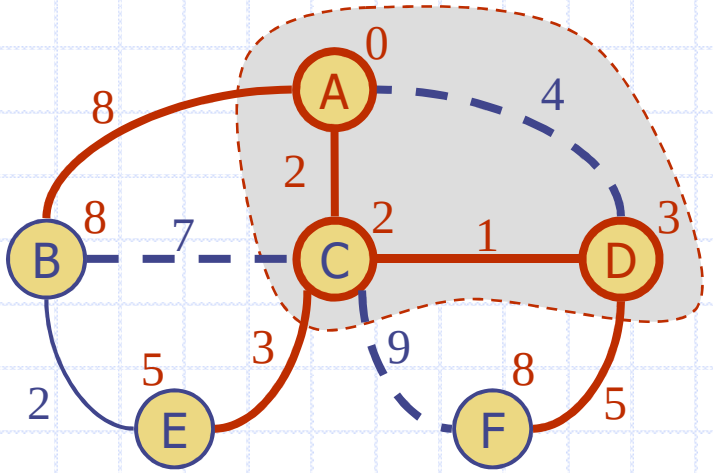
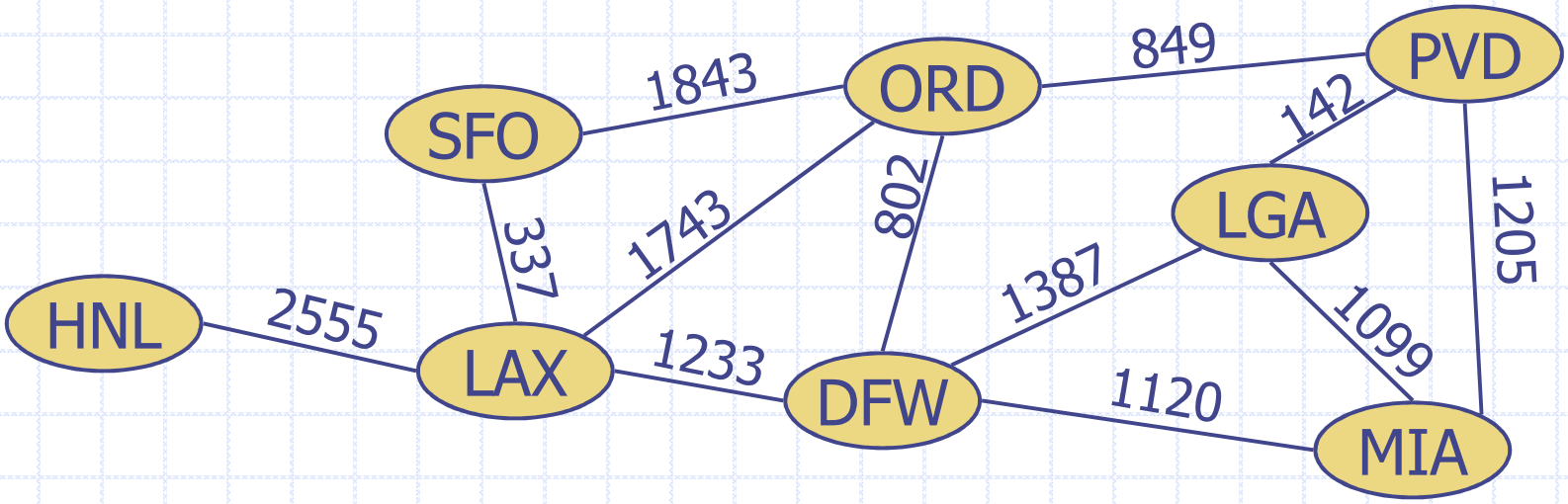
Ελάχιστες Διαδρομές



Γράφοι με βάρος

- Σε ένα γράφο με βάρη, σε κάθε ακμή αντιστοιχεί μια αριθμητική τιμή, που ονομάζεται το βάρος της ακμής
- Τα βάρη στις ακμές μπορεί να αντιπροσωπεύουν αποστάσεις, κόστος κλπ.
- Παράδειγμα:
  - Σε ένα γράφο διαδρομών πτήσεων, το βάρος μιας ακμής παριστάνει την απόσταση σε μίλια μεταξύ των αεροδρομίων στα άκρα της



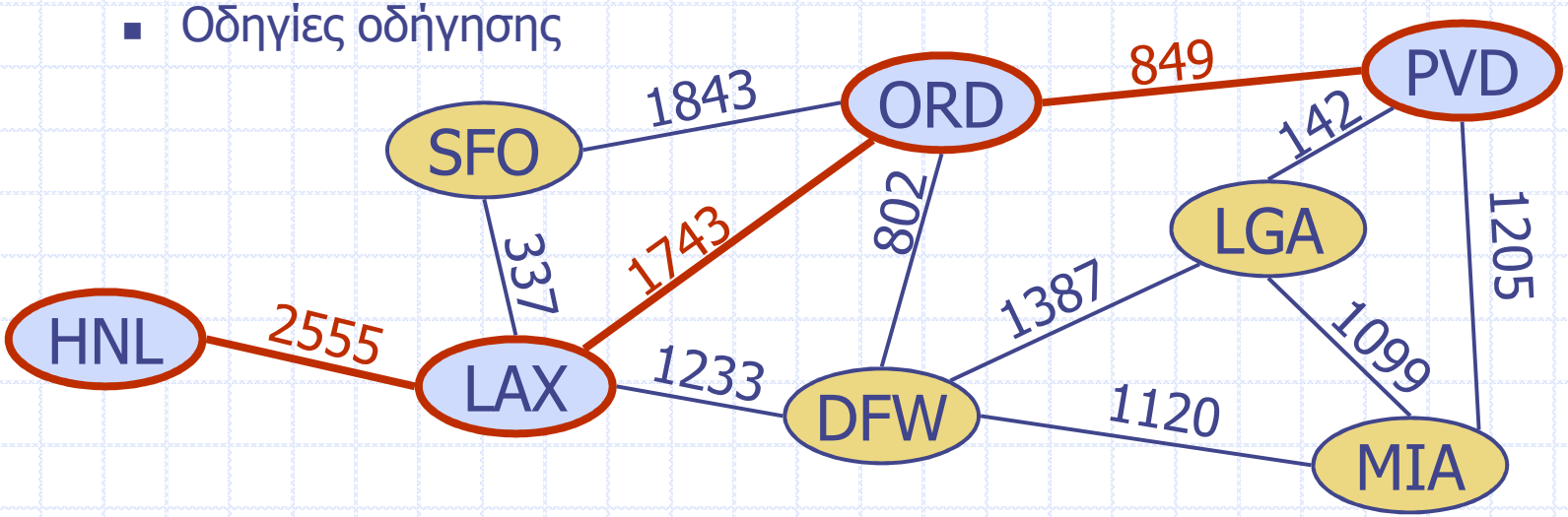
Μήκος μιας διαδρομής είναι το άθροισμα των βαρών (ή κόστους) των ακμών της διαδρομής P. Δηλαδή αν $P=((v_0,v_1),(v_1,v_2),\ldots,(v_{k-1},v_k))$ τότε το μήκος της διαδρομής P είναι:

$$w(P) = \sum_{i=0}^{k-1} w((v_i, v_{i+1}))$$

Η απόσταση από μια κορυφή v σε μια κορυφή u σε ένα γράφο G συμβολίζεται με d(v,u) και είναι το μήκος μιας διαδρομής ελάχιστου μήκους από την v στην u (ονομάζεται και συντομότερη διαδρομή).

