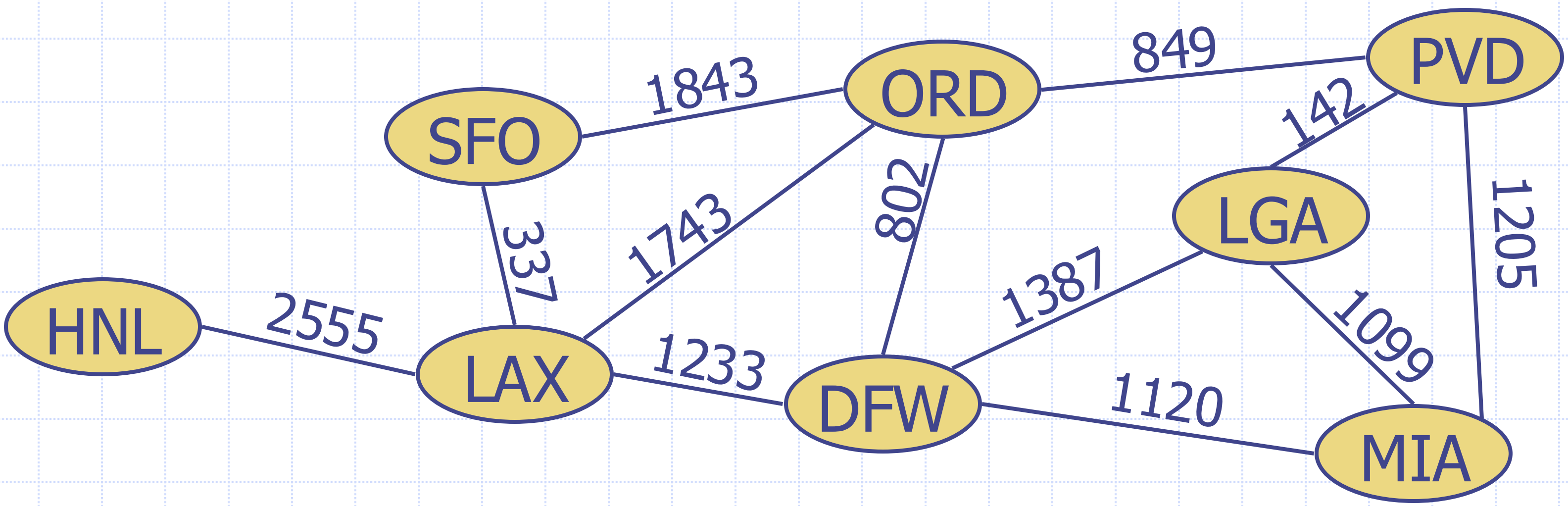
Παρουσίαση για χρήση με το σύγγραμμα, Αλγόριθμοι Σχεδίαση και Εφαρμογές, των M. T. Goodrich and R. Tamassia, Wiley, 2015 (στα ελληνικά από εκδόσεις Μ. Γκιούρδας)

Συντομότερες διαδρομές

Lightning strike, 2009. U.S. government image. NOAA.

