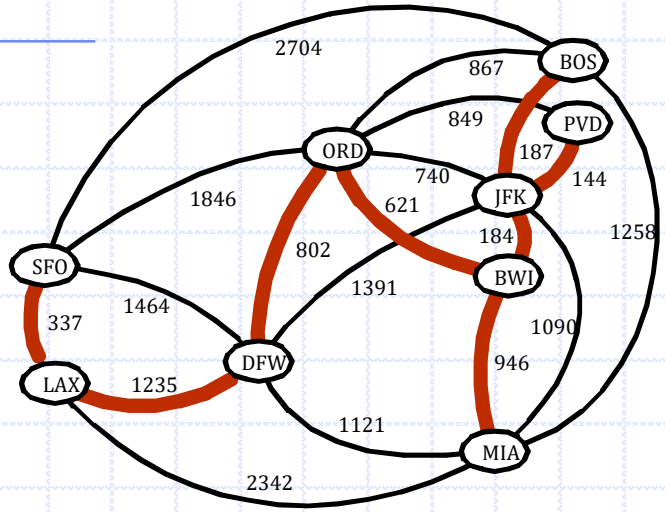
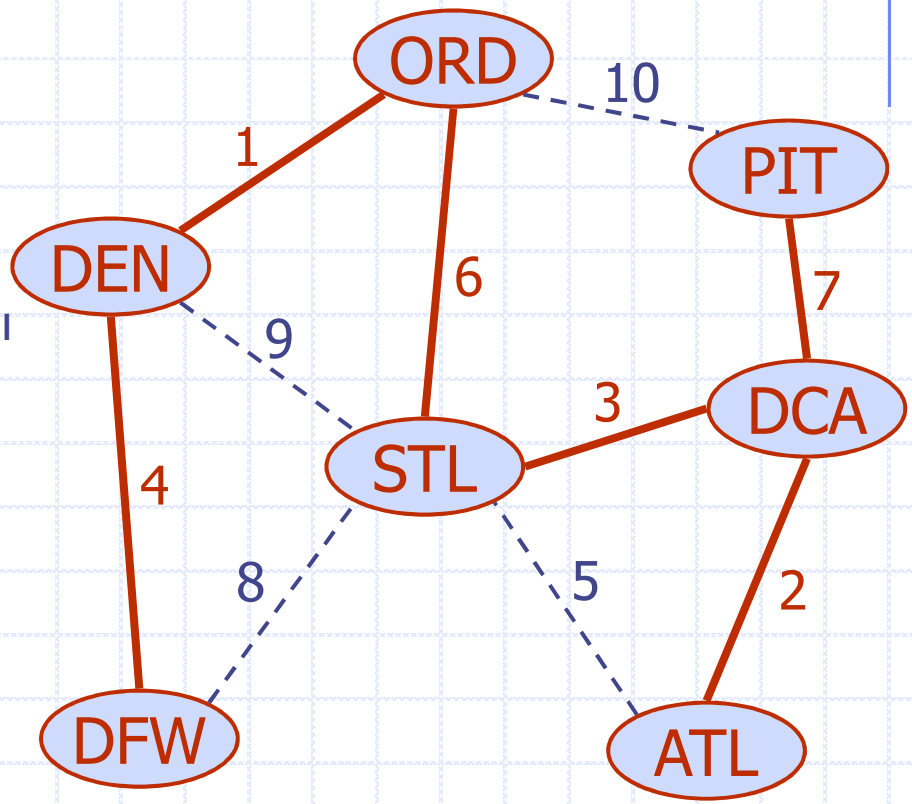
Ελάχιστα Ζευγνύοντα Δένδρα



Ελάχιστα Ζευγνύοντα Δένδρα

- Ζευγνύον υπογράφος
  - Υπογράφος ενός γράφου G που περιέχει όλες τις κορυφές του G
- Ζευγνύον δένδρο
  - Ζευγνύον υπογράφος που είναι ένα (ελεύθερο) δένδρο
- Ελάχιστο ζευγνύον δένδρο (MST)
  - Σε ένα γράφο με βάρη ένα ζευγνύον δένδρο με ελάχιστο συνολικό βάρος ακμών
- Εφαρμογές
  - Δίκτυα επικοινωνιών
  - Δίκτυα μεταφορών



Δοθέντος ενός μη κατευθυνόμενου γράφου G με βάρη, να βρεθεί ένα δένδρο T που να έχει όλες τις κορυφές του G και να ελαχιστοποιεί το άθροισμα:

$$w(T) = \sum_{(v,u) \in T} w((v,u))$$

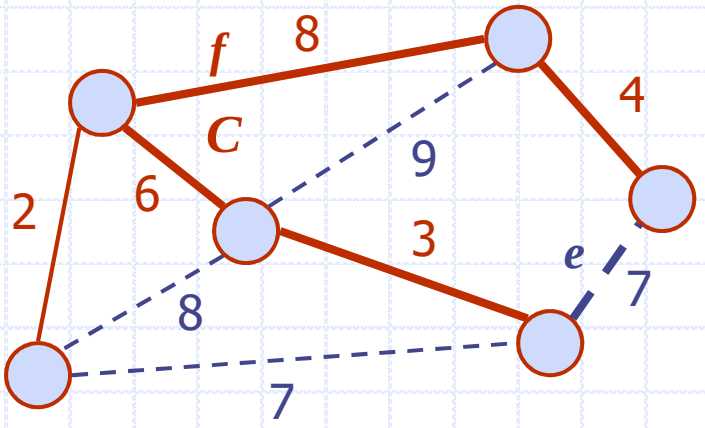
Ιδιότητα του Κύκλου

Ιδιότητα του Κύκλου:

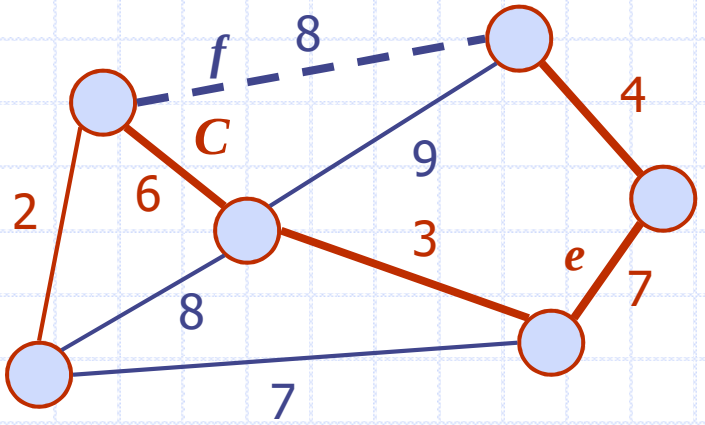
- Έστω T ένα ελάχιστο ζευγνύον δένδρο ενός γράφου με βάρη G
- Έστω e μια ακμή του G που δεν ανήκει στο T και έστω C ο κύκλος που σχηματίζεται από την e στον T
- Για κάθε ακμή f του C, $weight(f) \leq weight(e)$

