
Q = {q}
while Q ≠ ∅ :
    u ← extract-min(Q);
    S ← S ∪ {u}
    Update(u,v,w)
```

d(s)=0 prev(s)=NIL

d(y)=5 prev(y)=s

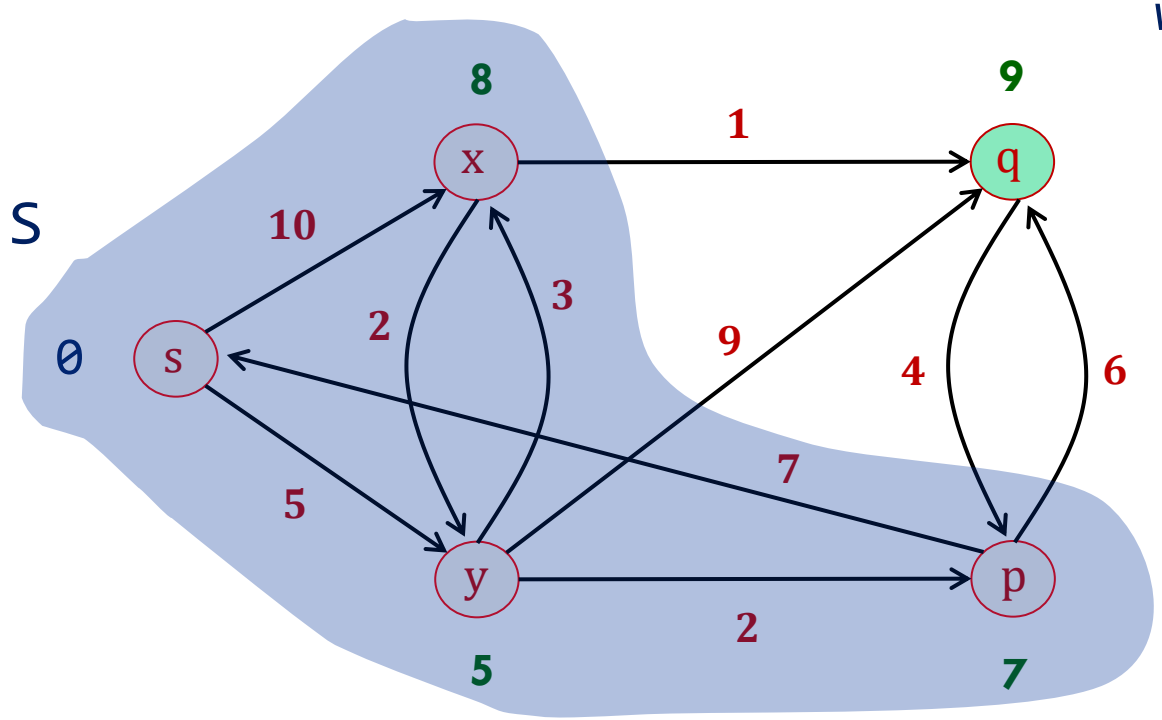
d(p)=7 prev(p)=y

d(x)=8 prev(x)=y

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

```
Q = {q}
while Q ≠ ∅ :
    u ← extract-min(Q);
```



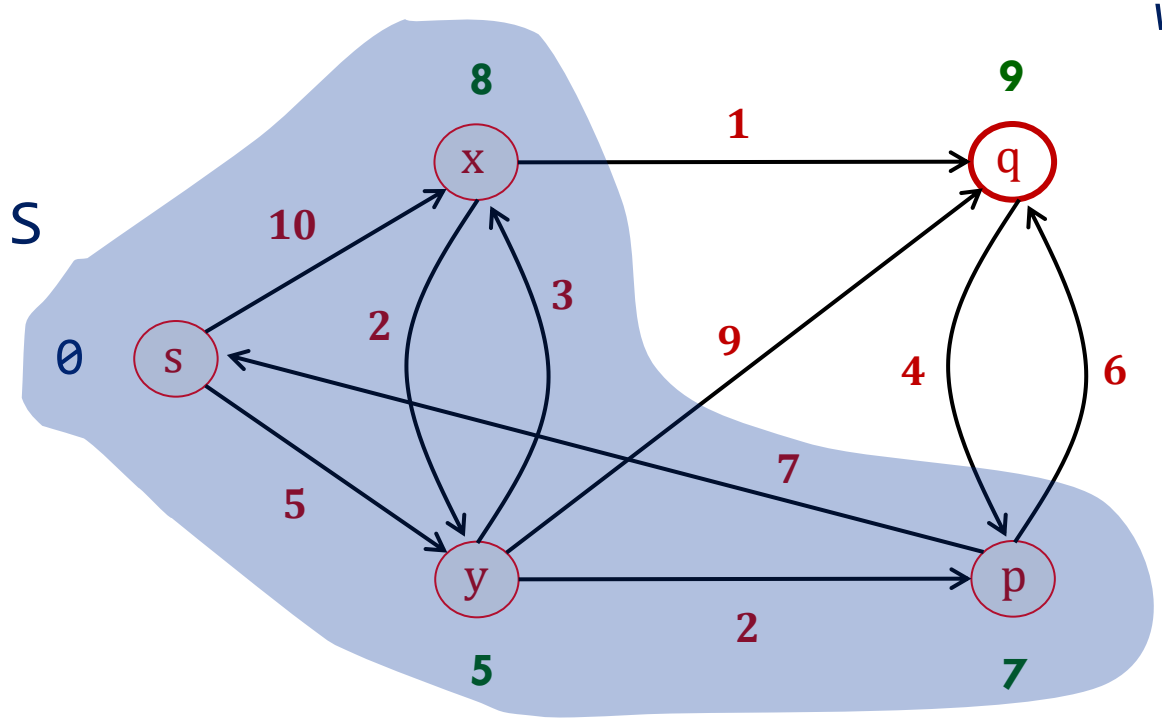
d(s)=0 d(y)=5 d(p)=7 d(x)=8

prev(s)=NIL prev(y)=s prev(p)=y prev(x)=y

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

```
Q = {}
while Q ≠ ∅ :
    u ← extract-min(Q);
```

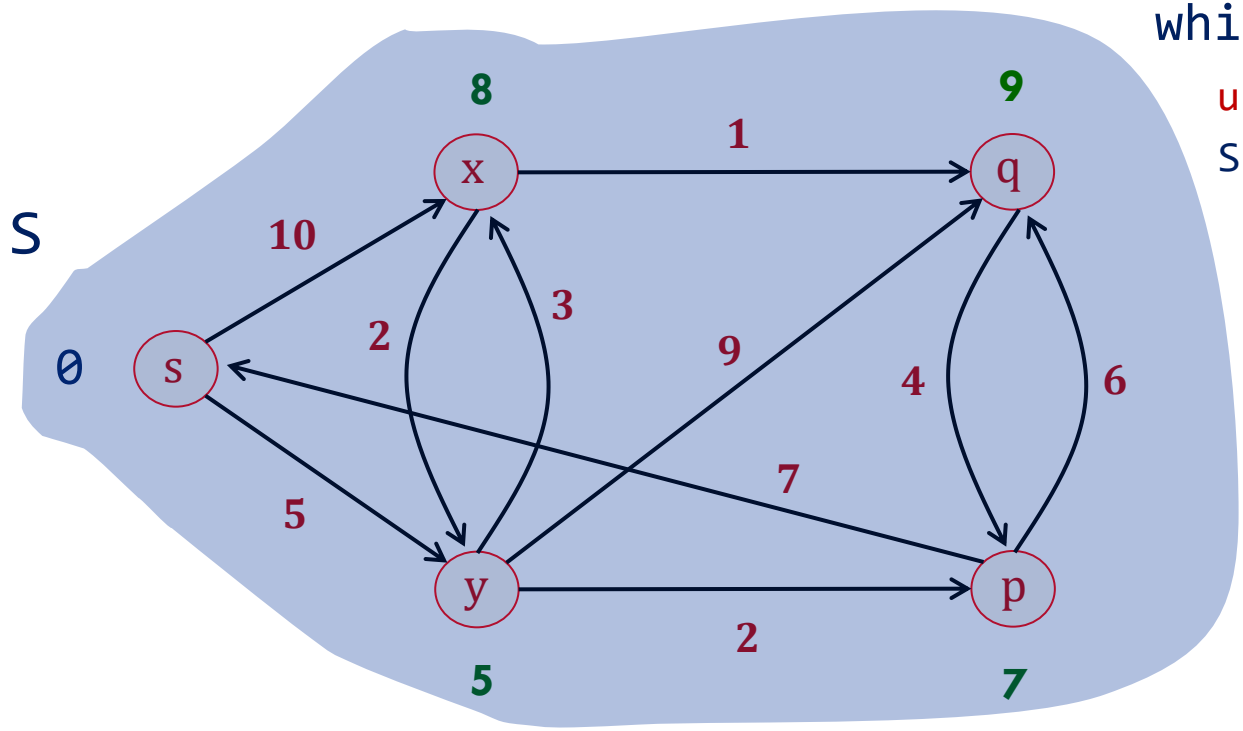


d(s)=0	d(y)=5	d(p)=7	d(x)=8	d(q)=9
prev(s)=NIL	prev(y)=s	prev(p)=y	prev(x)=y	prev(q)=x

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

```
Q = {}
while Q ≠ ∅ :
    u ← extract-min(Q);
    S ← S ∪ {u}
```

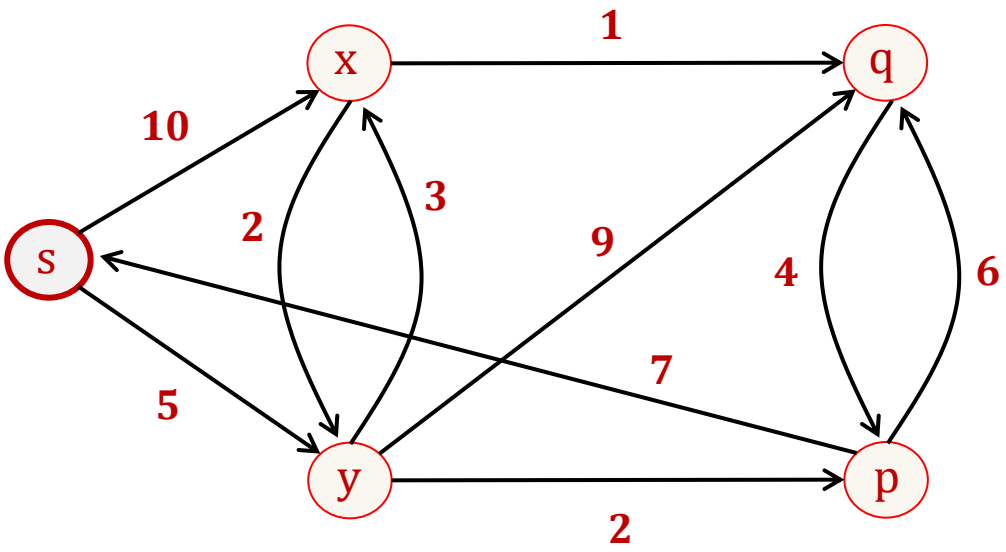


d(s)=0	d(y)=5	d(p)=7	d(x)=8	d(q)=9
prev(s)=NIL	prev(y)=s	prev(p)=y	prev(x)=y	prev(q)=x

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

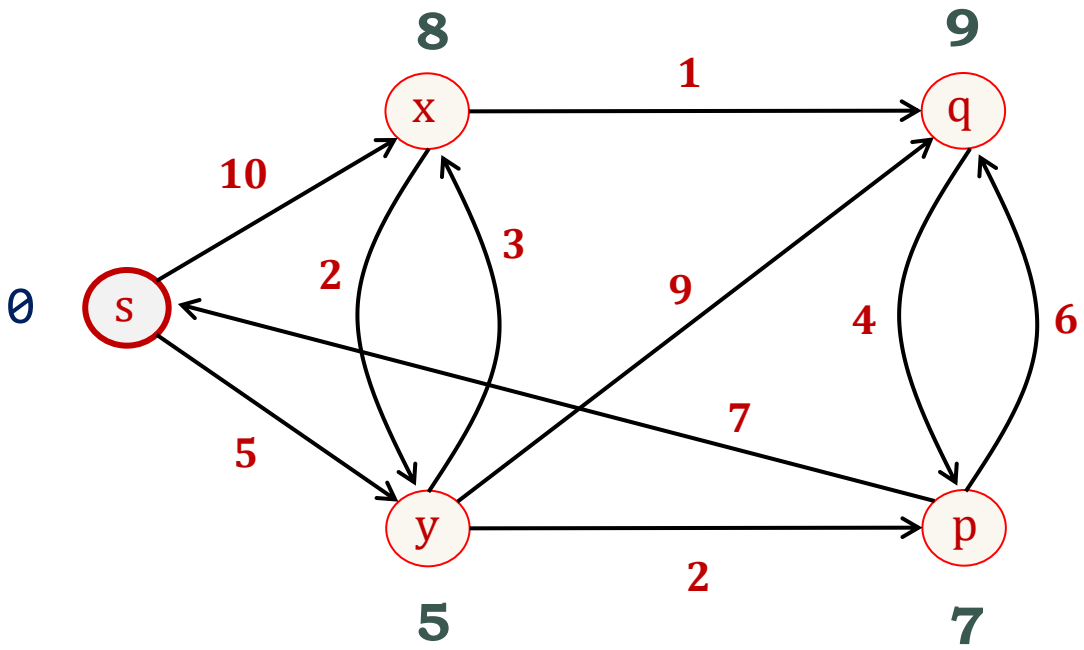
```
Q = {}
while Q ≠ ∅ :
    return d(), prev()
```



d(s)=0	d(y)=5	d(p)=7	d(x)=8	d(q)=9
prev(s)=NIL	prev(y)=s	prev(p)=y	prev(x)=y	prev(q)=x

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

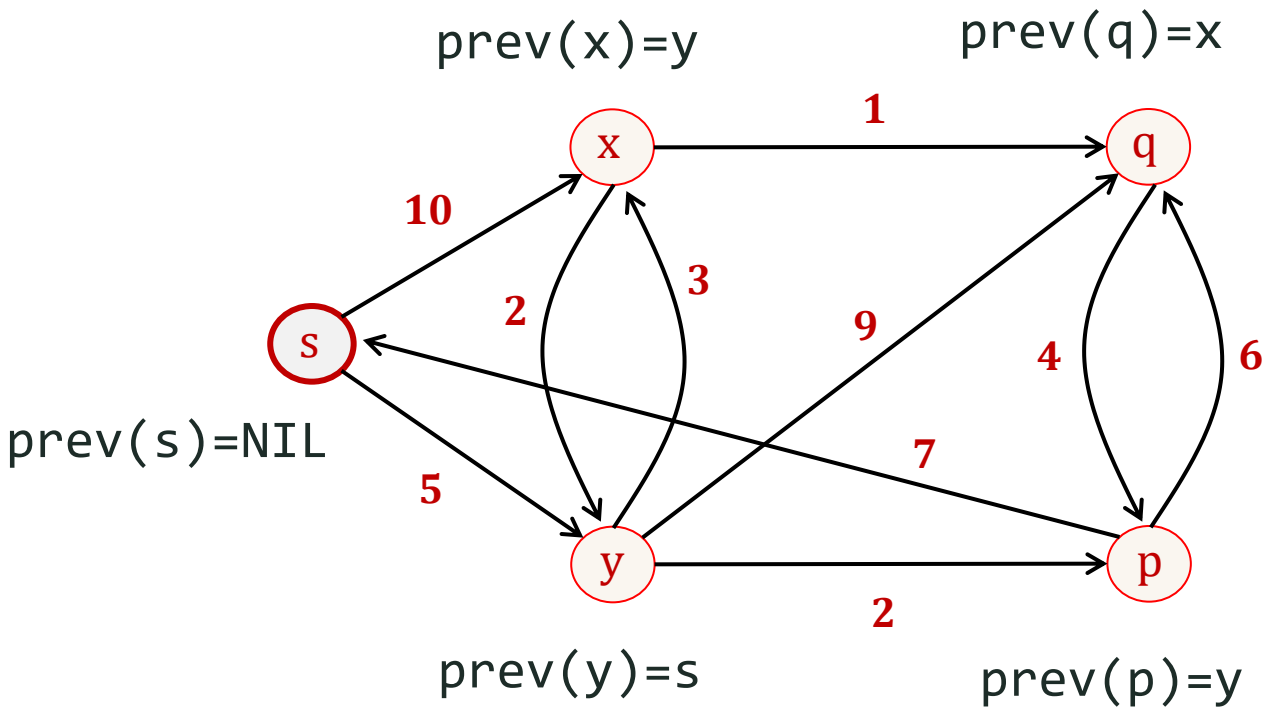


Έξοδος Αλγόριθμου

Κόστος ελάχιστων διαδρομών

Update() – Αλγόριθμος Dijkstra

Παράδειγμα

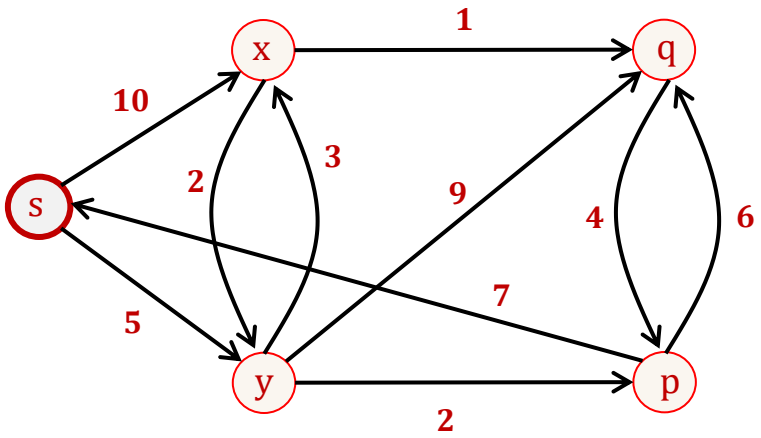


Έξοδος Αλγόριθμου

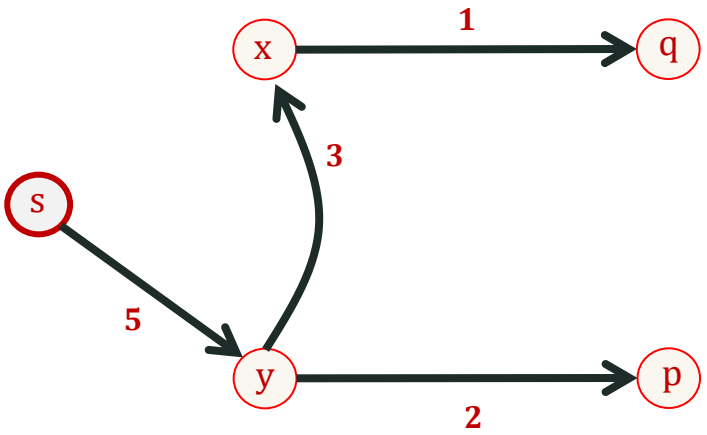
Προκάτοχοι ελάχιστων διαδρομών

Update() – Αλγόριθμος Dijkstra

- Παράδειγμα

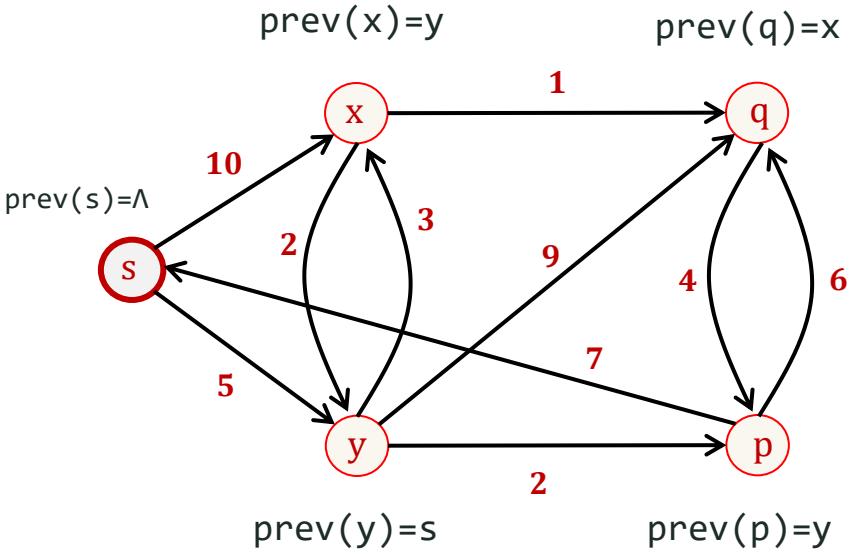


Δένδρο Ελάχιστων Διαδρομών



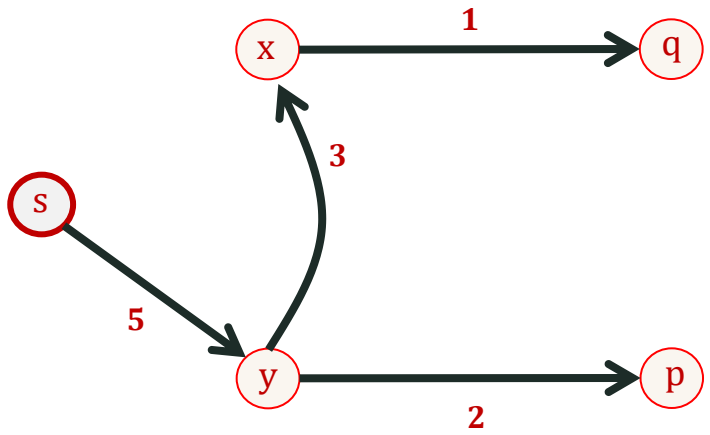
Update() – Αλγόριθμος Dijkstra

Παράδειγμα



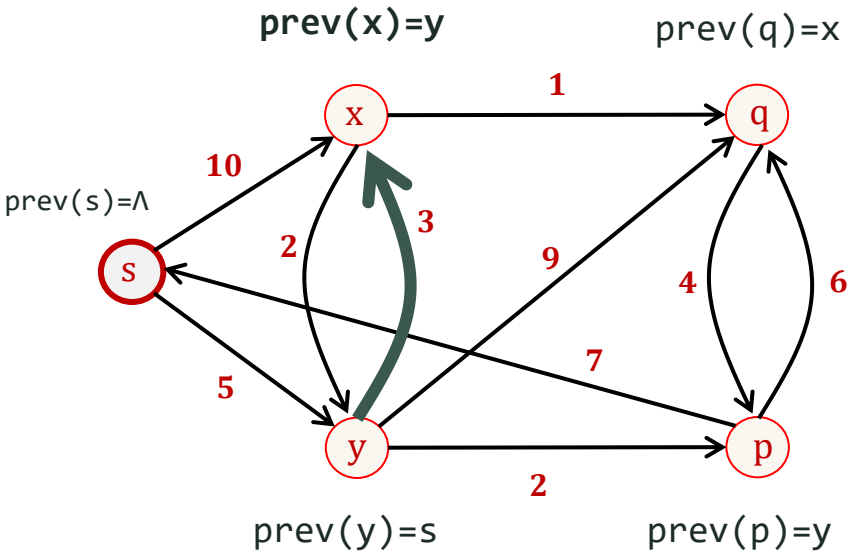
Δένδρο Ελάχιστων Διαδρομών

ΚΑΤΑΣΚΕΥΗ



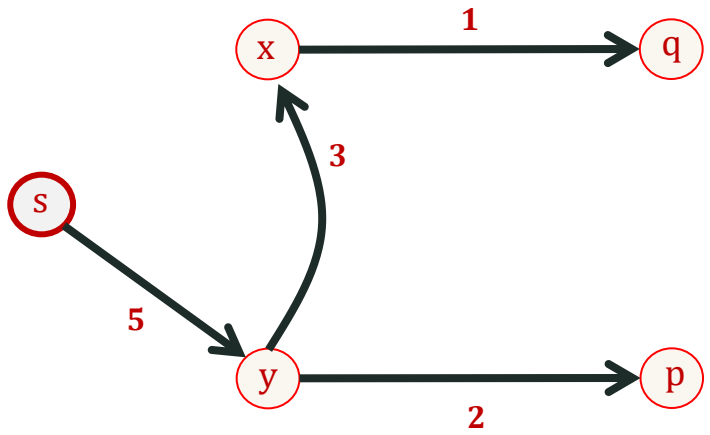
Update() – Αλγόριθμος Dijkstra

Παράδειγμα



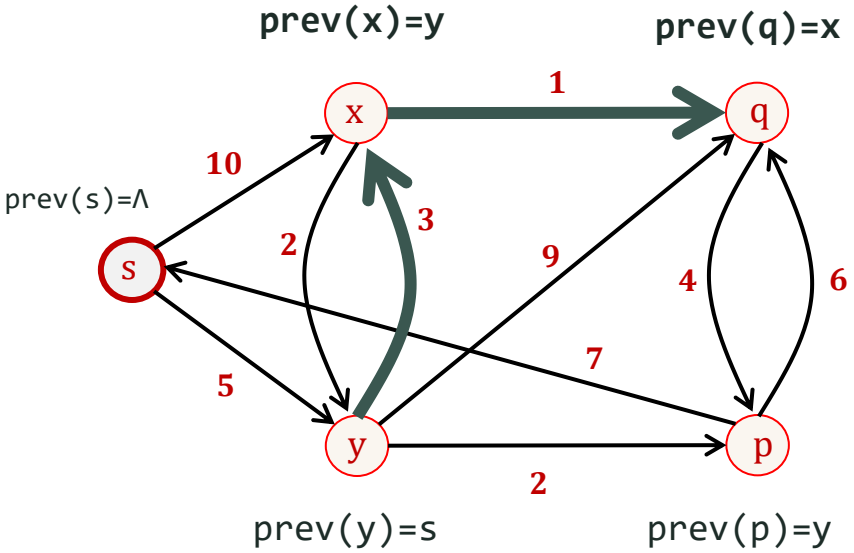
Δένδρο Ελάχιστων Διαδρομών

ΚΑΤΑΣΚΕΥΗ



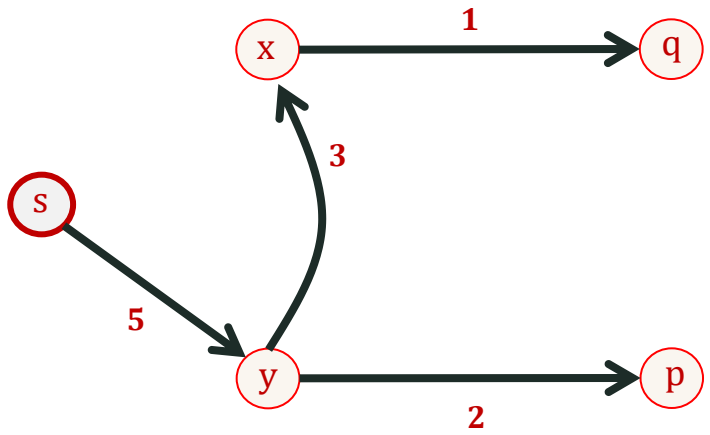
Update() – Αλγόριθμος Dijkstra

Παράδειγμα



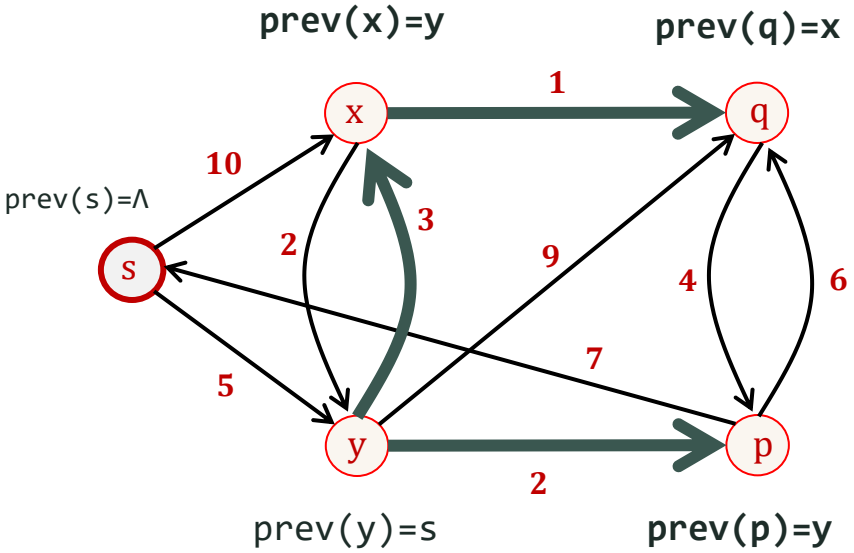
Δένδρο Ελάχιστων Διαδρομών

ΚΑΤΑΣΚΕΥΗ



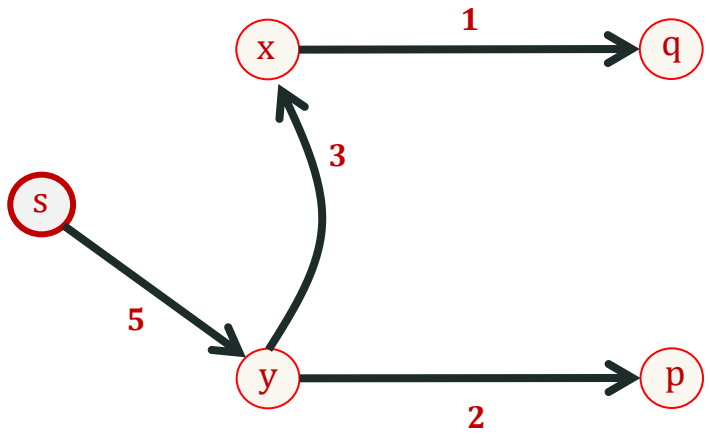
Update() – Αλγόριθμος Dijkstra

Παράδειγμα



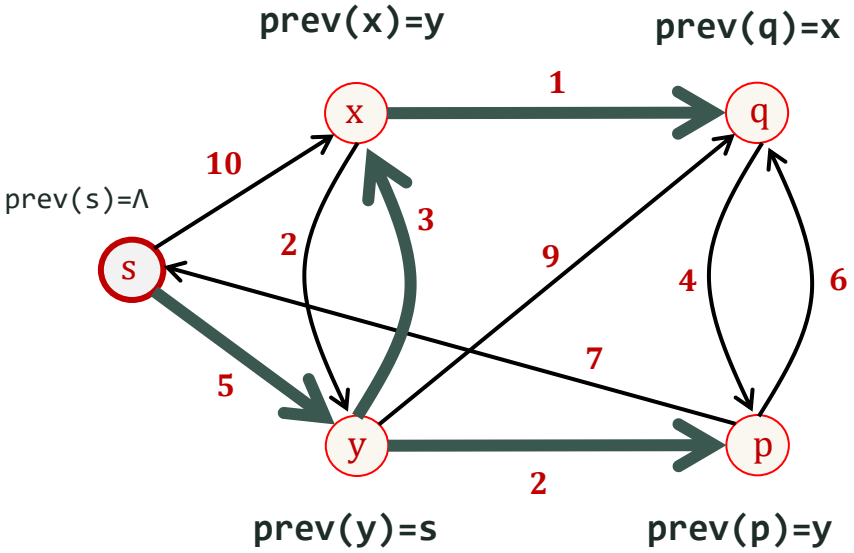
Δένδρο Ελάχιστων Διαδρομών

ΚΑΤΑΣΚΕΥΗ



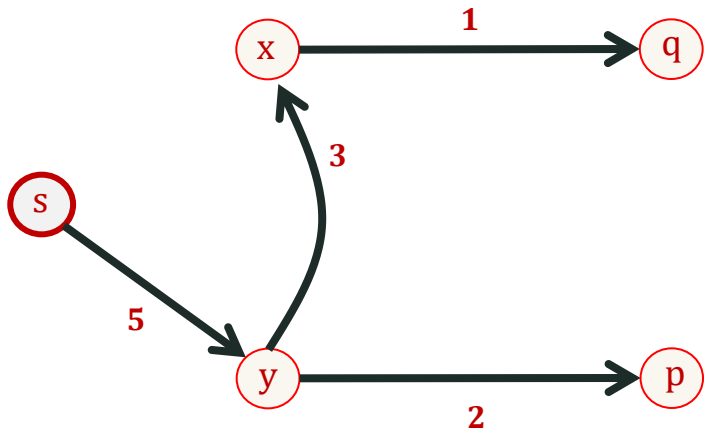
Update() – Αλγόριθμος Dijkstra

Παράδειγμα



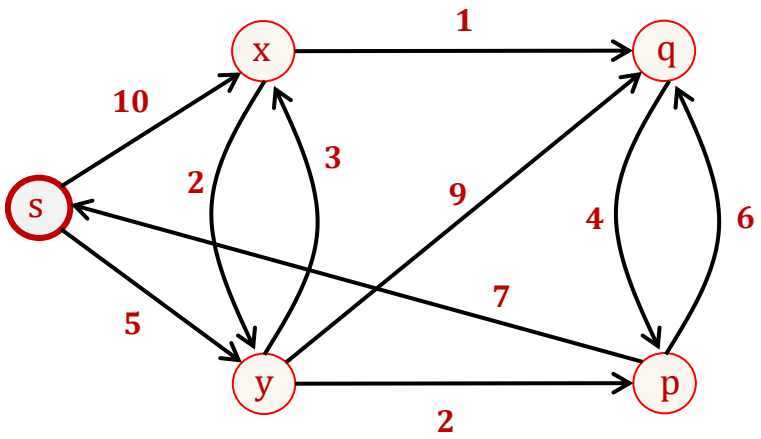
Δένδρο Ελάχιστων Διαδρομών

ΚΑΤΑΣΚΕΥΗ



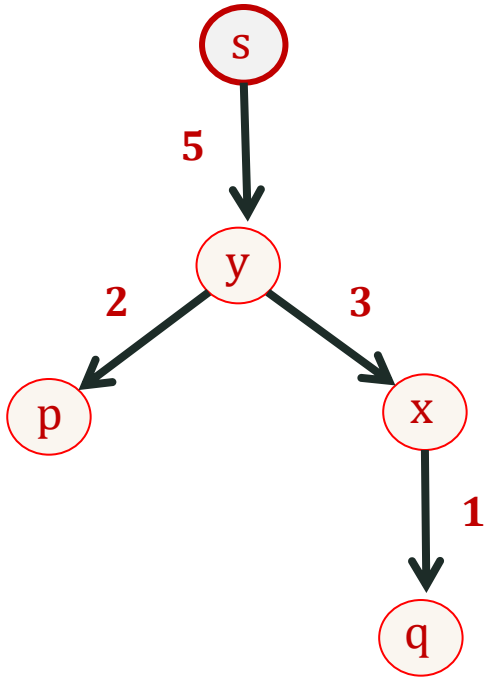
Update() – Αλγόριθμος Dijkstra

Παράδειγμα



Δένδρο Ελάχιστων Διαδρομών

ΚΑΤΑΣΚΕΥΗ



Πολυπλοκότητα Dijkstra

Πολυπλοκότητα Αλγόριθμου Dijkstra

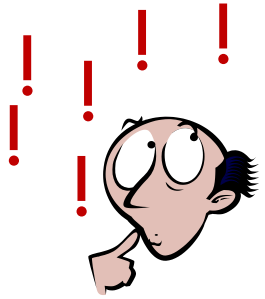
- Δομικά παρόμοιος με τον αλγόριθμο BFS
- Πιο αργός από BFS (ουρά προτεραιότητας vs απλής ουράς)
- Απαιτεί **n** λειτουργίες insert στην ουρά, και επομένως:

  n `extract-min`

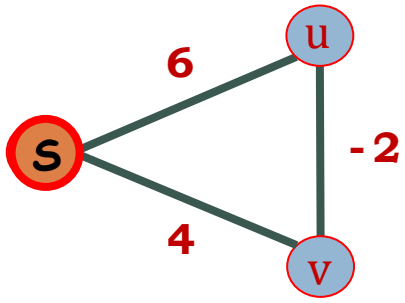
  n+m `insert/updata`

- Υλοποίηση ουράς με Heap $\Rightarrow$ **O((n+m)logn)**

2. Αλγόριθμος Bellman-Ford



Βασική Ιδέα !!!



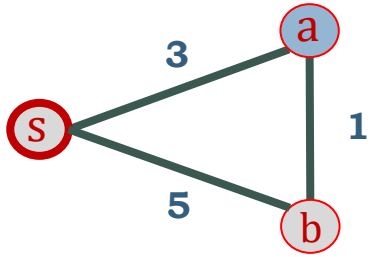
Ο αλγόριθμος των Bellman-Ford επιλύει το πρόβλημα των ελαχίστων διαδρομών στην πιο γενική περίπτωση όπου το γράφημα μπορεί να έχει αρνητικά ακμικά βάρη.

Ενημερώνει με `Update()` ΟΛΛΕΣ τις ακμές του γραφήματος `G=(V,E)` `n-1` φορές, όπου `n=|V|` !

Ενδιαφέρουσες Παρατηρήσεις

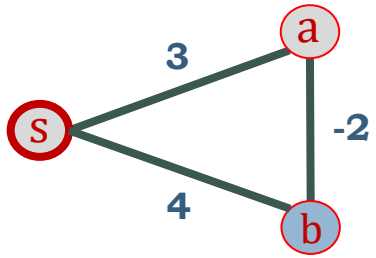
- Ο αλγόριθμος του Dijkstra λειτουργεί σωστά διότι η ε.δ **s ~> v «περνάει»** αποκλειστικά από κόμβους που είναι **πιο κοντά** στον **s**.

  Η ε.δ **(s,a,b)** από τον **s** στον **b «περνά»** μέσω του **a** που είναι *πιο κοντά* στον **s**.



- Αυτό **ΔΕΝ ισχύει** με αρνητικά βάρη !

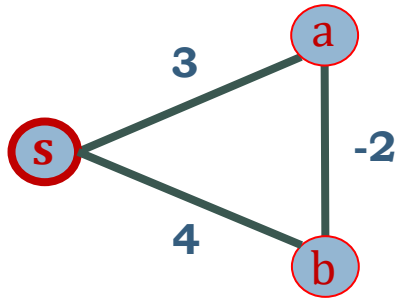
  Η ε.δ **(s,b,a)** από τον **s** στον **b «περνά»** μέσω του **b** που είναι *πιο μακριά* στον **s**.



Ενδιαφέρουσες Παρατηρήσεις

- Πράγματι, εδώ ο Dijkstra δεν λειτουργεί σωστά !!!

Θα δώσει λανθασμένα : d(s,a)=3
Ενώ ισχύει: d(s,a)=2



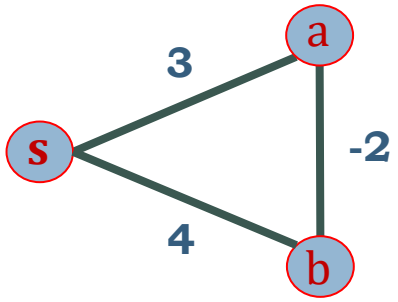
Τι πρέπει να αλλάξουμε στον Dijkstra ?

Ενδιαφέρουσες Παρατηρήσεις

- **Ιδιότητα των τιμών d(.)**

  Κατά τη διάρκεια λειτουργίας του Dijkstra, ισχύει:

  d(.) είναι είτε υπερεκτιμημένες
  είτε βέλτιστες !!!



- Αρχικοποιούνται d(.)= ∅ και αλλάζουν μόνο:

```
Update(u,v,w):
    d(v) ← min{d(v), d(u) + w(u,v)}
```

Ενδιαφέρουσες Παρατηρήσεις

- Η διαδικασία `Update(u,v,w)`:

(1) Δίνει **την τιμή της ε.δ.** στην **`d(v)`** όταν ο κόμβος `u` είναι προτελευταίος στην ε.δ. `s ~> v`

(2) Δεν δίνει ποτέ στην `d(v)` τιμή **`<`** από την τιμή της ε.δ. `s ~> v`

Όταν η `d(v)` πάρει την τιμή της ε.δ, στη συνέχεια όσες φορές και να εφαρμόσουμε την

`Update(u,v,w)`

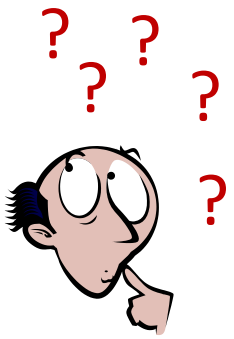
η τιμή της `d(v)` δεν αλλάζει !!!

Ενδιαφέρουσες Παρατηρήσεις

- Αλγόριθμο του Dijkstra:

  « Μια **ακολουθία εκτελέσεων** της `Update()`»

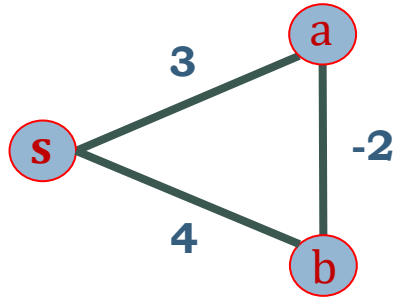
- Η ακολουθία των `Update()` του Dijkstra δεν λειτουργεί σωστά με αρνητικά βάρη !



Ερώτημα !!!

Υπάρχει κάποια άλλη ακολουθία από `Update()` που λειτουργεί σωστά ?

Εάν ναι, τι ιδιότητες πρέπει να έχει ?

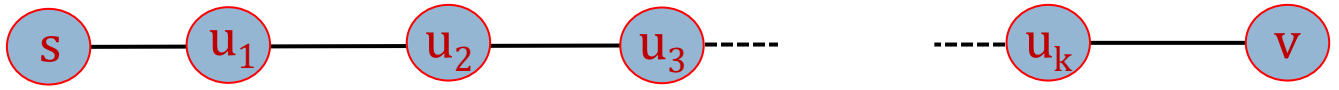


`(s,a)`, `(s,b)`, `(a,b)`

Ενδιαφέρουσες Παρατηρήσεις

- Έστω P: s ~> v μια ε.δ.

Η P σε γράφημα τάξης n έχει το πολύ n-1 ακμές (γιατί ?)



- Εάν η ακολουθία εκτελέσεων Update() είναι:

$(s,u_1), (u_1,u_2), (u_2,u_3), \ldots, (u_k,v)$

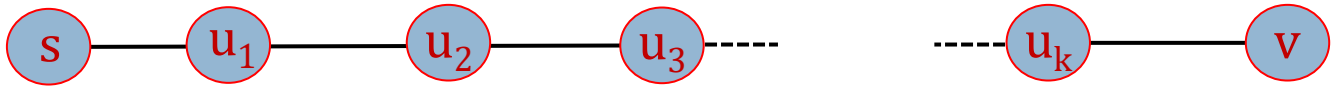
τότε, λόγω τις (1), η τιμή d(v) θα υπολογιστεί σωστά !

(1) Η διαδικασία Update(u,v,w) δίνει την τιμή της ε.δ. στην d(v) όταν ο κόμβος u είναι προτελευταίος στην ε.δ s ~> v

Ενδιαφέρουσες Παρατηρήσεις

- Έστω P: s ~> v μια ε.δ.

Η P σε γράφημα τάξης n έχει το πολύ n-1 ακμές (γιατί ?)



- Όταν υπολογιστεί η βέλτιστη τιμή d(v), λόγω της (2), όποιες άλλες ενημερώσεις Update() γίνουν στις ακμές

  (s,u_1), (u_1,u_2), (u_2,u_3), …, (u_k,v)

  δεν επηρεάζουν το αποτέλεσμα !

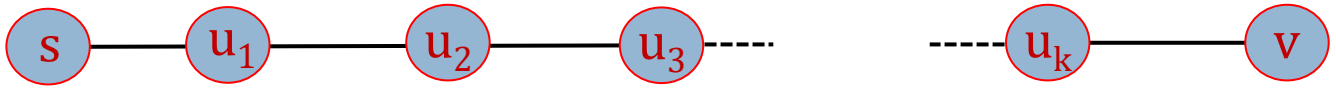
(2) Δεν θα δώσει ποτέ στην d(v) τιμή μικρότερη από την τιμή της ε.δ s ~> v

Ενδιαφέρουσες Παρατηρήσεις

- **Όμως !!!** …

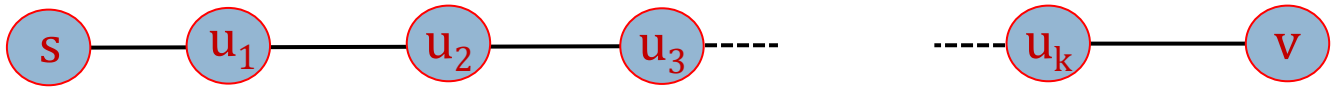
ΔΕΝ γνωρίζουμε εκ των προτέρων όλες τις ε.δ του G και, επομένως,

ΔΕΝ είμαστε βέβαιοι ότι ενημερώνουμε, με `Update()`, τις **σωστές ακμές** με τη **σωστή σειρά** !



Πρόβλημα !!!... Μπορώ να σκεφτώ κάτι ?

- **Ορίστε μια Λύση !!!!**



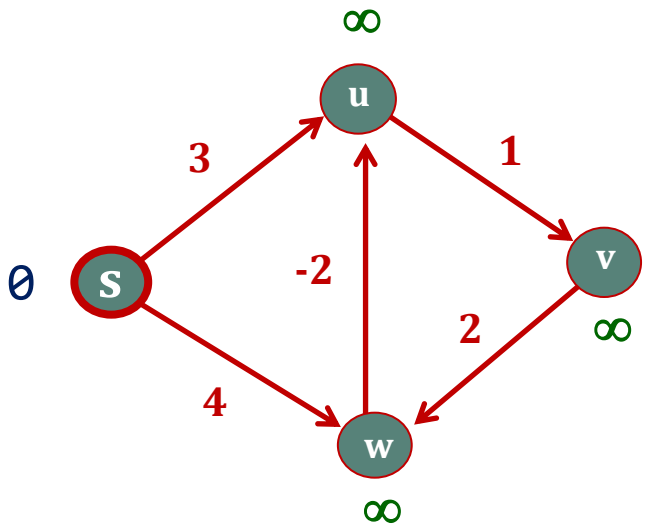
Ενημέρωσε, με `Update()`, ΟΛΕΣ τις ακμές `n-1` φορές !

Η διαδικασία αυτή είναι, απλά, ο γνωστός αλγόριθμος των

Bellman-Ford

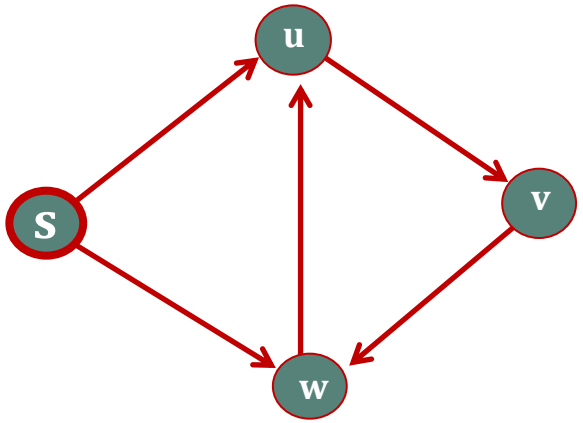
2. Αλγόριθμος Bellman-Ford

