Αλγόριθμος Prim
Αλγόριθμος Kruskal

Αλγόριθμος Prim

- Ξεκινάμε από ένα δένδρο T αποτελούμενο από **ένα μόνο** κόμβο.

- Στη συνέχεια, σε κάθε επανάληψη, αυξάνουμε το δένδρο T συνδέοντάς το στο **πλησιέστερο ελεύθερο κόμβο** ως προς την έννοια του βάρους (κόστους).

Αλγόριθμος Prim

G=(V,E)

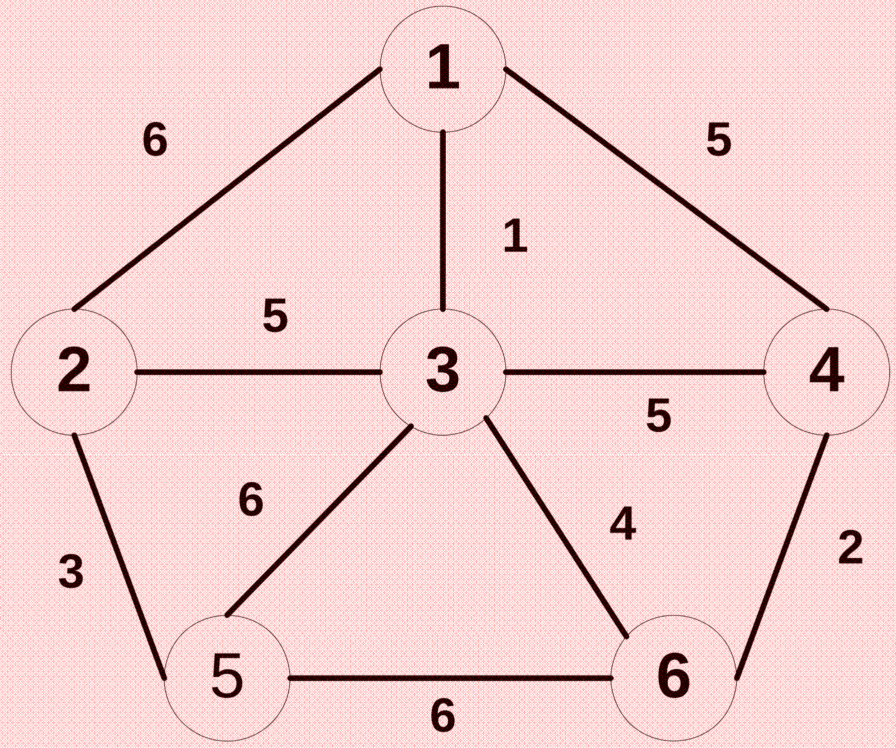
T={u, οποιοσδήποτε κόμβος}

Repeat

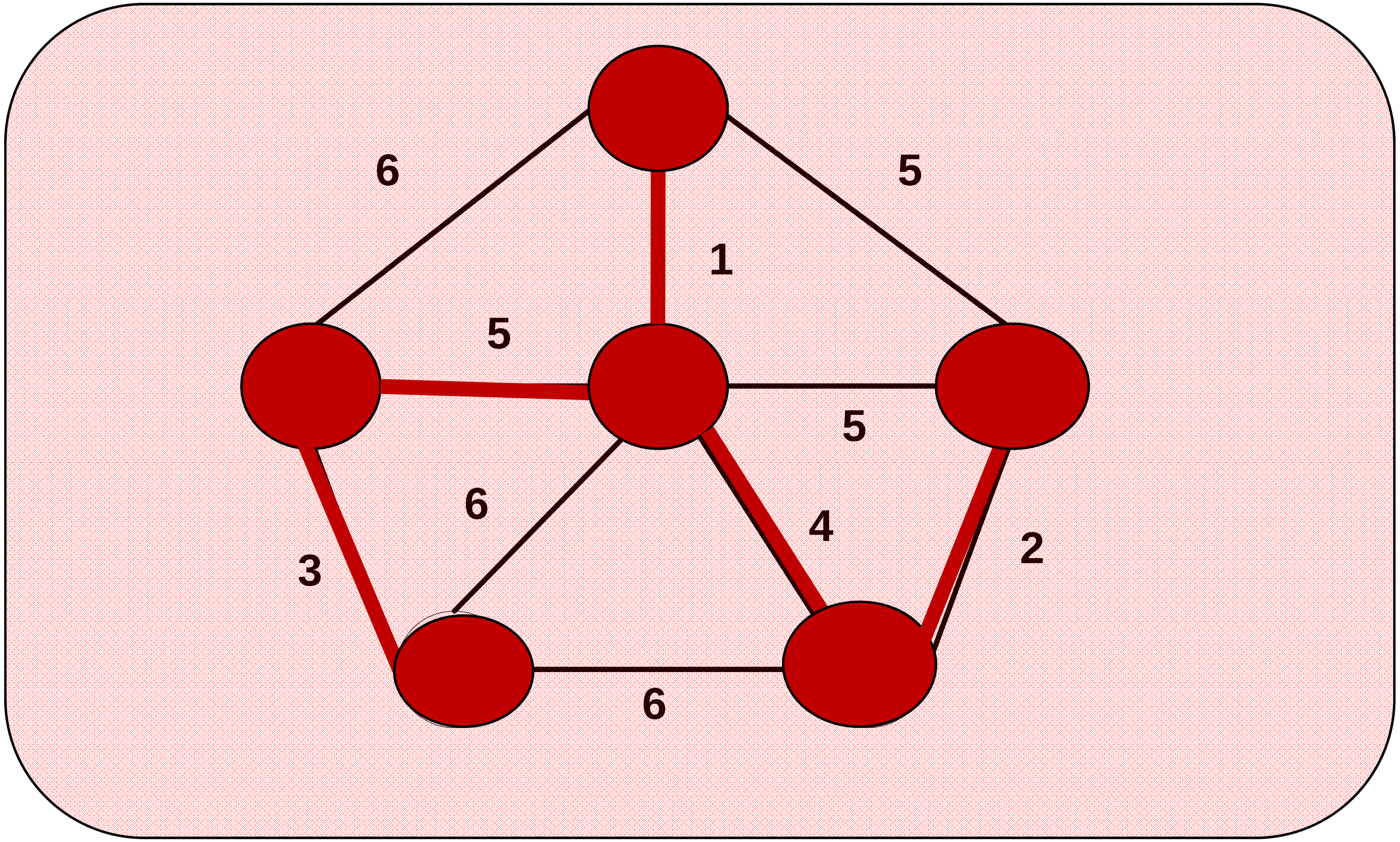
$T \leftarrow T \cup \{v\}$, v πλησιέστερος ελεύθερος κόμβος στο T

(****)