Απόδειξη:

- Αν $weight(f) > weight(e)$ μπορούμε να έχουμε ένα ζευγνύον δένδρο με μικρότερο βάρος αντικαθιστώντας την f με την e



Αντικαθιστώντας την f με την e έχουμε ένα καλύτερο ζευγνύον δένδρο



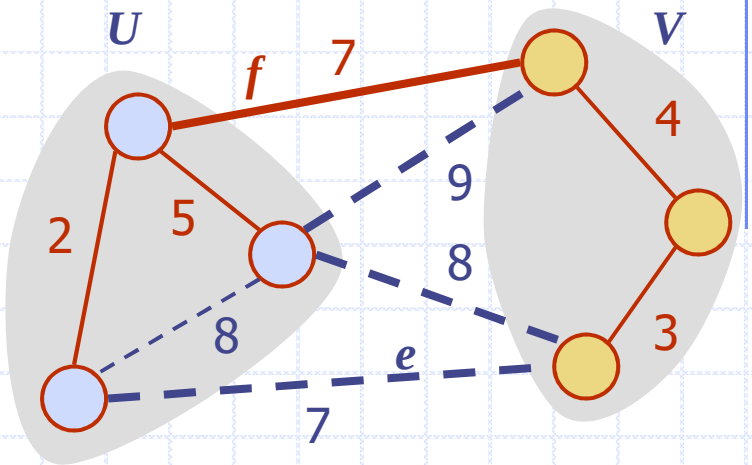
Ιδιότητα της Διαμέρισης

Ιδιότητα της διαμέρισης:

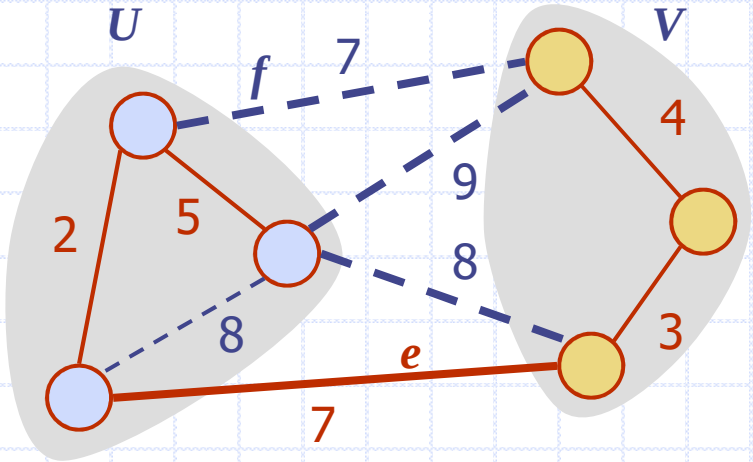
- Έστω μια διαμέριση των κορυφών του G σε υποσύνολα U και V
- Έστω e μια ακμή με ελάχιστο βάρος μεταξύ της διαμέρισης
- Υπάρχει ένα ελάχιστο ζευγνύον δένδρο του G οπου περιέχει την ακμή e

Απόδειξη:

- Έστω T ένα MST του G
- Αν το T δεν περιέχει την e, έστω ο κύκλος C που σχηματίζεται από την e με το T και έστω f μια ακμή του C μεταξύ της διαμέρισης
- Με βάση την ιδιότητα του κύκλου,
$$weight(f) \leq weight(e)$$
- Επομένως , $weight(f) = weight(e)$
- Έχουμε ένα άλλο MST αντικαθιστώντας την f με την e