Ελάχιστες Διαδρομές

- Δίδεται ένας γράφος με βάρη και δύο κορυφές u και v, θέλουμε να βρούμε μια διαδρομή μεταξύ των u και v με ελάχιστο συνολικό βάρος.
  - Μήκος της διαδρομής είναι το άθροισμα των βαρών των ακμών της.
- Παράδειγμα:
  - Ελάχιστη διαδρομή μεταξύ Providence και Honolulu
- Εφαρμογές
  - Δρομολόγηση στο Internet
  - Κρατήσεις σε πτήσεις
  - Οδηγίες οδήγησης



Ιδιότητες των Ελάχιστων Διαδρομών

Ιδιότητα 1:

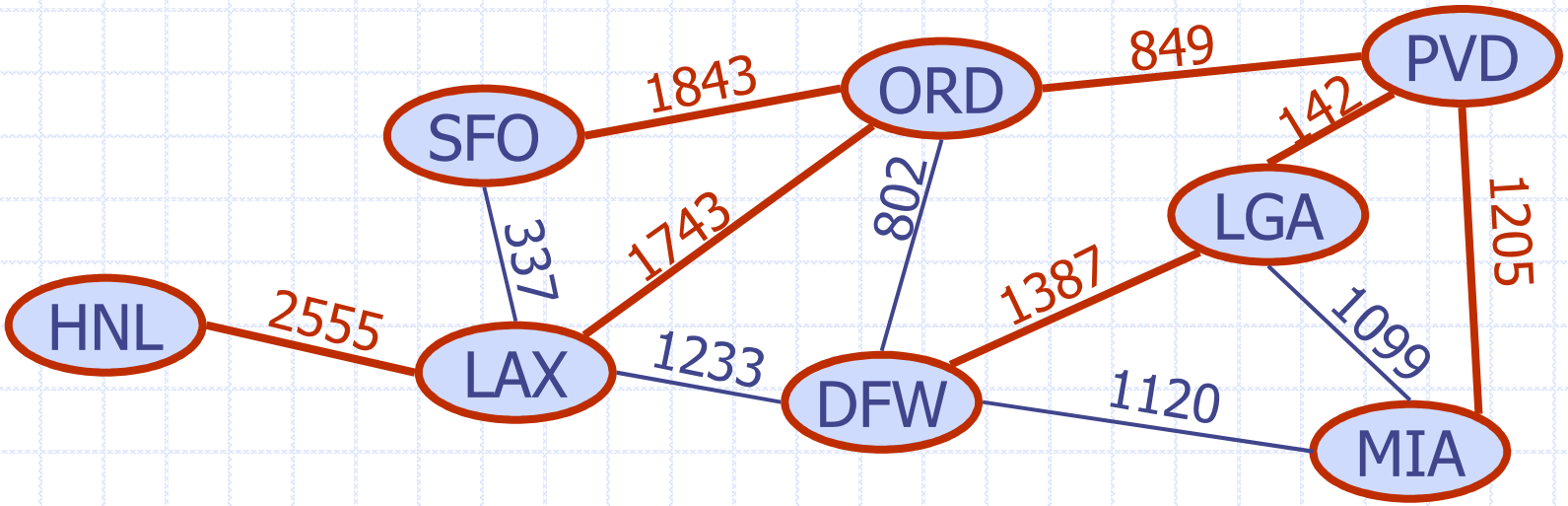
Μια υποδιαδρομή μιας ελάχιστης διαδρομής είναι και η ίδια ελάχιστη

Ιδιότητα 2:

Υπάρχει ένα δένδρο ελάχιστων διαδρομών από την κορυφή εκκίνησης προς όλες τις άλλες κορυφές

Παράδειγμα:

Δένδρο ελάχιστων διαδρομών από το Providence



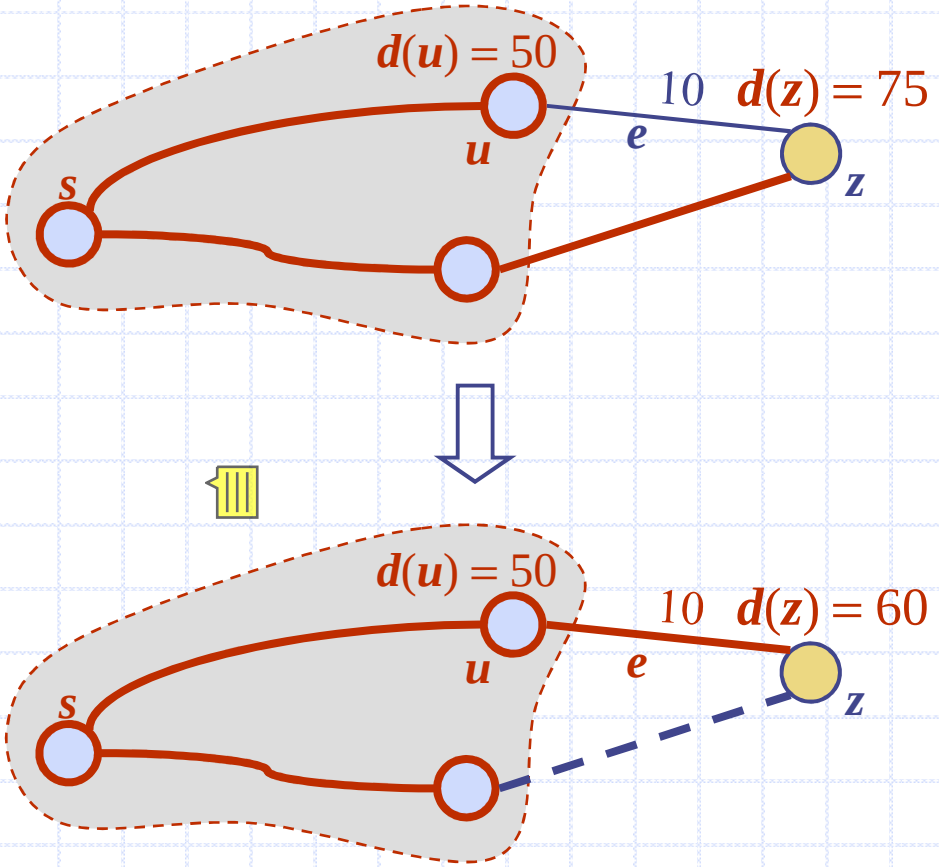
Αλγόριθμος Dijkstra

- Η απόσταση μιας κορυφής v από μια κορυφή s είναι το μήκος της ελάχιστης διαδρομής μεταξύ s και v
- Ο αλγόριθμος του Dijkstra υπολογίζει τις αποστάσεις όλων των κορυφών από δοθείσα αρχική κορυφή s
- Υποθέσεις:
  - ο γράφος είναι συνεκτικός
  - οι ακμές δεν είναι κατευθυνόμενες
  - τα βάρη των ακμών δεν είναι αρνητικά
- Αυξάνουμε ένα "νέφος" από κορυφές, αρχίζοντας με την s και καλύπτοντας τελικά όλες τις κορυφές
- Με κάθε κορυφή v αποθηκεύουμε μια ετικέτα $d(v)$ που παριστάνει την απόσταση του v από το s στον υπογράφο που αποτελείται από το νέφος και τις προσκείμενες κορυφές του
- Σε κάθε βήμα
  - Προσθέτουμε στο νέφος την κορυφή u εκτός νέφους με την ετικέτα μικρότερης απόστασης, $d(u)$
  - Ενημερώνουμε τις ετικέτες των διαδοχικών κορυφών της u

