

F11 - Grafalgoritmer

5DV149 Datastrukturer och algoritmer

Kapitel 17

Niclas Börnin

niclas.borlin@cs.umu.se

2024-02-15 Tor

Innehåll

1. Traversering
 - ▶ Bredden-först
 - ▶ Djupet-först
2. Finna kortaste vägen
 - ▶ Från en nod till alla andra noder:
 - ▶ Dijkstras algoritm
 - ▶ Från alla noder till alla andra noder:
 - ▶ Floyds algoritm
3. Konstruera ett (minsta) uppspännande träd
 - ▶ Kruskals algoritm
 - ▶ Prims algoritm

1. Traversering av grafer

Blank

Bredden-först-traversering

Bredden-först-traversering

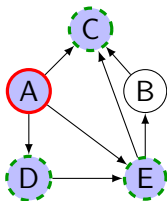
- ▶ Man undersöker först noden, sedan dess grannar, grannarnas grannar, osv.
- ▶ Grafen kan innehålla **cykler** — risk för oändlig loop
 - ▶ Markera om noden har **setts**
- ▶ En **kö** hjälper oss hålla reda på grannarna
- ▶ Endast noder till vilka det finns en väg från utgångsnoden kommer att besökas

Algorithm, bredden-först-traversering av graf

```
Algorithm g=Traverse-bf-order(n: Node, g: Graph)  
// Input: A node n in a graph g to be traversed  
  
    // Mark the starting node as seen  
    (n, g) ← Set-seen(n, g)  
    // Put it in an empty queue  
    q ← Enqueue(n, Queue-empty())  
  
    while not Isempty(q) do  
        // Pick first node from queue  
        n ← Front(q)  
        q ← Dequeue(q)  
        // Get its neighbours  
        neighbour-set ← Neighbours(n, g)  
        for each neighbour b in neighbour-set do  
            if not Is-seen(b, g) then  
                // Mark unseen node as seen and put it in the queue  
                (b, g) ← Set-seen(b, g)  
                q ← Enqueue(b, q)
```

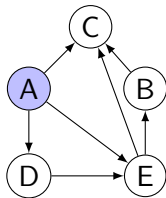
Visualiseringssymboler

- ▶ **Aktuell** nod markeras med **röd ring**
- ▶ **Ljusblå färg** betyder **sedd** (*seen*) nod
- ▶ Noder i **kön** markeras med **grönstreckad cirkel**
- ▶ Bågar som motsvarar hur vi "kom till" en aktuell nod markeras med **tjock blå linje**



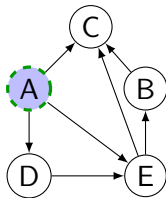
$g = \text{Traverse-breadth-first}(A, g)$

► $(A, g) \leftarrow \text{Set-seen}(A, g);$



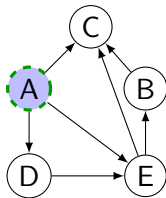
$g = \text{Traverse-breadth-first}(A, g)$

- ▶ $(A, g) \leftarrow \text{Set-seen}(A, g);$
- ▶ $q = \{A\};$



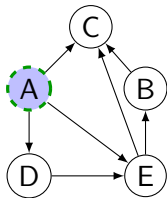
$g = \text{Traverse-breadth-first}(A, g)$

- ▶ $(A, g) \leftarrow \text{Set-seen}(A, g);$
- ▶ $q = \{A\};$
- ▶ while not lsempy(q)...



$g = \text{Traverse-breadth-first}(A, g)$

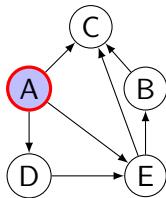
► while not lsempy(q)...



$g = \text{Traverse-breadth-first}(A, g)$

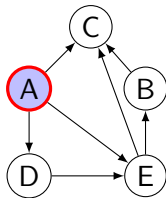
▶ while not lsempy(q)...

▶ $n = A$; $q = \{\}$;



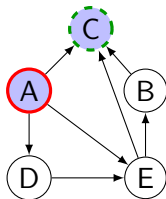
$g = \text{Traverse-breadth-first}(A, g)$

- ▶ while not lsempy(q)...
- ▶ $n = A$; $q = \{\}$;
- ▶ neighbours = {C, E, D}



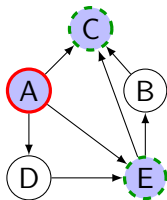
$g = \text{Traverse-breadth-first}(A, g)$

- ▶ while not lsempy(q)...
- ▶ $n = A$; $q = \{\}$;
- ▶ neighbours = {C, E, D}
- ▶ C not seen
 - ▶ $(C, g) \leftarrow \text{Set-seen}(C, g)$; $q = \{C\}$;



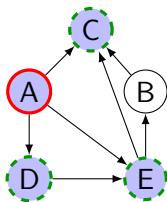
$g = \text{Traverse-breadth-first}(A, g)$

- ▶ while not lsempy(q)...
- ▶ $n = A$; $q = \{\}$;
- ▶ neighbours = {C, E, D}
- ▶ C not seen
 - ▶ $(C, g) \leftarrow \text{Set-seen}(C, g)$; $q = \{C\}$;
- ▶ E not seen
 - ▶ $(E, g) \leftarrow \text{Set-seen}(E, g)$; $q = \{C, E\}$;



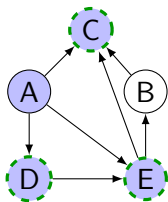
$g = \text{Traverse-breadth-first}(A, g)$

- ▶ while not lsempy(q)...
- ▶ $n = A$; $q = \{\}$;
- ▶ neighbours = {C, E, D}
- ▶ C not seen
 - ▶ $(C, g) \leftarrow \text{Set-seen}(C, g)$; $q = \{C\}$;
- ▶ E not seen
 - ▶ $(E, g) \leftarrow \text{Set-seen}(E, g)$; $q = \{C, E\}$;
- ▶ D not seen
 - ▶ $(D, g) \leftarrow \text{Set-seen}(D, g)$; $q = \{C, E, D\}$;



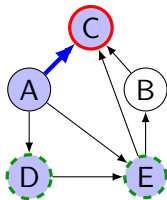
$g = \text{Traverse-breadth-first}(A, g)$

► while not lsempy(q)...



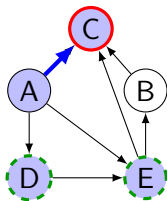
$g = \text{Traverse-breadth-first}(A, g)$

- ▶ while not lsempy(q)...
- ▶ $n = C$; $q = \{E, D\}$;



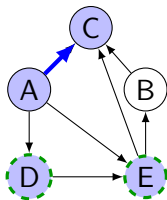
$g = \text{Traverse-breadth-first}(A, g)$

- ▶ while not lsempy(q)...
- ▶ $n = C$; $q = \{E, D\}$;
- ▶ neighbours = $\{\}$



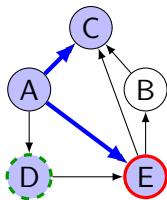
$g = \text{Traverse-breadth-first}(A, g)$

► while not lsempy(q)...



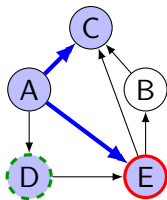
$g = \text{Traverse-breadth-first}(A, g)$

- ▶ while not lsempy(q)...
- ▶ $n = E$; $q = \{D\}$;



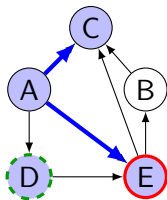
$g = \text{Traverse-breadth-first}(A, g)$

- ▶ while not lseempty(q)...
- ▶ $n = E$; $q = \{D\}$;
- ▶ neighbours = $\{C, B\}$



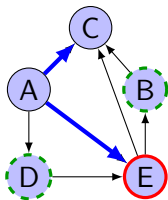
$g = \text{Traverse-breadth-first}(A, g)$

- ▶ while not lsempy(q)...
- ▶ $n = E$; $q = \{D\}$;
- ▶ neighbours = $\{C, B\}$
- ▶ C seen



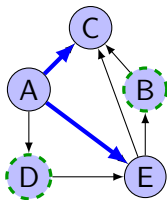
$g = \text{Traverse-breadth-first}(A, g)$

- ▶ while not lsempy(q)...
- ▶ $n = E$; $q = \{D\}$;
- ▶ neighbours = $\{C, B\}$
- ▶ C seen
- ▶ B not seen
 - ▶ $(B, g) \leftarrow \text{Set-seen}(B, g)$; $q = \{D, B\}$;



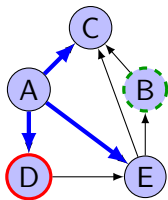
$g = \text{Traverse-breadth-first}(A, g)$

► while not lsempy(q)...



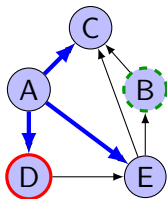
$g = \text{Traverse-breadth-first}(A, g)$

- ▶ while not lsempy(q)...
- ▶ $n = D$; $q = \{B\}$;



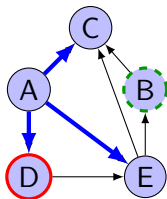
$g = \text{Traverse-breadth-first}(A, g)$

- ▶ while not lsempy(q)...
- ▶ $n = D$; $q = \{B\}$;
- ▶ neighbours = $\{E\}$



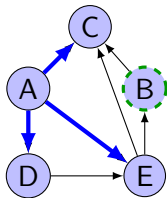
$g = \text{Traverse-breadth-first}(A, g)$

- ▶ while not lsempy(q)...
- ▶ $n = D$; $q = \{B\}$;
- ▶ neighbours = $\{E\}$
- ▶ E seen



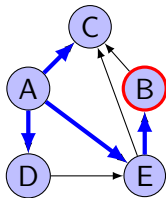
$g = \text{Traverse-breadth-first}(A, g)$

► while not lsempy(q)...



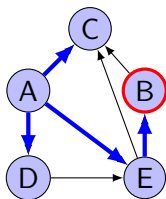
$g = \text{Traverse-breadth-first}(A, g)$

- ▶ while not lsempy(q)...
- ▶ $n = B$; $q = \{\}$;



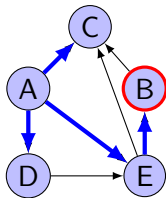
$g = \text{Traverse-breadth-first}(A, g)$

- ▶ while not lsempy(q)...
- ▶ $n = B$; $q = \{\}$;
- ▶ neighbours = {C}



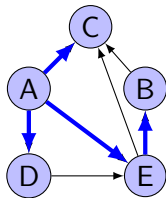
$g = \text{Traverse-breadth-first}(A, g)$

- ▶ while not lseempty(q)...
- ▶ $n = B$; $q = \{\}$;
- ▶ neighbours = {C}
- ▶ C seen



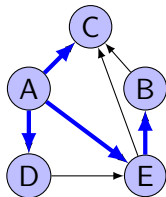
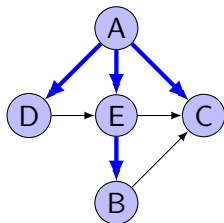
$g = \text{Traverse-breadth-first}(A, g)$

► while not lsempy(q)...



$g = \text{Traverse-breadth-first}(A, g)$

- Klar!
- Notera att de blå bågarna utgör ett *uppspännande träd*



Djupet-först-traversering

Djupet-först-traversering

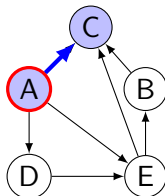
- ▶ Ansats:
 1. Starta i en utgångsnod
 2. Besök dess grannar **djupet-först, rekursivt**
- ▶ Grafen kan innehålla **cykler** — risk för oändlig loop
 - ▶ Lösning: Håll reda på om noden är **besökt** eller ej
 - ▶ Gör rekursivt anrop endast för **icke besökta** noder
 - ▶ Motsvarar att undersöka en labyrinth genom att markera de vägar man gått med färg
- ▶ Endast de noder man kan nå från utgångsnoden kommer att besökas

Algorithm för djupet-först-traversering av graf

```
Algorithm Traverse-depth-first(n: Node, g: Graph)  
// Input: A node n in a graph g to be traversed  
// Output: The modified graph after traversal  
  
// Mark the start node as visited.  
(n, g)  $\leftarrow$  Set-visited(n, g)  
// Get all its neighbours  
neighbour-set  $\leftarrow$  Neighbours(n, g)  
for each neighbour b in neighbour-set do  
    if not Is-visited(b, g) then  
        // Visit unless visited  
        g  $\leftarrow$  Traverse-depth-first(b, g)  
return g
```

Visualiseringssymboler

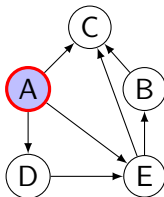
- ▶ Aktuell nod n markeras med **röd ring**
- ▶ Ljusblå färg betyder **besökt** (*visited*) nod
- ▶ Överstrukna noder i grannmängden N illustrerar noder **redan** **behandlade** i `for`-loopen
- ▶ Vid rekursivt anrop läggs aktuell nod n och grannmängden N på en **stack**
- ▶ Bågarna som motsvarar **rekursiva anrop** markeras med **tjock blå linje**



($n=A$, $\{\cancel{C}, E, D\}$)

$g = \text{Traverse-depth-first}(A, g)$ för riktad graf

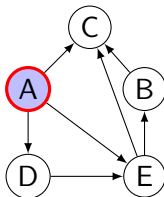
► $n \leftarrow A$, markera som besökt



($n=A$)

$g = \text{Traverse-depth-first}(A, g)$ för riktad graf

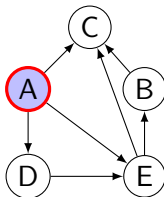
- ▶ $n \leftarrow A$, markera som besökt
- ▶ Grannar: $\{C, E, D\}$



$(n=A, \{C, E, D\})$

$g = \text{Traverse-depth-first}(A, g)$ för riktad graf

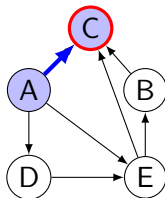
- ▶ $n \leftarrow A$, markera som besökt
- ▶ Grannar: $\{C, E, D\}$
- ▶ C ej besökt \rightarrow anropa $\text{Traverse-depth-first}(C, g)$.



$(n=A, \{C, E, D\})$

$g = \text{Traverse-depth-first}(C, g)$ för riktad graf

► $n \leftarrow C$, markera som besökt

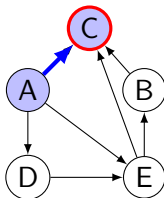


($n=C$)

($n=A, \{C, E, D\}$)

$g = \text{Traverse-depth-first}(C, g)$ för riktad graf

- ▶ $n \leftarrow C$, markera som besökt
- ▶ Inga grannar.

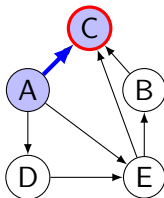


$(n=C, \{\})$

$(n=A, \{C, E, D\})$

$g = \text{Traverse-depth-first}(C, g)$ för riktad graf

- ▶ $n \leftarrow C$, markera som besökt
- ▶ Inga grannar.
- ▶ Färdig med C, återvänd

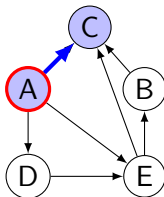


$(n=C, \{\})$

$(n=A, \{C, E, D\})$

$g = \text{Traverse-depth-first}(A, g)$ för riktad graf

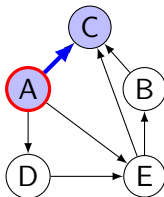
- ▶ $n \leftarrow A$, markera som besökt
- ▶ Grannar: $\{C, E, D\}$
- ▶ C ej besökt \rightarrow anropa $\text{Traverse-depth-first}(C, g)$.



$(n=A, \{C, E, D\})$

$g = \text{Traverse-depth-first}(A, g)$ för riktad graf

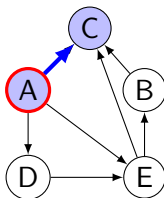
- ▶ $n \leftarrow A$, markera som besökt
- ▶ Grannar: $\{C, E, D\}$
- ▶ C ej besökt \rightarrow anropa $\text{Traverse-depth-first}(C, g)$.
- ▶ C färdig \rightarrow Grannar: $\{\cancel{C}, E, D\}$



$(n=A, \{\cancel{C}, E, D\})$

$g = \text{Traverse-depth-first}(A, g)$ för riktad graf

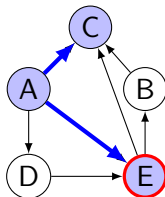
- ▶ $n \leftarrow A$, markera som besökt
- ▶ Grannar: $\{C, E, D\}$
- ▶ C ej besökt \rightarrow anropa $\text{Traverse-depth-first}(C, g)$.
- ▶ C färdig \rightarrow Grannar: $\{\cancel{C}, E, D\}$
- ▶ E ej besökt \rightarrow anropa $\text{Traverse-depth-first}(E, g)$.



$(n=A, \{\cancel{C}, E, D\})$

$g = \text{Traverse-depth-first}(E, g)$ för riktad graf

► $n \leftarrow E$, markera som besökt

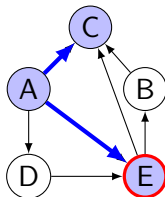


($n=E$)

($n=A, \{ \textcolor{blue}{C}, E, D \}$)

$g = \text{Traverse-depth-first}(E, g)$ för riktad graf

- ▶ $n \leftarrow E$, markera som besökt
- ▶ Grannar: $\{B, C\}$

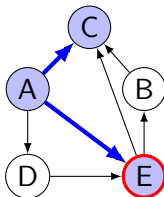


$(n=E, \{B, C\})$

$(n=A, \{\cancel{C}, E, D\})$

$g = \text{Traverse-depth-first}(E, g)$ för riktad graf

- ▶ $n \leftarrow E$, markera som besökt
- ▶ Grannar: $\{B, C\}$
- ▶ B ej besökt \rightarrow anropa $\text{Traverse-depth-first}(B, g)$

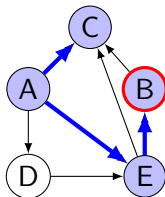


$(n=E, \{B, C\})$

$(n=A, \{\cancel{C}, E, D\})$

$g = \text{Traverse-depth-first}(B, g)$ för riktad graf

► $n \leftarrow B$, markera som besökt



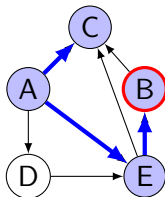
($n=B$)

($n=E, \{B, C\}$)

($n=A, \{\textcolor{blue}{C}, E, D\}$)

$g = \text{Traverse-depth-first}(B, g)$ för riktad graf

- ▶ $n \leftarrow B$, markera som besökt
- ▶ Grannar: $\{C\}$



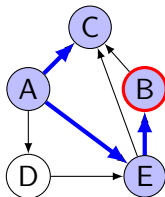
$(n=B, \{C\})$

$(n=E, \{B, C\})$

$(n=A, \{\cancel{C}, E, D\})$

$g = \text{Traverse-depth-first}(B, g)$ för riktad graf

- ▶ $n \leftarrow B$, markera som besökt
- ▶ Grannar: $\{C\}$
- ▶ C besökt \rightarrow Grannar: $\{\emptyset\}$



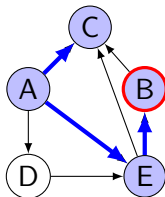
$(n=B, \{\emptyset\})$

$(n=E, \{B, C\})$

$(n=A, \{\emptyset, E, D\})$

$g = \text{Traverse-depth-first}(B, g)$ för riktad graf

- ▶ $n \leftarrow B$, markera som besökt
- ▶ Grannar: $\{C\}$
- ▶ C besökt \rightarrow Grannar: $\{\emptyset\}$
- ▶ Färdig med B, återvänd



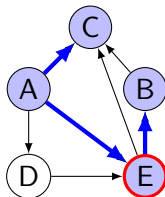
$(n=B, \{\emptyset\})$

$(n=E, \{B, C\})$

$(n=A, \{\emptyset, E, D\})$

$g = \text{Traverse-depth-first}(E, g)$ för riktad graf

- ▶ $n \leftarrow E$, markera som besökt
- ▶ Grannar: $\{B, C\}$
- ▶ B ej besökt \rightarrow anropa $\text{Traverse-depth-first}(B, g)$

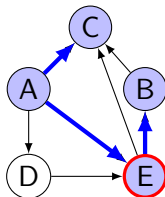


$(n=E, \{B, C\})$

$(n=A, \{\cancel{C}, E, D\})$

$g = \text{Traverse-depth-first}(E, g)$ för riktad graf

- ▶ $n \leftarrow E$, markera som besökt
- ▶ Grannar: $\{B, C\}$
- ▶ B ej besökt \rightarrow anropa $\text{Traverse-depth-first}(B, g)$
- ▶ B färdig \rightarrow Grannar: $\{\cancel{B}, C\}$

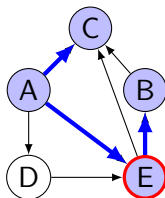


$(n=E, \{\cancel{B}, C\})$

$(n=A, \{\cancel{C}, E, D\})$

$g = \text{Traverse-depth-first}(E, g)$ för riktad graf

- ▶ $n \leftarrow E$, markera som besökt
- ▶ Grannar: $\{B, C\}$
- ▶ B ej besökt \rightarrow anropa $\text{Traverse-depth-first}(B, g)$
- ▶ B färdig \rightarrow Grannar: $\{\cancel{B}, C\}$
- ▶ C besökt \rightarrow Grannar: $\{\cancel{B}, \cancel{C}\}$

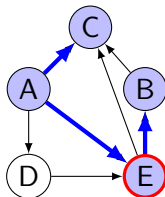


$(n=E, \{\cancel{B}, \cancel{C}\})$

$(n=A, \{\cancel{C}, E, D\})$

$g = \text{Traverse-depth-first}(E, g)$ för riktad graf

- ▶ $n \leftarrow E$, markera som besökt
- ▶ Grannar: $\{B, C\}$
- ▶ B ej besökt \rightarrow anropa $\text{Traverse-depth-first}(B, g)$
- ▶ B färdig \rightarrow Grannar: $\{\cancel{B}, C\}$
- ▶ C besökt \rightarrow Grannar: $\{\cancel{B}, \cancel{C}\}$
- ▶ Färdig med E, återvänd

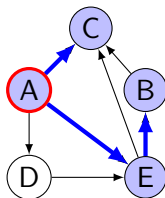


$(n=E, \{\cancel{B}, \cancel{C}\})$

$(n=A, \{\cancel{C}, E, D\})$

$g = \text{Traverse-depth-first}(A, g)$ för riktad graf

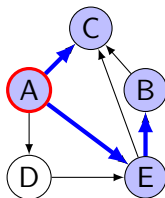
- ▶ $n \leftarrow A$, markera som besökt
- ▶ Grannar: $\{C, E, D\}$
- ▶ C ej besökt \rightarrow anropa $\text{Traverse-depth-first}(C, g)$.
- ▶ C färdig \rightarrow Grannar: $\{\cancel{C}, E, D\}$
- ▶ E ej besökt \rightarrow anropa $\text{Traverse-depth-first}(E, g)$.