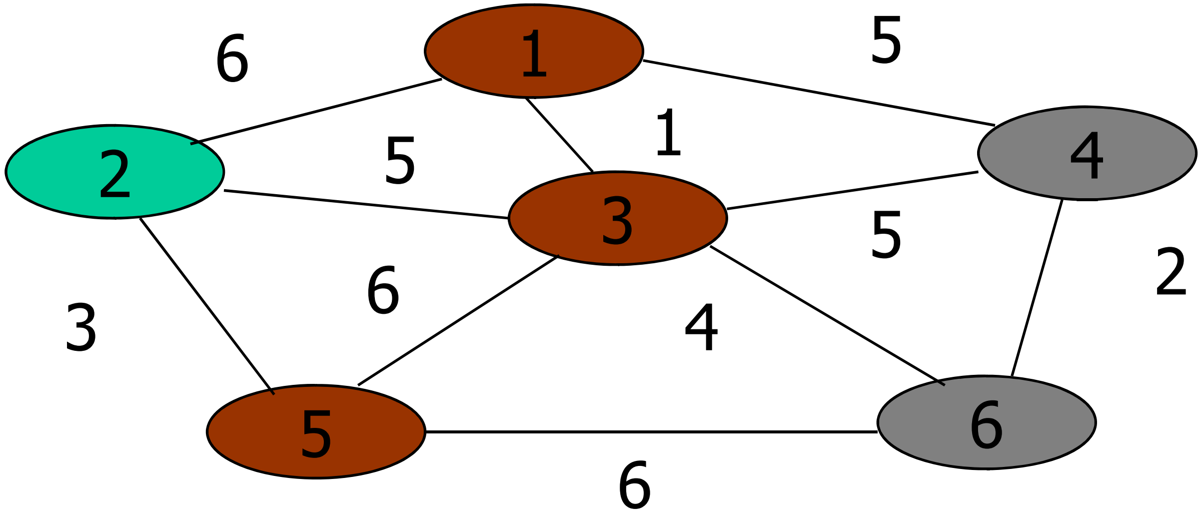
Παράδειγμα



Παράδειγμα



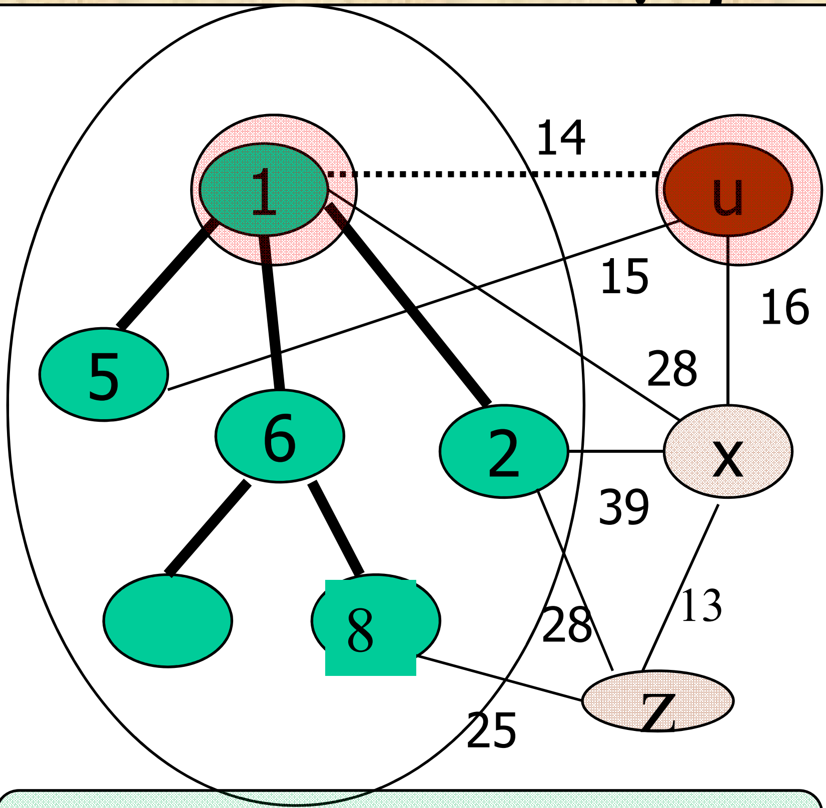
Αλγόριθμος Prim: Δομές



near[x]=
- x, αν x ανήκει στο T
- *0 αν [x,y] δεν ανήκει στο E, για όλα τα y του T*
- **y, αν [x,y] ανήκει** στο E, **y** ο πλησιέστερος κόμβος

dist[x]=
- **$W_{x,y}$ αν near[x]=y**
- *$+\infty$ αν near[x]=0*
- 0 αν near[x]=x