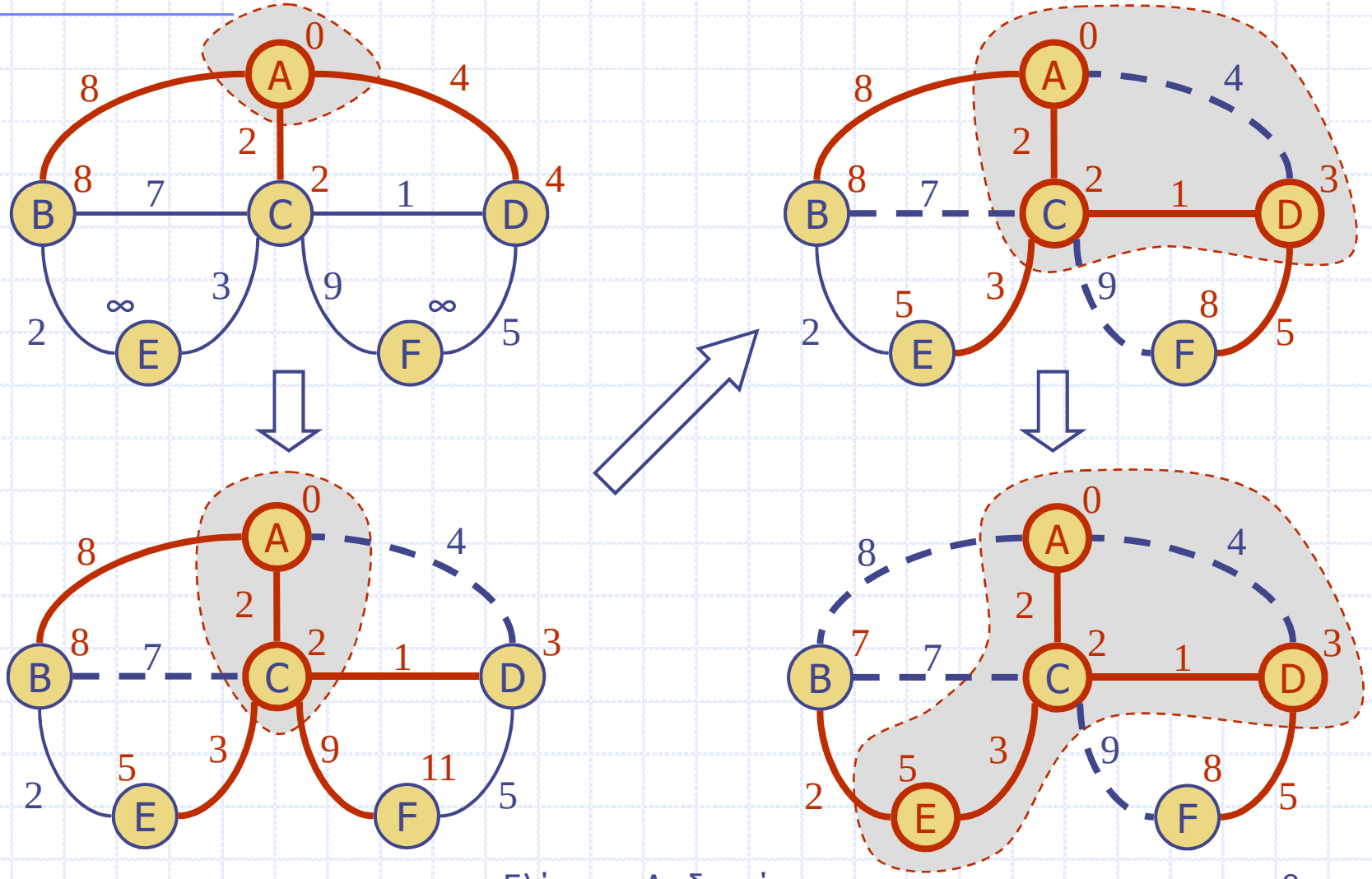
Ενημέρωση Ακμών

- Έστω μια ακμή $e = (u,z)$ τέτοια ώστε
  - u είναι η κορυφή που έχει προστεθεί πιο πρόσφατα στο νέφος
  - z δεν ανήκει στο νέφος s
- Η ενημέρωση της ακμής e τροποποιεί την απόσταση $d(z)$ σε:

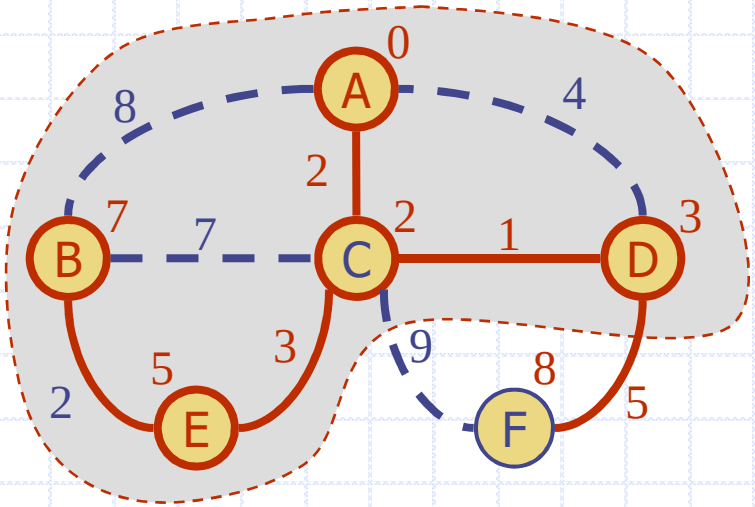
$$d(z) \leftarrow \min\{d(z), d(u) + weight(e)\}$$

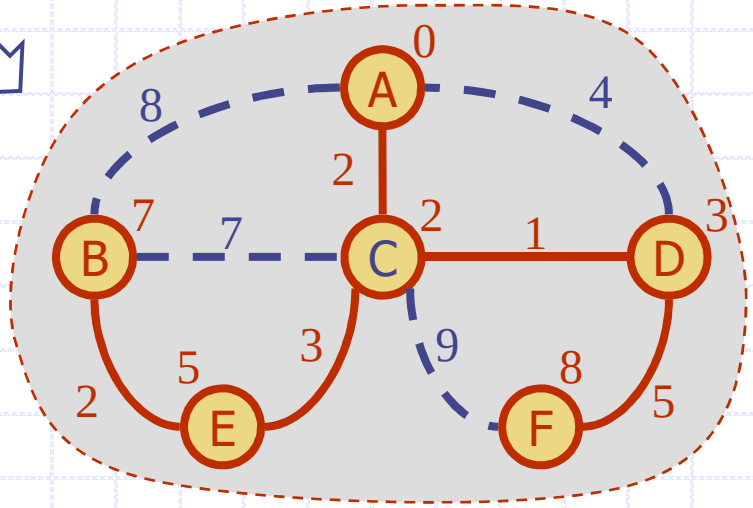


Παράδειγμα



Παράδειγμα (συν.)