$(n=A, \{\cancel{C}, E, D\})$

$g = \text{Traverse-depth-first}(A, g)$ för riktad graf

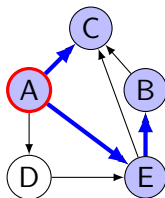
- ▶ $n \leftarrow A$, markera som besökt
- ▶ Grannar: $\{C, E, D\}$
- ▶ C ej besökt \rightarrow anropa $\text{Traverse-depth-first}(C, g)$.
- ▶ C färdig \rightarrow Grannar: $\{\cancel{C}, E, D\}$
- ▶ E ej besökt \rightarrow anropa $\text{Traverse-depth-first}(E, g)$.
- ▶ E färdig \rightarrow Grannar: $\{\cancel{C}, \cancel{E}, D\}$



$(n=A, \{\cancel{C}, \cancel{E}, D\})$

$g = \text{Traverse-depth-first}(A, g)$ för riktad graf

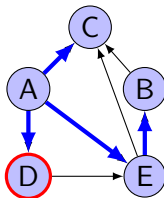
- ▶ $n \leftarrow A$, markera som besökt
- ▶ Grannar: $\{C, E, D\}$
- ▶ C ej besökt \rightarrow anropa $\text{Traverse-depth-first}(C, g)$.
- ▶ C färdig \rightarrow Grannar: $\{\cancel{C}, E, D\}$
- ▶ E ej besökt \rightarrow anropa $\text{Traverse-depth-first}(E, g)$.
- ▶ E färdig \rightarrow Grannar: $\{\cancel{C}, \cancel{E}, D\}$
- ▶ D ej besökt \rightarrow anropa $\text{Traverse-depth-first}(D, g)$.



$(n=A, \{\cancel{C}, \cancel{E}, D\})$

$g = \text{Traverse-depth-first}(D, g)$ för riktad graf

- $n \leftarrow D$, markera som besökt

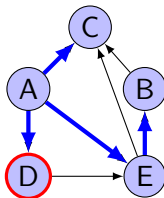


($n=D$)

($n=A, \{\cancel{C}, \cancel{E}, D\}$)

$g = \text{Traverse-depth-first}(D, g)$ för riktad graf

- ▶ $n \leftarrow D$, markera som besökt
- ▶ Grannar: $\{E\}$

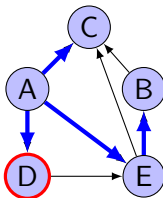


$(n=D, \{E\})$

$(n=A, \{\cancel{C}, \cancel{E}, D\})$

$g = \text{Traverse-depth-first}(D, g)$ för riktad graf

- ▶ $n \leftarrow D$, markera som besökt
- ▶ Grannar: $\{E\}$
- ▶ E besökt \rightarrow Grannar: $\{\cancel{E}\}$

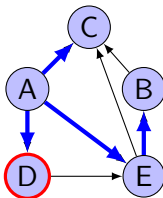


$(n=D, \{\cancel{E}\})$

$(n=A, \{\cancel{C}, \cancel{E}, D\})$

$g = \text{Traverse-depth-first}(D, g)$ för riktad graf

- ▶ $n \leftarrow D$, markera som besökt
- ▶ Grannar: $\{E\}$
- ▶ E besökt \rightarrow Grannar: $\{\cancel{E}\}$
- ▶ Färdig med D, återvänd

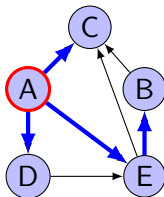


$(n=D, \{\cancel{E}\})$

$(n=A, \{\cancel{C}, \cancel{E}, D\})$

$g = \text{Traverse-depth-first}(A, g)$ för riktad graf

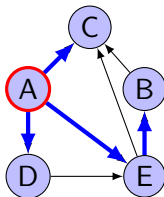
- ▶ $n \leftarrow A$, markera som besökt
- ▶ Grannar: $\{C, E, D\}$
- ▶ C ej besökt \rightarrow anropa $\text{Traverse-depth-first}(C, g)$.
- ▶ C färdig \rightarrow Grannar: $\{\cancel{C}, E, D\}$
- ▶ E ej besökt \rightarrow anropa $\text{Traverse-depth-first}(E, g)$.
- ▶ E färdig \rightarrow Grannar: $\{\cancel{C}, \cancel{E}, D\}$
- ▶ D ej besökt \rightarrow anropa $\text{Traverse-depth-first}(D, g)$.



$(n=A, \{\cancel{C}, \cancel{E}, D\})$

$g = \text{Traverse-depth-first}(A, g)$ för riktad graf

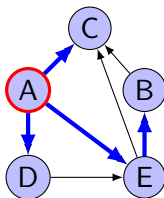
- ▶ $n \leftarrow A$, markera som besökt
- ▶ Grannar: $\{C, E, D\}$
- ▶ C ej besökt \rightarrow anropa $\text{Traverse-depth-first}(C, g)$.
- ▶ C färdig \rightarrow Grannar: $\{\cancel{C}, E, D\}$
- ▶ E ej besökt \rightarrow anropa $\text{Traverse-depth-first}(E, g)$.
- ▶ E färdig \rightarrow Grannar: $\{\cancel{C}, \cancel{E}, D\}$
- ▶ D ej besökt \rightarrow anropa $\text{Traverse-depth-first}(D, g)$.
- ▶ D färdig \rightarrow Grannar: $\{\cancel{C}, \cancel{E}, \cancel{D}\}$



$(n=A, \{\cancel{C}, \cancel{E}, \cancel{D}\})$

$g = \text{Traverse-depth-first}(A, g)$ för riktad graf

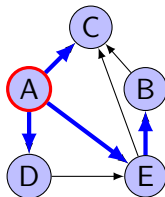
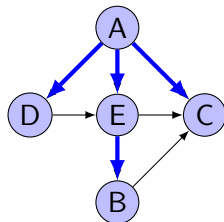
- ▶ $n \leftarrow A$, markera som besökt
- ▶ Grannar: $\{C, E, D\}$
- ▶ C ej besökt \rightarrow anropa $\text{Traverse-depth-first}(C, g)$.
- ▶ C färdig \rightarrow Grannar: $\{\cancel{C}, E, D\}$
- ▶ E ej besökt \rightarrow anropa $\text{Traverse-depth-first}(E, g)$.
- ▶ E färdig \rightarrow Grannar: $\{\cancel{C}, \cancel{E}, D\}$
- ▶ D ej besökt \rightarrow anropa $\text{Traverse-depth-first}(D, g)$.
- ▶ D färdig \rightarrow Grannar: $\{\cancel{C}, \cancel{E}, \cancel{D}\}$
- ▶ Färdig med A, återvänd



$(n=A, \{\cancel{C}, \cancel{E}, \cancel{D}\})$

Klar

- ▶ Även här fick vi ett *uppspännande träd*



Algorithm för djupet-först-traversering av graf (igen)

```
Algorithm Traverse-depth-first(n: Node, g: Graph)  
// Input: A node n in a graph g to be traversed  
// Output: The modified graph after traversal  
  
// Mark the start node as visited.  
(n, g)  $\leftarrow$  Set-visited(n, g)  
// Get all its neighbours  
neighbour-set  $\leftarrow$  Neighbours(n, g)  
for each neighbour b in neighbour-set do  
    if not Is-visited(b, g) then  
        // Visit unless visited  
        g  $\leftarrow$  Traverse-depth-first(b, g)  
return g
```

Fråga

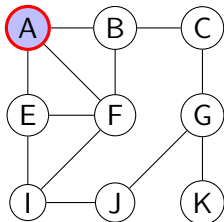
- ▶ Hur behöver algoritmen modifieras för att fungera på en oriktad graf?

Fråga

- ▶ Hur behöver algoritmen modifieras för att fungera på en **oriktad** graf?
 - ▶ Inte alls!
 - ▶ Funktionen **Neighbours** hanterar det

$g = \text{Traverse-depth-first}(A, g)$ för oriktad graf

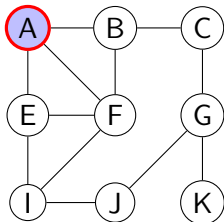
► $n \leftarrow A$, markera som besökt



($n=A$)

$g = \text{Traverse-depth-first}(A, g)$ för oriktad graf

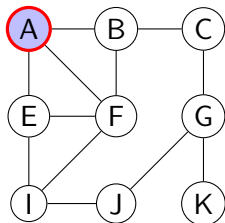
- ▶ $n \leftarrow A$, markera som besökt
- ▶ Grannar: $\{E, F, B\}$



$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(A, g)$ för oriktad graf

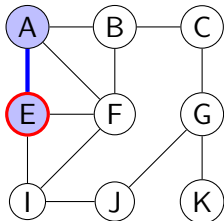
- ▶ $n \leftarrow A$, markera som besökt
- ▶ Grannar: $\{E, F, B\}$
- ▶ E ej besökt \rightarrow anropa $\text{Traverse-depth-first}(E, g)$.



$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(E, g)$ för oriktad graf

► $n \leftarrow E$, markera som besökt

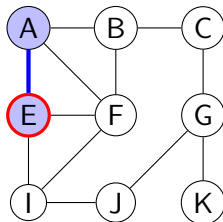


($n=E$)

($n=A, \{E, F, B\}$)

$g = \text{Traverse-depth-first}(E, g)$ för oriktad graf

- ▶ $n \leftarrow E$, markera som besökt
- ▶ Grannar: $\{I, F, A\}$

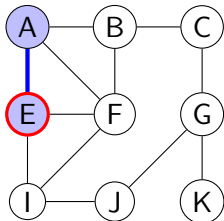


$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(E, g)$ för oriktad graf

- ▶ $n \leftarrow E$, markera som besökt
- ▶ Grannar: $\{I, F, A\}$
- ▶ I ej besökt \rightarrow anropa $\text{Traverse-depth-first}(I, g)$.

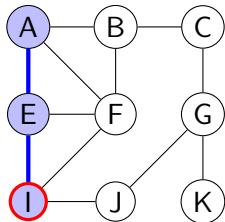


$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(l, g)$ för oriktad graf

► $n \leftarrow l$, markera som besökt



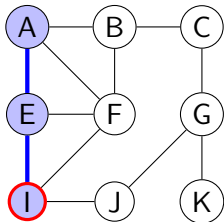
($n=I$)

($n=E, \{I, F, A\}$)

($n=A, \{E, F, B\}$)

$g = \text{Traverse-depth-first}(l, g)$ för oriktad graf

- ▶ $n \leftarrow l$, markera som besökt
- ▶ Grannar: $\{E, J, F\}$



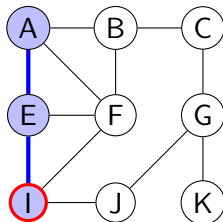
$(n=l, \{E, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(I, g)$ för oriktad graf

- ▶ $n \leftarrow I$, markera som besökt
- ▶ Grannar: $\{E, J, F\}$
- ▶ E redan besökt \rightarrow Grannar: $\{\cancel{E}, J, F\}$



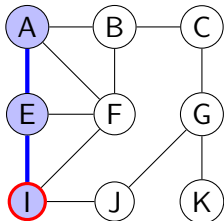
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(I, g)$ för oriktad graf

- ▶ $n \leftarrow I$, markera som besökt
- ▶ Grannar: $\{E, J, F\}$
- ▶ E redan besökt \rightarrow Grannar: $\{\cancel{E}, J, F\}$
- ▶ J ej besökt \rightarrow anropa $\text{Traverse-depth-first}(J, g)$.



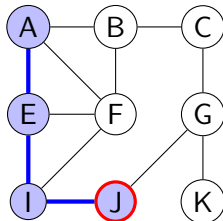
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(J, g)$ för oriktad graf

► $n \leftarrow J$, markera som besökt



($n=J$)

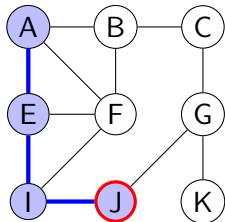
($n=I$, {~~E~~, J, F})

($n=E$, {I, F, A})

($n=A$, {E, F, B})

$g = \text{Traverse-depth-first}(J, g)$ för oriktad graf

- ▶ $n \leftarrow J$, markera som besökt
- ▶ Grannar: $\{G, I\}$



$(n=J, \{G, I\})$

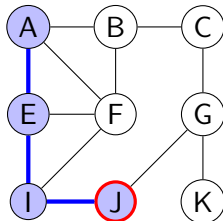
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(J, g)$ för oriktad graf

- ▶ $n \leftarrow J$, markera som besökt
- ▶ Grannar: $\{G, I\}$
- ▶ G ej besökt \rightarrow anropa $\text{Traverse-depth-first}(G, g)$.



$(n=J, \{G, I\})$

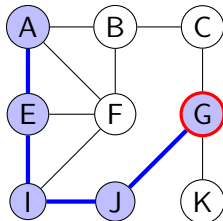
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(G, g)$ för oriktad graf

► $n \leftarrow G$, markera som besökt



($n=G$)

($n=J$, { G, I })

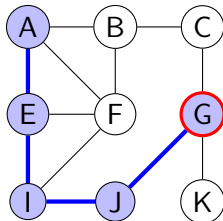
($n=I$, { ~~E~~ , J, F })

($n=E$, { I, F, A })

($n=A$, { E, F, B })

$g = \text{Traverse-depth-first}(G, g)$ för oriktad graf

- ▶ $n \leftarrow G$, markera som besökt
- ▶ Grannar: $\{C, K, J\}$



$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

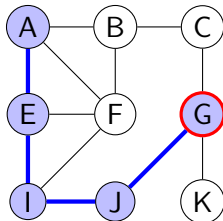
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(G, g)$ för oriktad graf

- ▶ $n \leftarrow G$, markera som besökt
- ▶ Grannar: $\{C, K, J\}$
- ▶ C ej besökt \rightarrow anropa $\text{Traverse-depth-first}(C, g)$.



$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

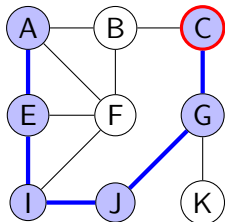
$(n=I, \{E, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(C, g)$ för oriktad graf

► $n \leftarrow C$, markera som besökt



($n=C$)

($n=G, \{C, K, J\}$)

($n=J, \{G, I\}$)

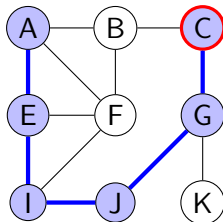
($n=I, \{\cancel{E}, J, F\}$)

($n=E, \{I, F, A\}$)

($n=A, \{E, F, B\}$)

$g = \text{Traverse-depth-first}(C, g)$ för oriktad graf

- ▶ $n \leftarrow C$, markera som besökt
- ▶ Grannar: $\{G, B\}$



($n=C$, $\{G, B\}$)

($n=G$, $\{C, K, J\}$)

($n=J$, $\{G, I\}$)

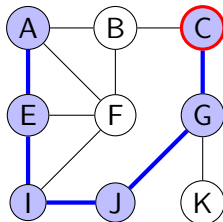
($n=I$, $\{\cancel{E}, J, F\}$)

($n=E$, $\{I, F, A\}$)

($n=A$, $\{E, F, B\}$)

$g = \text{Traverse-depth-first}(C, g)$ för oriktad graf

- ▶ $n \leftarrow C$, markera som besökt
- ▶ Grannar: $\{G, B\}$
- ▶ G redan besökt \rightarrow Grannar: $\{\cancel{G}, B\}$



$(n=C, \{\cancel{G}, B\})$

$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

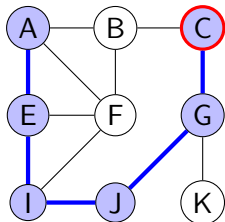
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(C, g)$ för oriktad graf

- ▶ $n \leftarrow C$, markera som besökt
- ▶ Grannar: $\{G, B\}$
- ▶ G redan besökt \rightarrow Grannar: $\{\cancel{G}, B\}$
- ▶ B ej besökt \rightarrow anropa $\text{Traverse-depth-first}(B, g)$.



$(n=C, \{\cancel{G}, B\})$

$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

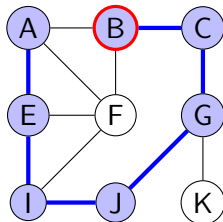
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(B, g)$ för oriktad graf

► $n \leftarrow B$, markera som besökt



($n=B$)

($n=C, \{\cancel{A}, B\}$)

($n=G, \{C, K, J\}$)

($n=J, \{G, I\}$)

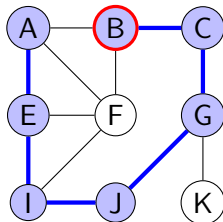
($n=I, \{\cancel{E}, J, F\}$)

($n=E, \{I, F, A\}$)

($n=A, \{E, F, B\}$)

$g = \text{Traverse-depth-first}(B, g)$ för oriktad graf

- ▶ $n \leftarrow B$, markera som besökt
- ▶ Grannar: $\{A, F, C\}$



$(n=B, \{A, F, C\})$

$(n=C, \{\cancel{A}, B\})$

$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

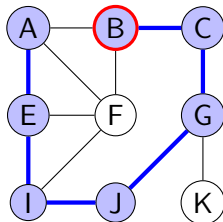
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(B, g)$ för oriktad graf

- ▶ $n \leftarrow B$, markera som besökt
- ▶ Grannar: $\{A, F, C\}$
- ▶ A redan besökt \rightarrow Grannar: $\{\cancel{A}, F, C\}$



$(n=B, \{\cancel{A}, F, C\})$

$(n=C, \{\cancel{C}, B\})$

$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

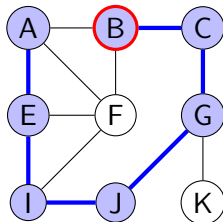
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(B, g)$ för oriktad graf

- ▶ $n \leftarrow B$, markera som besökt
- ▶ Grannar: $\{A, F, C\}$
- ▶ A redan besökt \rightarrow Grannar: $\{\cancel{A}, F, C\}$
- ▶ F ej besökt \rightarrow anropa $\text{Traverse-depth-first}(F, g)$.



$(n=B, \{\cancel{A}, F, C\})$

$(n=C, \{\cancel{C}, B\})$

$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

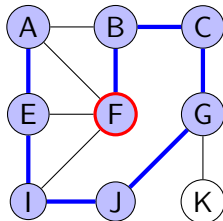
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(F, g)$ för oriktad graf

► $n \leftarrow F$, markera som besökt



($n=F$)

($n=B$, {~~A~~, F, C})

($n=C$, {~~G~~, B})

($n=G$, {C, K, J})

($n=J$, {G, I})

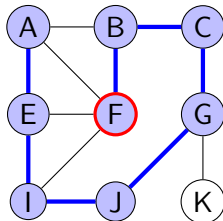
($n=I$, {~~E~~, J, F})

($n=E$, {I, F, A})

($n=A$, {E, F, B})

$g = \text{Traverse-depth-first}(F, g)$ för oriktad graf

- ▶ $n \leftarrow F$, markera som besökt
- ▶ Grannar: $\{B, A, E, I\}$



$(n=F, \{B, A, E, I\})$

$(n=B, \{\cancel{A}, F, C\})$

$(n=C, \{\cancel{G}, B\})$

$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

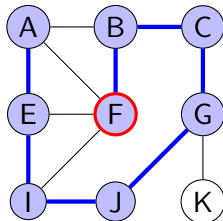
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(F, g)$ för oriktad graf

- ▶ $n \leftarrow F$, markera som besökt
- ▶ Grannar: $\{B, A, E, I\}$
- ▶ B besökt \rightarrow Grannar: $\{\cancel{B}, A, E, I\}$



$(n=F, \{\cancel{B}, A, E, I\})$

$(n=B, \{\cancel{A}, F, C\})$

$(n=C, \{\cancel{G}, B\})$

$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

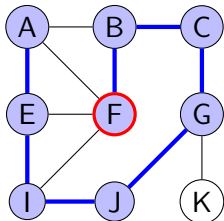
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(F, g)$ för oriktad graf

- ▶ $n \leftarrow F$, markera som besökt
- ▶ Grannar: $\{B, A, E, I\}$
- ▶ B besökt \rightarrow Grannar: $\{\cancel{B}, A, E, I\}$
- ▶ A besökt \rightarrow Grannar: $\{\cancel{B}, \cancel{A}, E, I\}$

$$(n=F, \{\cancel{B}, \cancel{A}, E, I\})$$
$$(n=B, \{A, F, C\})$$
$$(n=C, \{\cancel{G}, B\})$$
$$(n=G, \{C, K, J\})$$
$$(n=J, \{G, I\})$$
$$(n=1, \{\cancel{E}, J, F\})$$
$$(n=E, \{I, F, A\})$$
$$(n=A, \{E, F, B\})$$


$g = \text{Traverse-depth-first}(F, g)$ för oriktad graf

- ▶ $n \leftarrow F$, markera som besökt
- ▶ Grannar: $\{B, A, E, I\}$
- ▶ B besökt \rightarrow Grannar: $\{\cancel{B}, A, E, I\}$
- ▶ A besökt \rightarrow Grannar: $\{\cancel{B}, \cancel{A}, E, I\}$
- ▶ E besökt \rightarrow Grannar: $\{\cancel{B}, \cancel{A}, \cancel{E}, I\}$

$(n=F, \{\cancel{B}, \cancel{A}, \cancel{E}, I\})$

$(n=B, \{\cancel{A}, F, C\})$

$(n=C, \{\cancel{G}, B\})$

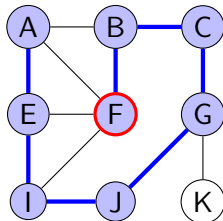
$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$



$g = \text{Traverse-depth-first}(F, g)$ för oriktad graf

- ▶ $n \leftarrow F$, markera som besökt
- ▶ Grannar: $\{B, A, E, I\}$
- ▶ B besökt \rightarrow Grannar: $\{\cancel{B}, A, E, I\}$
- ▶ A besökt \rightarrow Grannar: $\{\cancel{B}, \cancel{A}, E, I\}$
- ▶ E besökt \rightarrow Grannar: $\{\cancel{B}, \cancel{A}, \cancel{E}, I\}$
- ▶ I besökt \rightarrow Grannar: $\{\cancel{B}, \cancel{A}, \cancel{E}, \cancel{I}\}$

$(n=F, \{\cancel{B}, \cancel{A}, \cancel{E}, \cancel{I}\})$

$(n=B, \{\cancel{A}, F, C\})$

$(n=C, \{\cancel{G}, B\})$

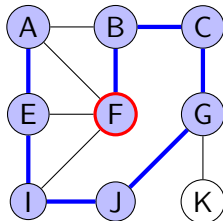
$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$



$g = \text{Traverse-depth-first}(F, g)$ för oriktad graf

- ▶ $n \leftarrow F$, markera som besökt
- ▶ Grannar: $\{B, A, E, I\}$
- ▶ B besökt \rightarrow Grannar: $\{\cancel{B}, A, E, I\}$
- ▶ A besökt \rightarrow Grannar: $\{\cancel{B}, \cancel{A}, E, I\}$
- ▶ E besökt \rightarrow Grannar: $\{\cancel{B}, \cancel{A}, \cancel{E}, I\}$
- ▶ I besökt \rightarrow Grannar: $\{\cancel{B}, \cancel{A}, \cancel{E}, \cancel{I}\}$
- ▶ Färdig med F, återvänd

$(n=F, \{\cancel{B}, \cancel{A}, \cancel{E}, \cancel{I}\})$

$(n=B, \{\cancel{A}, F, C\})$

$(n=C, \{\cancel{G}, B\})$

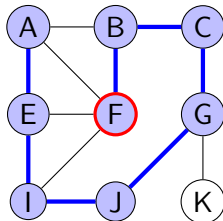
$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

$(n=I, \{\cancel{E}, J, F\})$

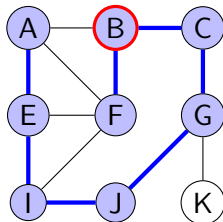
$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$



$g = \text{Traverse-depth-first}(B, g)$ för oriktad graf

- ▶ $n \leftarrow B$, markera som besökt
- ▶ Grannar: $\{A, F, C\}$
- ▶ A redan besökt \rightarrow Grannar: $\{\cancel{A}, F, C\}$
- ▶ F ej besökt \rightarrow anropa $\text{Traverse-depth-first}(F, g)$.