```
Initialize(G,s)
 1.  για κάθε κόμβο v∈V:
          d(v) ← ∞
          prev(v) ← NIL
 2.  d(s) ← 0
```

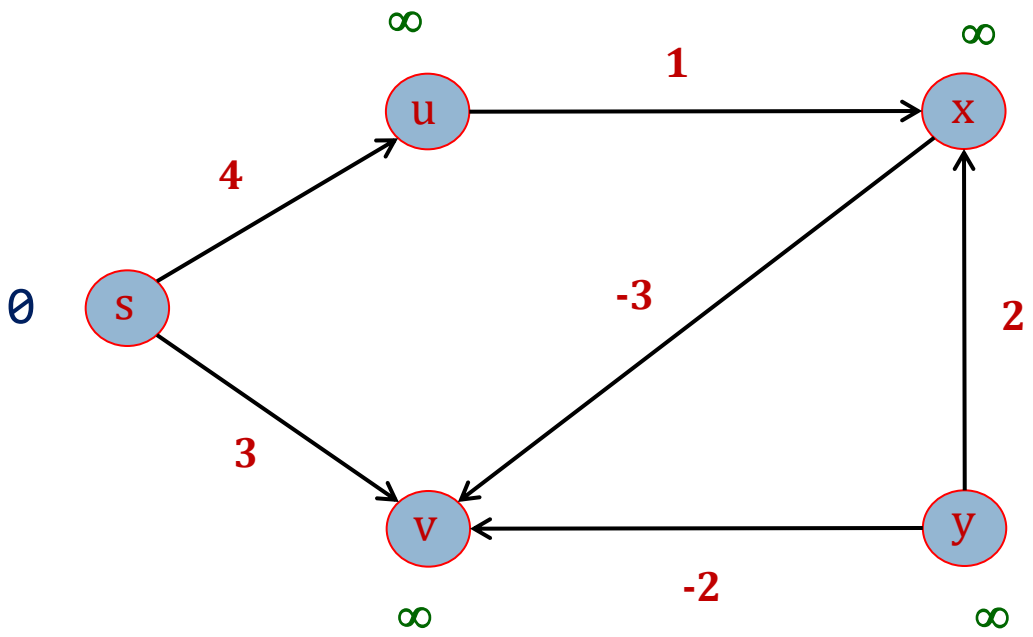


2. Αλγόριθμος Bellman-Ford

```
Bellman-Ford(G,w,s)
1. Initialize(G,s)
2. επανάλαβε n-1 φορές:
        για κάθε ακμή (u,v)∈E:
            Update(u,v,w)
3. return d(.)
```



Παραδείγματα



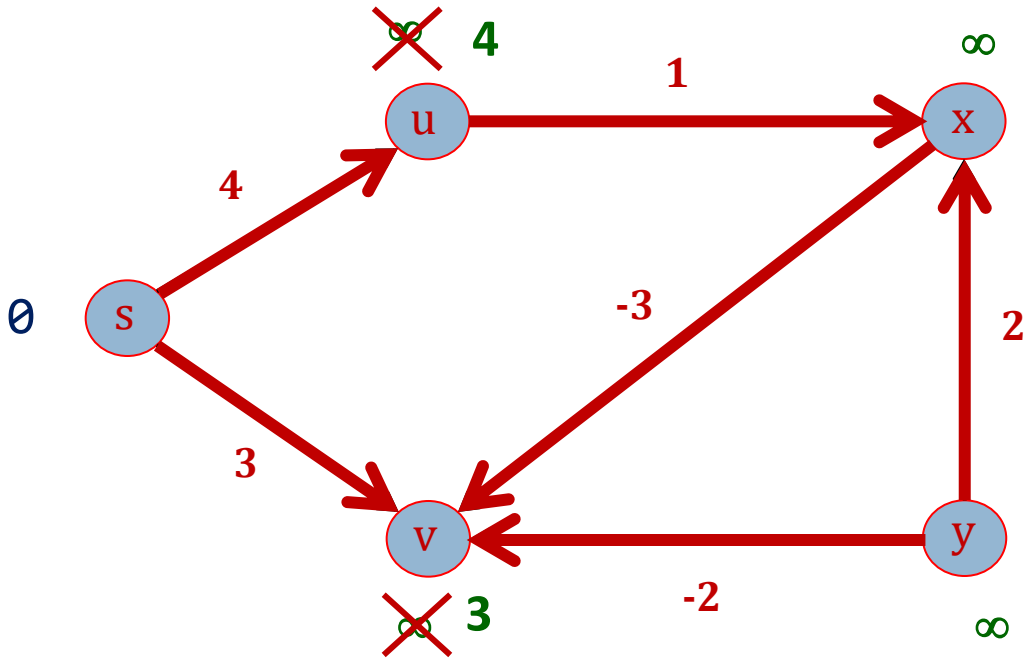
n = 5
m = 6

Update στις
6 ακμές
4 φορές

V = {s, u, v, x, y}

E = {(y,x),(u,x),(y,v),(s,u),(x,v),(s,v)}

Παράδειγμα 1



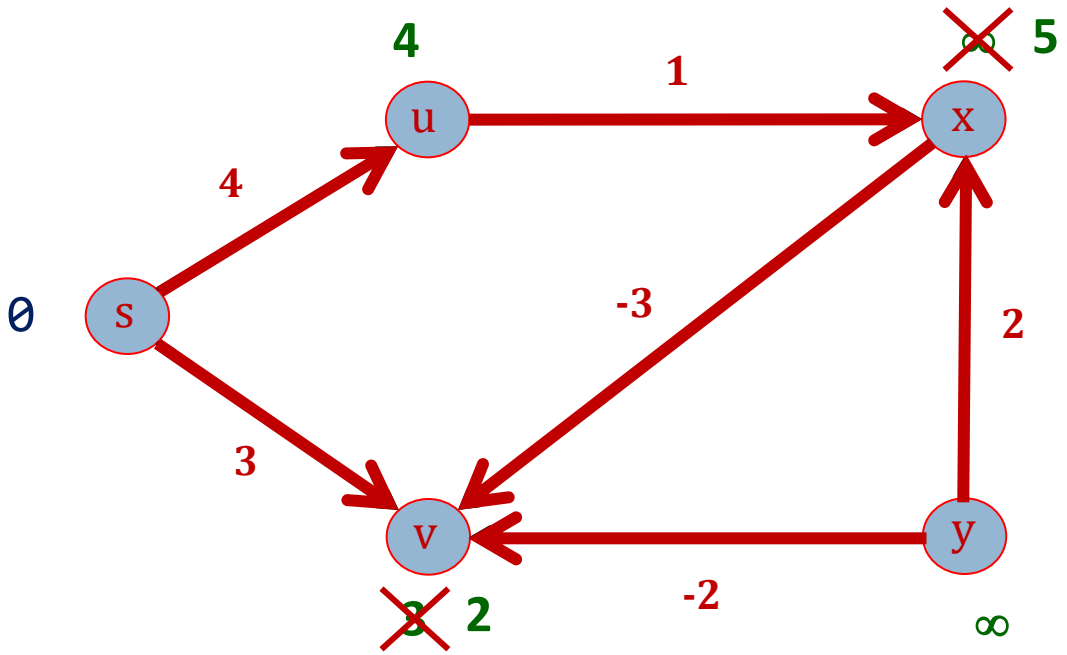
n = 5
m = 6

Update στις
6 ακμές
4 φορές

1η Επανάληψη

S1: (y,x), (u,x), (y,v), (s,u), (x,v), (s,v)

Παράδειγμα 1



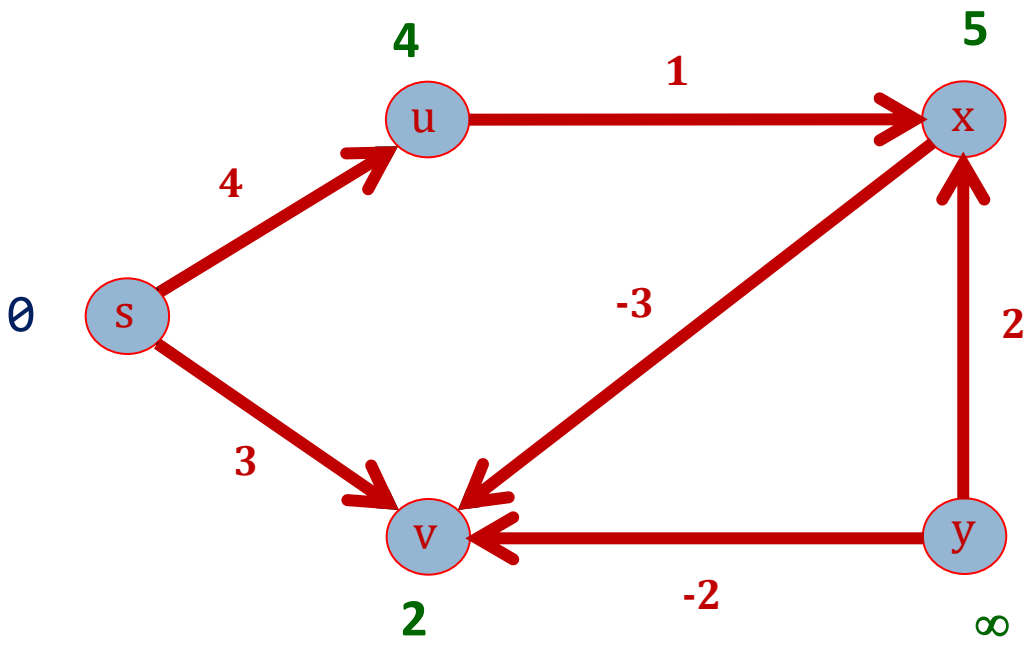
n = 5
m = 6

Update στις
6 ακμές
4 φορές

2η Επανάληψη

S1: (y,x), (u,x), (y,v), (s,u), (x,v), (s,v)

Παράδειγμα 1



n = 5
m = 6

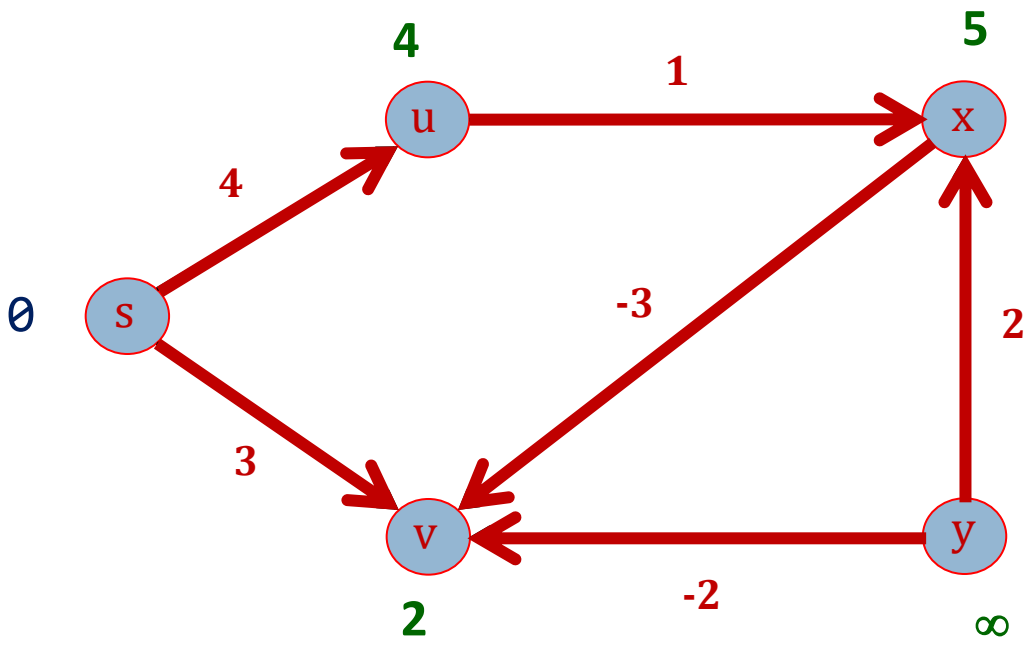
Update στις
6 ακμές
4 φορές

3η Επανάληψη

Καμία Αλλαγή !

S1: (y,x), (u,x), (y,v), (s,u), (x,v), (s,v)

Παράδειγμα 1



n = 5
m = 6

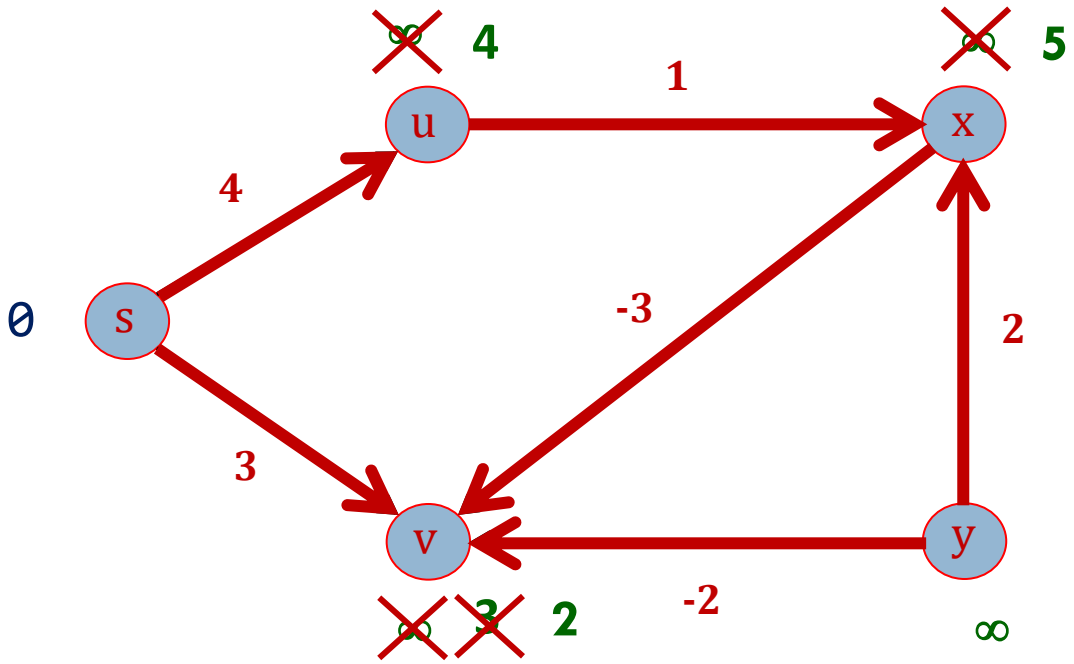
Update στις
6 ακμές
4 φορές

4η Επανάληψη

Καμία Αλλαγή !
Γιατί ?

S1: (y,x), (u,x), (y,v), (s,u), (x,v), (s,v)

Παράδειγμα 2



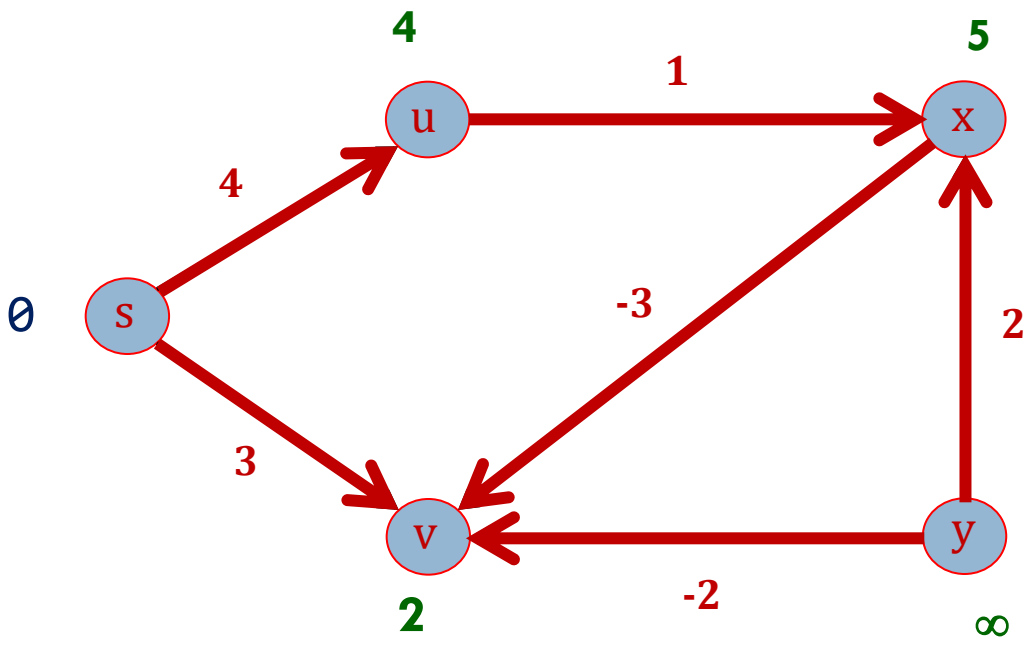
n = 5
m = 6

Update στις
6 ακμές
4 φορές

1η Επανάληψη

S2: (s,u), (u,x), (y,x), (s,v), (x,v), (y,v)

Παράδειγμα 2



n = 5
m = 6

Update στις
6 ακμές
4 φορές

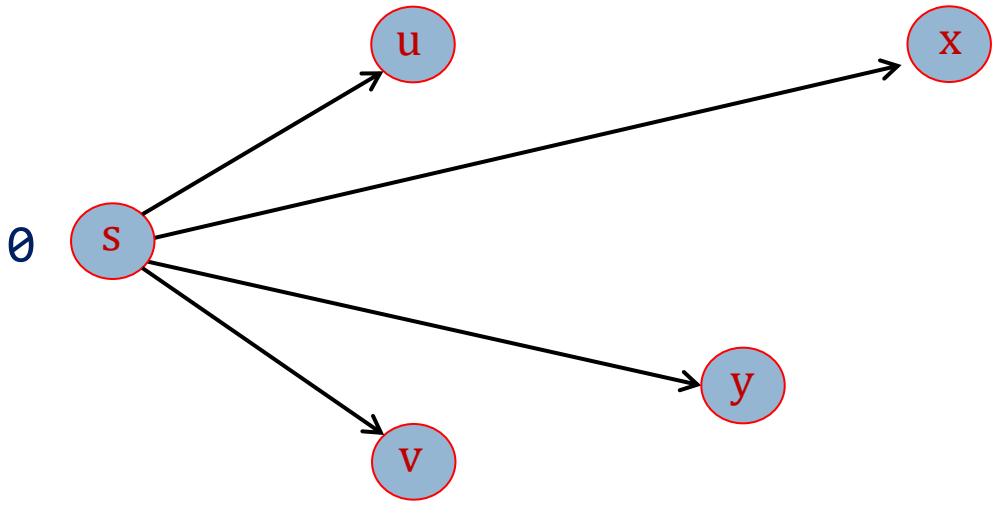
2η Επανάληψη

Καμία Αλλαγή !
Γιατί ?

S2: (s,u), (u,x), (y,x), (s,v), (x,v), (y,v)

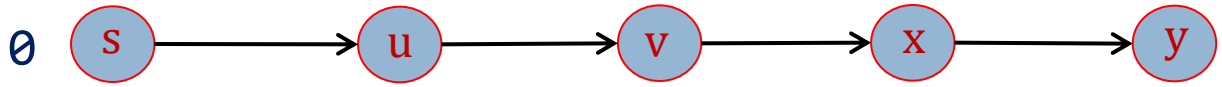
Αλγόριθμος Bellman-Ford

Παρατήρηση !!!



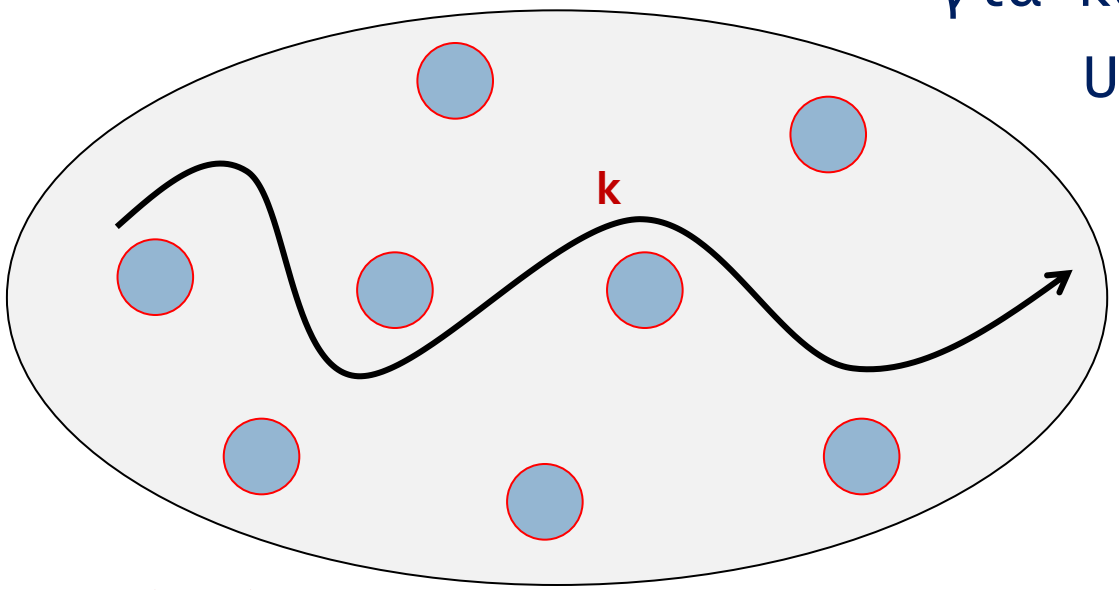
n = 5
m = 6

Update στις
6 ακμές
4 φορές



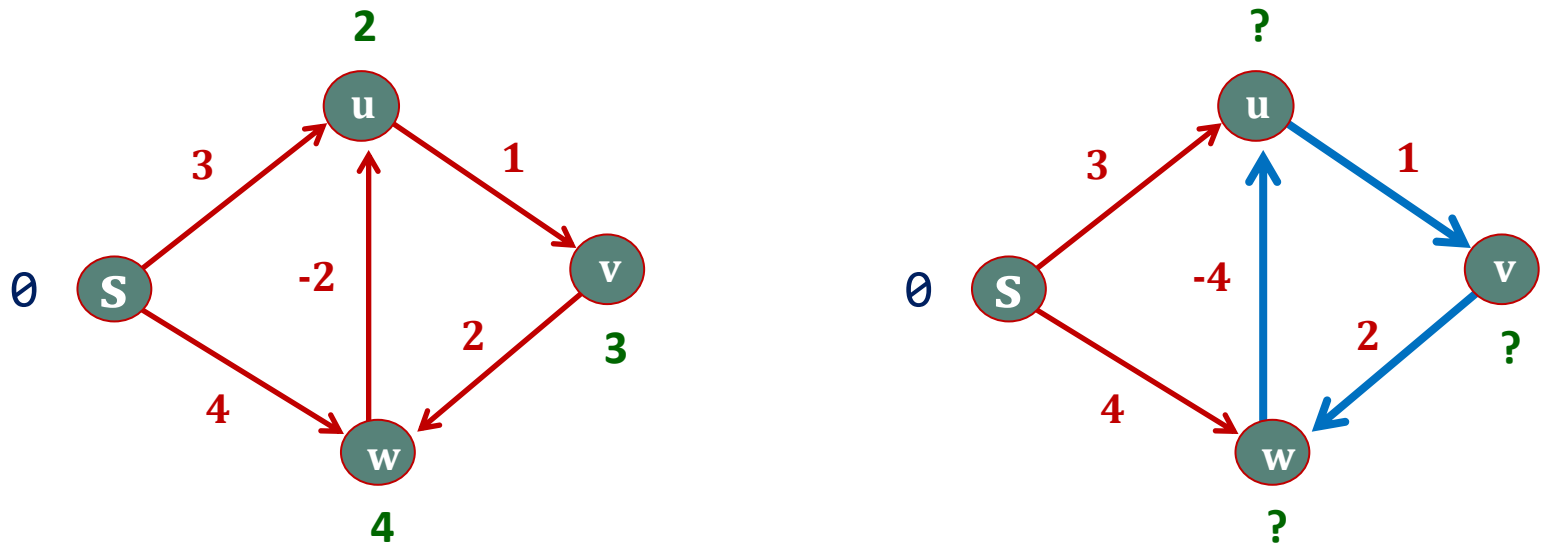
Παρατήρηση !!!

2. επανάλαβε n-1 φορές:
   για κάθε ακμή (u,v)∈E:
   Update(u,v,w)



Διάμετρος k

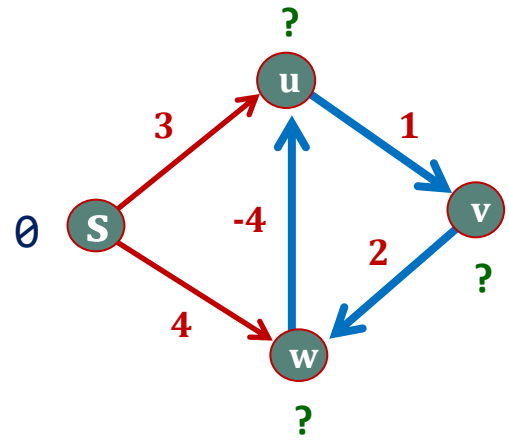
Αρνητικοί Κύκλοι



Εάν υπάρχει αρνητικός κύκλος δεν έχει νόημα να αναζητούμε ελάχιστες διαδρομές !!!

Αρνητικοί Κύκλοι

```
Bellman-Ford(G,w,s)
1.  Initialize(G,s)
2.  επανάλαβε n-1 φορές:
        για κάθε ακμή (u,v)∈E:
            Update(u,v,w)
3.  για κάθε ακμή (u,v)∈E:
        if d(v) > d(u) + w(u,v) then
            return FALSE
4.  return d(.)
```



Πολυπλοκότητα Bellman-Ford

Βήμα 1: Αρχικές τιμές: **O(n)**, n=|V|

Βήμα 2: Εκτελείται n-1 φορές και γίνεται Update() σε m ακμές κάθε φορά.

Επομένως, η πολυπλοκότητα του βήματος 2 είναι **O(n·m)**

Πολυπλοκότητα Bellman-Ford

Βήμα 3: Ελέγχεται m φορές η συνθήκη:

```
d(v) > d(u) + w(u,v)
```

Επομένως, η πολυπλοκότητα του βήματος 3 είναι **O(n·m)**

Συνολική Πολυπλοκότητα: **O(n·m)**

Παρατήρηση !!!

Dijkstra

```
u ← extract-min(Q);
για κάθε ακμή (u,v)∈E:
        Update(u,v,w)
```

Bellman-Ford

```
επανάλαβε n-1 φορές:
    για κάθε ακμή (u,v)∈E:
        Update(u,v,w)
```

```
if d(v) > d(u) + w(u,v) then
      d(v) ← d(u) + w(u,v)
```

Παρατήρηση !!!

Dijkstra

```
u ← extract-min(Q);
για κάθε ακμή (u,v)∈E:
        Update(u,v,w)
```

Bellman-Ford

```
επανάλαβε n-1 φορές:
    για κάθε ακμή (u,v)∈E:
        Update(u,v,w)
```

```
if d(v) > d(u) + w(u,v) then
      d(v) ← d(u) + w(u,v)
      prev(v) ← u
```

Παρατήρηση !!!

Dijkstra

```
u ← extract-min(Q);
για κάθε ακμή (u,v)∈E:
        Update(u,v,w)
```

Bellman-Ford

