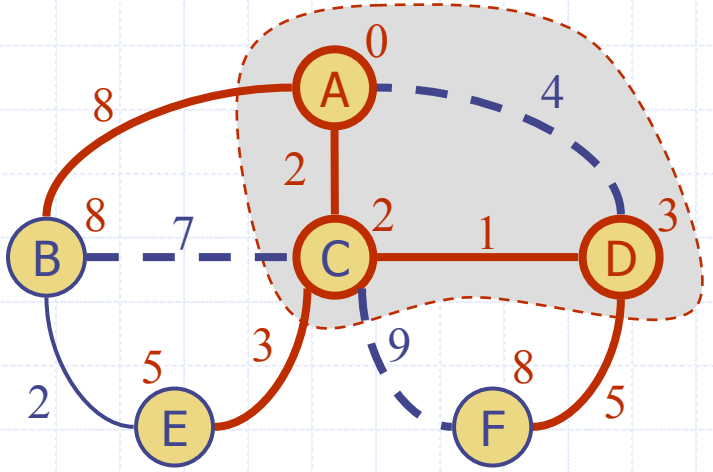
ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Αλγόριθμοι και πολυπλοκότητα

Τα συντομότερα μονοπάτια(Shortest Paths)

Ιωάννης Τόλλης
Τμήμα Επιστήμης Υπολογιστών

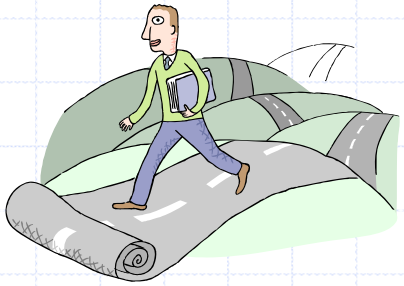
Τα συντομότερα Μονοπάτια(Shortest Paths)



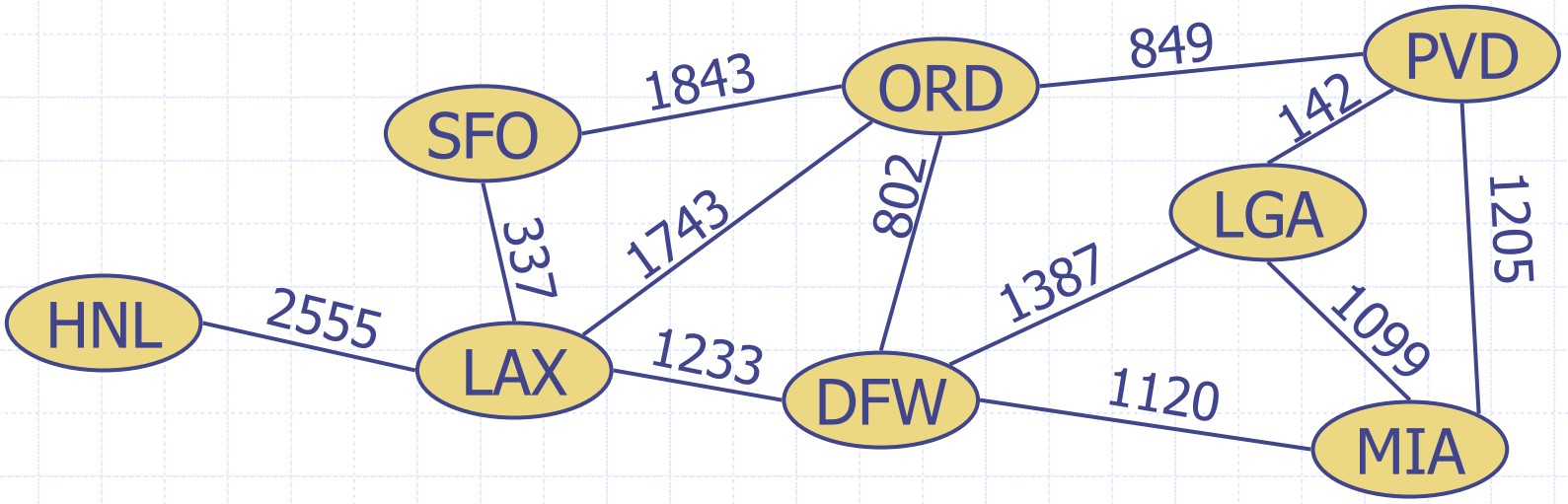
Περίληψη και ανάγνωση

- Βεβαρημένοι Γράφοι
  - Πρόβλημα συντομότερων μονοπατιών
  - Ιδιότητες συντομότερων μονοπατιών
- Αλγόριθμος του Dijkstra
  - Αλγόριθμος
  - Χαλάρωση ακμών
- Ο Αλγόριθμος του Bellman-Ford
- Συντομότερα μονοπάτια σε dags
- Συντομότερα μονοπάτια για όλα τα ζευγάρια