Η αντικατάσταση της f με την e δίνει άλλο MST

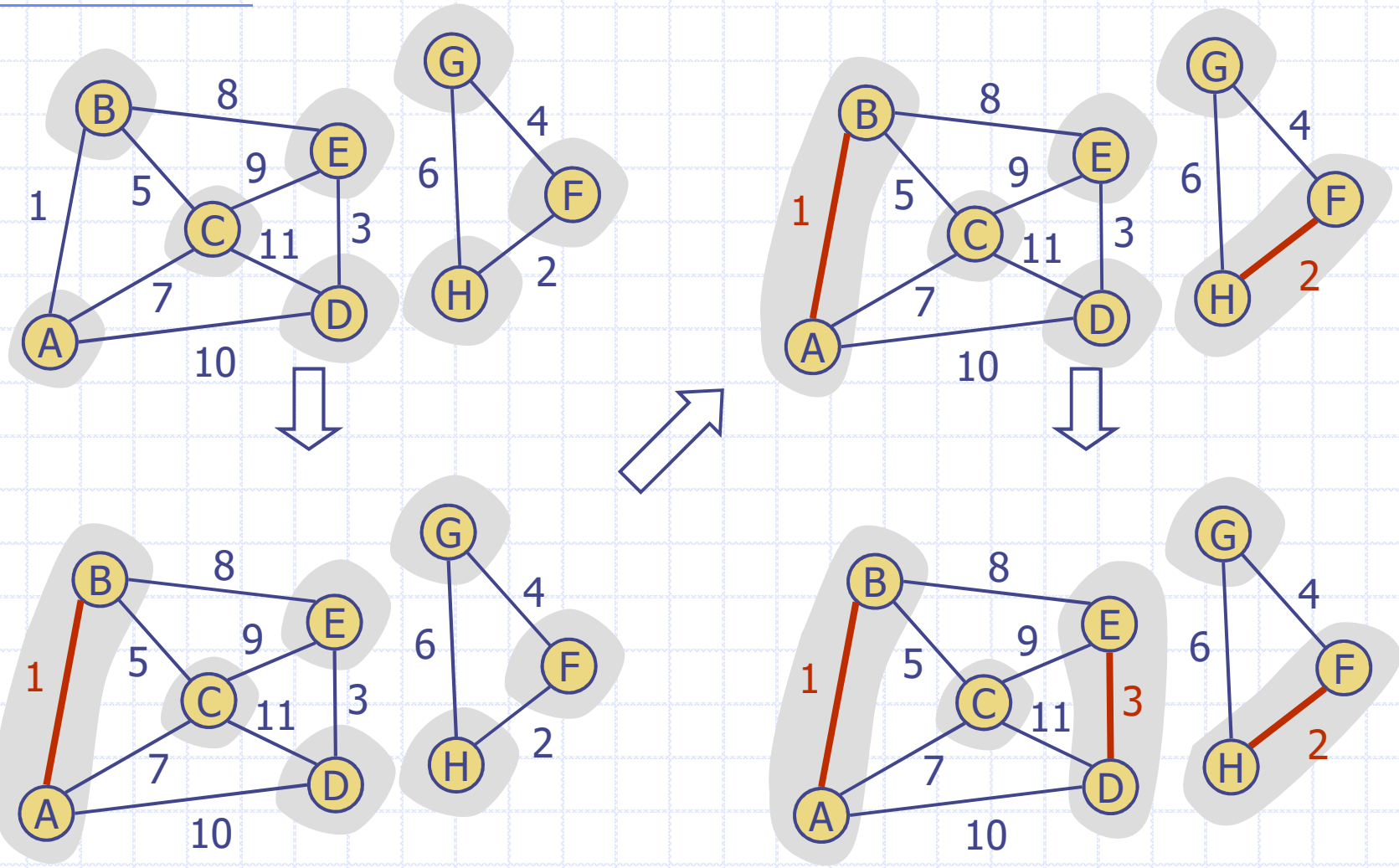


Αλγόριθμος του Kruskal

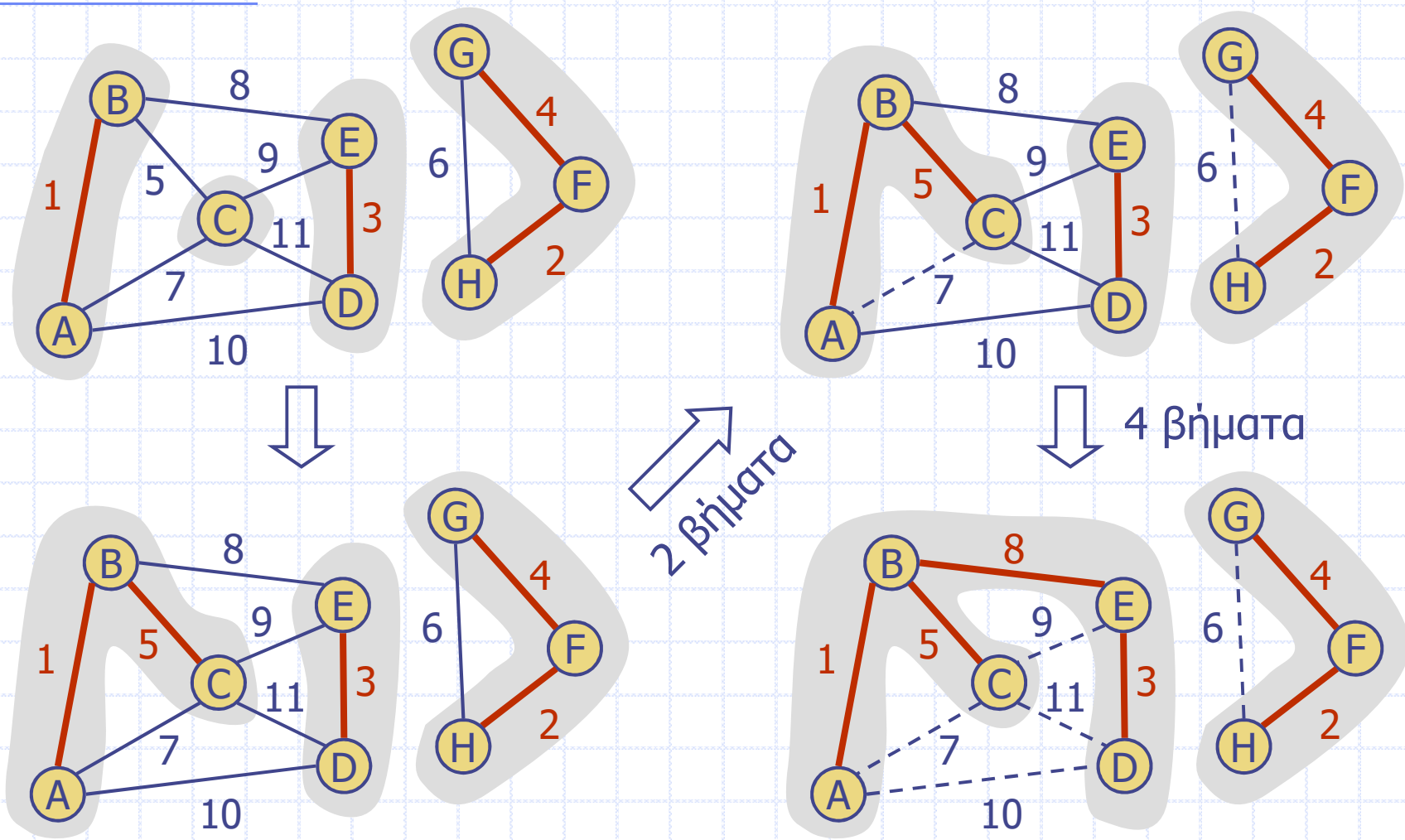
- Οι κορυφές διαμερίζονται σε ομάδες
  - Αρχικά, ομάδες μιας κορυφής
  - Κρατάμε ένα MST για κάθε ομάδα
  - Συγχώνευση των "πλησιέστερων" ομάδων και των MST τους
- Μια ουρά προτεραιότητας αποθηκεύει τις ακμές εκτός ομάδων
  - Κλειδί: βάρος
  - Στοιχείο: ακμή
- Στο τέλος του αλγορίθμου
  - Μια ομάδα και ένα MST

Algorithm ***KruskalMST(G)***
  for each vertex ***v*** in ***G*** **do**
    Δημιούργησε μια ομάδα που περιέχει το ***v***
  Έστω ***Q*** μια ουρά προτεραιότητας.
  Insert all edges into ***Q***
  T ← ∅
  {***T*** είναι η ένωση των MST των ομάδων}
  while ***T*** has fewer than ***n*** − 1 edges **do**
  e ← ***Q.removeMin().getValue()***
    [***u, v***] ← ***G.endVertices(e)***
    A ← ***getCluster(u)***
    B ← ***getCluster(v)***
    if ***A*** ≠ ***B*** **then**
      Add edge ***e*** to ***T***
      mergeClusters(A, B)
  return ***T***

Παράδειγμα



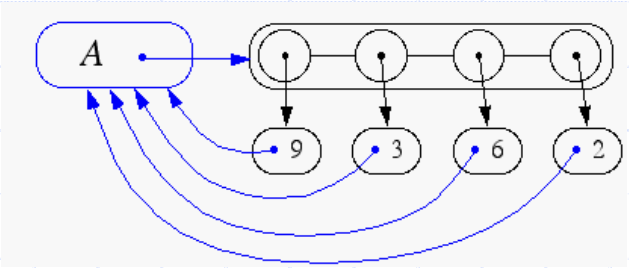
Παράδειγμα (συν.)