```
επανάλαβε n-1 φορές:
      για κάθε ακμή (u,v)∈E:
            Update(u,v,w)
```

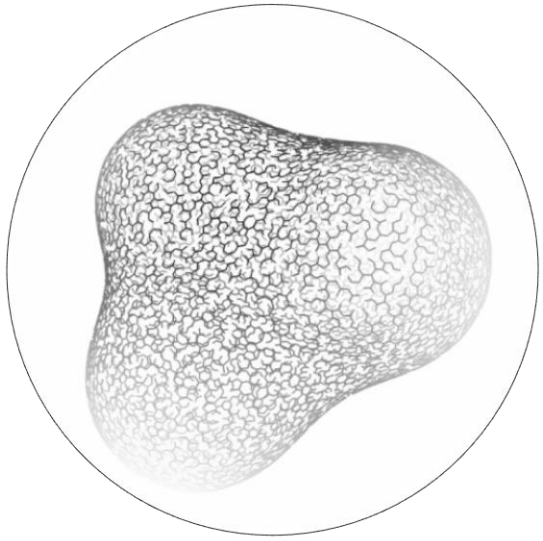
```
if d(v) > d(u) + w(u,v) then
      d(v) ← d(u) + w(u,v)
       prev(v) ← u
      f(v) ← characteristic(u)
```

Ελάχιστες Διαδρομές – Παρατηρήσεις

- **Μπορούμε να υπολογίσουμε Ελάχιστες Διαδρομές σε ένα γράφημα χρησιμοποιώντας**

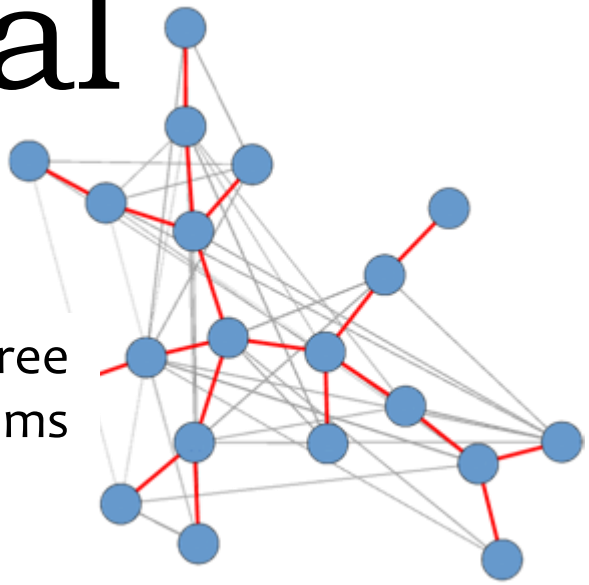
Δυναμικό Προγραμματισμό

Ελάχιστα Συνδετικά Δέντρα

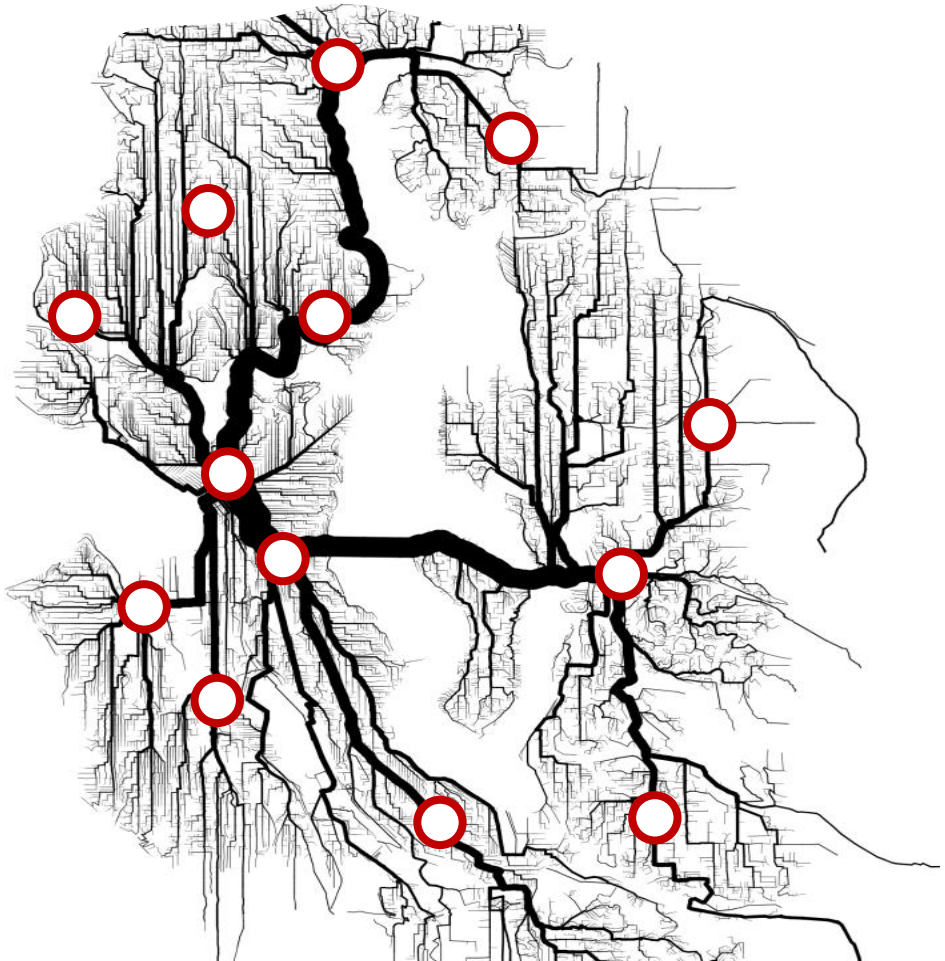


kruskal
prim

minimum spanning tree
algorithms

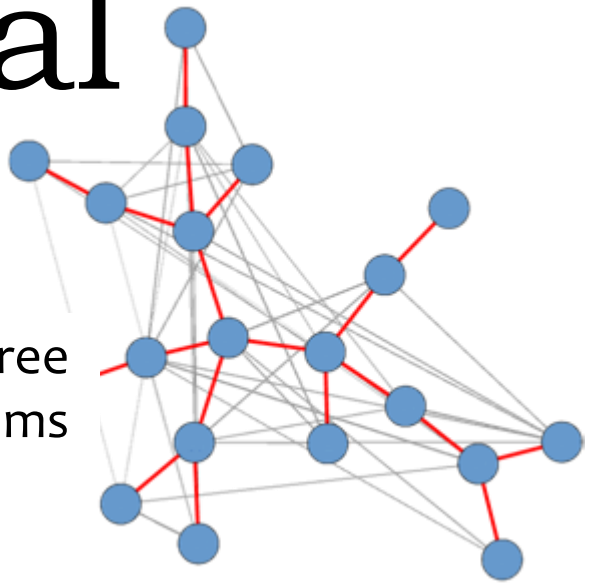


Ελάχιστα Συνδετικά Δέντρα



kruskal
prim

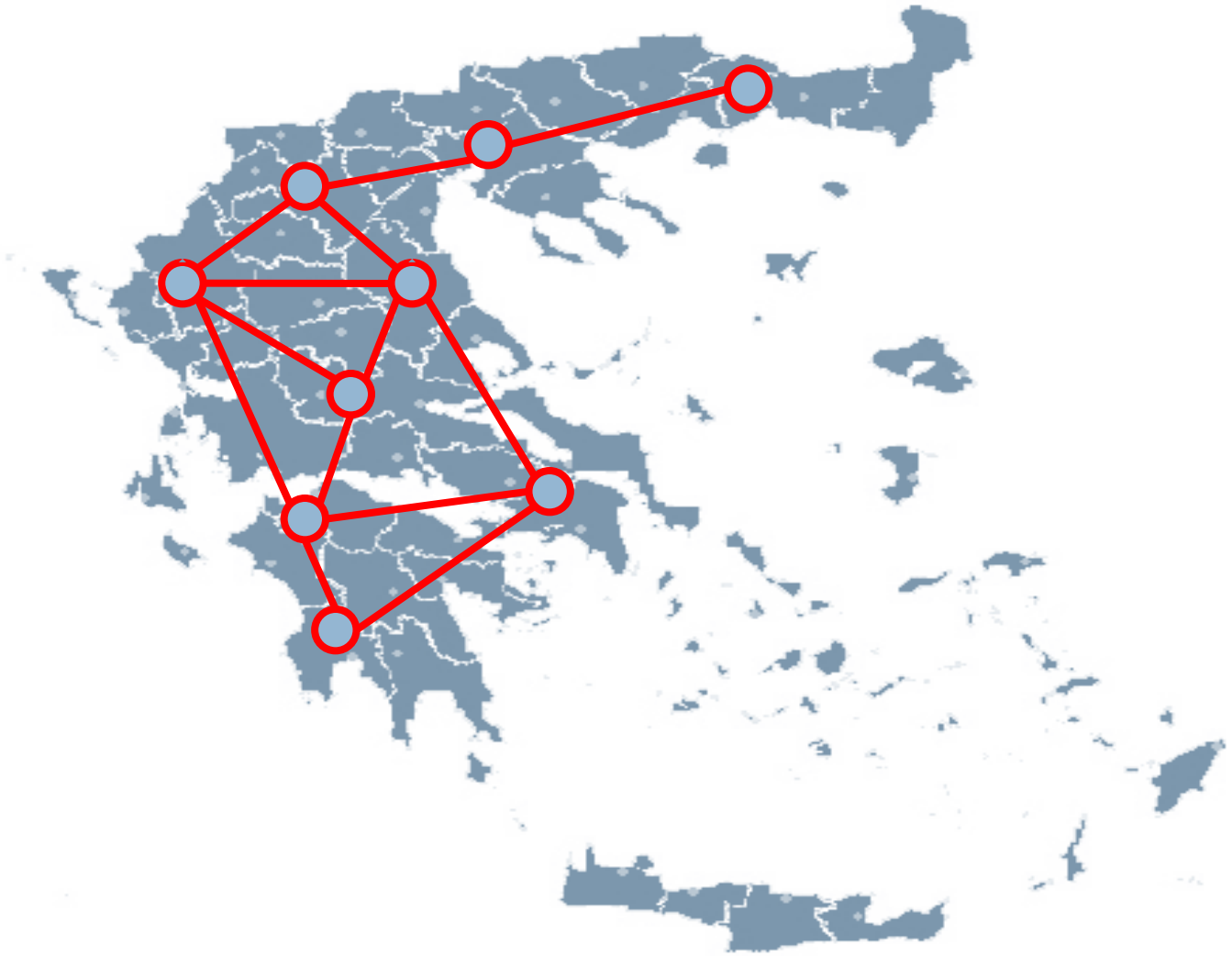
minimum spanning tree
algorithms



Πρόβλημα

Οδικό Δίκτυο

Πρόβλημα

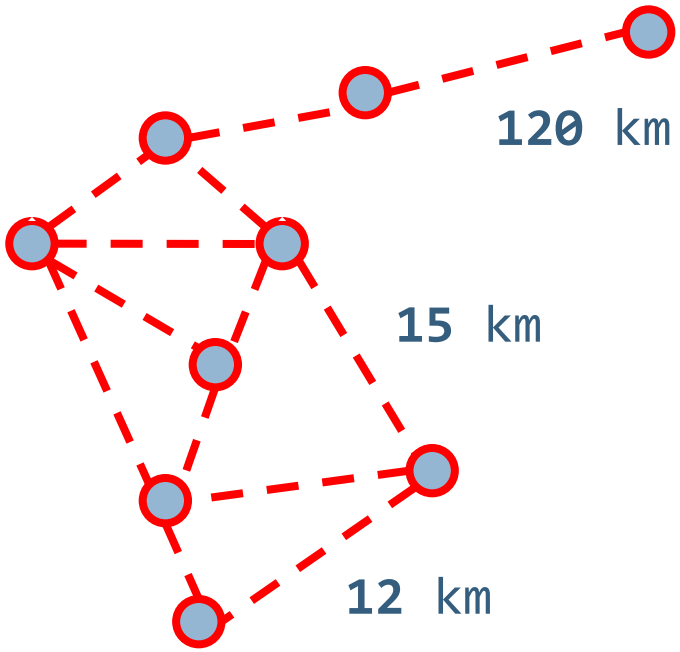


Οδικό Δίκτυο

Πρόβλημα

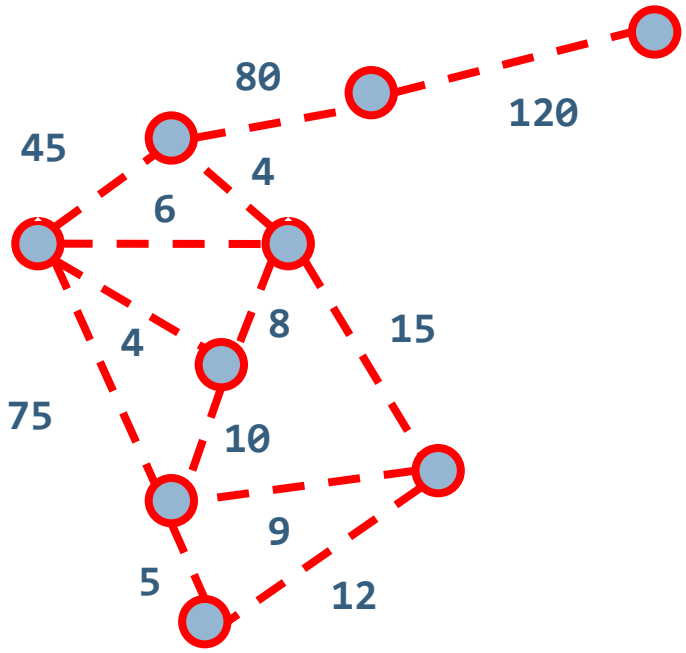
Καταστροφή Δικτύου

Πρόβλημα



Ανακατασκευή Δικτύου

Πρόβλημα



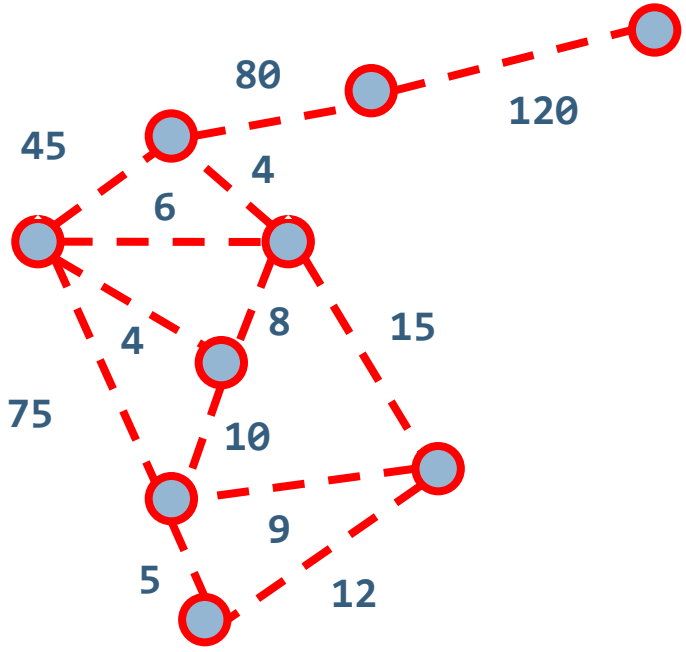
Ανακατασκευή Δικτύου

Κόστος ανά χιλιόμετρο 20.000 ευρώ

✓ **Ολική Επικοινωνία** μεταξύ των Πόλεων

✓ **Ελάχιστο Κόστος** ανακατασκευής

Πρόβλημα



Ανακατασκευή Δικτύου

Κόστος ανά χιλιόμετρο 20.000 ευρώ

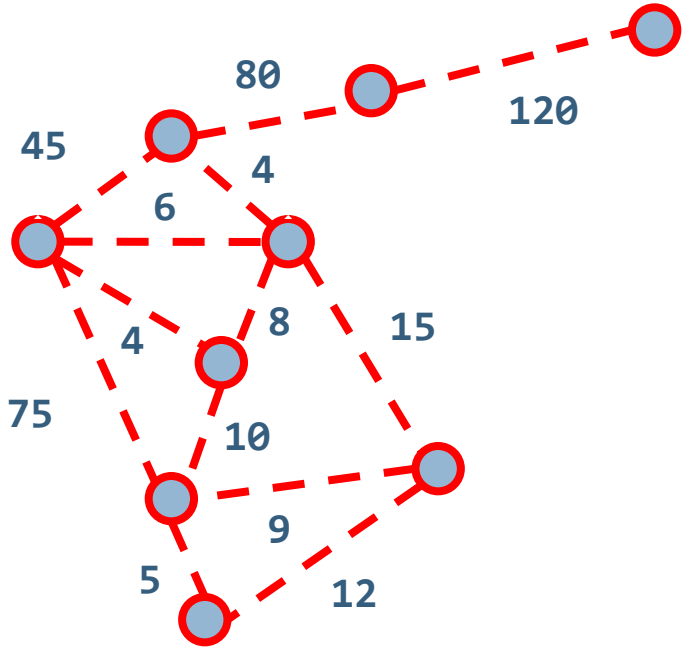
✓ **Ολική Επικοινωνία** μεταξύ των Πόλεων

✓ **Ελάχιστο Κόστος** ανακατασκευής

Ιδιότητα 1

Έστω T οι ακμές του δικτύου οι οποίες δίδουν μια λύση ελάχιστου κόστους για το πρόβλημα αποκατάστασης του δικτύου. Τότε οι ακμές του T παράγουν ένα δένδρο !

Πρόβλημα



Ανακατασκευή Δικτύου

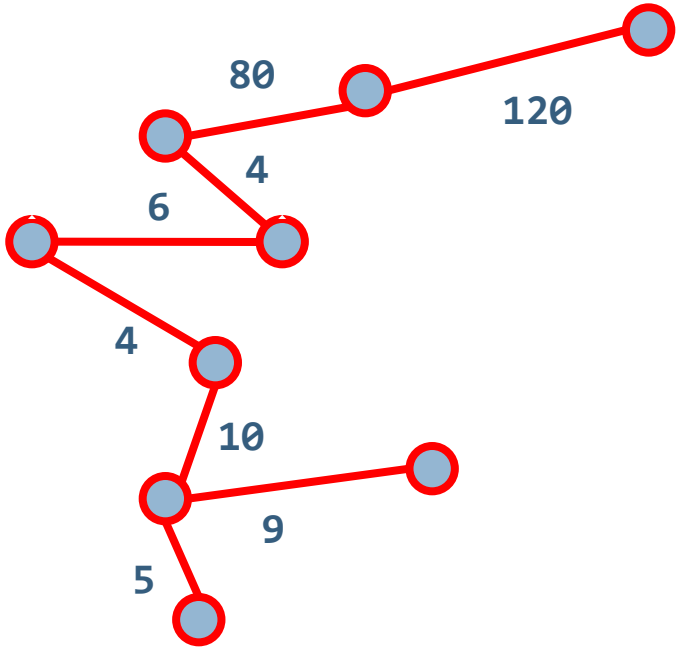
Κόστος ανά χιλιόμετρο 20.000 ευρώ

Επομένως,
η λύση πρέπει να είναι **συνεκτική** και **άκυκλη**

Ιδιότητα 1

Έστω T οι ακμές του δικτύου οι οποίες δίδουν μια λύση ελάχιστου κόστους για το πρόβλημα αποκατάστασης του δικτύου. Τότε οι ακμές του T παράγουν ένα δένδρο !

Πρόβλημα



ΔΕΝΔΡΟ

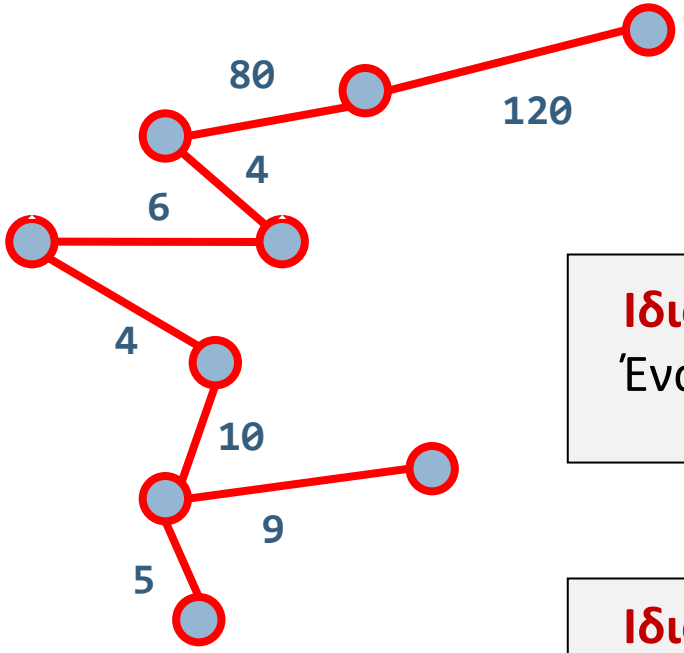
Ανακατασκευή Δικτύου

Κόστος ανά χιλιόμετρο 20.000 ευρώ

Ολική Επικοινωνία
ΝΑΙ

Συνολικό Κόστος για 238 Km
180.229 euro

Ιδιότητες Δένδρων



ΔΕΝΔΡΟ

Ιδιότητα 2
Ένα δένδρο με n κόμβους έχει n-1 ακμές

Ιδιότητα 3
Ένα μη-κατευθυνόμενο συνεκτικό γράφημα με n κόμβους και n-1 ακμές είναι δένδρο

Συνδετικά Δέντρα (Spanning Tree)

- Έστω $G = (V, E)$ ένα *μη-κατευθυνόμενο* γράφημα, *έμβαρο* με *συνάρτηση βάρους* $w : E \rightarrow R$), και έστω

$T = (V, E_T)$ ένα υπογράφημα του G.

✓ Το T ονομάζεται Συνδετικό ή Γενετικό Δέντρο (Spanning Tree) του G εάν είναι συνεκτικό και δεν περιέχει κύκλους.

Συνδετικά Δέντρα (Spanning Tree)

- Έστω $G = (V, E)$ ένα *μη-κατευθυνόμενο* γράφημα, *έμβαρο* με *συνάρτηση βάρους* $w : E \rightarrow R$), και έστω

$T = (V, E_T)$ ένα υπογράφημα του G.

✓ Το T ονομάζεται Συνδετικό ή Γενετικό Δέντρο (Spanning Tree) του G εάν είναι συνεκτικό και δεν περιέχει κύκλους.

✓ Το *βάρος ή κόστος* ενός συνδετικού δένδρου T ορίζεται ως το άθροισμα των βαρών των ακμών του, δηλ.

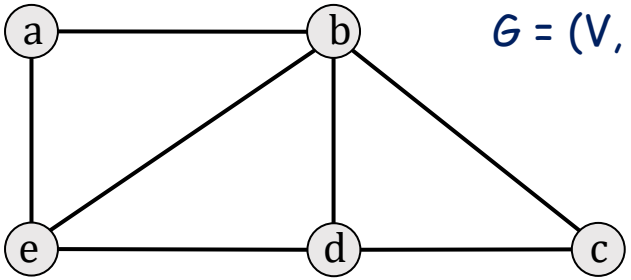
$$w(T) = \sum w(u,v), \ \forall (u,v) \in E_T.$$

Συνδετικά Δέντρα (Spanning Tree)

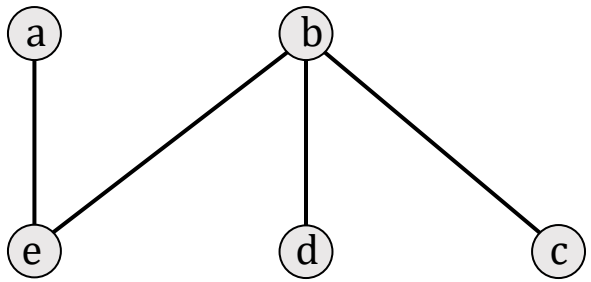
- Παράδειγμα

  Ένα συνεκτικό γράφημα G.

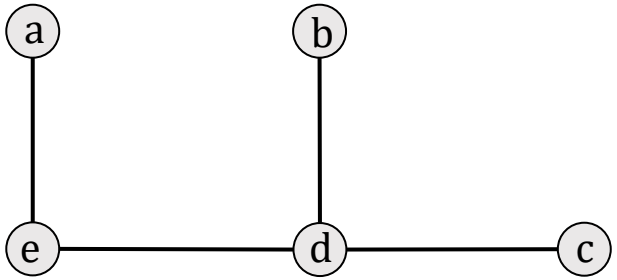
  Δύο συνδετικά δέντρα $T_1 = (V, E_1)$ και $T_2 = (V, E_2)$ του γραφήματος G.



$G = (V, E)$



$T_1 = (V, E_1)$



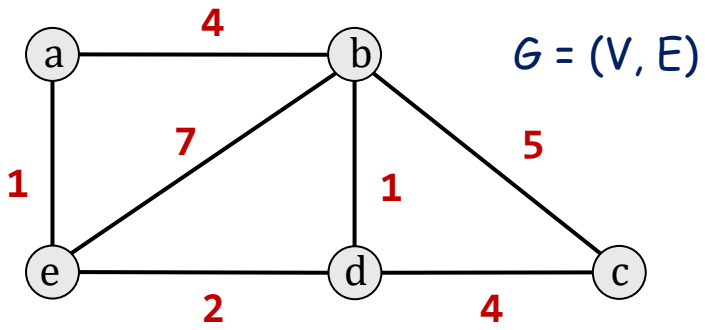
$T_2 = (V, E_2)$

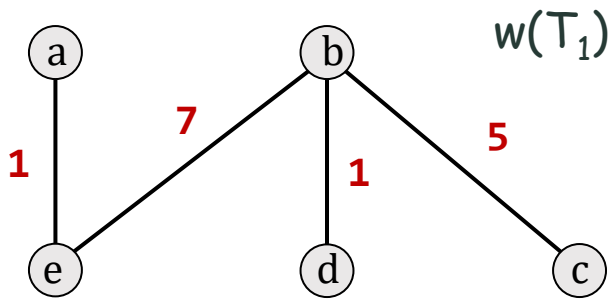
Συνδετικά Δέντρα (Spanning Tree)

- Παράδειγμα

  Ένα έμβαρο συνεκτικό γράφημα G.

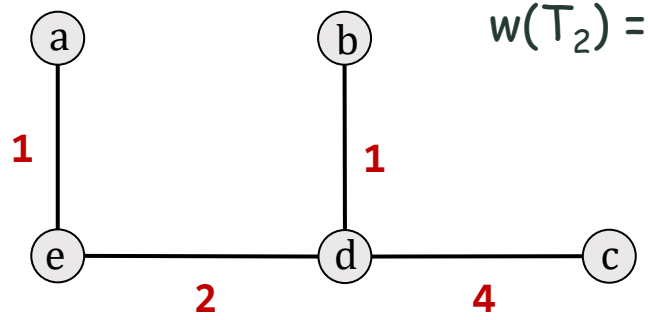
  Δύο συνδετικά δέντρα $T_1 = (V, E_1)$ και $T_2 = (V, E_2)$ του γραφήματος G.





$w(T_1) = 14$

$T_1 = (V, E_1)$



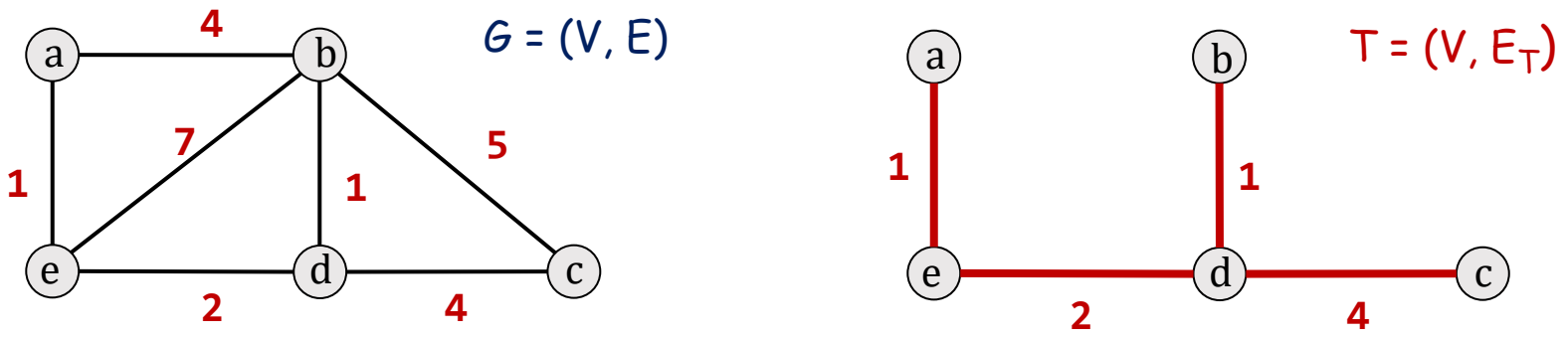
$w(T_2) = 8$

$T_2 = (V, E_2)$

Ελάχιστα Συνδετικά Δέντρα (Minimum Spanning Tree)

Ορισμός

Ένα Ελάχιστο Συνδετικό Δέντρο ενός έμβαρου γραφήματος $G = (V, E)$ είναι ένα συνδετικό δένδρο $T = (V, E_T)$ το οποίο έχει το *ελάχιστο δυνατό κόστος* !!!



Παρατηρήσεις !!!

- ✓ Εάν το *G* δεν είναι πολύ απλό γράφημα, τότε θα έχει εκθετικά πολλά διαφορετικά συνδετικά δένδρα.
- ✓ Επομένως, δεν είναι καθόλου προφανής ο αποδοτικός τρόπος εύρεσης του ελάχιστου δένδρου μεταξύ αυτών των επιλογών !!!
- ✓ Οι αλγόριθμοι που θα παρουσιάσουμε βασίζονται στην Ιδιότητα της Αποκοπής ή Τομή η οποία οδηγεί σε μια αποτελεσματική άπληστη στρατηγική επίλυσης του MST.
- ✓ Θα θεωρήσουμε, για ευκολία, ότι όλες οι ακμές έχουν διαφορετικά βάρη.

Spanning tree : Συνδετικό, Γενετικό, Σκελετικό ή Ζευγνύον δένδρο.

Θεώρημα 1 (Ιδιότητα Αποκοπής – Cut Property)

Έστω $G = (V, E)$ συνεκτικό μη-κατευθυνόμενο έμβαρο γράφημα και έστω:

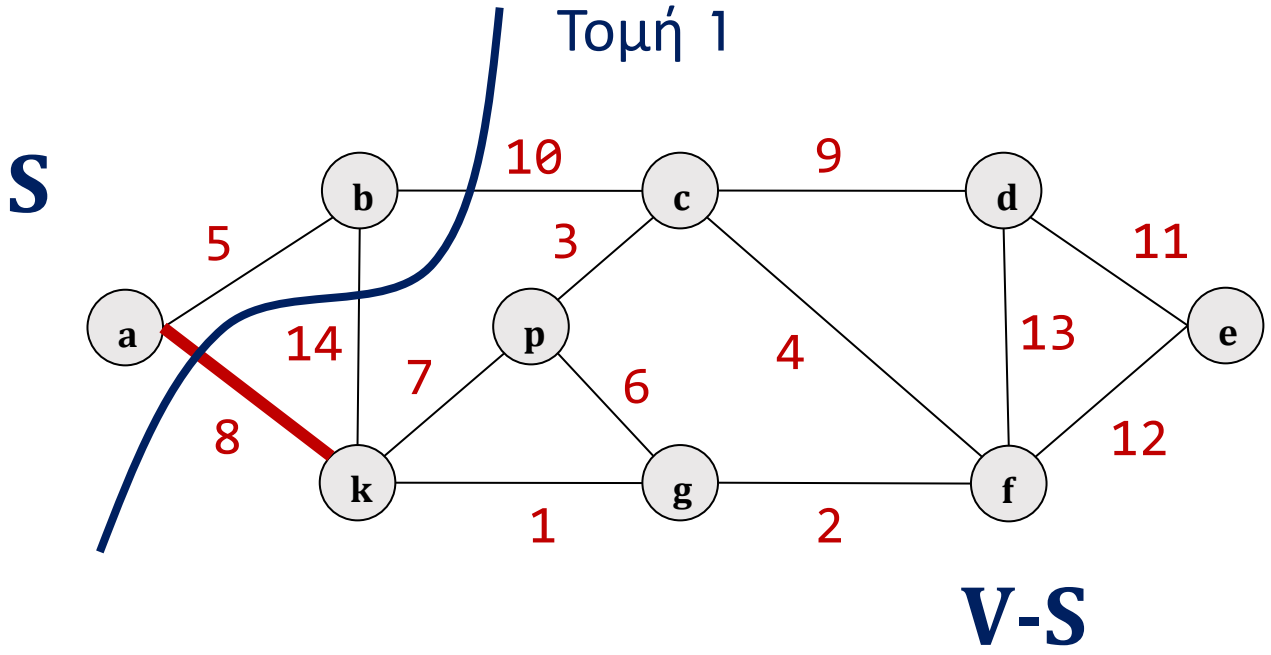
- $(S, V-S)$ μια **τομή** του V (διαμέριση του V), όπου S υποσύνολο κόμβων του G που δεν είναι ούτε κενό ούτε ίσο με το V, και
- $e \in E$ η ακμή με το **min-βάρος** που τέμνει την τομή $(S, V-S)$, δηλ. έχει το ένα άκρο στο σύνολο S και το άλλο στο $V-S$.

Τότε,

Κάθε MST του G περιέχει την ακμή e

Παράδειγμα

Θεώρημα 1

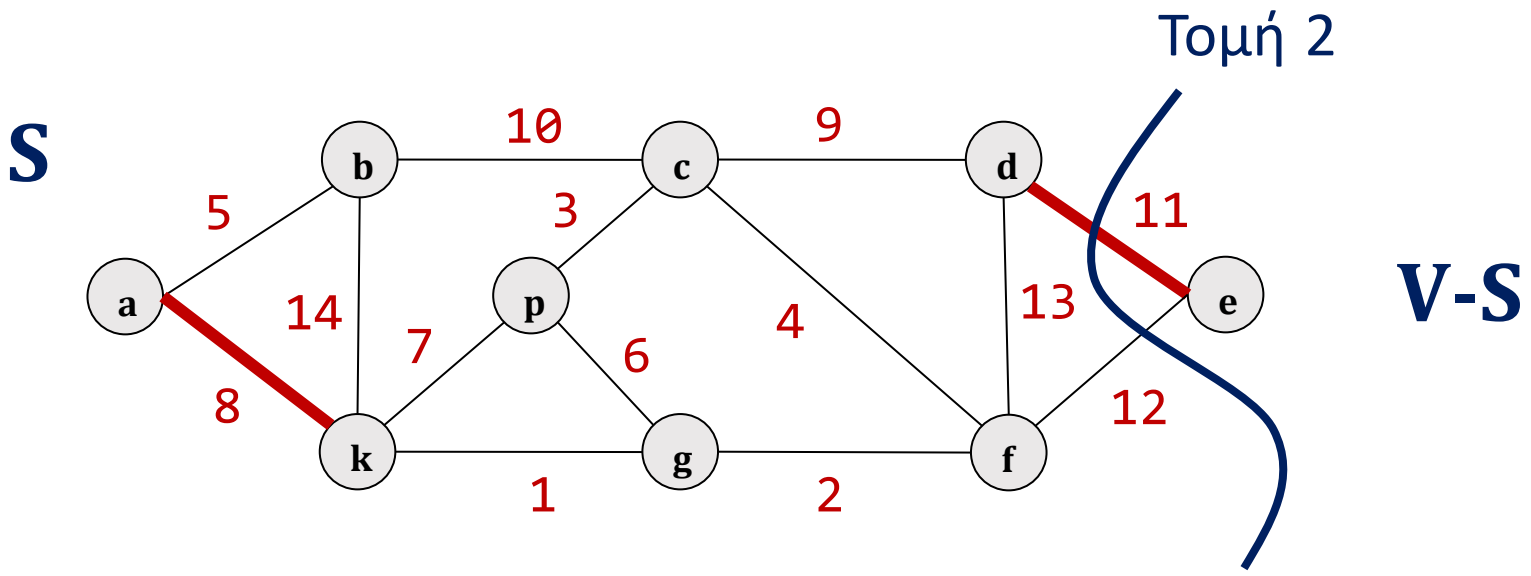


Κάθε MST του G περιέχει την ακμή (a, k)

διότι είναι η ακμή με το min-βάρος του τέμνει την Τομή 1, όπου S = {a, b} και V-S = {k, p, …, e}