$(n=B, \{\cancel{A}, F, C\})$

$(n=C, \{\cancel{C}, B\})$

$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

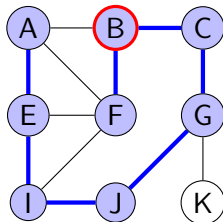
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(B, g)$ för oriktad graf

- ▶ $n \leftarrow B$, markera som besökt
- ▶ Grannar: $\{A, F, C\}$
- ▶ A redan besökt \rightarrow Grannar: $\{\cancel{A}, F, C\}$
- ▶ F ej besökt \rightarrow anropa $\text{Traverse-depth-first}(F, g)$.
- ▶ F färdig \rightarrow Grannar: $\{\cancel{A}, \cancel{F}, C\}$



$(n=B, \{\cancel{A}, \cancel{F}, C\})$

$(n=C, \{\cancel{C}, B\})$

$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(B, g)$ för oriktad graf

- ▶ $n \leftarrow B$, markera som besökt
- ▶ Grannar: $\{A, F, C\}$
- ▶ A redan besökt \rightarrow Grannar: $\{\cancel{A}, F, C\}$
- ▶ F ej besökt \rightarrow anropa $\text{Traverse-depth-first}(F, g)$.
- ▶ F färdig \rightarrow Grannar: $\{\cancel{A}, \cancel{F}, C\}$
- ▶ C besökt \rightarrow Grannar: $\{\cancel{A}, \cancel{F}, \cancel{C}\}$

$(n=B, \{\cancel{A}, \cancel{F}, \cancel{C}\})$

$(n=C, \{\cancel{G}, B\})$

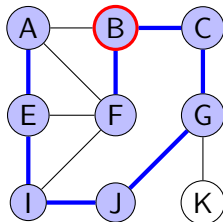
$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$



$g = \text{Traverse-depth-first}(B, g)$ för oriktad graf

- ▶ $n \leftarrow B$, markera som besökt
- ▶ Grannar: $\{A, F, C\}$
- ▶ A redan besökt \rightarrow Grannar: $\{\cancel{A}, F, C\}$
- ▶ F ej besökt \rightarrow anropa $\text{Traverse-depth-first}(F, g)$.
- ▶ F färdig \rightarrow Grannar: $\{\cancel{A}, \cancel{F}, C\}$
- ▶ C besökt \rightarrow Grannar: $\{\cancel{A}, \cancel{F}, \cancel{C}\}$
- ▶ Färdig med B, återvänd

$(n=B, \{\cancel{A}, \cancel{F}, \cancel{C}\})$

$(n=C, \{\cancel{G}, B\})$

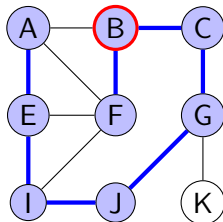
$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

$(n=I, \{\cancel{E}, J, F\})$

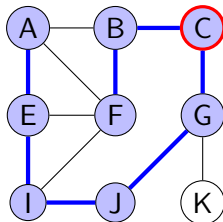
$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$



$g = \text{Traverse-depth-first}(C, g)$ för oriktad graf

- ▶ $n \leftarrow C$, markera som besökt
- ▶ Grannar: $\{G, B\}$
- ▶ G redan besökt \rightarrow Grannar: $\{\cancel{G}, B\}$
- ▶ B ej besökt \rightarrow anropa $\text{Traverse-depth-first}(B, g)$.



$(n=C, \{\cancel{G}, B\})$

$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

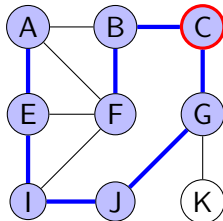
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(C, g)$ för oriktad graf

- ▶ $n \leftarrow C$, markera som besökt
- ▶ Grannar: $\{G, B\}$
- ▶ G redan besökt \rightarrow Grannar: $\{\cancel{G}, B\}$
- ▶ B ej besökt \rightarrow anropa $\text{Traverse-depth-first}(B, g)$.
- ▶ B färdig \rightarrow Grannar: $\{\cancel{G}, \cancel{B}\}$



$(n=C, \{\cancel{G}, \cancel{B}\})$

$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

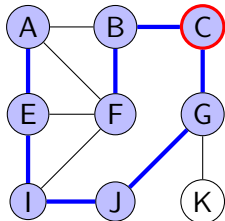
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(C, g)$ för oriktad graf

- ▶ $n \leftarrow C$, markera som besökt
- ▶ Grannar: $\{G, B\}$
- ▶ G redan besökt \rightarrow Grannar: $\{\cancel{G}, B\}$
- ▶ B ej besökt \rightarrow anropa $\text{Traverse-depth-first}(B, g)$.
- ▶ B färdig \rightarrow Grannar: $\{\cancel{G}, \cancel{B}\}$
- ▶ Färdig med C, återvänd



$(n=C, \{\cancel{G}, \cancel{B}\})$

$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

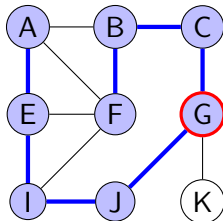
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(G, g)$ för oriktad graf

- ▶ $n \leftarrow G$, markera som besökt
- ▶ Grannar: $\{C, K, J\}$
- ▶ C ej besökt \rightarrow anropa $\text{Traverse-depth-first}(C, g)$.



$(n=G, \{C, K, J\})$

$(n=J, \{G, I\})$

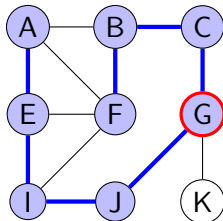
$(n=I, \{\text{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(G, g)$ för oriktad graf

- ▶ $n \leftarrow G$, markera som besökt
- ▶ Grannar: $\{C, K, J\}$
- ▶ C ej besökt \rightarrow anropa $\text{Traverse-depth-first}(C, g)$.
- ▶ C färdig \rightarrow Grannar: $\{\cancel{C}, K, J\}$



$(n=G, \{\cancel{C}, K, J\})$

$(n=J, \{G, I\})$

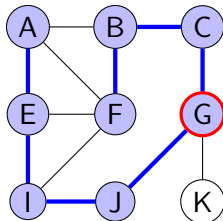
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(G, g)$ för oriktad graf

- ▶ $n \leftarrow G$, markera som besökt
- ▶ Grannar: $\{C, K, J\}$
- ▶ C ej besökt \rightarrow anropa $\text{Traverse-depth-first}(C, g)$.
- ▶ C färdig \rightarrow Grannar: $\{\cancel{C}, K, J\}$
- ▶ K ej besökt \rightarrow anropa $\text{Traverse-depth-first}(K, g)$.



$(n=G, \{\cancel{C}, K, J\})$

$(n=J, \{G, I\})$

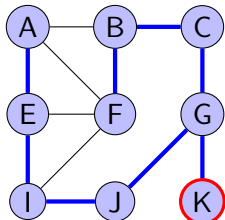
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(K, g)$ för oriktad graf

► $n \leftarrow K$, markera som besökt



($n=K$)

($n=G$, {~~C~~, K, J})

($n=J$, {G, I})

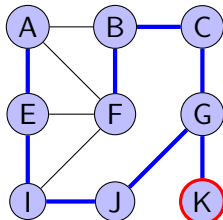
($n=I$, {~~E~~, J, F})

($n=E$, {I, F, A})

($n=A$, {E, F, B})

$g = \text{Traverse-depth-first}(K, g)$ för oriktad graf

- ▶ $n \leftarrow K$, markera som besökt
- ▶ Grannar: $\{G\}$



($n=K$, $\{G\}$)

($n=G$, $\{\cancel{C}, K, J\}$)

($n=J$, $\{G, I\}$)

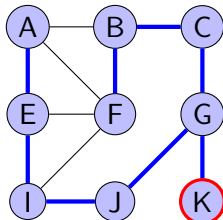
($n=I$, $\{\cancel{E}, J, F\}$)

($n=E$, $\{I, F, A\}$)

($n=A$, $\{E, F, B\}$)

$g = \text{Traverse-depth-first}(K, g)$ för oriktad graf

- ▶ $n \leftarrow K$, markera som besökt
- ▶ Grannar: $\{G\}$
- ▶ G besökt \rightarrow Grannar: $\{\cancel{G}\}$



$(n=K, \{\cancel{G}\})$

$(n=G, \{\cancel{G}, K, J\})$

$(n=J, \{G, I\})$

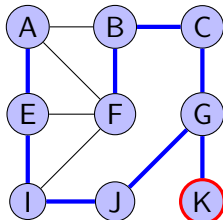
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(K, g)$ för oriktad graf

- ▶ $n \leftarrow K$, markera som besökt
- ▶ Grannar: $\{G\}$
- ▶ G besökt \rightarrow Grannar: $\{\cancel{G}\}$
- ▶ Färdig med K , återvänd



$(n=K, \{\cancel{G}\})$

$(n=G, \{\cancel{G}, K, J\})$

$(n=J, \{G, I\})$

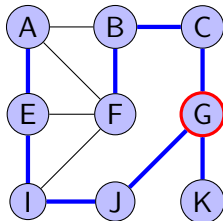
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(G, g)$ för oriktad graf

- ▶ $n \leftarrow G$, markera som besökt
- ▶ Grannar: $\{C, K, J\}$
- ▶ C ej besökt \rightarrow anropa $\text{Traverse-depth-first}(C, g)$.
- ▶ C färdig \rightarrow Grannar: $\{\cancel{C}, K, J\}$
- ▶ K ej besökt \rightarrow anropa $\text{Traverse-depth-first}(K, g)$.



$(n=G, \{\cancel{C}, K, J\})$

$(n=J, \{G, I\})$

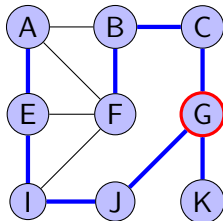
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(G, g)$ för oriktad graf

- ▶ $n \leftarrow G$, markera som besökt
- ▶ Grannar: $\{C, K, J\}$
- ▶ C ej besökt \rightarrow anropa $\text{Traverse-depth-first}(C, g)$.
- ▶ C färdig \rightarrow Grannar: $\{\cancel{C}, K, J\}$
- ▶ K ej besökt \rightarrow anropa $\text{Traverse-depth-first}(K, g)$.
- ▶ K färdig \rightarrow Grannar: $\{\cancel{C}, \cancel{K}, J\}$



$(n=G, \{\cancel{C}, \cancel{K}, J\})$

$(n=J, \{G, I\})$

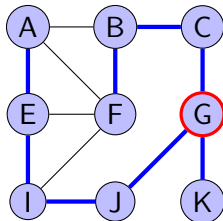
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(G, g)$ för oriktad graf

- ▶ $n \leftarrow G$, markera som besökt
- ▶ Grannar: $\{C, K, J\}$
- ▶ C ej besökt \rightarrow anropa $\text{Traverse-depth-first}(C, g)$.
- ▶ C färdig \rightarrow Grannar: $\{\cancel{C}, K, J\}$
- ▶ K ej besökt \rightarrow anropa $\text{Traverse-depth-first}(K, g)$.
- ▶ K färdig \rightarrow Grannar: $\{\cancel{C}, \cancel{K}, J\}$
- ▶ J besökt \rightarrow Grannar: $\{\cancel{C}, \cancel{K}, \cancel{J}\}$



$(n=G, \{\cancel{C}, \cancel{K}, \cancel{J}\})$

$(n=J, \{G, I\})$

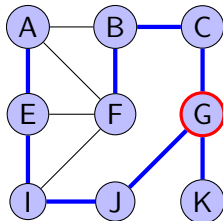
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(G, g)$ för oriktad graf

- ▶ $n \leftarrow G$, markera som besökt
- ▶ Grannar: $\{C, K, J\}$
- ▶ C ej besökt \rightarrow anropa $\text{Traverse-depth-first}(C, g)$.
- ▶ C färdig \rightarrow Grannar: $\{\cancel{C}, K, J\}$
- ▶ K ej besökt \rightarrow anropa $\text{Traverse-depth-first}(K, g)$.
- ▶ K färdig \rightarrow Grannar: $\{\cancel{C}, \cancel{K}, J\}$
- ▶ J besökt \rightarrow Grannar: $\{\cancel{C}, \cancel{K}, \cancel{J}\}$
- ▶ Färdig med G, återvänd



$(n=G, \{\cancel{C}, \cancel{K}, \cancel{J}\})$

$(n=J, \{G, I\})$

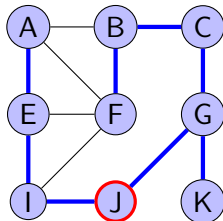
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(J, g)$ för oriktad graf

- ▶ $n \leftarrow J$, markera som besökt
- ▶ Grannar: $\{G, I\}$
- ▶ G ej besökt \rightarrow anropa $\text{Traverse-depth-first}(G, g)$.



$(n=J, \{G, I\})$

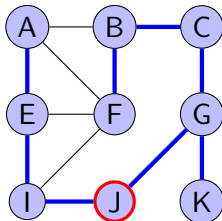
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

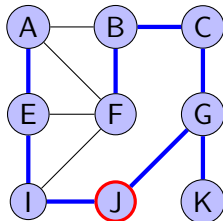
$g = \text{Traverse-depth-first}(J, g)$ för oriktad graf

- ▶ $n \leftarrow J$, markera som besökt
- ▶ Grannar: $\{G, I\}$
- ▶ G ej besökt \rightarrow anropa $\text{Traverse-depth-first}(G, g)$.
- ▶ G färdig \rightarrow Grannar: $\{\text{G}, I\}$


$$(n=J, \{\cancel{G}, I\})$$
$$(n=1, \{\cancel{E}, J, F\})$$
$$(n=E, \{I, F, A\})$$
$$(n=A, \{E, F, B\})$$

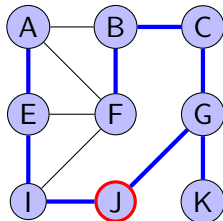
$g = \text{Traverse-depth-first}(J, g)$ för oriktad graf

- ▶ $n \leftarrow J$, markera som besökt
- ▶ Grannar: $\{G, I\}$
- ▶ G ej besökt \rightarrow anropa $\text{Traverse-depth-first}(G, g)$.
- ▶ G färdig \rightarrow Grannar: $\{\cancel{G}, I\}$
- ▶ I besökt \rightarrow Grannar: $\{\cancel{G}, \cancel{I}\}$


$$(n=J, \{\cancel{G}, I\})$$
$$(n=1, \{\cancel{E}, J, F\})$$
$$(n=E, \{I, F, A\})$$
$$(n=A, \{E, F, B\})$$

$g = \text{Traverse-depth-first}(J, g)$ för oriktad graf

- ▶ $n \leftarrow J$, markera som besökt
- ▶ Grannar: $\{G, I\}$
- ▶ G ej besökt \rightarrow anropa $\text{Traverse-depth-first}(G, g)$.
- ▶ G färdig \rightarrow Grannar: $\{\cancel{G}, I\}$
- ▶ I besökt \rightarrow Grannar: $\{\cancel{G}, \cancel{I}\}$
- ▶ Färdig med J, återvänd



$(n=J, \{\cancel{G}, \cancel{I}\})$

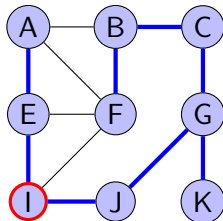
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(I, g)$ för oriktad graf

- ▶ $n \leftarrow I$, markera som besökt
- ▶ Grannar: $\{E, J, F\}$
- ▶ E redan besökt \rightarrow Grannar: $\{\cancel{E}, J, F\}$
- ▶ J ej besökt \rightarrow anropa $\text{Traverse-depth-first}(J, g)$.



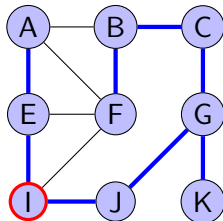
$(n=I, \{\cancel{E}, J, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(I, g)$ för oriktad graf

- ▶ $n \leftarrow I$, markera som besökt
- ▶ Grannar: $\{E, J, F\}$
- ▶ E redan besökt \rightarrow Grannar: $\{\cancel{E}, J, F\}$
- ▶ J ej besökt \rightarrow anropa $\text{Traverse-depth-first}(J, g)$.
- ▶ J färdig \rightarrow Grannar: $\{\cancel{E}, \cancel{J}, F\}$



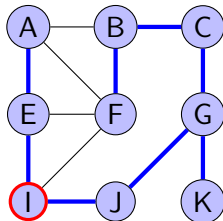
$(n=I, \{\cancel{E}, \cancel{J}, F\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(I, g)$ för oriktad graf

- ▶ $n \leftarrow I$, markera som besökt
- ▶ Grannar: $\{E, J, F\}$
- ▶ E redan besökt \rightarrow Grannar: $\{\cancel{E}, J, F\}$
- ▶ J ej besökt \rightarrow anropa $\text{Traverse-depth-first}(J, g)$.
- ▶ J färdig \rightarrow Grannar: $\{\cancel{E}, \cancel{J}, F\}$
- ▶ F besökt \rightarrow Grannar: $\{\cancel{E}, \cancel{J}, \cancel{F}\}$



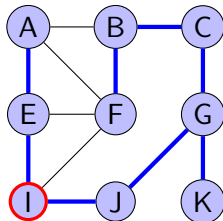
$(n=I, \{\cancel{E}, \cancel{J}, \cancel{F}\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(I, g)$ för oriktad graf

- ▶ $n \leftarrow I$, markera som besökt
- ▶ Grannar: $\{E, J, F\}$
- ▶ E redan besökt \rightarrow Grannar: $\{\cancel{E}, J, F\}$
- ▶ J ej besökt \rightarrow anropa $\text{Traverse-depth-first}(J, g)$.
- ▶ J färdig \rightarrow Grannar: $\{\cancel{E}, \cancel{J}, F\}$
- ▶ F besökt \rightarrow Grannar: $\{\cancel{E}, \cancel{J}, \cancel{F}\}$
- ▶ Färdig med I, återvänd



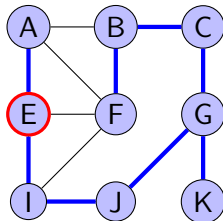
$(n=I, \{\cancel{E}, \cancel{J}, \cancel{F}\})$

$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(E, g)$ för oriktad graf

- ▶ $n \leftarrow E$, markera som besökt
- ▶ Grannar: $\{I, F, A\}$
- ▶ I ej besökt \rightarrow anropa $\text{Traverse-depth-first}(I, g)$.