Δομή Δεδομένων για τον αλγόριθμου του Kruskal

- Ο αλγόριθμος διατηρεί ένα δάσος από δένδρα
- Μια ουρά προτεραιότητας εξάγει τις ακμές με αύξουσα σειρά βάρους
- Μια ακμή είναι αποδεκτή αν συνδέει διακριτά δένδρα
- Χρειαζόμαστε μια δομή δεδομένων που διατηρεί μια διαμέριση, π.χ., μια συλλογή από ξένα σύνολα, με πράξεις:
  - makeSet(u): δημιούργησε ένα σύνολο που αποτελείται από το u
  - find(u): επιστρέφει τα σύνολο που αποθηκεύει το u
  - union(A, B): αντικαθιστά τα σύνολα A και B με την ένωσή τους

Διαμέριση που βασίζεται σε λίστα



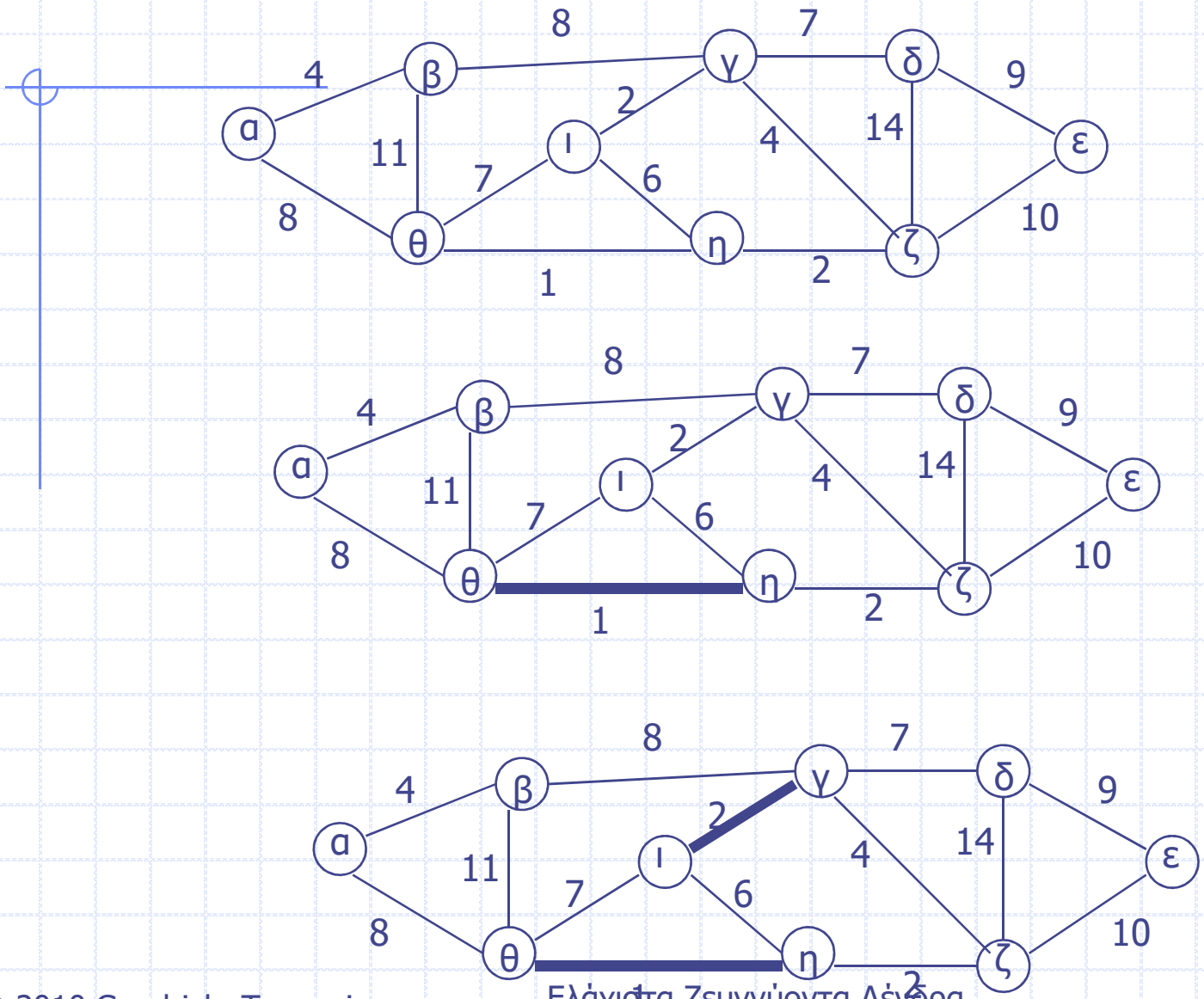
- Κάθε σύνολο αποθηκεύεται σε μια ακολουθία
- Κάθε στοιχείο έχει μια αναφορά πίσω στο σύνολο
  - η πράξη find(u) θέλει O(1) χρόνο, και επιστρέφει το σύνολο στο οποίο ανήκει η u.
  - στην πράξη union(A,B), μεταφέρουμε τα στοιχεία του μικρότερου συνόλου στην ακολουθία του μεγαλυτέρου συνόλου και ενημερώνουμε τις αναφορές τους
  - ο χρόνος για την πράξη union(A,B) είναι min(|A|, |B|)
- Όποτε επεξεργαζόμαστε ένα στοιχείο, πάει σε ένα σύνολο τουλάχιστον διπλού μεγέθους, επομένως η επεξεργασία κάθε στοιχείου γίνεται το πολύ log n φορές