Παράδειγμα

Θεώρημα 1

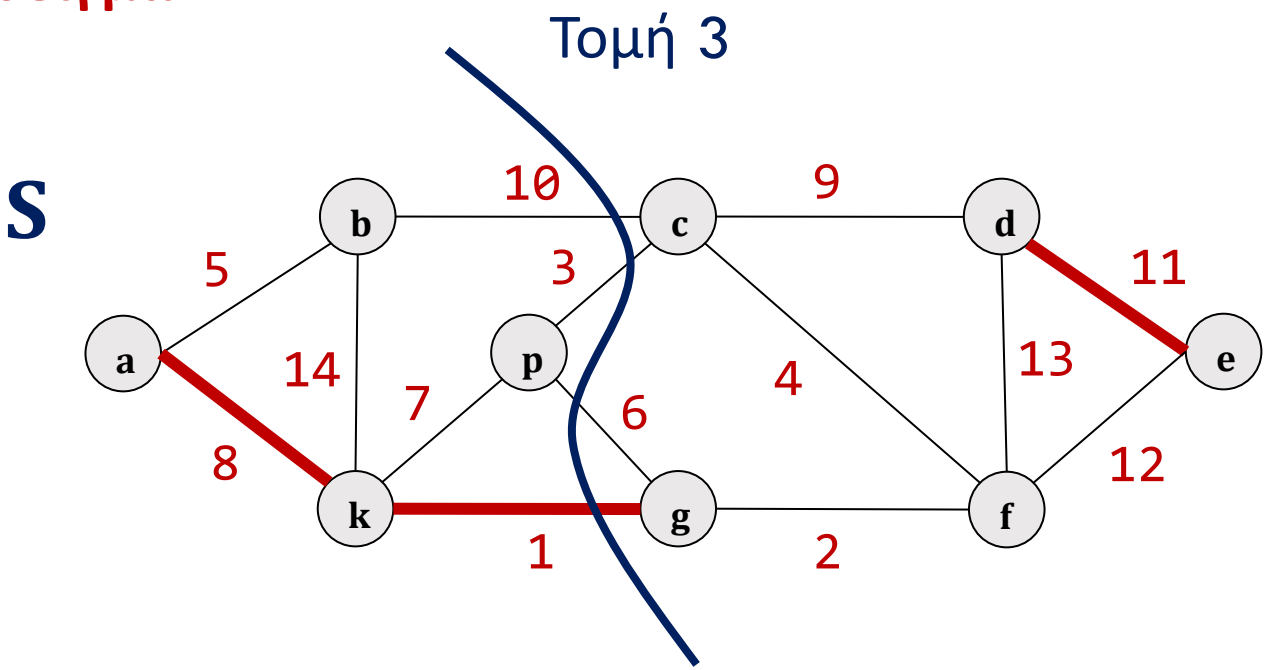


Κάθε MST του G περιέχει την ακμή (d, e)

διότι είναι η ακμή με το min-βάρος του τέμνει την Τομή 2

Παράδειγμα

Θεώρημα 1

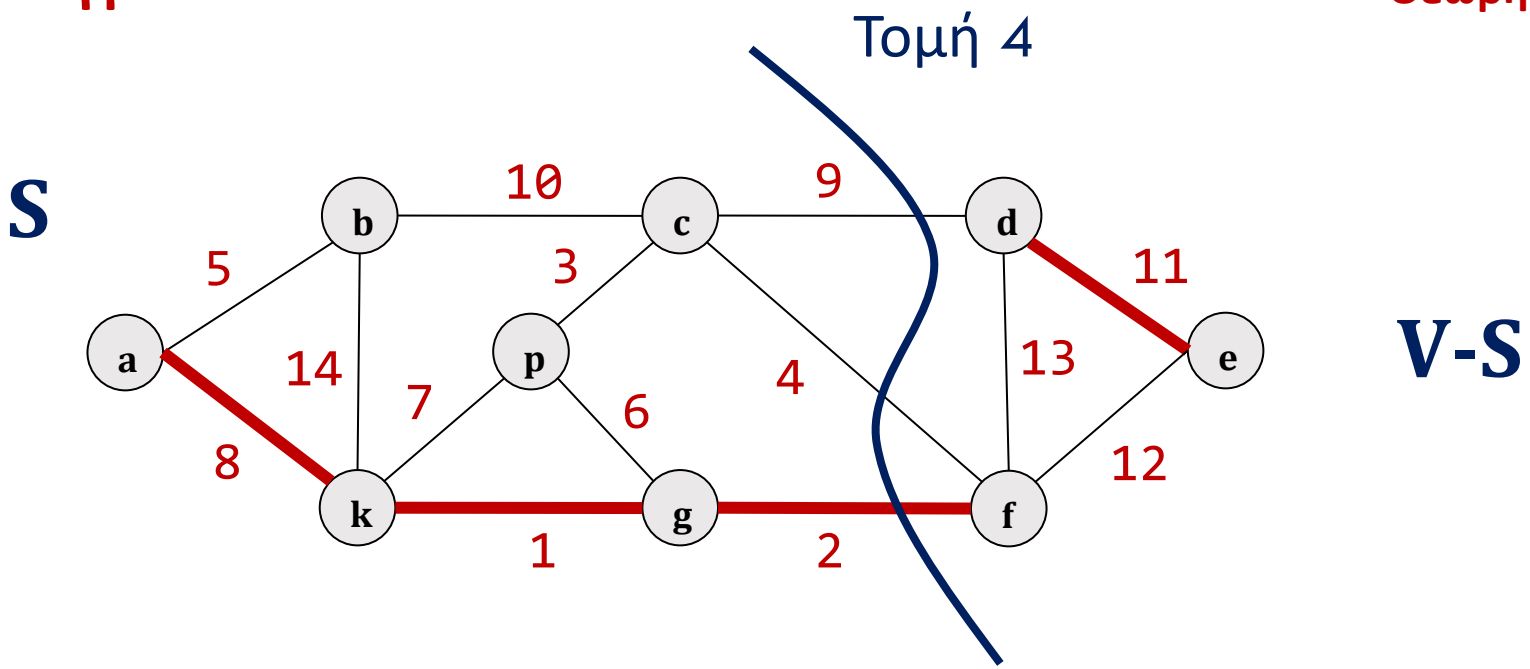


Κάθε MST του G περιέχει την ακμή (k, g)

διότι είναι η ακμή με το min-βάρος του τέμνει την Τομή 3

Παράδειγμα

Θεώρημα 1



Κάθε MST του G περιέχει την ακμή (g, f)

διότι είναι η ακμή με το min-βάρος του τέμνει την Τομή 4

Πόρισμα 1

Έστω $G = (V, E)$ συνεκτικό μη-κατευθυνόμενο έμβαρο γράφημα και έστω:

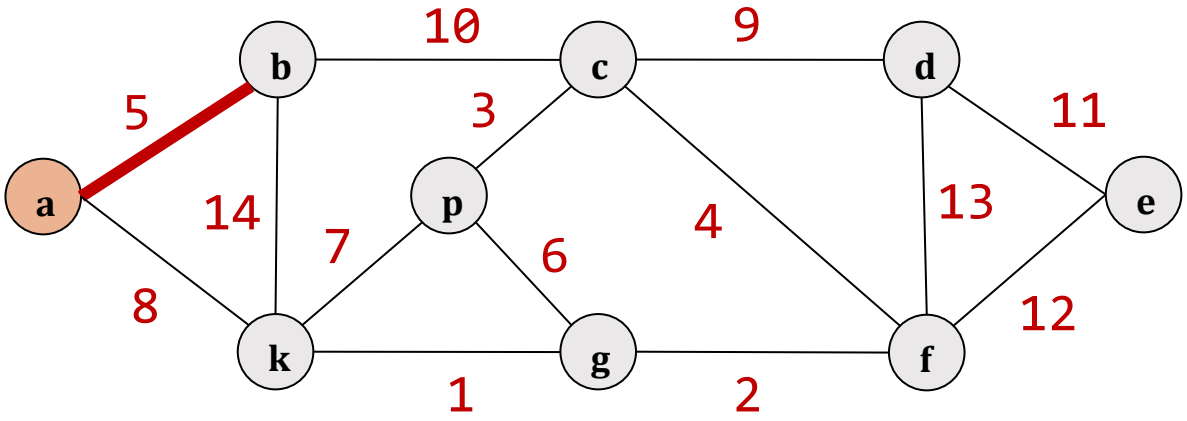
- $c(u) \in V$ ο κόμβος του G τέτοιος ώστε: $(u, c(u)) \in E$ είναι η ακμή με το **min-βάρος** που προσπίπτει στον κόμβο u, $\forall\, u \in V$.

Τότε,

Κάθε MST του G περιέχει την ακμή $(u, c(u))$

Παράδειγμα

Πόρισμα 1

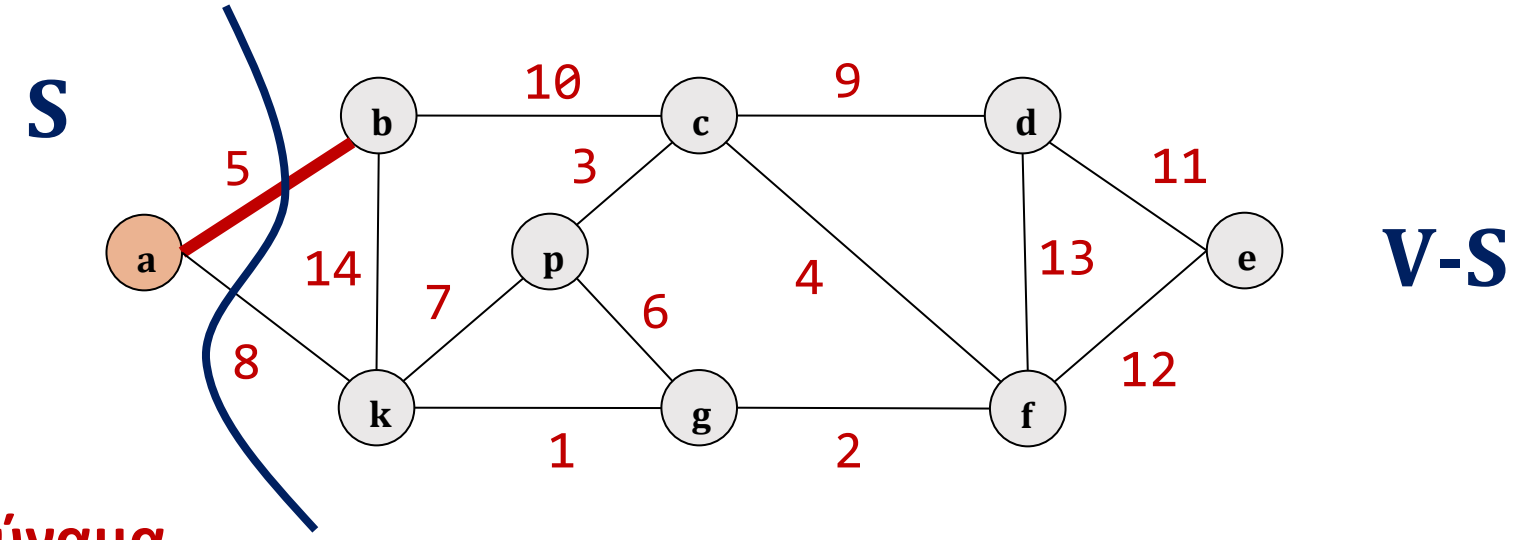


Κάθε MST του G περιέχει την ακμή (a, b)

διότι είναι η ακμή με το min-βάρος που προσπίπτει στον κόμβο a

Παράδειγμα

Πόρισμα 1



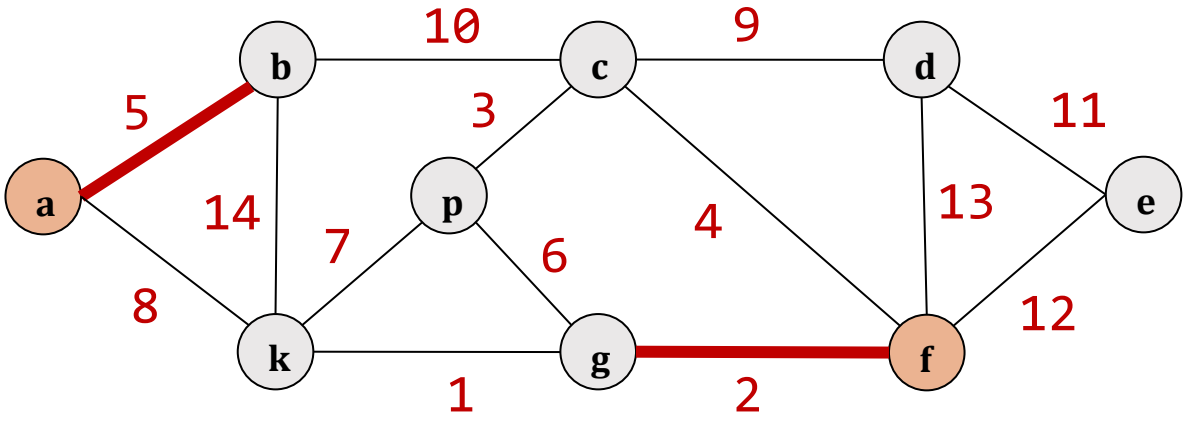
Ισοδύναμα

Κάθε MST του G περιέχει την ακμή (a, b)

διότι είναι η ακμή με το min-βάρος του τέμνει την Τομή με S = {a} και V-S = {b, k, g, …, e}

Παράδειγμα

Πόρισμα 1

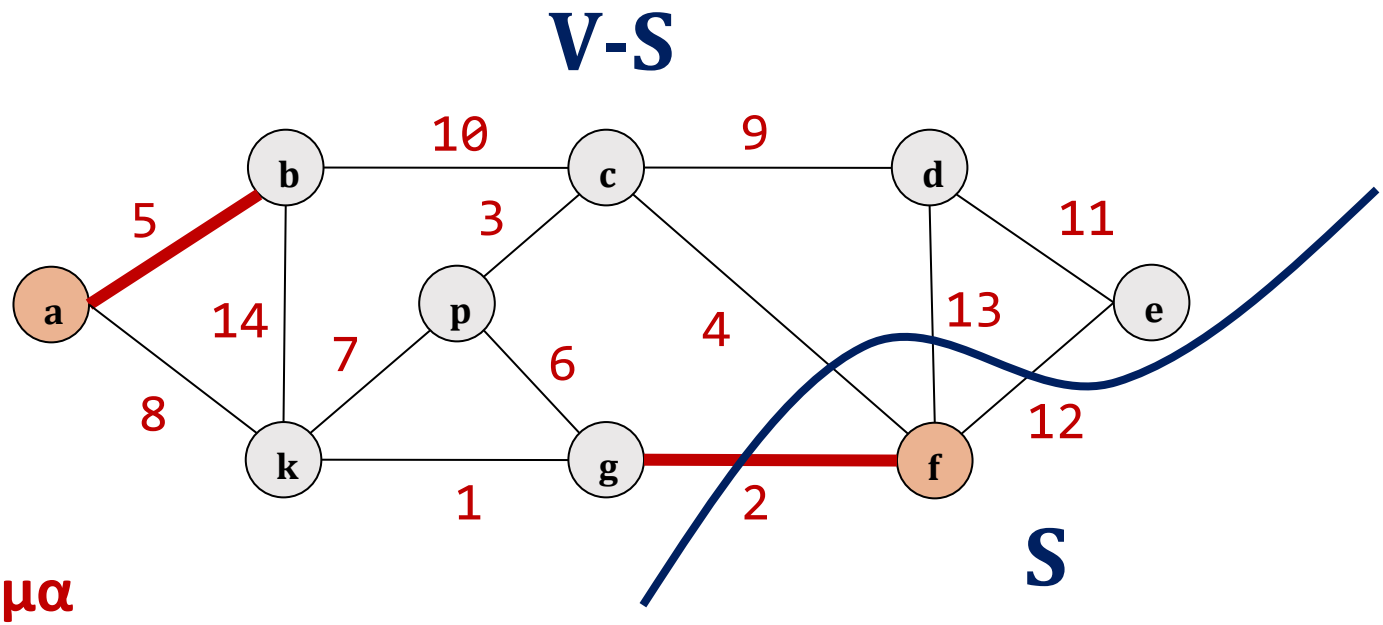


Κάθε MST του G περιέχει την ακμή (f, g)

διότι είναι η ακμή με το min-βάρος που προσπίπτει στον κόμβο f

Παράδειγμα

Πόρισμα 1



Ισοδύναμα

Κάθε MST του G περιέχει την ακμή (f, g)

διότι είναι η ακμή με το min-βάρος του τέμνει την Τομή με S = {f} και V-S = {g, k, p, …, e}

Ελάχιστα Συνδετικά Δέντρα – Γενικός Αλγόριθμος

Αλγόριθμος Generic-MST(G, w)

1. $A \leftarrow \varnothing$

2. Εφόσον το A δεν παράγει ένα MST του G:

> βρες μια ακμή (u,v) "κατάλληλη" για το A
>
> $A \leftarrow A \cup \{(u,v)\}$

3. Επίστρεψε $T \leftarrow A$

Αλγόριθμος Generic-MST(G, w)

Κατά την διάρκεια εκτέλεσης του αλγόριθμου, το A είναι ένα υποσύνολο των ακμών ενός **MST** T του $G=(V,E)$.

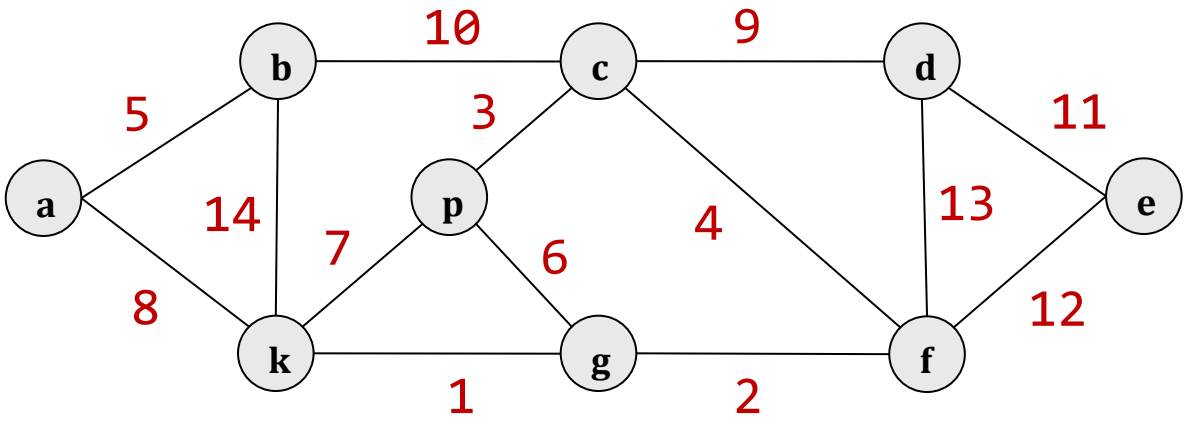
✓ **Κατάλληλη Ακμή :** Η ακμή (u,v) είναι «κατάλληλη» για το A **εάν** το $A \cup \{(u,v)\}$ **παραμένει** υποσύνολο των ακμών ενός MST

Επιλέγουμε ως κατάλληλη ακμή (u,v) να είναι η ακμή min-βάρους μιας Τομή $(S, V-S)$ που «σέβεται» το A,

δηλ. $\forall\ (x,y) \in A \ \Rightarrow\ x, y \in S$ ή $x, y \in V-S$

Παράδειγμα Εκτέλεσης του Αλγόριθμου **Generic-MST(G, w)**

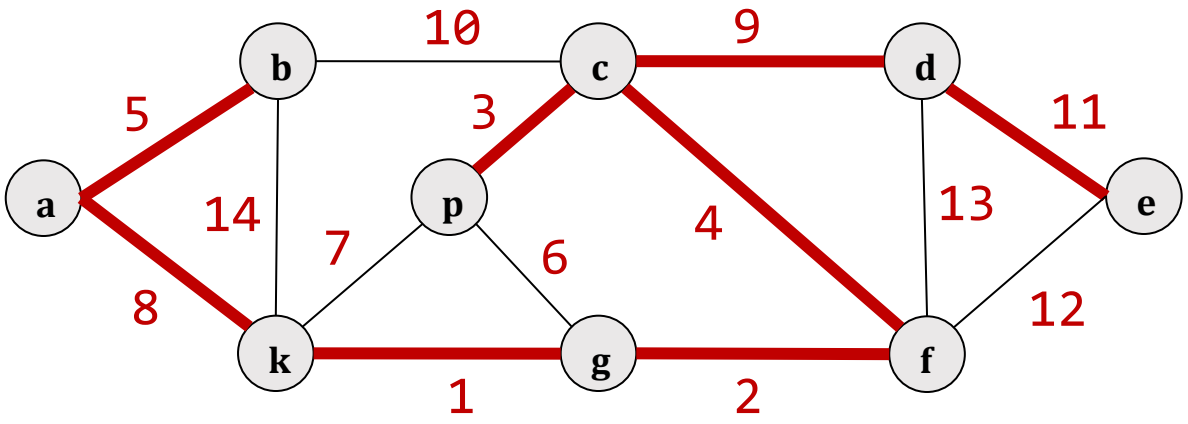
Ένα έμβαρο μη-κατευθυνόμενο γράφημα *G*



Τάξης **n = 9** και μεγέθους **m = 14** !!!

Παράδειγμα Εκτέλεσης του Αλγόριθμου **Generic-MST(G, w)**

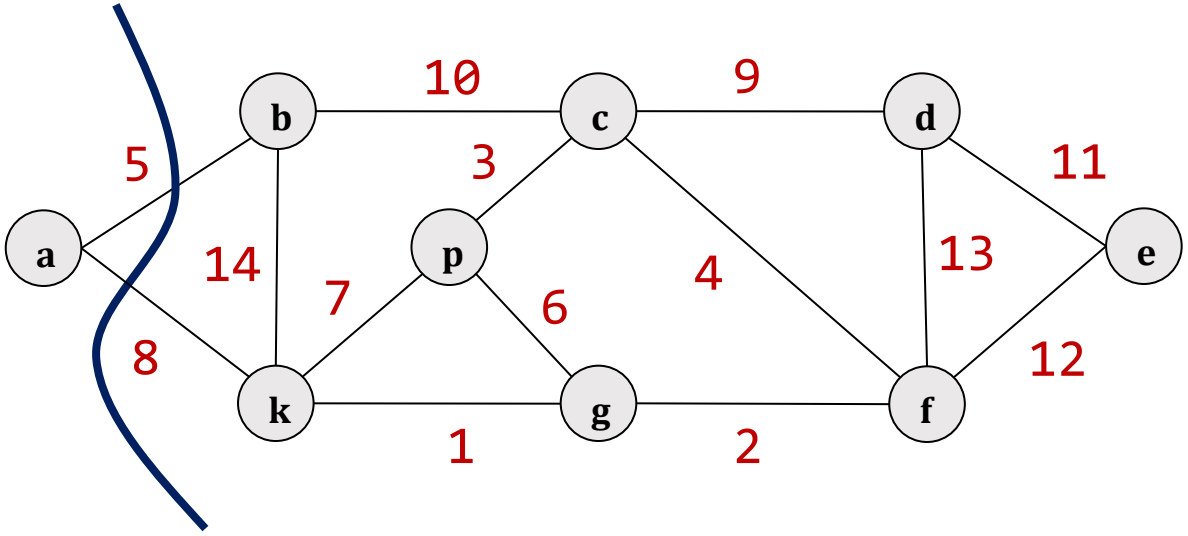
Ένα MST του G



Επειδή όλες οι ακμές έχουν διαφορετικά βάρη το MST είναι μοναδικό !!!

Παράδειγμα Εκτέλεσης του Αλγόριθμου **Generic-MST(G, w)**

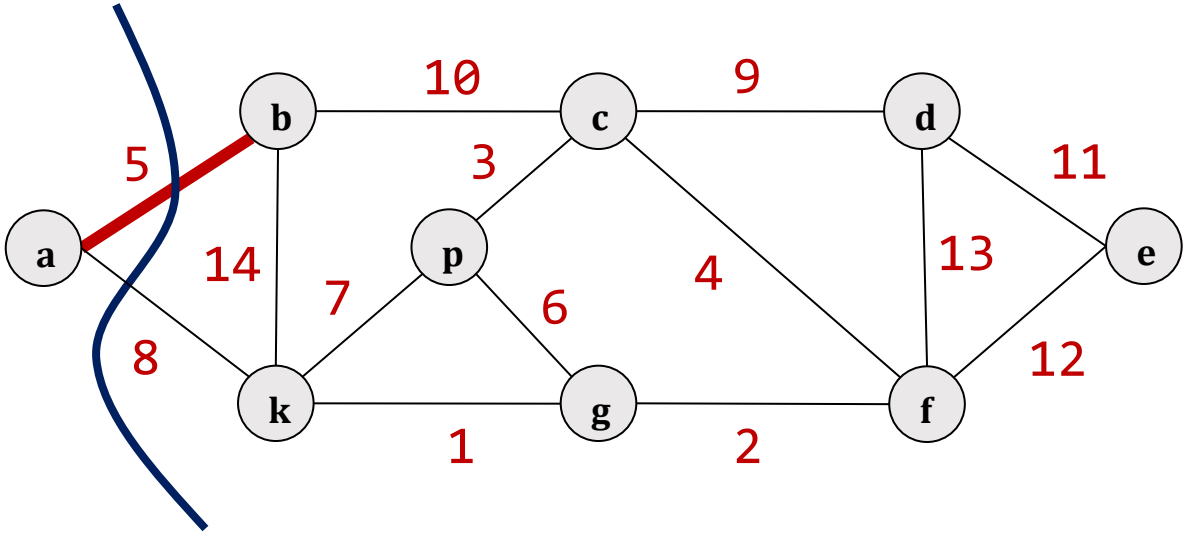
Επανάληψη 1



Επειδή όλες οι ακμές έχουν διαφορετικά βάρη το MST είναι μοναδικό !!!

Παράδειγμα Εκτέλεσης του Αλγόριθμου **Generic-MST(G, w)**

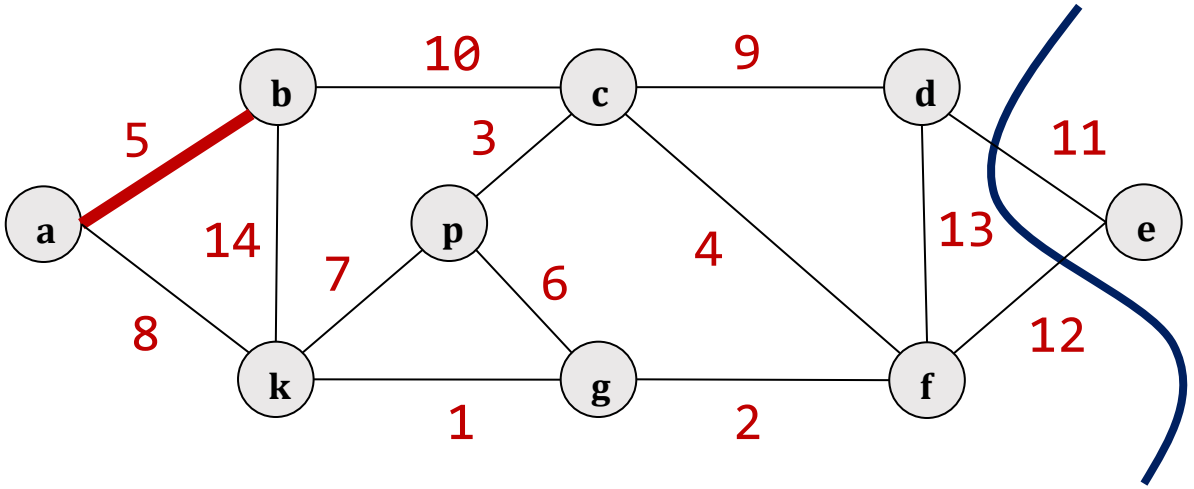
Επανάληψη 1



Επειδή όλες οι ακμές έχουν διαφορετικά βάρη το MST είναι μοναδικό !!!

Παράδειγμα Εκτέλεσης του Αλγόριθμου **Generic-MST(G, w)**

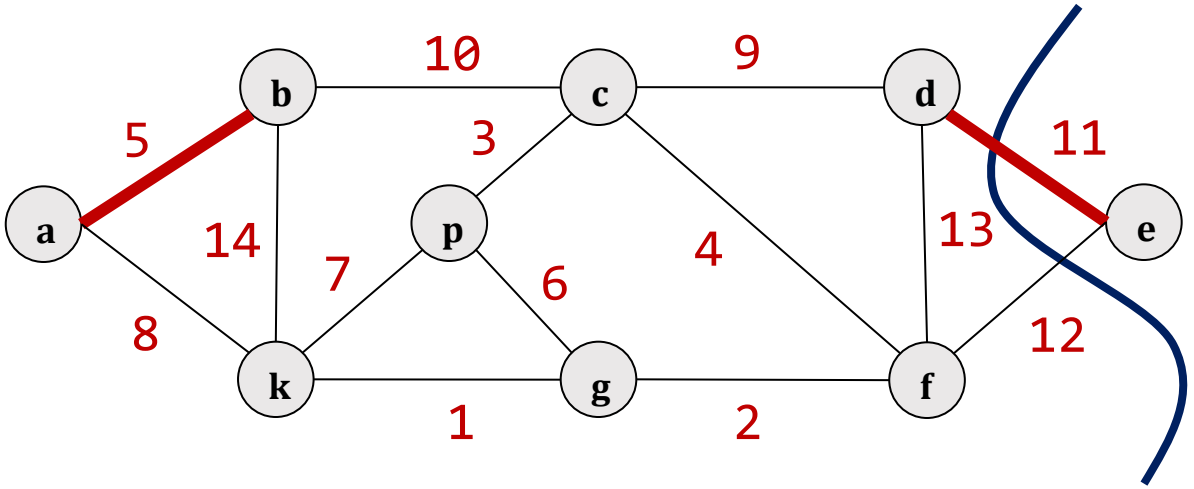
Επανάληψη 2



Επειδή όλες οι ακμές έχουν διαφορετικά βάρη το MST είναι μοναδικό !!!

Παράδειγμα Εκτέλεσης του Αλγόριθμου **Generic-MST(G, w)**

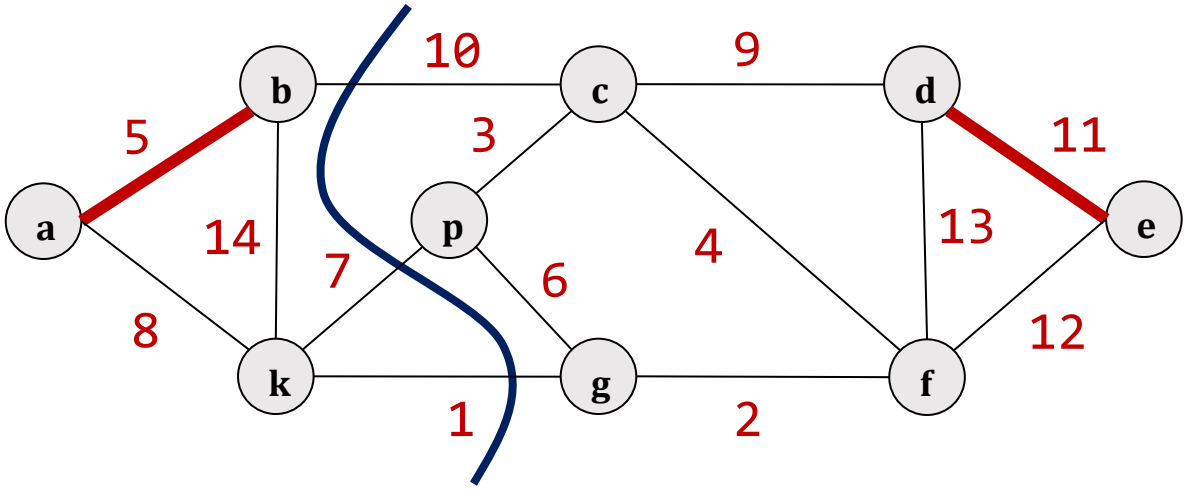
Επανάληψη 2



Επειδή όλες οι ακμές έχουν διαφορετικά βάρη το MST είναι μοναδικό !!!

Παράδειγμα Εκτέλεσης του Αλγόριθμου **Generic-MST(G, w)**

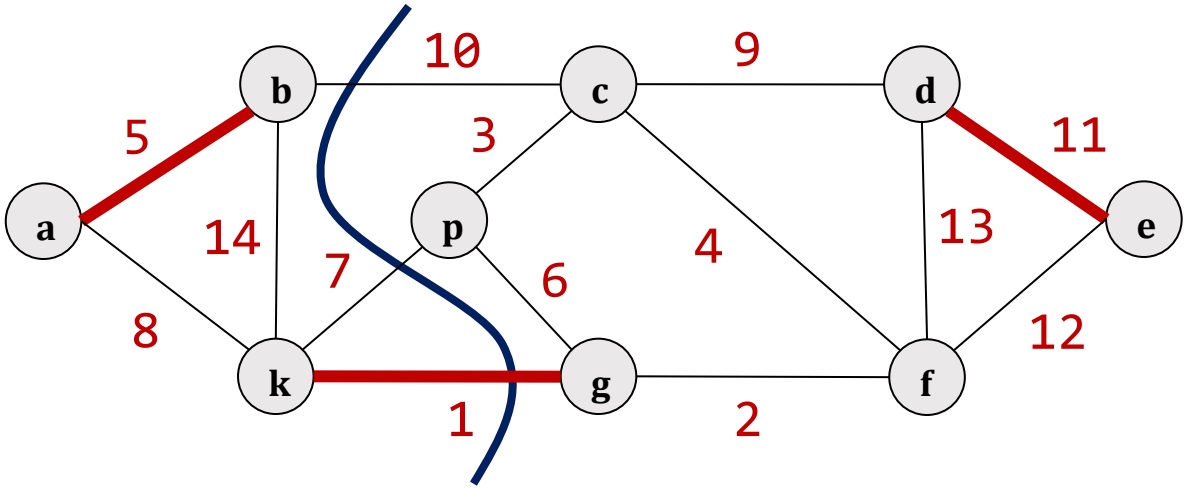
Επανάληψη 3



Επειδή όλες οι ακμές έχουν διαφορετικά βάρη το MST είναι μοναδικό !!!

Παράδειγμα Εκτέλεσης του Αλγόριθμου **Generic-MST(G, w)**

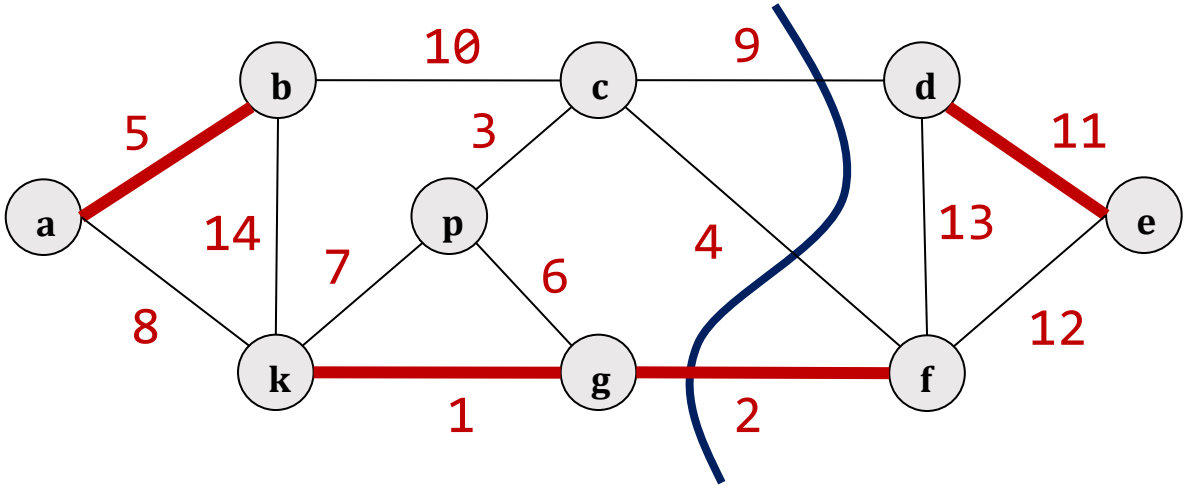
Επανάληψη 3



Επειδή όλες οι ακμές έχουν διαφορετικά βάρη το MST είναι μοναδικό !!!

Παράδειγμα Εκτέλεσης του Αλγόριθμου **Generic-MST(G, w)**

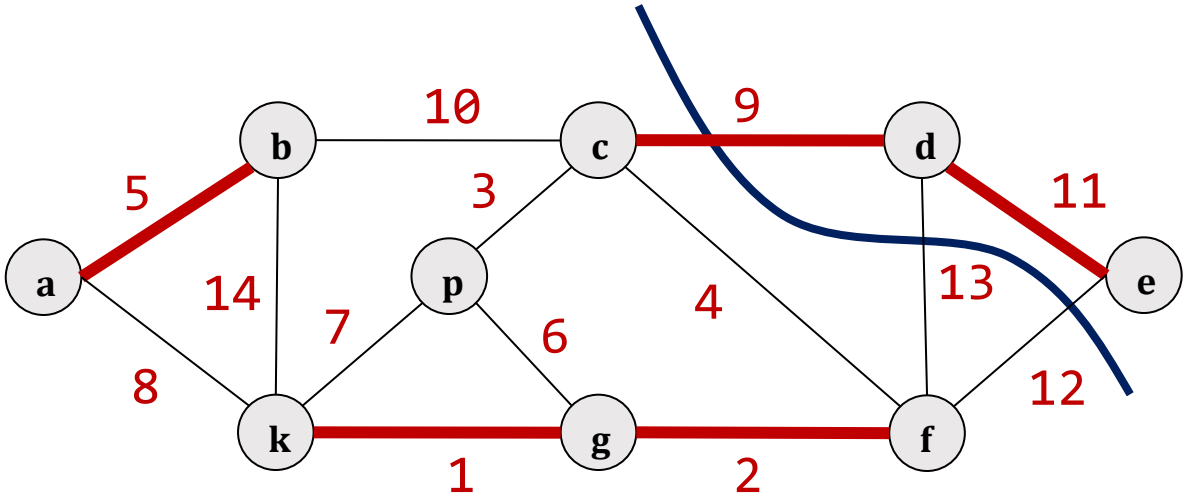
Επανάληψη 4



Επειδή όλες οι ακμές έχουν διαφορετικά βάρη το MST είναι μοναδικό !!!

- Παράδειγμα Εκτέλεσης του Αλγόριθμου **Generic-MST(G, w)**

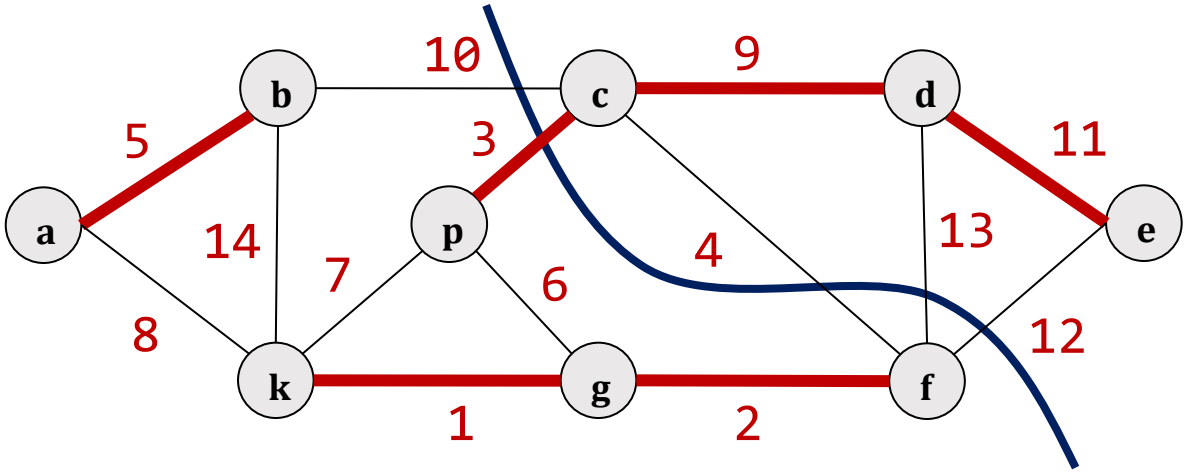
Επανάληψη 5



Επειδή όλες οι ακμές έχουν διαφορετικά βάρη το MST είναι μοναδικό !!!

Παράδειγμα Εκτέλεσης του Αλγόριθμου **Generic-MST(G, w)**

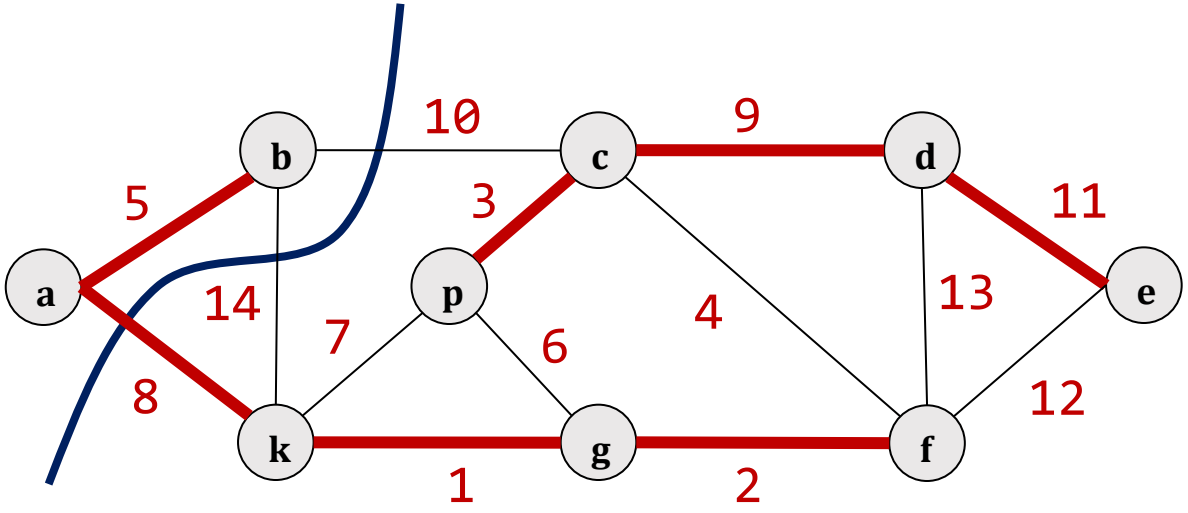
Επανάληψη 6



Επειδή όλες οι ακμές έχουν διαφορετικά βάρη το MST είναι μοναδικό !!!

- Παράδειγμα Εκτέλεσης του Αλγόριθμου **Generic-MST(G, w)**

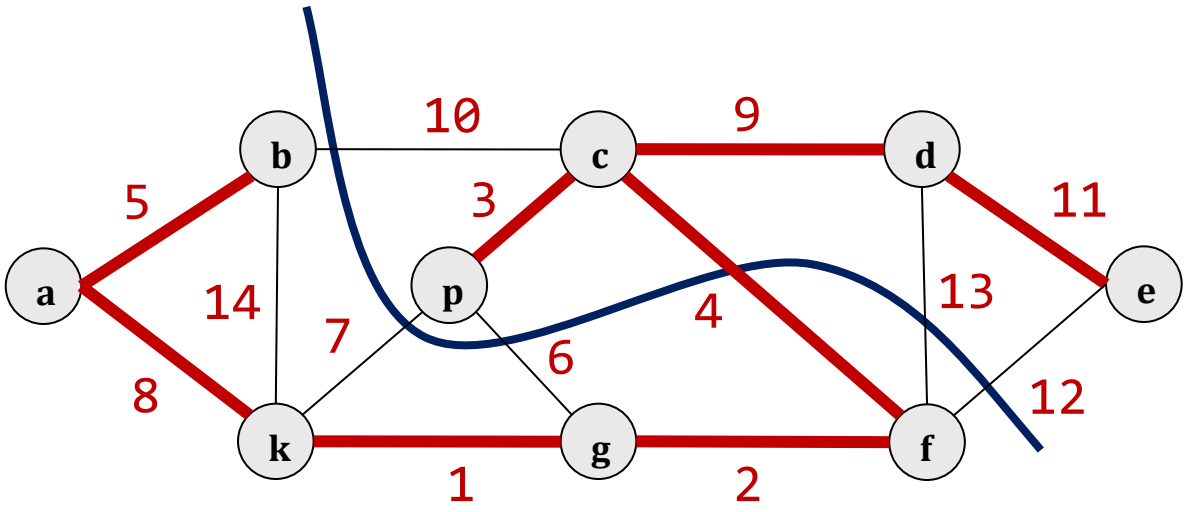
Επανάληψη 7



Επειδή όλες οι ακμές έχουν διαφορετικά βάρη το MST είναι μοναδικό !!!

Παράδειγμα Εκτέλεσης του Αλγόριθμου **Generic-MST(G, w)**

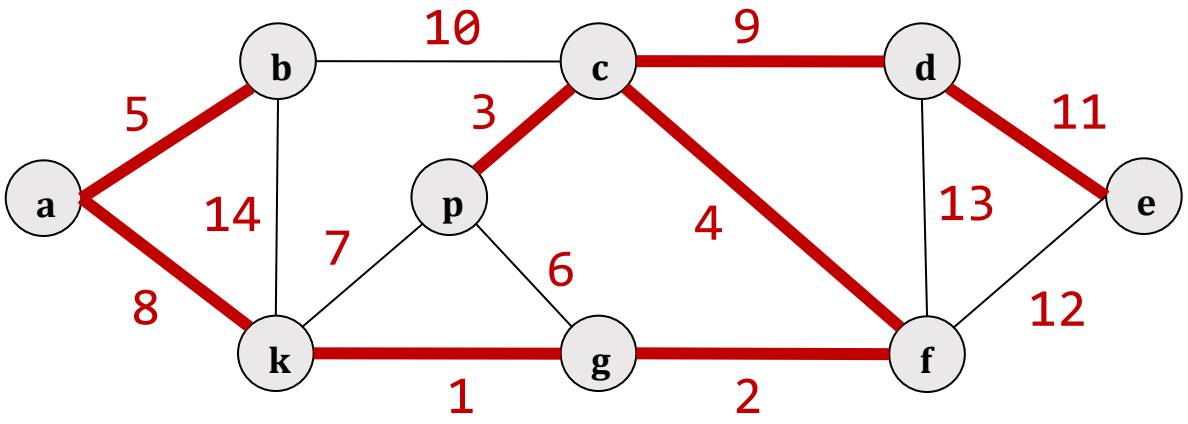
Επανάληψη 8



Επειδή όλες οι ακμές έχουν διαφορετικά βάρη το MST είναι μοναδικό !!!

Παράδειγμα Εκτέλεσης του Αλγόριθμου **Generic-MST(G, w)**

Έξοδος αλγόριθμου T



Επειδή όλες οι ακμές έχουν διαφορετικά βάρη το MST είναι μοναδικό !!!

Αλγόριθμοι Kruskal & Prim

- Θα παρουσιάσουμε δύο γνωστούς αλγορίθμους για το πρόβλημα υπολογισμού ενός MST ενός έμβαρου G = (V, E).

1. **Αλγόριθμος Kruskal**
2. **Αλγόριθμος Prim**