Αλγόριθμος Prim: Δομές

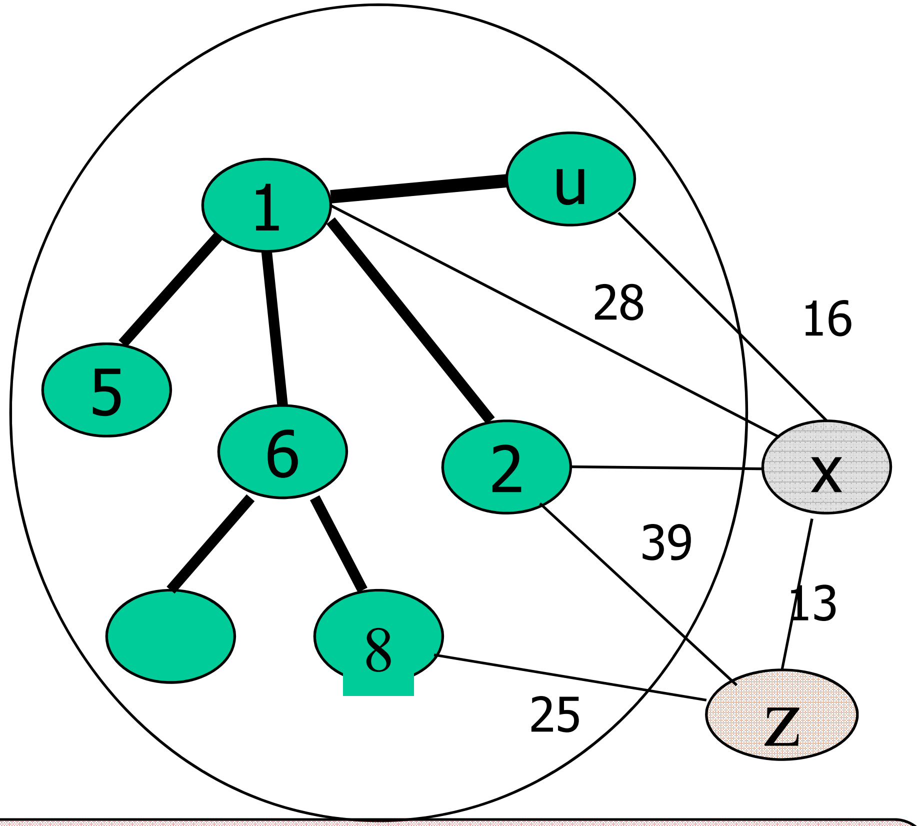


near[u]=1, dist[u]=14

near[x]=1, dist[x]=28

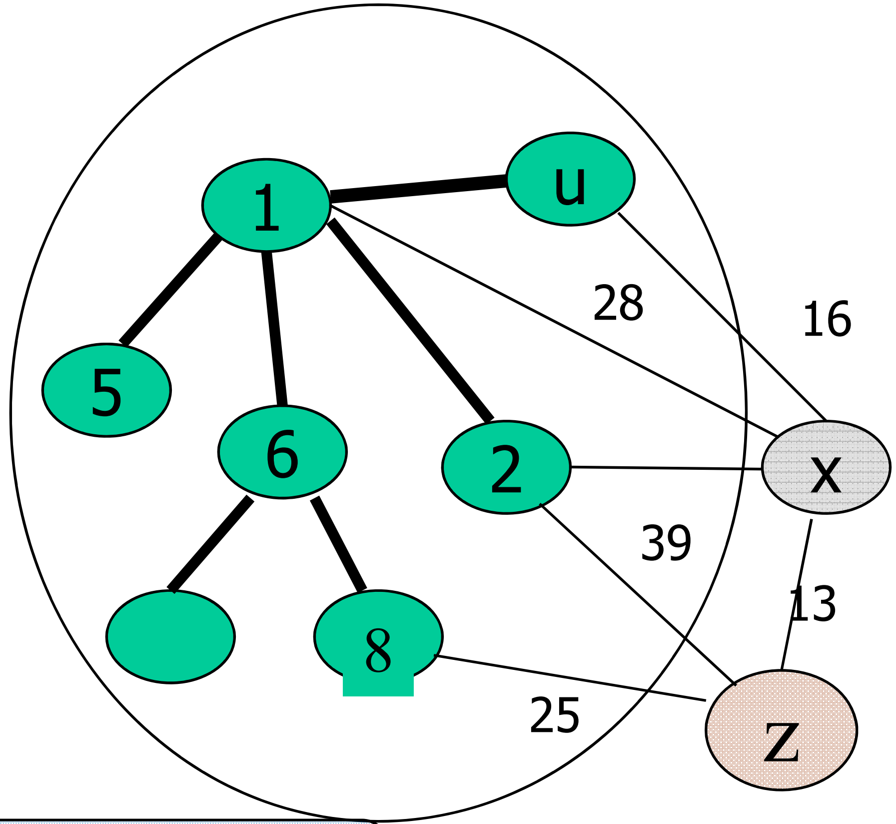
near[z]=8, dist[z]=25

Αλγόριθμος Prim: Δομές