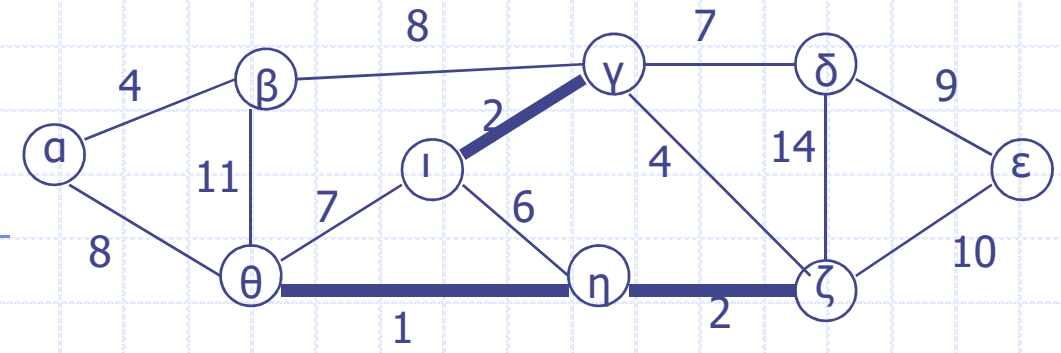
Υλοποίηση που βασίζεται σε διαμέριση

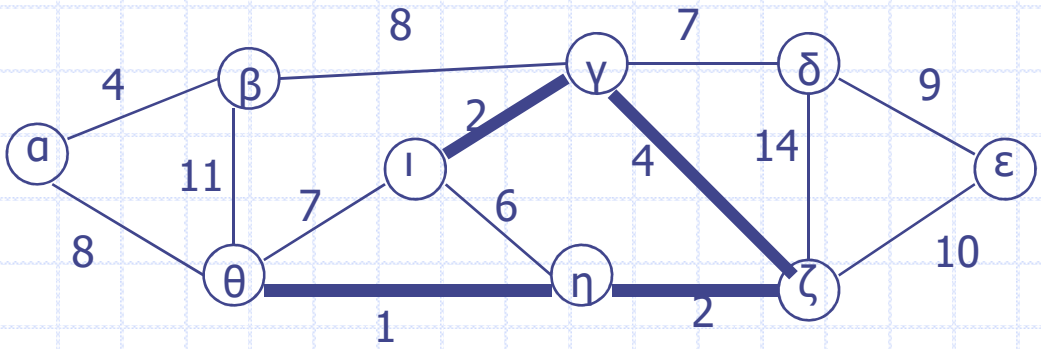
- Εκδοχή του αλγορίθμου Kruskal που βασίζεται σε διαμέριση
  - Οι ομάδες συγχωνεύονται σαν ενώσεις
  - Θέσεις των ομάδων με την find
- Χρόνος τρεξίματος $O((n + m) \log n)$
  - PQ πράξεις $O(m \log n)$
  - UF πράξεις $O(n \log n)$

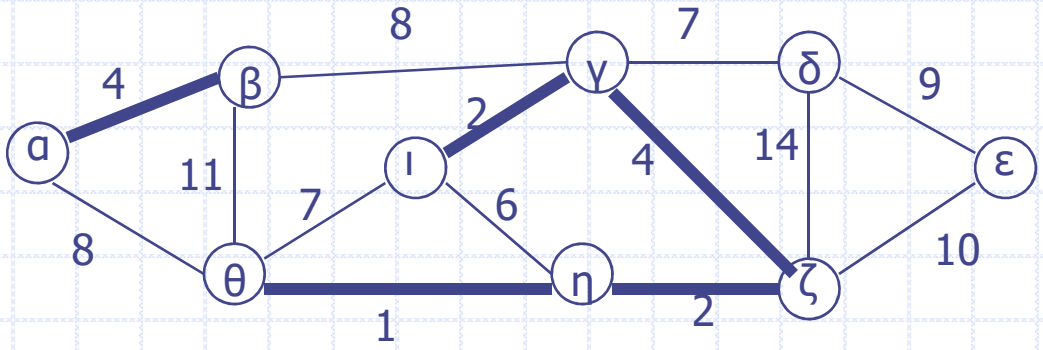
Algorithm *KruskalMST*(*G*)
  Initialize a partition *P*
  for each vertex *v* in *G* **do**
    P.makeSet(*v*)
  let *Q* be a priority queue.
  Insert all edges into *Q*
  $T \leftarrow \emptyset$
  {*T* είναι η ένωση των MSTs of the clusters}
  while *T* has fewer than $n - 1$ edges **do**
  $e \leftarrow$ *Q.removeMin*().*getValue*()
    $[u, v] \leftarrow$ *G.endVertices*(*e*)
    $A \leftarrow$ *P.find*(*u*)
    $B \leftarrow$ *P.find*(*v*)
    if $A \neq B$ **then**
      Add edge *e* to *T*
      P.union(*A*, *B*)
  return *T*

