Ελάχιστα Επικαλύπτοντα Δένδρα (Minimum Spanning Trees)

Θα εξετάσουμε απροσανατόλιστα γραφήματα με βάρη.

Υπογράφημα Επικάλυψης (Spanning Subgraph)

- Υπογράφημα ενός γραφήματος **G** που περιέχει **όλες τις κορυφές** του **G**.

Δένδρο Επικάλυψης (Spanning Tree)

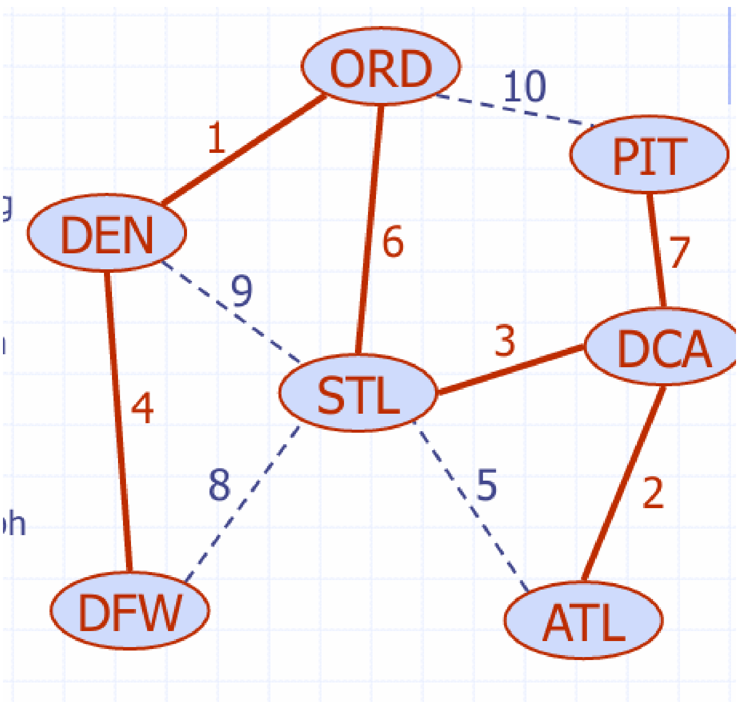
- Υπογράφημα επικάλυψης που είναι και **(ελεύθερο) δένδρο**.

Ελάχιστο Δένδρο Επικάλυψης (Minimum Spanning Tree, MST)

- Δένδρο επικάλυψης ενός **γραφήματος με βάρη**, του οποίου το **συνολικό βάρος των ακμών είναι το ελάχιστο δυνατό**.

❑ Εφαρμογές

- Δίκτυα επικοινωνιών
- Δίκτυα μεταφορών



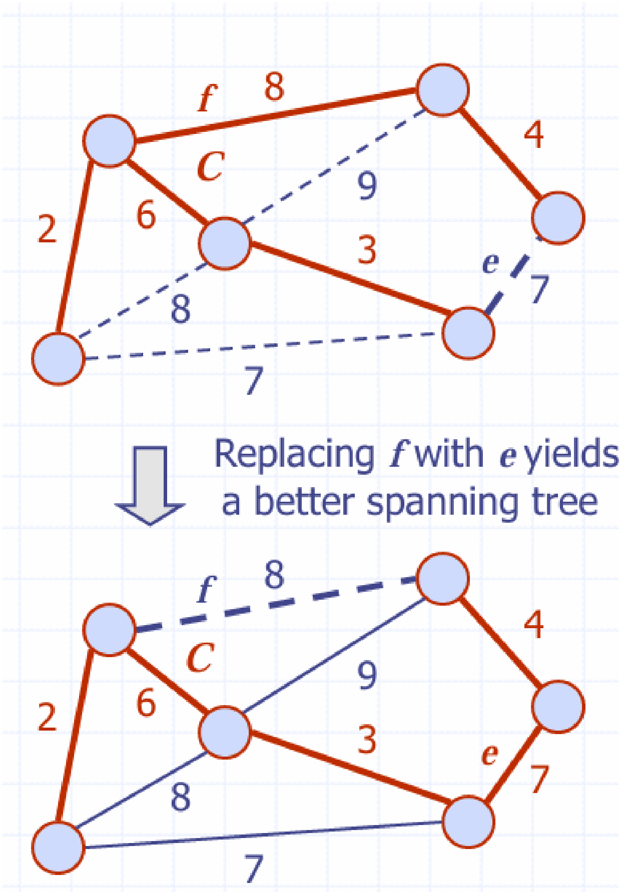
Ιδιότητα Κύκλου (Cycle Property):