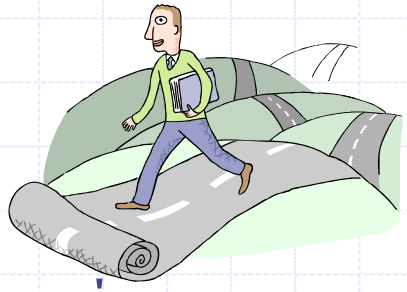
Βεβαρημένοι Γράφοι



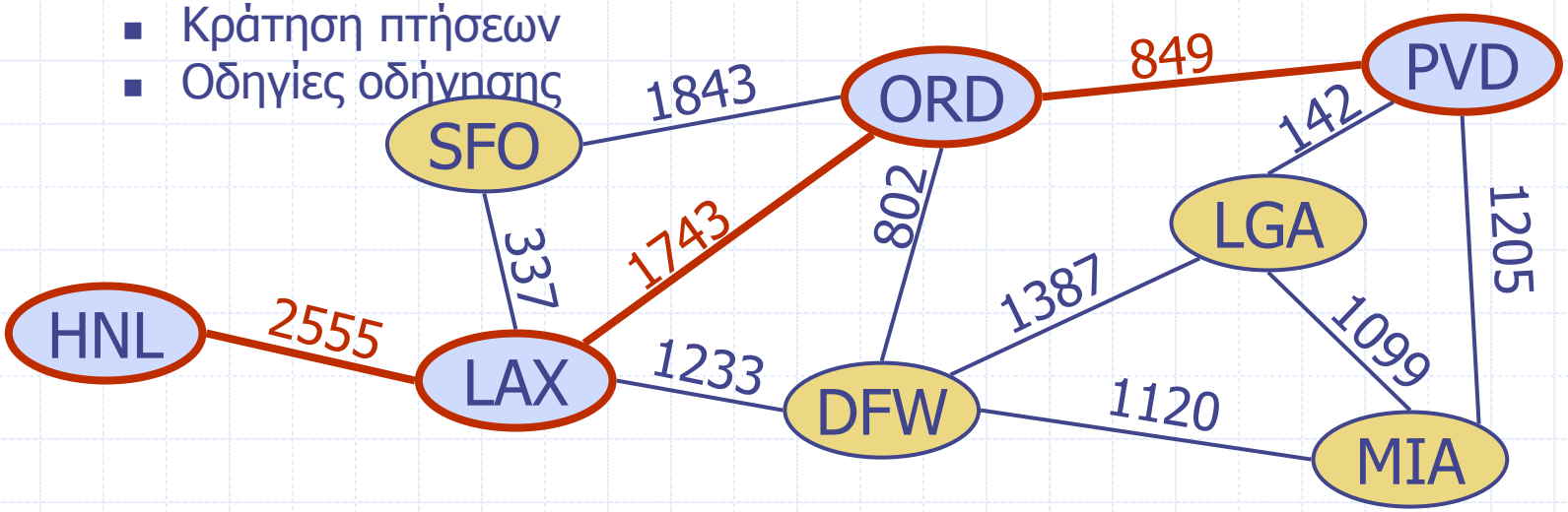
- Σε ένα βεβαρημένο γράφο, κάθε ακμή έχει μία αριθμητική τιμή που σχετίζεται με αυτή, λέγεται το βάρος της ακμής
- Τα βάρη των ακμών μπορεί να αναπαριστούν, αποστάσεις, κόστη, κτλ.
- Παράδειγμα:
  - Σε ένα γράφο πτήσεων, το βάρος μίας ακμής αναπαριστά την απόσταση σε χιλιόμετρα ανάμεσα σε σημεία τέλους αεροδρομίων



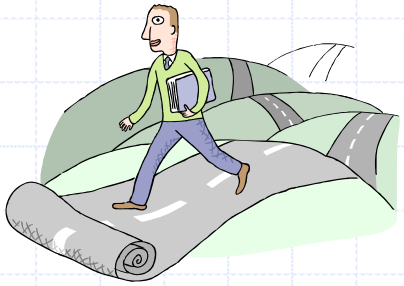
Το πρόβλημα του συντομότερου μονοπατιού



- Δεδομένου ενός βεβαρημένου γράφου και δύο κορυφών u και v, θέλουμε να βρούμε ένα μονοπάτι με ελάχιστο συνολικό βάρος μεταξύ u και v.
  - Μήκος ενός μονοπατιού είναι το άθροισμα των βαρών των ακμών του.
- Παράδειγμα:
  - Συντομότερο μονοπάτι μεταξύ Providence και Honolulu
- Εφαρμογές
  - Δρομολόγηση πακέτων διαδικτύου
  - Κράτηση πτήσεων
  - Οδηγίες οδήγησης



Ιδιότητες συντομότερου μονοπατιού



Ιδιότητα 1:

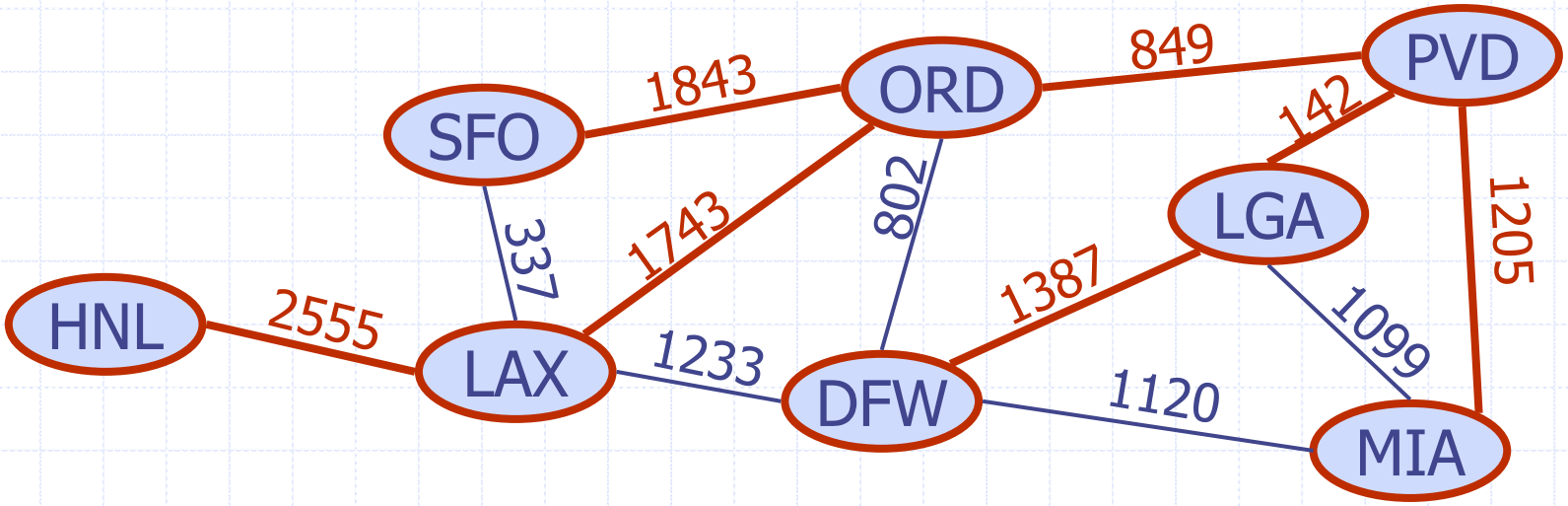
Ένα υπομονοπάτι του συντομότερου μονοπατιού είναι από μόνο του ένα συντομότερο μονοπάτι