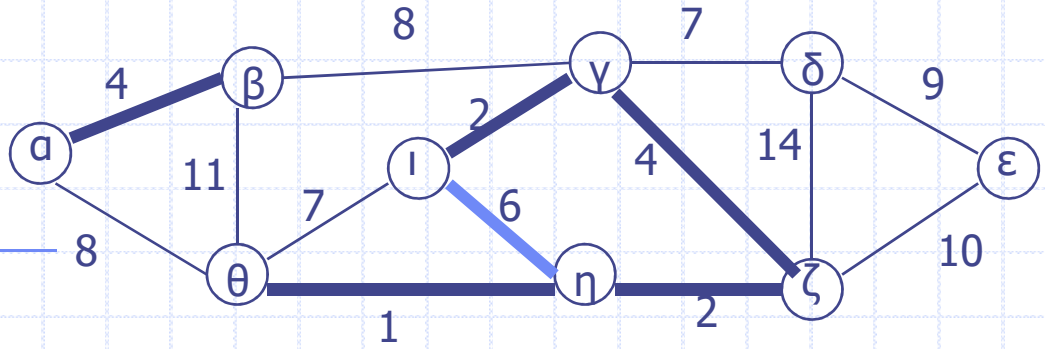
Παράδειγμα

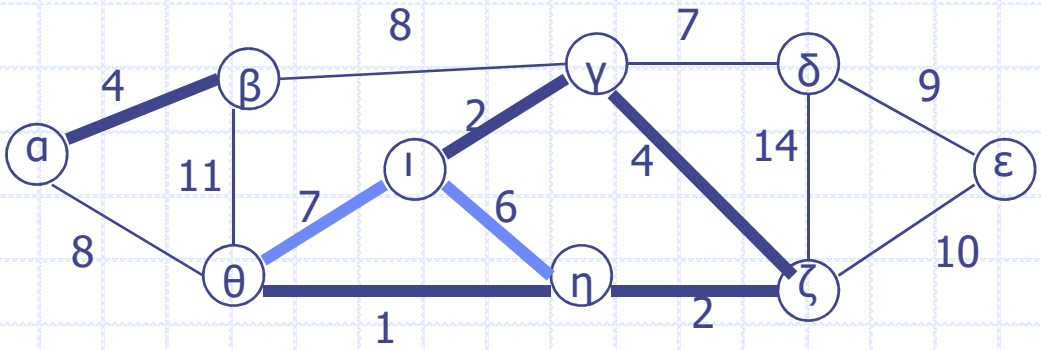


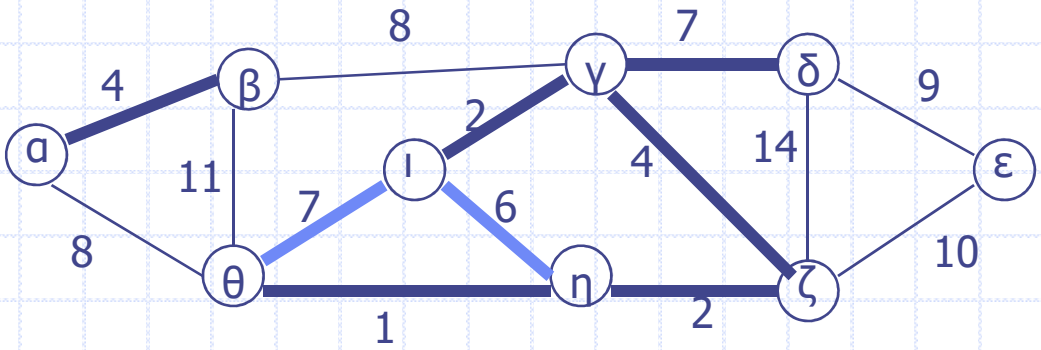

8
7
4
β
γ
δ
9
2
α
ι
14
ε
11
4
7
6
8
θ
η
ζ
10
1
2

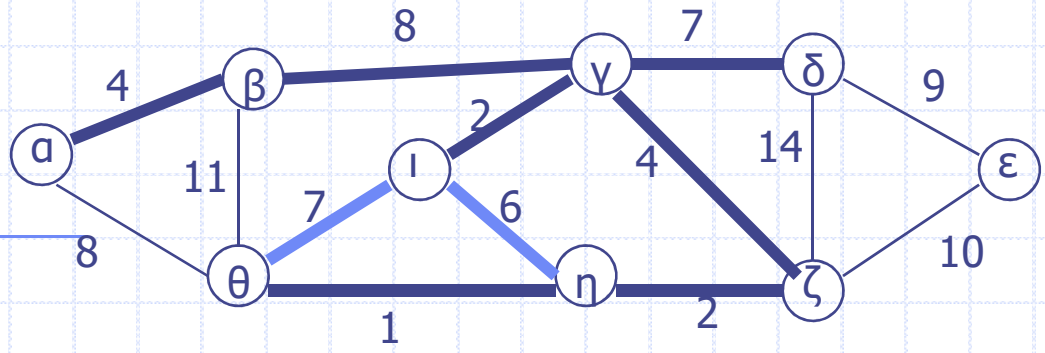

8
7
4
β
γ
δ
9
2
α
ι
14
ε
11
4
7
6
8
θ
η
ζ
10
1
2

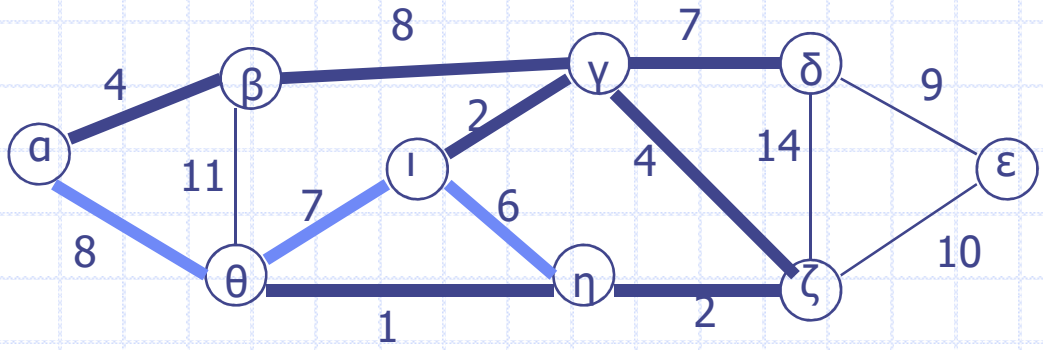

8
7
4
β
γ
δ
9
2
α
ι
14
ε
11
4
7
6
8
θ
η
ζ
10
1
2

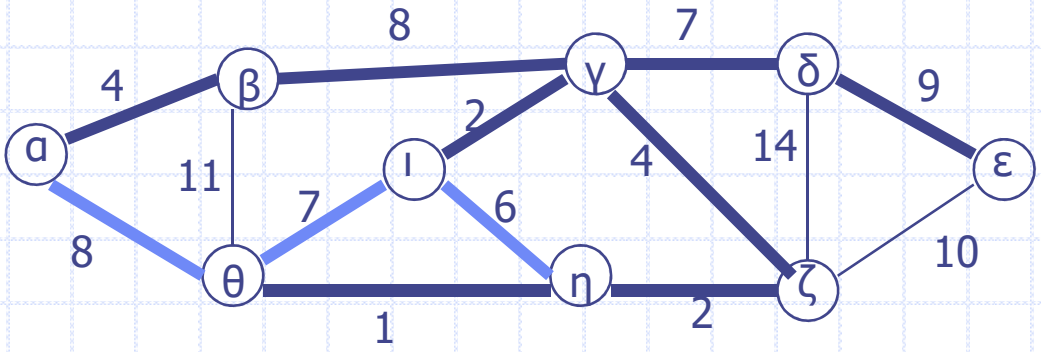

α
β
γ
δ
ε
ι
θ
η
ζ
4
8
7
9
11
2
14
7
6
4
8
10
1
2

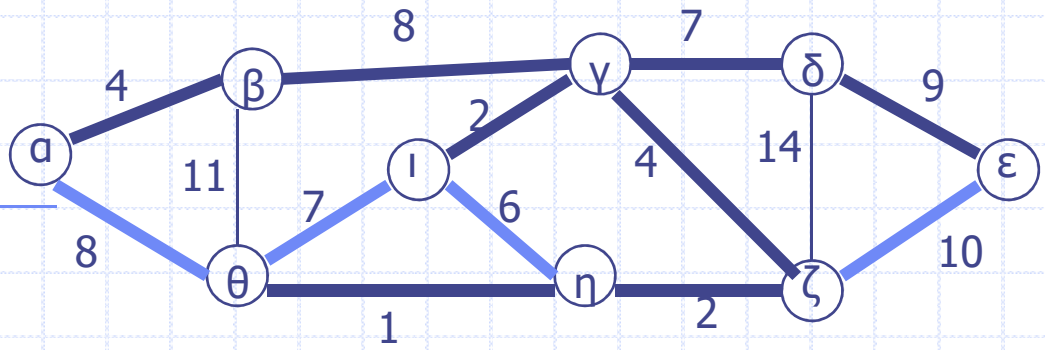

α
β
γ
δ
ε
ι
θ
η
ζ
4
8
7
9
11
2
14
7
6
4
8
10
1
2

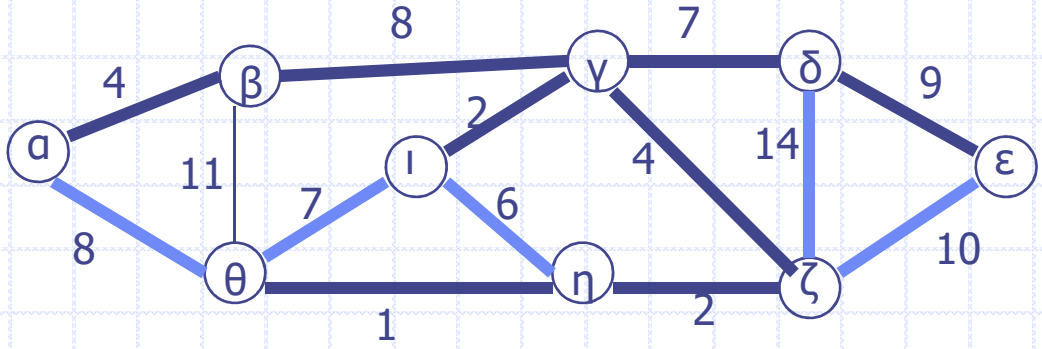

α
β
γ
δ
ε
ι
θ
η
ζ
4
8
7
9
11
2
14
7
6
4
8
10
1
2


α
β
γ
δ
ε
ι
θ
η
ζ
4
8
7
9
2
11
14
4
7
6
8
10
1
2


α
β
γ
δ
ε
ι
θ
η
ζ
4
8