Αλγόριθμος Dijkstra

- Αποθηκεύουμε με την ένδειξη θέσης τις κορυφές εκτός νέφους σε μια προσαρμοσμένη ουρά προτεραιότητας που βασίζεται σε σωρό
  - Κλειδί: απόσταση
  - Τιμή: κορυφή
  - Θυμίζουμε ότι η μέθοδος ***replaceKey***(*l*,*k*) αλλάζει το κλειδί της καταχώρησης *l*
- Με κάθε κορυφή αποθηκεύουμε δυο ετικέτες:
  - Απόσταση
  - Καταχώρηση στην ουρά προτεραιότητας

```
Algorithm DijkstraDistances(G, s)
    Q ← νέα ουρά προτεραιότητας βασισμένη σε σωρό
    for all v ∈ G.vertices()
        if v = s
            setDistance(v, 0)
        else
            setDistance(v, ∞)
        l ← Q.insert(getDistance(v), v)
        setEntry(v, l)
    while ¬Q.isEmpty()
        l ← Q.removeMin()
        u ← l.getValue()
        for all e ∈ G.incidentEdges(u) { relax e }
            z ← G.opposite(u,e)
            r ← getDistance(u) + weight(e)
            if r < getDistance(z)
                setDistance(z,r)
                Q.replaceKey(getEntry(z), r)
```

Ανάλυση του αλγορίθμου του Dijkstra

- Πράξεις γράφου
  - Η μέθοδος incidentEdges καλείται μια φορά για κάθε κορυφή
- Πράξεις ετικέτας
  - Θέτουμε/παίρνουμε τις ετικέτες απόστασης και θέσης της κορυφής z $O(\deg(z))$ φορές
  - Η πράξη αυτή θέλει χρόνο $O(1)$
- Πράξεις ουράς προτεραιότητας
  - Κάθε κορυφή εισάγεται και διαγράφεται μια φορά από την ουρά προτεραιότητας, όπου κάθε εισαγωγή ή διαγραφή απαιτεί χρόνο $O(\log n)$
  - Το κλειδί μιας κορυφής την ουρά προτεραιότητας τροποποιείται το πολύ $\deg(w)$ φορές, όπου η αλλαγή κάθε κλειδιού απαιτεί χρόνο $O(\log n)$
- Ο αλγόριθμος Dijkstra τρέχει σε $O((n + m) \log n)$ χρόνο εφόσον ο γράφος παριστάνεται με δομή λίστας γειτνίασης
  - Θυμίζουμε ότι ισχύει $\sum_v \deg(v) = 2m$
- Ο χρόνος τρεξίματος εκφράζεται σαν $O(m \log n)$ αφού ο γράφος είναι συνεκτικός

Δένδρο ελάχιστων διαδρομών

- Χρησιμοποιώντας τη μέθοδο πλαισίου προτύπου, μπορούμε να επεκτείνουμε τον αλγόριθμο του Dijkstra για να μας δώσει ένα δένδρο ελάχιστων διαδρομών από την αρχική κορυφή σε όλες τις άλλες κορυφές
- Με κάθε κορυφή αποθηκεύουμε μια τρίτη ετικέτα :
  - η ακμή γονέας στο δένδρο ελάχιστης διαδρομής
- Στο βήμα ενημέρωσης της ακμής, τροποποιούμε την ετικέτα του γονέα

Algorithm *DijkstraShortestPathsTree(G, s)*

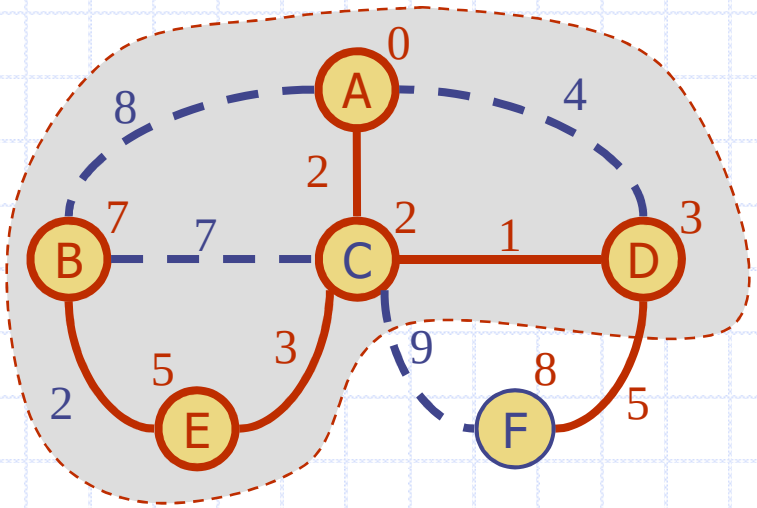
```
…

for all v ∈ G.vertices()
  …
  setParent(v, ∅)
  …

  for all e ∈ G.incidentEdges(u)
    { relax edge e }
    z ← G.opposite(u,e)
    r ← getDistance(u) + weight(e)
    if r < getDistance(z)
      setDistance(z, r)
      setParent(z,e)
      Q.replaceKey(getEntry(z),r)
```

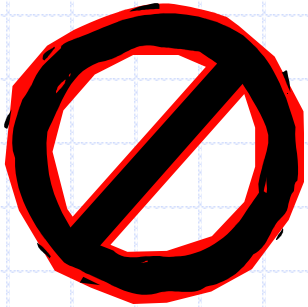
Γιατί δουλεύει ο αλγόριθμος του Dijkstra

- Ο αλγόριθμος του Dijkstra βασίζεται στην άπληστη μέθοδο. Προσθέτει κορυφές με αυξανόμενη απόσταση.
  - Υποθέστε ότι δεν έχει βρει όλες τις ελάχιστες αποστάσεις. Έστω F η πρώτη λάθος κορυφή που επεξεργάστηκε ο αλγόριθμος.
  - Όταν εξετάσθηκε ο προηγούμενος κόμβος, D, στην πραγματική ελάχιστη διαδρομή, η απόσταση του ήταν σωστή
  - Αλλά η ακμή (D,F) τροποποιήθηκε εκείνη τη στιγμή!
  - Επομένως, όσο ισχύει d(F)≥d(D), οι αποστάσεις δεν μπορεί να είναι λάθος. Δηλαδή, δεν υπάρχει λάθος κορυφή

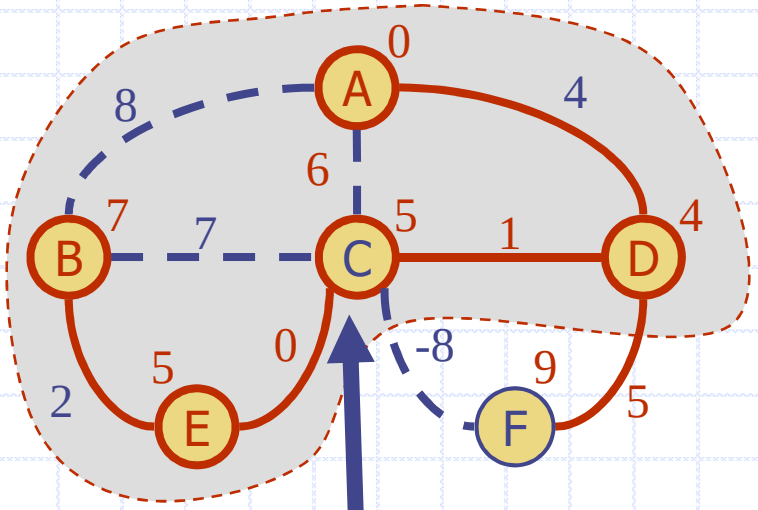


Στον αλγόριθμο Dijkstra, οποτεδήποτε μια κορυφή, u, σύρεται στο νέφος η ετικέτα D[u] ισούται με το d(v,u)