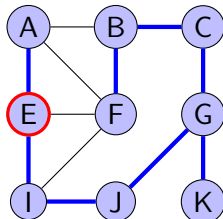


$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(E, g)$ för oriktad graf

- ▶ $n \leftarrow E$, markera som besökt
- ▶ Grannar: $\{I, F, A\}$
- ▶ I ej besökt \rightarrow anropa $\text{Traverse-depth-first}(I, g)$.
- ▶ I färdig \rightarrow Grannar: $\{I, F, A\}$

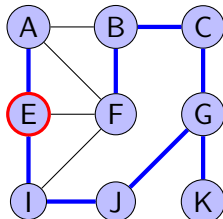


$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(E, g)$ för oriktad graf

- ▶ $n \leftarrow E$, markera som besökt
- ▶ Grannar: $\{I, F, A\}$
- ▶ I ej besökt \rightarrow anropa $\text{Traverse-depth-first}(I, g)$.
- ▶ I färdig \rightarrow Grannar: $\{I, F, A\}$
- ▶ F besökt \rightarrow Grannar: $\{I, \cancel{F}, A\}$

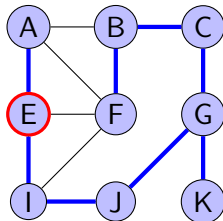


$(n=E, \{I, \cancel{F}, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(E, g)$ för oriktad graf

- ▶ $n \leftarrow E$, markera som besökt
- ▶ Grannar: $\{I, F, A\}$
- ▶ I ej besökt \rightarrow anropa $\text{Traverse-depth-first}(I, g)$.
- ▶ I färdig \rightarrow Grannar: $\{I, F, A\}$
- ▶ F besökt \rightarrow Grannar: $\{I, F, A\}$
- ▶ A besökt \rightarrow Grannar: $\{I, F, A\}$

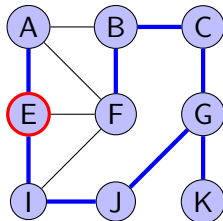


$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(E, g)$ för oriktad graf

- ▶ $n \leftarrow E$, markera som besökt
- ▶ Grannar: $\{I, F, A\}$
- ▶ I ej besökt \rightarrow anropa $\text{Traverse-depth-first}(I, g)$.
- ▶ I färdig \rightarrow Grannar: $\{I, F, A\}$
- ▶ F besökt \rightarrow Grannar: $\{I, F, A\}$
- ▶ A besökt \rightarrow Grannar: $\{I, F, A\}$
- ▶ Färdig med E, återvänd

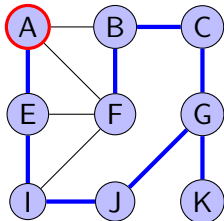


$(n=E, \{I, F, A\})$

$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(A, g)$ för oriktad graf

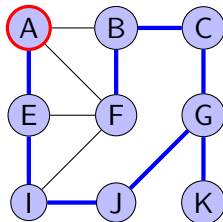
- ▶ $n \leftarrow A$, markera som besökt
- ▶ Grannar: $\{E, F, B\}$
- ▶ E ej besökt \rightarrow anropa $\text{Traverse-depth-first}(E, g)$.



$(n=A, \{E, F, B\})$

$g = \text{Traverse-depth-first}(A, g)$ för oriktad graf

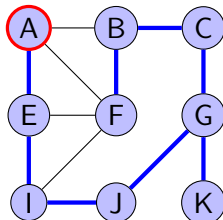
- ▶ $n \leftarrow A$, markera som besökt
- ▶ Grannar: $\{E, F, B\}$
- ▶ E ej besökt \rightarrow anropa $\text{Traverse-depth-first}(E, g)$.
- ▶ E färdig \rightarrow Grannar: $\{\cancel{E}, F, B\}$



$(n=A, \{\cancel{E}, F, B\})$

$g = \text{Traverse-depth-first}(A, g)$ för oriktad graf

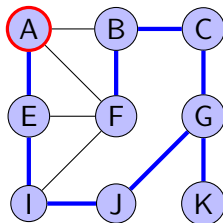
- ▶ $n \leftarrow A$, markera som besökt
- ▶ Grannar: $\{E, F, B\}$
- ▶ E ej besökt \rightarrow anropa $\text{Traverse-depth-first}(E, g)$.
- ▶ E färdig \rightarrow Grannar: $\{\cancel{E}, F, B\}$
- ▶ F besökt \rightarrow Grannar: $\{\cancel{E}, \cancel{F}, B\}$



$(n=A, \{\cancel{E}, \cancel{F}, B\})$

$g = \text{Traverse-depth-first}(A, g)$ för oriktad graf

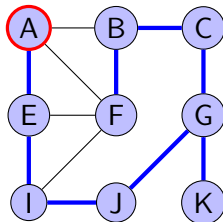
- ▶ $n \leftarrow A$, markera som besökt
- ▶ Grannar: $\{E, F, B\}$
- ▶ E ej besökt \rightarrow anropa $\text{Traverse-depth-first}(E, g)$.
- ▶ E färdig \rightarrow Grannar: $\{\cancel{E}, F, B\}$
- ▶ F besökt \rightarrow Grannar: $\{\cancel{E}, \cancel{F}, B\}$
- ▶ B besökt \rightarrow Grannar: $\{\cancel{E}, \cancel{F}, \cancel{B}\}$



$(n=A, \{\cancel{E}, \cancel{F}, \cancel{B}\})$

$g = \text{Traverse-depth-first}(A, g)$ för oriktad graf

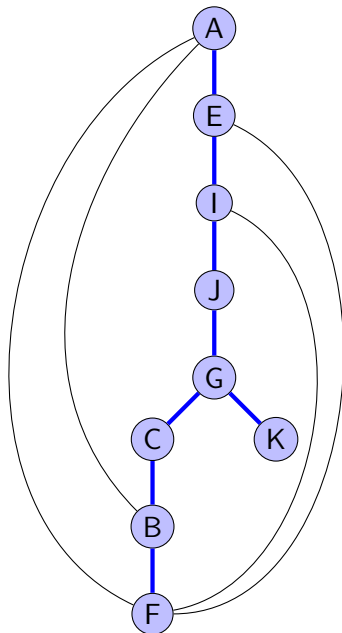
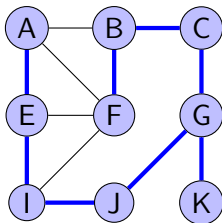
- ▶ $n \leftarrow A$, markera som besökt
- ▶ Grannar: $\{E, F, B\}$
- ▶ E ej besökt \rightarrow anropa $\text{Traverse-depth-first}(E, g)$.
- ▶ E färdig \rightarrow Grannar: $\{\cancel{E}, F, B\}$
- ▶ F besökt \rightarrow Grannar: $\{\cancel{E}, \cancel{F}, B\}$
- ▶ B besökt \rightarrow Grannar: $\{\cancel{E}, \cancel{F}, \cancel{B}\}$
- ▶ Färdig med A, återvänd



$(n=A, \{\cancel{E}, \cancel{F}, \cancel{B}\})$

Klart!

- Vi fick ett uppspännande träd



Tidskomplexitet för Bredden-först, djupet-först-traversering

- ▶ Låt grafen ha n noder och m bågar
- ▶ Varje nod besöks exakt en gång
 - ▶ Den nodrelaterade kostnaden: $O(n)$
- ▶ För varje nod undersöker man **alla bågar** till grannarna
 - ▶ Kostnaden att hitta grannarna varierar:
 - ▶ Mängdorienterad specifikation:
 - ▶ $O(m)$ per nod
 - ▶ Totalt: $O(mn)$ för alla bågar
 - ▶ Navigeringsorienterade specifikation:
 - ▶ $O(\text{grad}(v))$ per nod
 - ▶ Totalt: $O(\sum_v \text{grad}(v)) = O(m)$ för alla bågar
- ▶ Total komplexitet:
 - ▶ Mängdorienterad: $O(n) + O(mn) = O(mn)$
 - ▶ Navigeringsorienterad: $O(n) + O(m) = O(m + n)$

Blank

2. Kortaste-vägen-algoritmer

Kortaste vägen

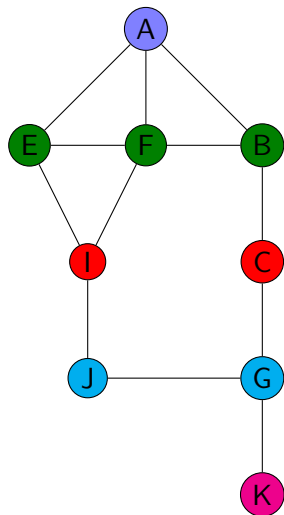
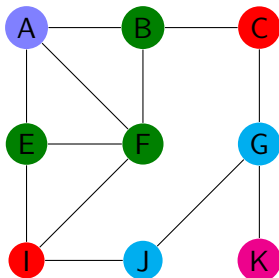
- ▶ Om grafen har **lika vikt** på alla bågar kan bredden-först-traversering användas för att beräkna kortaste vägen från en nod till alla andra noder
- ▶ Krävs minimal modifiering av algoritmen:
 - ▶ Lägg till ett attribut **avstånd** (*distance*) till varje nod
 - ▶ Avståndet från startnoden n till sig själv är 0
 - ▶ Kostnaden att gå från en nod p till sin granne är 1

Kortaste-vägen-algoritm vid lika vikt

```
Algorithm Shortest-path-eq-weight (n: Node, g: Graph)  
  // Input: A node n in a graph g to be traversed  
  // Output: The modified graph after traversal  
  
  q ← Enqueue (n, Queue-empty())  
  (n, g) ← Set-seen (n, g)  
  // Distance to start node is zero  
  (n, g) ← Set-distance (n, g, 0)  
  while not Isempty (q) do  
    n ← Front (q)  
    q ← Dequeue (q)  
    neighbour-set ← Neighbours (n, g)  
    for each neighbour b in neighbour-set do  
      if not Is-seen (b, g) then  
        (b, g) ← Set-seen (b, g)  
        // Compute and set distance to new node  
        d ← Get-distance (n, g) + 1  
        (b, g) ← Set-distance (b, g, d)  
        q ← Enqueue (b, q)  
  return g
```

Kortaste vägen vid lika vikt/kostnad

- Startnod = A.



Kortaste vägen-algoritmer

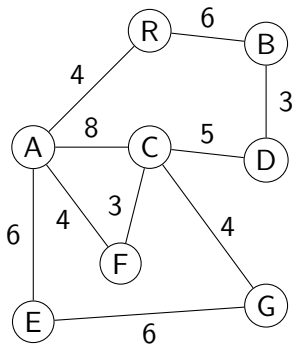
- ▶ Bredden-först-traversering ger oss **längden** på vägen från utgångsnoden till alla andra
 - ▶ Kan även ge **vägen** om vi sparar den
 - ▶ Om vikterna lika får vi **kortaste** vägen
- ▶ För **olika** vikter ska vi titta på två algoritmer:
 - ▶ Floyd
 - ▶ Matrisorienterad
 - ▶ Alla-till-alla-avstånd
 - ▶ Dijkstra
 - ▶ Graforienterad, använder prioritetskö
 - ▶ En-till-alla-avstånd

Floyd's shortest path

Floyd's shortest path

- ▶ Bygger på **matrisrepresentation** M av grafen.
- ▶ Vid starten innehåller M de **direkta** avstånden mellan noderna
 - ▶ Avståndet till sig själv är 0
 - ▶ Saknas både används ∞

	A	B	C	D	E	F	G	R
A	0	∞	8	∞	6	4	∞	4
B	∞	0	∞	3	∞	∞	∞	6
C	8	∞	0	5	∞	3	4	∞
D	∞	3	5	0	∞	∞	∞	∞
E	6	∞	∞	∞	0	∞	6	∞
F	4	∞	3	∞	∞	0	∞	∞
G	∞	∞	4	∞	6	∞	0	∞
R	4	6	∞	∞	∞	∞	∞	0



Floyds shortest path, algorithm

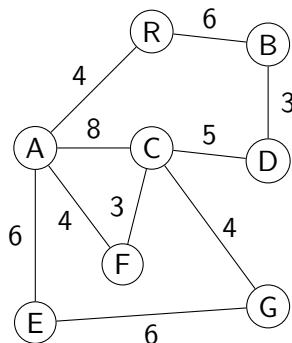
```
Algorithm Floyd-shortest-distance (g: Graph)  
// Input: A graph g to find shortest paths in  
  
// Get matrix representation of the graph  
M  $\leftarrow$  Get-matrix-representation(g)  
n  $\leftarrow$  Get-number-of-nodes(g)  
for k=1 to n do  
  for i=1 to n do  
    for j=1 to n do  
      if M(i,j) > M(i,k) + M(k,j) then  
        // We found a shorter path from i to j  
        M(i,j) = M(i,k) + M(k,j)
```

- ▶ $M(i,j)$ innehåller kortaste avståndet **hittills** mellan i och j
- ▶ $M(i,k) + M(k,j)$ är avståndet mellan i och j **via** k
- ▶ Vid slut innehåller $M(i,j)$ kortaste avståndet mellan i och j via **alla noder**

Floyds shortest path, exempel 1

► Vid starten

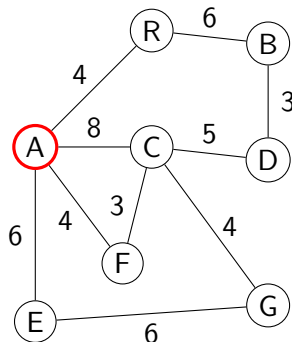
	A	B	C	D	E	F	G	R
A	0	∞	8	∞	6	4	∞	4
B	∞	0	∞	3	∞	∞	∞	6
C	8	∞	0	5	∞	3	4	∞
D	∞	3	5	0	∞	∞	∞	∞
E	6	∞	∞	∞	0	∞	6	∞
F	4	∞	3	∞	∞	0	∞	∞
G	∞	∞	4	∞	6	∞	0	∞
R	4	6	∞	∞	∞	∞	∞	0



Floyds shortest path, exempel 1

- Efter $k=1$ (vägar via A)

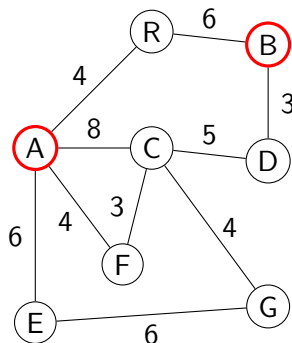
	A	B	C	D	E	F	G	R
A	0	∞	8	∞	6	4	∞	4
B	∞	0	∞	3	∞	∞	∞	6
C	8	∞	0	5	14	3	4	12
D	∞	3	5	0	∞	∞	∞	∞
E	6	∞	14	∞	0	10	6	10
F	4	∞	3	∞	10	0	∞	8
G	∞	∞	4	∞	6	∞	0	∞
R	4	6	12	∞	10	8	∞	0



Floyds shortest path, exempel 1

- Efter $k=2$ (vägar via B)

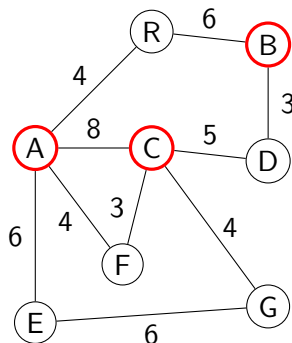
	A	B	C	D	E	F	G	R
A	0	∞	8	∞	6	4	∞	4
B	∞	0	∞	3	∞	∞	∞	6
C	8	∞	0	5	14	3	4	12
D	∞	3	5	0	∞	∞	∞	9
E	6	∞	14	∞	0	10	6	10
F	4	∞	3	∞	10	0	∞	8
G	∞	∞	4	∞	6	∞	0	∞
R	4	6	12	9	10	8	∞	0



Floyds shortest path, exempel 1

- Efter $k=3$ (vägar via C)

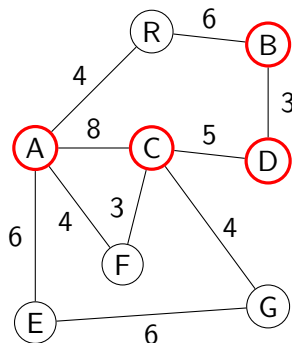
	A	B	C	D	E	F	G	R
A	0	∞	8	13	6	4	12	4
B	∞	0	∞	3	∞	∞	∞	6
C	8	∞	0	5	14	3	4	12
D	13	3	5	0	19	8	9	9
E	6	∞	14	19	0	10	6	10
F	4	∞	3	8	10	0	7	8
G	12	∞	4	9	6	7	0	16
R	4	6	12	9	10	8	16	0



Floyds shortest path, exempel 1

- Efter $k=4$ (vägar via D)

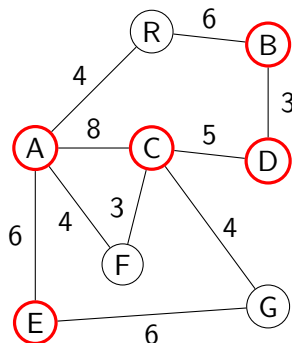
	A	B	C	D	E	F	G	R
A	0	16	8	13	6	4	12	4
B	16	0	8	3	22	11	12	6
C	8	8	0	5	14	3	4	12
D	13	3	5	0	19	8	9	9
E	6	22	14	19	0	10	6	10
F	4	11	3	8	10	0	7	8
G	12	12	4	9	6	7	0	16
R	4	6	12	9	10	8	16	0



Floyds shortest path, exempel 1

- Efter $k=5$ (vägar via E)

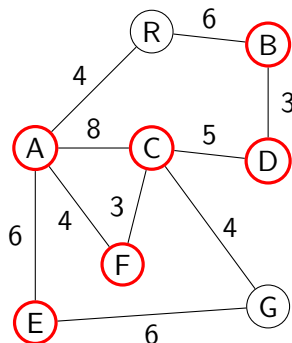
	A	B	C	D	E	F	G	R
A	0	16	8	13	6	4	12	4
B	16	0	8	3	22	11	12	6
C	8	8	0	5	14	3	4	12
D	13	3	5	0	19	8	9	9
E	6	22	14	19	0	10	6	10
F	4	11	3	8	10	0	7	8
G	12	12	4	9	6	7	0	16
R	4	6	12	9	10	8	16	0



Floyds shortest path, exempel 1

- Efter $k=6$ (vägar via F)

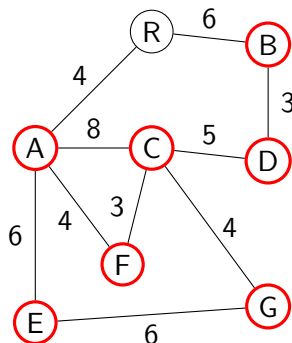
	A	B	C	D	E	F	G	R
A	0	16 15	8 7	13 12	6	4	12 11	4
B	16 15	0	8	3	22 21	11	12	6
C	8 7	8	0	5	14 13	3	4	12 11
D	13 12	3	5	0	19 18	8	9	9
E	6	22 21	14 13	19 18	0	10	6	10
F	4	11	3	8	10	0	7	8
G	12 11	12	4	9	6	7	0	16 15
R	4	6	12 11	9	10	8	16 15	0



Floyds shortest path, exempel 1

- Efter $k=7$ (vägar via G)

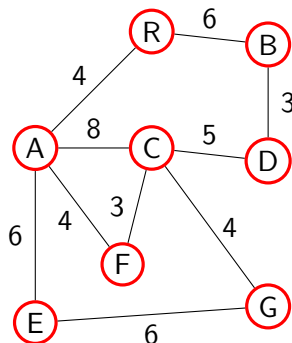
	A	B	C	D	E	F	G	R
A	0	15	7	12	6	4	11	4
B	15	0	8	3	²¹ 18	11	12	6
C	7	8	0	5	¹³ 10	3	4	11
D	12	3	5	0	¹⁸ 15	8	9	9
E	6	²¹ 18	¹³ 10	¹⁸ 15	0	10	6	10
F	4	11	3	8	10	0	7	8
G	11	12	4	9	6	7	0	15
R	4	6	11	9	10	8	15	0



Floyds shortest path, exempel 1

- Efter $k=8$ (vägar via R)

	A	B	C	D	E	F	G	R
A	0	¹⁵ ₁₀	7	12	6	4	11	4
B	¹⁵ ₁₀	0	8	3	¹⁸ ₁₆	11	12	6
C	7	8	0	5	10	3	4	11
D	12	3	5	0	15	8	9	9
E	6	¹⁸ ₁₆	10	15	0	10	6	10
F	4	11	3	8	10	0	7	8
G	11	12	4	9	6	7	0	15
R	4	6	11	9	10	8	15	0



Floyd, komplexitet

```
Algorithm Floyd-shortest-distance(g: Graph)  
// Input: A graph g to find shortest paths in  
  
// Get matrix representation of the graph  
M  $\leftarrow$  Get-matrix-representation(g)  
n  $\leftarrow$  Get-number-of-nodes(g)  
for k=1 to n do  
    for i=1 to n do  
        for j=1 to n do  
            if M(i, j) > M(i, k) + M(k, j) then  
                // We found a shorter path from i to j  
                M(i, j) = M(i, k) + M(k, j)
```

► Komplexitet?

Floyd, komplexitet

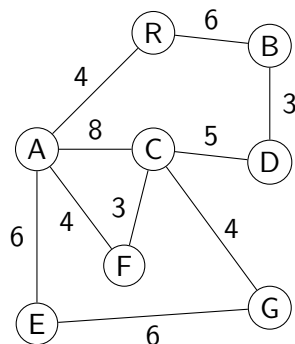
```
Algorithm Floyd-shortest-distance (g: Graph)  
// Input: A graph g to find shortest paths in  
  
// Get matrix representation of the graph  
M  $\leftarrow$  Get-matrix-representation (g)  
n  $\leftarrow$  Get-number-of-nodes (g)  
for k=1 to n do  
    for i=1 to n do  
        for j=1 to n do  
            if M(i, j) > M(i, k) + M(k, j) then  
                // We found a shorter path from i to j  
                M(i, j) = M(i, k) + M(k, j)
```

- Komplexitet?
- Trippel-loop: $O(n^3)$

Floyds shortest path, hitta kortaste vägen

- ▶ M innehåller kortaste **avstånden** men hur få tag på **vägen**?
- ▶ Modifiera algoritmen till att spara en **föregångarmatris**.