7
9
2
11
14
4
7
6
8
10
1
2


α
β
γ
δ
ε
ι
θ
η
ζ
4
8
7
9
2
11
14
4
7
6
8
10
1
2


α
β
γ
δ
ε
ι
θ
η
ζ
4
8
7
9
11
2
14
4
7
6
8
10
1
2


α
β
γ
δ
ε
ι
θ
η
ζ
4
8
7
9
11
2
14
4
7
6
8
10
1
2

Αλγόριθμος του Prim-Jarnik

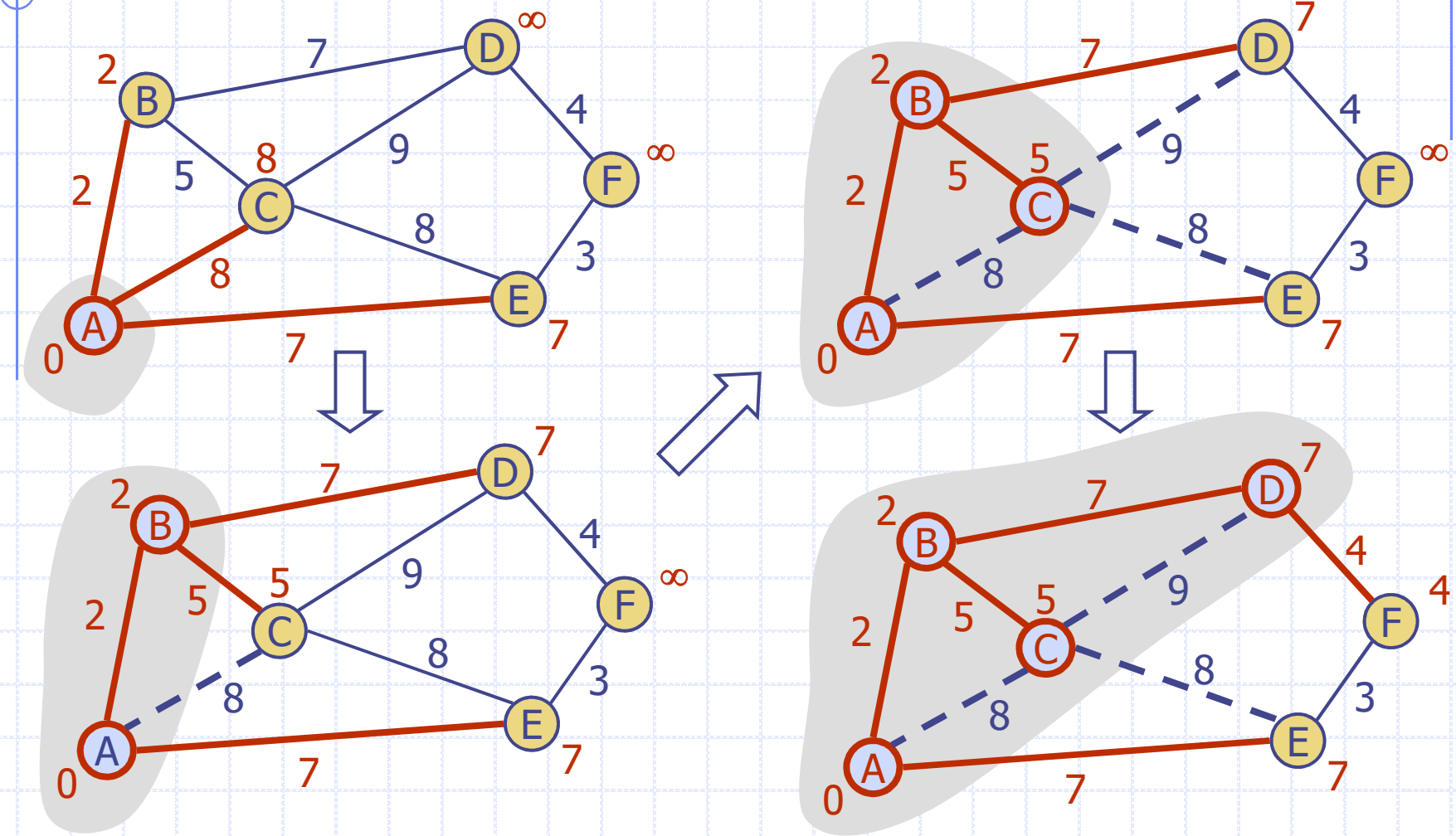
- Μοιάζει με τον αλγόριθμο του Dijkstra
- Επιλέγουμε τυχαία μια κορυφή s και μεγαλώνουμε το MST σαν ένα νέφος από κορυφές, που αρχίζουν από την s
- Με κάθε κορυφή v αποθηκεύουμε την ετικέτα $d(v)$ που παριστάνει το μικρότερο βάρος ακμής που συνδέει την v με μια κορυφή στο νέφος
- Σε κάθε βήμα:
  - Προσθέτουμε στο νέφος την κορυφή u εκτός νέφους με την ετικέτα μικρότερου βάρους
  - Ενημερώνουμε τις ετικέτες των διαδοχικών κορυφών της u

Αλγόριθμος Prim-Jarnik (συν.)

