
**ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ**
**ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ**

# Δομές Δεδομένων

Ιωάννης Γ. Τόλλης
Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

# 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.

# 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.

.

# Minimum Spanning Tree

# Γενικά σημεία για μελέτη

- Minimum Spanning Trees (§7.3)
  - Ορισμοί
  - Ένα σημαντικό γεγονός
- Αλγόριθμος των Prim-Jarnik's (§7.3.2)
- Ο αλγόριθμος του Kruskal (§7.3.1)

# Minimum Spanning Tree

Spanning υπογράφος

- Ένας υπογράφος του $G$ που περιέχει όλους τους κόμβους του $G$

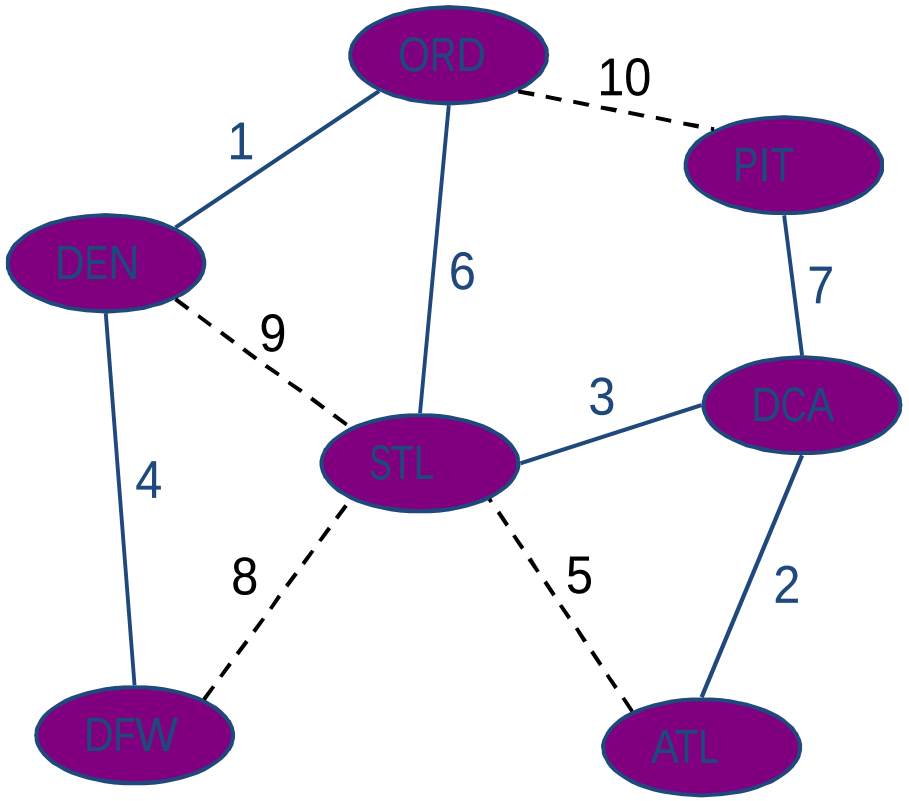
Spanning δέντρο

- Ένας Spanning υπογράφος που είναι από μόνος του (ελεύθερο ) δέντρο

Minimum spanning tree (MST)

- Το Spanning δέντρο ενός ζυγισμένου γράφου με το ελάχιστο άθροισμα βαρών των ακμών