Άπληστοι αλγόριθμοι

Βασική Ιδέα!!!

1. Αλγόριθμος Kruskal

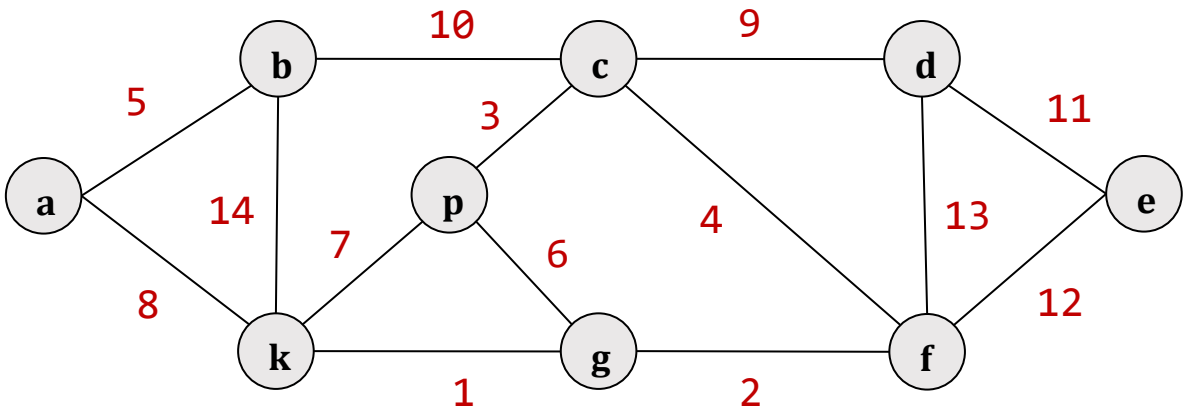
Εξετάζει τις ακμές σύμφωνα με τα βάρη τους (από μικρότερο προς μεγαλύτερο), και προσθέτει σε ένα δάσος Τ, αρχικά **n** κόμβων, αυτές που δεν δημιουργούν κύκλο έως ότου γίνει δένδρο.

2. Αλγόριθμος Prim

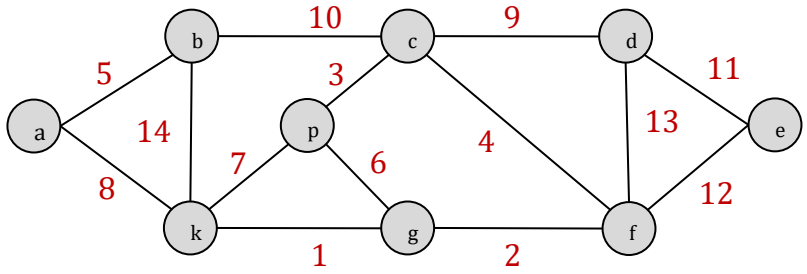
Επεκτείνει ένα δένδρο Τ, αρχικά **1** κόμβου, προσθέτοντας σε αυτό ακμές με το ελάχιστο βάρος έως ότου γίνει δένδρο **n** κόμβων.

Παράδειγμα

1. Αλγόριθμος Kruskal **2. Αλγόριθμος Prim**

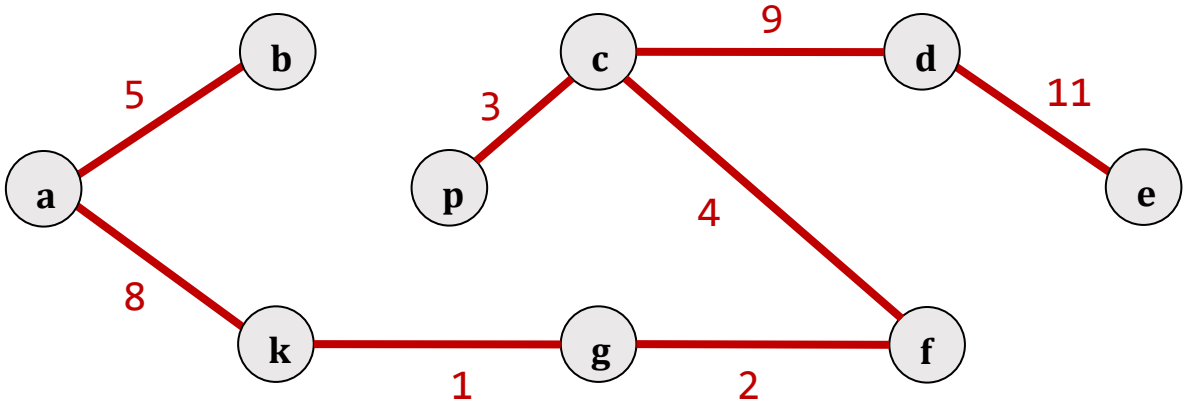


Παράδειγμα

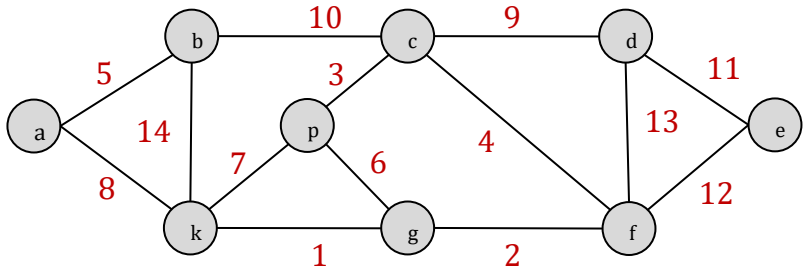


1. Αλγόριθμος Kruskal

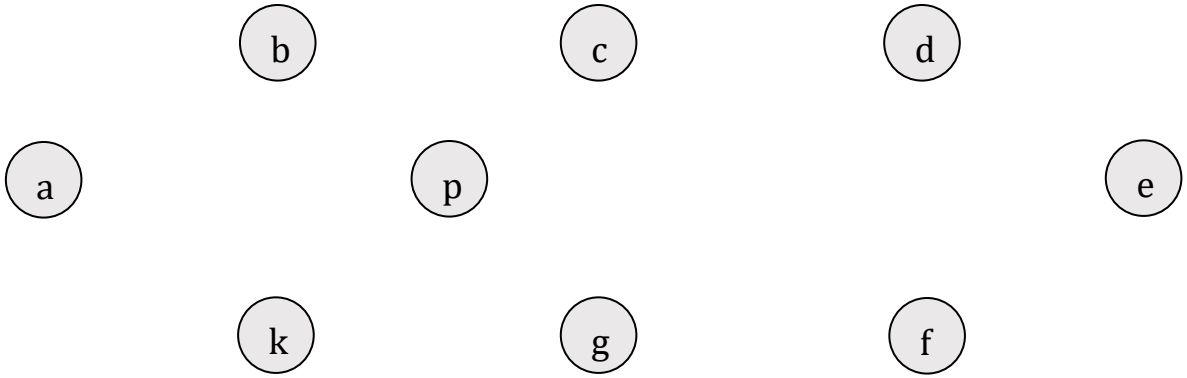
Ελάχιστο Γενετικό Δένδρο



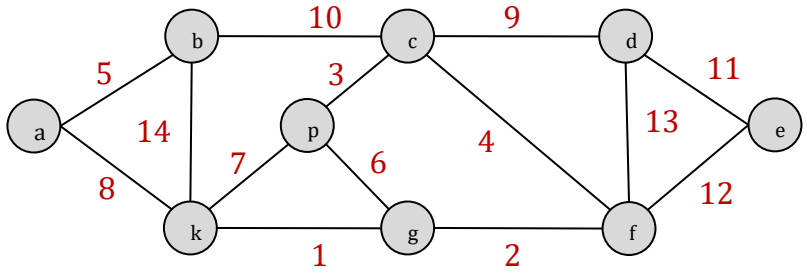
Παράδειγμα



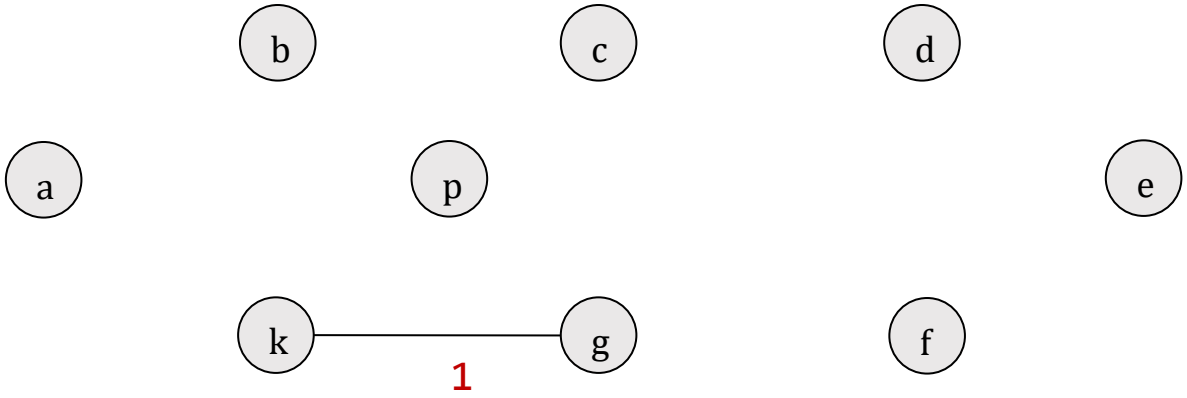
1. Αλγόριθμος Kruskal



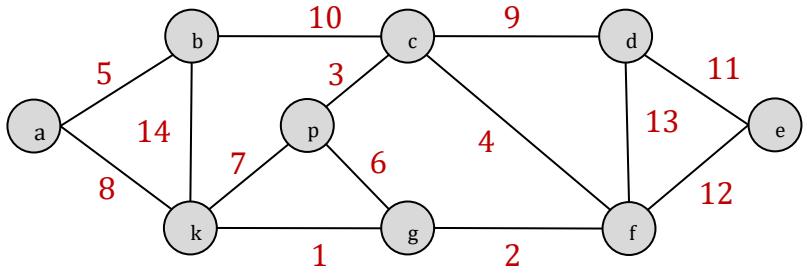
Παράδειγμα



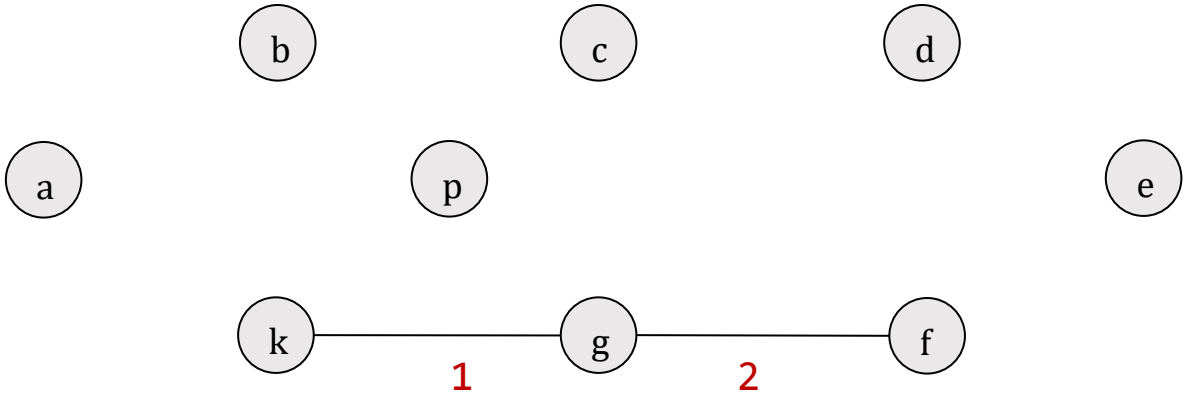
1. Αλγόριθμος Kruskal



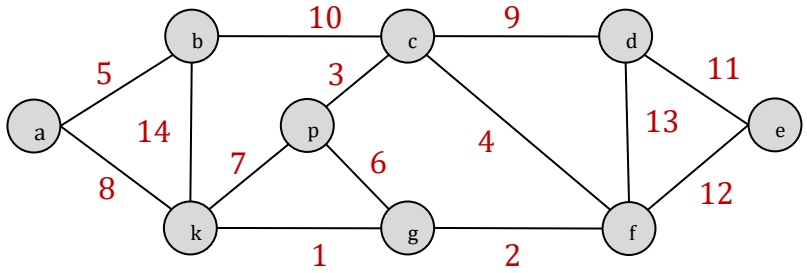
Παράδειγμα



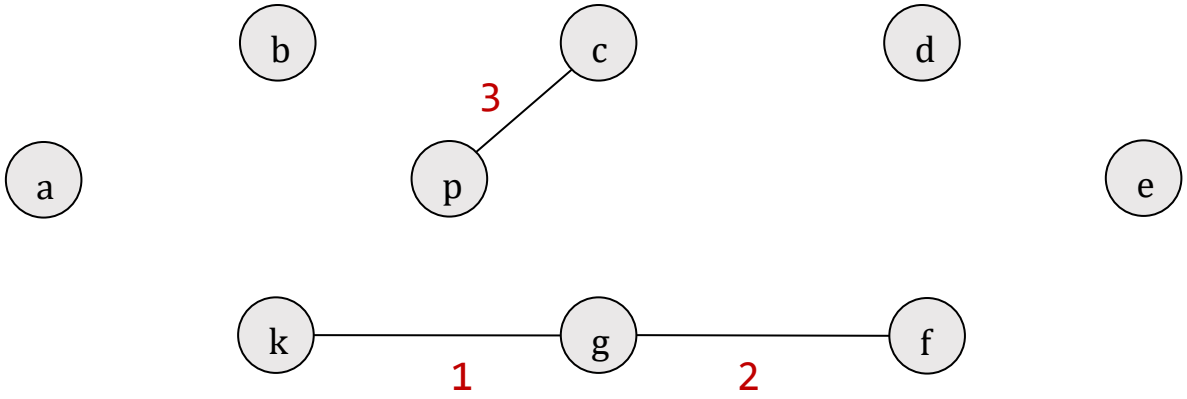
1. Αλγόριθμος Kruskal



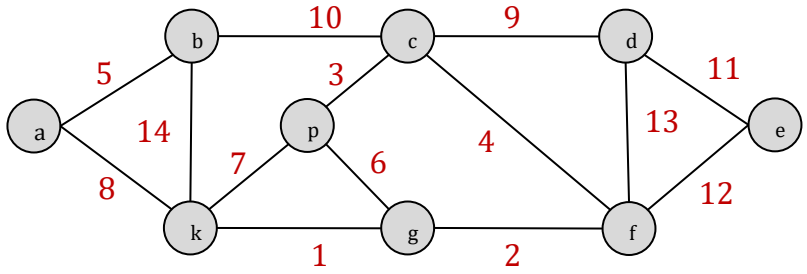
Παράδειγμα



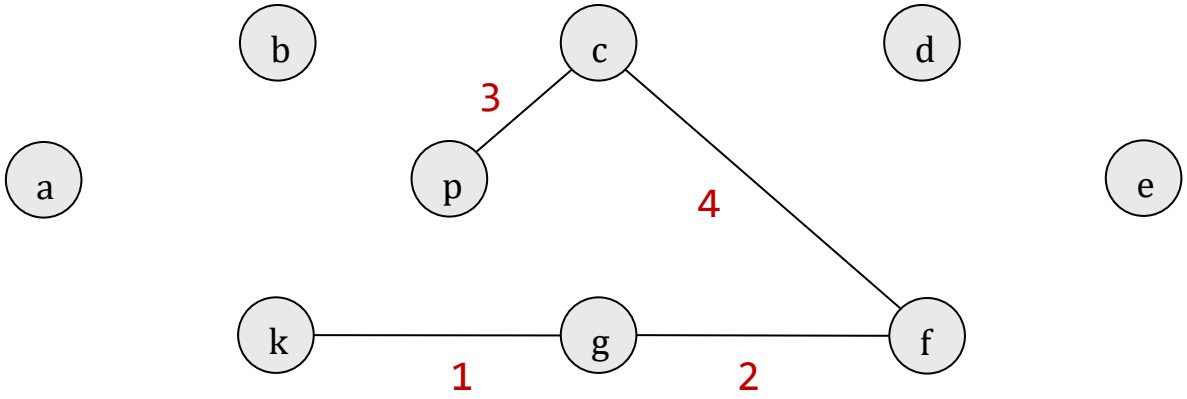
1. Αλγόριθμος Kruskal



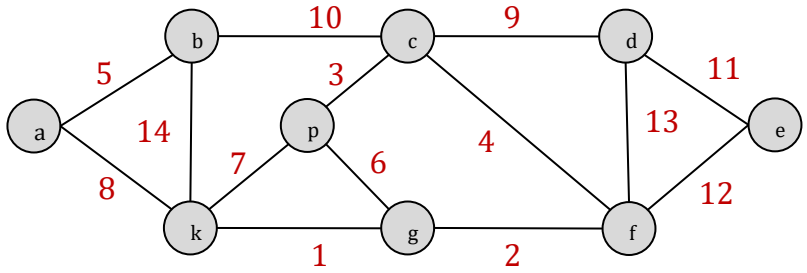
Παράδειγμα



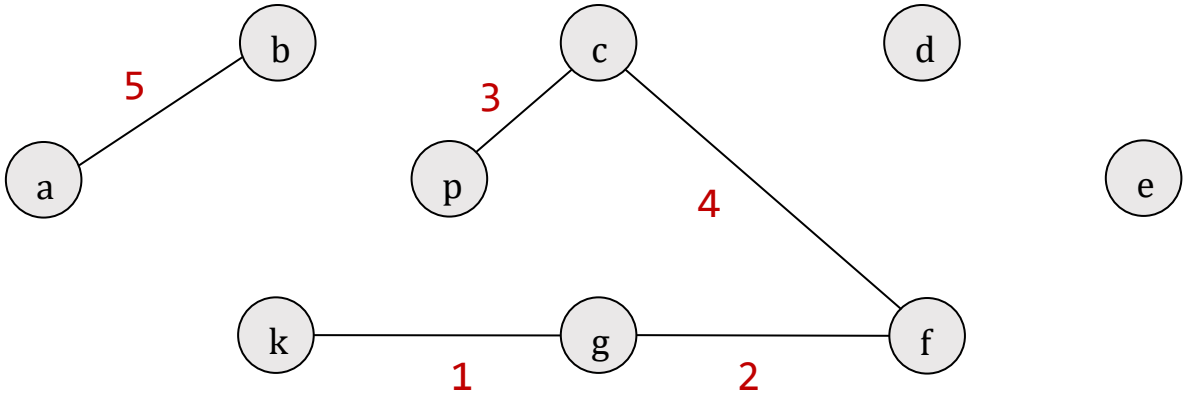
1. Αλγόριθμος Kruskal



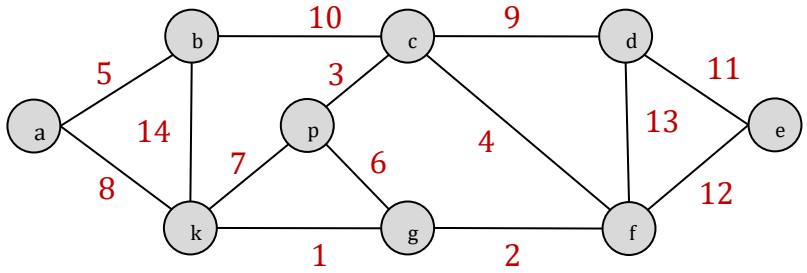
Παράδειγμα



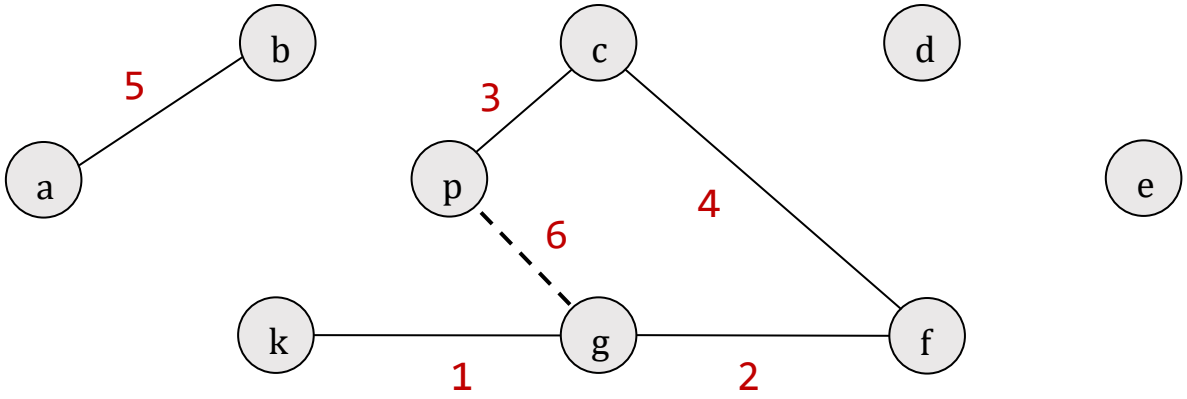
1. Αλγόριθμος Kruskal



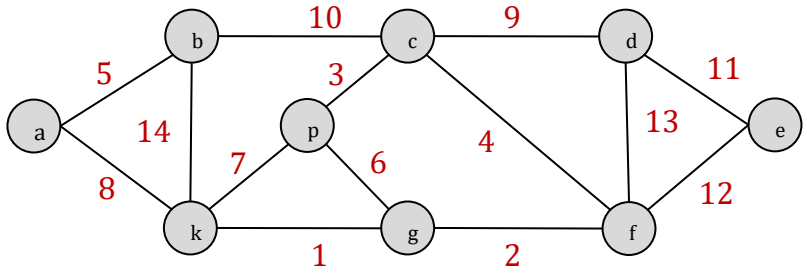
Παράδειγμα



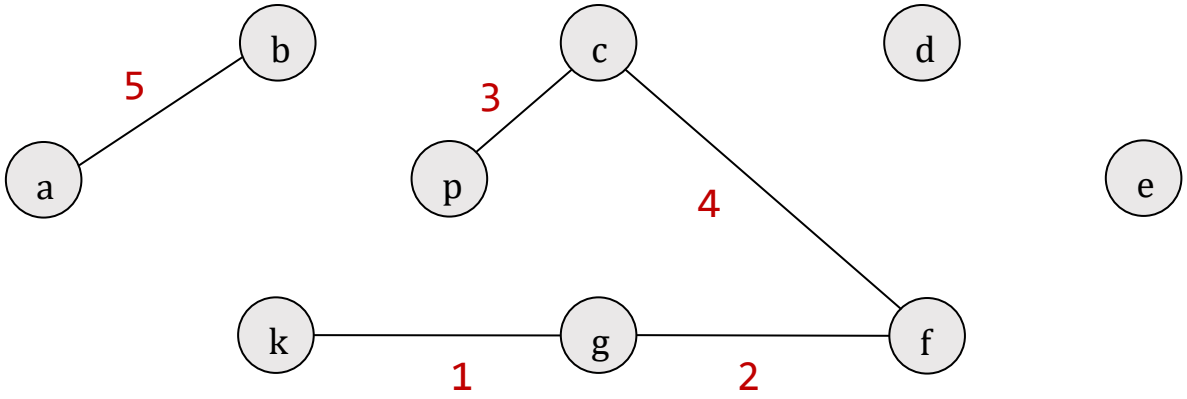
1. Αλγόριθμος Kruskal



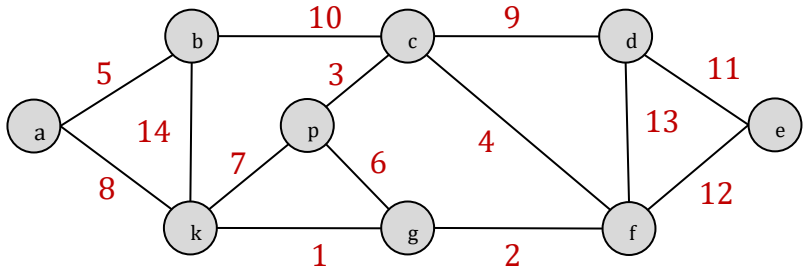
Παράδειγμα



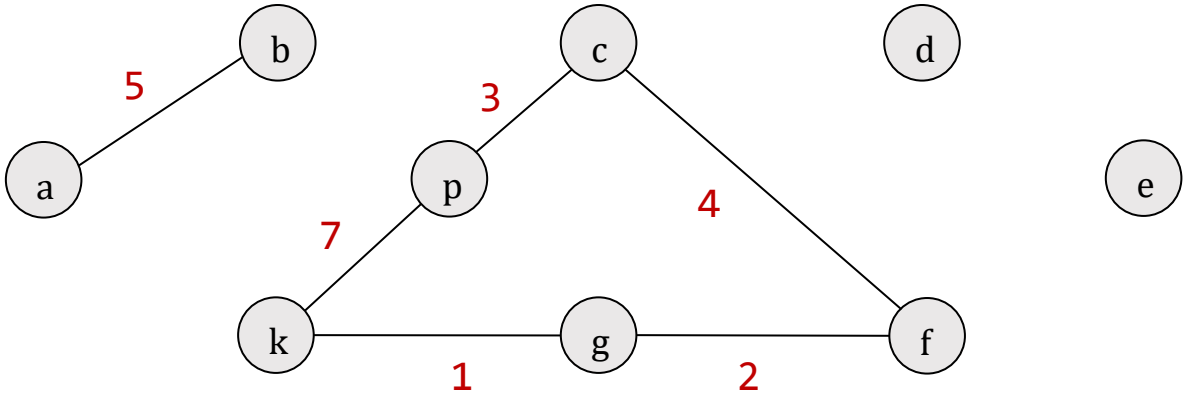
1. Αλγόριθμος Kruskal



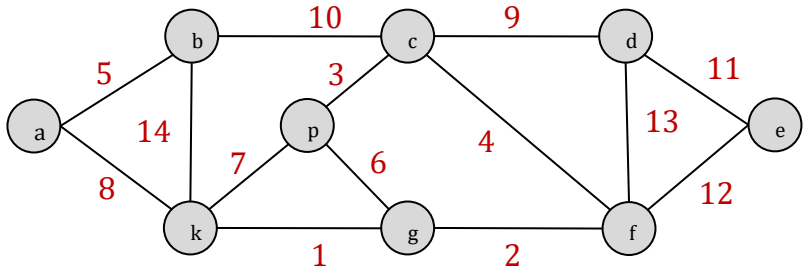
Παράδειγμα



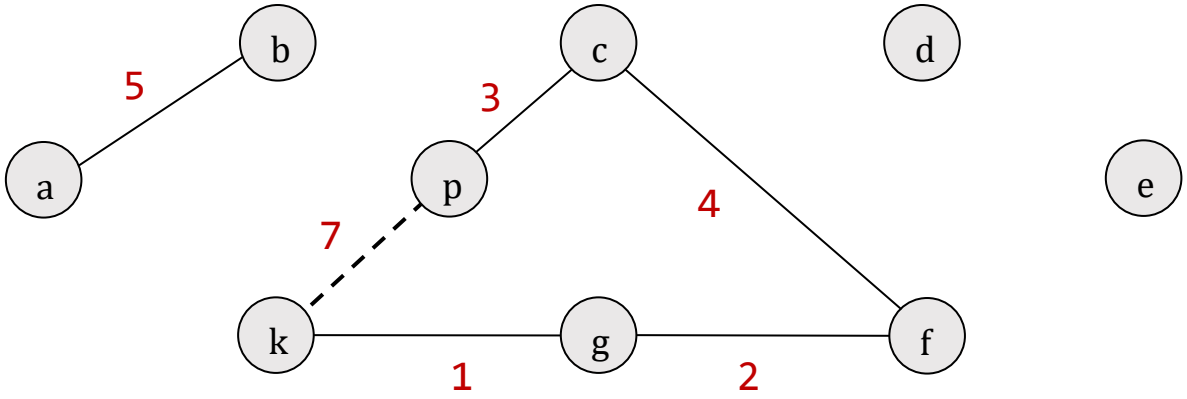
1. Αλγόριθμος Kruskal



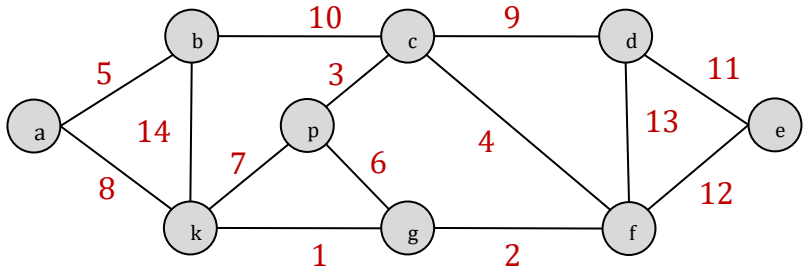
Παράδειγμα



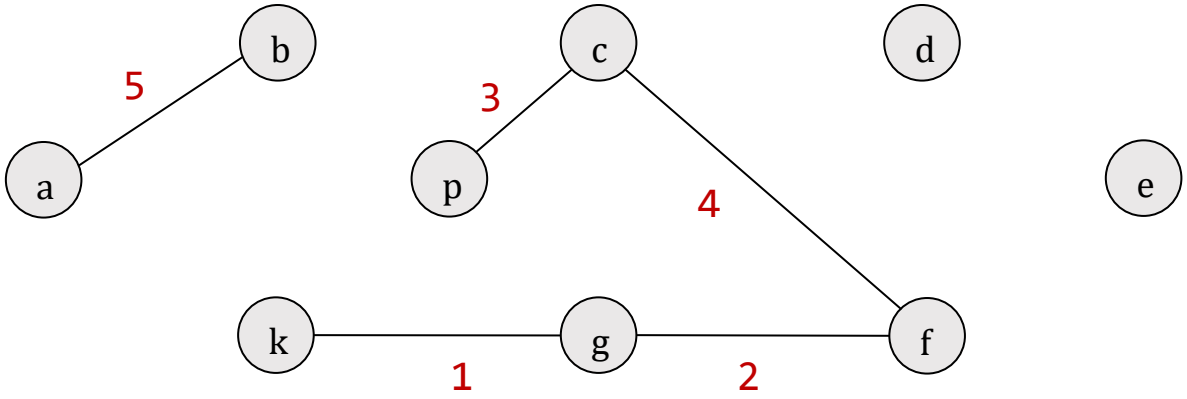
1. Αλγόριθμος Kruskal



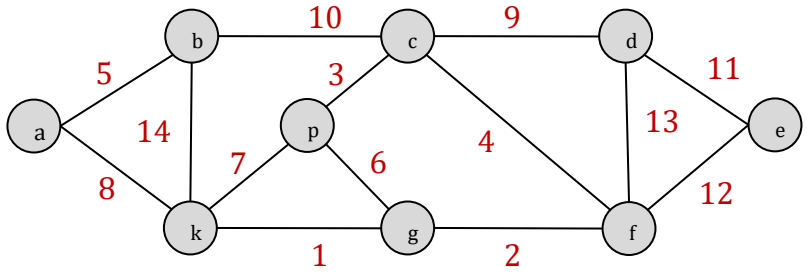
Παράδειγμα



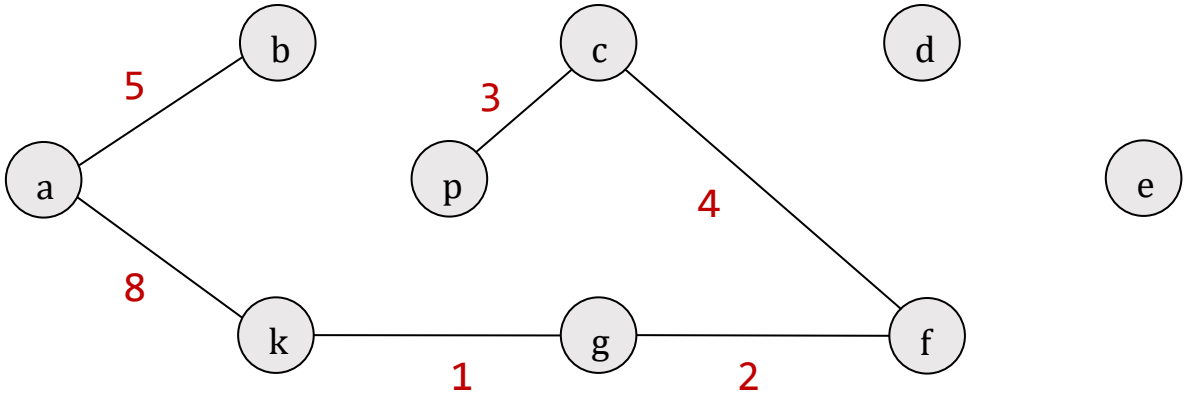
1. Αλγόριθμος Kruskal



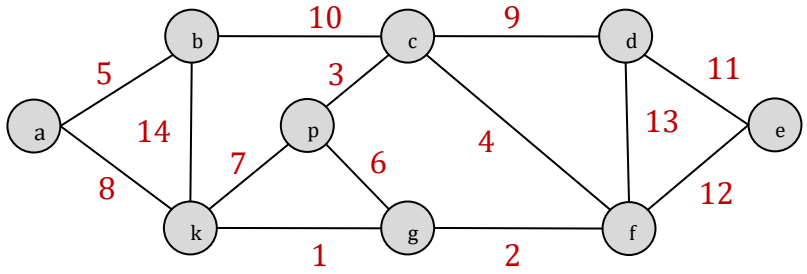
Παράδειγμα



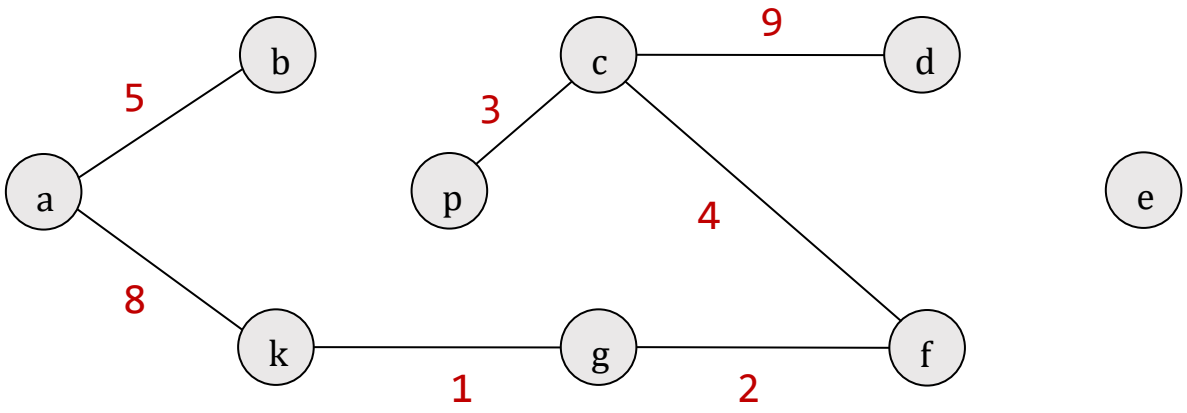
1. Αλγόριθμος Kruskal



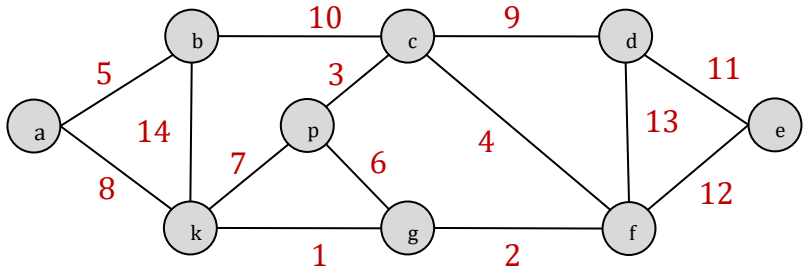
Παράδειγμα



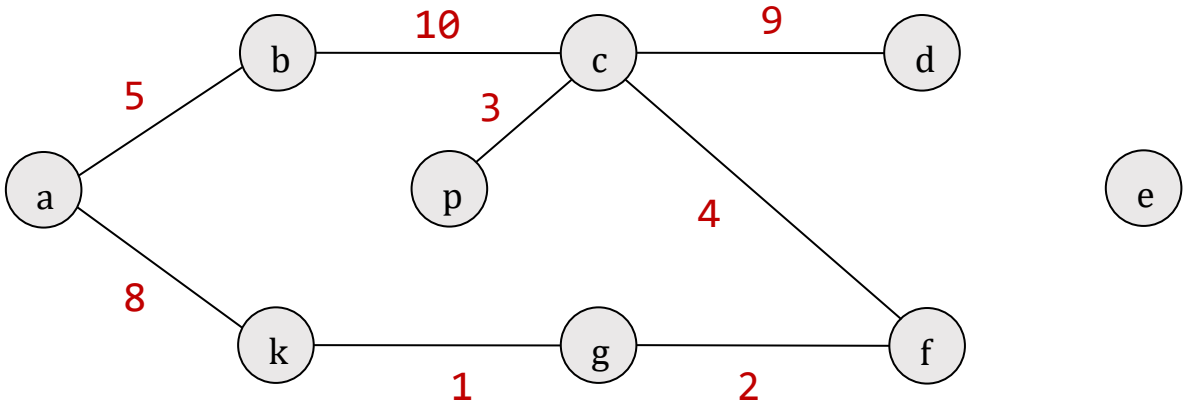
1. Αλγόριθμος Kruskal



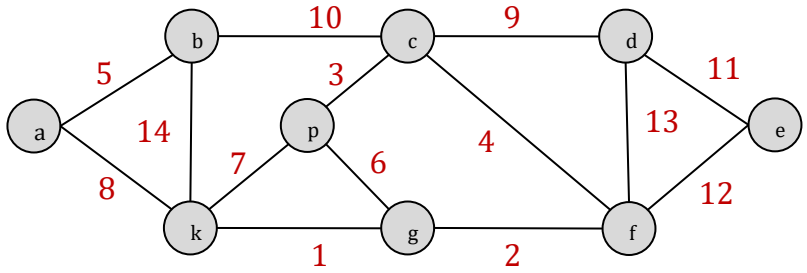
Παράδειγμα



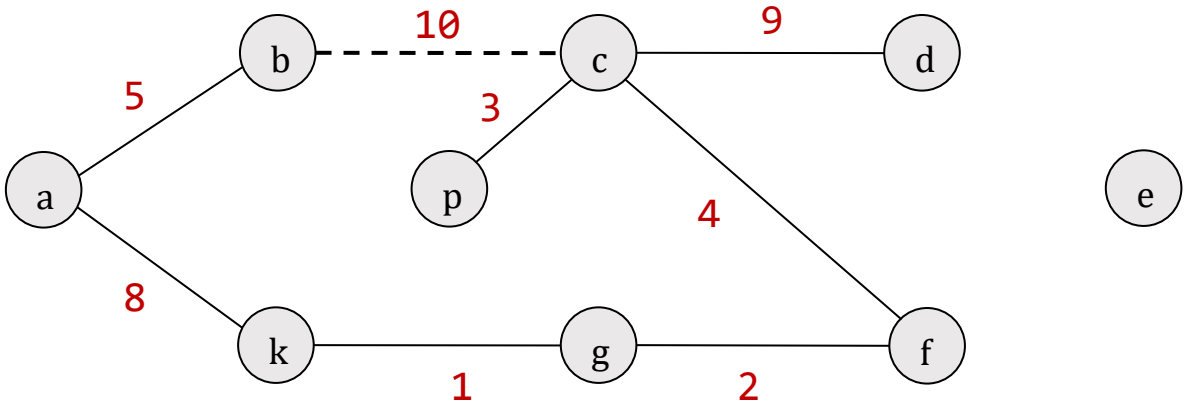
1. Αλγόριθμος Kruskal



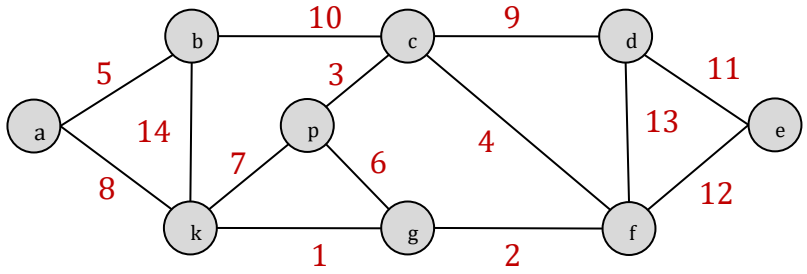
Παράδειγμα



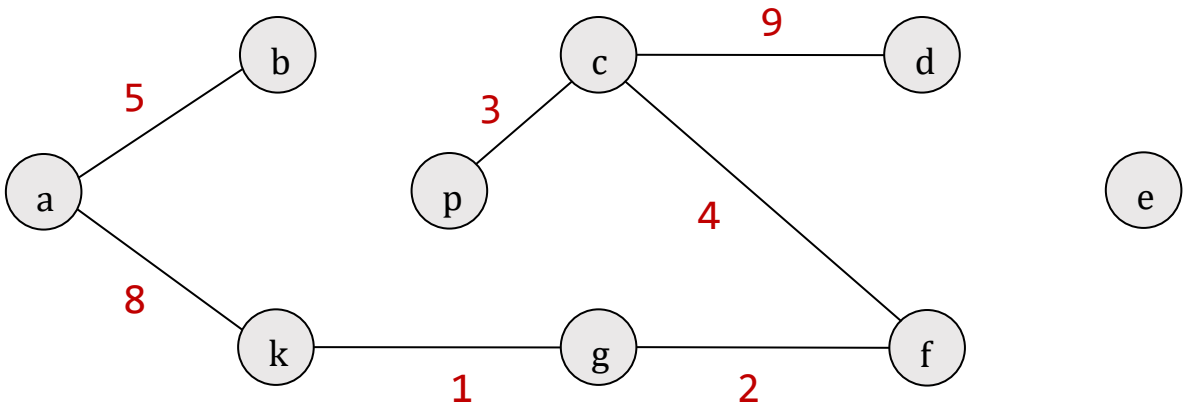
1. Αλγόριθμος Kruskal



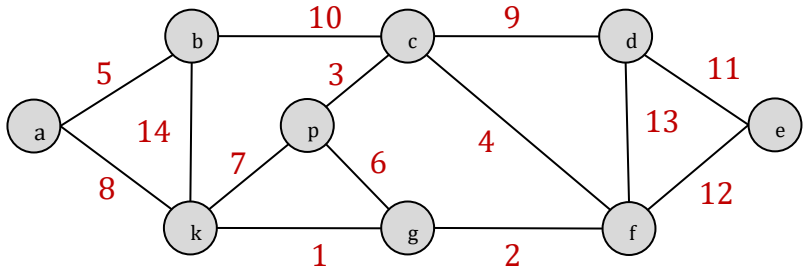
Παράδειγμα



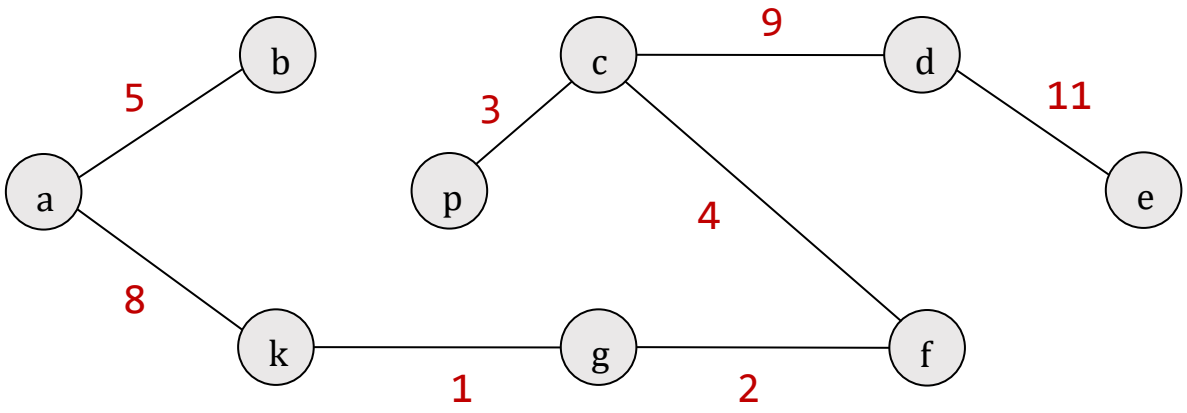
1. Αλγόριθμος Kruskal



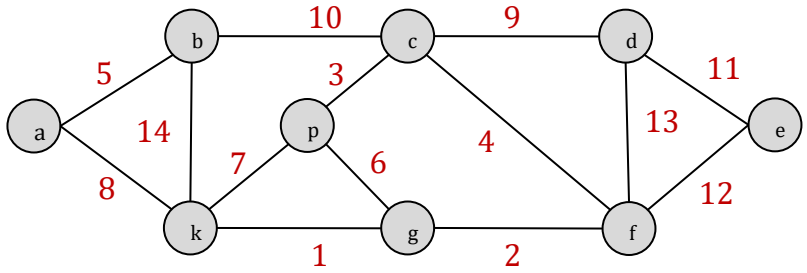
Παράδειγμα



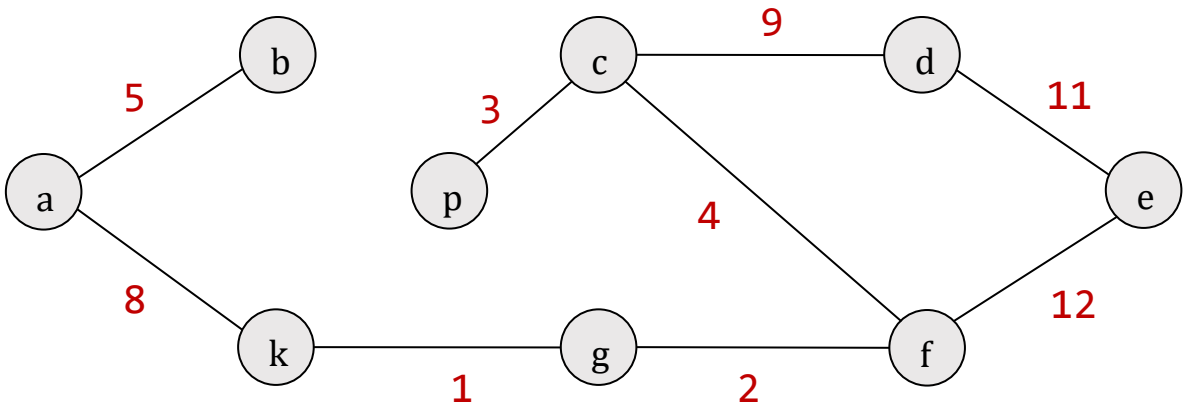
1. Αλγόριθμος Kruskal



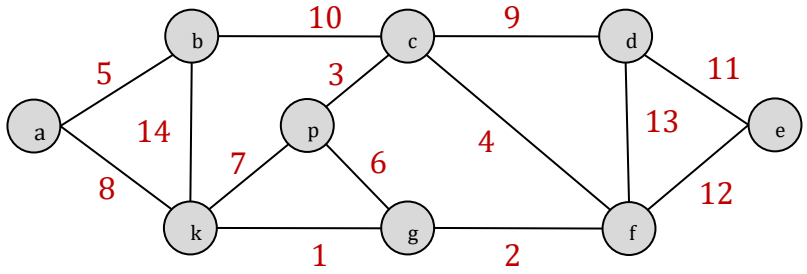
Παράδειγμα



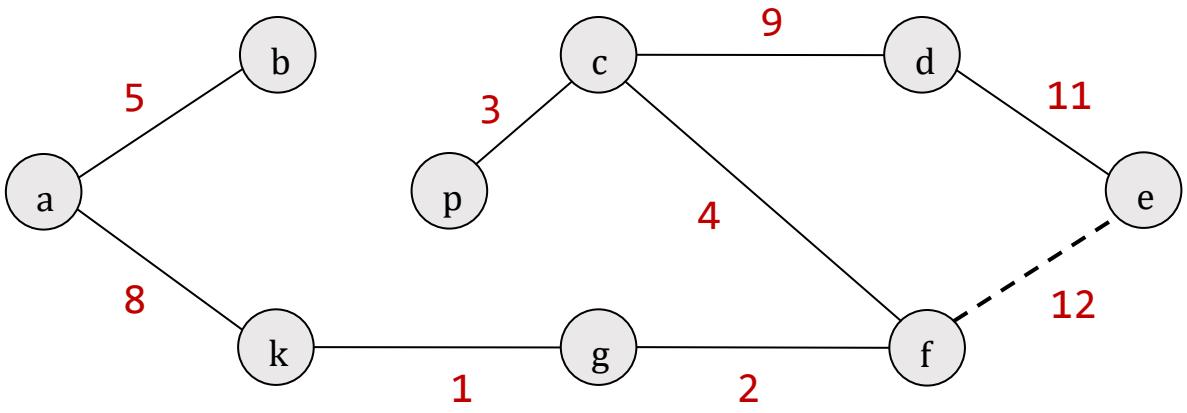
1. Αλγόριθμος Kruskal



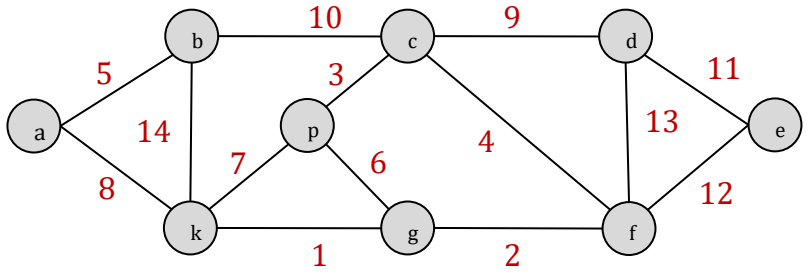
Παράδειγμα



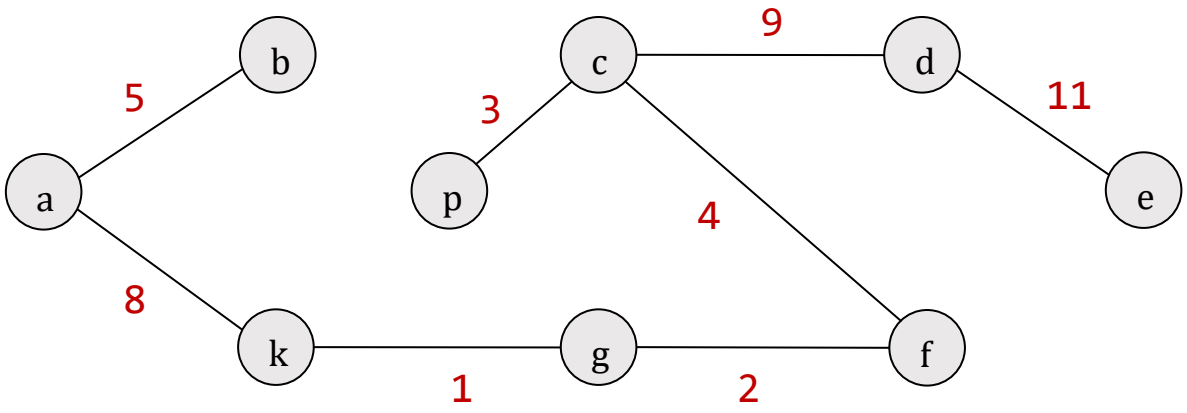
1. Αλγόριθμος Kruskal



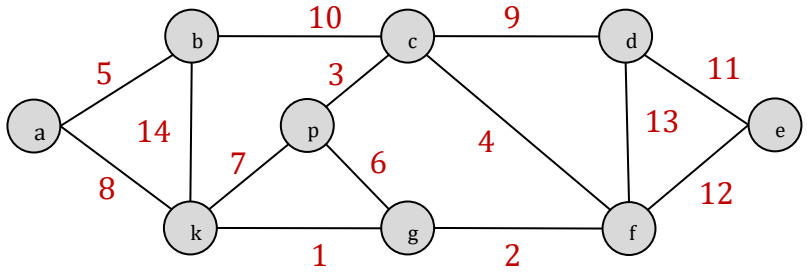
Παράδειγμα



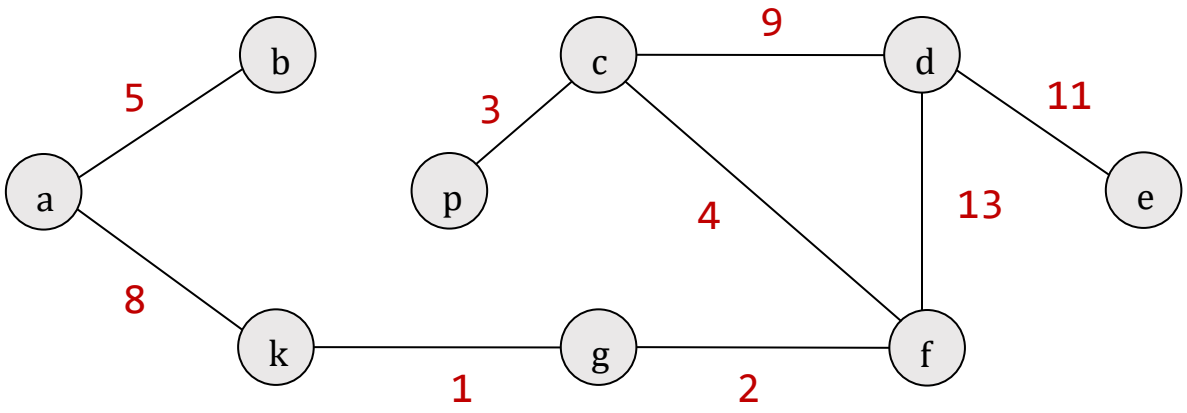
1. Αλγόριθμος Kruskal



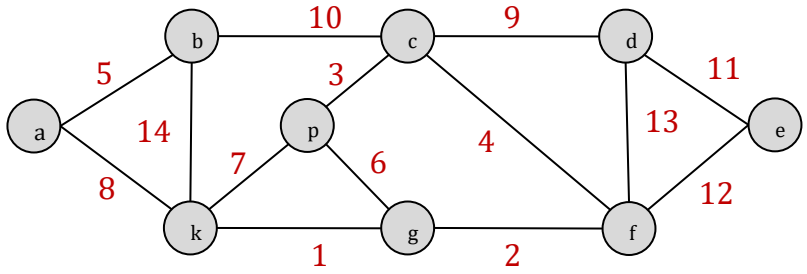
Παράδειγμα



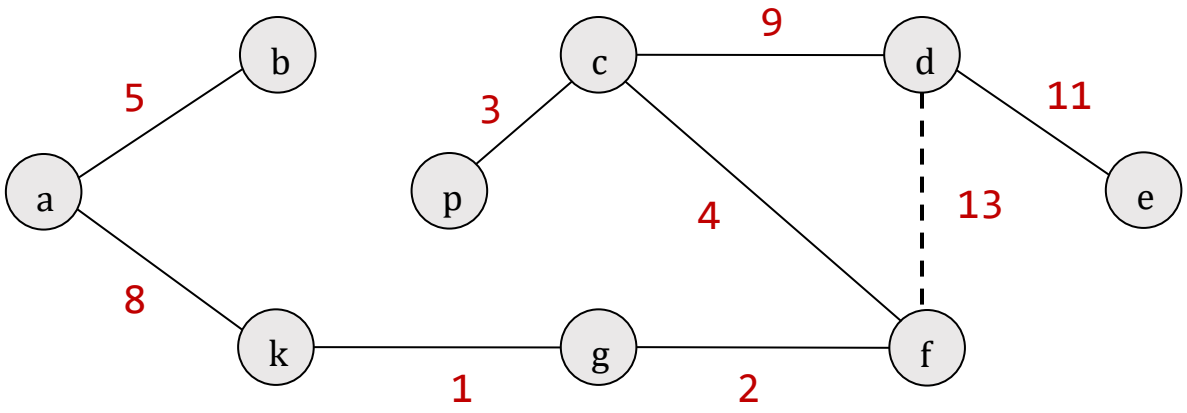
1. Αλγόριθμος Kruskal



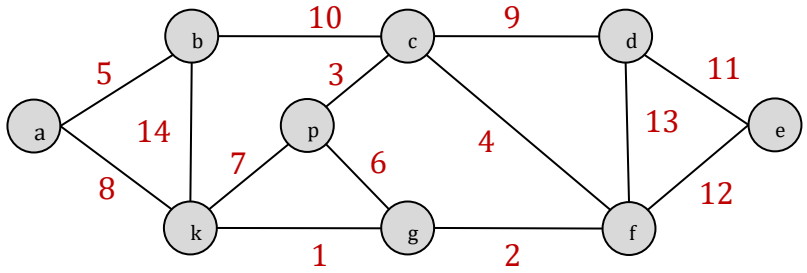
Παράδειγμα



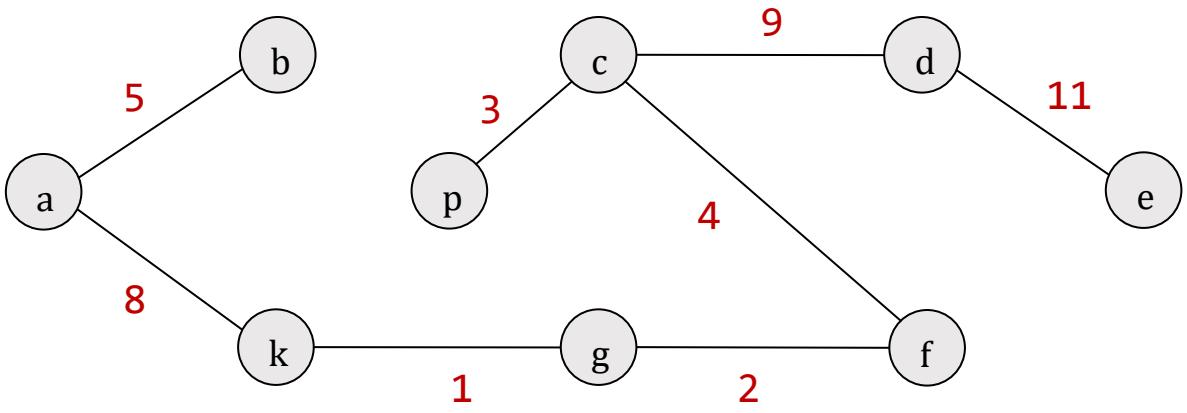
1. Αλγόριθμος Kruskal



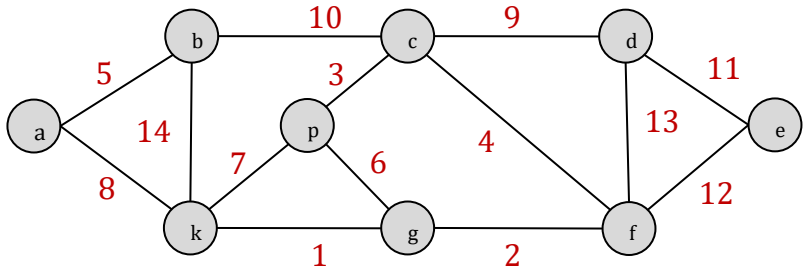
Παράδειγμα



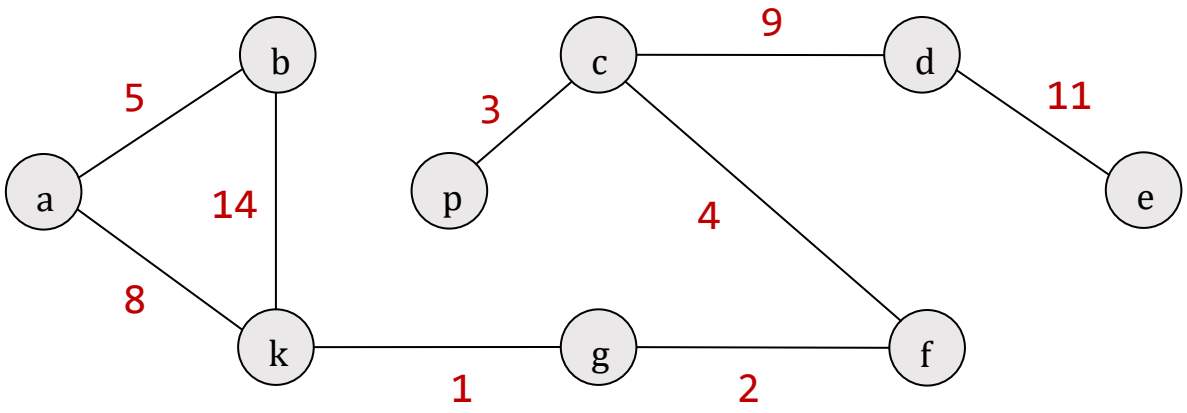
1. Αλγόριθμος Kruskal



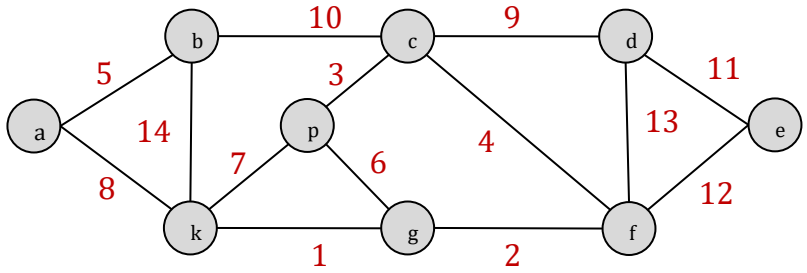
Παράδειγμα



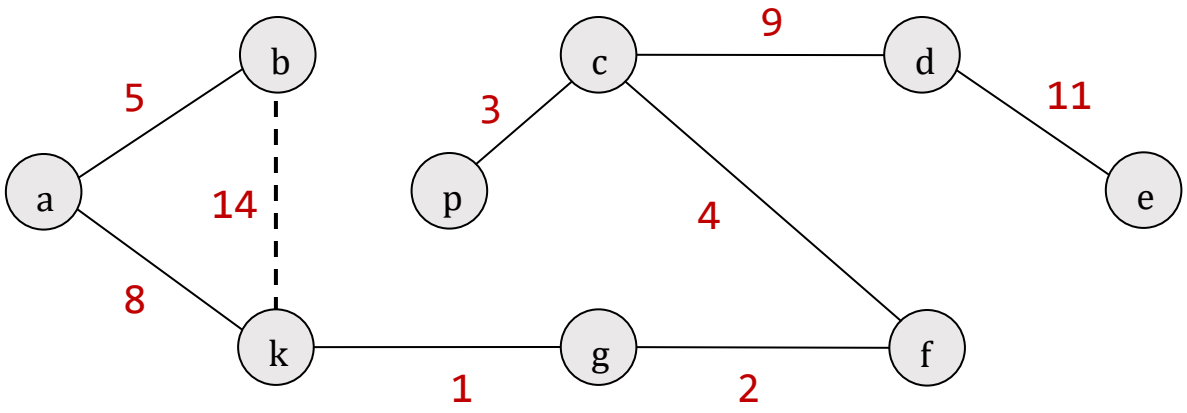
1. Αλγόριθμος Kruskal



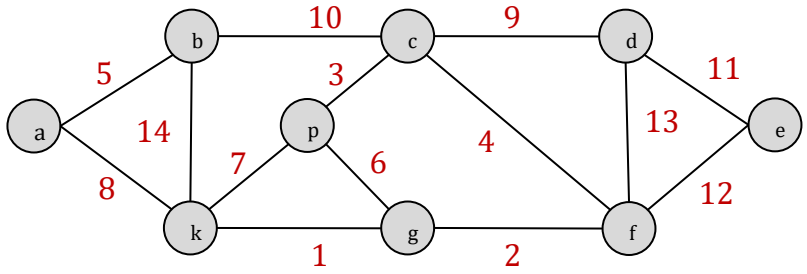
Παράδειγμα



1. Αλγόριθμος Kruskal

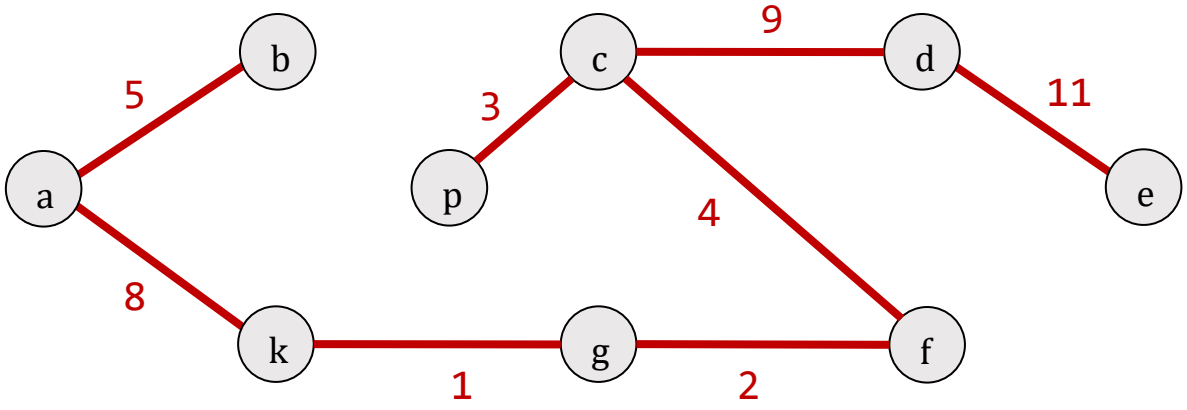


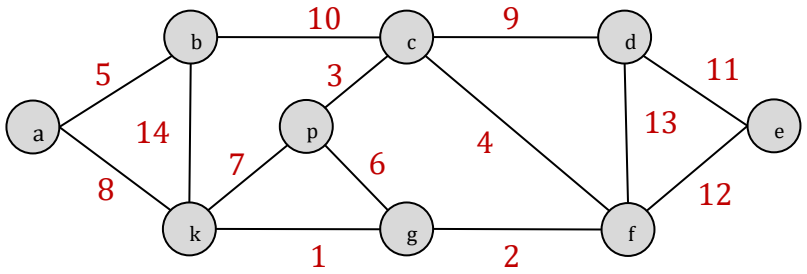
Παράδειγμα



1. Αλγόριθμος Kruskal

Ελάχιστο Γενετικό Δένδρο

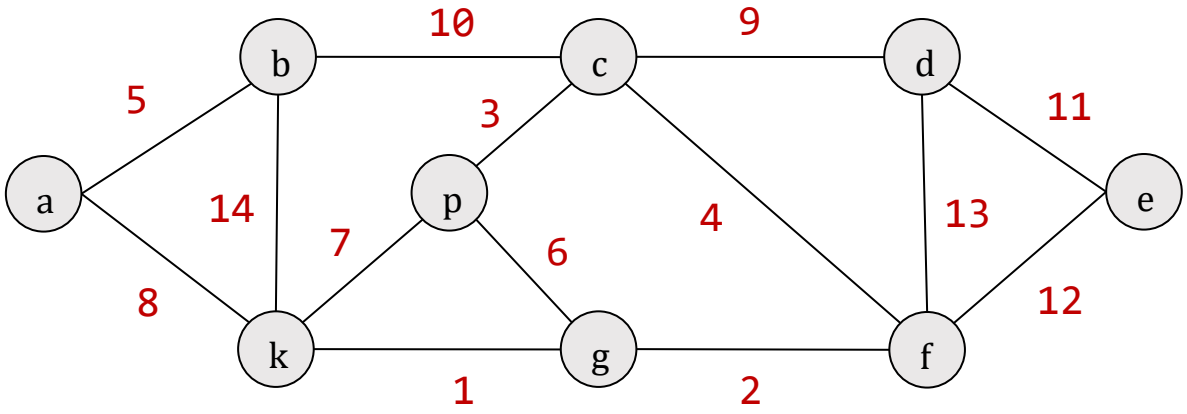


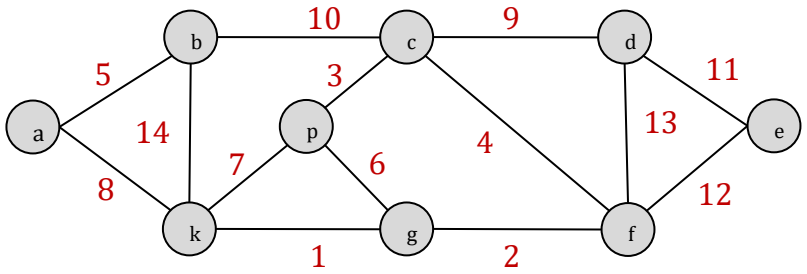


Παράδειγμα

2. Αλγόριθμος Prim

Γράφημα G

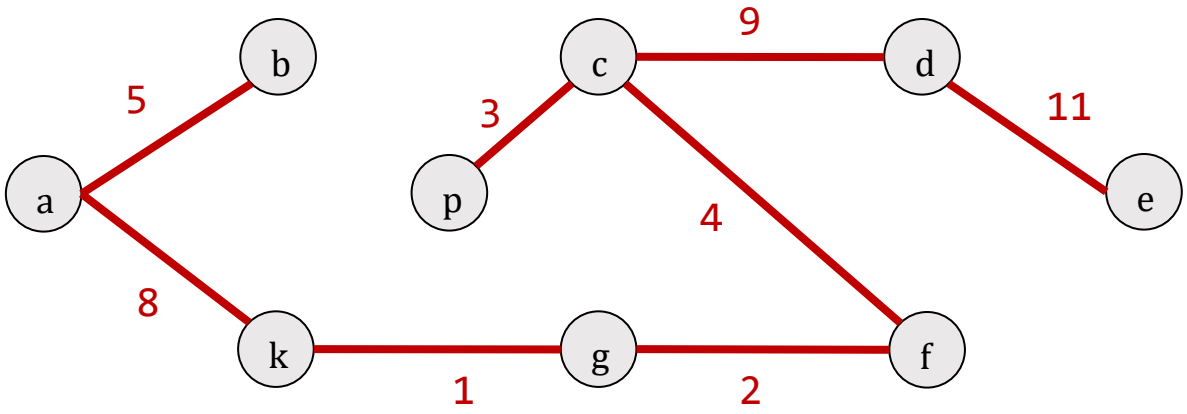




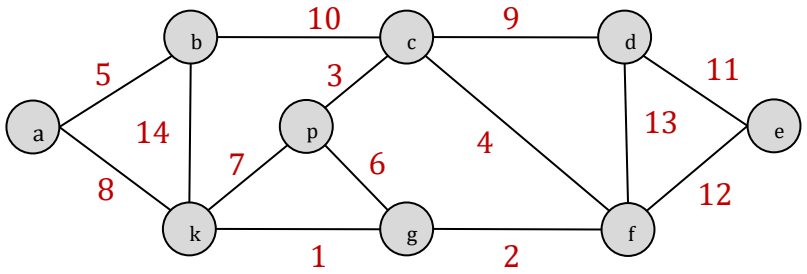
Παράδειγμα

2. Αλγόριθμος Prim

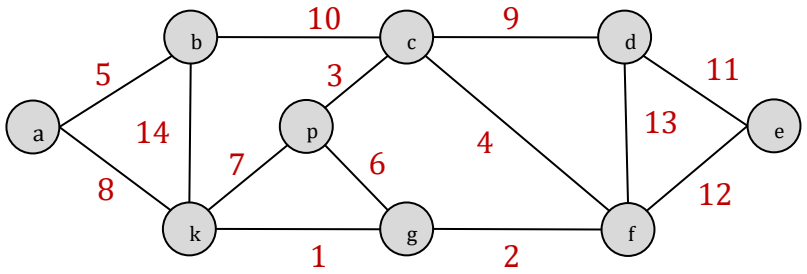
Ελάχιστο Γενετικό Δένδρο Τ



Παράδειγμα

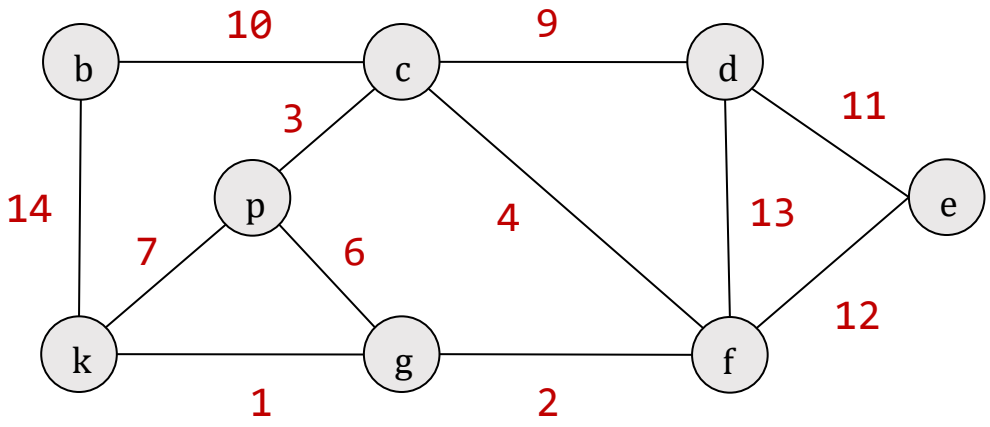


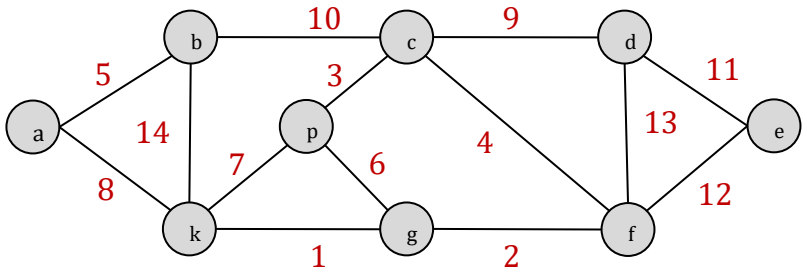
2. Αλγόριθμος Prim



Παράδειγμα

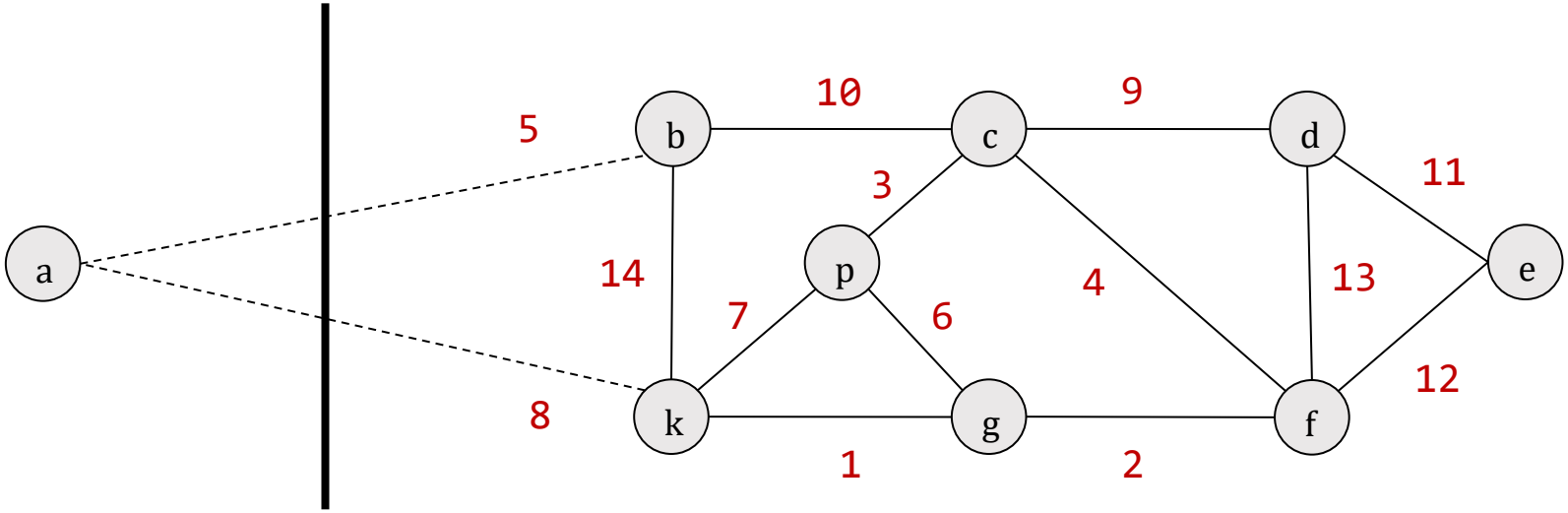
2. Αλγόριθμος Prim



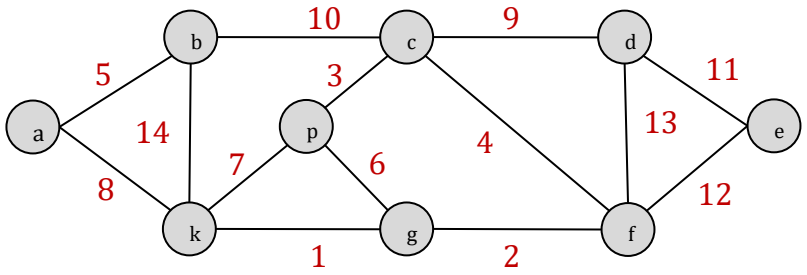


Παράδειγμα

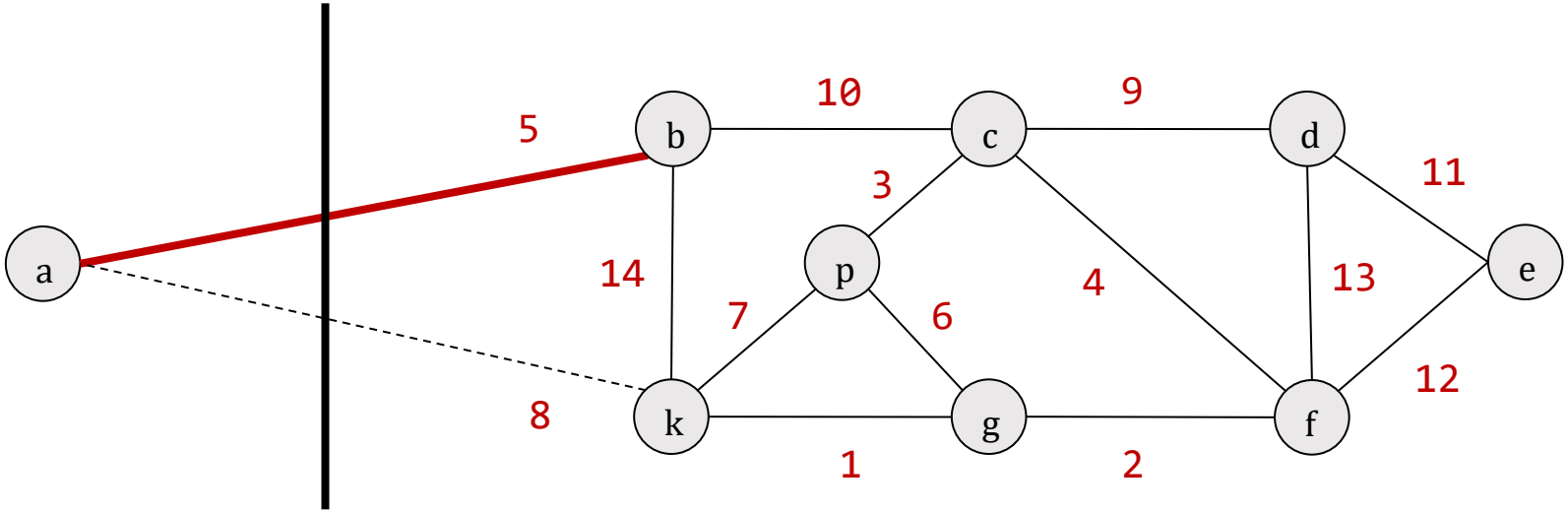
2. Αλγόριθμος Prim

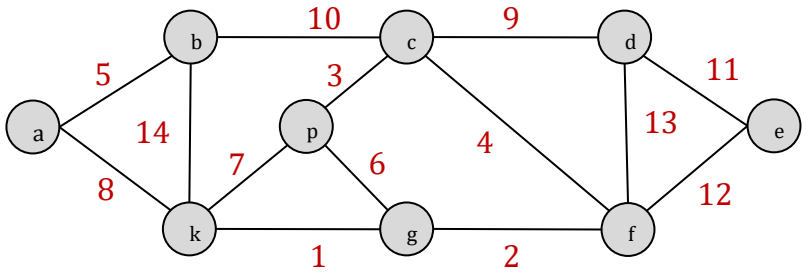


Παράδειγμα



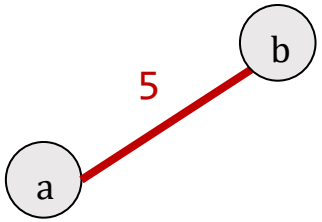
2. Αλγόριθμος Prim

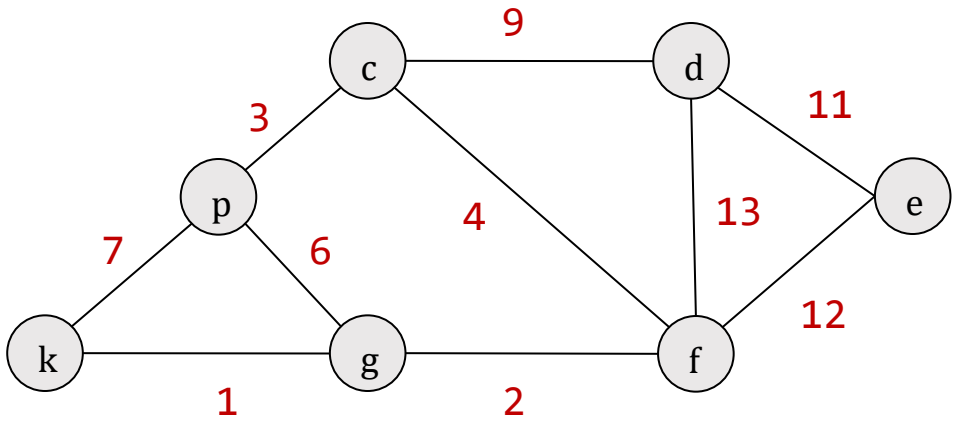


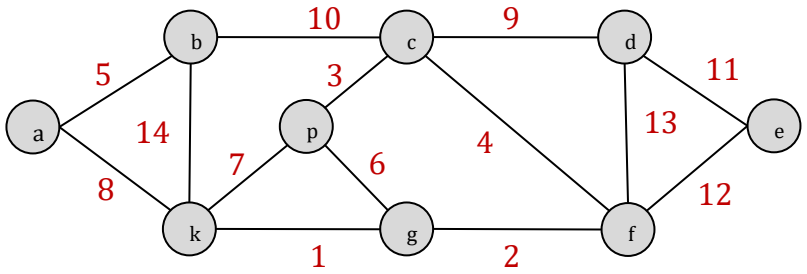


Παράδειγμα

2. Αλγόριθμος Prim

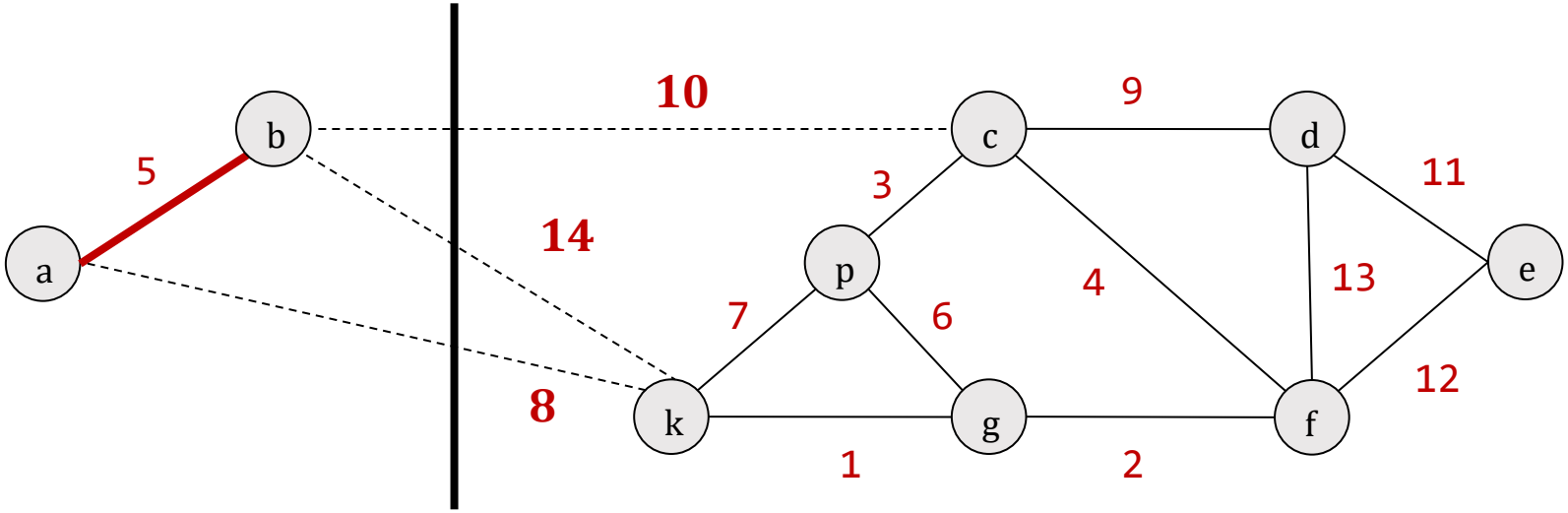


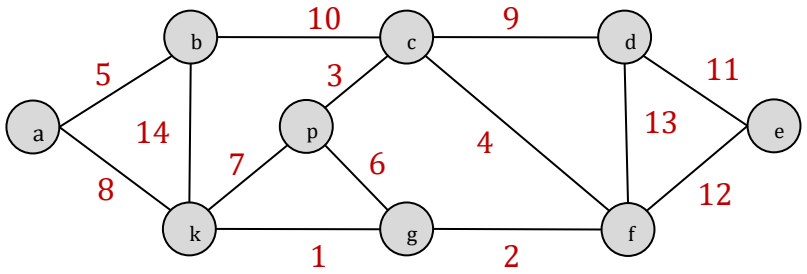




Παράδειγμα

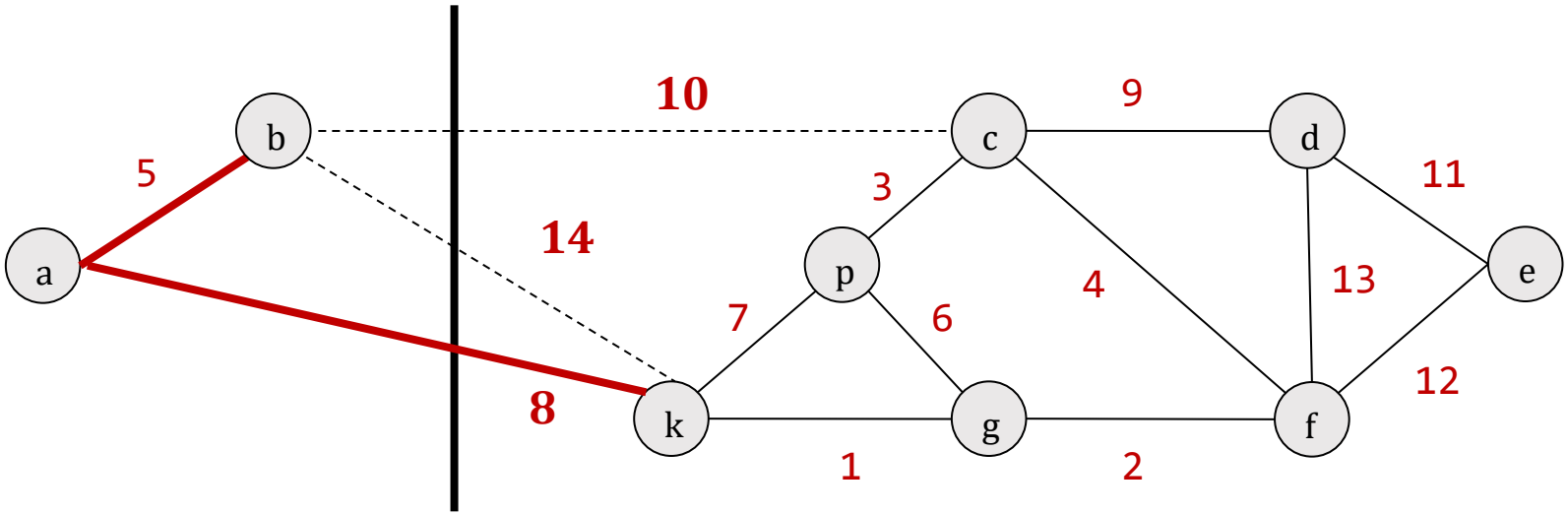
2. Αλγόριθμος Prim



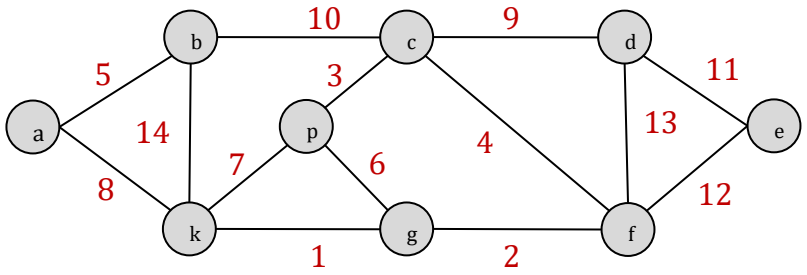


Παράδειγμα

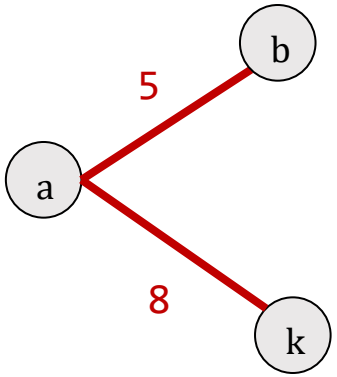
2. Αλγόριθμος Prim

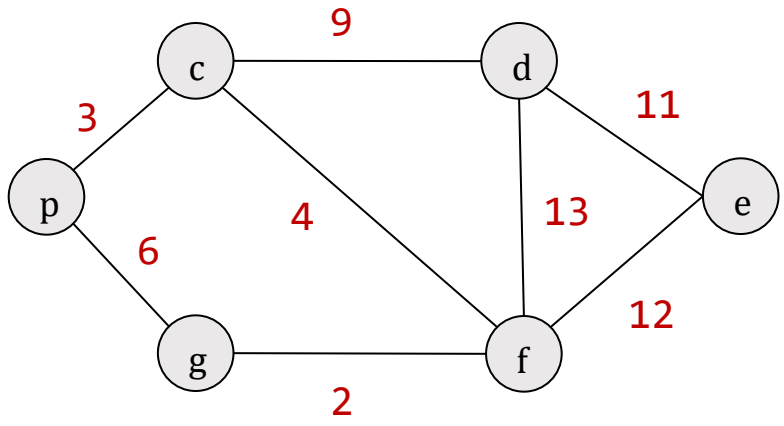