Γιατί δεν δουλεύει για Ακμές με αρνητικά βάρη

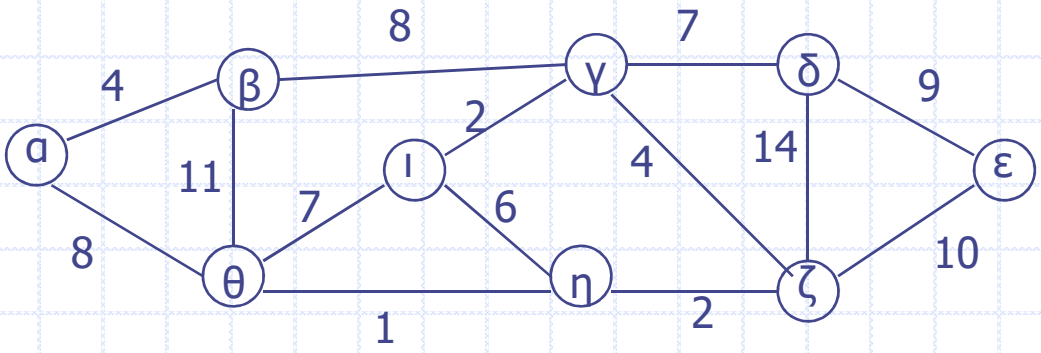


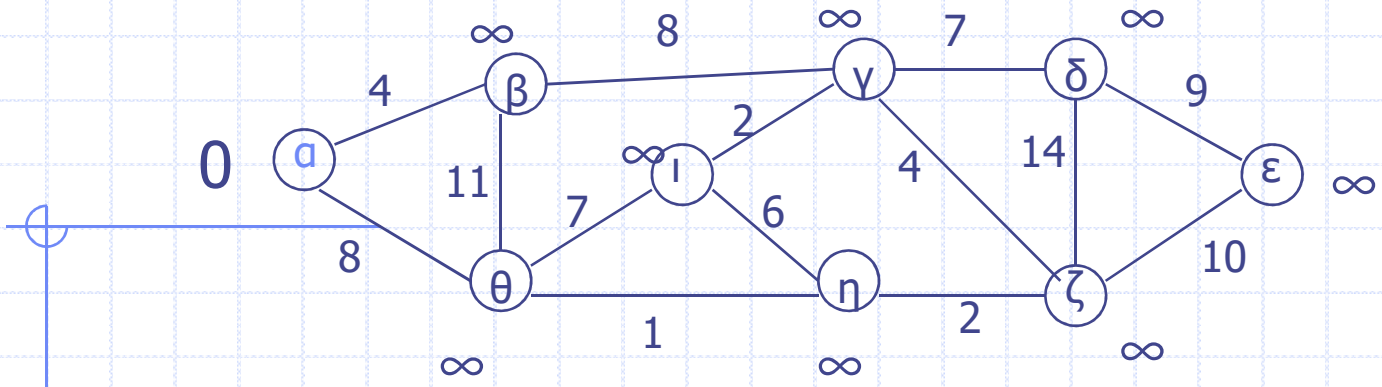
- Ο αλγόριθμος του Dijkstra βασίζεται στην άπληστη μέθοδο. Προσθέτει κορυφές με αύξουσα απόσταση.
  - Αν αργότερα επρόκειτο να προστεθεί στο νέφος ένας κόμβος με προσκείμενη αρνητική ακμή, θα κατέστρεφε τις αποστάσεις κορυφών που βρίσκονται ήδη στο νέφος.

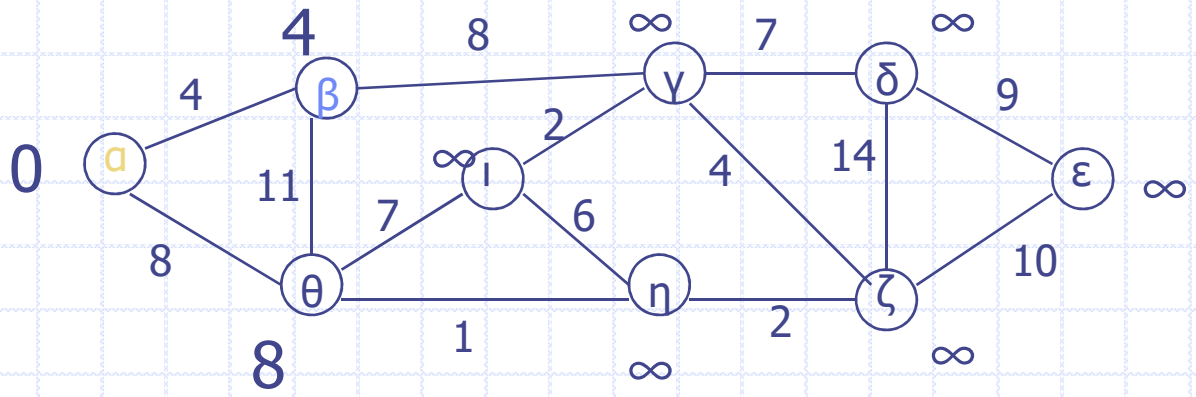


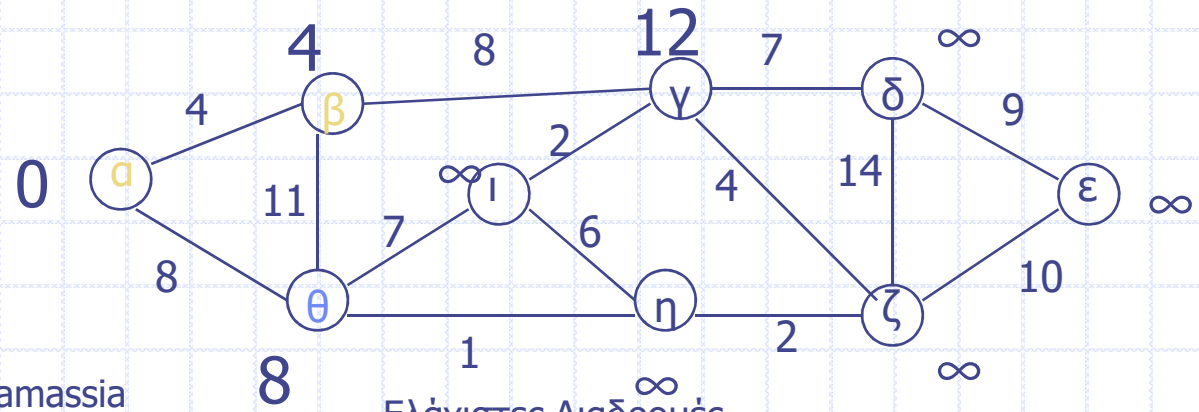
Η πραγματική απόσταση του C είναι 1, αλλά είναι ήδη στο νέφος με d(C)=5!