near[u]=u, dist[u]=0

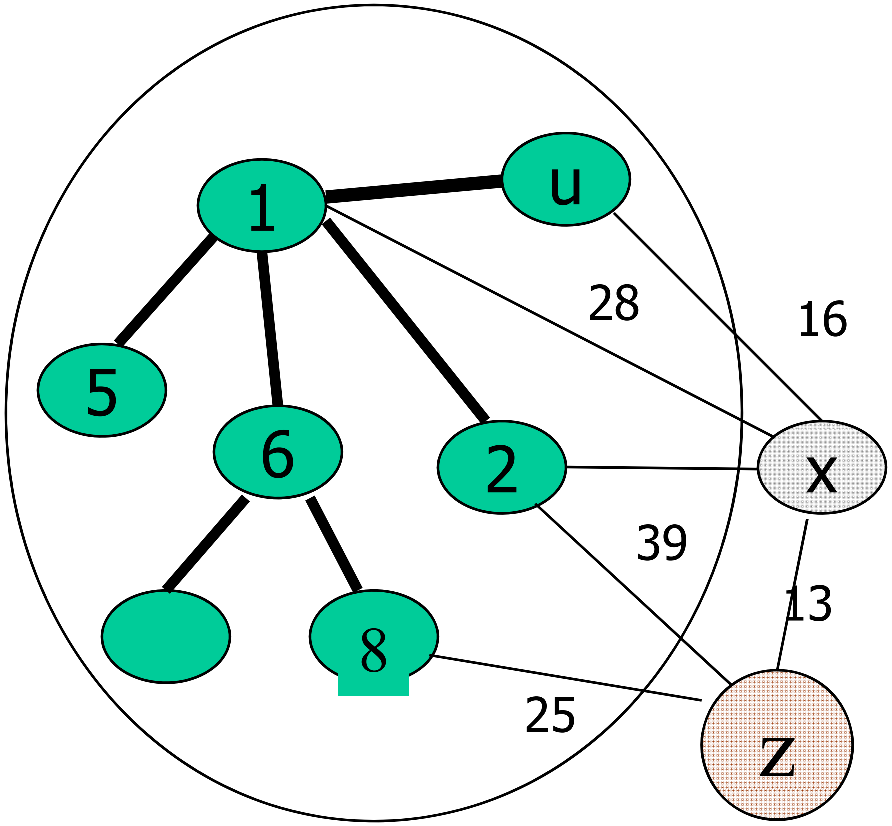
Αλγόριθμος Prim: Δομές



near[u]=u, dist[u]=0

near[x]=u, dist[x]=16 (ενημέρωση μόνο των γειτόνων του εισερχόμενου κόμβου)