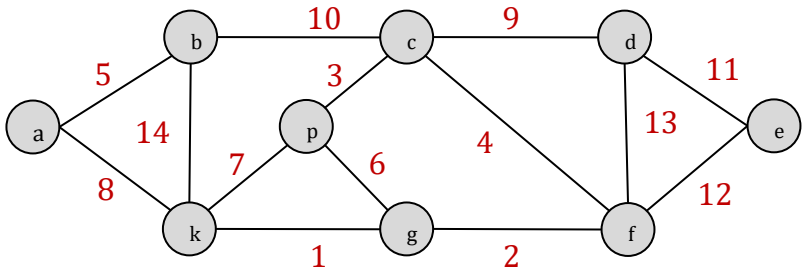
Παράδειγμα



2. Αλγόριθμος Prim

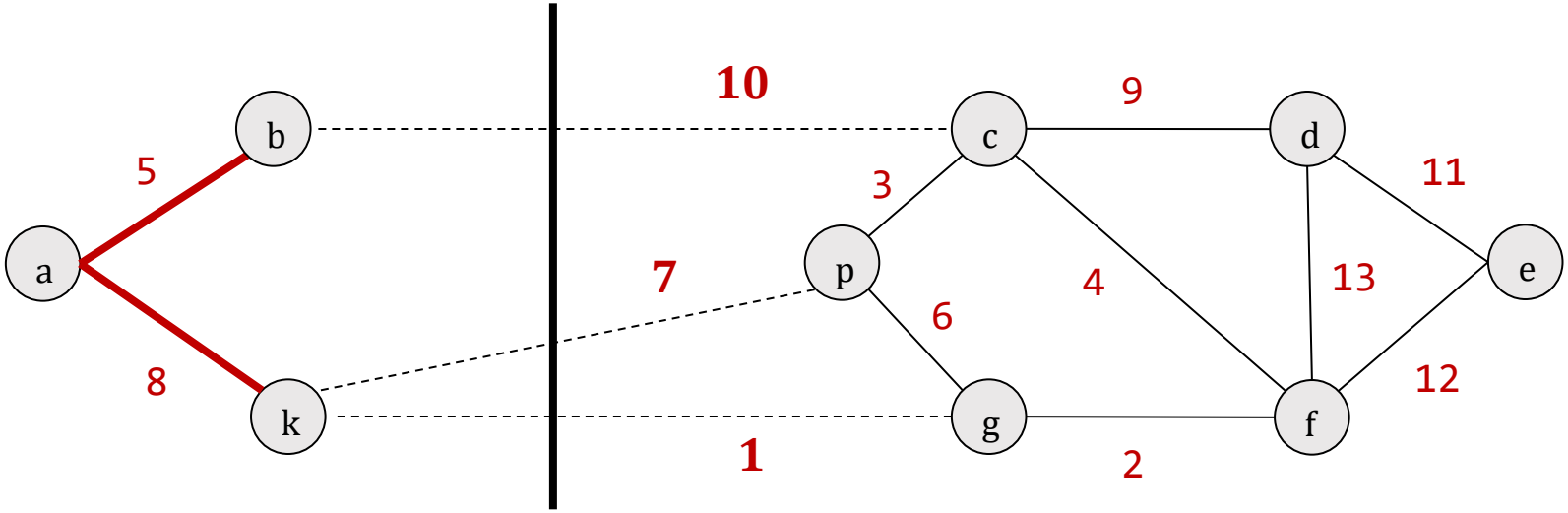


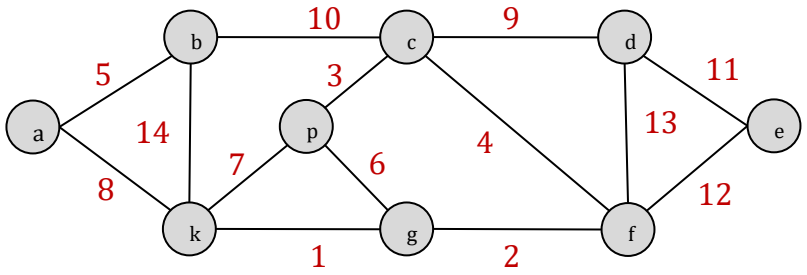




Παράδειγμα

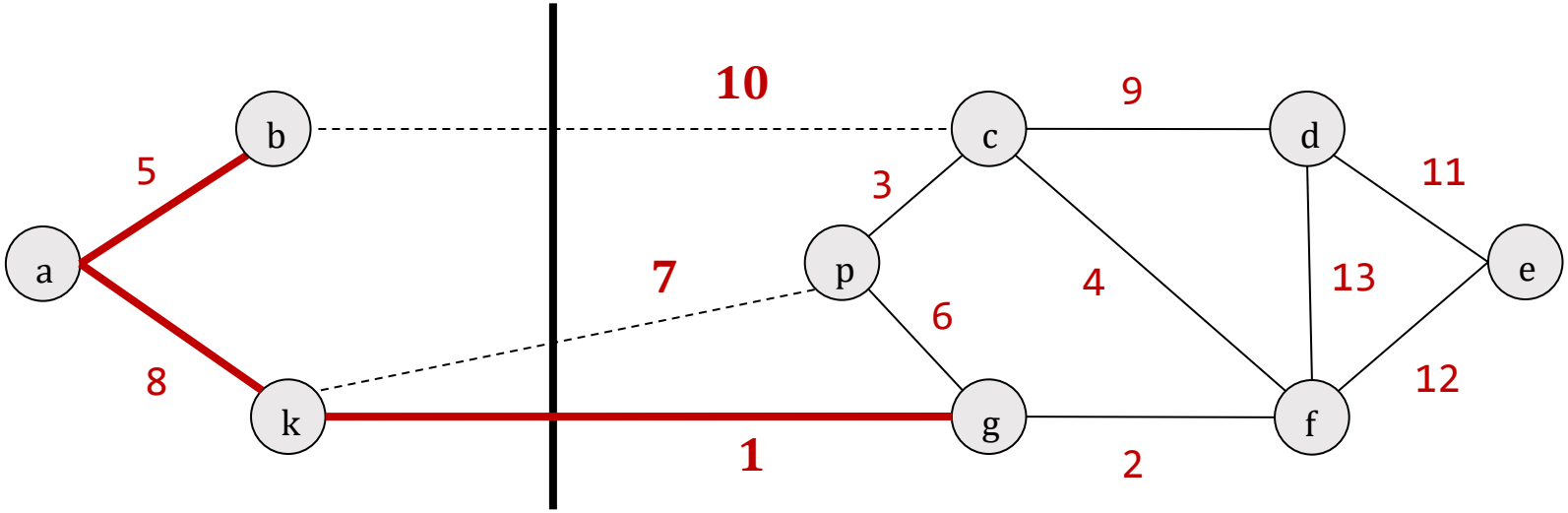
2. Αλγόριθμος Prim

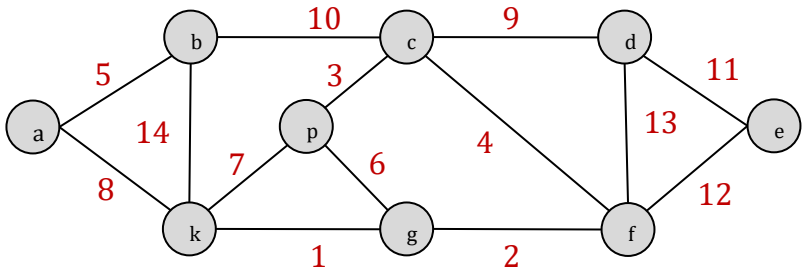




Παράδειγμα

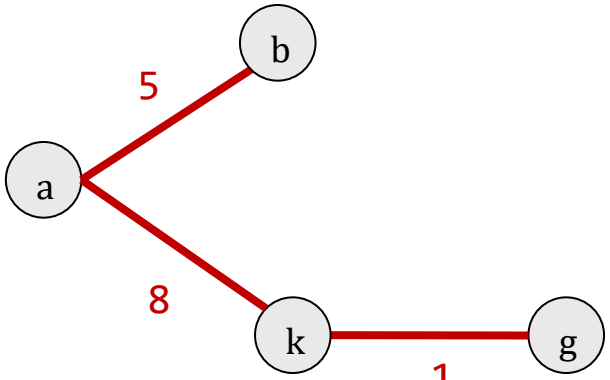
2. Αλγόριθμος Prim

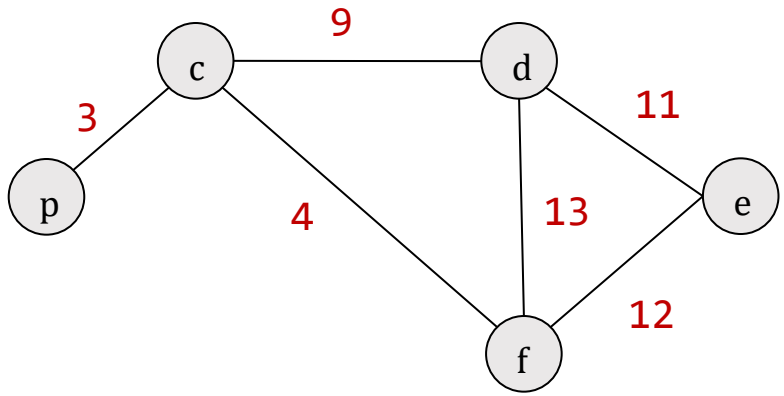




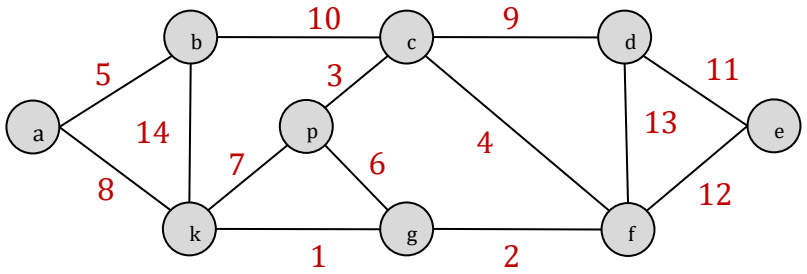
Παράδειγμα

2. Αλγόριθμος Prim

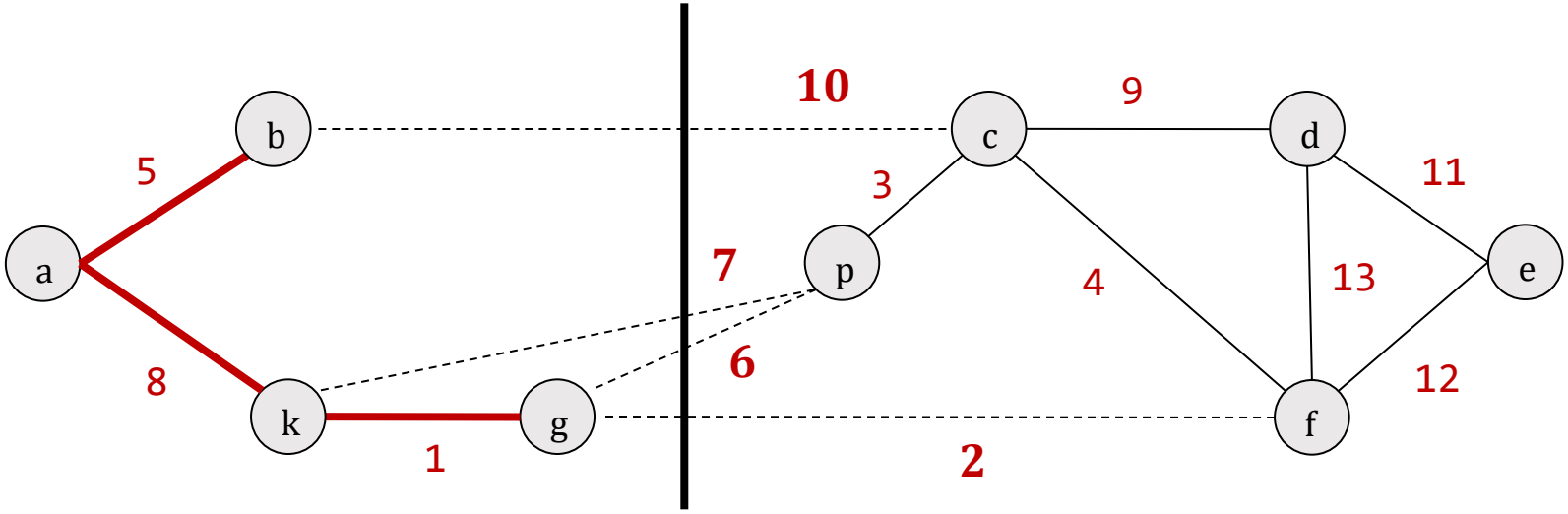




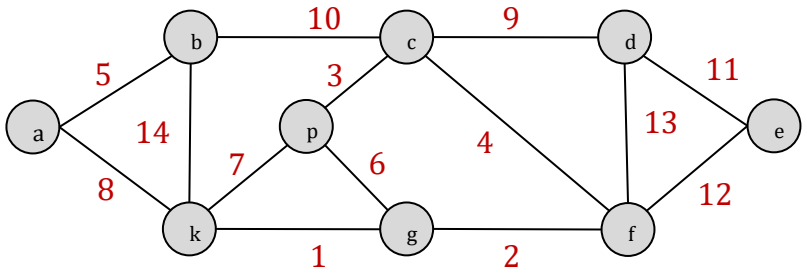
Παράδειγμα



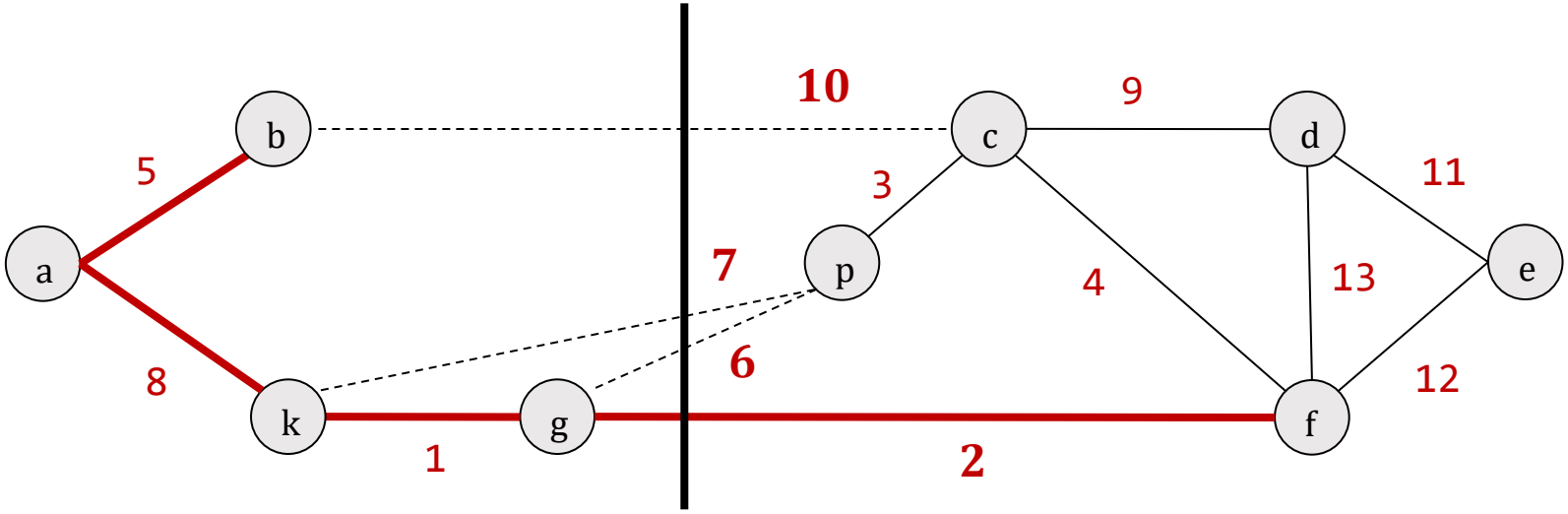
2. Αλγόριθμος Prim

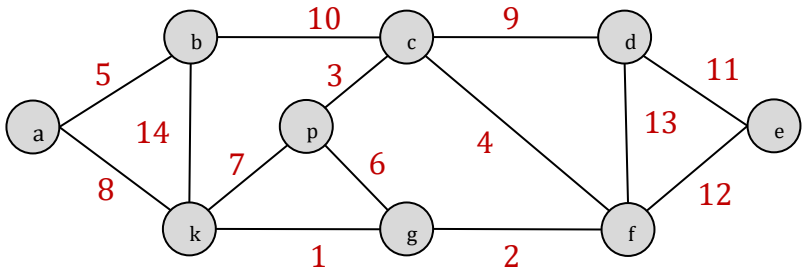


Παράδειγμα



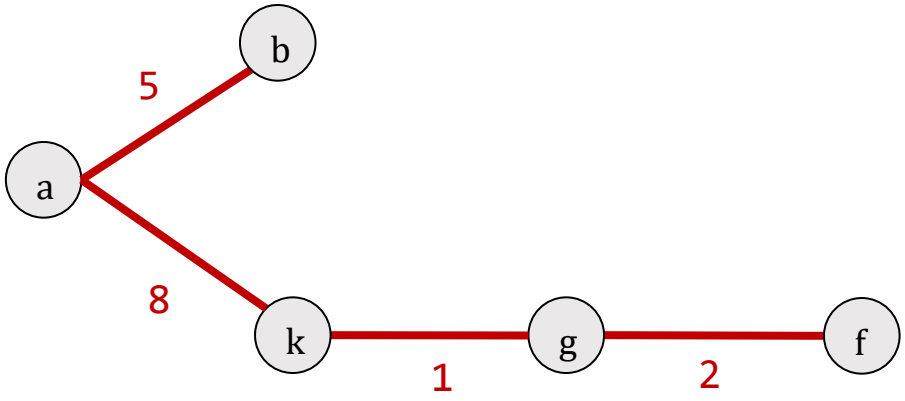
2. Αλγόριθμος Prim

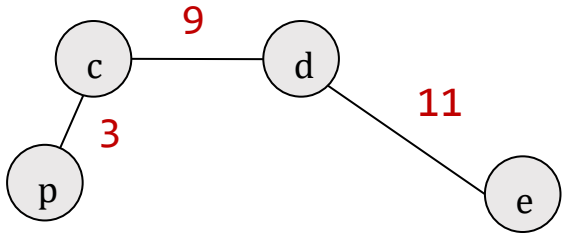


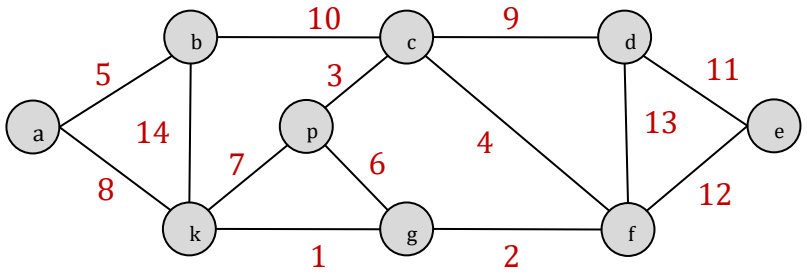


Παράδειγμα

2. Αλγόριθμος Prim

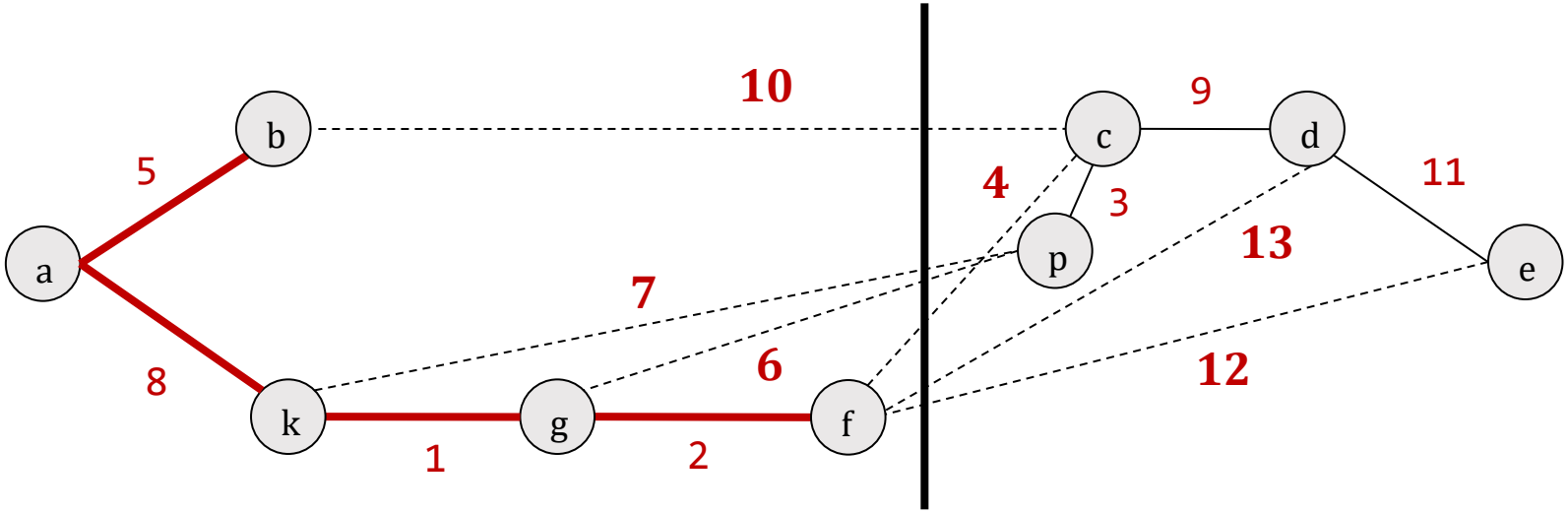




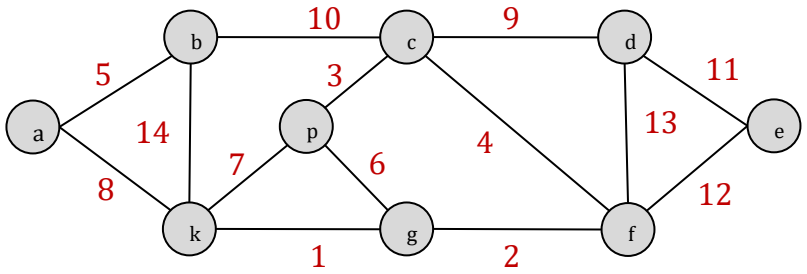


Παράδειγμα

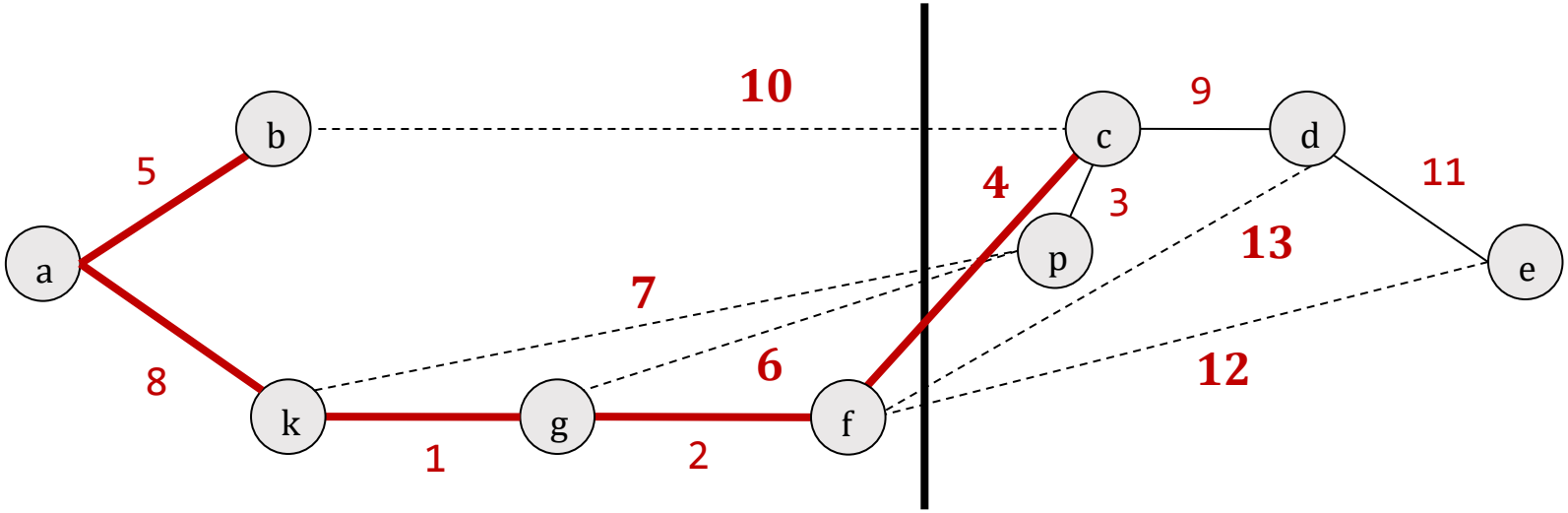
2. Αλγόριθμος Prim

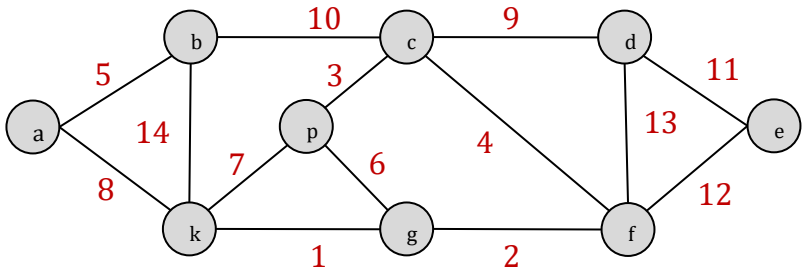


Παράδειγμα



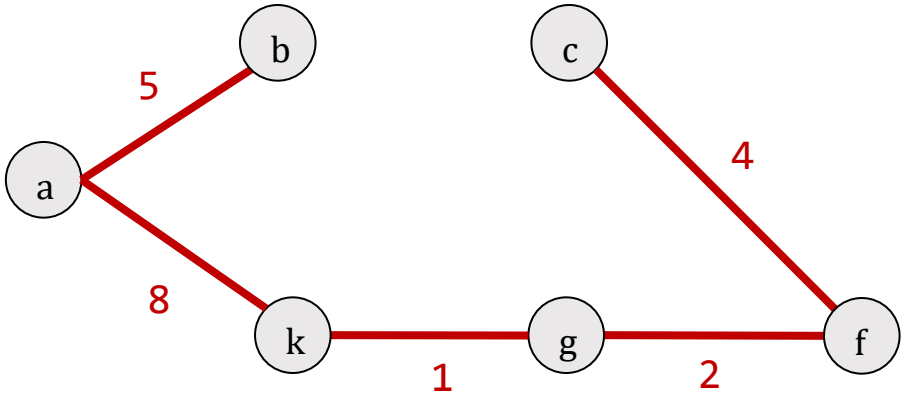
2. Αλγόριθμος Prim

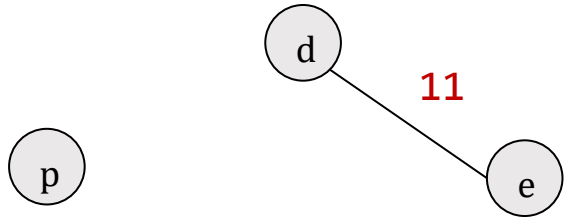


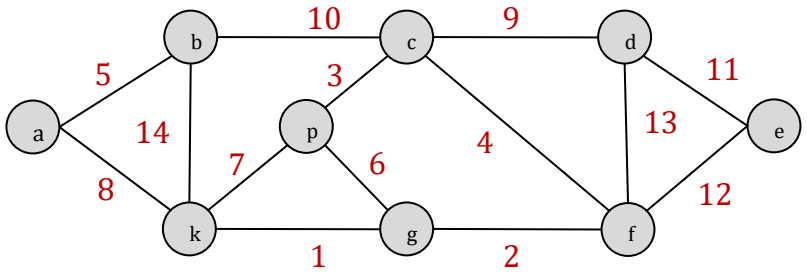


Παράδειγμα

2. Αλγόριθμος Prim

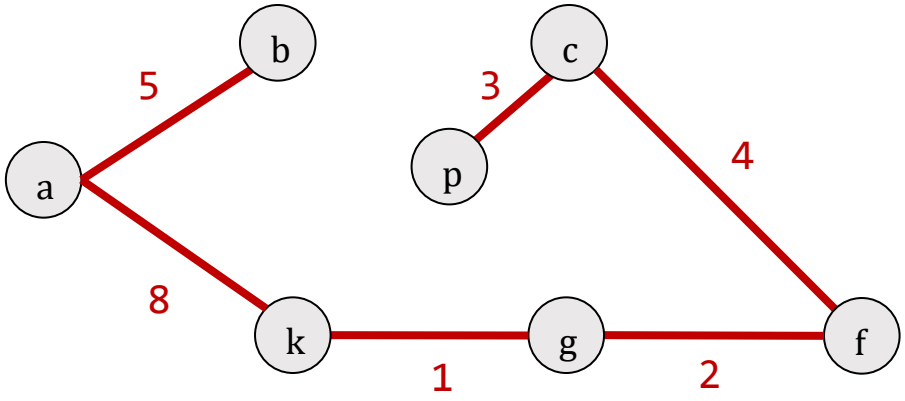


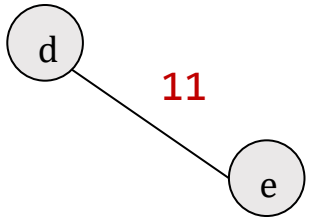


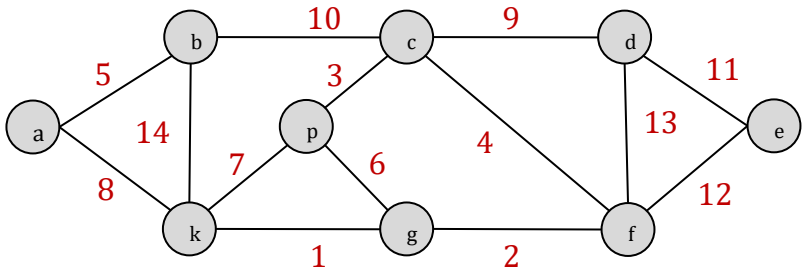


Παράδειγμα

2. Αλγόριθμος Prim

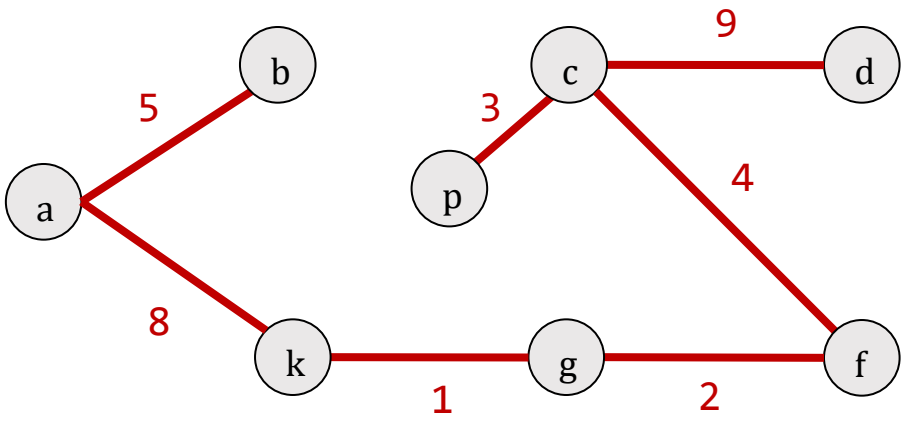




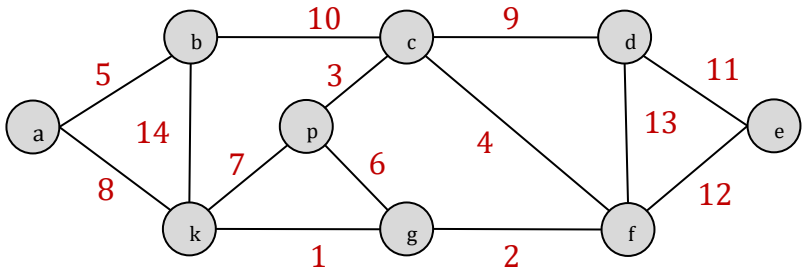


Παράδειγμα

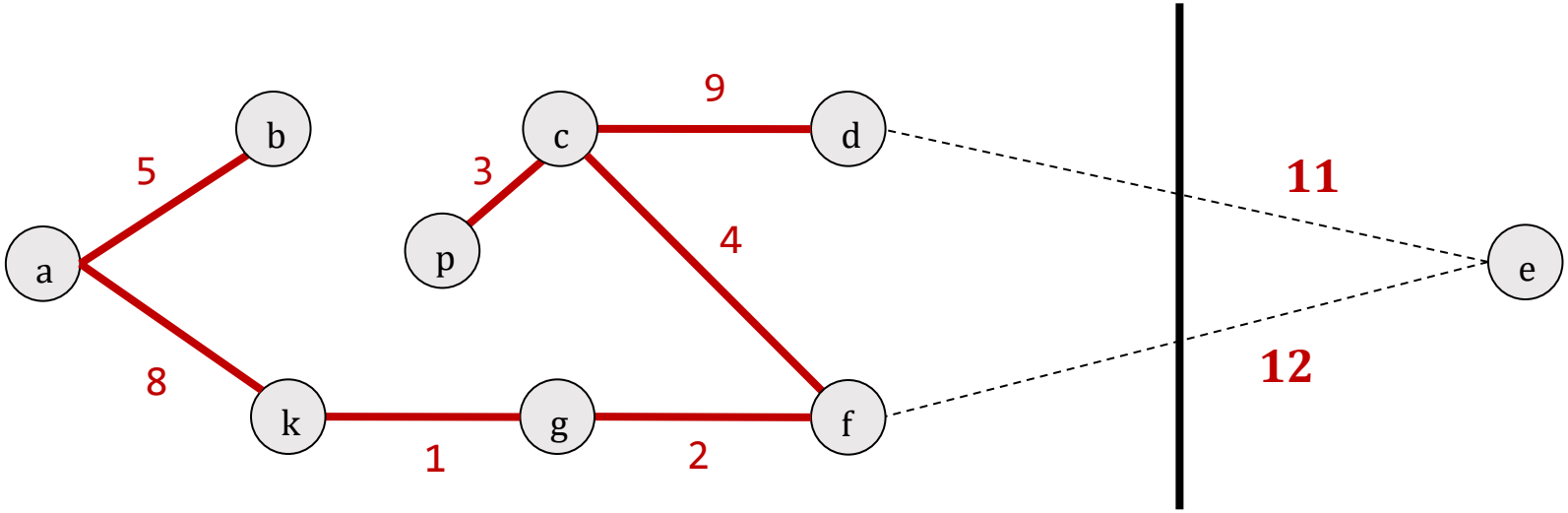
2. Αλγόριθμος Prim



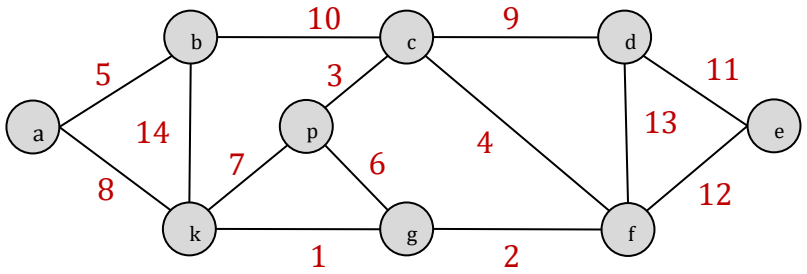
Παράδειγμα



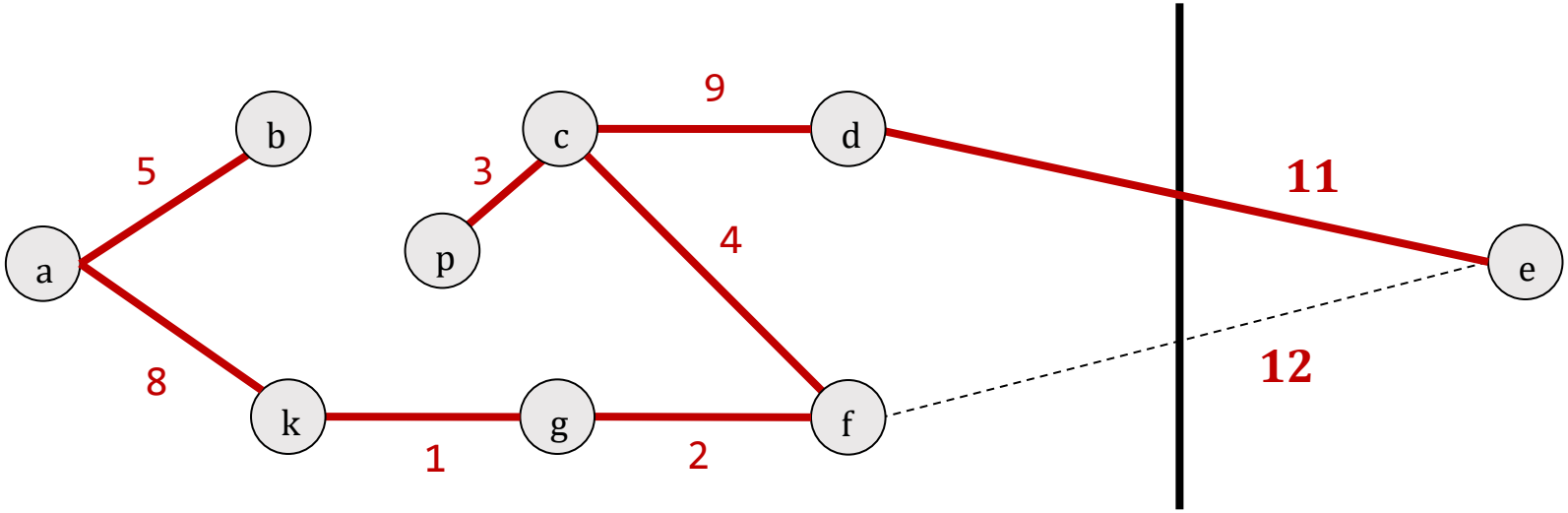
2. Αλγόριθμος Prim

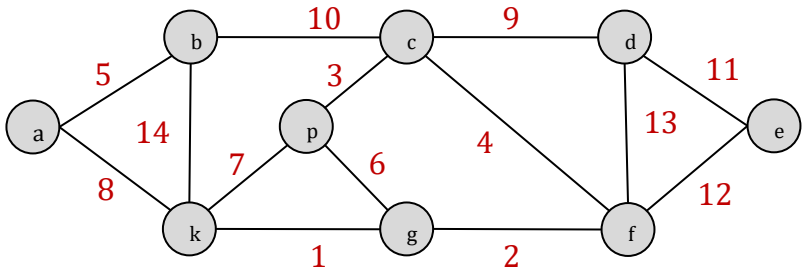


Παράδειγμα



2. Αλγόριθμος Prim

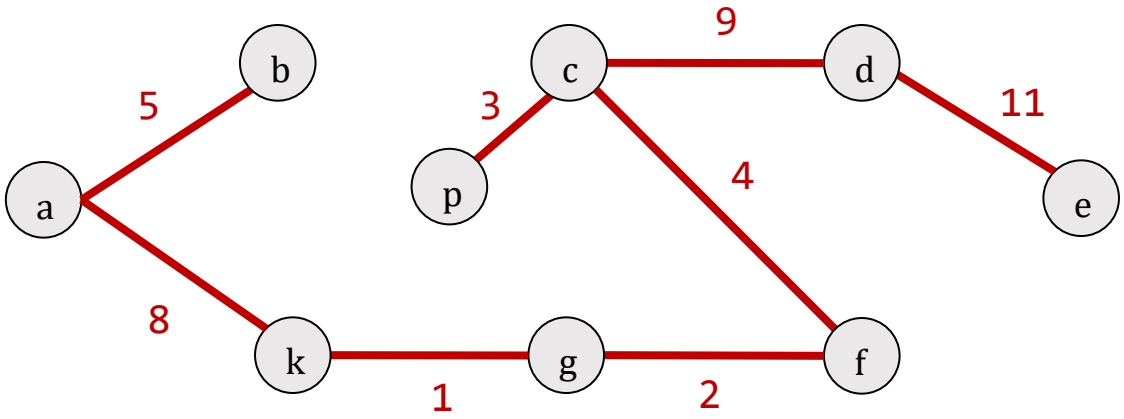




Παράδειγμα

2. Αλγόριθμος Prim

Ελάχιστο Γενετικό Δένδρο Τ



Ορθότητα Αλγορίθμων Kruskal και Prim

Η ορθότητα των δύο αλγορίθμων των **Kruskal** και **Prim** βασίζεται στην

Ιδιότητα της Αποκοπής ή Τομής (S, V-S)

Θεώρημα 1 (Ιδιότητα Αποκοπής – Cut Property)

Έστω $G = (V, E)$ συνεκτικό μη-κατευθυνόμενο έμβαρο γράφημα και έστω:

- S ένα τυχαίο υποσύνολο κόμβων που δεν είναι ούτε κενό ούτε ίσο με το V, και
- $e=(u,v) \in E$ είναι η ακμή με το **min βάρος** που έχει το ένα άκρο στο S και το άλλο στο $V-S$.

Τότε,

Κάθε MST του G περιέχει την ακμή $e=(u,v)$

Απόδειξη

- Έστω T ένα γενετικό (συνδετικό) δένδρο του G το οποίο δεν περιέχει την ακμή $e = (u,v)$.
- Θα δείξουμε ότι το T δεν είναι ένα MST (Ελάχιστο Συνδετικό Δένδρο) του G .

Επιχείρημα Ανταλλαγής:

Θα προσδιορίσουμε μια ακμή h στο T τέτοια ώστε $w(e) < w(h)$ και η οποία έχει την ιδιότητα ότι η ανταλλαγή της με την e μας δίδει ένα άλλο γενετικό δένδρο T' τέτοιο ώστε $w(T') < w(T)$!!!

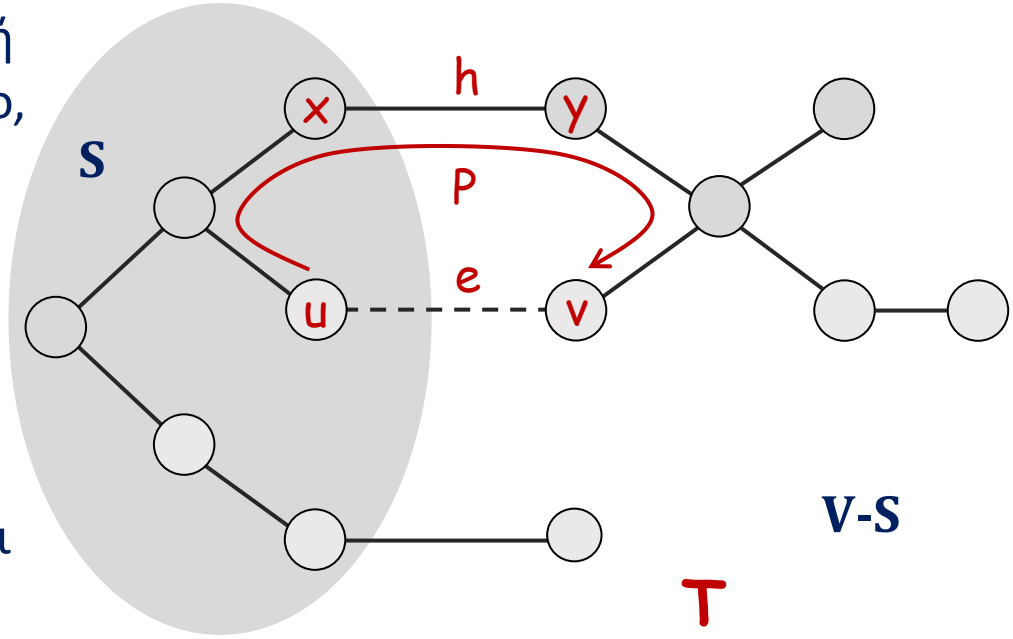
Και επομένως,

Θα έχουμε δείξει ότι το T δεν μπορεί να είναι ένα MST του G.

Απόδειξη

- Επειδή το T δεν περιέχει την ακμή e = (u,v) και είναι γενετικό δένδρο, υπάρχει στο T διαδρομή P από τον u στον v.
- Έστω y ο πρώτος κόμβος της P που ανήκει στο V-S, και x ο ακριβώς πριν από αυτόν στην διαδρομή P που ανήκει στο S, και έστω h = (x, y) η ακμή που τους συνδέει.

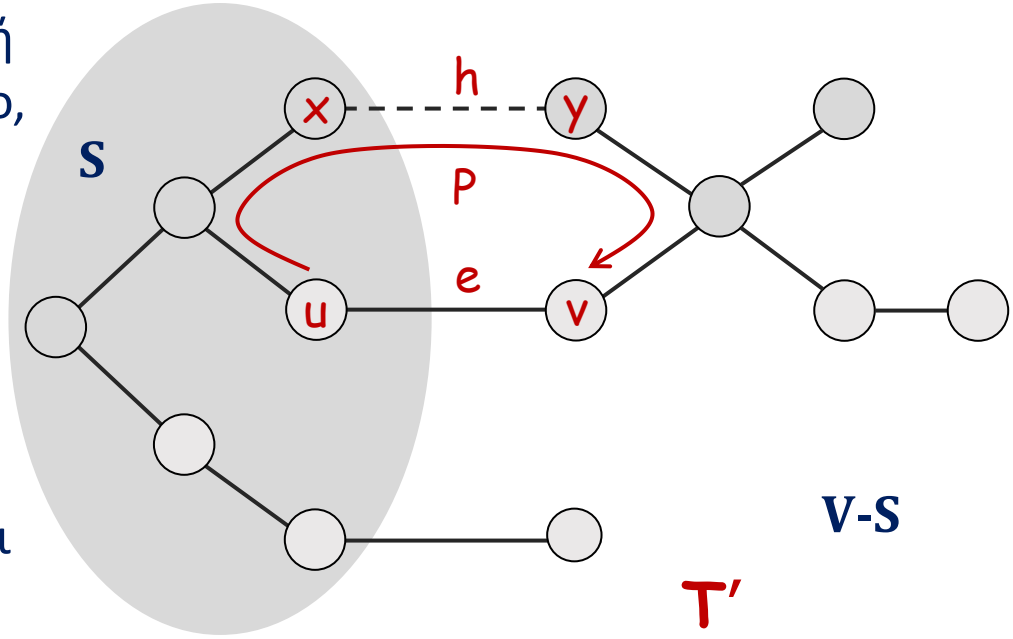
Σχήμα Απόδειξης



Απόδειξη

- Επειδή το T δεν περιέχει την ακμή e = (u,v) και είναι γενετικό δένδρο, υπάρχει στο T διαδρομή P από τον u στον v.
- Έστω y ο πρώτος κόμβος της P που ανήκει στο V-S, και x ο ακριβώς πριν από αυτόν στην διαδρομή P που ανήκει στο S, και έστω h = (x, y) η ακμή που τους συνδέει.

Σχήμα Απόδειξης



- Ανταλλάσουμε την h με την e και παίρνουμε το σύνολο ακμών T' = T - {h} ∪ {e}.
- **Ισχυρισμός**: Το T' είναι ένα γενετικό δένδρο του G !!!

Απόδειξη

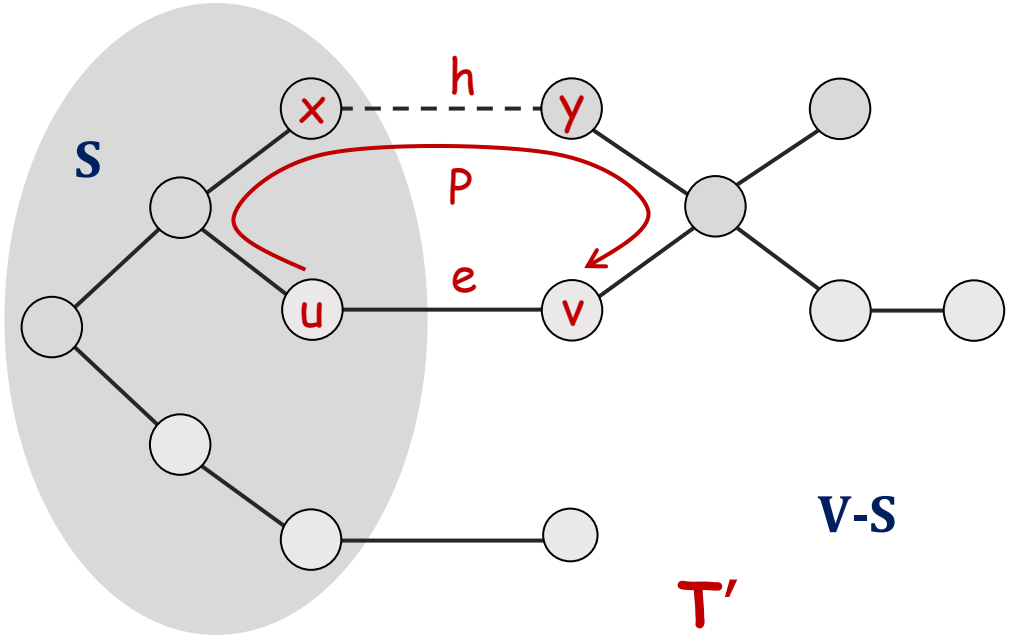
Ισχυρισμός: Το T′ είναι ένα γενετικό δένδρο.

Αρκεί να δείξουμε ότι το γράφημα (V, T′) είναι :

(1) συνεκτικό, και

(2) δεν περιέχει κύκλους.

Σχήμα Απόδειξης



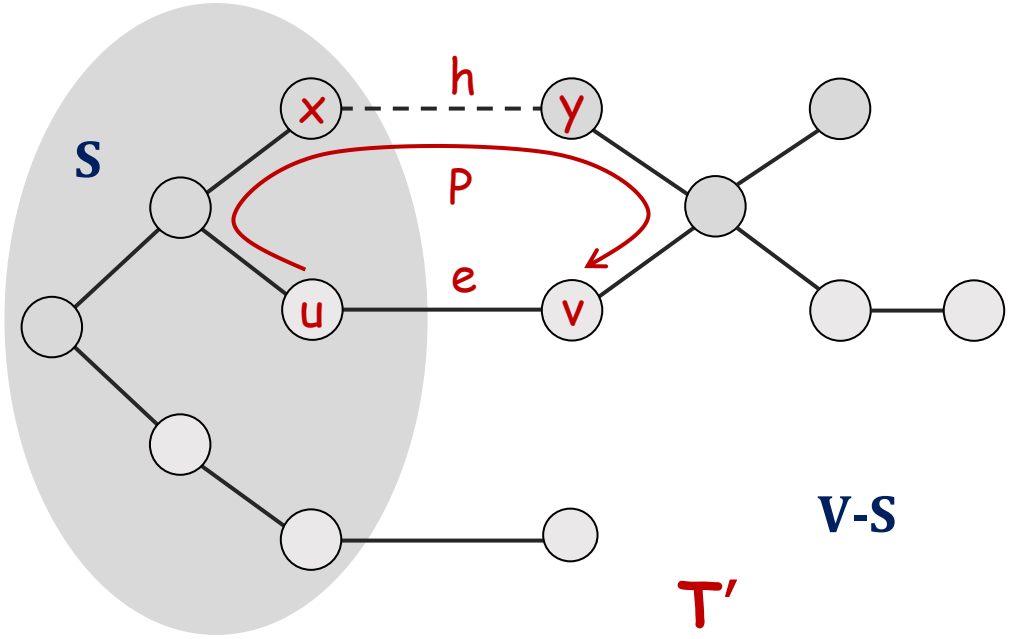
Απόδειξη

Ισχυρισμός: Το T′ είναι ένα γενετικό δένδρο.

(1) Προφανώς το γράφημα (V, T′) είναι συνεκτικό :

- ✓ Το γράφημα (V, T) είναι συνεκτικό, και
- ✓ κάθε διαδρομή του (V, T) που χρησιμοποιούσε την ακμή h= (x, y) μπορεί τώρα να επαναδρομολογηθεί στο (V, T′) ως εξής: x -> u, e, v -> y

Σχήμα Απόδειξης



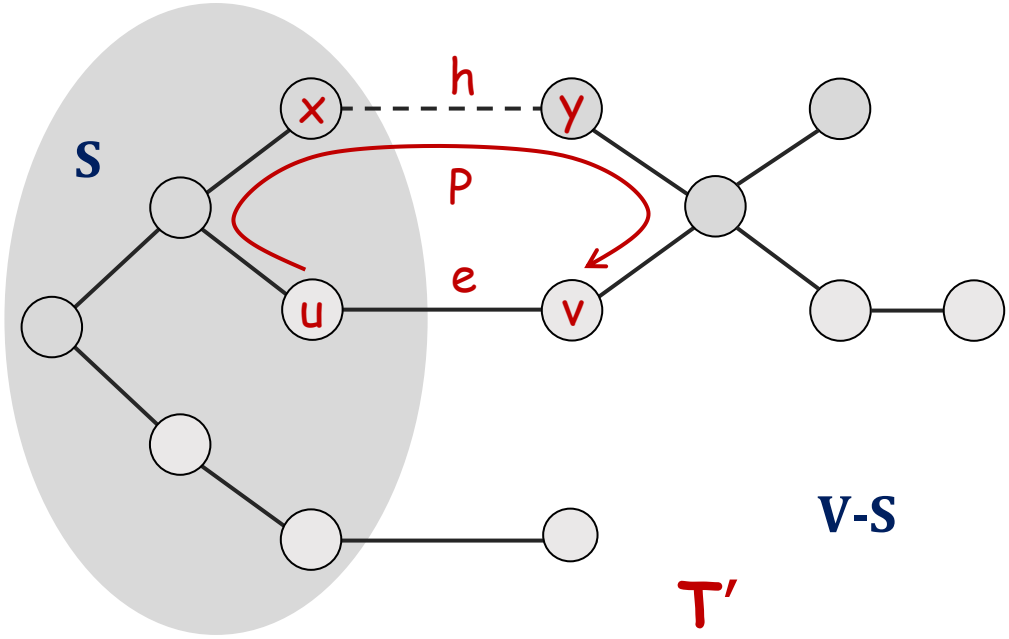
Απόδειξη

Ισχυρισμός: Το T′ είναι ένα γενετικό δένδρο.

(2) Το γράφημα (V, T′) δεν περιέχει κύκλους :

- ✓ Ο μόνος κύκλος στο (V, T′ ∪ {h}) είναι αυτός που αποτελείται από την e και την διαδρομή P, και
- ✓ αυτός ο κύκλος δεν ανήκει στο γράφημα (V, T′) λόγω της διαγραφής της ακμής h.

Σχήμα Απόδειξης



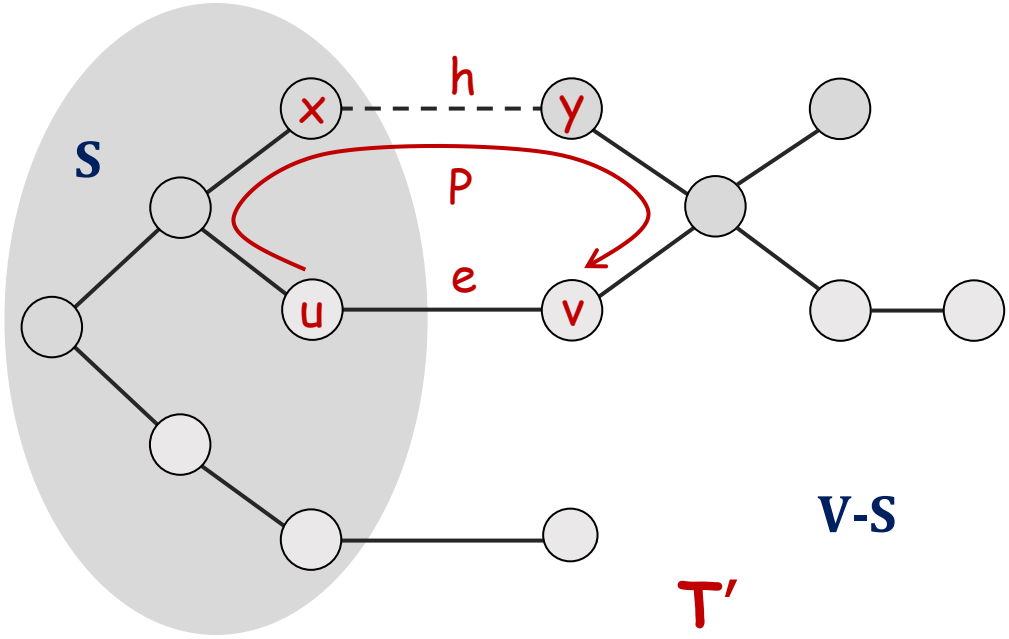
Απόδειξη

- Η ακμή h έχει το ένα άκρο στο S και το άλλο στο V-S.
- Όμως η e είναι η ακμή με το min βάρος με αυτή την ιδιότητα, και επομένως

$$w(e) < w(h)$$

Η ανισότητα είναι γνήσια, επειδή θεωρήσαμε ότι δεν υπάρχουν ακμές με το ίδιο βάρος.

Σχήμα Απόδειξης



Άρα, το συνολικό κόστος του T' είναι μικρότερο από το κόστος του T, και επομένως το T δεν μπορεί να είναι ένα MST του G. ■

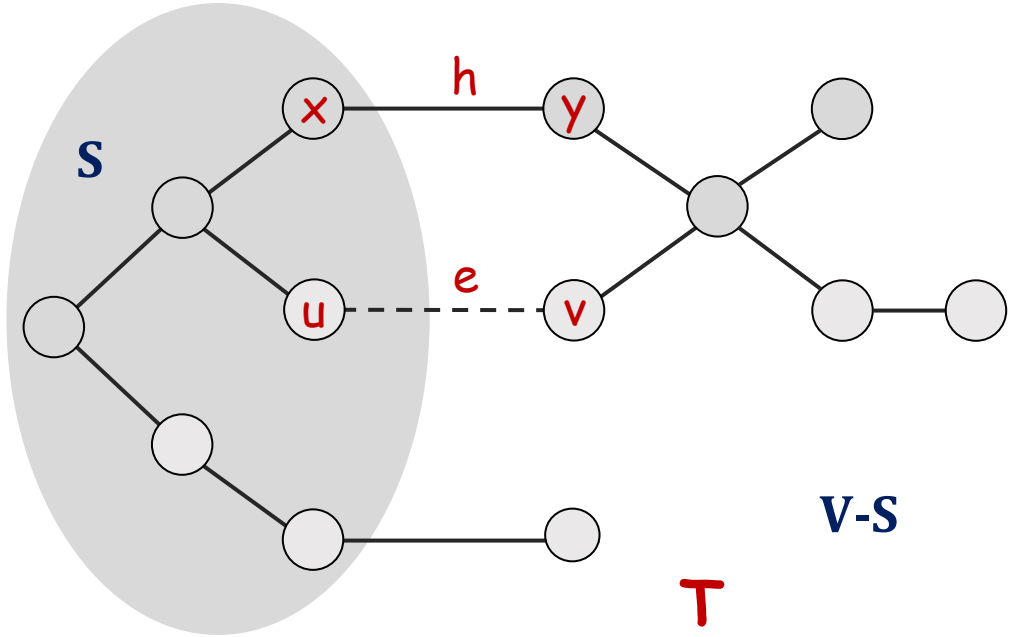