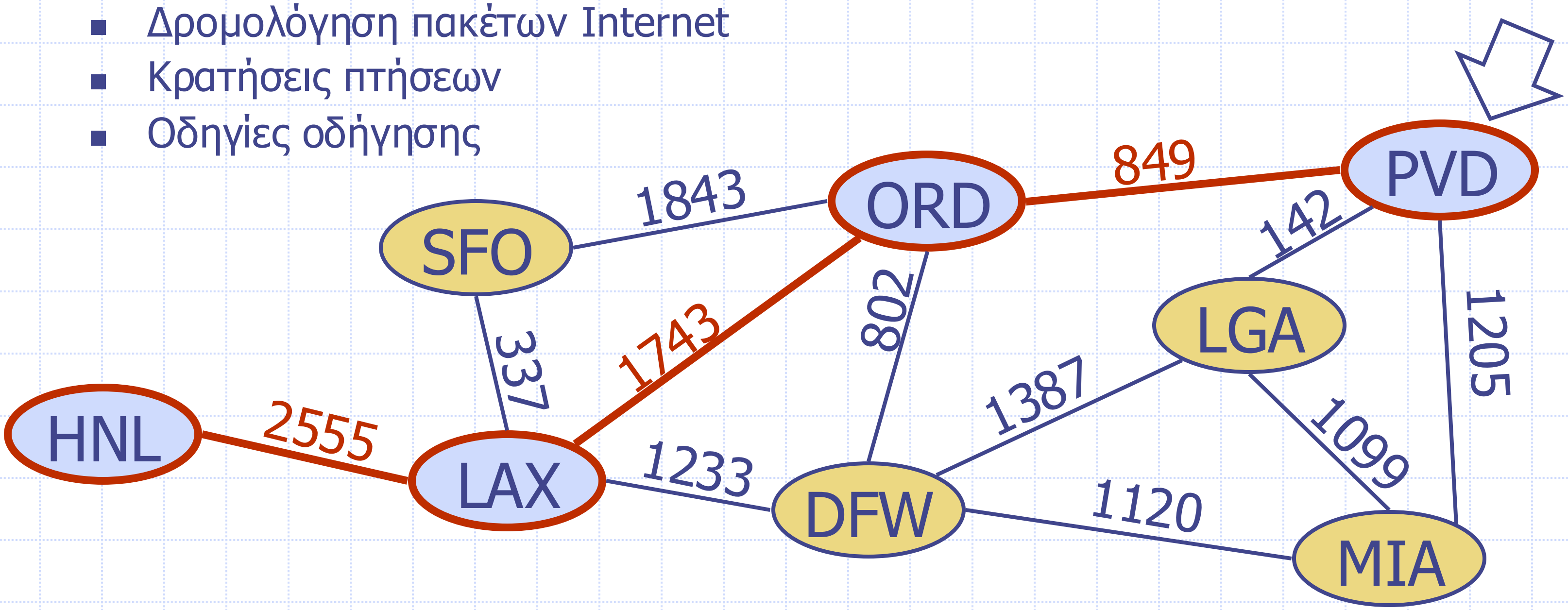
Σταθμισμένοι γράφοι

- Σε έναν σταθμισμένο γράφο (weighted graph), κάθε ακμή συσχετίζεται με μία αριθμητική τιμή, που ονομάζεται βάρος της ακμής.
- Τα βάρη των ακμών μπορούν να αντιπροσωπεύουν αποστάσεις, κόστη, κ.λπ.
- Παράδειγμα:
  - Σε ένα γράφο αεροπορικών πτήσεων, το βάρος μίας ακμής αντιπροσωπεύει την απόσταση σε μίλια μεταξύ των αεροδρομίων στα άκρα της ακμής



Συντομότερες διαδρομές

- Δεδομένου ενός σταθμισμένου γράφου και δύο κορυφών ***u*** και ***v*** θέλουμε να βρούμε τη διαδρομή με το μικρότερο συνολικό βάρος μεταξύ των κορυφών ***u*** και ***v***.
  - Το μήκος της διαδρομής είναι το άθροισμα των βαρών των ακμών που περιέχει.
- Παράδειγμα:
  - Η συντομότερη διαδρομή μεταξύ Providence και Honolulu
- Εφαρμογές
  - Δρομολόγηση πακέτων Internet
  - Κρατήσεις πτήσεων
  - Οδηγίες οδήγησης



Ιδιότητες συντομότερης διαδρομής

Ιδιότητα 1:

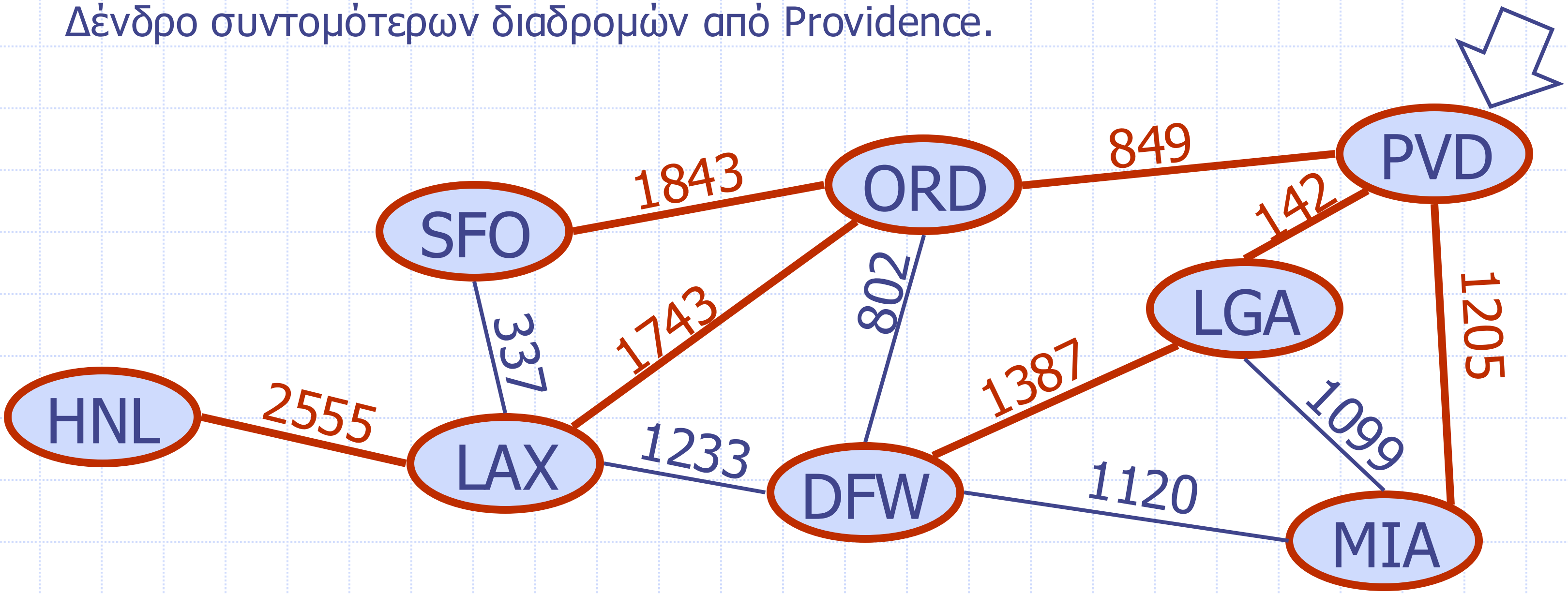
Μια υπό-διαδρομή της συντομότερης διαδρομής είναι η ίδια μια συντομότερη διαδρομή.