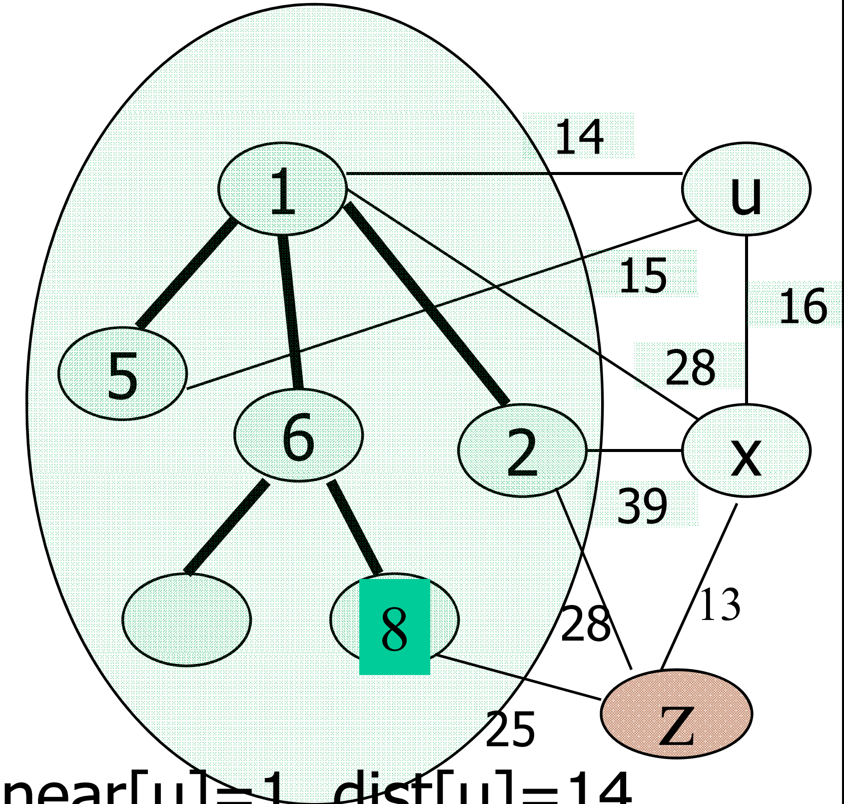
Αλγόριθμος Prim: Δομές



near[u]=u, dist[u]=0

near[x]=u, dist[x]=16 (ενημέρωση μόνο των γειτόνων του εισερχόμενου κόμβου)