Παρατήρηση στην Απόδειξη

Προσοχή στη Απόδειξη!!!

Δεν μπορούμε να αποδείξουμε το Θεώρημα επιλέγοντας απλώς οποιαδήποτε ακμή του T που έχει το ένα άκρο στο ένα άκρο στο S και το άλλο στο V-S.

- Πρέπει να μεριμνήσουμε να βρούμε την κατάλληλη ακμή !

Σχήμα Απόδειξης



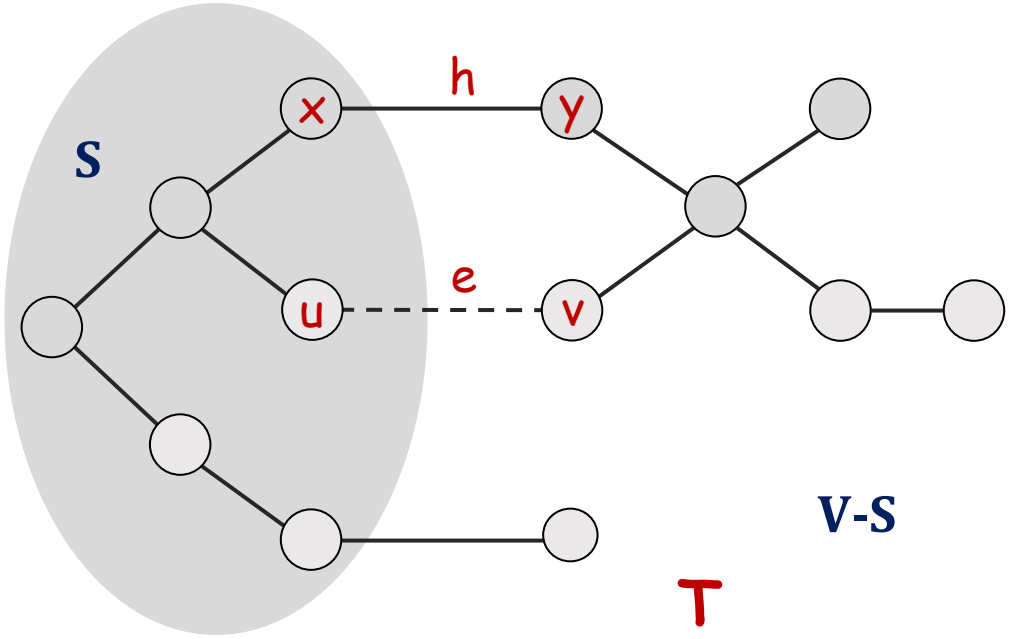
Δείτε το ακόλουθο συντομότερο αλλά *λανθασμένο* επιχείρημα για την απόδειξη του Θεωρήματος 1.

Παρατήρηση στην Απόδειξη

Λανθασμένο Επιχείρημα !!!

Έστω T ένα γενετικό δένδρο που δεν περιέχει την ακμή e = (u,v) με το min κόστος και η οποία που έχει το ένα άκρο στο S και το άλλο στο V-S.

Σχήμα Απόδειξης



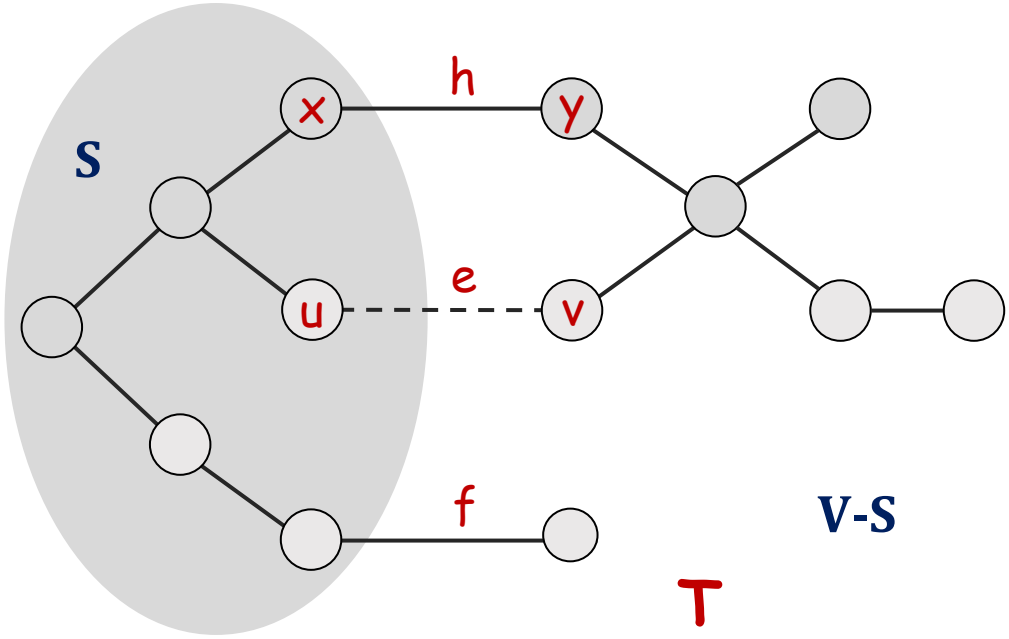
Παρατήρηση στην Απόδειξη

Λανθασμένο Επιχείρημα !!!

Έστω T ένα γενετικό δένδρο που δεν περιέχει την ακμή e = (u,v) με το min κόστος και η οποία που έχει το ένα άκρο στο S και το άλλο στο V-S.

Επειδή το T είναι γενετικό δένδρο, πρέπει να περιέχει μια ακμή f με το ένα άκρο στο S και το άλλο στο V-S.

Σχήμα Απόδειξης



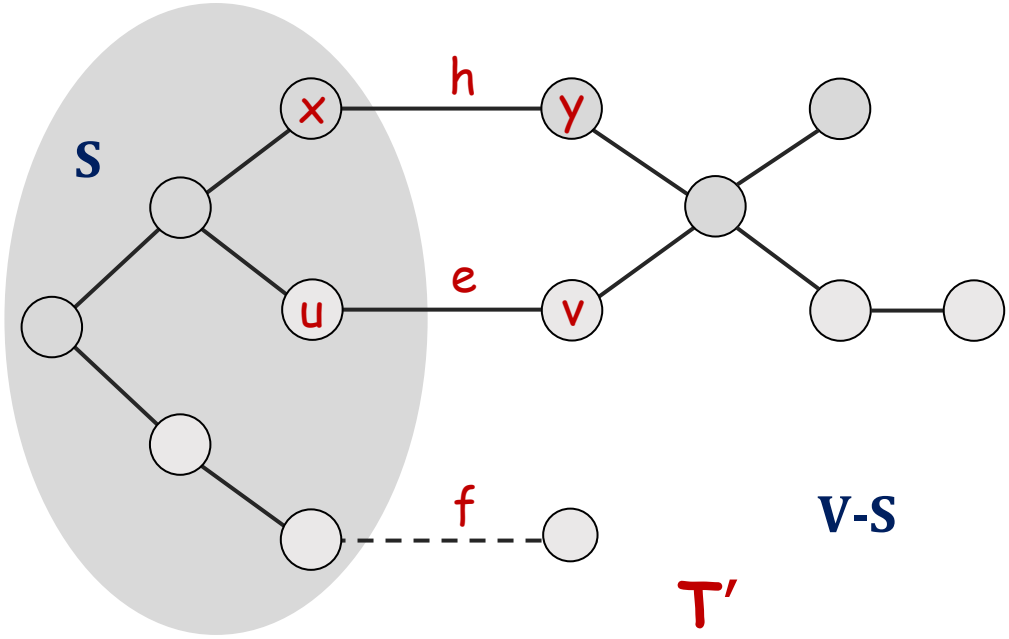
Παρατήρηση στην Απόδειξη

Λανθασμένο Επιχείρημα !!!

Έστω T ένα γενετικό δένδρο που δεν περιέχει την ακμή e = (u,v) με το min κόστος και η οποία που έχει το ένα άκρο στο S και το άλλο στο V-S.

Επειδή το T είναι γενετικό δένδρο, πρέπει να περιέχει μια ακμή f με το ένα άκρο στο S και το άλλο στο V-S.

Σχήμα Απόδειξης



Ισχύει w(e) < w(f), και επομένως,

το T' = T - {f} ∪ {e} είναι ένα γενετικό δένδρο του G τέτοιο ώστε w(T') < w(T).

Παρατήρηση στην Απόδειξη

Λανθασμένο Επιχείρημα !!!

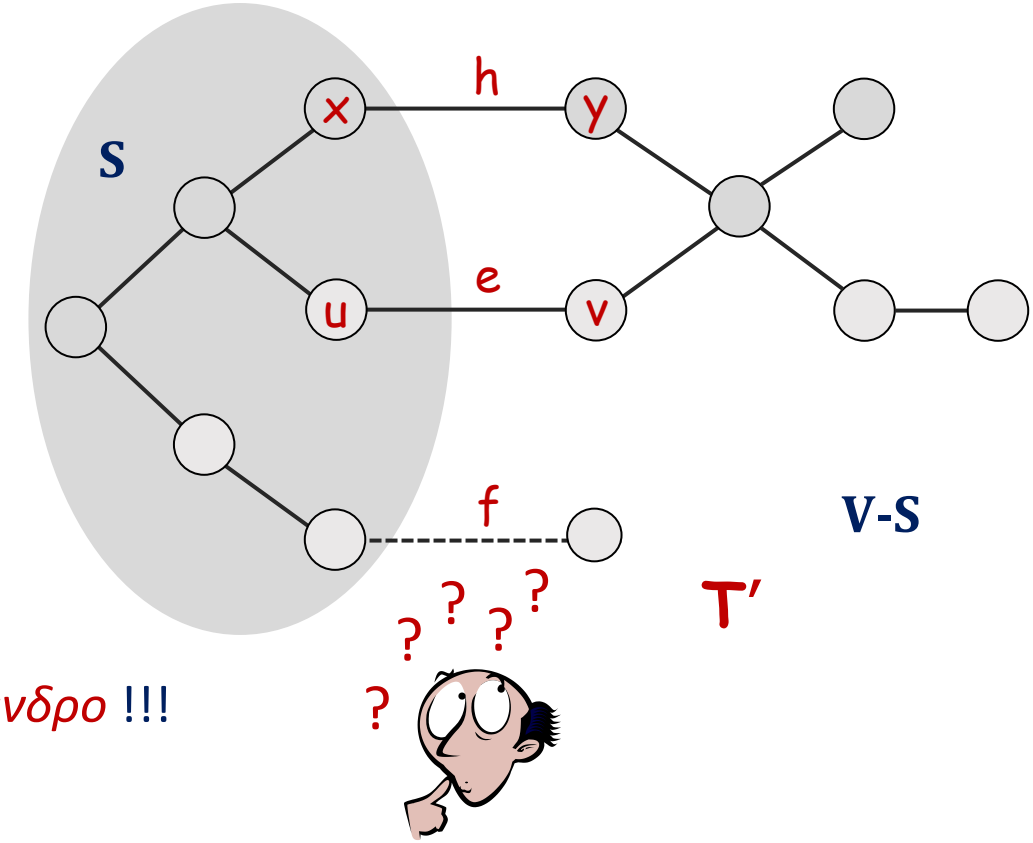
Το πρόβλημα με αυτό το επιχείρημα

- ✓ δεν βρίσκεται στον ισχυρισμό ότι υπάρχει η f,
- ✓ ούτε στο ότι το T′ είναι τέτοιο ώστε $w(T') < w(T)$.

Το πρόβλημα είναι ότι:

το T′ μπορεί να *μην είναι γενετικό δένδρο* !!!

Σχήμα Απόδειξης

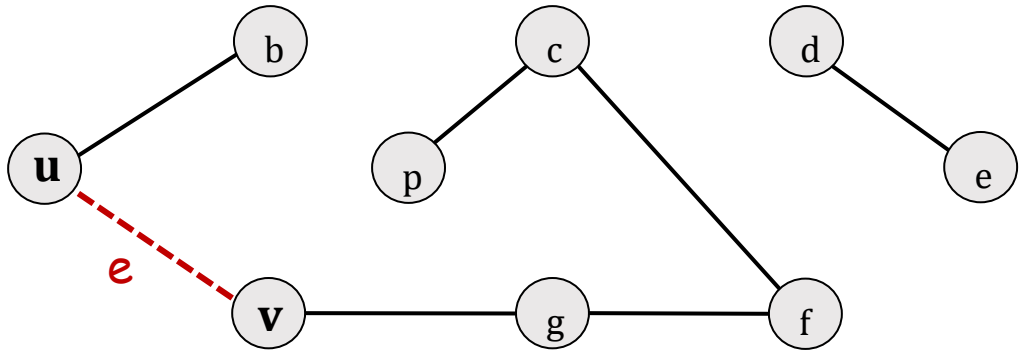


Θεώρημα (Ορθότητα Kruskal)

Ο αλγόριθμος του Kruskal παράγει ένα MST του $G = (V, E)$.

Απόδειξη

- Έστω $e = (u,v)$ η ακμή που προστέθηκε από τον αλγόριθμο σε κάποιο βήμα του.

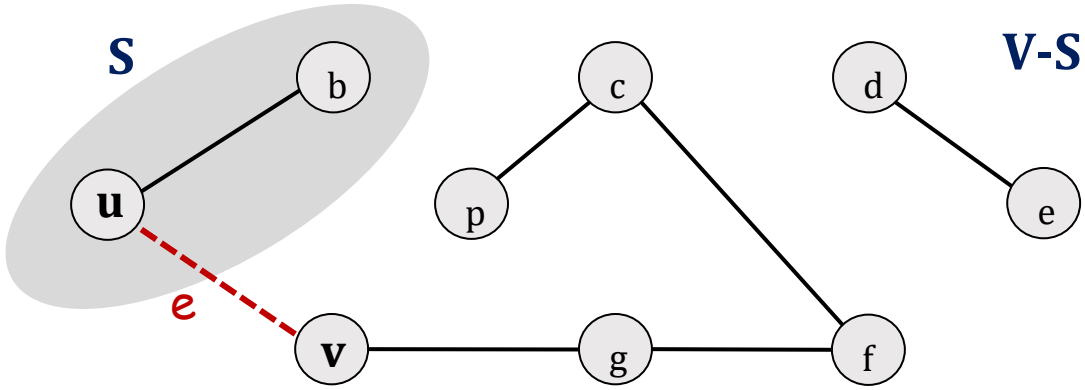


Θεώρημα (Ορθότητα Kruskal)

Ο αλγόριθμος του Kruskal παράγει ένα MST του $G = (V, E)$.

Απόδειξη

- Έστω $e = (u,v)$ η ακμή που προστέθηκε από τον αλγόριθμο σε κάποιο βήμα του, έστω i.



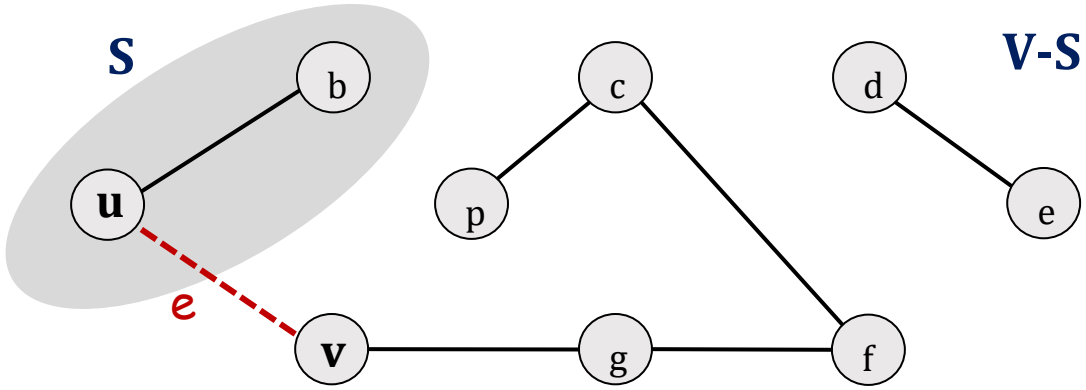
- Έστω έστω S το σύνολο όλων των κόμβων που είναι προσβάσιμοι από τον κόμβο u ακριβώς πριν την προσθήκη της e.
- Προφανώς $u \in S$, αλλά όμως $v \notin S$ διότι η προσθήκη της δεν δημιουργεί κύκλο, και επομένως $v \in V\text{-}S$.

Θεώρημα (Ορθότητα Kruskal)

Ο αλγόριθμος του Kruskal παράγει ένα MST του $G = (V, E)$.

Απόδειξη

- Έστω $e = (u,v)$ η ακμή που προστέθηκε από τον αλγόριθμο σε κάποιο βήμα του, έστω i.



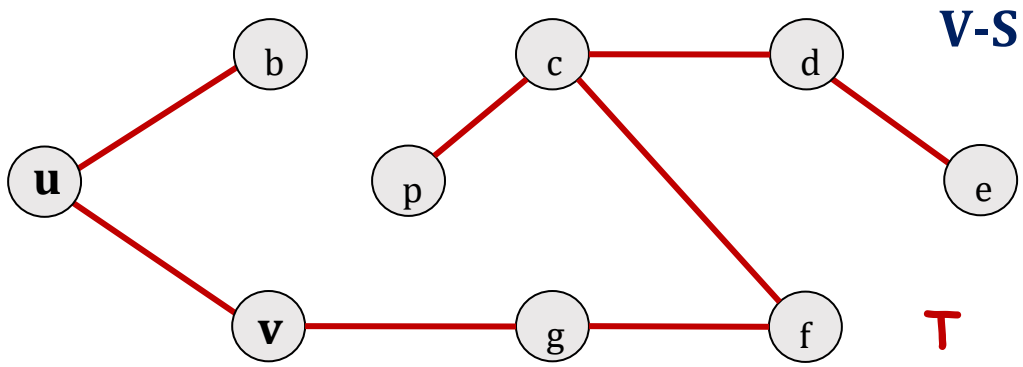
- Επιπρόσθετα, μέχρι το Βήμα i ο αλγόριθμος δεν έχει συναντήσει καμία άλλη ακμή από το S στο V-S, γιατί μια τέτοια ακμή δεν δημιουργεί κύκλο και επομένως εάν την είχε συναντήσει θα την είχε προσθέσει.
- Έτσι η e είναι η ακμή min βάρους με το ένα άκρο στο S και το άλλο στο V-S, και άρα από Θεώρημα 1 η ακμή e ανήκει σε κάθε MST του G.

Θεώρημα (Ορθότητα Kruskal)

Ο αλγόριθμος του Kruskal παράγει ένα MST του $G = (V, E)$.

Απόδειξη

- Αρκεί να αποδείξουμε ότι όταν ο αλγόριθμος τερματίσει, το T που επιστρέφει είναι ένα γενετικό δένδρο του G.



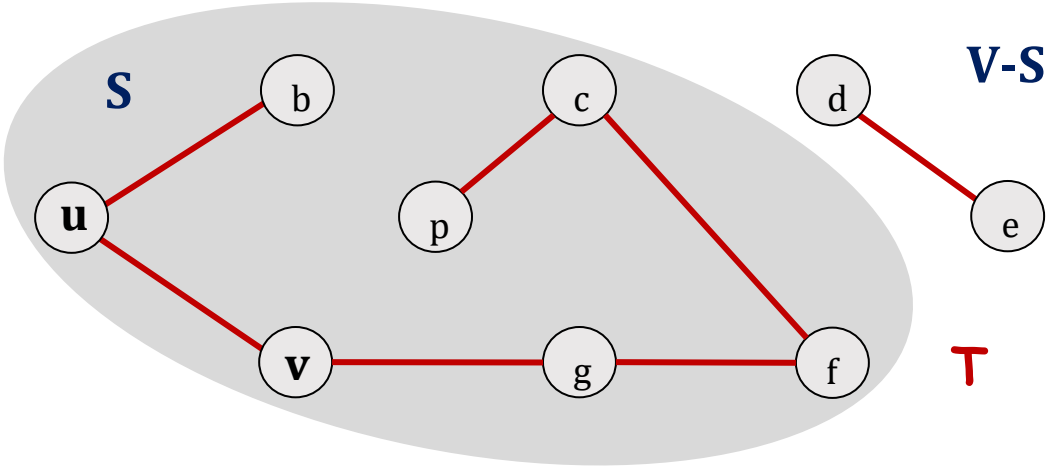
- Προφανώς το T δεν περιέχει κύκλους, από τον σχεδιασμό του αλγόριθμου.

Θεώρημα (Ορθότητα Kruskal)

Ο αλγόριθμος του Kruskal παράγει ένα MST του $G = (V, E)$.

Απόδειξη

- Αρκεί να αποδείξουμε ότι όταν ο αλγόριθμος τερματίσει, το T που επιστρέφει είναι ένα γενετικό δένδρο του G.



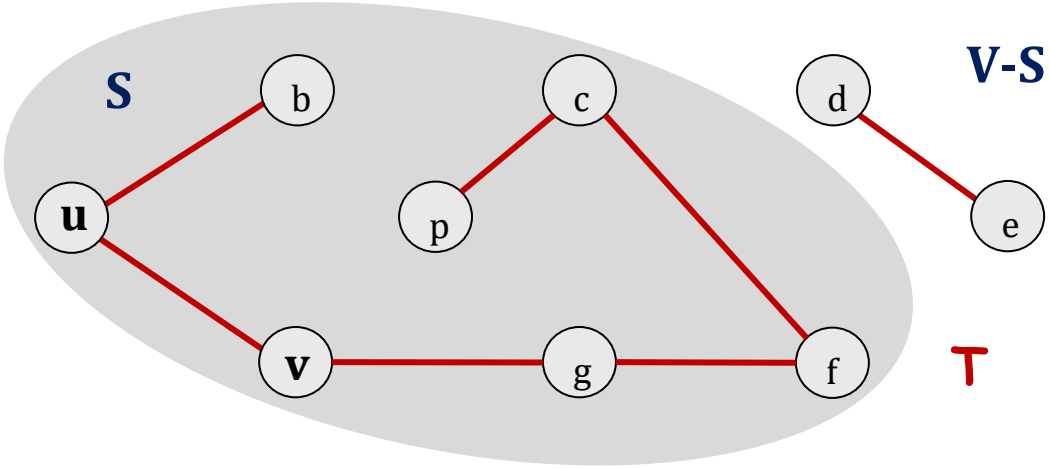
- Προφανώς το T δεν περιέχει κύκλους, από τον σχεδιασμό του αλγόριθμου.
- Εάν το T δεν ήταν συνεκτικό, τότε θα υπήρχε ένα μη-κενό σύνολο κόμβων S (όχι ίσο με το V) τέτοιο ώστε να μην υπάρχει ακμή από το S στο V-S.

Θεώρημα (Ορθότητα Kruskal)

Ο αλγόριθμος του Kruskal παράγει ένα MST του $G = (V, E)$.

Απόδειξη

- Αρκεί να αποδείξουμε ότι όταν ο αλγόριθμος τερματίσει, το T που επιστρέφει είναι ένα γενετικό δένδρο του G.



- Όμως αυτό αντιφάσκει με την λειτουργία του αλγόριθμου:

  Επειδή το γράφημα G είναι συνεκτικό, υπάρχει τουλάχιστον μια ακμή μεταξύ των S και $V-S$, και επομένως ο αλγόριθμος προσθέτει από αυτές τις ακμές την πρώτη που θα συναντήσει. ■

Θεώρημα (Ορθότητα Prim)

Ο αλγόριθμος του Prim παράγει ένα MST του $G = (V, E)$.

Απόδειξη

- Θα δείξουμε ότι ο αλγόριθμος προσθέτει μόνο ακμές που ανήκουν σε κάθε MST του G.
- Σε κάθε επανάληψη ο αλγόριθμος έχει δημιουργήσει ένα σύνολο $S \subseteq V$, για το οποίο έχει κατασκευάσει ένα μερικό γενετικό δένδρο, και επιλέγει την ακμή $e = (u,v)$ min βάρους με το ένα άκρο στο S και το άλλο στο $V-S$.
- Άρα από Θεώρημα 1 η ακμή e ανήκει σε κάθε MST του G.
- Είναι απλό να αποδείξουμε ότι όταν ο αλγόριθμος τερματίσει, το T που επιστρέφει είναι ένα γενετικό δένδρο του G και άρα ένα MST αυτού. ■

Αλγόριθμος Kruskal

MST-Kruskal(G, w)

1. $A \leftarrow \varnothing$
2. Για κάθε κόμβο $v \in V$ εκτέλεσε MAKE-SET(v)