- Έστω ότι **T** είναι ένα ελάχιστο επικαλύπτον δένδρο (Minimum Spanning Tree - MST) ενός γραφήματος **G** με βάρη.
- Έστω **e** μία ακμή του **G** που **δεν** ανήκει στο **T** και έστω **C** ο κύκλος που σχηματίζεται προσθέτοντας την **e** στο **T**.
- Για κάθε ακμή **f** του κύκλου **C**, ισχύει:

  βάρος(f) ≤ βάρος(e)

Απόδειξη:

- Με άτοπο (απόδειξη με υπόθεση του αντιθέτου).
- Αν **βάρος(f) > βάρος(e)**, τότε μπορούμε να πάρουμε ένα νέο επικαλύπτον δένδρο με μικρότερο συνολικό βάρος, αντικαθιστώντας την **f** με την **e**, κάτι που έρχεται σε αντίθεση με την υπόθεση ότι το **T** είναι ελάχιστο.



Ιδιότητα Διαμέρισης (Partition Property):

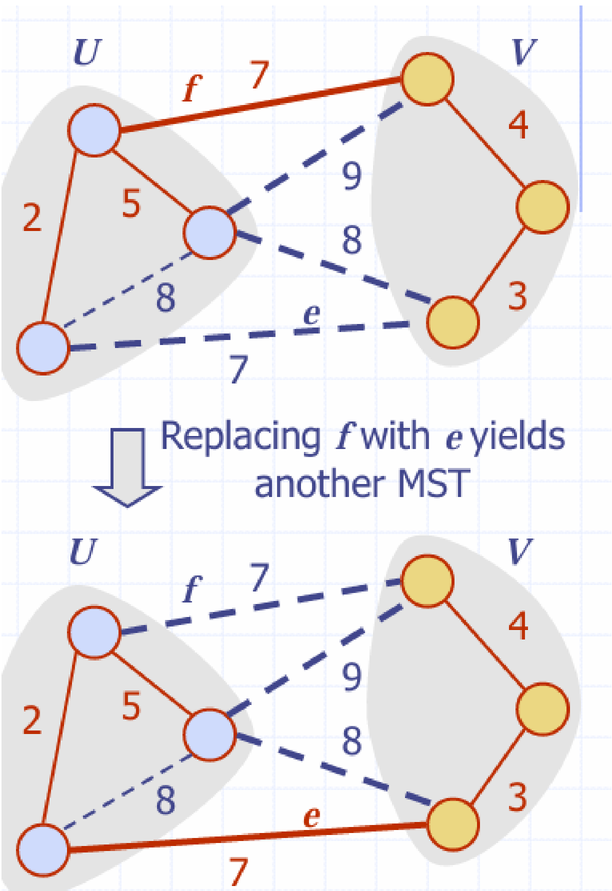
- Έστω ότι έχουμε μια διαμέριση των κορυφών του γραφήματος **G** σε δύο υποσύνολα **U** και **V**.
- Έστω **e** η ακμή ελάχιστου βάρους που συνδέει κορυφές από διαφορετικά σύνολα (δηλαδή που "διασχίζει" τη διαμέριση).
- Υπάρχει ένα ελάχιστο επικαλύπτον δένδρο (MST) του **G** που περιέχει την ακμή **e**.

Απόδειξη:

- Έστω **T** ένα ελάχιστο επικαλύπτον δένδρο του **G**.
- Αν το **T** δεν περιέχει την **e**, τότε προσθέτοντάς την στο **T** δημιουργείται ένας κύκλος **C**.
- Μέσα στον κύκλο **C**, έστω **f** μια άλλη ακμή που επίσης διασχίζει τη διαμέριση.
- Από την Ιδιότητα του Κύκλου (Cycle Property), ισχύει ότι:

  βάρος(f) ≤ βάρος(e)
- Άρα, αφού η **e** είναι η ακμή με **ελάχιστο βάρος** που διασχίζει τη διαμέριση, έχουμε:

  βάρος(f) = βάρος(e)
- Επομένως, μπορούμε να αντικαταστήσουμε την ακμή **f** με την **e** και να πάρουμε **ένα άλλο MST**, το οποίο περιέχει την **e**.