- Εφαρμογές
  - Δίκτυα επικοινωνιών
  - Μεταφορικά δίκτυα



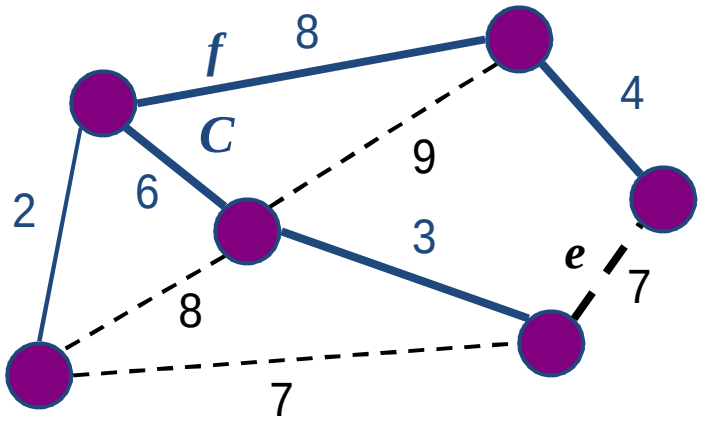
# Ιδιότητα Κύκλου

**Ιδιότητα Κύκλου:**

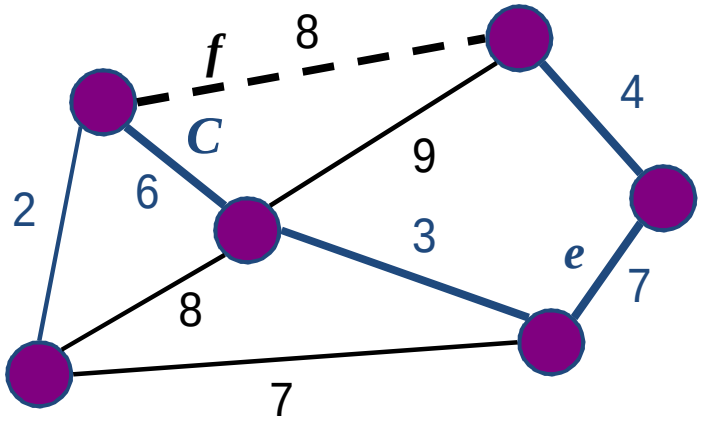
- Έστω $T$ το ελάχιστο spanning δέντρο ενός ζυγισμένου γράφου $G$
- Έστω $e$ μια ακμή του $G$ η οποία δεν ανήκει στο $T$ και $C$ είναι ο κύκλος που σχηματίζεται από την $e$ μαζί με το $T$
- Για κάθε ακμή $f$ του $C$, $weight(f) \leq weight(e)$

**Απόδειξη:**

- Εις άτοπον απαγωγή
- Αν $weight(f) > weight(e)$ μπορούμε να έχουμε ένα spanning δέντρο μικρότερου βάρους αντικαθιστώντας την $e$ με την $f$




Αντικαθιστώντας την $f$ με την $e$ δημιουργείται καλύτερο spanning δέντρο




Minimum Spanning Tree

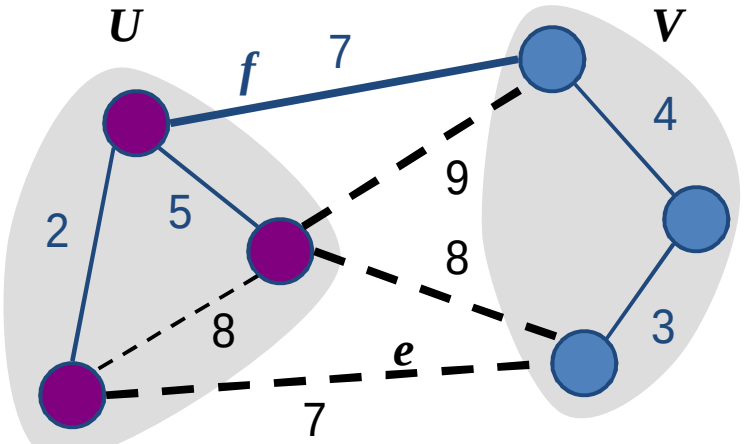
# Ιδιότητα τμηματοποίησης

**Ιδιότητα τμηματοποίησης:**

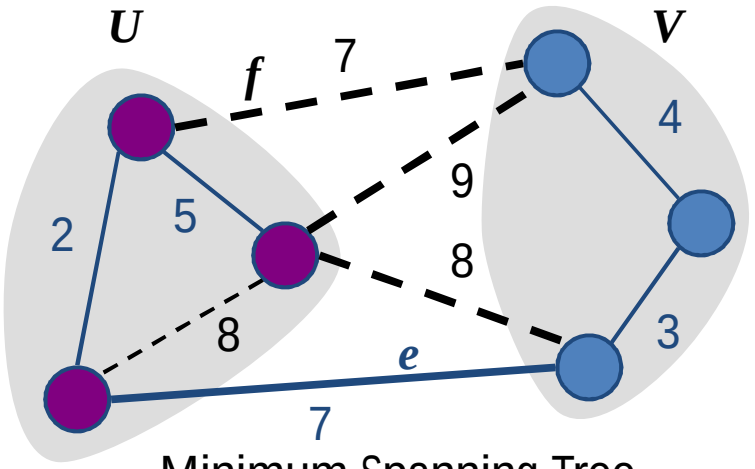
- Θεωρείστε ένα τμήμα των κόμβων του $G$ μέσα στα υποσύνολα $U$ και $V$
- Έστω $e$ είναι μια ακμή ελάχιστου βάρους κατά μήκος του τμήματος
- Υπάρχει ένα ελάχιστο spanning δέντρο του $G$ που περιέχει την ακμή $e$