Αλγόριθμος Prim: Δομές



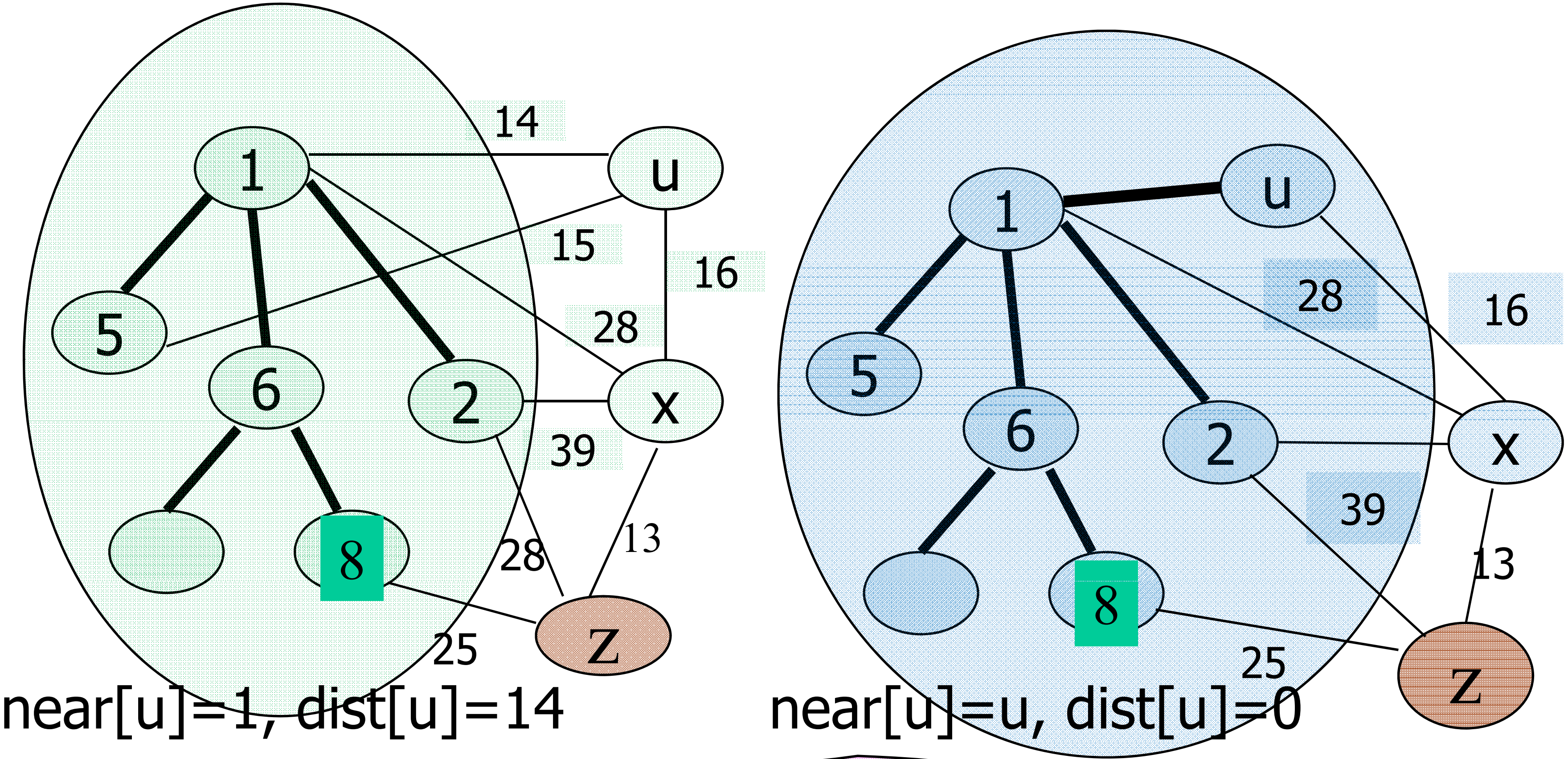
near[u]=1, dist[u]=14

near[x]=1, dist[x]=28

near[z]=8, dist[z]=25

Οι κόμβοι που δεν είναι γείτονες του εισερχόμενου κόμβου δεν επηρεάζονται

Αλγόριθμος Prim: Δομές



near[u]=1, dist[u]=14 near[u]=u, dist[u]=0

near[x]=1, dist[x]=28 near[x]=u, dist[x]=16

near[z]=8, dist[z]=25 **near[z]=8, dist[z]=25**

Οι κόμβοι που δεν είναι γείτονες του εισερχόμενου κόμβου δεν επηρεάζονται

Αλγόριθμος Prim (nearest neighbour)

initialization;

choose s arbitrary; near[s] ← s; dist(s)=0;

For i in Γ(s) do

near(i) ← s;

dist(i) ← $w_{s,i}$;