	A	B	C	D	E	F	G	R
A	0	10	7	12	6	4	11	4
B	10	0	8	3	16	11	12	6
C	7	8	0	5	10	3	4	11
D	12	3	5	0	15	8	9	9
E	6	16	10	15	0	10	6	10
F	4	11	3	8	10	0	7	8
G	11	12	4	9	6	7	0	15
R	4	6	11	9	10	8	15	0

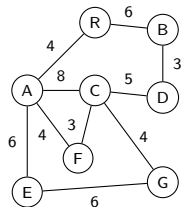


Floyds algorithm, modifierad

```
Algorithm Floyd-shortest-path(g: Graph)  
// Input: A graph g to find shortest paths in  
M ← Get-matrix-representation(g)  
n ← Get-number-of-nodes(g)  
// Set up the initial path matrix  
for i=1 to n do  
  for j=1 to n do  
    if i==j or M(i,j)==inf then  
      // No direct path from i to j  
      Path(i,j) = -1  
    else  
      // We came to node j from node i  
      Path(i,j) = i  
for k=1 to n do  
  for i=1 to n do  
    for j=1 to n do  
      if M(i,j) > M(i,k) + M(k,j) then  
        // Remember the new distance...  
        M(i,j) = M(i,k) + M(k,j)  
        // ...and how we came to j  
        Path(i,j) = Path(k,j)
```

Floyds shortest path, exempel 2

► Efter initiering

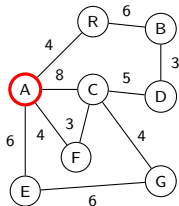


	A	B	C	D	E	F	G	R
A	0	∞	8	∞	6	4	∞	4
B	∞	0	∞	3	∞	∞	∞	6
C	8	∞	0	5	∞	3	4	∞
D	∞	3	5	0	∞	∞	∞	∞
E	6	∞	∞	∞	0	∞	6	∞
F	4	∞	3	∞	∞	0	∞	∞
G	∞	∞	4	∞	6	∞	0	∞
R	4	6	∞	∞	∞	∞	∞	0

A	B	C	D	E	F	G	R
—	—	A	—	A	A	—	A
—	—	—	B	—	—	—	B
C	—	—	C	—	C	C	—
—	D	D	—	—	—	—	—
E	—	—	—	—	—	E	—
F	—	F	—	—	—	—	—
—	—	G	—	G	—	—	—
R	R	—	—	—	—	—	—

Floyds shortest path, exempel 2

- Efter $k=1$ (vägar via A)

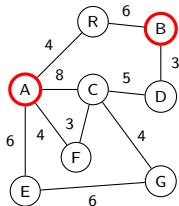


	A	B	C	D	E	F	G	R
A	0	∞	8	∞	6	4	∞	4
B	∞	0	∞	3	∞	∞	∞	6
C	8	∞	0	5	14	3	4	12
D	∞	3	5	0	∞	∞	∞	∞
E	6	∞	14	∞	0	10	6	10
F	4	∞	3	∞	10	0	∞	8
G	∞	∞	4	∞	6	∞	0	∞
R	4	6	12	∞	10	8	∞	0

	A	B	C	D	E	F	G	R
A	—	—	A	—	A	A	—	A
B	—	—	—	B	—	—	—	B
C	C	—	—	C	A	C	C	A
D	—	D	D	—	—	—	—	—
E	E	—	A	—	—	A	E	A
F	F	—	F	—	A	—	—	A
G	—	—	G	—	G	—	—	—
R	R	R	A	—	A	A	—	—

Floyds shortest path, exempel 2

- Efter $k=2$ (vägar via B)

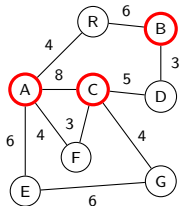


	A	B	C	D	E	F	G	R
A	0	∞	8	∞	6	4	∞	4
B	∞	0	∞	3	∞	∞	∞	6
C	8	∞	0	5	14	3	4	12
D	∞	3	5	0	∞	∞	∞	∞
E	6	∞	14	∞	0	10	6	10
F	4	∞	3	∞	10	0	∞	8
G	∞	∞	4	∞	6	∞	0	∞
R	4	6	12	∞	10	8	∞	0

A	B	C	D	E	F	G	R
—	—	A	—	A	A	—	A
—	—	—	B	—	—	—	B
C	—	—	C	A	C	C	A
—	D	D	—	—	—	—	B
E	—	A	—	—	A	E	A
F	—	F	—	A	—	—	A
—	—	G	—	G	—	—	—
R	R	A	B	A	A	—	—

Floyds shortest path, exempel 2

- Efter $k=3$ (vägar via C)

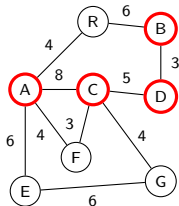


	A	B	C	D	E	F	G	R
A	0	∞	8	13	6	4	12	4
B	∞	0	∞	3	∞	∞	∞	6
C	8	∞	0	5	14	3	4	12
D	13	3	5	0	19	8	9	9
E	6	∞	14	19	0	10	6	10
F	4	∞	3	8	10	0	7	8
G	12	∞	4	9	6	7	0	16
R	4	6	12	9	10	8	16	0

A	B	C	D	E	F	G	R
—	—	A	C	A	A	C	A
—	—	—	B	—	—	—	B
C	—	—	C	A	C	C	A
C	D	D	—	A	C	C	B
E	—	A	C	—	A	E	A
F	—	F	C	A	—	C	A
C	—	G	C	G	C	—	A
R	R	A	B	A	A	C	—

Floyds shortest path, exempel 2

- Efter $k=4$ (vägar via D)

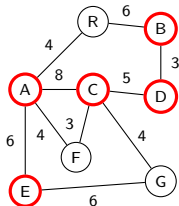


	A	B	C	D	E	F	G	R
A	0	16	8	13	6	4	12	4
B	16	0	8	3	22	11	12	6
C	8	8	0	5	14	3	4	12
D	13	3	5	0	19	8	9	9
E	6	22	14	19	0	10	6	10
F	4	11	3	8	10	0	7	8
G	12	12	4	9	6	7	0	16
R	4	6	12	9	10	8	16	0

A	B	C	D	E	F	G	R
—	D	A	C	A	A	C	A
C	—	D	B	A	C	C	B
C	D	—	C	A	C	C	A
C	D	D	—	A	C	C	B
E	D	A	C	—	A	E	A
F	D	F	C	A	—	C	A
C	D	G	C	G	C	—	A
R	R	A	B	A	A	C	—

Floyds shortest path, exempel 2

- Efter $k=5$ (vägar via E)

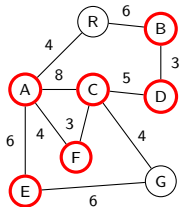


	A	B	C	D	E	F	G	R
A	0	16	8	13	6	4	12	4
B	16	0	8	3	22	11	12	6
C	8	8	0	5	14	3	4	12
D	13	3	5	0	19	8	9	9
E	6	22	14	19	0	10	6	10
F	4	11	3	8	10	0	7	8
G	12	12	4	9	6	7	0	16
R	4	6	12	9	10	8	16	0

	A	B	C	D	E	F	G	R
A	—	D	A	C	A	A	C	A
B	C	—	D	B	A	C	C	B
C	C	D	—	C	A	C	C	A
D	C	D	D	—	A	C	C	B
E	E	D	A	C	—	A	E	A
F	F	D	F	C	A	—	C	A
G	C	D	G	C	G	C	—	A
R	R	R	A	B	A	A	C	—

Floyds shortest path, exempel 2

- Efter $k=6$ (vägar via F)

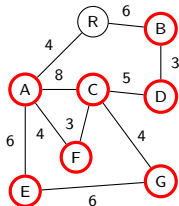


	A	B	C	D	E	F	G	R
A	0	¹⁶ 15	⁸ 7	¹³ 12	6	4	¹² 11	4
B	¹⁶ 15	0	8	3	²² 21	11	12	6
C	⁸ 7	8	0	5	¹⁴ 13	3	4	¹² 11
D	¹³ 12	3	5	0	¹⁹ 18	8	9	9
E	6	²² 21	¹⁴ 13	¹⁹ 18	0	10	6	10
F	4	11	3	8	10	0	7	8
G	¹² 11	12	4	9	6	7	0	¹⁶ 15
R	4	6	¹² 11	9	10	8	¹⁶ 15	0

	A	B	C	D	E	F	G	R
A	—	D	^A F	C	A	A	C	A
B	^C F	—	D	B	A	C	C	B
C	^C F	D	—	C	A	C	C	A
D	^C F	D	D	—	A	C	C	B
E	E	D	^A F	C	—	A	E	A
F	F	D	F	C	A	—	C	A
G	^C F	D	G	C	G	C	—	A
R	R	R	^A F	B	A	A	C	—

Floyds shortest path, exempel 2

- Efter $k=7$ (vägar via G)

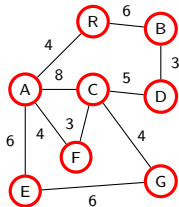


	A	B	C	D	E	F	G	R
A	0	15	7	12	6	4	11	4
B	15	0	8	3	21 18	11	12	6
C	7	8	0	5	13 10	3	4	11
D	12	3	5	0	18 15	8	9	9
E	6	21 18	13 10	18 15	0	10	6	10
F	4	11	3	8	10	0	7	8
G	11	12	4	9	6	7	0	15
R	4	6	11	9	10	8	15	0

A	B	C	D	E	F	G	R
—	D	F	C	A	A	C	A
F	—	D	B	A G	C	C	B
F	D	—	C	A G	C	C	A
F	D	D	—	A G	C	C	B
E	D	F G	C	—	A	E	A
F	D	F	C	A	—	C	A
F	D	G	C	G	C	—	A
R	R	F	B	A	A	C	—

Floyds shortest path, exempel 2

- Efter $k=8$ (vägar via R)

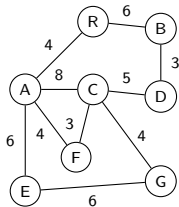


	A	B	C	D	E	F	G	R
A	0	¹⁵ ₁₀	7	12	6	4	11	4
B	¹⁵ ₁₀	0	8	3	¹⁸ ₁₆	11	12	6
C	7	8	0	5	10	3	4	11
D	12	3	5	0	15	8	9	9
E	6	¹⁸ ₁₆	10	15	0	10	6	10
F	4	11	3	8	10	0	7	8
G	11	12	4	9	6	7	0	15
R	4	6	11	9	10	8	15	0

A	B	C	D	E	F	G	R
—	^D _R	F	C	A	A	C	A
^F _R	—	D	B	^G _A	C	C	B
F	D	—	C	G	C	C	A
F	D	D	—	G	C	C	B
E	^D _R	G	C	—	A	E	A
F	D	F	C	A	—	C	A
F	D	G	C	G	C	—	A
R	R	F	B	A	A	C	—

Floyds shortest path, hitta kortaste vägen

- Vad är kortaste vägen mellan A och C?

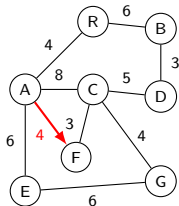


	A	B	C	D	E	F	G	R
A	0	10	7	12	6	4	11	4
B	10	0	8	3	16	11	12	6
C	7	8	0	5	10	3	4	11
D	12	3	5	0	15	8	9	9
E	6	16	10	15	0	10	6	10
F	4	11	3	8	10	0	7	8
G	11	12	4	9	6	7	0	15
R	4	6	11	9	10	8	15	0

A	B	C	D	E	F	G	R
-	R	F	C	A	A	C	A
R	-	D	B	A	C	C	B
F	D	-	C	G	C	C	A
F	D	D	-	G	C	C	B
E	R	G	C	-	A	E	A
F	D	F	C	A	-	C	A
F	D	G	C	G	C	-	A
R	R	F	B	A	A	C	-

Floyds shortest path, hitta kortaste vägen

- Vad är kortaste vägen mellan A och C?

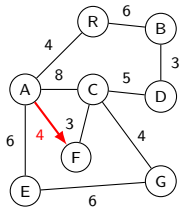


	A	B	C	D	E	F	G	R
A	0	10	7	12	6	4	11	4
B	10	0	8	3	16	11	12	6
C	7	8	0	5	10	3	4	11
D	12	3	5	0	15	8	9	9
E	6	16	10	15	0	10	6	10
F	4	11	3	8	10	0	7	8
G	11	12	4	9	6	7	0	15
R	4	6	11	9	10	8	15	0

A	B	C	D	E	F	G	R
-	R	F	C	A	A	C	A
R	-	D	B	A	C	C	B
F	D	-	C	G	C	C	A
F	D	D	-	G	C	C	B
E	R	G	C	-	A	E	A
F	D	F	C	A	-	C	A
F	D	G	C	G	C	-	A
R	R	F	B	A	A	C	-

Floyds shortest path, hitta kortaste vägen

- Vad är kortaste vägen mellan A och C?

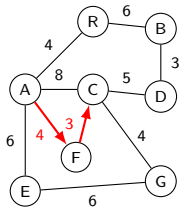


	A	B	C	D	E	F	G	R
A	0	10	7	12	6	4	11	4
B	10	0	8	3	16	11	12	6
C	7	8	0	5	10	3	4	11
D	12	3	5	0	15	8	9	9
E	6	16	10	15	0	10	6	10
F	4	11	3	8	10	0	7	8
G	11	12	4	9	6	7	0	15
R	4	6	11	9	10	8	15	0

A	B	C	D	E	F	G	R
-	R	F	C	A	A	C	A
R	-	D	B	A	C	C	B
F	D	-	C	G	C	C	A
F	D	D	-	G	C	C	B
E	R	G	C	-	A	E	A
F	D	F	C	A	-	C	A
F	D	G	C	G	C	-	A
R	R	F	B	A	A	C	-

Floyds shortest path, hitta kortaste vägen

- Vad är kortaste vägen mellan A och C?

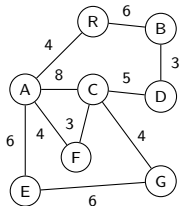


	A	B	C	D	E	F	G	R
A	0	10	7	12	6	4	11	4
B	10	0	8	3	16	11	12	6
C	7	8	0	5	10	3	4	11
D	12	3	5	0	15	8	9	9
E	6	16	10	15	0	10	6	10
F	4	11	3	8	10	0	7	8
G	11	12	4	9	6	7	0	15
R	4	6	11	9	10	8	15	0

	A	B	C	D	E	F	G	R
A	-	R	F	C	A	A	C	A
B	R	-	D	B	A	C	C	B
C	F	D	-	C	G	C	C	A
D	F	D	D	-	G	C	C	B
E	E	R	G	C	-	A	E	A
F	F	D	F	C	A	-	C	A
G	F	D	G	C	G	C	-	A
R	R	R	F	B	A	A	C	-

Floyds shortest path, hitta kortaste vägen

- Vad är kortaste vägen mellan R och G?

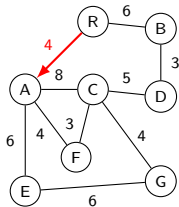


	A	B	C	D	E	F	G	R
A	0	10	7	12	6	4	11	4
B	10	0	8	3	16	11	12	6
C	7	8	0	5	10	3	4	11
D	12	3	5	0	15	8	9	9
E	6	16	10	15	0	10	6	10
F	4	11	3	8	10	0	7	8
G	11	12	4	9	6	7	0	15
R	4	6	11	9	10	8	15	0

A	B	C	D	E	F	G	R
-	R	F	C	A	A	C	A
R	-	D	B	A	C	C	B
F	D	-	C	G	C	C	A
F	D	D	-	G	C	C	B
E	R	G	C	-	A	E	A
F	D	F	C	A	-	C	A
F	D	G	C	G	C	-	A
R	R	F	B	A	A	C	-

Floyds shortest path, hitta kortaste vägen

- Vad är kortaste vägen mellan R och G?

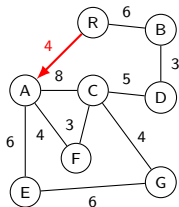


	A	B	C	D	E	F	G	R
A	0	10	7	12	6	4	11	4
B	10	0	8	3	16	11	12	6
C	7	8	0	5	10	3	4	11
D	12	3	5	0	15	8	9	9
E	6	16	10	15	0	10	6	10
F	4	11	3	8	10	0	7	8
G	11	12	4	9	6	7	0	15
R	4	6	11	9	10	8	15	0

A	B	C	D	E	F	G	R
-	R	F	C	A	A	C	A
R	-	D	B	A	C	C	B
F	D	-	C	G	C	C	A
F	D	D	-	G	C	C	B
E	R	G	C	-	A	E	A
F	D	F	C	A	-	C	A
F	D	G	C	G	C	-	A
R	R	F	B	A	A	C	-

Floyds shortest path, hitta kortaste vägen

- Vad är kortaste vägen mellan R och G?

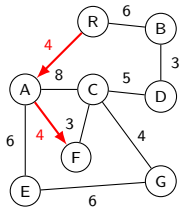


	A	B	C	D	E	F	G	R
A	0	10	7	12	6	4	11	4
B	10	0	8	3	16	11	12	6
C	7	8	0	5	10	3	4	11
D	12	3	5	0	15	8	9	9
E	6	16	10	15	0	10	6	10
F	4	11	3	8	10	0	7	8
G	11	12	4	9	6	7	0	15
R	4	6	11	9	10	8	15	0

A	B	C	D	E	F	G	R
-	R	F	C	A	A	C	A
R	-	D	B	A	C	C	B
F	D	-	C	G	C	C	A
F	D	D	-	G	C	C	B
E	R	G	C	-	A	E	A
F	D	F	C	A	-	C	A
F	D	G	C	G	C	-	A
R	R	F	B	A	A	C	-

Floyds shortest path, hitta kortaste vägen

- Vad är kortaste vägen mellan R och G?

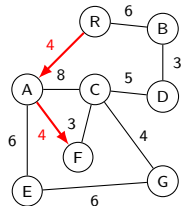


	A	B	C	D	E	F	G	R
A	0	10	7	12	6	4	11	4
B	10	0	8	3	16	11	12	6
C	7	8	0	5	10	3	4	11
D	12	3	5	0	15	8	9	9
E	6	16	10	15	0	10	6	10
F	4	11	3	8	10	0	7	8
G	11	12	4	9	6	7	0	15
R	4	6	11	9	10	8	15	0

A	B	C	D	E	F	G	R
-	R	F	C	A	A	C	A
R	-	D	B	A	C	C	B
F	D	-	C	G	C	C	A
F	D	D	-	G	C	C	B
E	R	G	C	-	A	E	A
F	D	F	C	A	-	C	A
F	D	G	C	G	C	-	A
R	R	F	B	A	A	C	-

Floyds shortest path, hitta kortaste vägen

- Vad är kortaste vägen mellan R och G?

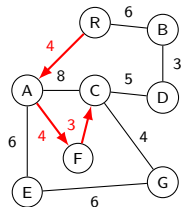


	A	B	C	D	E	F	G	R
A	0	10	7	12	6	4	11	4
B	10	0	8	3	16	11	12	6
C	7	8	0	5	10	3	4	11
D	12	3	5	0	15	8	9	9
E	6	16	10	15	0	10	6	10
F	4	11	3	8	10	0	7	8
G	11	12	4	9	6	7	0	15
R	4	6	11	9	10	8	15	0

A	B	C	D	E	F	G	R
-	R	F	C	A	A	C	A
R	-	D	B	A	C	C	B
F	D	-	C	G	C	C	A
F	D	D	-	G	C	C	B
E	R	G	C	-	A	E	A
F	D	F	C	A	-	C	A
F	D	G	C	G	C	-	A
R	R	F	B	A	A	C	-

Floyds shortest path, hitta kortaste vägen

- Vad är kortaste vägen mellan R och G?

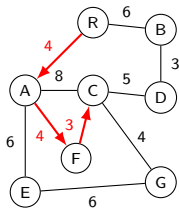


	A	B	C	D	E	F	G	R
A	0	10	7	12	6	4	11	4
B	10	0	8	3	16	11	12	6
C	7	8	0	5	10	3	4	11
D	12	3	5	0	15	8	9	9
E	6	16	10	15	0	10	6	10
F	4	11	3	8	10	0	7	8
G	11	12	4	9	6	7	0	15
R	4	6	11	9	10	8	15	0

A	B	C	D	E	F	G	R
-	R	F	C	A	A	C	A
R	-	D	B	A	C	C	B
F	D	-	C	G	C	C	A
F	D	D	-	G	C	C	B
E	R	G	C	-	A	E	A
F	D	F	C	A	-	C	A
F	D	G	C	G	C	-	A
R	R	F	B	A	A	C	-

Floyds shortest path, hitta kortaste vägen

- Vad är kortaste vägen mellan R och G?

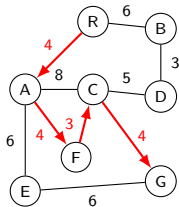


	A	B	C	D	E	F	G	R
A	0	10	7	12	6	4	11	4
B	10	0	8	3	16	11	12	6
C	7	8	0	5	10	3	4	11
D	12	3	5	0	15	8	9	9
E	6	16	10	15	0	10	6	10
F	4	11	3	8	10	0	7	8
G	11	12	4	9	6	7	0	15
R	4	6	11	9	10	8	15	0

A	B	C	D	E	F	G	R
-	R	F	C	A	A	C	A
R	-	D	B	A	C	C	B
F	D	-	C	G	C	C	A
F	D	D	-	G	C	C	B
E	R	G	C	-	A	E	A
F	D	F	C	A	-	C	A
F	D	G	C	G	C	-	A
R	R	F	B	A	A	C	-

Floyds shortest path, hitta kortaste vägen

- Vad är kortaste vägen mellan R och G?