**Απόδειξη:**

- Έστω $T$ ένα MST του $G$
- ΑΝ το $T$ δεν περιέχει την $e$, θεωρείστε τον κύκλο $C$ που σχηματίζεται από την $e$ με το $T$ και έστω $f$ μια ακμή του $C$ κατά μήκος του τμήματος
- Από την κυκλική ιδιότητα,
  $$weight(f) \leq weight(e)$$
- Επομένως, $weight(f) = weight(e)$
- Παίρνουμε άλλο MST αντικαθιστώντας την $f$ με την $e$




Αντικαθιστώντας την $f$ με την $e$ προκύπτει άλλο MST




Minimum Spanning Tree

# Ο αλγόριθμος των Prim-Jarnik's

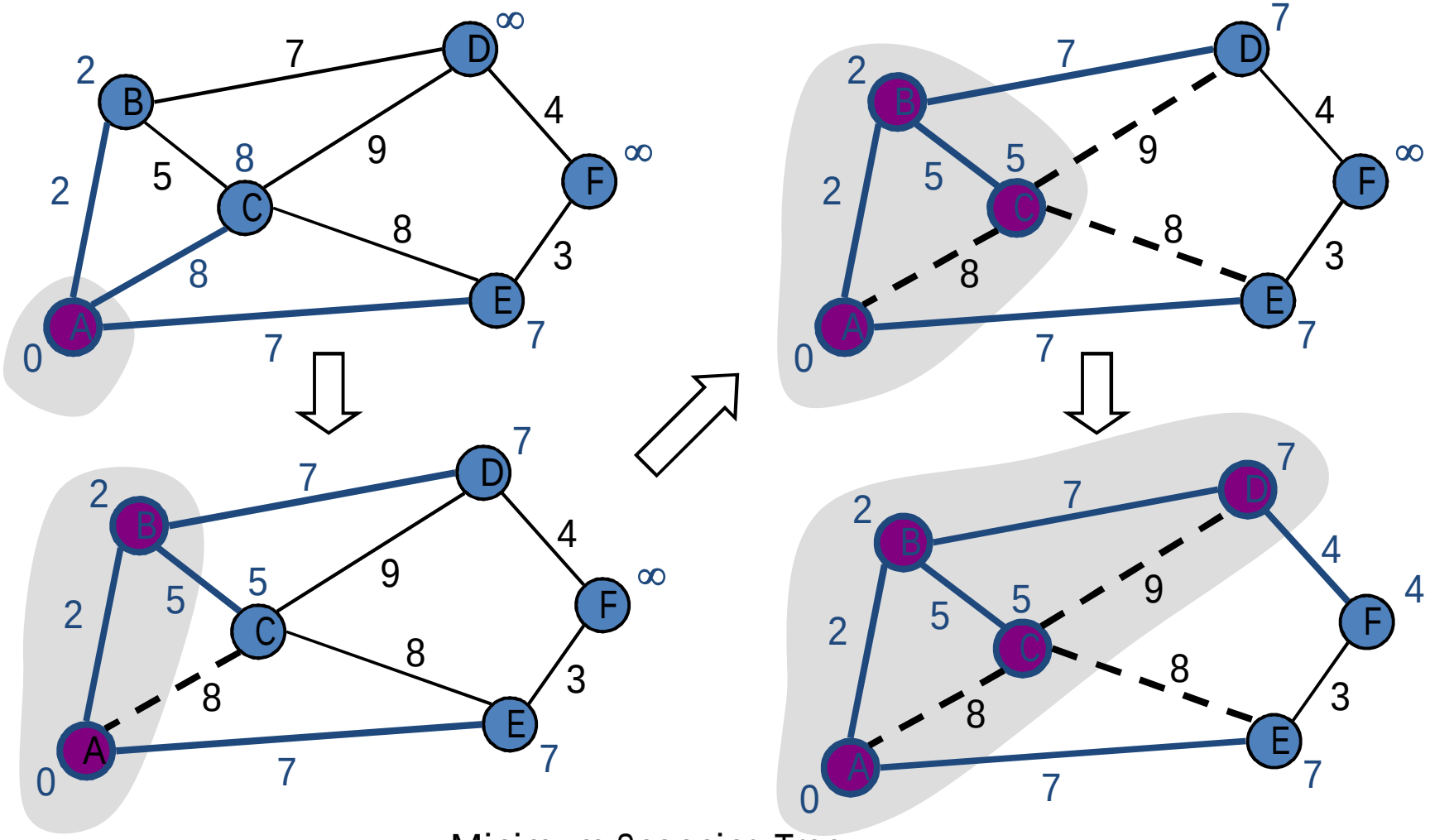
- Ο αλγόριθμος των Prim-Jarnik's για τον υπολογισμό ενός MST είναι παρόμοιος με τον αλγόριθμο του Dijkstra
- Θεωρούμε ότι ο γράφος είναι συνδεδεμένος
- Διαλέγουμε έναν τυχαίο κόμβο $s$ και δημιουργούμε το MST σαν ένα σύννεφο από κόμβους, αρχίζοντας από τον $s$
- Μαζί με κάθε κόμβο $v$ αποθηκεύουμε και μια ετικέτα $d(v)$ που αναπαριστά το μικρότερο βάρος μιας ακμής που συνδέει τον $v$ με έναν κόμβο στο σύννεφο
- Σε κάθε βήμα
  - Αποθηκεύουμε στο σύννεφο τον κόμβο $u$ με την μικρότερη τιμή της ετικέτας απόστασης
  - Ανανεώνουμε τις ετικέτες των κόμβων που είναι προσκείμενες στον $u$