for every node i other than s and not in Γ(s) do

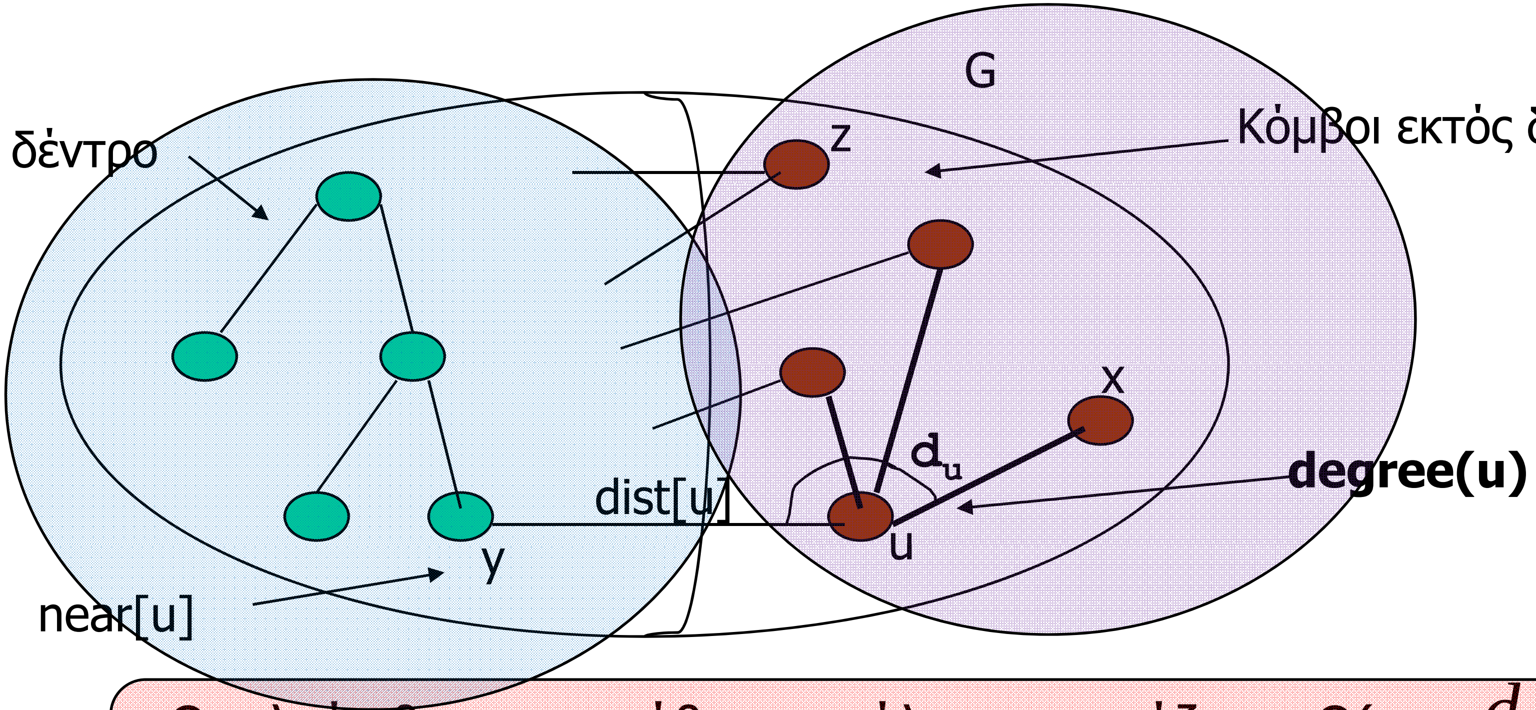
near(i) ← 0; dist(i) ← ∞;

$V_T \leftarrow \{s\}$; $E_T \leftarrow \varnothing$;

Αλγόριθμος Prim (nearest neighbour)

```
while |V_T| < n do
    u ← vertex in(V -V_T) with smallest dist(u);
    if (dist(u) >= ∞ then
        "graph disconnected"; exit;
    E_T ← E_T ∪ {(u, near(u))}; V_T ← V_T ∪ {u},
                        dist(u)=0; near(u)=u;
    for x in Γ(u) and x in (V - V_T) do
        if w_ux < dist(x) then
            dist(x) ← w_ux; near(x) ← u
end;   (****)
```

Πολυπλοκότητα Prim



Ο αλγόριθμος σε κάθε επανάληψη χρειάζεται O(n + d_i) πράξεις, όπου d_i ο βαθμός του εισερχόμενου στο δέντρο κόμβου στην i επανάληψη.

Αφού ο αλγόριθμος απαιτεί n-1 επαναλήψεις, η πολυπλοκότητα είναι O(n^2)

⇒ Αν G συνεκτικός ⟶ εύρεση του ΔΕΕΚ σε n-1 επαναλήψεις.

⇒ Αν στην i επανάληψη (i < n) δεν ευρίσκουμε πλευρά [x,y] με x στο δένδρο και y έξω από το δένδρο, τότε ο γράφος δεν είναι συνδεδεμένος

Δάσος επικάλυψης

Αν ο γράφος δεν είναι συνεκτικός, τότε ...

Ένα δάσος επικάλυψης ελαχίστου κόστους

Ο αλγόριθμος δίνει τη βέλτιστη λύση (από το θεώρημα βέλτιστου ΔΕΕΚ παρατηρώντας ότι το δάσος εκκίνησης συνίσταται από n μεμονωμένους κόμβους).