3. Ταξινόμησε τις ακμές κατά μη-φθίνουσα τάξη βαρών
4. Για κάθε $(u,v) \in E$, με τη σειρά της ταξινόμησης, εκτέλεσε:
5. Εάν FIND-SET(u) $\neq$ FIND-SET(v) τότε
6. $A \leftarrow A \cup \{(u,v)\}$
7. UNION(u,v)
8. Επίστρεψε $T \leftarrow A$

Πολυπλοκότητα Kruskal & Prim

- **Πολυπλοκότητα Kruskal**
  - **1-2:** **$O(n)$**, $n = |V|$
  - **3:** **$O(m \log m)$**
  - **4-7:** Εκτελούνται **m** `Union-Find` λειτουργίες.

    Επομένως, η πολυπλοκότητα του βήματος 2 είναι **$O(m \cdot \alpha(m,n))$**

    όπου,

    $\alpha(m,n)$ είναι η αντίστροφη της συνάρτησης του Ackermann.

1. $A \leftarrow \varnothing$
2. Για κάθε κόμβο $v \in V$ εκτέλεσε MAKE-SET(v)
3. Ταξινόμησε τις ακμές κατά μη-φθίνουσα τάξη βαρών
4. Για κάθε $(u,v) \in E$, με τη σειρά της ταξινόμησης, εκτέλεσε:
5. Εάν FIND-SET(u) $\neq$ FIND-SET(v) τότε
6. $A \leftarrow A \cup \{(u,v)\}$
7. UNION(u,v)
8. Επίστρεψε $T \leftarrow A$

Αλγόριθμος Prim

MST-Prim(G, w)

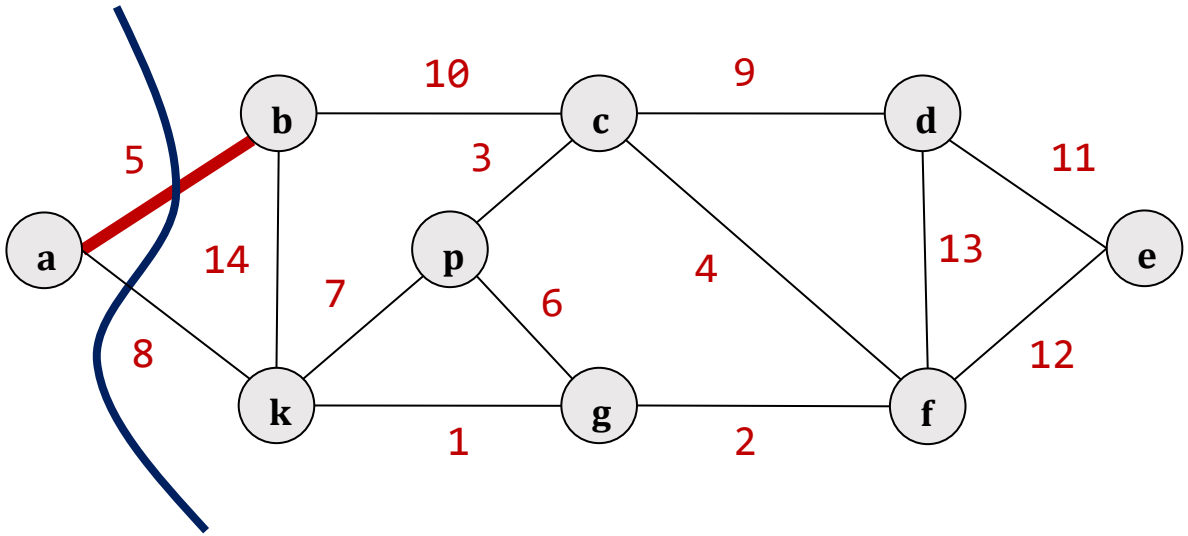
1. $T \leftarrow \varnothing$
2. $S \leftarrow$ u, όπου u τυχαίος κόμβος του G
3. Εφόσον $S \neq V$ εκτέλεσε:
4. Βρες μια ακμή (u,v) με min-βάρος : $u \in S$ και $v \in V\text{-}S$
5. $T \leftarrow T \cup \{(u,v)\}$
6. $B \leftarrow B \cup \{v\}$
7. Επίστρεψε T

Πολυπλοκότητα Kruskal & Prim

Πολυπλοκότητα Prim

- ✓ Πολυπλοκότητα: $\mathbf{O(n \cdot m)}$
- ✓ Εάν υλοποιηθεί με χρήση δομής Heap, τότε $\mathbf{O(m \cdot \log n)}$

1. $T \leftarrow \varnothing$
2. $S \leftarrow$ u, όπου u τυχαίος κόμβος του G
3. Εφόσον $S \neq V$ εκτέλεσε:
4. Βρες μια ακμή (u,v) με min-βάρος : $u \in S$ και $v \in V\text{-}S$
5. $T \leftarrow T \cup \{(u,v)\}$
6. $B \leftarrow B \cup \{v\}$
7. Επίστρεψε T

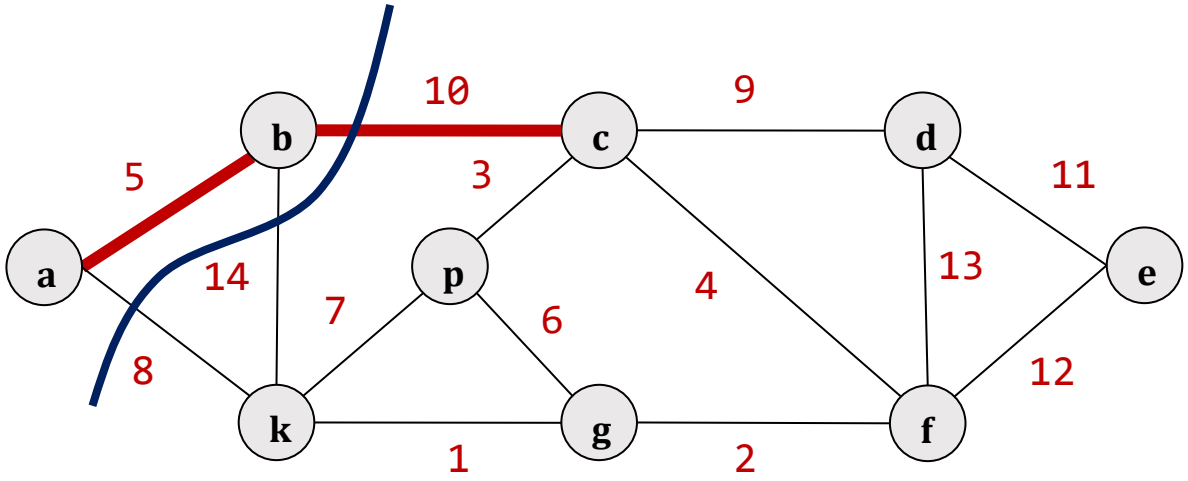


Πολυπλοκότητα Kruskal & Prim

Πολυπλοκότητα Prim

- ✓ Πολυπλοκότητα: $\mathbf{O(n \cdot m)}$
- ✓ Εάν υλοποιηθεί με χρήση δομής Heap, τότε $\mathbf{O(m \cdot \log n)}$

1. $T \leftarrow \varnothing$
2. $S \leftarrow$ u, όπου u τυχαίος κόμβος του G
3. Εφόσον $S \neq V$ εκτέλεσε:
4. Βρες μια ακμή (u,v) με min-βάρος : $u \in S$ και $v \in V\text{-}S$
5. $T \leftarrow T \cup \{(u,v)\}$
6. $B \leftarrow B \cup \{v\}$
7. Επίστρεψε T

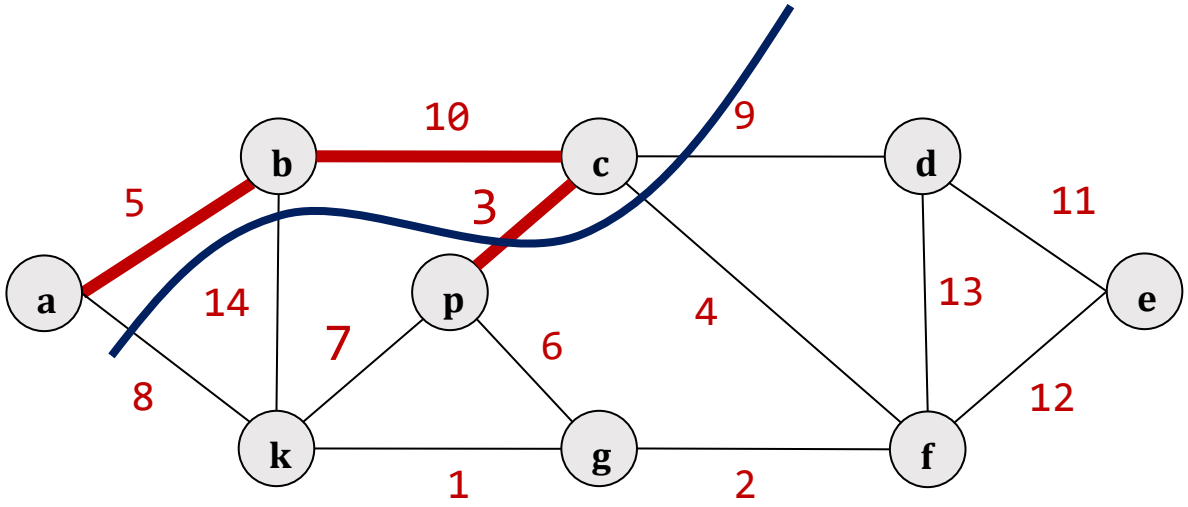


Πολυπλοκότητα Kruskal & Prim

Πολυπλοκότητα Prim

- ✓ Πολυπλοκότητα: $\mathbf{O(n \cdot m)}$
- ✓ Εάν υλοποιηθεί με χρήση δομής Heap, τότε $\mathbf{O(m \cdot \log n)}$

1. $T \leftarrow \varnothing$
2. $S \leftarrow$ u, όπου u τυχαίος κόμβος του G
3. Εφόσον $S \neq V$ εκτέλεσε:
4. Βρες μια ακμή (u,v) με min-βάρος : $u \in S$ και $v \in V\text{-}S$
5. $T \leftarrow T \cup \{(u,v)\}$
6. $B \leftarrow B \cup \{v\}$
7. Επίστρεψε T



Αλγόριθμος Boruvka

- ✓ Λειτουργεί σε γύρους.
  Αρχικά κάθε κόμβος είναι μόνος του μία συνιστώσα.
- ✓ Σε κάθε γύρο, **κάθε** συνεκτική συνιστώσα συνδέεται με την ελαφρύτερη δυνατή ακμή με κάποια από τις υπόλοιπες συνιστώσες.
  Χρειάζεται τρόπος επίλυσης 'ισοπαλιών'.

Πολυπλοκότητα:

$$O(m \cdot \log n)$$

Νύξη: σε κάθε γύρο το πλήθος συνιστωσών μειώνεται στο μισό.

Ερώτηση 1

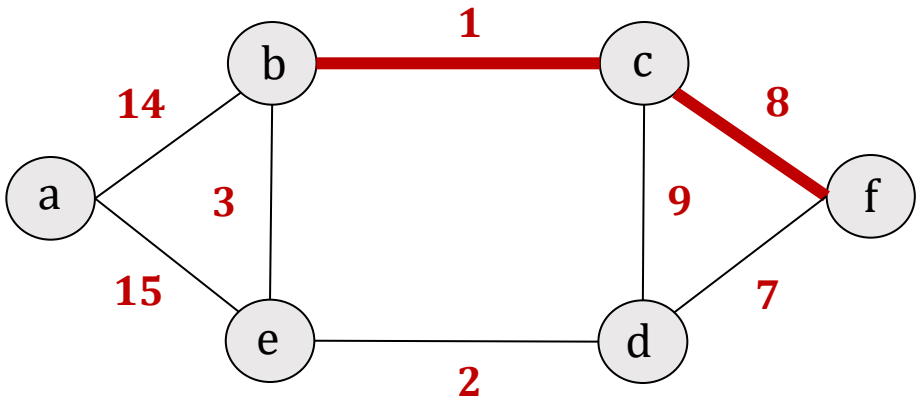
Μπορούμε να χρησιμοποιήσουμε έναν αλγόριθμο MST (για παράδειγμα του Prim) για τον υπολογισμό ΕΛΑΧΙΣΤΩΝ ΔΙΑΔΡΟΜΩΝ σε γράφημα από ένα κόμβο **s** ?

Όχι !!!

Γιατί ?

Ελάχιστα Συνδετικά Δέντρα – Παρατηρήσεις

Αντιπαράδειγμα !!!



Ελάχιστη διαδρομή

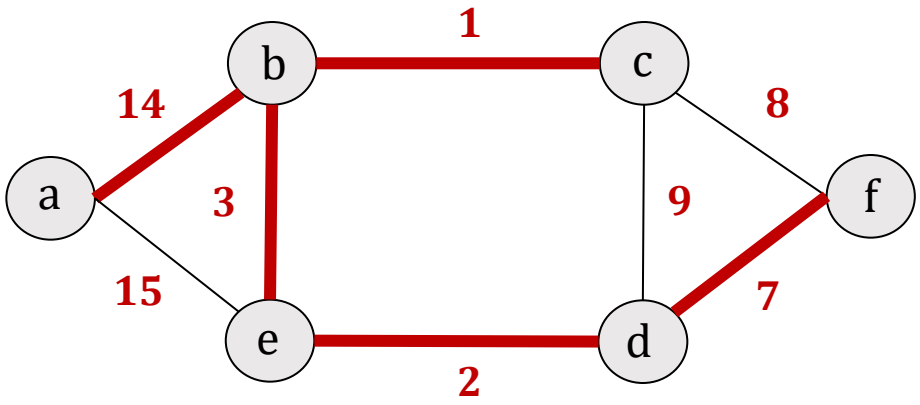
$P : b \rightsquigarrow f$

$w(P) = 9$

$P : (b, c, f)$

Ελάχιστα Συνδετικά Δέντρα – Παρατηρήσεις

Αντιπαράδειγμα !!!



Ελάχιστη διαδρομή ?

$P : b \rightsquigarrow f$

$w(P) = 12$

$P : (b, e, d, f)$

Αλγόριθμος Prim

Διαφορά των Αλγορίθμων Prim - Dijkstra

Και οι δύο αλγόριθμοι σε κάθε βήμα επιλέγουν μια ακμή που συνδέει έναν κόμβο ο οποίος χαρακτηρίζεται "εγγύτερος" !!!

Όμως,

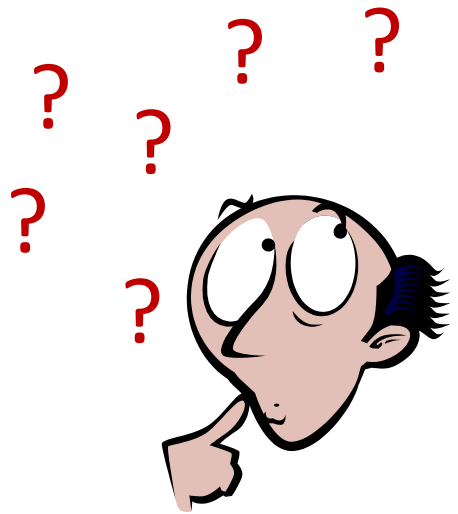
- ✓ **Prim** **"εγγύτερος"** ⇒ **εγγύτερος στο ΔΕΝΔΡΟ Τ**
- ✓ **Dijkstra** **"εγγύτερος"** ⇒ **εγγύτερος στο ΚΟΜΒΟ s**

Ερώτηση 2

Μπορούμε να χρησιμοποιήσουμε την τεχνική **Διαίρει-και-Βασίλευε** για τον υπολογισμό ενός MST ενός γραφήματος ?

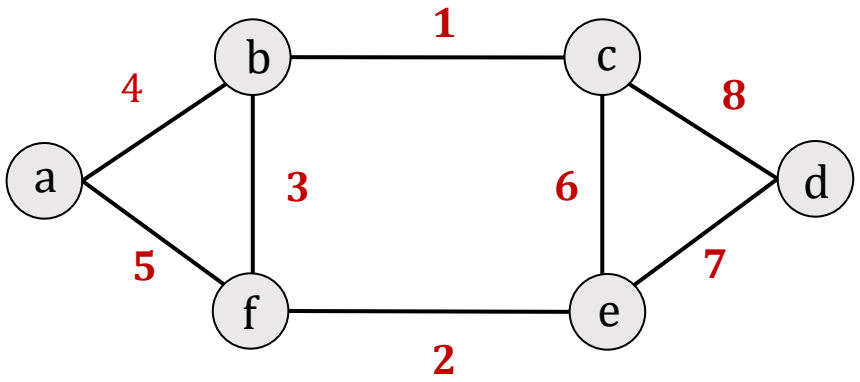
Όχι !!!

Γιατί ?



Ελάχιστα Συνδετικά Δέντρα – Παρατηρήσεις

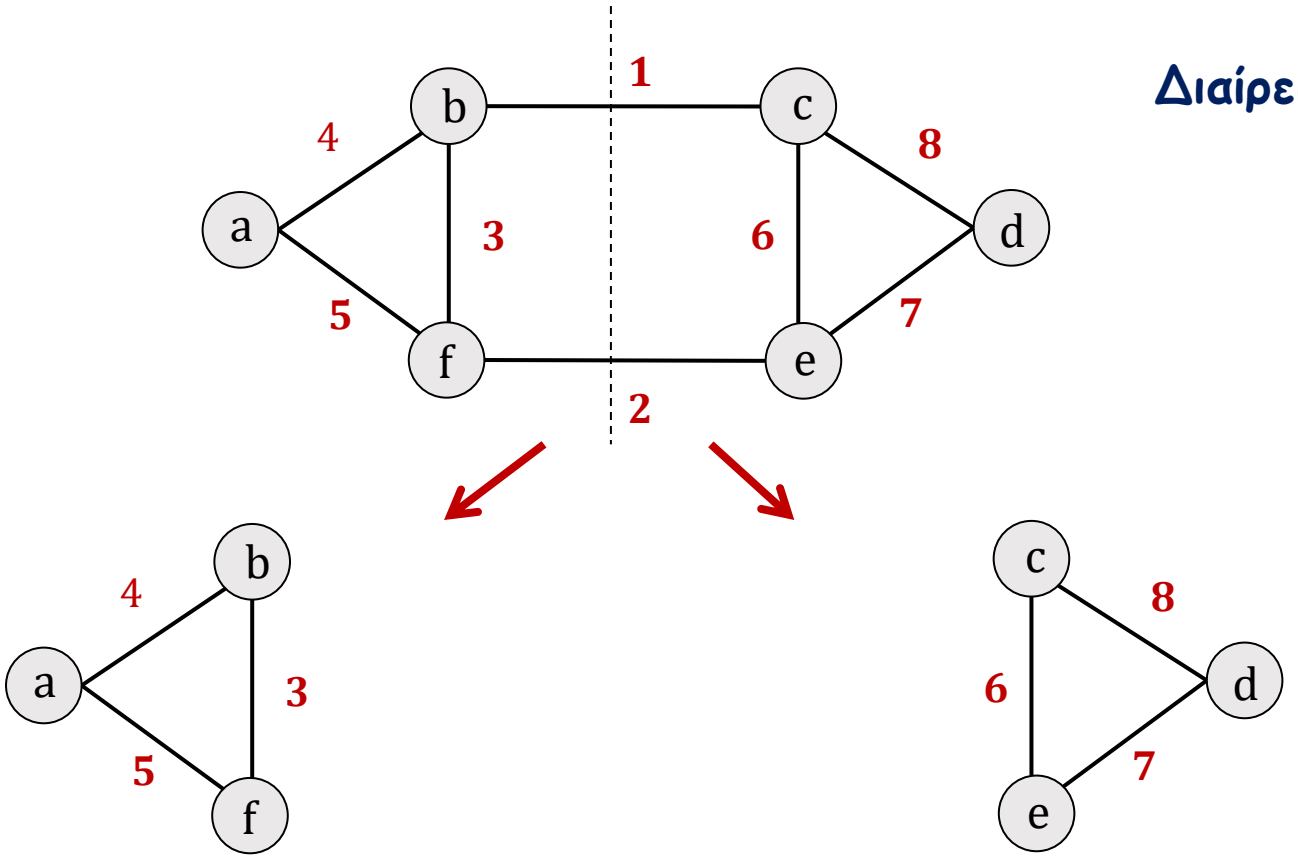
Αντιπαράδειγμα !!!



Διαίρει-και-Βασίλευε

Ελάχιστα Συνδετικά Δέντρα – Παρατηρήσεις

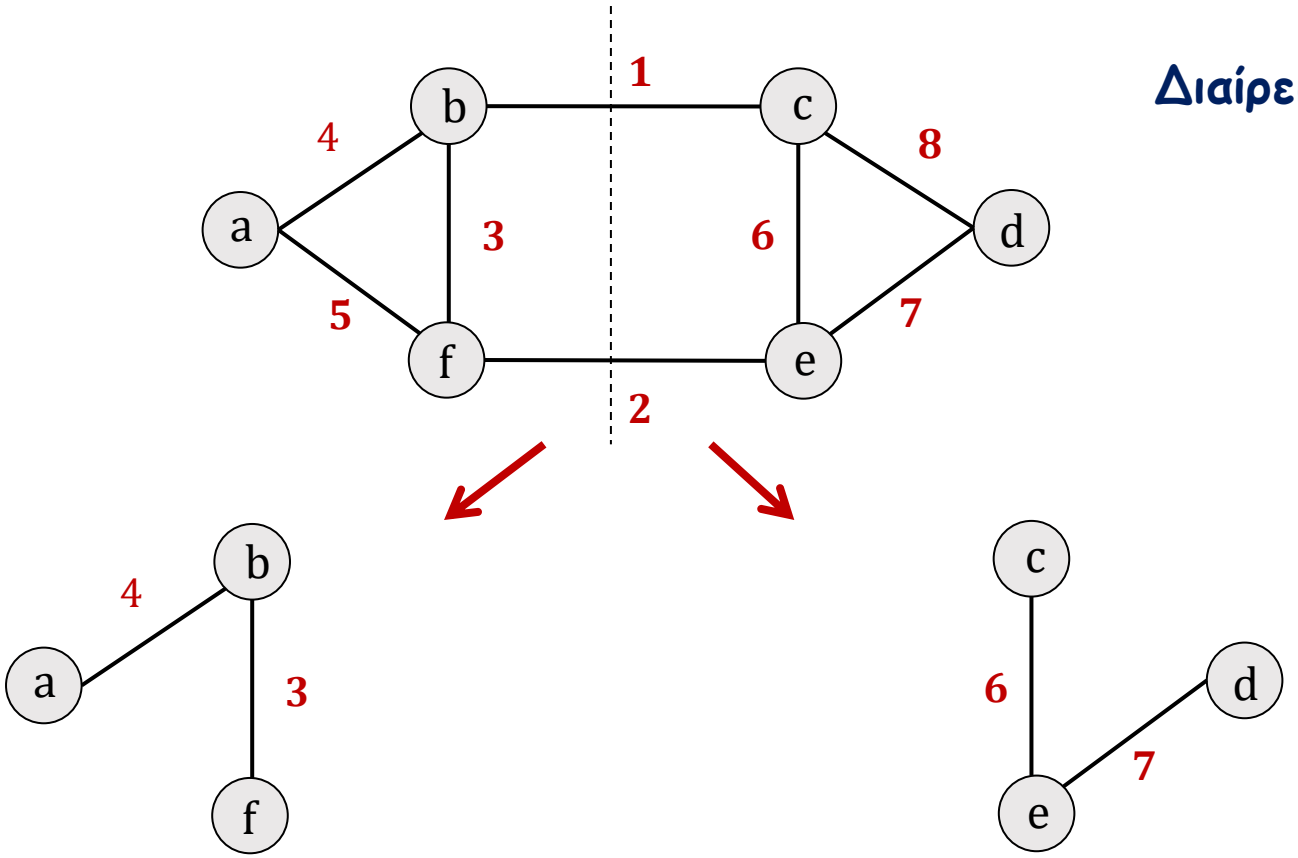
Αντιπαράδειγμα !!!



Διαίρει-και-Βασίλευε

Ελάχιστα Συνδετικά Δέντρα – Παρατηρήσεις

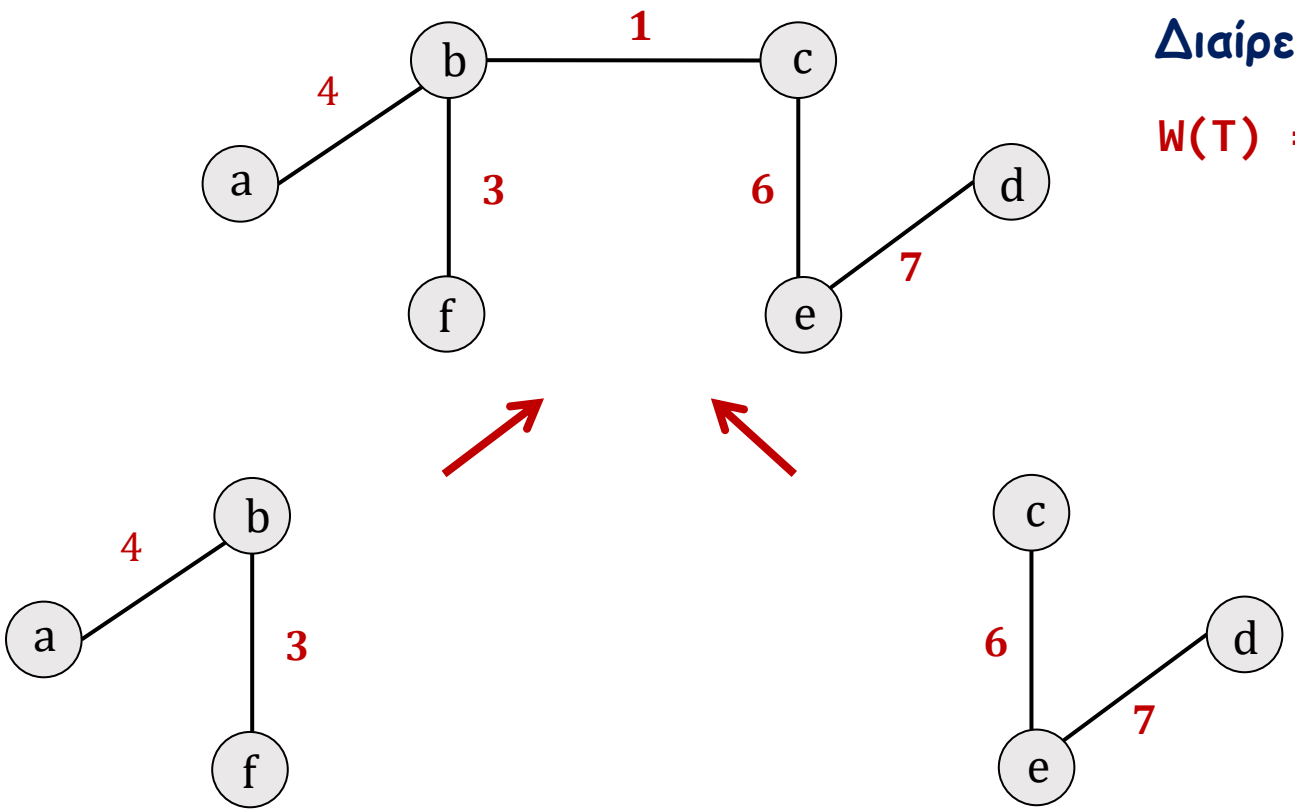
Αντιπαράδειγμα !!!



Διαίρει-και-Βασίλευε

Ελάχιστα Συνδετικά Δέντρα – Παρατηρήσεις

Αντιπαράδειγμα !!!

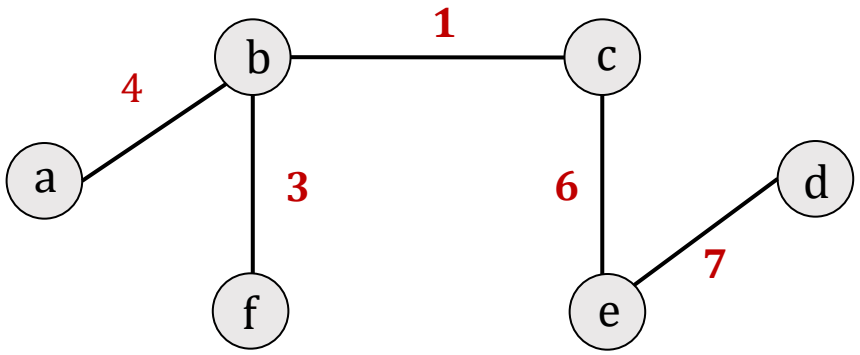


Διαίρει-και-Βασίλευε

W(T) = 21

Ελάχιστα Συνδετικά Δέντρα – Παρατηρήσεις

Αντιπαράδειγμα !!!

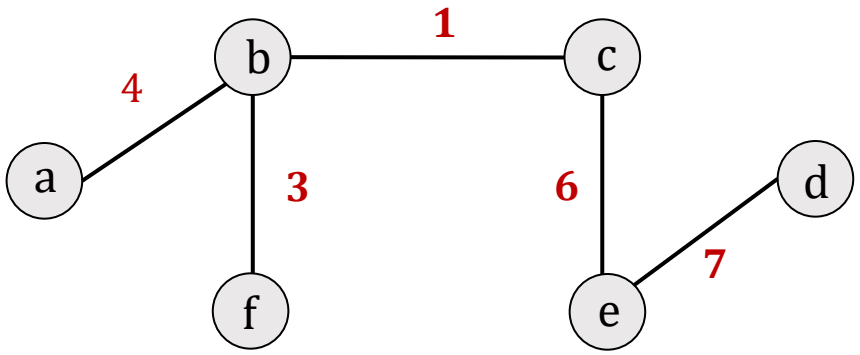


Διαίρει-και-Βασίλευε

W(T) = 21

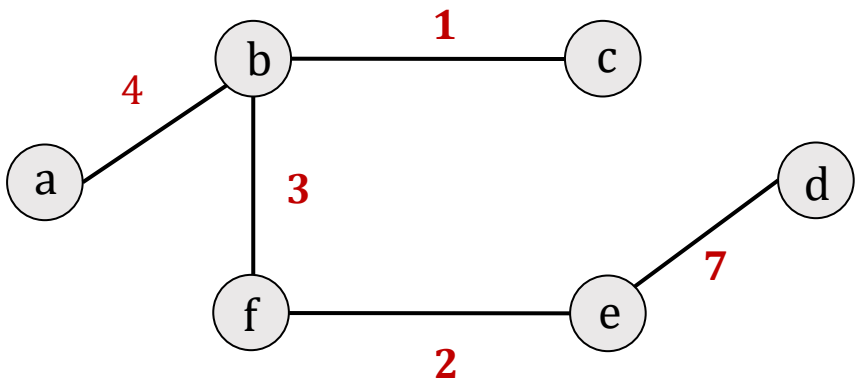
Ελάχιστα Συνδετικά Δέντρα – Παρατηρήσεις

Αντιπαράδειγμα !!!



Διαίρει-και-Βασίλευε

W(T) = 21



Kruskal

W(T) = 17

Ευχαριστώ για τη Συμμετοχή σας !!!

Σταύρος Δ. Νικολόπουλος

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Πανεπιστήμιο Ιωαννίνων

Webpage: www.cs.uoi.gr/~stavros