Ιδιότητα 2:

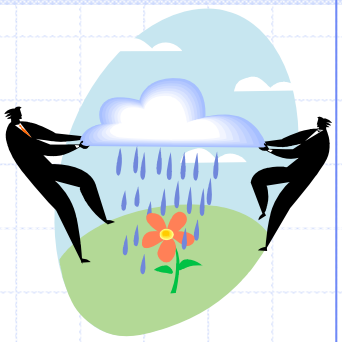
Υπάρχει ένα δέντρο συντομότερα μονοπάτια από μία αρχική κορυφή σε όλες τις άλλες κορυφές

Παράδειγμα:

Δέντρο συντομότερων μονοπατιών από την Πρόνοια(Providence)



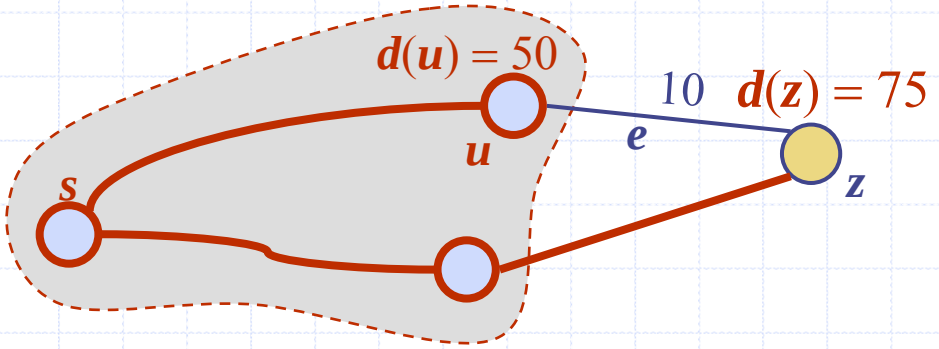
Ο Αλγόριθμος του Dijkstra



- Η απόσταση μίας κορυφής v από μία κορυφή s είναι το μήκος ενός συντομότερου μονοπατιού μεταξύ s και v
- Ο αλγόριθμος του Dijkstra υπολογίζει τις αποστάσεις όλων των κορυφών από μία δοθείσα αρχική κορυφή s
- Υποθέσεις:
  - Ο γράφος είναι συνδεδεμένος
  - Οι ακμές δεν έχουν κατεύθυνση
  - Τα βάρη των ακμών είναι μη αρνητικά
- Φτιάχνουμε ένα "**σύννεφο**" από κορυφές, ξεκινώντας από την s και τελικά καλύπτοντας όλες τις ακμές
- Κρατάμε για κάθε κορυφή v μία ετικέτα $d(v)$ που αναπαριστά την απόσταση της v από την s στον υπογράφο που αποτελείται από το σύννεφο και τις παρακείμενες κορυφές του
- Σε κάθε βήμα
  - Προσθέτουμε στο σύννεφο την κορυφή u έξω από το σύννεφο με την μικρότερη ετικέτα απόστασης, $d(u)$
  - Ανανεώνουμε τις ετικέτες των παρακείμενων κορυφών της u

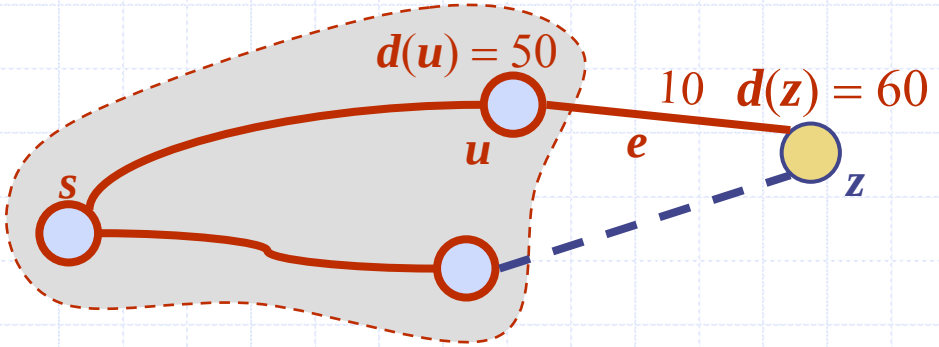
Χαλάρωση ακμών

- Σκεφτείτε μία ακμή $e = (u,z)$ όπως
  - Η u είναι η κορυφή που προστέθηκε πιο πρόσφατα στο σύννεφο
  - Η z δεν είναι στο σύννεφο