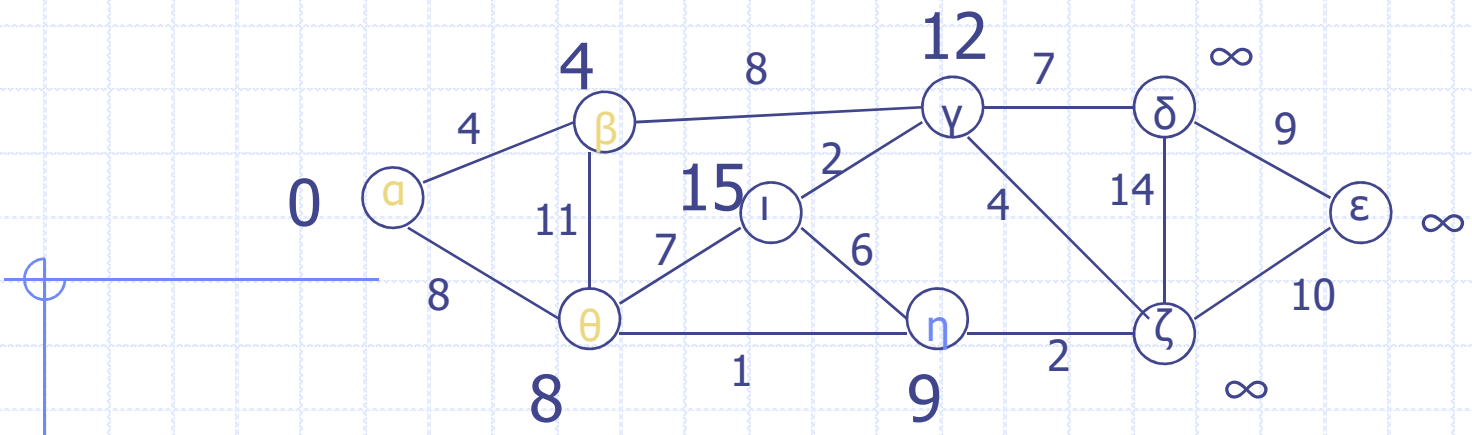
Παράδειγμα

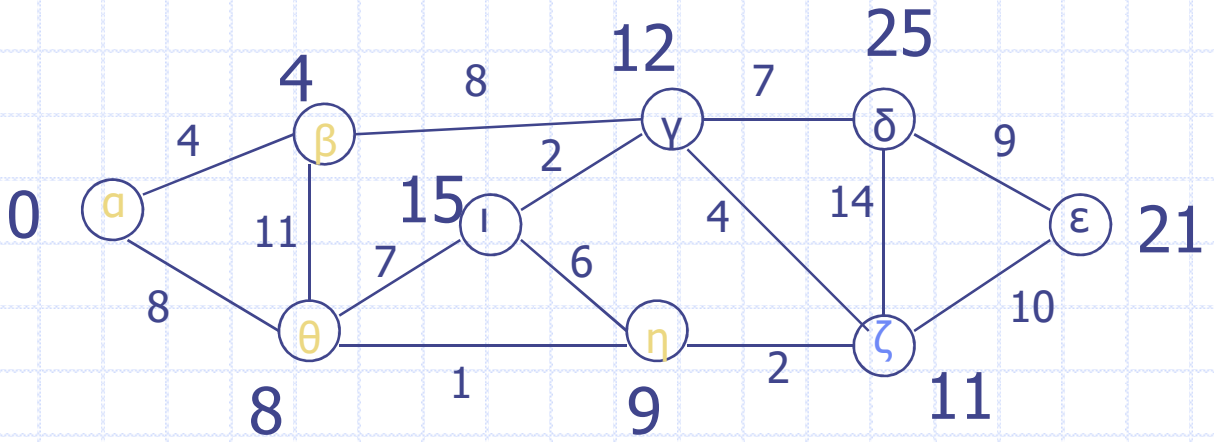


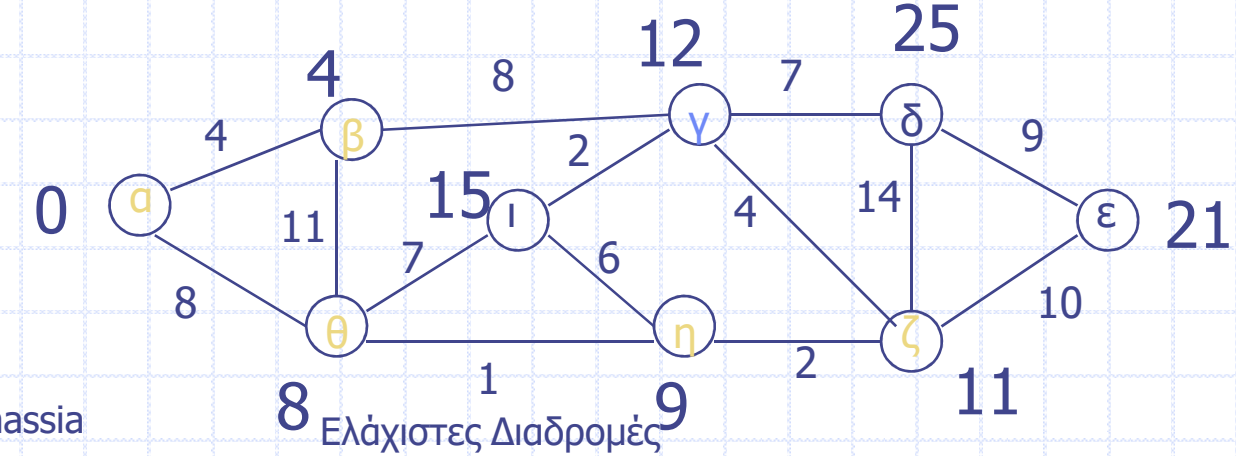

0
α
β
γ
δ
ε
ζ
η
θ
ι
∞
8
∞
7
∞
4
9
2
14
11
4
∞
7
6
8
10
1
2
∞
∞
∞

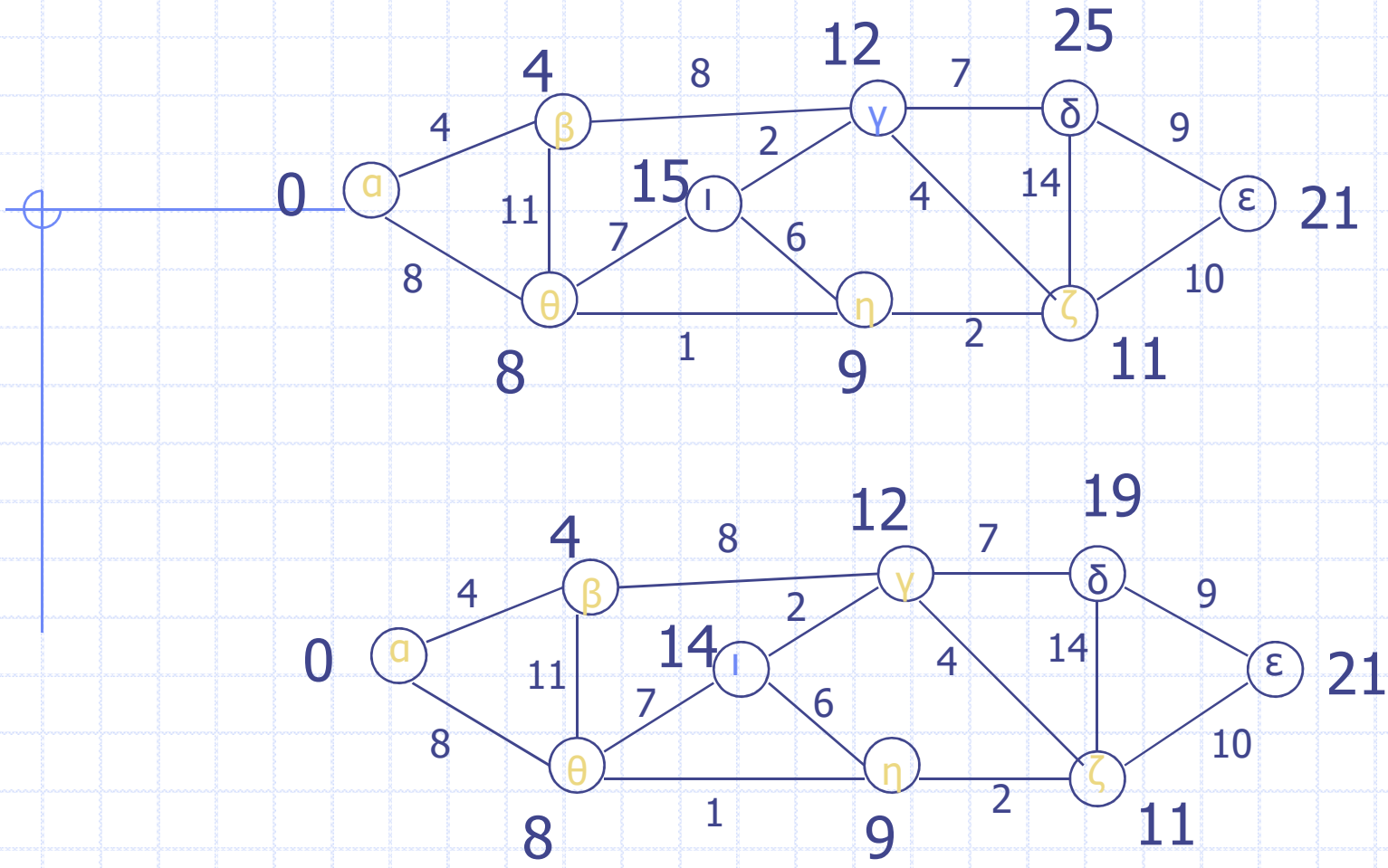

4
0
α
β
γ
δ
ε
ζ
η
θ
ι
8
∞
7
∞
4
9
2
14
11
4
∞
7
6
8
10
1
2
8
∞
∞

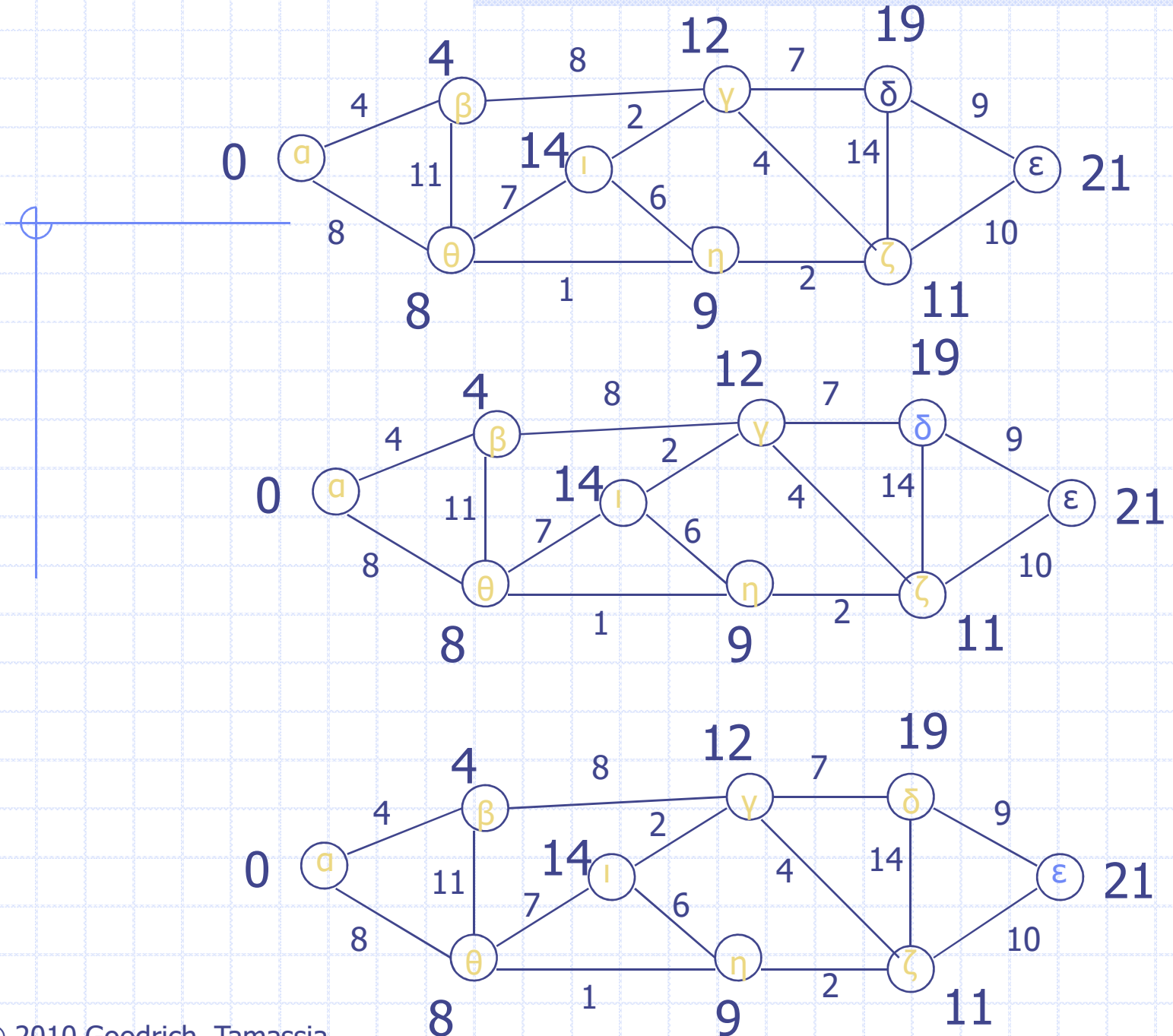

4
12
0
α
β
γ
δ
ε
ζ
η
θ
ι
8
7
∞
4
9
2
14
11
4
∞
7
6
8
10
1
2
8
∞
∞

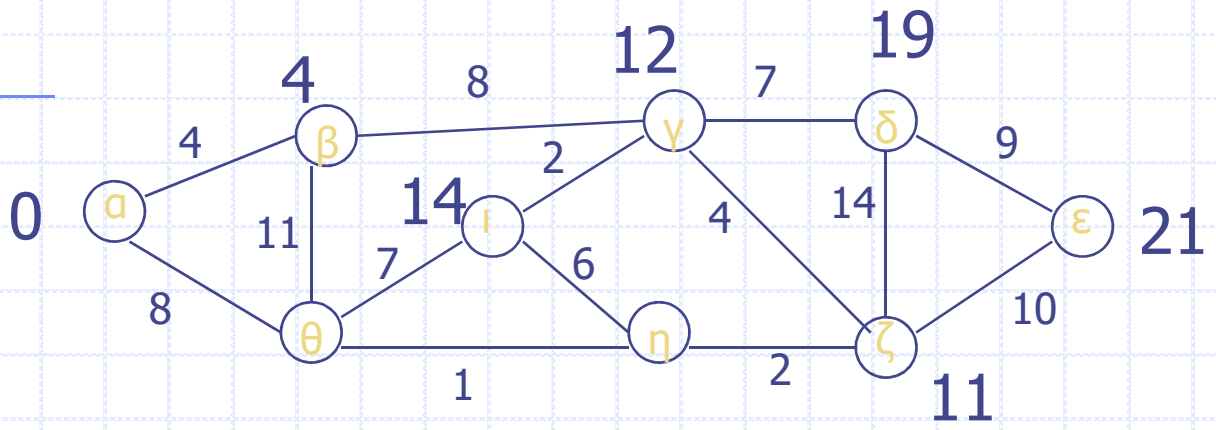

0
4
12
∞
15
∞
8
9
∞
α
β
γ
δ
ε
ι
θ
η
ζ
4
8
7
9
2
11
7
6
4
14
8
10
1
2


0
4
12
25
15
21
8
9
11
α
β
γ
δ
ε
ι
θ
η
ζ
4
8
7
9
2
11
7
6
4
14
8
10
1
2


0
4
12
25
15
21
8
9
11
α
β
γ
δ
ε
ι
θ
η
ζ
4
8
7
9
2
11
7
6
4
14
8
10
1
2


0
α
4
β
8
12
γ
7
25
δ
9
ε
21
15
ι
2
4
14
11
7
6
8
θ
η
ζ
10
1
2
8
9
11
4
8
12
γ
7
19
δ
9
ε
21
14
ι
2
4
14
0
α
4
β
11
7
6
8
θ