Αλγόριθμος Kruskal

- ❑ Διατηρούμε μια **διαμέριση** των κορυφών σε **συστάδες** (clusters)
  - ■ Αρχικά, κάθε κορυφή είναι μια ξεχωριστή συστάδα.
  - ■ Κρατάμε ένα MST για κάθε συστάδα.
  - ■ Συγχωνεύουμε τις «πιο κοντινές» συστάδες και τα MST τους.
- ❑ Χρησιμοποιούμε μία **ουρά προτεραιότητας** που περιέχει τις ακμές **εκτός των συστάδων**
  - ■ **Κλειδί**: το βάρος της ακμής
  - ■ **Τιμή**: η ακμή
- ❑ Στο τέλος του αλγορίθμου:
  - ■ Έχουμε **μία συστάδα** και **ένα MST**

scss

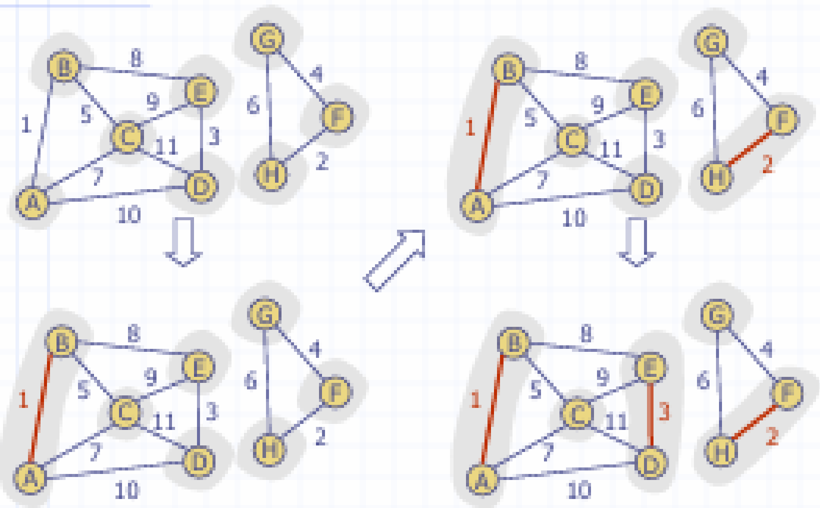
Αντιγραφή Επεξεργασία

```scss
Algorithm KruskalMST(G)
    foreach κορυφή v στο G do
        Δημιούργησε μία συστάδα που περιέχει μόνο τη v

    Q ← νέα ουρά προτεραιότητας
    Εισήγαγε όλες τις ακμές στην Q

    T ← ∅
    {Το T είναι η ένωση των MST των συστάδων}

    while T έχει λιγότερες από n - 1 ακμές do
        e ← Q.removeMin().getValue()
        [u, v] ← G.endVertices(e)
        A ← getCluster(u)
        B ← getCluster(v)

        if A ≠ B then
            Πρόσθεσε την ακμή e στο T
            mergeClusters(A, B)

    return T
```

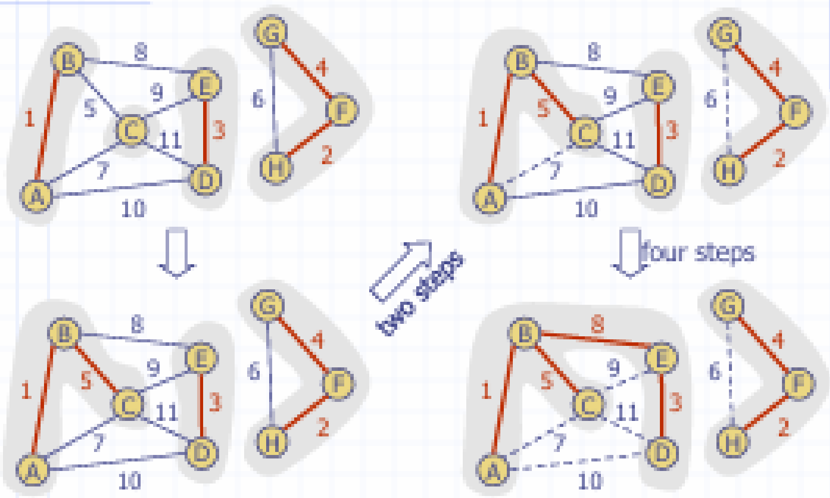
Σχόλια για την υλοποίηση:

- Η σύνταξη `αντικείμενο.συνάρτηση1.συνάρτηση2` είναι από αντικειμενοστραφή προγραμματισμό και σημαίνει ότι εφαρμόζουμε τη `συνάρτηση2` στο αποτέλεσμα της `συνάρτησης1`.
  Παράδειγμα: `Q.removeMin().getValue()` σημαίνει "πάρε την τιμή του στοιχείου με το ελάχιστο κλειδί από την ουρά προτεραιότητας".
- Η ακμή `e` που επιλέγεται στο `while` είναι αυτή με το **ελάχιστο βάρος** από όλες τις διαθέσιμες ακμές.
- Οι `u` και `v` είναι τα άκρα της ακμής `e`.
- Η **Ιδιότητα της Διαμέρισης (Partition Property)** εγγυάται ότι κάθε φορά που προσθέτουμε μία τέτοια ακμή, αυτή **ανήκει σε κάποιο MST** του γράφου.

Example



Example (cont'd)



Παράδειγμα (συνέχεια)

❑ Οι **κόκκινες ακμές** είναι οι ακμές του MST.
❑ Οι **γαλάζιες διακεκομμένες ακμές** είναι ακμές που εξετάστηκαν από τον αλγόριθμο αλλά απορρίφθηκαν επειδή τα άκρα τους βρίσκονταν ήδη στο ίδιο σύνολο.

Δομές Δεδομένων για τον Αλγόριθμο Kruskal

❑ Ο γράφος υλοποιείται με **λίστες γειτνίασης**.
❑ Ο αλγόριθμος διατηρεί ένα **δάσος δέντρων**.
❑ Μία **ουρά προτεραιότητας** εξάγει τις ακμές με αύξον βάρος (υλοποιείται ως ελάχιστος σωρός – *min heap*).
❑ Μια ακμή γίνεται αποδεκτή μόνο αν ενώνει **ξεχωριστά δέντρα**.

Χρειαζόμαστε μια δομή που υλοποιεί **διαμέριση** (σύνολα μη επικαλυπτόμενα), με λειτουργίες:

- `makeSet(u)`: δημιουργεί ένα σύνολο που περιέχει μόνο το u
- `findSet(u)`: επιστρέφει το σύνολο που περιέχει το u
- `union(A, B)`: συγχωνεύει τα σύνολα A και B

Δομές Δεδομένων για τον Kruskal (συνέχεια)

❑ Για την υλοποίηση των διακριτών συνόλων, θα χρησιμοποιήσουμε δέντρα διαμέρισης (disjoint forests) με τον αλγόριθμο weighted quick-union και συμπίεση διαδρομής με διπλασιασμό (path compression by halving).

Υλοποίηση με βάση τη Διαμέριση

Αλγόριθμος KruskalMST(G)

```less
Αρχικοποίησε μια διαμέριση P
Για κάθε κορυφή v του G κάνε:
    P.makeSet(v)

Q ← ουρά προτεραιότητας
Εισήγαγε όλες τις ακμές στην Q

T ← Ø  // T είναι η ένωση των MST των συστάδων

Όσο το T έχει λιγότερες από n-1 ακμές κάνε:
    e ← Q.removeMin().getValue()
    [u, v] ← G.endVertices(e)
    A ← P.findSet(u)
    B ← P.findSet(v)

    Αν A ≠ B τότε:
        Πρόσθεσε την ακμή e στο T
        P.union(A, B)

Επιστροφή T
```

Ανάλυση Πολυπλοκότητας

❑ Έστω *n* οι κορυφές και *m* οι ακμές του γράφου.