Ιδιότητα 2:

Υπάρχει ένα δένδρο συντομότερων διαδρομών από την αρχική κορυφή προς όλες τις άλλες κορυφές.

Παράδειγμα:

Δένδρο συντομότερων διαδρομών από Providence.



Ο αλγόριθμος του Dijkstra

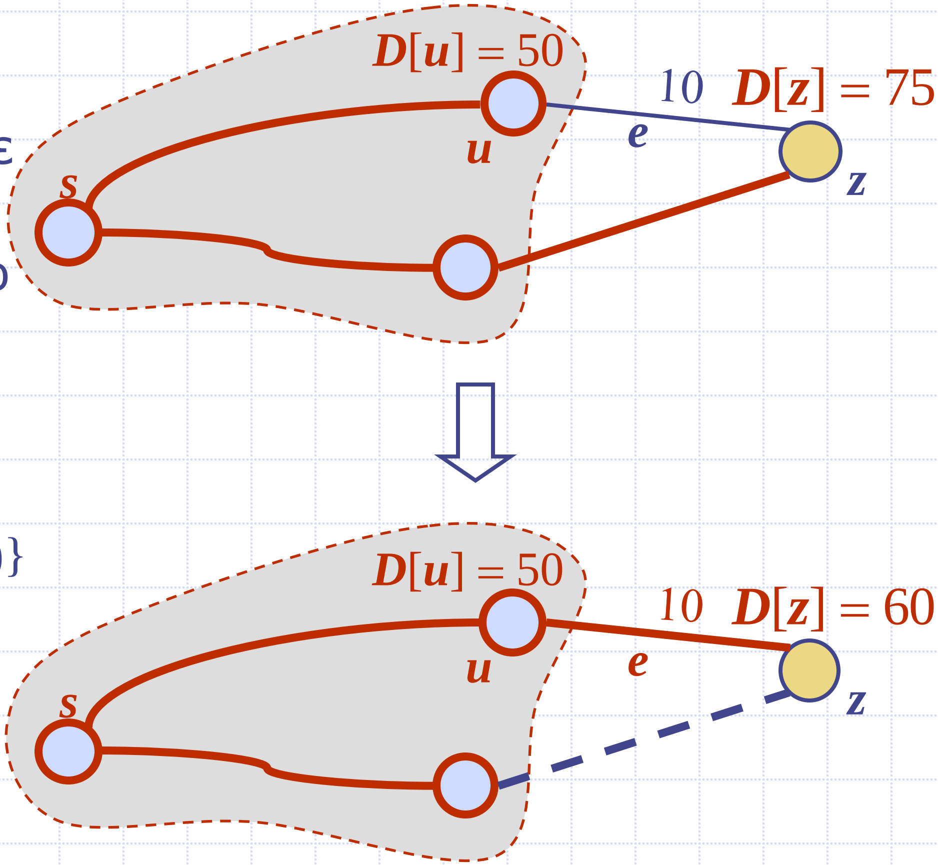
- Η απόσταση μίας κορυφής v από μία κορυφή s είναι το μήκος της συντομότερης διαδρομής ανάμεσα στις s και v
- Ο αλγόριθμος του Dijkstra υπολογίζει τις αποστάσεις όλων των κορυφών από μια δεδομένη αρχική κορυφή s
- Υποθέσεις:
  - ο γράφος είναι συνεκτικός
  - οι ακμές είναι μη κατευθυνόμενες
  - τα βάρη των ακμών είναι μη αρνητικά
- Ξεκινώντας από την κορυφή s αναπτύσσουμε ένα «σύννεφο» κορυφών, που εν τέλει θα συμπεριλάβει όλες τις κορυφές.
- Αποθηκεύουμε με κάθε κορυφή v μια ετικέτα $D[v]$ που αντιπροσωπεύει την απόσταση της κορυφής v από την s στον υπογράφο που αποτελείται από το σύννεφο και τις γειτονικές του κορυφές
- Σε κάθε βήμα:
  - Προσθέτουμε στο σύννεφο την κορυφή u εκτός του σύννεφου με την μικρότερη ετικέτα απόστασης, $D[u]$
  - Ενημερώνουμε τις ετικέτες των γειτονικών ακμών της u

Χαλάρωση ακμών

Χαλάρωση ακμής: ενημέρωση σε καλύτερη τιμή της συντομότερης διαδρομής αν χρησιμοποιηθεί η ακμή

- Θεωρήστε μία ακμή $e = (u, z)$ έτσι ώστε:
  - η u να είναι η κορυφή που πλέον πρόσφατα προστέθηκε στο σύννεφο
  - η z να μην είναι στο σύννεφο
- Η διαδικασία χαλάρωσης της ακμής e ενημερώνει την απόσταση $D[z]$ ως εξής:

$$D[z] \leftarrow \min\{D[z], D[u] + weight(e)\}$$



Ο αλγόριθμος του Dijkstra

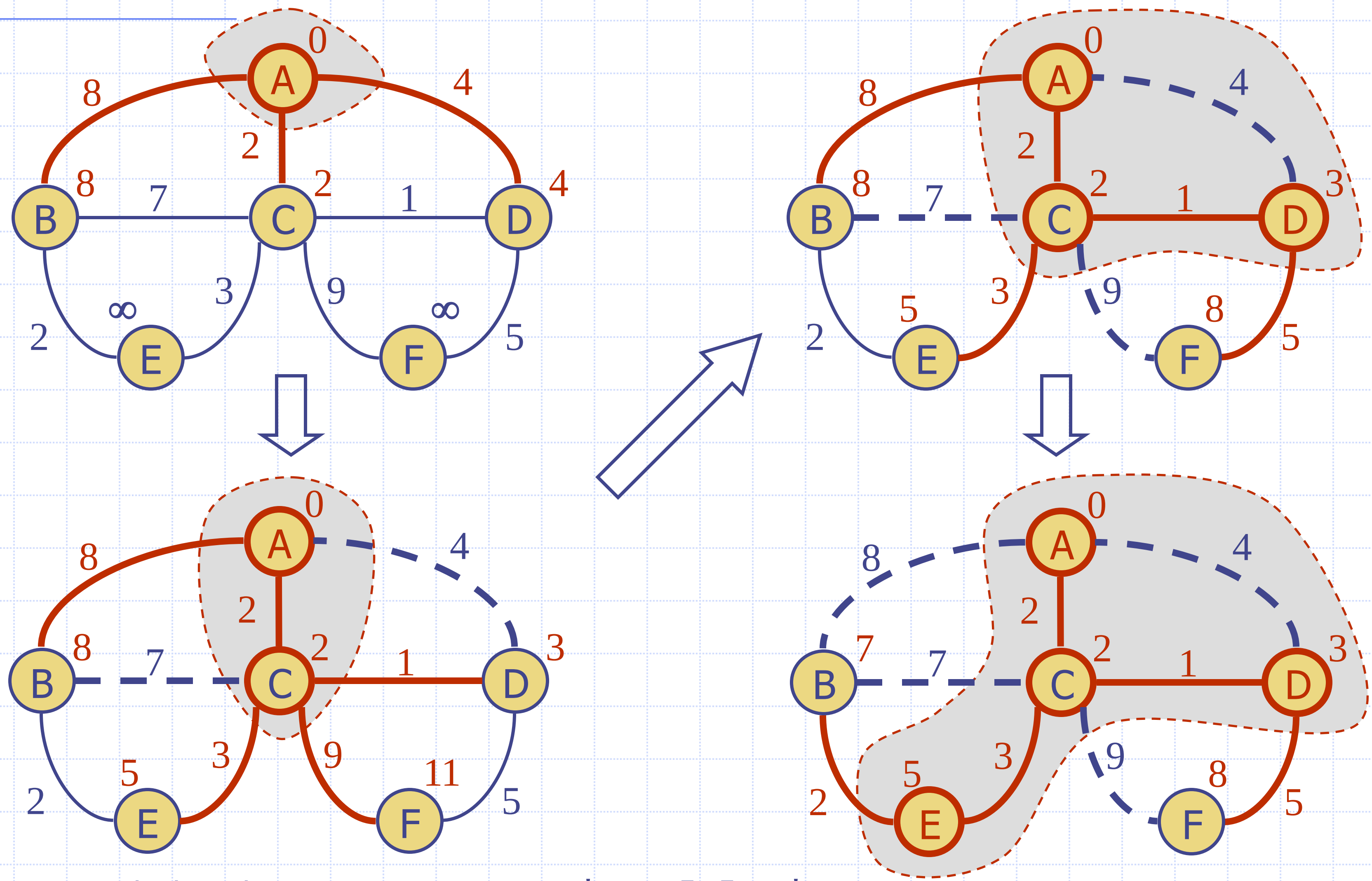
Algorithm DijkstraShortestPaths(G, v):

Input: A simple undirected weighted graph G with nonnegative edge weights, and a distinguished vertex v of G

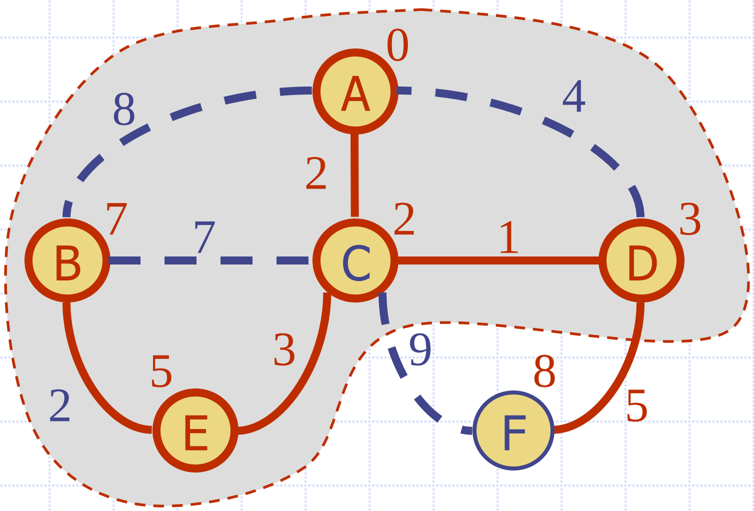
Output: A label, $D[u]$, for each vertex u of G, such that $D[u]$ is the distance from v to u in G