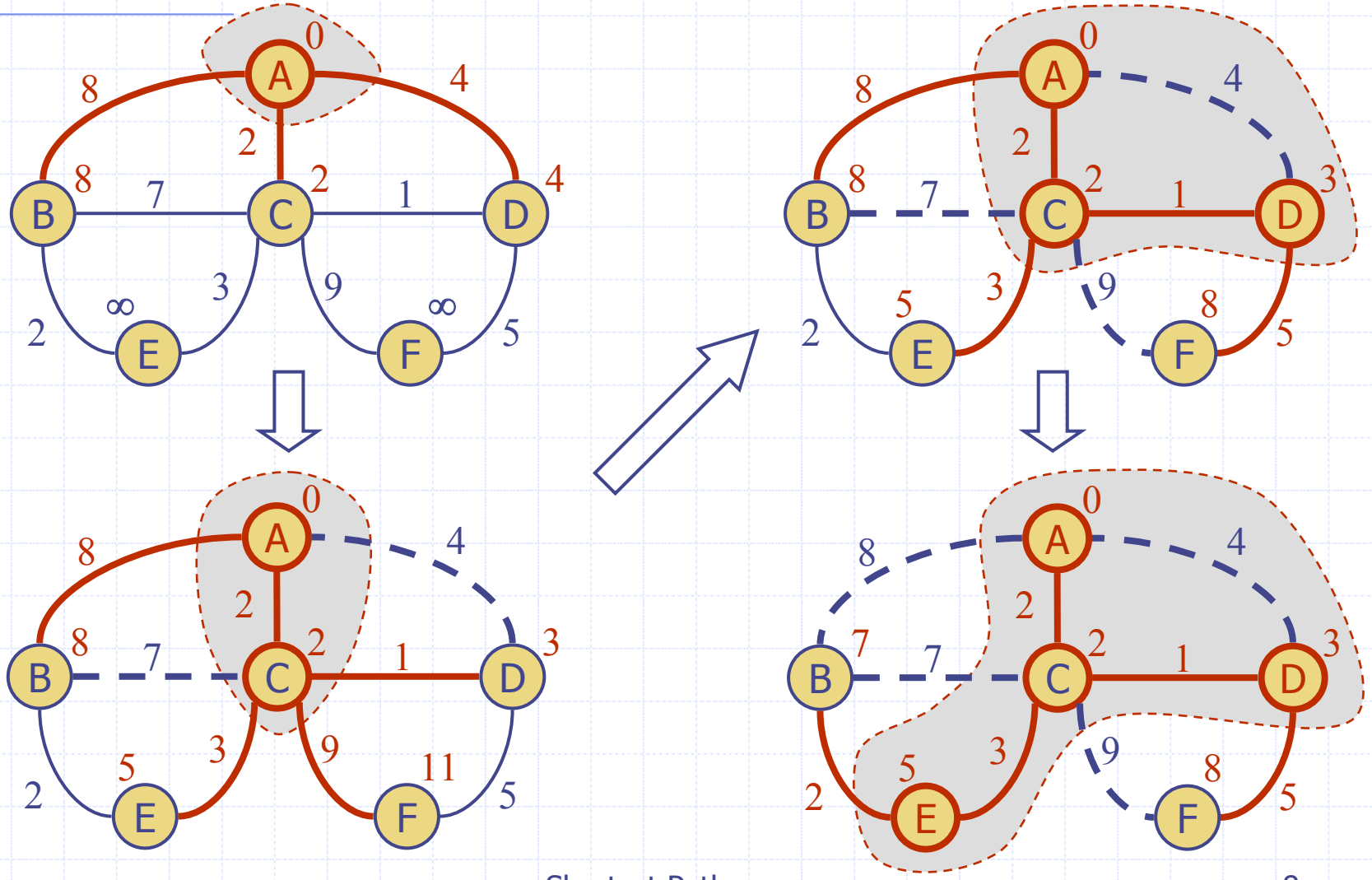


- Η χαλάρωση της ακμής e ανανεώνει την απόσταση $d(z)$ ως εξής :

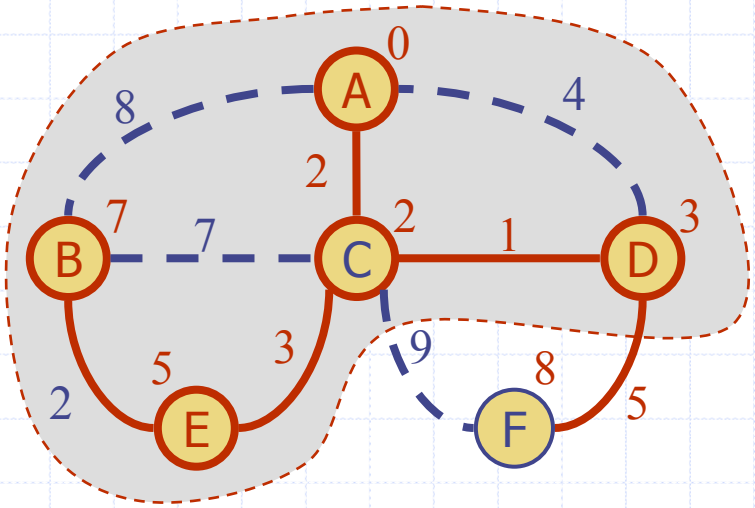
$d(z) \leftarrow \min\{d(z), d(u) + weight(e)\}$