- Η ουρά προτεραιότητας μπορεί να αρχικοποιηθεί:
  - Σε O(m log m) με επαναληπτικές εισαγωγές
  - Ή σε O(m) με το bottom-up κατασκευάσμα σωρού (όπως παρουσιάστηκε στο μάθημα)
- Οι απαλοιφές min από την ουρά προτεραιότητας: O(m log m)
- Εναλλακτική: Ταξινόμηση των ακμών κατά βάρος → O(m log m) με mergesort ή heapsort.
- Πλήθος λειτουργιών:
  - **n - 1 ενώσεις (union)**
  - **έως m αναζητήσεις (findSet)**
    Αυτές οι πράξεις εκτελούνται σε **O(n log m)** με τη δομή forest + path compression.

✅ Τελική Χρονική Πολυπλοκότητα: **O((n + m) log n)**

Αλγόριθμος Prim-Jarnik

❑ Παρόμοιος με τον αλγόριθμο του Dijkstra.
❑ Επιλέγουμε μια τυχαία κορυφή `s` και επεκτείνουμε το MST ως ένα **σύννεφο κορυφών**, ξεκινώντας από την `s`.
❑ Για κάθε κορυφή `v`, αποθηκεύουμε μια ετικέτα `D(v)` που δηλώνει το **μικρότερο βάρος ακμής** που συνδέει την `v` με κάποια κορυφή στο σύννεφο.

Σε κάθε βήμα:

1. Προσθέτουμε στο σύννεφο την κορυφή `u` **με τη μικρότερη ετικέτα D(u)**.
2. Ενημερώνουμε τις ετικέτες `D(z)` για όλες τις κορυφές `z` που είναι γειτονικές της `u`.

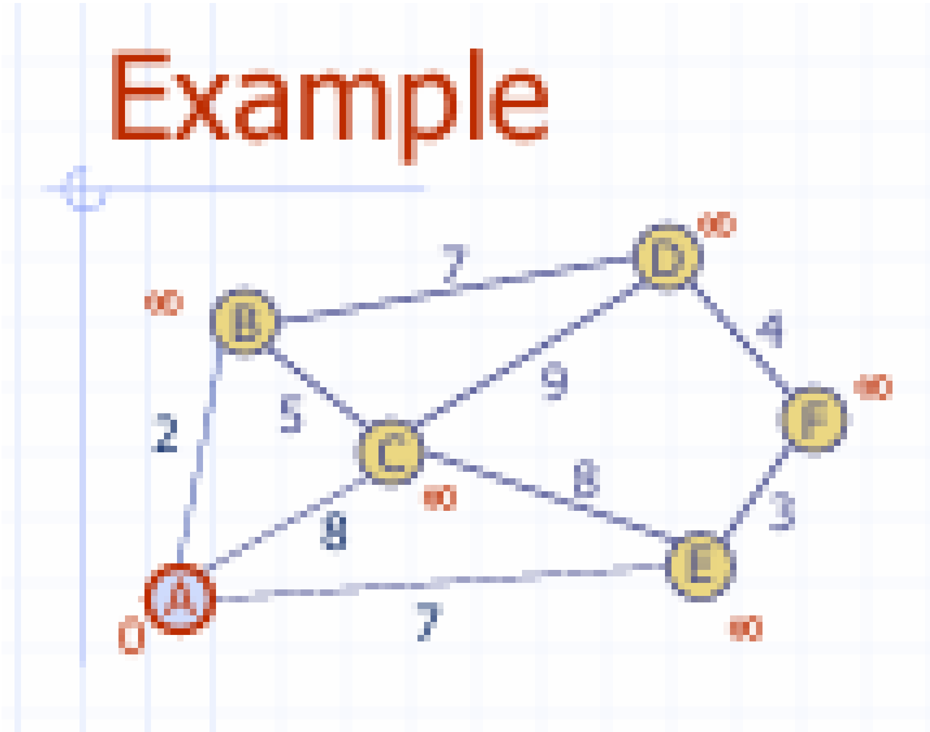
Υλοποίηση Αλγορίθμου Prim-Jarnik

- Ο γράφος αναπαρίσταται με λίστες γειτνίασης.
- Χρησιμοποιούμε **ουρά προτεραιότητας (min heap)**, όπου για κάθε κορυφή `v` αποθηκεύεται το ζεύγος `(v, e)` με:
  - `e` = η ακμή μικρότερου βάρους που συνδέει το `v` με το σύννεφο
  - `D(v)` = βάρος της ακμής `e` (το κλειδί στην ουρά)