	A	B	C	D	E	F	G	R
A	0	10	7	12	6	4	11	4
B	10	0	8	3	16	11	12	6
C	7	8	0	5	10	3	4	11
D	12	3	5	0	15	8	9	9
E	6	16	10	15	0	10	6	10
F	4	11	3	8	10	0	7	8
G	11	12	4	9	6	7	0	15
R	4	6	11	9	10	8	15	0

A	B	C	D	E	F	G	R
-	R	F	C	A	A	C	A
R	-	D	B	A	C	C	B
F	D	-	C	G	C	C	A
F	D	D	-	G	C	C	B
E	R	G	C	-	A	E	A
F	D	F	C	A	-	C	A
F	D	G	C	G	C	-	A
R	R	F	B	A	A	C	-

Blank

Dijkstra's shortest path

Dijkstras shortest path

- ▶ Söker kortaste vägen från **en** nod n till **alla andra** noder
 - ▶ Använder en **prioritetskö** av **obesökta** noder
- ▶ Fungerar enbart på grafer med **positiva** vikter
- ▶ Låt varje nod ha följande attribut:
 - ▶ Seen: Sann när vi hittat en väg till noden ("sett" noden)
 - ▶ Distance: Värdet på den **hittills** kortaste vägen fram till noden
 - ▶ Parent: Referens till **föregångaren** längs vägen

Dijkstras shortest path, algorithm

```
Algorithm Dijkstra-shortest-path(n: Node, g: Graph)
// Input: A graph g to find shortest path from node n

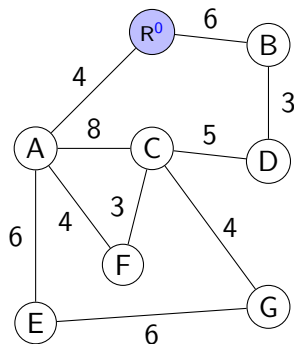
// Distance to start node is zero
n.distance  $\leftarrow$  0; n.seen  $\leftarrow$  True; n.parent  $\leftarrow$  NULL
// Initialize pqueue with start node
q  $\leftarrow$  Insert(n, Pqueue-empty())
while not Isempty(q)
    // Get node with shortest distance from queue
    n  $\leftarrow$  Inspect-first(q); q  $\leftarrow$  Delete-first(q)
    nd  $\leftarrow$  n.distance
    // ...and its neighbours
    neighbour-set  $\leftarrow$  Neighbours(n, g)
    for each neighbour b in neighbour-set do
        // Compute distance to b VIA n
        d  $\leftarrow$  nd + Get-weight(n, b, g)
        if not Is-seen(b, g) then
            // We've never seen b; this is the first path to arrive at b
            b.distance  $\leftarrow$  d
            b.seen  $\leftarrow$  True
            b.parent  $\leftarrow$  n
            // Add new node to pqueue
            q  $\leftarrow$  Insert(b, q)
        else if d < b.distance then
            // We've seen b before, but path via n is shorter
            b.distance  $\leftarrow$  d
            // Update how we came to b
            b.parent  $\leftarrow$  n
            // Update the pqueue based on the new distance
            q  $\leftarrow$  Update(b, q)
```

Dijkstras shortest path, visualisering

- ▶ Symboler:
 - ▶ Aktuell nod har röd ring
 - ▶ Sedda noder är ljusblåa
 - ▶ Nodens etikett har aktuellt avstånd som exponent
 - ▶ Noder i prioritetskön har grönstreckad ring
- ▶ Prioritetskön presenteras sorterad

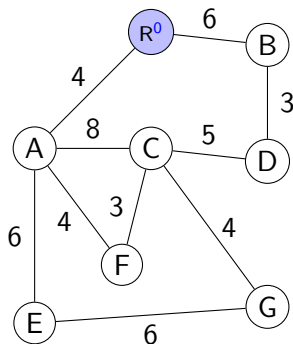
Dijkstras shortest path för oriktad graf

- ▶ R.seen = True
- ▶ R.distance = 0
- ▶ R.parent = NULL



Dijkstras shortest path för oriktad graf

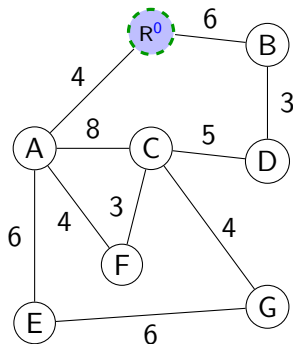
- ▶ $R.\text{seen} = \text{True}$
- ▶ $R.\text{distance} = 0$
- ▶ $R.\text{parent} = \text{NULL}$
- ▶ $q = \text{Insert}(R(T,0,-), \text{Pqueue-empty}())$



$$q = \{ R(T,0,-) \}$$

Dijkstras shortest path för oriktad graf

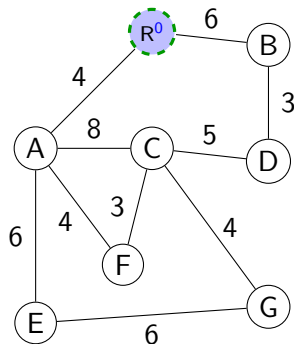
- ▶ $R.\text{seen} = \text{True}$
- ▶ $R.\text{distance} = 0$
- ▶ $R.\text{parent} = \text{NULL}$
- ▶ $q = \text{Insert}(R(T,0,-), \text{Pqueue-empty}())$
- ▶ while not $\text{Isempty}(q) \dots$



$$q = \{ R(T,0,-) \}$$

Dijkstras shortest path för oriktad graf

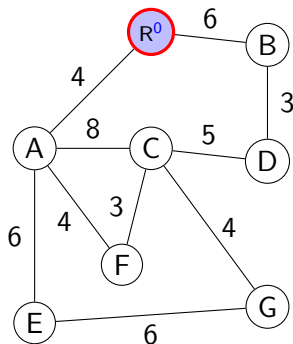
► while not lsempy(q)...



$$q = \{ R(T, 0, -) \}$$

Dijkstras shortest path för oriktad graf

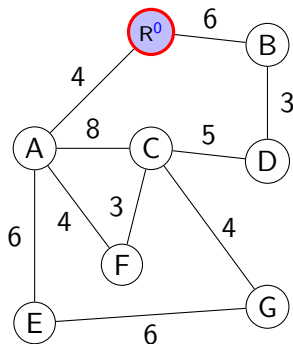
- ▶ while not lsempy(q)...
 - ▶ $n = R(T,0,-)$; $q = \text{Delete-first}(q)$
 - ▶ $n_d = n.\text{distance} = 0$



$$q = \{ \}$$

Dijkstras shortest path för oriktad graf

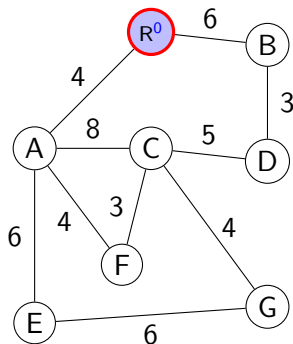
- ▶ while not lsempy(q)...
 - ▶ $n = R(T,0,-)$; $q = \text{Delete-first}(q)$
 - ▶ $n_d = n.\text{distance} = 0$
 - ▶ neighbour-set = {A,B}



$$q = \{ \}$$

Dijkstras shortest path för oriktad graf

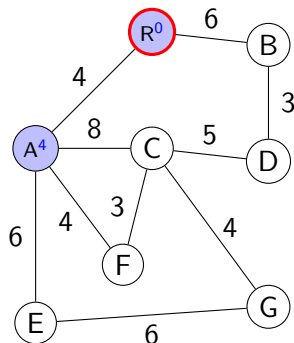
- ▶ while not lsempy(q)...
 - ▶ $n = R(T,0,-)$; $q = \text{Delete-first}(q)$
 - ▶ $n_d = n.\text{distance} = 0$
 - ▶ neighbour-set = {A,B}
 - ▶ A not seen



$$q = \{ \}$$

Dijkstras shortest path för oriktad graf

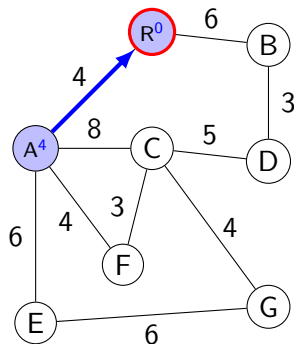
- ▶ while not lsempy(q)...
 - ▶ $n = R(T,0,-)$; $q = \text{Delete-first}(q)$
 - ▶ $n_d = n.\text{distance} = 0$
 - ▶ neighbour-set = {A,B}
 - ▶ A not seen
 - ▶ $d = n_d + \text{Get-weight}(n,A,g) = 4$
 - ▶ A.seen = True
 - ▶ A.distance = d



$$q = \{ \}$$

Dijkstras shortest path för oriktad graf

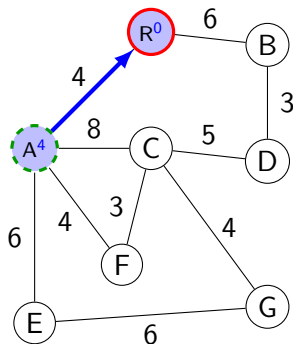
- ▶ while not lsempy(q)...
 - ▶ $n = R(T,0,-)$; $q = \text{Delete-first}(q)$
 - ▶ $n_d = n.\text{distance} = 0$
 - ▶ neighbour-set = {A,B}
 - ▶ A not seen
 - ▶ $d = n_d + \text{Get-weight}(n,A,g) = 4$
 - ▶ A.seen = True
 - ▶ A.distance = d
 - ▶ A.parent = R



$$q = \{ \}$$

Dijkstras shortest path för oriktad graf

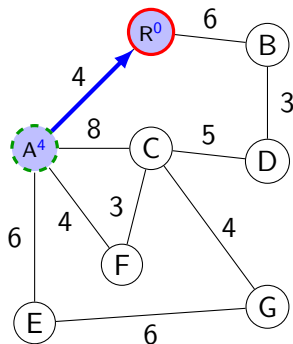
- ▶ while not lsempy(q)...
 - ▶ $n = R(T,0,-)$; $q = \text{Delete-first}(q)$
 - ▶ $n_d = n.\text{distance} = 0$
 - ▶ neighbour-set = {A,B}
 - ▶ A not seen
 - ▶ $d = n_d + \text{Get-weight}(n,A,g) = 4$
 - ▶ A.seen = True
 - ▶ A.distance = d
 - ▶ A.parent = R
 - ▶ $q = \text{Insert}(A(T,4,R),q)$



$$q = \{ A(T,4,R) \}$$

Dijkstras shortest path för oriktad graf

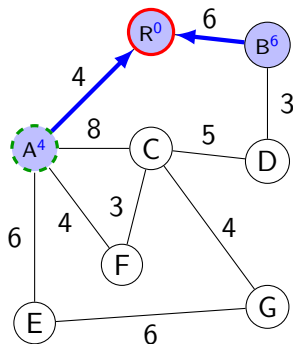
- ▶ while not lsempy(q)...
 - ▶ $n = R(T,0,-)$; $q = \text{Delete-first}(q)$
 - ▶ $n_d = n.\text{distance} = 0$
 - ▶ neighbour-set = $\{\cancel{A}, B\}$
 - ▶ B not seen



$$q = \{ A(T,4,R) \}$$

Dijkstras shortest path för oriktad graf

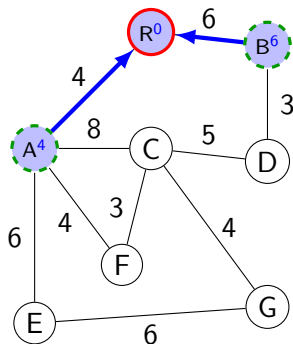
- ▶ while not lsempy(q)...
 - ▶ $n = R(T,0,-)$; $q = \text{Delete-first}(q)$
 - ▶ $n_d = n.\text{distance} = 0$
 - ▶ neighbour-set = $\{\cancel{A}, B\}$
 - ▶ B not seen
 - ▶ $d = n_d + \text{Get-weight}(n, B, g) = 6$
 - ▶ B.seen = True
 - ▶ B.distance = d
 - ▶ B.parent = R



$$q = \{ A(T,4,R) \}$$

Dijkstras shortest path för oriktad graf

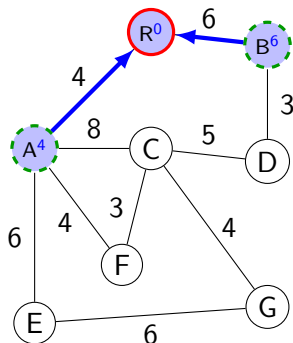
- ▶ while not lsempy(q)...
 - ▶ $n = R(T,0,-)$; $q = \text{Delete-first}(q)$
 - ▶ $n_d = n.\text{distance} = 0$
 - ▶ neighbour-set = $\{\cancel{A}, B\}$
 - ▶ B not seen
 - ▶ $d = n_d + \text{Get-weight}(n, B, g) = 6$
 - ▶ B.seen = True
 - ▶ B.distance = d
 - ▶ B.parent = R
 - ▶ $q = \text{Insert}(B(T,6,R), q)$



$$q = \{ A(T,4,R), B(T,6,R) \}$$

Dijkstras shortest path för oriktad graf

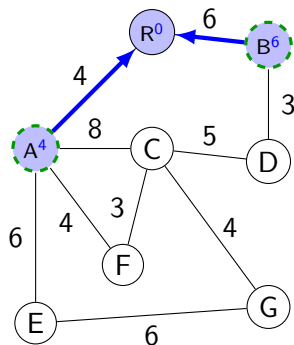
- ▶ while not lsempy(q)...
 - ▶ $n = R(T,0,-)$; $q = \text{Delete-first}(q)$
 - ▶ $n_d = n.\text{distance} = 0$
 - ▶ neighbour-set = $\{\cancel{A}, \cancel{B}\}$



$$q = \{ A(T,4,R), B(T,6,R) \}$$

Dijkstras shortest path för oriktad graf

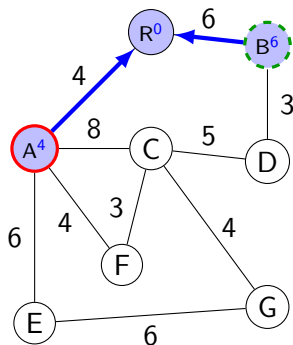
► while not lsempy(q)...



$$q = \{ A(T,4,R), B(T,6,R) \}$$

Dijkstras shortest path för oriktad graf

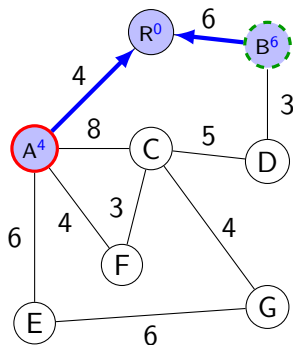
- ▶ while not lsempy(q)...
 - ▶ $n = A(T, 4, R)$; $q = \text{Delete-first}(q)$;



$$q = \{ B(T, 6, R) \}$$

Dijkstras shortest path för oriktad graf

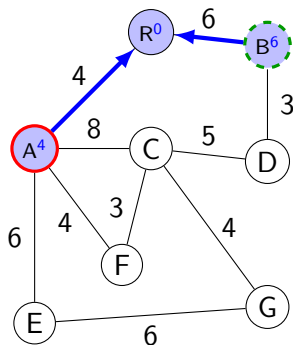
- ▶ while not lsempy(q)...
 - ▶ $n = A(T, 4, R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 4$;
 - ▶ neighbour-set = {E,R,F,C};



$$q = \{ B(T, 6, R) \}$$

Dijkstras shortest path för oriktad graf

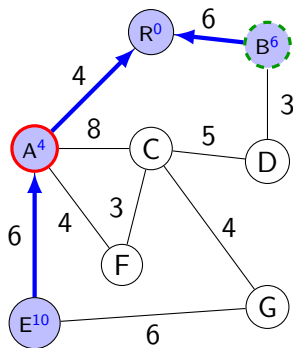
- ▶ while not lsempy(q)...
 - ▶ $n = A(T, 4, R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 4$;
 - ▶ neighbour-set = {E, R, F, C};
 - ▶ E not seen



$$q = \{ B(T, 6, R) \}$$

Dijkstras shortest path för oriktad graf

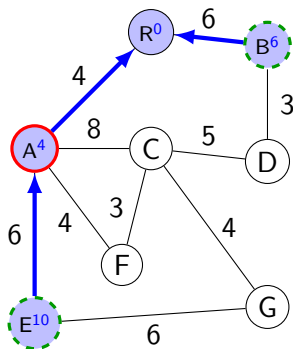
- ▶ while not lsempy(q)...
 - ▶ $n = A(T, 4, R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 4$;
 - ▶ neighbour-set = {E, R, F, C};
 - ▶ E not seen
 - ▶ $d = n_d + \text{Get-weight}(n, E, g) = 10$;
 - ▶ E.seen = True;
 - ▶ E.distance = d ;
 - ▶ E.parent = A;



$$q = \{ B(T, 6, R) \}$$

Dijkstras shortest path för oriktad graf

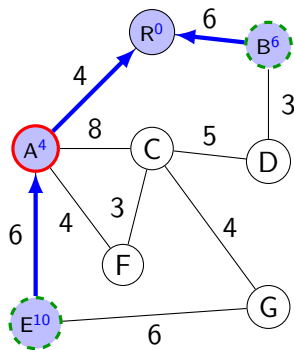
- ▶ while not lsempy(q)...
 - ▶ $n = A(T,4,R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 4$;
 - ▶ neighbour-set = {E,R,F,C};
 - ▶ E not seen
 - ▶ $d = n_d + \text{Get-weight}(n,E,g) = 10$;
 - ▶ E.seen = True;
 - ▶ E.distance = d ;
 - ▶ E.parent = A;
 - ▶ $q = \text{Insert}(E(T,10,A),q)$;



$$q = \{ B(T,6,R), E(T,10,A) \}$$

Dijkstras shortest path för oriktad graf

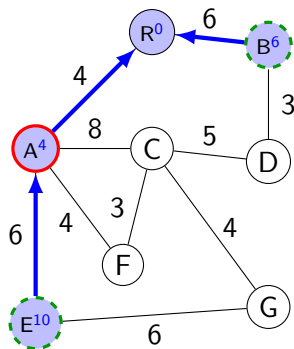
- ▶ while not lsempy(q)...
 - ▶ $n = A(T, 4, R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 4$;
 - ▶ neighbour-set = $\{\cancel{E}, R, F, C\}$;
 - ▶ R seen



$$q = \{ B(T, 6, R), E(T, 10, A) \}$$

Dijkstras shortest path för oriktad graf

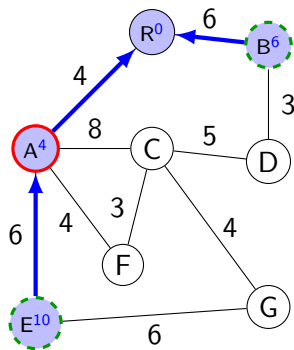
- ▶ while not lsempy(q)...
 - ▶ $n = A(T, 4, R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 4$;
 - ▶ neighbour-set = $\{\cancel{E}, R, F, C\}$;
 - ▶ R seen
 - ▶ $d = n_d + \text{Get-weight}(n, R, g) = 8$;
 - ▶ d **not** $< R.\text{distance}$



$$q = \{ B(T, 6, R), E(T, 10, A) \}$$

Dijkstras shortest path för oriktad graf

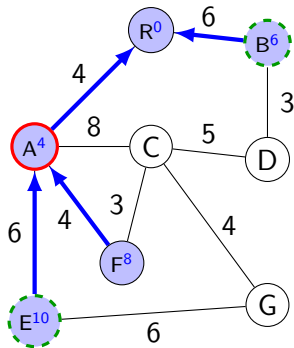
- ▶ while not lsempy(q)...
 - ▶ $n = A(T, 4, R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 4$;
 - ▶ neighbour-set = $\{\cancel{E}, \cancel{R}, F, C\}$;
 - ▶ F not seen



$$q = \{ B(T, 6, R), E(T, 10, A) \}$$

Dijkstras shortest path för oriktad graf

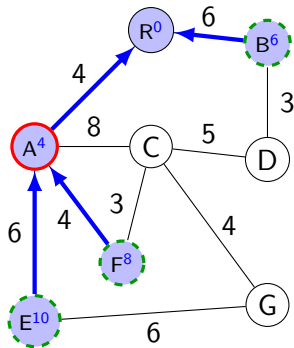
- ▶ while not lsempy(q)...
 - ▶ $n = A(T,4,R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 4$;
 - ▶ neighbour-set = $\{\cancel{E}, \cancel{R}, F, C\}$;
 - ▶ F not seen
 - ▶ $d = n_d + \text{Get-weight}(n, F, g) = 8$;
 - ▶ F.seen = True;
 - ▶ F.distance = d ;
 - ▶ F.parent = A;



$$q = \{ B(T,6,R), E(T,10,A) \}$$

Dijkstras shortest path för oriktad graf

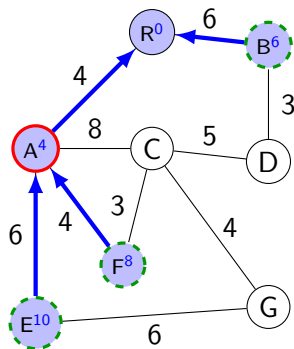
- ▶ while not lsempy(q)...
 - ▶ $n = A(T,4,R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 4$;
 - ▶ neighbour-set = $\{\cancel{E}, \cancel{R}, F, C\}$;
 - ▶ F not seen
 - ▶ $d = n_d + \text{Get-weight}(n, F, g) = 8$;
 - ▶ F.seen = True;
 - ▶ F.distance = d ;
 - ▶ F.parent = A;
 - ▶ $q = \text{Insert}(F(T,8,A), q)$;



$$q = \{ B(T,6,R), F(T,8,A), E(T,10,A) \}$$

Dijkstras shortest path för oriktad graf

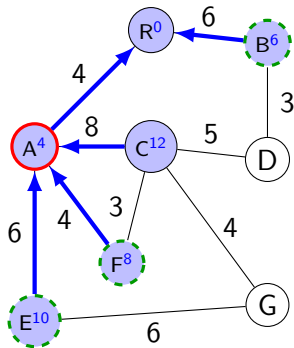
- ▶ while not lsempy(q)...
 - ▶ $n = A(T,4,R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 4$;
 - ▶ neighbour-set = $\{\cancel{E}, \cancel{R}, \cancel{F}, C\}$;
 - ▶ C not seen



$$q = \{ B(T,6,R), F(T,8,A), E(T,10,A) \}$$

Dijkstras shortest path för oriktad graf

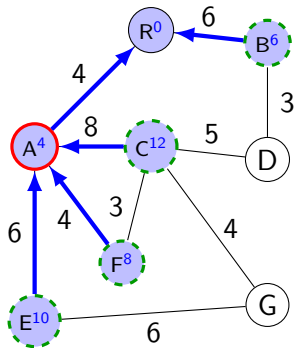
- ▶ while not lsempy(q)...
 - ▶ $n = A(T,4,R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 4$;
 - ▶ neighbour-set = $\{\cancel{E}, \cancel{R}, \cancel{F}, C\}$;
 - ▶ C not seen
 - ▶ $d = n_d + \text{Get-weight}(n,C,g) = 12$;
 - ▶ C.seen = True;
 - ▶ C.distance = d ;
 - ▶ C.parent = A;



$$q = \{ B(T,6,R), F(T,8,A), E(T,10,A) \}$$

Dijkstras shortest path för oriktad graf

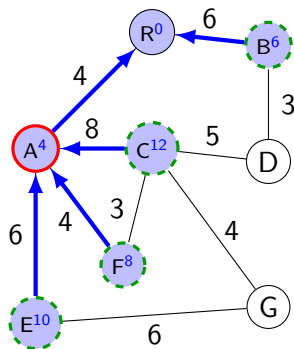
- ▶ while not lsempy(q)...
 - ▶ $n = A(T,4,R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 4$;
 - ▶ neighbour-set = $\{\cancel{E}, \cancel{R}, \cancel{F}, C\}$;
 - ▶ C not seen
 - ▶ $d = n_d + \text{Get-weight}(n,C,g) = 12$;
 - ▶ C.seen = True;
 - ▶ C.distance = d ;
 - ▶ C.parent = A;
 - ▶ $q = \text{Insert}(C(T,12,A),q)$;



$$q = \{ B(T,6,R), F(T,8,A), E(T,10,A), C(T,12,A) \}$$

Dijkstras shortest path för oriktad graf

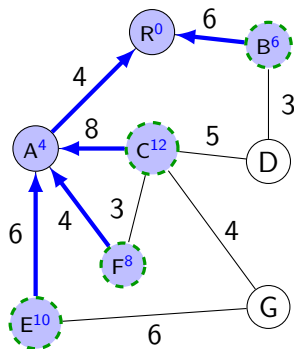
- ▶ while not lsempy(q)...
 - ▶ $n = A(T, 4, R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 4$;
 - ▶ neighbour-set = $\{\cancel{E}, \cancel{R}, \cancel{F}, \cancel{C}\}$;



$$q = \{ B(T, 6, R), F(T, 8, A), E(T, 10, A), C(T, 12, A) \}$$

Dijkstras shortest path för oriktad graf

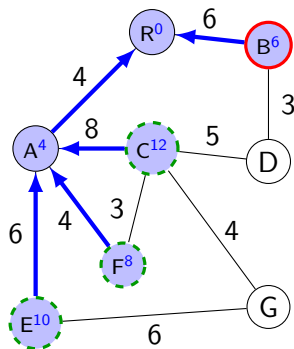
► while not lsempy(q)...