# Ο αλγόριθμος των Prim-Jarnik (συν.)

- Μια ουρά προτεραιότητας αποθηκεύει τους κόμβους έξω από το σύννεφο
  - Key: απόσταση
  - Element: κόμβος
- Locator-based μέθοδοι
  - *insert*(*k*,*e*) επιστρέφει έναν locator
  - *replaceKey*(*l*,*k*) αλλάζει το κλειδί ενός αντικειμένου
- Αποθηκεύουμε τρεις ετικέτες με κάθε κόμβο:
  - Απόσταση
  - Την ακμή πατέρα σε ένα MST
  - Τον Locator σε μια ουρά προτεραιότητας

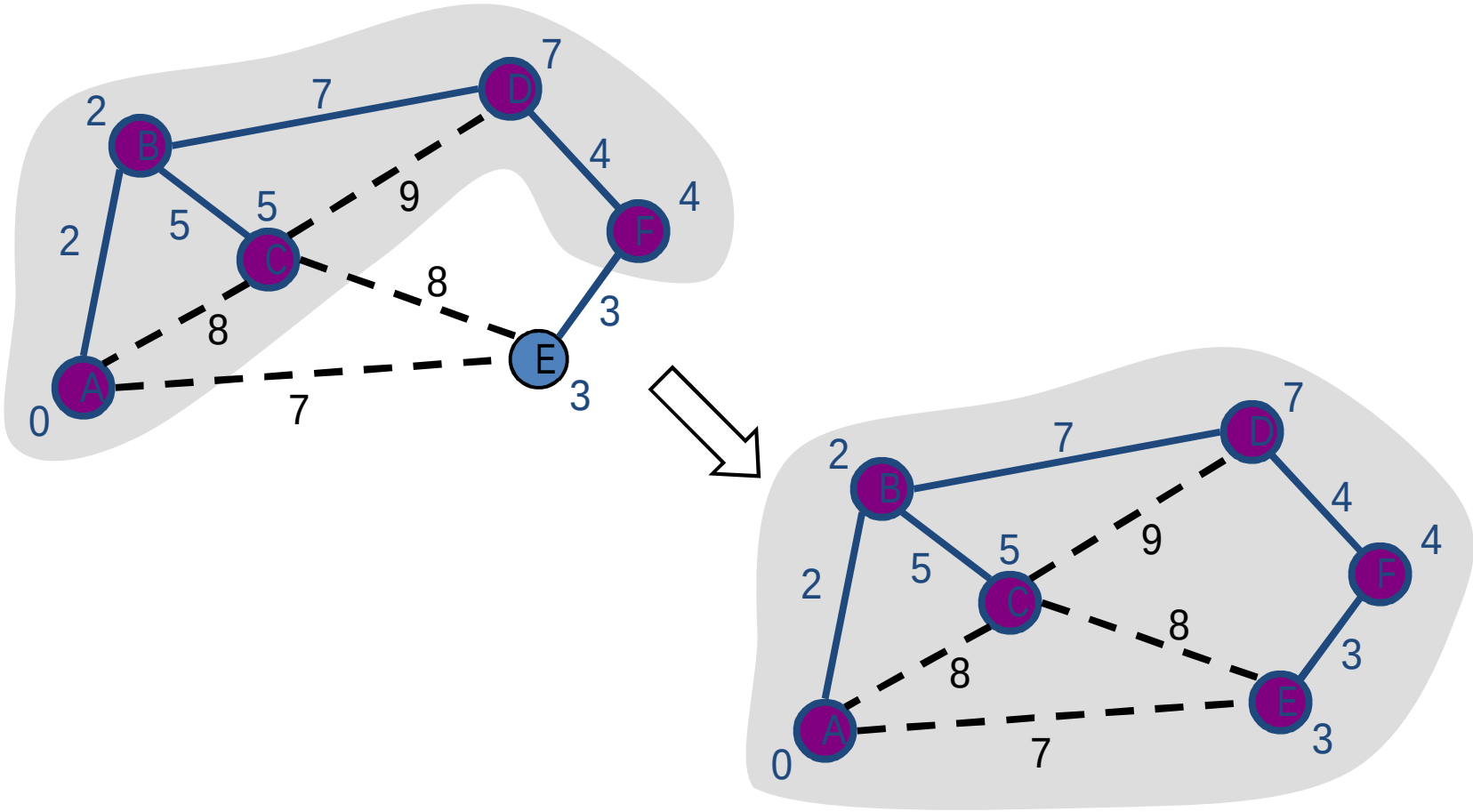
```
Algorithm PrimJarnikMST(G)
    Q ← new heap-based priority queue
    s ← a vertex of G
    for all v ∈ G.vertices()
        if v = s
            setDistance(v, 0)
        else
            setDistance(v, ∞)
        setParent(v, ∅)
        l ← Q.insert(getDistance(v), v)
        setLocator(v,l)
  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

# Παράδειγμα




Minimum Spanning Tree

# Παράδειγμα (συν.)



# Ανάλυση 1

- Λειτουργίες του γράφου
  - Η μέθοδος incidentEdges καλείται μια φορά για κάθε κόμβο
- Λειτουργίες ετικετών
  - Θέτουμε/ανακτούμε τις ετικέτες απόστασης,πατέρα και locator του κόμβου $z$ $O(\deg(z))$ φορές
  - Ανάθεση/ανάκτηση μιας ετικέτας παίρνει χρόνο $O(1)$
- Λειτουργίες ουρών προτεραιότητας
  - Κάθε κόμβος εισάγεται και απομακρύνεται μια φορά από την ουρά προτεραιότητας, όπου κάθε εισαγωγή και απομάκρυνση παίρνει χρόνο $O(\log n)$
  - Το κλειδί ενός κόμβου $w$ σε μια ουρά προτεραιότητας τροποποιείται το πολύ $\deg(w)$ φορές, όπου κάθε αλλαγή κλειδιού παίρνει χρόνο $O(\log n)$

# Ανάλυση 2

- Ο αλγόριθμος των Prim-Jarnik τρέχει σε χρόνο $O((n + m) \log n)$ δεδομένου ότι ο γράφος αναπαρίσταται με μια δομή adjacency λίστας