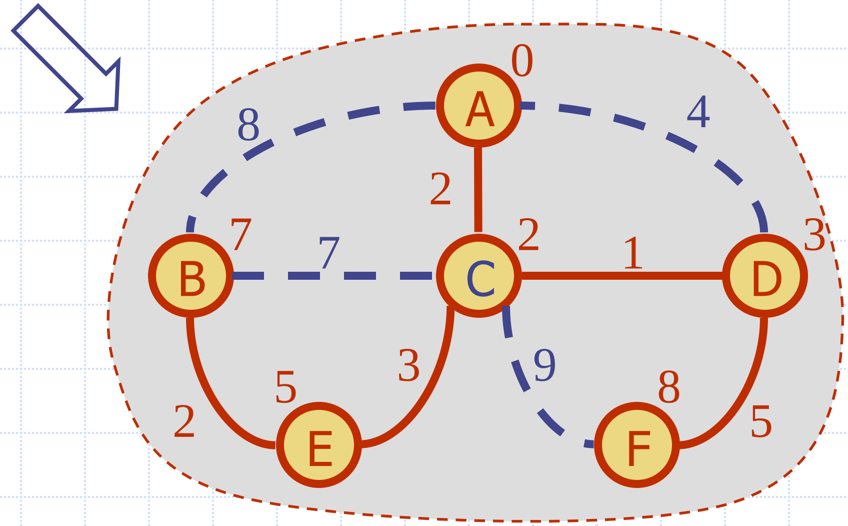
$D[v] \leftarrow 0$
for each vertex $u \neq v$ of G **do**
    $D[u] \leftarrow +\infty$
Let a priority queue, Q, contain all the vertices of G using the D labels as keys.
while Q is not empty **do**
    // pull a new vertex u into the cloud
    $u \leftarrow Q$.removeMin()
    for each vertex z adjacent to u such that z is in Q **do**
        // perform the ***relaxation*** procedure on edge (u, z)
        if $D[u] + w((u, z)) < D[z]$ **then**
            $D[z] \leftarrow D[u] + w((u, z))$
            Change the key for vertex z in Q to $D[z]$
return the label $D[u]$ of each vertex u

Παράδειγμα



Παράδειγμα (συνέχεια)



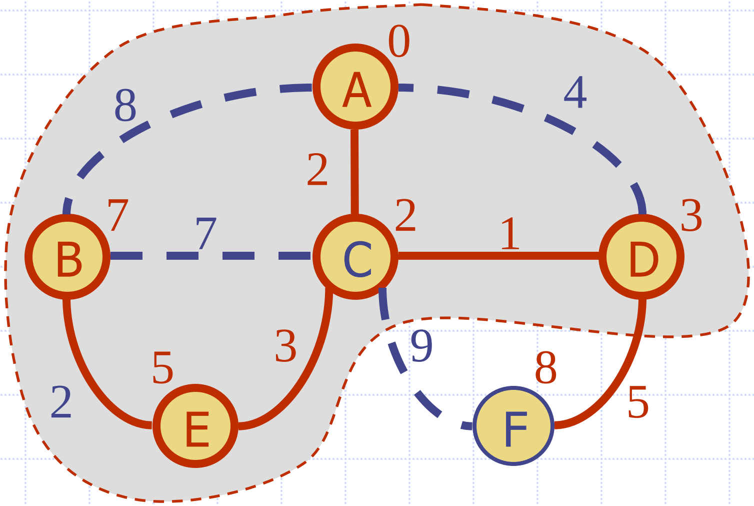


Ανάλυση αλγόριθμου του Dijkstra

- Λειτουργίες που εκτελούνται στο γράφο G με $\boldsymbol{n}$ κορυφές και $\boldsymbol{m}$ ακμές
  - Βρίσκουμε όλες τις ακμές κάθε κορυφής, μια φορά για κάθε κορυφή
- Λειτουργίες που επιδρούν στις ετικέτες
  - Θέτουμε/ανακτούμε τις ετικέτες απόστασης της κορυφής z, $O(\deg(z))$ φορές
  - Η ανάθεση/ανάκτηση τιμής μιας ετικέτας απαιτεί χρόνο $O(1)$
- Λειτουργίες στην ουρά προτεραιότητας
  - Κάθε κορυφή εισάγεται μία φορά και αφαιρείται μία φορά από την ουρά προτεραιότητας, κάθε εισαγωγή ή αφαίρεση απαιτεί χρόνο $O(\log n)$
  - Το κλειδί μιας κορυφής στην ουρά προτεραιότητας τροποποιείται το πολύ $\deg(w)$ φορές, κάθε αλλαγή κλειδιού είναι χρόνου $O(\log n)$
- Ο αλγόριθμος του Dijkstra εκτελείται σε χρόνο $O((n + m) \log n)$ όταν ο γράφος αναπαρίσταται με δομή λίστας/πίνακα γειτνίασης
  - Θυμηθείτε ότι $\Sigma_v \deg(v) = 2m$
- Ο χρόνος εκτέλεσης μπορεί να εκφραστεί ως $O(m \log n)$ εφόσον ο γράφος είναι συνεκτικός

Γιατί λειτουργεί ο αλγόριθμος του Dijkstra;

- Ο αλγόριθμος του Dijkstra βασίζεται στην άπληστη μέθοδο. Προσθέτει κορυφές σε αύξουσα σειρά απόστασης
  - Ας υποθέσουμε ότι δεν εντοπίζει όλες τις συντομότερες αποστάσεις. Έστω w η πρώτη κορυφή για την οποία ο αλγόριθμος επιστρέφει λάθος τιμή.
  - Όταν εξετάστηκε η προηγούμενη κορυφή u, η συντομότερη διαδρομή υπολογίστηκε ορθά μιας και η πρώτη λανθασμένη τιμή αφορά τη w
  - Αλλά πραγματοποιήθηκε χαλάρωση της ακμής (u,w) εκείνη τη στιγμή!
  - Έτσι, όσο ισχύει D[w]≥D[u], η απόσταση της w δεν μπορεί να είναι λάθος. Οπότε δεν υπάρχει λάθος κορυφή.