Αλγόριθμος PrimJarnikMST(G)

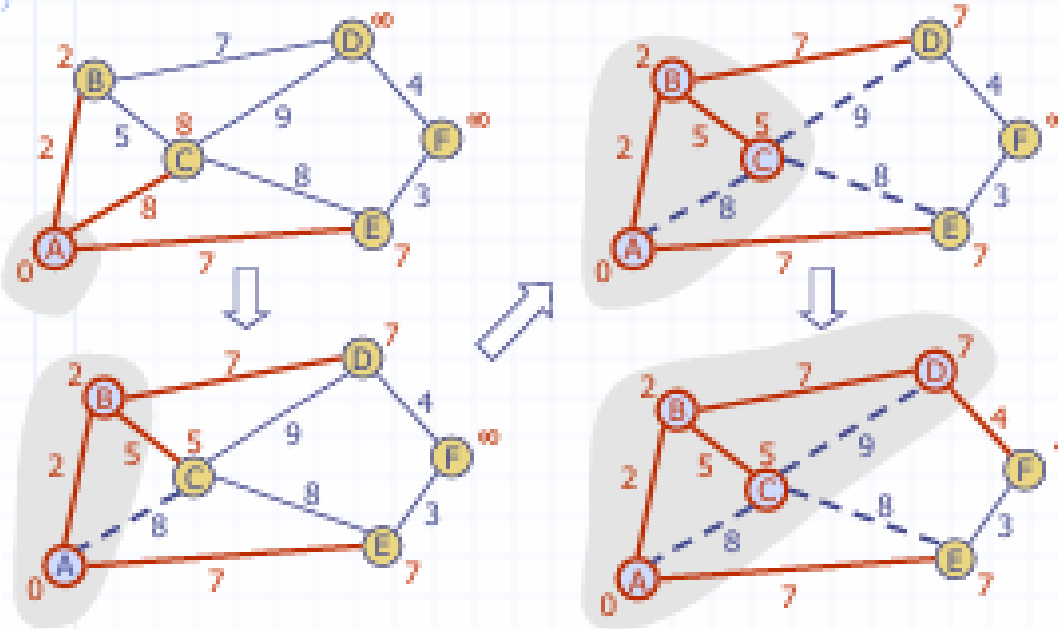
mathematica   Αντιγραφή   Επεξεργασία

```mathematica
Επίλεξε μια κορυφή v του G
D[v] ← 0
Για κάθε άλλη κορυφή u ≠ v κάνε:
    D[u] ← +∞

Αρχικοποίησε T ← ∅
Αρχικοποίησε ουρά προτεραιότητας Q με εγγραφές ((u, null), D[u]) για κάθε u

Όσο η Q δεν είναι κενή:
    (u, e) ← Q.removeMin()
    Πρόσθεσε την κορυφή u και την ακμή e στο T

    Για κάθε κορυφή z γειτονική της u και z ∈ Q:
        Αν weight((u, z)) < D[z] τότε:
            D[z] ← weight((u, z))
            Ενημέρωσε την τιμή της z σε (z, (u, z)) στην Q
            Ενημέρωσε το κλειδί της z σε D[z] στην Q

Επιστροφή T
```