$q = \{ B(T,6,R), F(T,8,A), E(T,10,A), C(T,12,A) \}$

Dijkstras shortest path för oriktad graf

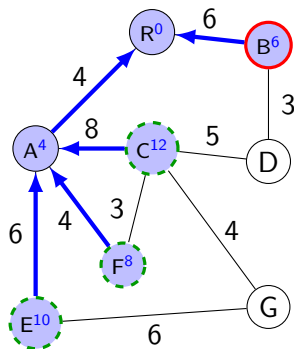
- ▶ while not Isempy(q)...
 - ▶ $n = B(T,6,R)$; $q = \text{Delete-first}(q)$;



$$q = \{ F(T,8,A), E(T,10,A), C(T,12,A) \}$$

Dijkstras shortest path för oriktad graf

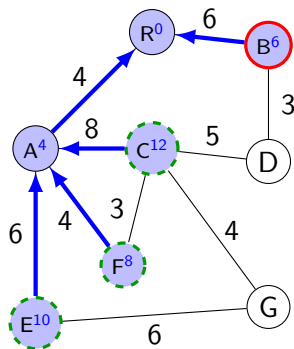
- ▶ while not lsempy(q)...
 - ▶ $n = B(T, 6, R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 6$;
 - ▶ neighbour-set = $\{D, R\}$;



$$q = \{ F(T, 8, A), E(T, 10, A), C(T, 12, A) \}$$

Dijkstras shortest path för oriktad graf

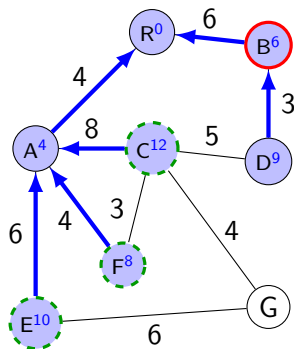
- ▶ while not lsempy(q)...
 - ▶ $n = B(T, 6, R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 6$;
 - ▶ neighbour-set = $\{D, R\}$;
 - ▶ D not seen



$$q = \{ F(T, 8, A), E(T, 10, A), C(T, 12, A) \}$$

Dijkstras shortest path för oriktad graf

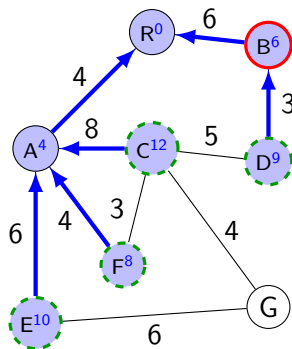
- ▶ while not lsempy(q)...
 - ▶ $n = B(T,6,R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 6$;
 - ▶ neighbour-set = $\{D,R\}$;
 - ▶ D not seen
 - ▶ $d = n_d + \text{Get-weight}(n,D,g) = 9$;
 - ▶ D.seen = True;
 - ▶ D.distance = d ;
 - ▶ D.parent = B;



$$q = \{ F(T,8,A), E(T,10,A), C(T,12,A) \}$$

Dijkstras shortest path för oriktad graf

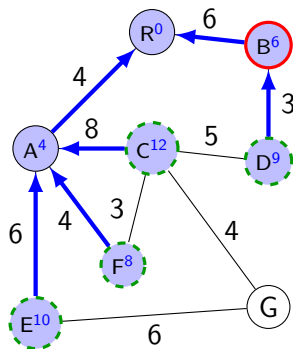
- ▶ while not lsempy(q)...
 - ▶ $n = B(T,6,R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 6$;
 - ▶ neighbour-set = $\{D,R\}$;
 - ▶ D not seen
 - ▶ $d = n_d + \text{Get-weight}(n,D,g) = 9$;
 - ▶ D.seen = True;
 - ▶ D.distance = d ;
 - ▶ D.parent = B;
 - ▶ $q = \text{Insert}(D(T,9,B),q)$;



$$q = \{ F(T,8,A), D(T,9,B), E(T,10,A), C(T,12,A) \}$$

Dijkstras shortest path för oriktad graf

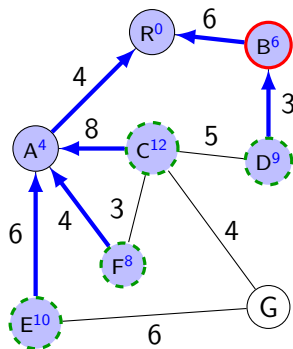
- ▶ while not lsempy(q)...
 - ▶ $n = B(T, 6, R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 6$;
 - ▶ neighbour-set = $\{\cancel{D}, R\}$;
 - ▶ R seen



$$q = \{ F(T, 8, A), D(T, 9, B), E(T, 10, A), C(T, 12, A) \}$$

Dijkstras shortest path för oriktad graf

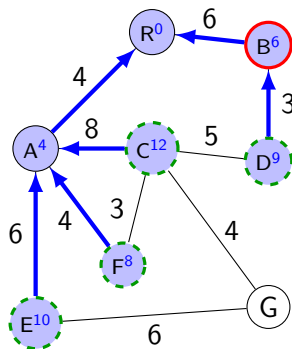
- ▶ while not lsempy(q)...
 - ▶ $n = B(T,6,R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 6$;
 - ▶ neighbour-set = $\{\cancel{D}, R\}$;
 - ▶ R seen
 - ▶ $d = n_d + \text{Get-weight}(n,R,g) = 12$;
 - ▶ d **not** $< R.\text{distance}$



$$q = \{ F(T,8,A), D(T,9,B), E(T,10,A), C(T,12,A) \}$$

Dijkstras shortest path för oriktad graf

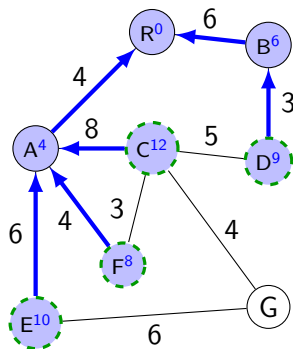
- ▶ while not lsempy(q)...
 - ▶ $n = B(T,6,R)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 6$;
 - ▶ neighbour-set = $\{\cancel{D}, \cancel{R}\}$;



$$q = \{ F(T,8,A), D(T,9,B), E(T,10,A), C(T,12,A) \}$$

Dijkstras shortest path för oriktad graf

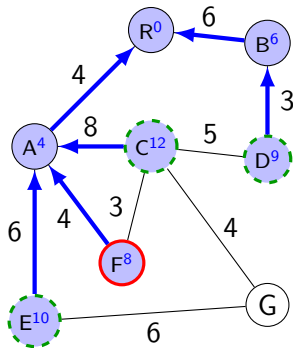
► while not lsempy(q)...



$$q = \{ F(T,8,A), D(T,9,B), E(T,10,A), C(T,12,A) \}$$

Dijkstras shortest path för oriktad graf

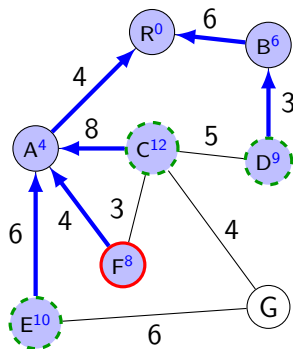
- ▶ while not lsempy(q)...
 - ▶ $n = F(T, 8, A)$; $q = \text{Delete-first}(q)$;



$$q = \{ D(T, 9, B), E(T, 10, A), C(T, 12, A) \}$$

Dijkstras shortest path för oriktad graf

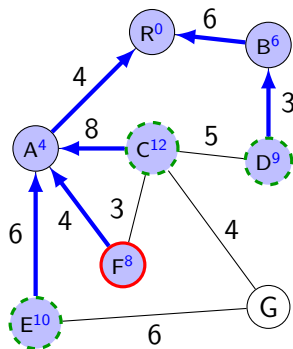
- ▶ while not lsempy(q)...
 - ▶ $n = F(T, 8, A)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 8$;
 - ▶ neighbour-set = $\{A, C\}$;



$$q = \{ D(T, 9, B), E(T, 10, A), C(T, 12, A) \}$$

Dijkstras shortest path för oriktad graf

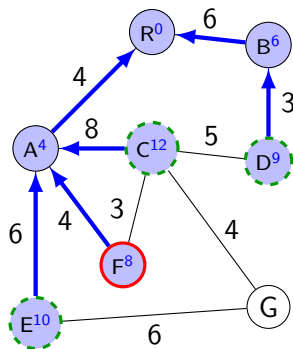
- ▶ while not lsempy(q)...
 - ▶ $n = F(T, 8, A)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 8$;
 - ▶ neighbour-set = $\{A, C\}$;
 - ▶ A seen



$$q = \{ D(T, 9, B), E(T, 10, A), C(T, 12, A) \}$$

Dijkstras shortest path för oriktad graf

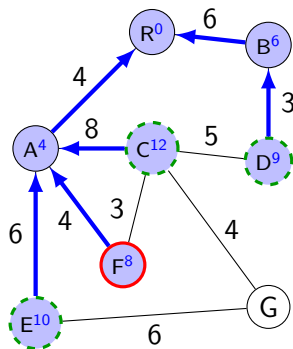
- ▶ while not lsempy(q)...
 - ▶ $n = F(T, 8, A)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 8$;
 - ▶ neighbour-set = $\{A, C\}$;
 - ▶ A seen
 - ▶ $d = n_d + \text{Get-weight}(n, A, g) = 12$;
 - ▶ d **not** $< A.\text{distance}$



$$q = \{ D(T, 9, B), E(T, 10, A), C(T, 12, A) \}$$

Dijkstras shortest path för oriktad graf

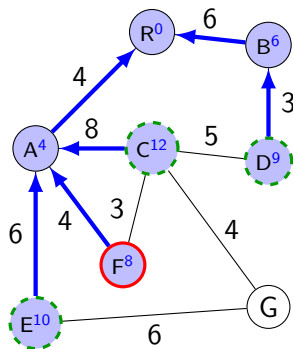
- ▶ while not lsempy(q)...
 - ▶ $n = F(T, 8, A)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 8$;
 - ▶ neighbour-set = $\{\cancel{A}, C\}$;
 - ▶ C seen



$$q = \{ D(T, 9, B), E(T, 10, A), C(T, 12, A) \}$$

Dijkstras shortest path för oriktad graf

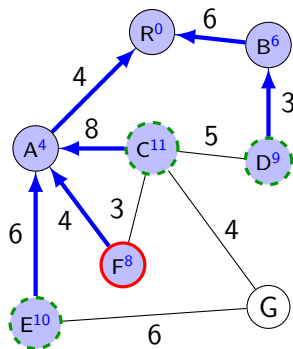
- ▶ while not lsempy(q)...
 - ▶ $n = F(T, 8, A)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 8$;
 - ▶ neighbour-set = $\{\cancel{A}, C\}$;
 - ▶ C seen
 - ▶ $d = n_d + \text{Get-weight}(n, C, g) = 11$;
 - ▶ d is $< C.\text{distance}$



$$q = \{ D(T, 9, B), E(T, 10, A), C(T, 12, A) \}$$

Dijkstras shortest path för oriktad graf

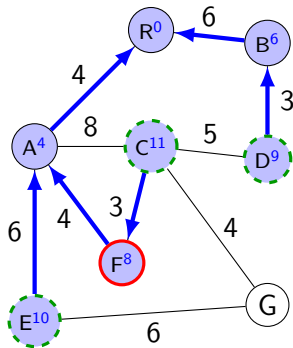
- ▶ while not lsempy(q)...
 - ▶ $n = F(T, 8, A)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 8$;
 - ▶ neighbour-set = $\{\cancel{A}, C\}$;
 - ▶ C seen
 - ▶ $d = n_d + \text{Get-weight}(n, C, g) = 11$;
 - ▶ d is $< C.\text{distance}$
 - ▶ $C.\text{distance} = d$;



$$q = \{ D(T, 9, B), E(T, 10, A), C(T, 12, A) \}$$

Dijkstras shortest path för oriktad graf

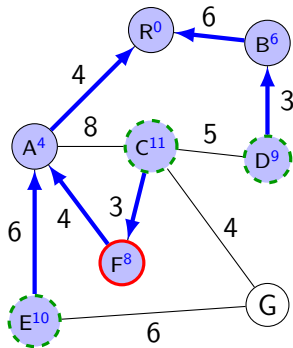
- ▶ while not lsempy(q)...
 - ▶ $n = F(T, 8, A)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 8$;
 - ▶ neighbour-set = $\{\cancel{A}, C\}$;
 - ▶ C seen
 - ▶ $d = n_d + \text{Get-weight}(n, C, g) = 11$;
 - ▶ d is $< C.\text{distance}$
 - ▶ $C.\text{distance} = d$;
 - ▶ $C.\text{parent} = F$;



$$q = \{ D(T, 9, B), E(T, 10, A), C(T, 12, A) \}$$

Dijkstras shortest path för oriktad graf

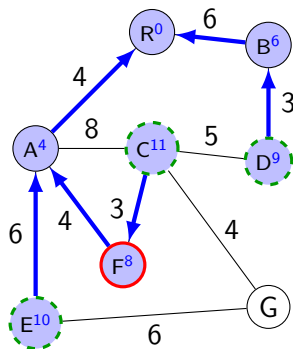
- ▶ while not lsempy(q)...
 - ▶ $n = F(T,8,A)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 8$;
 - ▶ neighbour-set = $\{\cancel{A}, C\}$;
 - ▶ C seen
 - ▶ $d = n_d + \text{Get-weight}(n, C, g) = 11$;
 - ▶ d is $< C.\text{distance}$
 - ▶ $C.\text{distance} = d$;
 - ▶ $C.\text{parent} = F$;
 - ▶ $q = \text{update}(C, q)$;



$$q = \{ D(T,9,B), E(T,10,A), C(T,11,F) \}$$

Dijkstras shortest path för oriktad graf

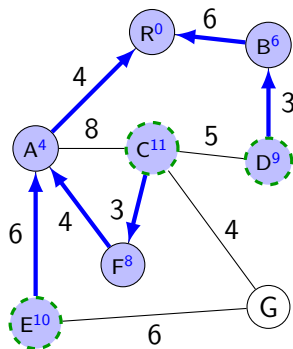
- ▶ while not lsempy(q)...
 - ▶ $n = F(T, 8, A)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 8$;
 - ▶ neighbour-set = $\{\cancel{A}, \cancel{C}\}$;



$$q = \{ D(T, 9, B), E(T, 10, A), C(T, 11, F) \}$$

Dijkstras shortest path för oriktad graf

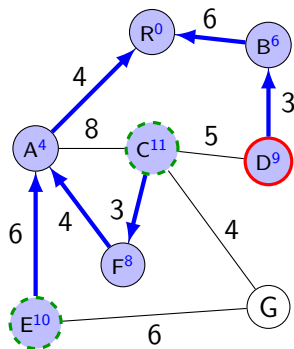
► while not lsempy(q)...



$$q = \{ D(T,9,B), E(T,10,A), C(T,11,F) \}$$

Dijkstras shortest path för oriktad graf

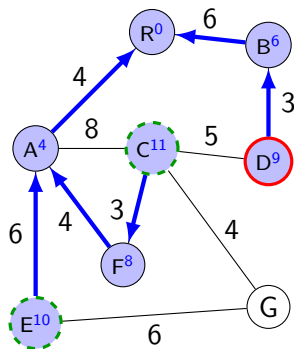
- ▶ while not lsempy(q)...
 - ▶ $n = D(T,9,B)$; $q = \text{Delete-first}(q)$;



$$q = \{ E(T,10,A), C(T,11,F) \}$$

Dijkstras shortest path för oriktad graf

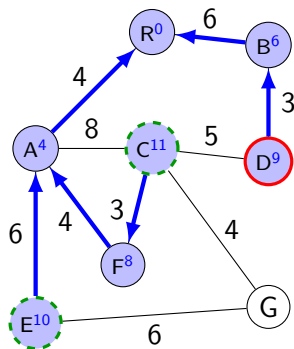
- ▶ while not lsempy(q)...
 - ▶ $n = D(T, 9, B)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 9$;
 - ▶ neighbour-set = {B,C};



$$q = \{ E(T, 10, A), C(T, 11, F) \}$$

Dijkstras shortest path för oriktad graf

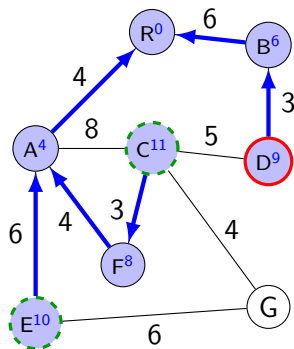
- ▶ while not lsempy(q)...
 - ▶ $n = D(T, 9, B)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 9$;
 - ▶ neighbour-set = {B,C};
 - ▶ B seen



$$q = \{ E(T, 10, A), C(T, 11, F) \}$$

Dijkstras shortest path för oriktad graf

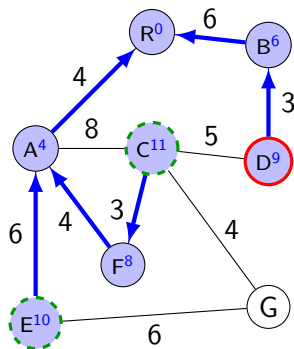
- ▶ while not lsempy(q)...
 - ▶ $n = D(T, 9, B)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 9$;
 - ▶ neighbour-set = {B,C};
 - ▶ B seen
 - ▶ $d = n_d + \text{Get-weight}(n, B, g) = 12$;
 - ▶ d **not** < B.distance



$$q = \{ E(T, 10, A), C(T, 11, F) \}$$

Dijkstras shortest path för oriktad graf

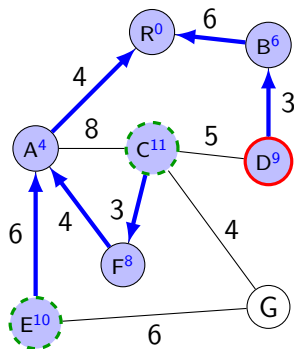
- ▶ while not lsempy(q)...
 - ▶ $n = D(T, 9, B)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 9$;
 - ▶ neighbour-set = $\{\cancel{B}, C\}$;
 - ▶ C seen



$$q = \{ E(T, 10, A), C(T, 11, F) \}$$

Dijkstras shortest path för oriktad graf

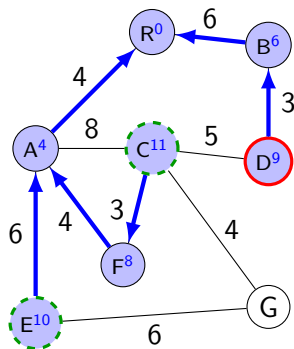
- ▶ while not lsempy(q)...
 - ▶ $n = D(T, 9, B)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 9$;
 - ▶ neighbour-set = $\{\cancel{B}, C\}$;
 - ▶ C seen
 - ▶ $d = n_d + \text{Get-weight}(n, C, g) = 14$;
 - ▶ d **not** $< C.\text{distance}$



$$q = \{ E(T, 10, A), C(T, 11, F) \}$$

Dijkstras shortest path för oriktad graf

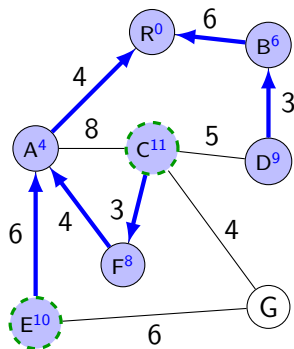
- ▶ while not lsempy(q)...
 - ▶ $n = D(T, 9, B)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 9$;
 - ▶ neighbour-set = $\{B, C\}$;



$$q = \{ E(T, 10, A), C(T, 11, F) \}$$

Dijkstras shortest path för oriktad graf

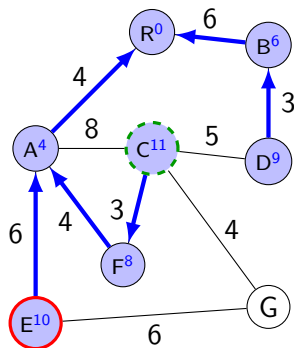
► while not lsempy(q)...