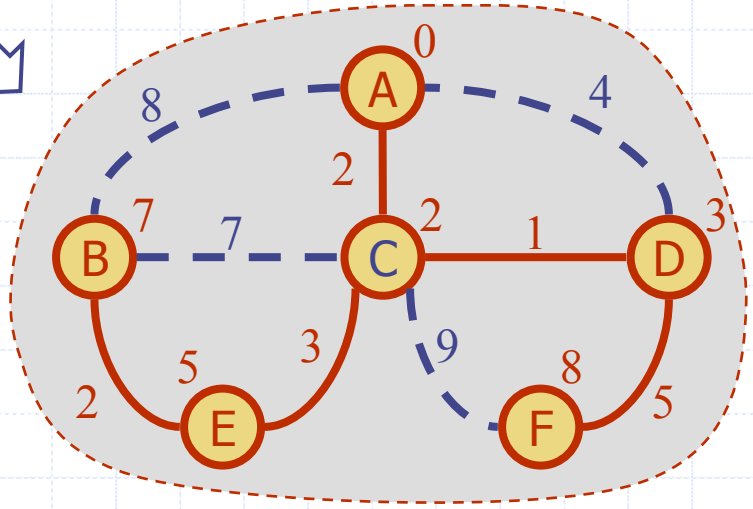


Παράδειγμα

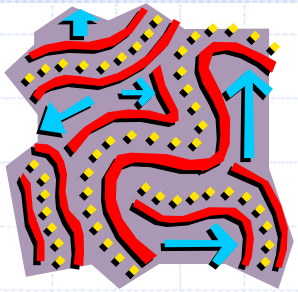


Παράδειγμα (συνέχεια)





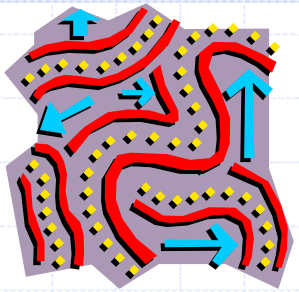
Ο Αλγόριθμος του Dijkstra



- Μία ουρά προτεραιότητας κρατάει τις κορυφές έξω από το σύννεφο
  - Κλειδί: απόσταση
  - Στοιχείο: κορυφή
- Μέθοδοι βασισμένες στον εντοπισμό
  - *insert*(*k*,*e*) επιστρέφει έναν locator
  - *replaceKey*(*l*,*k*) αλλάζει το κλειδί ενός στοιχείου
- Κρατάμε δύο ετικέτες για κάθε κορυφή:
  - Την απόσταση (ετικέτα d(v))
  - Τον locator σε ουρά προτεραιότητας

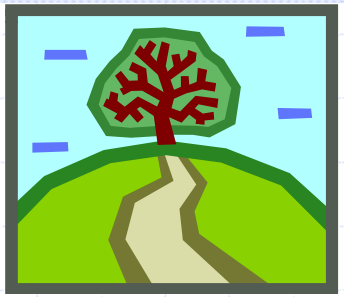
```
Algorithm DijkstraDistances(G, s)
    Q ← new heap-based priority queue
    for all v ∈ G.vertices()
        if v = s
            setDistance(v, 0)
        else
            setDistance(v, ∞)
        l ← Q.insert(getDistance(v), v)
        setLocator(v,l)
    while ¬Q.isEmpty()
        u ← Q.removeMin()
        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)
                Q.replaceKey(getLocator(z),r)
```

Ανάλυση



- Λειτουργίες γράφου
  - Η μέθοδος incidentEdges καλείται μία φορά once για κάθε κορυφή
- Λειτουργίες ετικετών
  - Θέτουμε/παίρνουμε τις ετικέτες της απόσταση και του locator της κορυφής z $O(\deg(z))$ φορές
  - θέτοντας/παίρνοντας μία ετικέτα κοστίζει $O(1)$ χρόνο
- Λειτουργίες της ουράς προτεραιότητας
  - Κάθε κορυφή εισάγεται μία φορά και διαγράφεται μία φορά από ουράς προτεραιότητας, όπου κάθε εισαγωγή και εξαγωγή κοστίζει $O(\log n)$ χρόνο
  - Το κλειδί της κορυφής στην ουρά προτεραιότητας τροποποιείται το πολύ $\deg(w)$ φορές, όπου κάθε αλλαγή κλειδιού παίρνει $O(\log n)$χρόνο
- Ο αλγόριθμος του Dijkstra τρέχει σε $O((n + m) \log n)$ χρόνο υπό τον όρο ότι η γραφική παράσταση αντιπροσωπεύεται από τη δομή καταλόγων γειτνίασης
  - Θυμίζουμε ότι $\sum_v \deg(v) = 2m$
- Ο χρόνος εκτέλεσης μπορεί επίσης να εκφραστεί σαν $O(m \log n)$ καθώς ο γράφος είναι συνδεδεμένος

Επέκταση



- Χρησιμοποιώντας την πρότυπη μέθοδο σχεδίασης, μπορούμε να επεκτείνουμε τον αλγόριθμο του Dijkstra ώστε να επιστρέφει ένα δέντρο συντομότερων μονοπατιών από την αρχική κορυφή σε όλες τις άλλες κορυφές
- Κρατάμε με κάθε κορυφή μια τρίτη ετικέτα:
  - Πατρική ακμή στο δέντρο συντομότερου μονοπατιού
- Στο βήμα χαλάρωσης ακμών, ανανεώνουμε την πατρική ετικέτα

Algorithm *DijkstraShortestPathsTree*(*G*, *s*)

…

for all $v \in G.vertices()$

…

setParent(*v*, ∅)

…

for all $e \in G.incidentEdges(u)$

{ χαλάρωση ακμής *e* }

$z \leftarrow G.opposite(u,e)$

$r \leftarrow getDistance(u) + weight(e)$

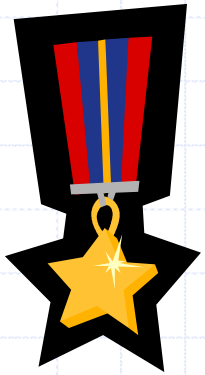
if $r < getDistance(z)$

setDistance(*z*,*r*)