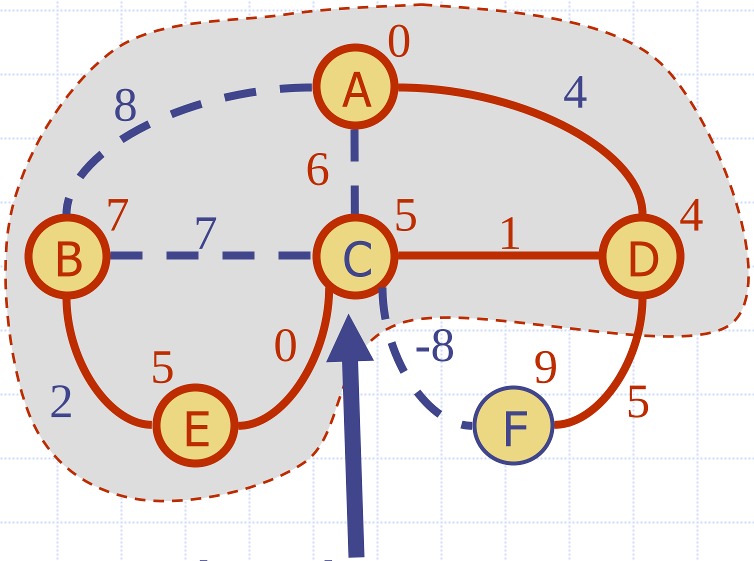


(u,w) = (D,F) σε αυτό το παράδειγμα

Γιατί δεν λειτουργεί για ακμές αρνητικού βάρους;

Ο αλγόριθμος του Dijkstra βασίζεται στην άπληστη μέθοδο. Προσθέτει κορυφές σε αύξουσα σειρά απόστασης.

- Εάν μία κορυφή με ακμή αρνητικού βάρους προστεθεί αργά στο σύννεφο, θα επηρεάσει τις αποστάσεις των κορυφών που υπάρχουν ήδη στο σύννεφο.



Η πραγματική απόσταση της C’ s είναι 1, αλλά έχει εισαχθεί ήδη στο σύννεφο με D[C]=5!

Ο αλγόριθμος των Bellman-Ford

- Λειτουργεί ακόμη και με ακμές αρνητικού βάρους
- Πρέπει να υποθέσουμε ότι έχουμε κατευθυνόμενες ακμές (αλλιώς θα είχαμε κύκλους αρνητικού βάρους)
- Η i επανάληψη βρίσκει όλες τις συντομότερες διαδρομές που χρησιμοποιούν i ακμές.
- Χρόνος εκτέλεσης: O(nm).
- Μπορεί να επεκταθεί ώστε να ανιχνεύει, εφόσον υπάρχει, κύκλο αρνητικού βάρους
  - Πως;

Ο αλγόριθμος των Bellman-Ford

Algorithm BellmanFordShortestPaths($\vec{G}, v$):

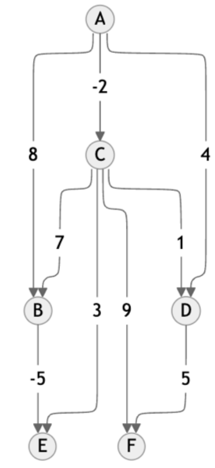
Input: A weighted directed graph $\vec{G}$ with n vertices, and a vertex v of $\vec{G}$