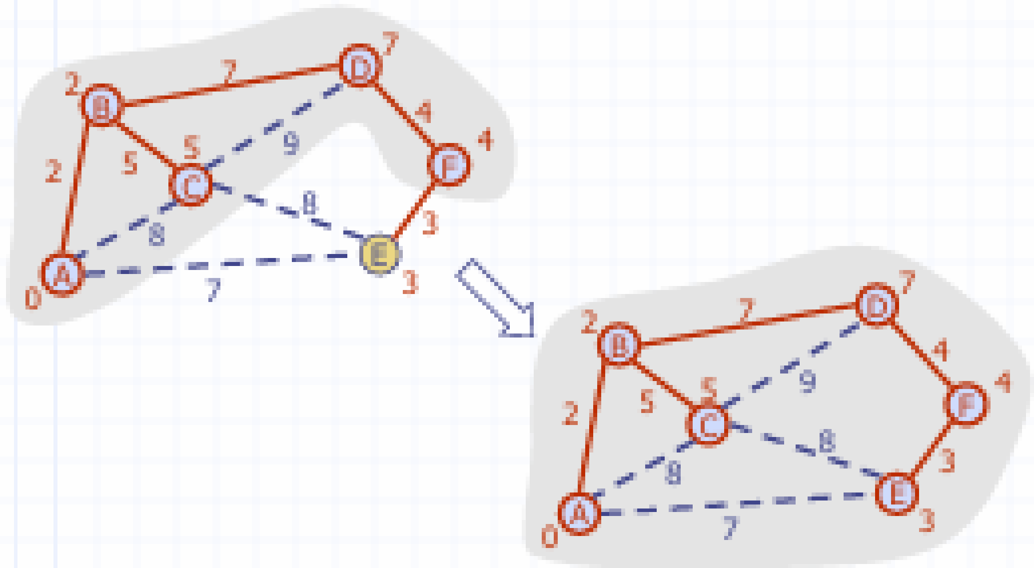


Αυτός είναι ο γράφος **πριν εκτελεστεί ο βρόχος** `while`. Οι τιμές **D[v]** εμφανίζονται με **κόκκινο χρώμα** κοντά στις κορυφές.
Στην **πρώτη επανάληψη** του βρόχου `while`, θα επιλεγεί η κορυφή **A**, και στη συνέχεια ο αλγόριθμος θα προχωρήσει όπως φαίνεται στη **επόμενη διαφάνεια**.

Example



Example (contd.)



Παράδειγμα (συνέχεια)

- Σε μια υλοποίηση, μπορούμε να χρησιμοποιήσουμε έναν **πολύ μεγάλο θετικό ακέραιο** στη θέση του ∞. Όλα τα βάρη θεωρούνται τότε μικρότερα από αυτόν τον ακέραιο.
- Στα προηγούμενα σχήματα, **οι κόκκινες ακμές** δηλώνουν ακμές του Ελάχιστου Επικαλύπτοντος Δένδρου (MST) όταν βρίσκονται μέσα στο «σύννεφο».
- Οι κόκκινες ακμές επίσης δηλώνουν τις ακμές **ελάχιστου βάρους** που συνδέουν κορυφές του MST (συννέφου) με κορυφές που βρίσκονται ακόμη στην ουρά Q.
- **Οι μπλε διακεκομμένες ακμές** δηλώνουν ακμές που απορρίφθηκαν, **μέσα στη συνθήκη if** του αλγορίθμου, υπέρ κόκκινων ακμών με μικρότερο βάρος.
- Με την ολοκλήρωση του αλγορίθμου, **οι κόκκινες ακμές σχηματίζουν το MST**.

Ανάλυση Πολυπλοκότητας

- Έστω **n** και **m** το πλήθος των κορυφών και ακμών αντίστοιχα του γράφου εισόδου.
- Ο βρόχος **for** εκτελείται σε χρόνο **O(n)**.
- Αφού η ουρά προτεραιότητας υλοποιείται ως **min-heap**, μπορεί να αρχικοποιηθεί σε χρόνο **O(n log n)** με επαναλαμβανόμενες εισαγωγές, ή σε **O(n)** χρησιμοποιώντας τον **bottom-up αλγόριθμο** που παρουσιάστηκε στις διαλέξεις για heaps.
- Μπορούμε να εξάγουμε την κορυφή **u** από την ουρά προτεραιότητας σε χρόνο **O(log n)**. Άρα, η πολυπλοκότητα για την εξαγωγή όλων των κορυφών είναι **O(n log n)**.

Ανάλυση Πολυπλοκότητας (συνέχεια)

- Μπορούμε να εκτελέσουμε τις **δύο εντολές αλλαγής** μέσα στη συνθήκη `if` σε χρόνο **O(log n)** (πώς μπορούμε να ενισχύσουμε την υλοποίηση της ουράς προτεραιότητας για να πετύχουμε αυτό το όριο;).
- Αυτή η ενημέρωση γίνεται **το πολύ μία φορά** για κάθε ακμή (u, z), επομένως το συνολικό πλήθος των ενημερώσεων είναι **O(m log n)**.
- Συνεπώς, ο αλγόριθμος **Prim-Jarnik** εκτελείται σε **χρόνο O((n + m) log n)**.