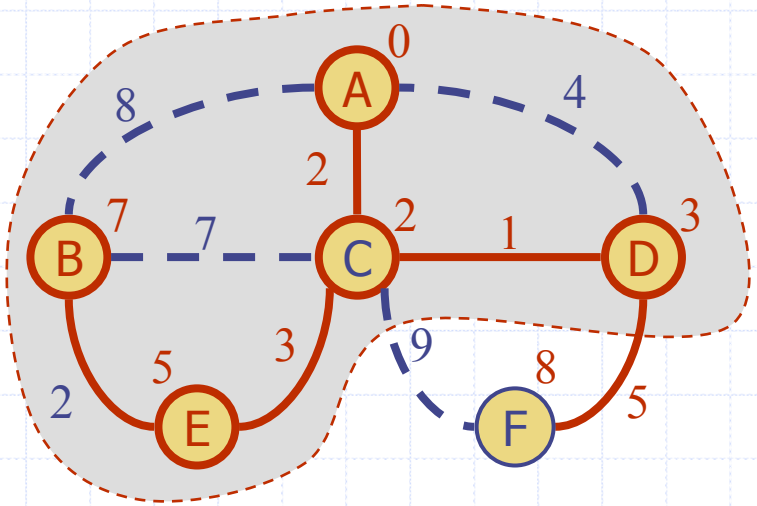
setParent(*z*,*e*)

Q.replaceKey(*getLocator*(*z*),*r*)

Γιατί ο αλγόριθμος του Dijkstra δουλεύει

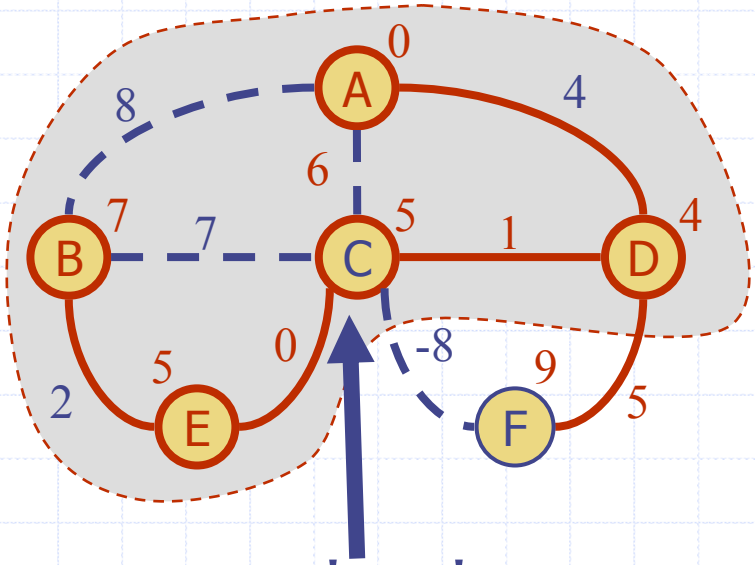


- Ο αλγόριθμος του Dijkstra βασίζεται στην άπληστη μέθοδο. Προσθέτει κορυφές αυξάνοντας την απόσταση.
  - Υποθέστε ότι δεν βρίσκει όλες τις μικρότερες αποστάσεις. Έστω F η πρώτη λαθεμένη κορυφή που επεξεργάζεται ο αλγόριθμος.
  - Όταν ο προηγούμενος κόμβος, D, στο άληθινό συντομότερο μονοπάτι είχε λειφθεί υπόψη, η απόστασή του ήταν σωστή.
  - Αλλά η ακμή (D,F) ήταν **χαλαρωμένη** εκείνη τη στιγμή!
  - Έτσι, εφόσον d(F)≥d(D), Η απόσταση της F δεν μπορεί να είναι λάθος. Αυτό ήταν, δεν υπάρχει λάθος κορυφή.



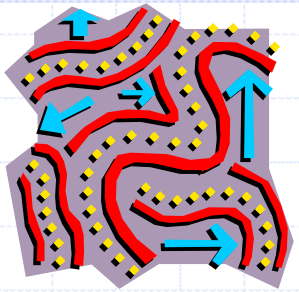
Γιατί δεν δουλεύει για Αρνητικά βάρη ακμών

- Ο αλγόριθμος του Dijkstra βασίζεται στην άπληστη μέθοδο. Προσθέτει κορυφές αυξάνοντας την απόσταση.
  - Αν ένας κόμβος με αρνητική συναφή ακμή επρόκειτο να προτεθεί αργά στο σύννεφο, θα μπορούσε να «βρωμίσει» τις αποστάσεις για τις κορυφές ήδη στο σύννεφο.