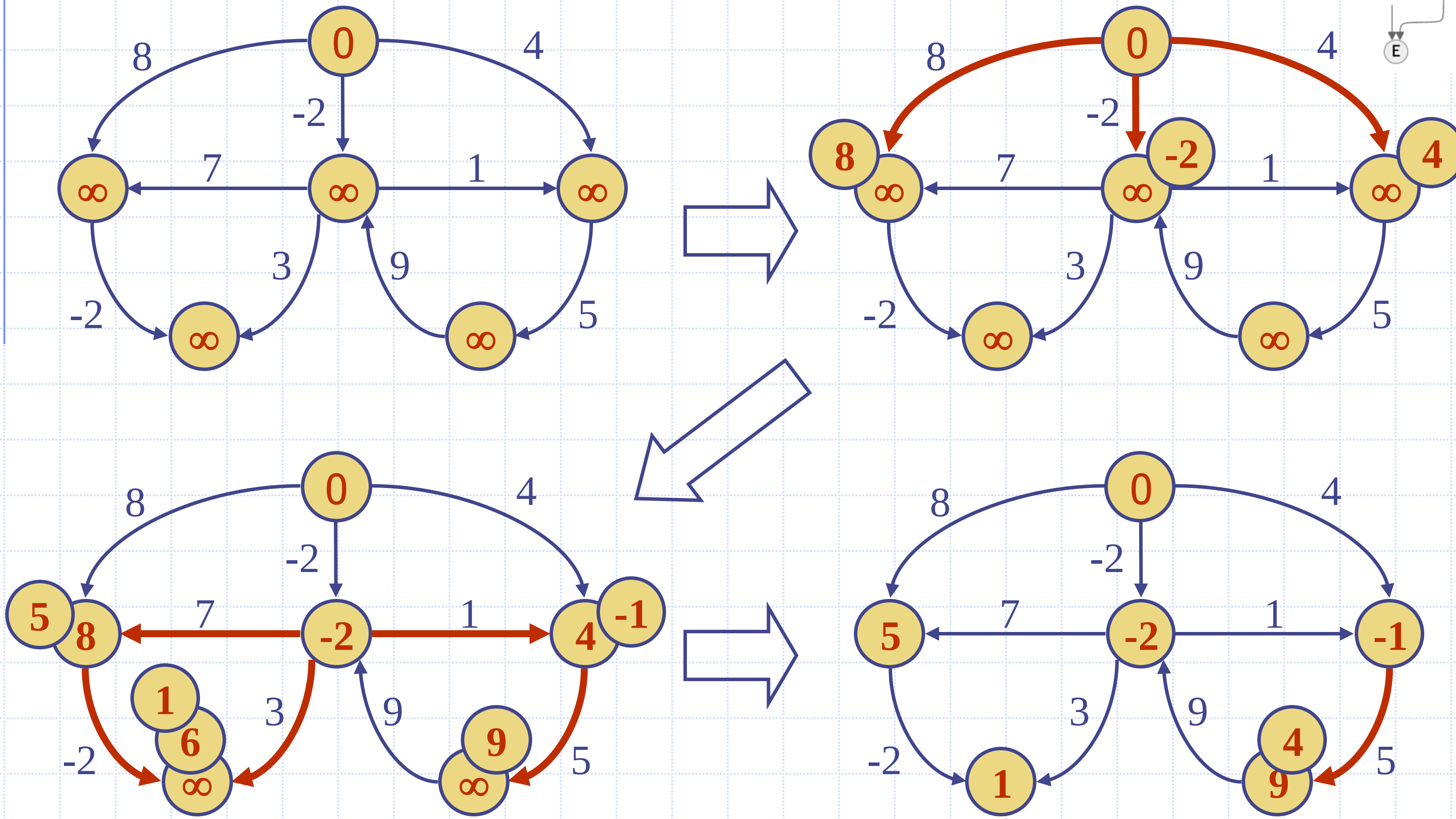
Output: A label $D[u]$, for each vertex u of $\vec{G}$, such that $D[u]$ is the distance from v to u in $\vec{G}$, or an indication that $\vec{G}$ has a negative-weight cycle

```
D[v] ← 0
for each vertex u ≠ v of G do
    D[u] ← +∞
for i ← 1 to n − 1 do
    for each (directed) edge (u, z) outgoing from u do
        // Perform the relaxation operation on (u, z)
        if D[u] + w((u, z)) < D[z] then
            D[z] ← D[u] + w((u, z))
if there are no edges left with potential relaxation operations then
    return the label D[u] of each vertex u
else
    return "G contains a negative-weight cycle"
```

Παράδειγμα Bellman-Ford

Οι κορυφές έχουν ετικέτες με τις D[v] τιμές τους





Αλγόριθμος συντομότερης διαδρομής για DAGs

- Μπορούμε να χρησιμοποιήσουμε έναν ειδικό αλγόριθμο εύρεσης συντομότερης διαδρομής που εφαρμόζεται σε κατευθυνόμενους ακυκλικούς γράφους (directed acyclic graphs - DAGs)
- Λειτουργεί ακόμη και με ακμές αρνητικού βάρους
- Χρησιμοποιεί τη λεγόμενη τοπολογική διάταξη (topological order)
- Δεν χρησιμοποιεί ιδιαίτερα σύνθετες δομές δεδομένων
- Είναι πολύ γρηγορότερος από τον αλγόριθμο του Dijkstra
- Χρόνος εκτέλεσης: O(n+m)

Αλγόριθμος συντομότερων διαδρομών για DAGs

Algorithm DAGShortestPaths($\vec{G}, s$):

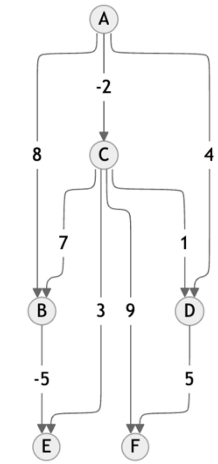
Input: A weighted directed acyclic graph (DAG) $\vec{G}$ with n vertices and m edges, and a distinguished vertex s in $\vec{G}$