- Θυμηθείτε ότι $\Sigma_v \deg(v) = 2m$
- Ο χρόνος εκτέλεσης είναι $O(m \log n)$ δεδομένου ότι ο γράφος είναι συνδεδεμένος

# Dijkstra vs. Prim-Jarnik

```
Algorithm DijkstraShortestPaths(G, s)
    Q ← new heap-based priority queue

    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()
        u ← Q.removeMin()
        for all e ∈ G.incidentEdges(u)
            z ← G.opposite(u,e)
            r ← getDistance(u) + weight(e)
            if r < getDistance(z)
                setDistance(z,r)
                setParent(z,e)
                Q.replaceKey(getLocator(z),r)
```

```
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()
        u ← Q.removeMin()
        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(getLocator(z),r)
```

# Ο αλγόριθμος του Kruskal

- Μια ουρά προτεραιότητας αποθηκεύει τις ακμές έξω από το σύννεφο
  - Key: βάρος
  - Element: ακμή
- Στο τέλος του αλγόριθμου
  - Μένουμε με ένα σύννεφο που περιγράφει το MST
  - Ένα δέντρο *T* το οποίο είναι το MST

**Algorithm** ***KruskalMST(G)***
  **for** each vertex ***V*** in ***G*** **do**
    define a ***Cloud(v)*** of ← {***v***}
  **let** ***Q*** be a priority queue.
  Insert all edges into ***Q*** using their weights as the key
  ***T*** ← ∅
  **while** ***T*** has fewer than ***n***-1 edges **do**
    edge ***e*** = ***T.removeMin()***
    Let ***u, v*** be the endpoints of ***e***
    **if** ***Cloud(v)*** ≠ ***Cloud(u)*** **then**
      Add edge ***e*** to ***T***
      Merge ***Cloud(v)*** and ***Cloud(u)***
  **return** ***T***

# Τέλος Ενότητας