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

Ο αλγόριθμος του Bellman-Ford

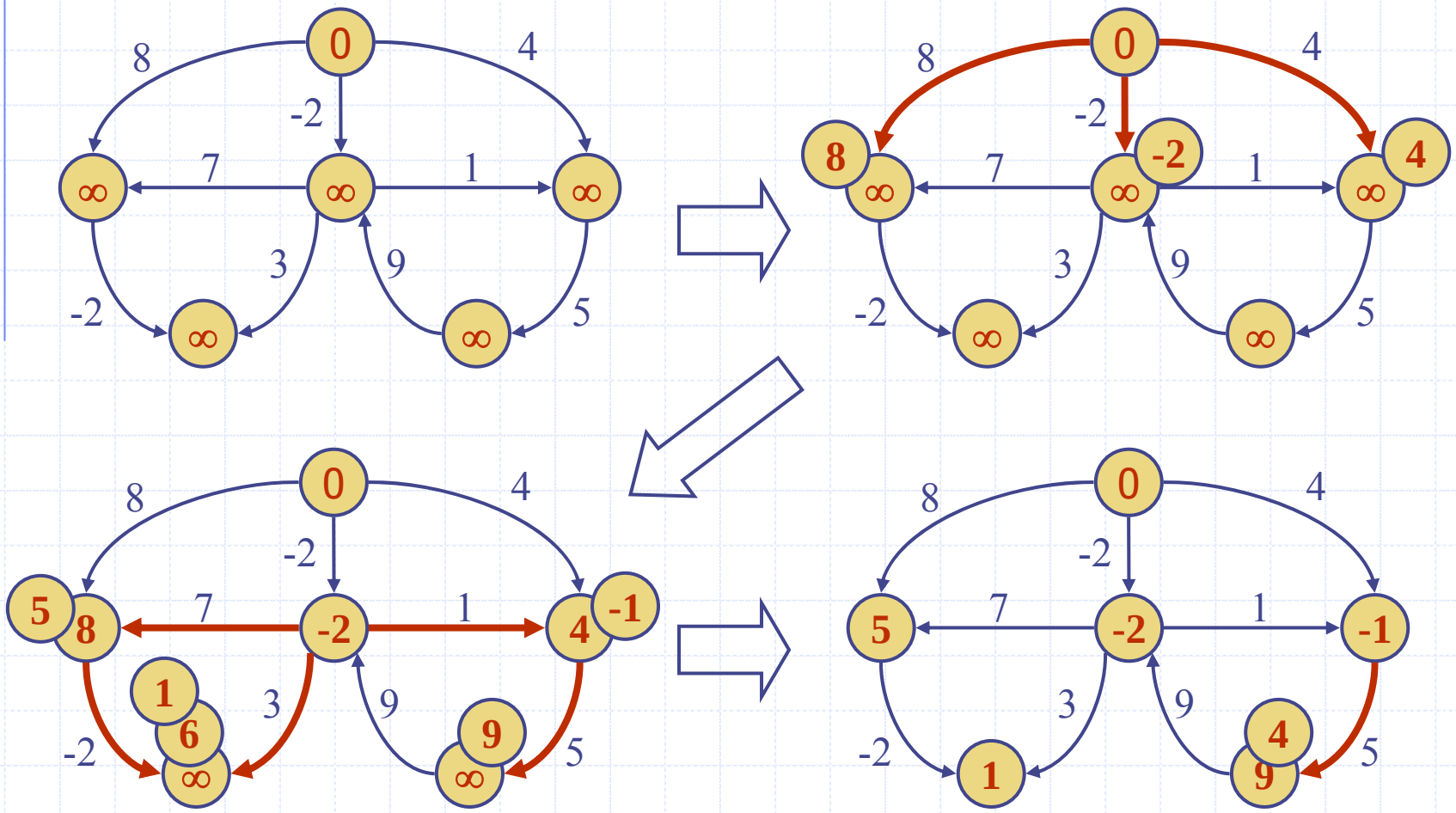


- Δουλεύει ακόμα και με αρνητικά βάρη ακμών
- Πρέπει να υποθέσουμε οτι οι ακμές είναι κατευθυνόμενες (ειδάλλως θα έχουμε αρνητικά βεβαρημένους κύκλους)
- Η επανάληψη (το Iteration) i βρίσκει όλα τα συντομότερα μονοπάτια που χρησιμοποιούν i ακμές.
- Χρόνος εκτέλεσης: O(nm).
- Μπορούμε να επεκταθούμε για να ανιχνεύσουμε έναν αρνητικά βεβαρημένο κύκλο αν υπάρχει
  - πώς?