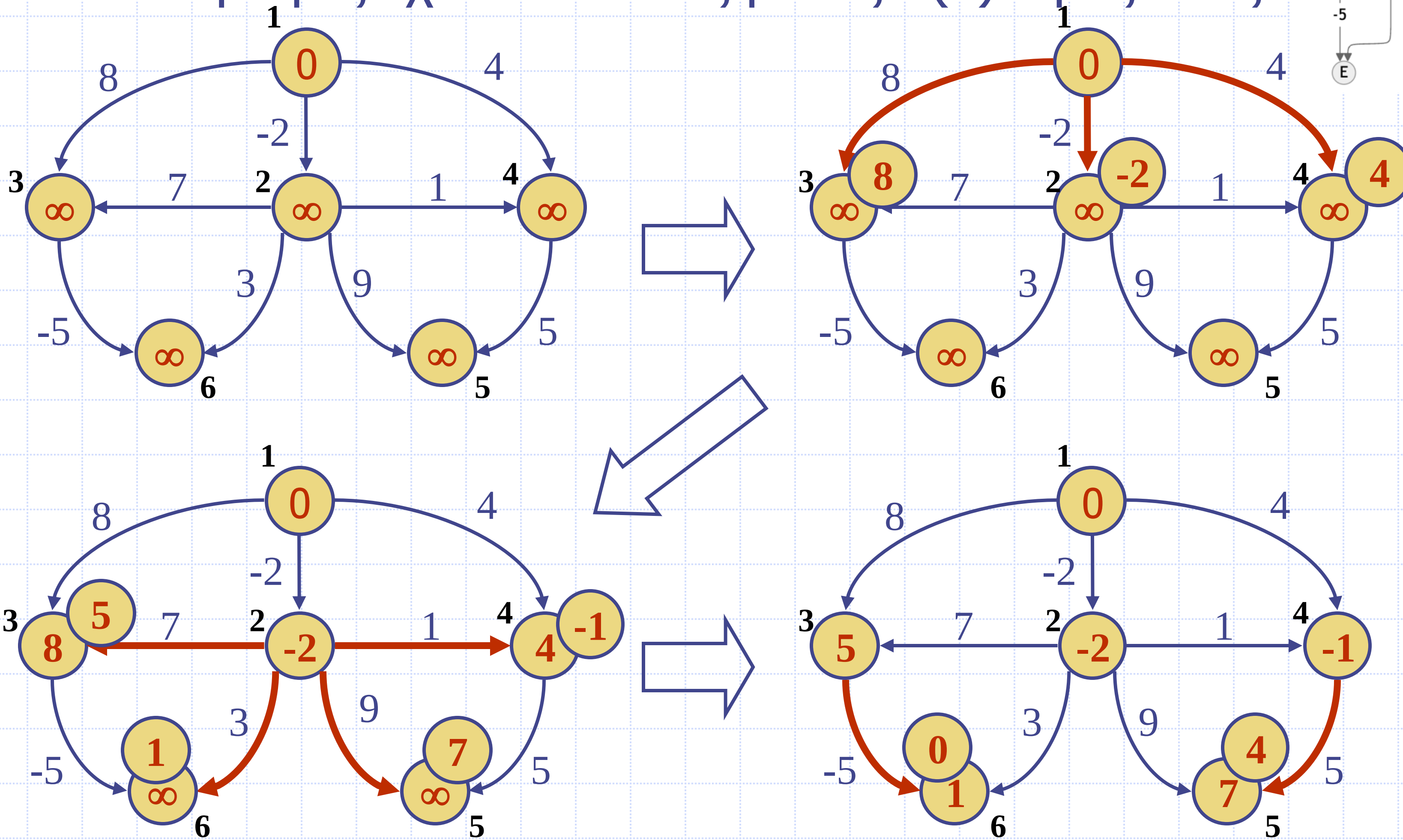
Output: A label $D[u]$, for each vertex u of $\vec{G}$, such that $D[u]$ is the distance from v to u in $\vec{G}$

Compute a topological ordering $(v_1, v_2, \ldots, v_n)$ for $\vec{G}$
$D[s] \leftarrow 0$
for each vertex $u \neq s$ of $\vec{G}$ **do**
  $D[u] \leftarrow +\infty$
for $i \leftarrow 1$ to $n - 1$ **do**
  // Relax each outgoing edge from v_i
  for each edge (v_i, u) outgoing from v_i **do**
    if $D[v_i] + w((v_i, u)) < D[u]$ **then**
      $D[u] \leftarrow D[v_i] + w((v_i, u))$
Output the distance labels D as the distances from s.

Παράδειγμα DAG

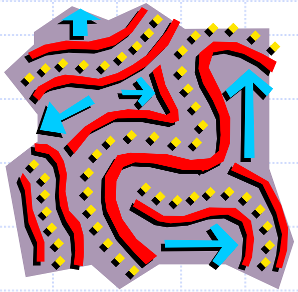
Οι κορυφές έχουν ετικέτες με τις d(v) τιμές τους





(two steps)

Συντομότερες διαδρομές όλων των ζευγών



- Εύρεση της απόστασης μεταξύ κάθε ζεύγους κορυφών σε έναν σταθμισμένο κατευθυνόμενο γράφο G.
- Μπορούμε να κάνουμε **n** κλήσεις του αλγορίθμου του Dijkstra (εάν δεν υπάρχουν αρνητικές ακμές), σε χρόνο O(nm log n).
- Αντίστοιχα, **n** κλήσεις στον αλγόριθμο Bellman-Ford σε χρόνο $O(n^2m)$.
- Μπορούμε να επιτύχουμε χρόνο $O(n^3)$ χρησιμοποιώντας δυναμικό προγραμματισμό (παρόμοιο με τον αλγόριθμο **Floyd-Warshall**).

```
Algorithm AllPair(G) {assumes vertices 1,…,n}
 for all vertex pairs (i,j)
   if i = j
      D_0[i,i] ← 0
   else if (i,j) is an edge in G
      D_0[i,j] ← weight of edge (i,j)
   else
      D_0[i,j] ← + ∞
 for k ← 1 to n do
   for i ← 1 to n do
     for j ← 1 to n do
       D_k[i,j] ← min{D_{k-1}[i,j], D_{k-1}[i,k]+D_{k-1}[k,j]}
 return D_n
```