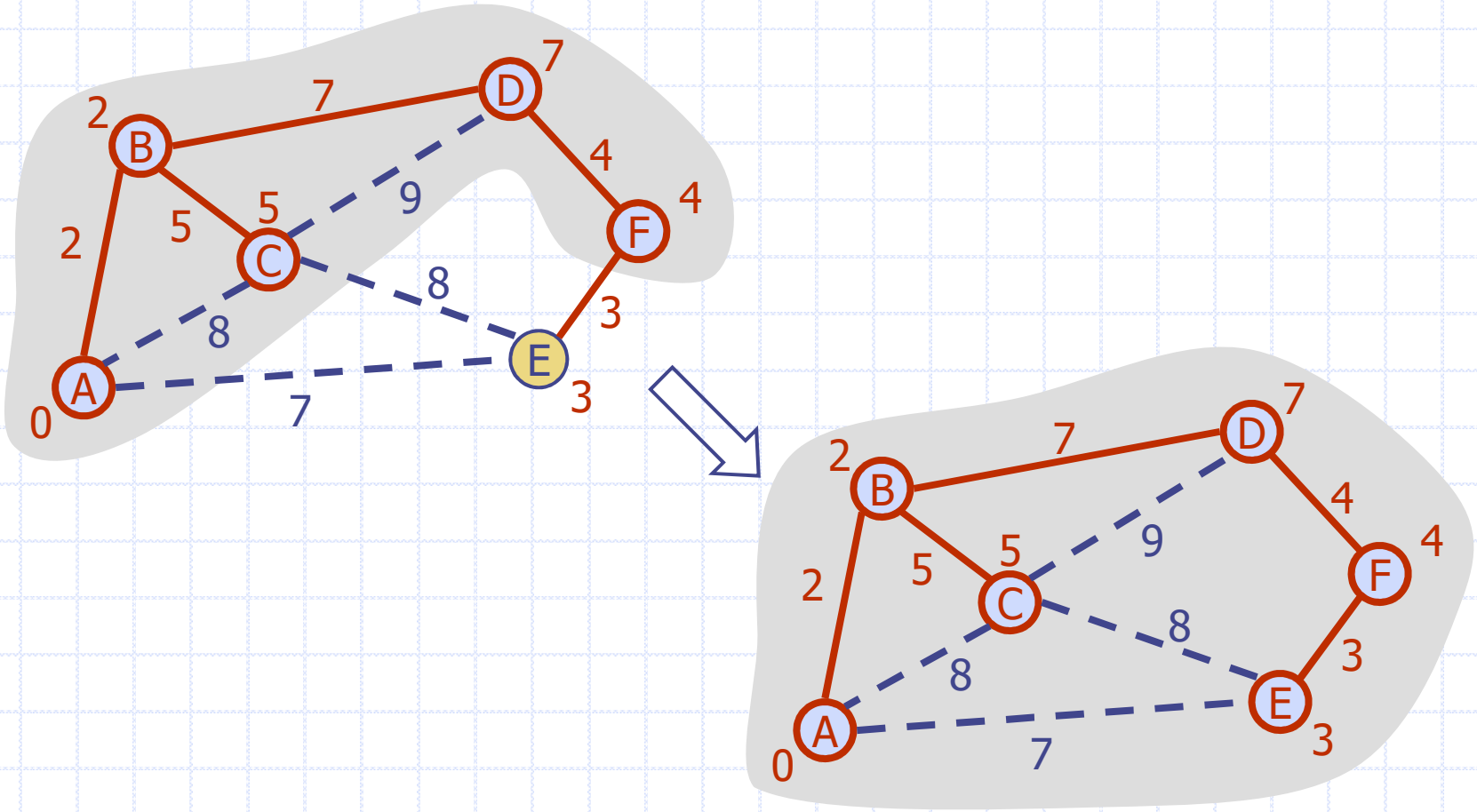
- Μια προσαρμοσμένη ουρά προτεραιότητας που βασίζεται σε σωρό αποθηκεύει κορυφές εκτός νέφους με καταχωρήσεις ενήμερες της θέσης
  - Κλειδί: απόσταση
  - Τιμή: κορυφή
  - Θυμηθείτε την μέθοδο *replaceKey*(*l*,*k*) που αλλάζει το κλειδί της καταχώρησης *l*
- Αποθηκεύουμε τρεις ετικέτες με κάθε κορυφή:
  - Απόσταση
  - Ο κόμβος γονέας στο MST
  - Καταχώρηση στην ουρά προτεραιότητας

```
Algorithm PrimJarnikMST(G)
    Q ← new heap-based priority queue
    s ← a vertex of G
    for all v ∈ G.vertices()
        if v = s
            setDistance(v, 0)
        else
            setDistance(v, ∞)
        setParent(v, ∅)
        l ← Q.insert(getDistance(v), v)
        setLocator(v,l)
    while ¬Q.isEmpty()
        l ← Q.removeMin()
        u ← l.getValue()
        for all e ∈ G.incidentEdges(u)
            z ← G.opposite(u,e)
            r ← weight(e)
            if r < getDistance(z)
                setDistance(z, r)
                setParent(z,e)
                Q.replaceKey(getEntry(z), r)
```

Παράδειγμα


A
B
C
D
E
F
0
2
7
∞
2
5
8
9
4
∞
8
3
8
7
7
2
7
∞
2
5
5
9
4
∞
8
3
8
7
7
7
2
7
7
2
5
5
9
4
∞
8
3
8
7
7
7
2
7
7
2
5
5
9
4
4
8
3
8
7
7

Παράδειγμα (συν.)



Ανάλυση

- Πράξεις γράφων
  - Η μέθοδος incidentEdges καλείται μια φορά για κάθε κορυφή
- Πράξεις ετικέτας
  - Θέτουμε/παίρνουμε την απόσταση, γονέα και εντοπιστή ετικέτας της κορυφής z $O(\deg(z))$ φορές
  - Να θέσουμε ή να άρουμε μια ετικέτα θέλει $O(1)$ χρόνο
- Πράξεις ουράς προτεραιότητας
  - Μια κορυφή εισάγεται μια φορά και διαγράφεται μια φορά από την ουρά προτεραιότητας, και κάθε εισαγωγή ή διαγραφή απαιτεί χρόνο $O(\log n)$
  - Το κλειδί μιας κορυφής w στην ουρά προτεραιότητας τροποποιείται το πολύ $\deg(w)$ φορές, όπου κάθε αλλαγή του κλειδιού απαιτεί χρόνο $O(\log n)$
- Ο αλγόριθμος Prim-Jarnik τρέχει σε χρόνο $O((n+m)\log n)$ εφόσον ο γράφος παριστάνεται από δομή λίστας γειτνίασης
  - Ισχύει $\sum_v \deg(v) = 2m$
- Ο χρόνος είναι $O(m \log n)$ εφόσον ο γράφος είναι συνεκτικός