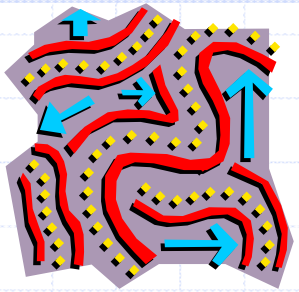
```
Algorithm BellmanFord(G, s)
    for all v ∈ G.vertices()
        if v = s
            setDistance(v, 0)
        else
            setDistance(v, ∞)
    for i ← 1 to n-1 do
        for each e ∈ G.edges()
            { χαλάρωση ακμής e }
            u ← G.origin(e)
            z ← G.opposite(u,e)
 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

Παράδειγμα Bellman-Ford

Οι κόμβοι ονομάζονται με τις d(v) τιμές τους



Αλγόριθμος DAG-based

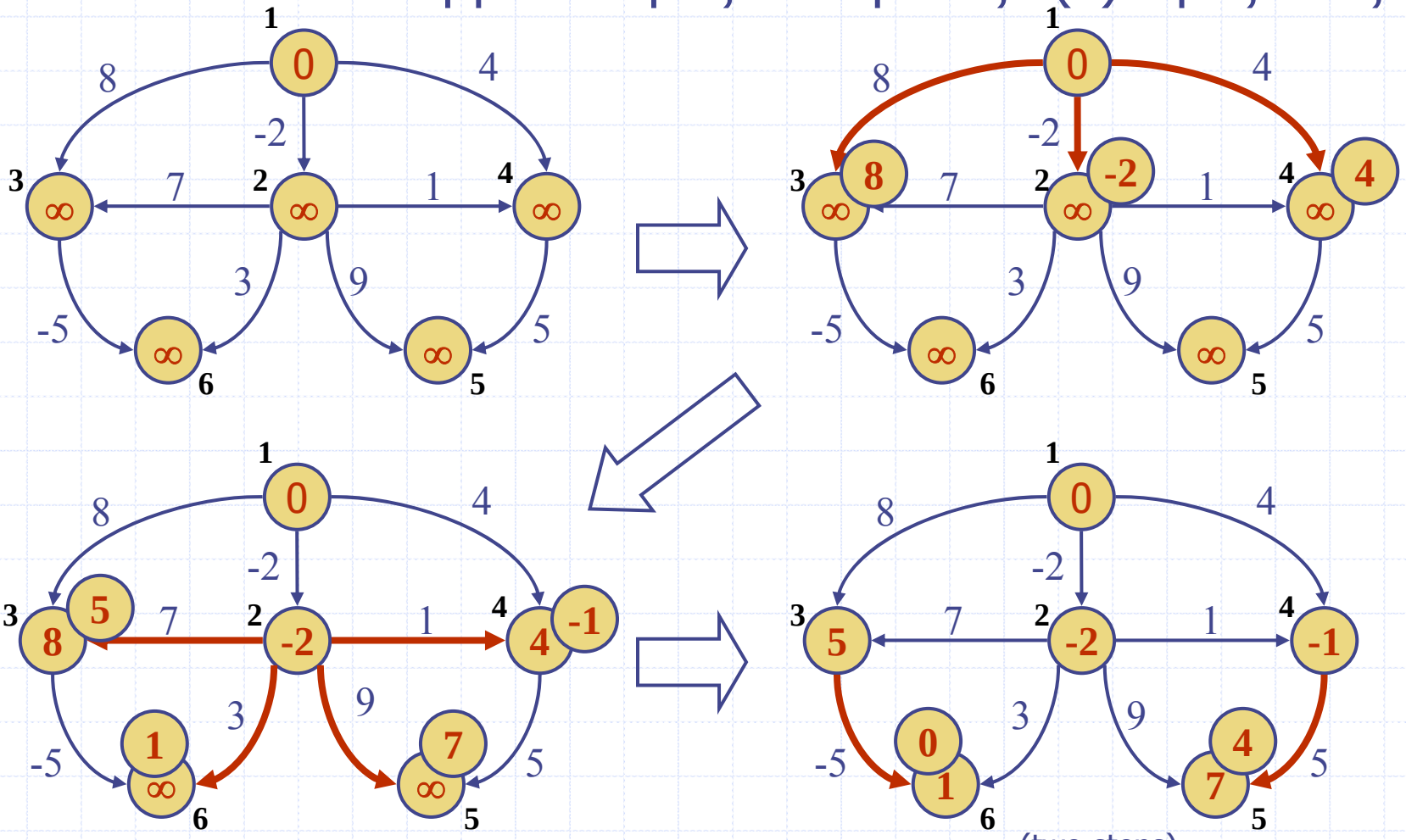


- Δουλεύει ακόμα και με αρνητικά βάρη ακμών
- Χρησιμοποιεί τοπολογική σειρά
- Δεν χρησιμοποιεί φανταχτερές δομές δεδομένων
- Είναι πολύ γρηγορότερος από τον αλγόριθμο του Dijkstra
- Χρόνος εκτέλεσης: O(n+m).

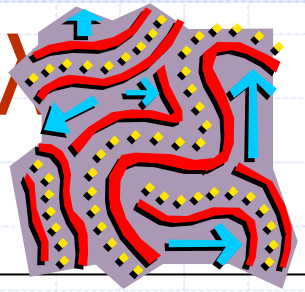
```
Algorithm DagDistances(G, s)
    for all v ∈ G.vertices()
        if v = s
            setDistance(v, 0)
        else
            setDistance(v, ∞)
    Perform a topological sort of the vertices
    for u ← 1 to n do    {σε τοπολογική σειρά}
        for each e ∈ G.outEdges(u)
            { χαλάρωση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

Παράδειγμα DAG

Οι κόμβοι ονομάζονται με τις d(v) τιμές τους



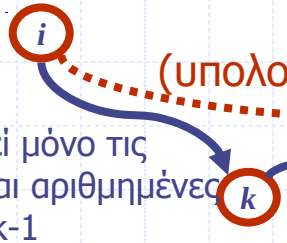
Συντομότερα μονοπάτια για όλα τα ζευγάρια



- Βρίσκει την απόσταση μεταξύ κάθε ζεγαριού κορυφών σε ένα βεβαρημένο κατευθυνόμενο γράφο G.
- Μπορούμε να κάνουμε n κλήσεις του αλγόριθμου του Dijkstra(αν δεν υπάρχουν αρνητικές ακμές), το οποίο παίρνει O(nmlog n) χρόνο.
- Επιπλέον, οι n κλήσεις του Bellman-Ford μπορούν να πάρουν $O(n^2m)$ χρόνο.
- Μπορούμε να επιτύχουμε $O(n^3)$ χρόνο χρησιμοποιώντας δυναμικό προγραμματισμό (παρόμοιο με τον αλγόριθμο του Floyd-Warshall).

Algorithm *AllPair(G)* {υποθέστε κορυφές 1,…,*n*}
for all *vertex pairs (i,j)*
  if $i = j$
    $D_0[i,i] \leftarrow 0$
  else if *(i,j) is an edge in G*
    $D_0[i,j] \leftarrow$ *weight of edge (i,j)*
  else
    $D_0[i,j] \leftarrow +\infty$
for $k \leftarrow 1$ **to** n **do**
  for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** n **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k]+D_{k-1}[k,j]\}$
return D_n

Χρησιμοποιεί μόνο τις κορυφές που είναι αριθμημένες 1,…,k
(υπολογίζει το βάρος αυτής της ακμής)



Χρησιμοποιεί μόνο τις κορυφές που είναι αριθμημένες 1,…,k-1

Χρησιμοποιεί μόνο τις κορυφές που είναι αριθμημένες 1,…,k-1

Τέλος Ενότητας

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Σημειώματα

Σημείωμα αδειοδότησης

- Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση, Όχι Παράγωγο Έργο 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».

[1] http://creativecommons.org/licenses/by-nc-nd/4.0/

- Ως Μη Εμπορική ορίζεται η χρήση:
  - που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
  - που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
  - που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

- Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Σημείωμα Αναφοράς

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.