η
ζ
10
1
2
8
9
11




0
4
12
19
14
21
11
α
β
γ
δ
ε
ζ
η
θ
ι
4
8
7
9
2
11
14
4
7
6
8
10
1
2

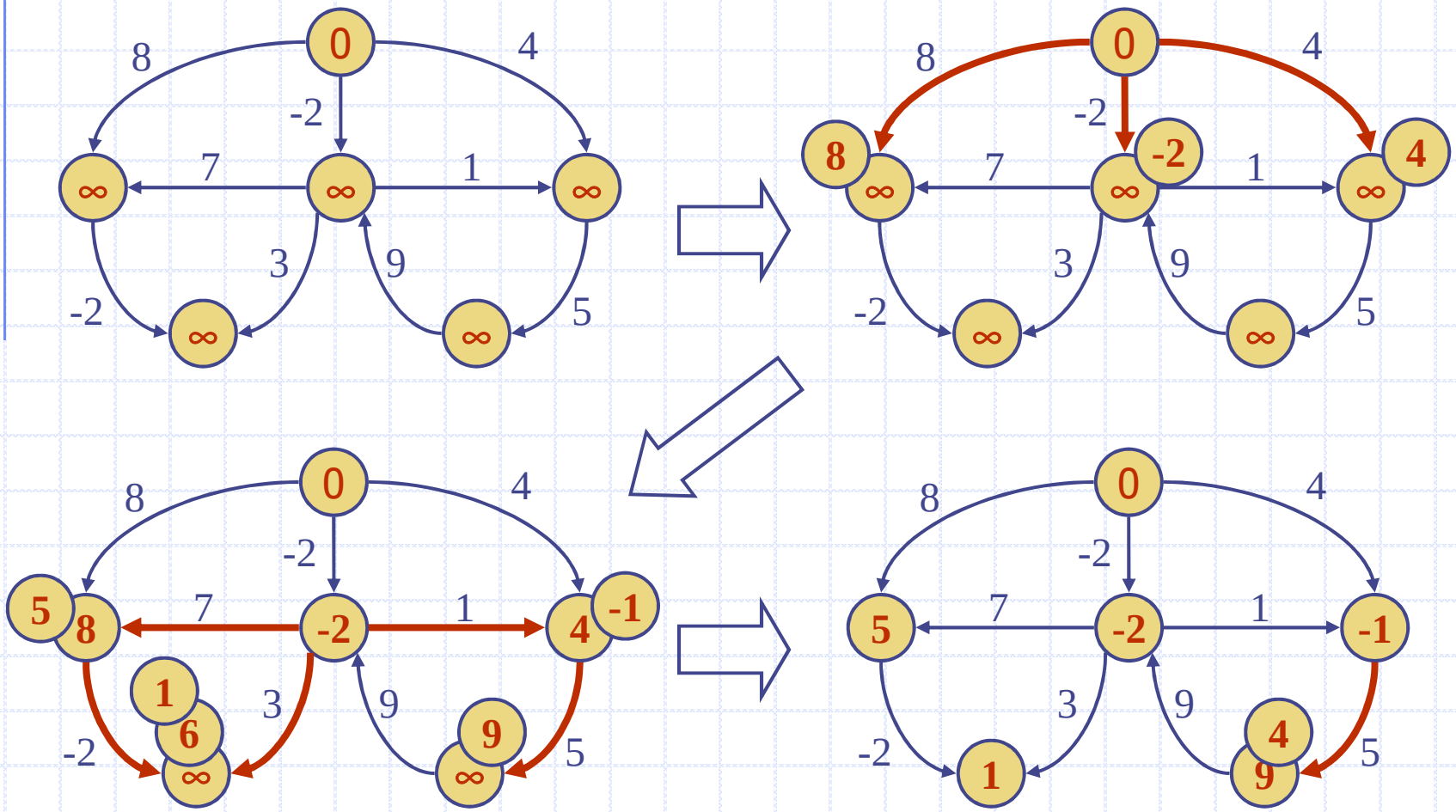
Αλγόριθμος Bellman-Ford (δεν είναι στο βιβλίο)

- Δουλεύει και με αρνητικά βάρη ακμών
- Υποθέτουμε κατευθυνόμενες ακμές (διαφορετικά θα είχαμε κύκλους αρνητικού βάρους)
- Η i επανάληψη βρίσκει όλες τις ελάχιστες διαδρομές που χρησιμοποιούν i ακμές.
- Χρόνος τρεξίματος: O(nm).
- Μπορεί να επεκταθεί για την εύρεση ύπαρξης αρνητικού κύκλου
  - Πως?

```
Algorithm BellmanFord(G, s)
  for all v ∈ G.vertices()
    if v = s
      setDistance(v, 0)
    else
      setDistance(v, ∞)
  for i ← 1 to n − 1 do
    for each e ∈ G.edges()
      { relax edge e }
      u ← G.origin(e)
      z ← G.opposite(u,e)
      r ← getDistance(u) + weight(e)
      if r < getDistance(z)
        setDistance(z,r)
```

Bellman-Ford Example

Οι κόμβοι έχουν ετικέτα την d(v) τιμή τους



Αλγόριθμος που βασίζεται σε DAG (δεν είναι στο βιβλίο)

- Δουλεύει και με αρνητικά βάρη ακμών
- Χρησιμοποιεί τοπολογική διάταξη
- Δεν χρησιμοποιεί κάποια ιδιαίτερη δομή δεδομένων
- Είναι πιο γρήγορος από τον αλγόριθμο Dijkstra's
- Χρόνος τρεξίματος: O(n+m).

```
Algorithm DagDistances(G, s)
  for all v ∈ G.vertices()
    if v = s
      setDistance(v, 0)
    else
      setDistance(v, ∞)
  { Perform a topological sort of the vertices }
  for u ← 1 to n do    {in topological order}
    for each e ∈ G.outEdges(u)
      { relax edge e }
      z ← G.opposite(u,e)
      r ← getDistance(u) + weight(e)
      if r < getDistance(z)
        setDistance(z,r)
```

Παράδειγμα DAG

Οι κόμβοι έχουν ετικέτα την d(v) τιμή τους