$$q = \{ E(T,10,A), C(T,11,F) \}$$

Dijkstras shortest path för oriktad graf

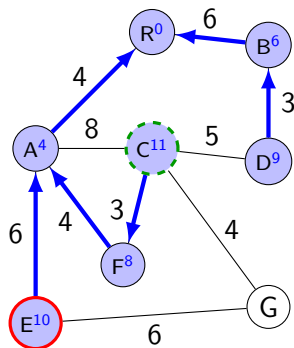
- ▶ while not lsempy(q)...
 - ▶ $n = E(T, 10, A)$; $q = \text{Delete-first}(q)$;



$$q = \{ C(T, 11, F) \}$$

Dijkstras shortest path för oriktad graf

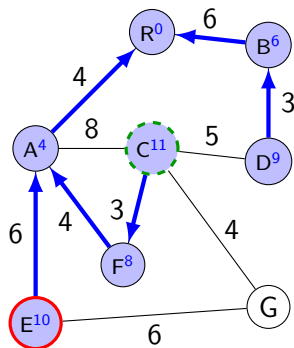
- ▶ while not lsempy(q)...
 - ▶ $n = E(T, 10, A)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 10$;
 - ▶ neighbour-set = $\{A, G\}$;



$$q = \{ C(T, 11, F) \}$$

Dijkstras shortest path för oriktad graf

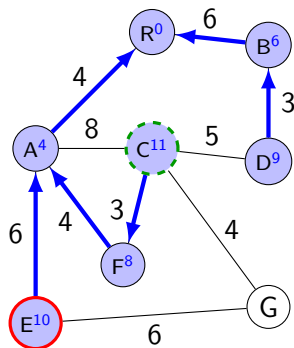
- ▶ while not lsempy(q)...
 - ▶ $n = E(T, 10, A)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 10$;
 - ▶ neighbour-set = {A,G};
 - ▶ A seen



$$q = \{ C(T, 11, F) \}$$

Dijkstras shortest path för oriktad graf

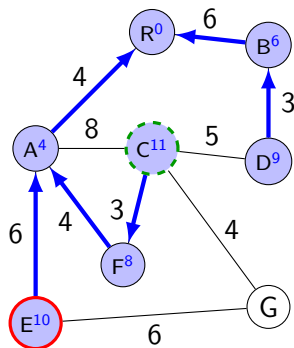
- ▶ while not lsempy(q)...
 - ▶ $n = E(T, 10, A)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 10$;
 - ▶ neighbour-set = $\{A, G\}$;
 - ▶ A seen
 - ▶ $d = n_d + \text{Get-weight}(n, A, g) = 16$;
 - ▶ d **not** $< A.\text{distance}$



$$q = \{ C(T, 11, F) \}$$

Dijkstras shortest path för oriktad graf

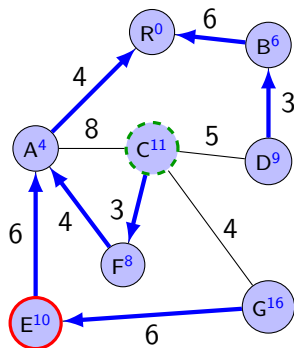
- ▶ while not lsempy(q)...
 - ▶ $n = E(T, 10, A)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 10$;
 - ▶ neighbour-set = $\{A, G\}$;
 - ▶ G not seen



$$q = \{ C(T, 11, F) \}$$

Dijkstras shortest path för oriktad graf

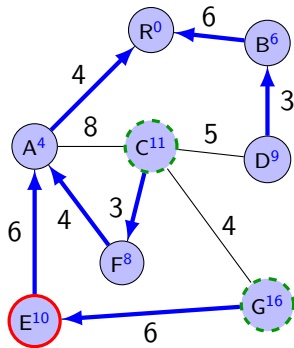
- ▶ while not lsempy(q)...
 - ▶ $n = E(T, 10, A)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 10$;
 - ▶ neighbour-set = $\{A, G\}$;
 - ▶ G not seen
 - ▶ $d = n_d + \text{Get-weight}(n, G, g) = 16$;
 - ▶ G.seen = True;
 - ▶ G.distance = d ;
 - ▶ G.parent = E;



$$q = \{ C(T, 11, F) \}$$

Dijkstras shortest path för oriktad graf

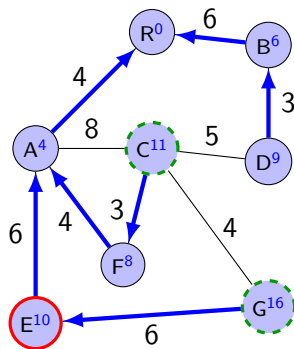
- ▶ while not lsempy(q)...
 - ▶ $n = E(T,10,A)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 10$;
 - ▶ neighbour-set = $\{A, G\}$;
 - ▶ G not seen
 - ▶ $d = n_d + \text{Get-weight}(n, G, g) = 16$;
 - ▶ G.seen = True;
 - ▶ G.distance = d ;
 - ▶ G.parent = E;
 - ▶ $q = \text{Insert}(G(T,16,E), q)$;



$$q = \{ C(T,11,F), G(T,16,E) \}$$

Dijkstras shortest path för oriktad graf

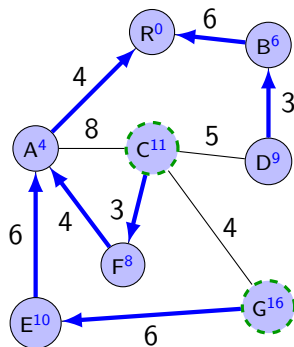
- ▶ while not lsempy(q)...
 - ▶ $n = E(T, 10, A)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 10$;
 - ▶ neighbour-set = $\{\cancel{A}, \cancel{G}\}$;



$$q = \{ C(T, 11, F), G(T, 16, E) \}$$

Dijkstras shortest path för oriktad graf

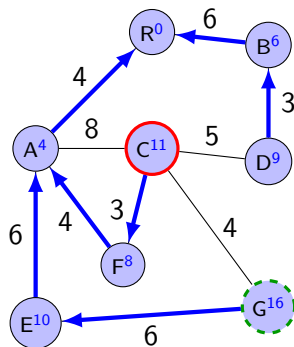
► while not lsempy(q)...



$$q = \{ C(T,11,F), G(T,16,E) \}$$

Dijkstras shortest path för oriktad graf

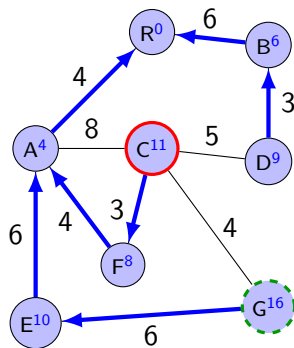
- ▶ while not lsempy(q)...
 - ▶ $n = C(T, 11, F)$; $q = \text{Delete-first}(q)$;



$$q = \{ G(T, 16, E) \}$$

Dijkstras shortest path för oriktad graf

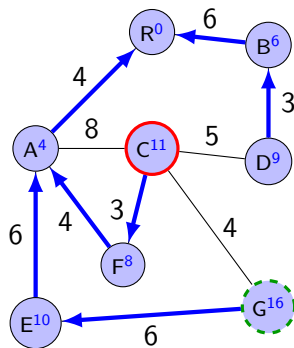
- ▶ while not lsempy(q)...
 - ▶ $n = C(T, 11, F)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 11$;
 - ▶ neighbour-set = {A,F,G,D};



$$q = \{ G(T, 16, E) \}$$

Dijkstras shortest path för oriktad graf

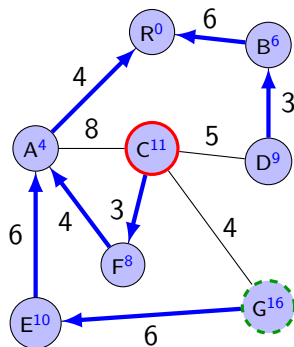
- ▶ while not lsempy(q)...
 - ▶ $n = C(T, 11, F)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 11$;
 - ▶ neighbour-set = {A, F, G, D};
 - ▶ A seen



$$q = \{ G(T, 16, E) \}$$

Dijkstras shortest path för oriktad graf

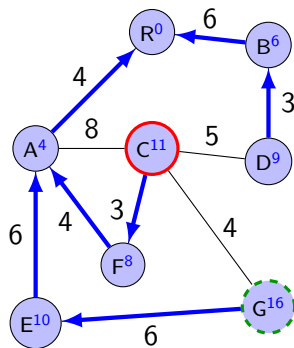
- ▶ while not lsempy(q)...
 - ▶ $n = C(T, 11, F)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 11$;
 - ▶ neighbour-set = {A,F,G,D};
 - ▶ A seen
 - ▶ $d = n_d + \text{Get-weight}(n, A, g) = 19$;
 - ▶ d **not** < A.distance



$$q = \{ G(T, 16, E) \}$$

Dijkstras shortest path för oriktad graf

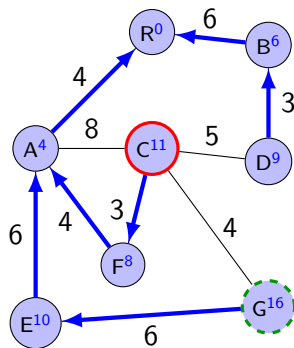
- ▶ while not lsempy(q)...
 - ▶ $n = C(T, 11, F)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 11$;
 - ▶ neighbour-set = $\{\cancel{A}, F, G, D\}$;
 - ▶ F seen



$$q = \{ G(T, 16, E) \}$$

Dijkstras shortest path för oriktad graf

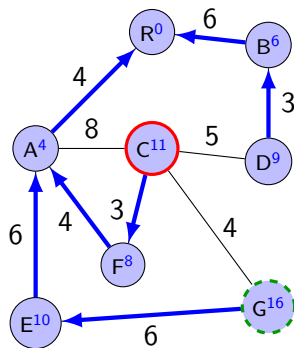
- ▶ while not lsempy(q)...
 - ▶ $n = C(T, 11, F)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 11$;
 - ▶ neighbour-set = $\{\cancel{A}, F, G, D\}$;
 - ▶ F seen
 - ▶ $d = n_d + \text{Get-weight}(n, F, g) = 14$;
 - ▶ d **not** $< F.\text{distance}$



$$q = \{ G(T, 16, E) \}$$

Dijkstras shortest path för oriktad graf

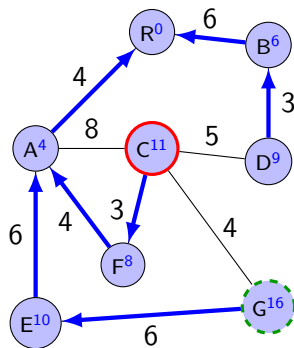
- ▶ while not lsempy(q)...
 - ▶ $n = C(T, 11, F)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 11$;
 - ▶ neighbour-set = $\{\cancel{A}, \cancel{F}, G, D\}$;
 - ▶ G seen



$$q = \{ G(T, 16, E) \}$$

Dijkstras shortest path för oriktad graf

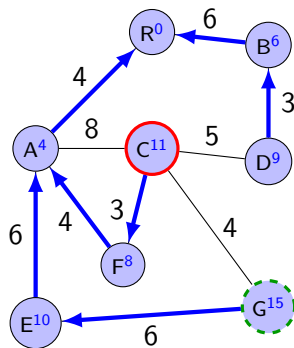
- ▶ while not lsempy(q)...
 - ▶ $n = C(T, 11, F)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 11$;
 - ▶ neighbour-set = $\{\cancel{A}, \cancel{F}, G, D\}$;
 - ▶ G seen
 - ▶ $d = n_d + \text{Get-weight}(n, G, g) = 15$;
 - ▶ d is $< G.\text{distance}$



$$q = \{ G(T, 16, E) \}$$

Dijkstras shortest path för oriktad graf

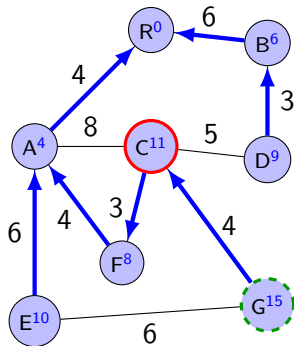
- ▶ while not lsempy(q)...
 - ▶ $n = C(T, 11, F)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 11$;
 - ▶ neighbour-set = $\{\cancel{A}, \cancel{F}, G, D\}$;
 - ▶ G seen
 - ▶ $d = n_d + \text{Get-weight}(n, G, g) = 15$;
 - ▶ d is $< G.\text{distance}$
 - ▶ $G.\text{distance} = d$;



$$q = \{ G(T, 16, E) \}$$

Dijkstras shortest path för oriktad graf

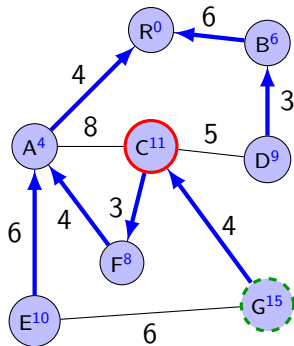
- ▶ while not lsempy(q)...
 - ▶ $n = C(T, 11, F)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 11$;
 - ▶ neighbour-set = $\{\cancel{A}, \cancel{F}, G, D\}$;
 - ▶ G seen
 - ▶ $d = n_d + \text{Get-weight}(n, G, g) = 15$;
 - ▶ d is $< G.\text{distance}$
 - ▶ $G.\text{distance} = d$;
 - ▶ $G.\text{parent} = C$;



$$q = \{ G(T, 16, E) \}$$

Dijkstras shortest path för oriktad graf

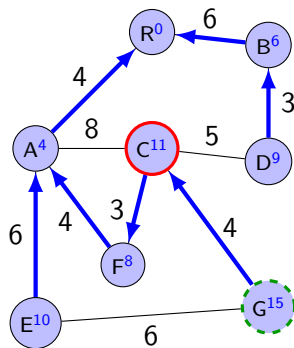
- ▶ while not lsempy(q)...
 - ▶ $n = C(T, 11, F)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 11$;
 - ▶ neighbour-set = $\{\cancel{A}, \cancel{F}, G, D\}$;
 - ▶ G seen
 - ▶ $d = n_d + \text{Get-weight}(n, G, g) = 15$;
 - ▶ d is $< G.\text{distance}$
 - ▶ $G.\text{distance} = d$;
 - ▶ $G.\text{parent} = C$;
 - ▶ $q = \text{update}(G, q)$;



$$q = \{ G(T, 15, C) \}$$

Dijkstras shortest path för oriktad graf

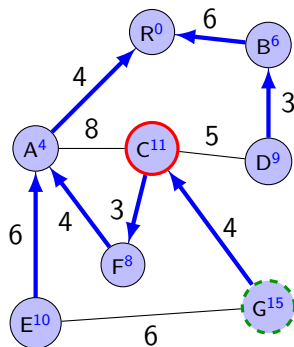
- ▶ while not lsempy(q)...
 - ▶ $n = C(T, 11, F)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 11$;
 - ▶ neighbour-set = $\{\cancel{A}, \cancel{F}, \cancel{G}, D\}$;
 - ▶ D seen



$$q = \{ G(T, 15, C) \}$$

Dijkstras shortest path för oriktad graf

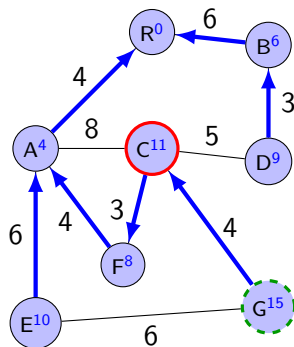
- ▶ while not lsempy(q)...
 - ▶ $n = C(T, 11, F)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 11$;
 - ▶ neighbour-set = $\{\cancel{A}, \cancel{F}, \cancel{G}, D\}$;
 - ▶ D seen
 - ▶ $d = n_d + \text{Get-weight}(n, D, g) = 16$;
 - ▶ d **not** $< D.\text{distance}$



$$q = \{ G(T, 15, C) \}$$

Dijkstras shortest path för oriktad graf

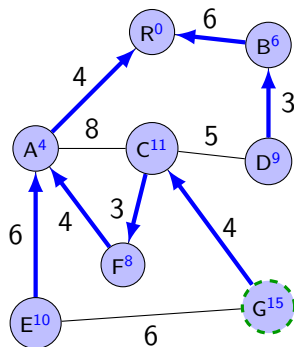
- ▶ while not lsempy(q)...
 - ▶ $n = C(T, 11, F)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 11$;
 - ▶ neighbour-set = $\{\cancel{A}, \cancel{F}, \cancel{G}, \cancel{D}\}$;



$$q = \{ G(T, 15, C) \}$$

Dijkstras shortest path för oriktad graf

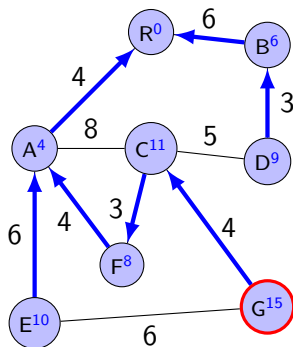
► while not lsempy(q)...



$$q = \{ G(T,15,C) \}$$

Dijkstras shortest path för oriktad graf

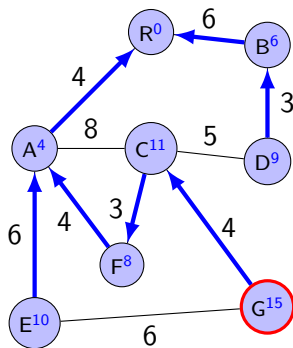
- ▶ while not lsempy(q)...
 - ▶ $n = G(T, 15, C)$; $q = \text{Delete-first}(q)$;



$$q = \{ \}$$

Dijkstras shortest path för oriktad graf

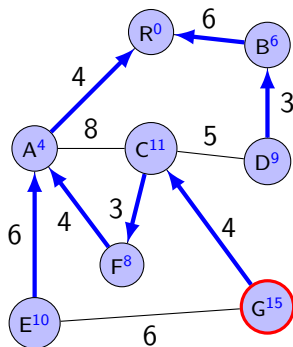
- ▶ while not lsempy(q)...
 - ▶ $n = G(T, 15, C)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 15$;
 - ▶ neighbour-set = $\{E, C\}$;



$$q = \{ \}$$

Dijkstras shortest path för oriktad graf

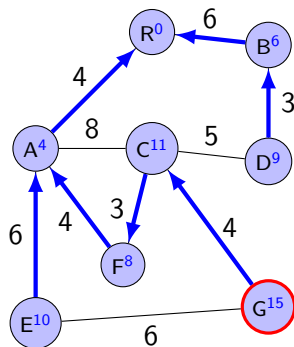
- ▶ while not lsempy(q)...
 - ▶ $n = G(T, 15, C)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 15$;
 - ▶ neighbour-set = $\{E, C\}$;
 - ▶ E seen



$$q = \{ \}$$

Dijkstras shortest path för oriktad graf

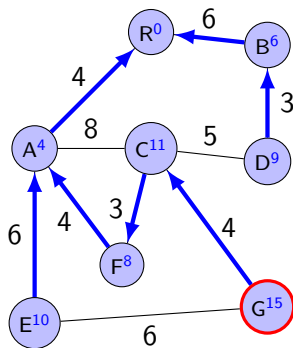
- ▶ while not lsempy(q)...
 - ▶ $n = G(T, 15, C)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 15$;
 - ▶ neighbour-set = $\{E, C\}$;
 - ▶ E seen
 - ▶ $d = n_d + \text{Get-weight}(n, E, g) = 21$;
 - ▶ d **not** $< E.\text{distance}$



$$q = \{ \}$$

Dijkstras shortest path för oriktad graf

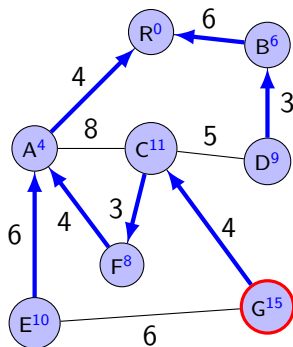
- ▶ while not lsempy(q)...
 - ▶ $n = G(T, 15, C)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 15$;
 - ▶ neighbour-set = $\{\text{E}, C\}$;
 - ▶ C seen



$$q = \{ \}$$

Dijkstras shortest path för oriktad graf

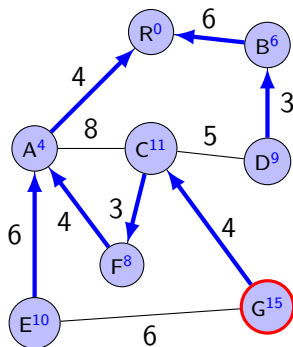
- ▶ while not lsempy(q)...
 - ▶ $n = G(T, 15, C)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 15$;
 - ▶ neighbour-set = $\{\text{E}, C\}$;
 - ▶ C seen
 - ▶ $d = n_d + \text{Get-weight}(n, C, g) = 19$;
 - ▶ d **not** $< C.\text{distance}$



$$q = \{ \}$$

Dijkstras shortest path för oriktad graf

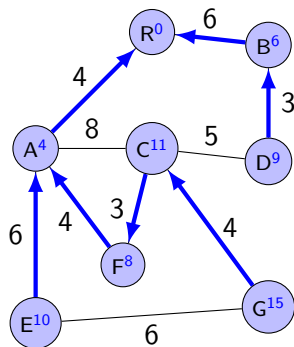
- ▶ while not lsempy(q)...
 - ▶ $n = G(T, 15, C)$; $q = \text{Delete-first}(q)$;
 - ▶ $n_d = n.\text{distance} = 15$;
 - ▶ neighbour-set = $\{\cancel{E}, \cancel{C}\}$;



$$q = \{ \}$$

Dijkstras shortest path för oriktad graf

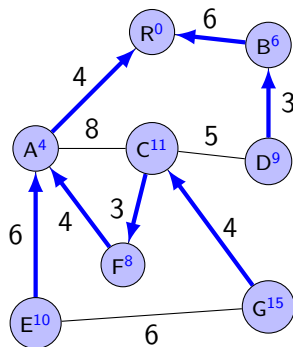
► while not lsempy(q)...



$q = \{ \}$

Dijkstras shortest path för oriktad graf

- Klar!
- Varje nod innehåller nu
 - *avståndet* till startnoden
 - *bågen* som leder tillbaka till startnoden



Komplexitet?

```
Algorithm Dijkstra-shortest-path(n: Node, g: Graph)
// Input: A graph g to find shortest path from node n

// Distance to start node is zero
n.distance ← 0; n.seen ← True; n.parent ← NULL
// Initialize pqueue with start node
q ← Insert(n, Pqueue-empty())
while not Isempty(q)
    // Get node with shortest distance from queue
    n ← Inspect-first(q); q ← Delete-first(q)
    nd ← n.distance
    // ...and its neighbours
    neighbour-set ← Neighbours(n, g)
    for each neighbour b in neighbour-set do
        // Compute distance to b VIA n
        d ← nd + Get-weight(n, b, g)
        if not Is-seen(b, g) then
            // We've never seen b; this is the first path to arrive at b
            b.distance ← d
            b.seen ← True
            b.parent ← n
            // Add new node to pqueue
            q ← Insert(b, q)
        else if d < b.distance then
            // We've seen b before, but path via n is shorter
            b.distance ← d
            // Update how we came to b
            b.parent ← n
            // Update the pqueue based on the new distance
            q ← Update(b, q)
```

Dijkstras shortest path, komplexitet

- ▶ Vi **sätter in** varje nod i prioritetsskön en gång:
 - ▶ Totalt $n \cdot O(\text{Insert})$
- ▶ Vi **läser av** varje nod i prioritetsskön en gång
 - ▶ Totalt $n \cdot O(\text{Inspect-first})$
- ▶ Vi **tar ut** varje nod ur prioritetsskön en gång
 - ▶ Totalt $n \cdot O(\text{Delete-first})$
- ▶ Vi kan behöva **uppdatera** element i prioritetsskön
 - ▶ Maximalt m gånger: $m \cdot O(\text{update})$
- ▶ Totalt för olika konstruktioner av prioritetsskön:
 - ▶ Osorterad lista (av **referenser** till noderna):
 - ▶ $nO(1) + nO(n) + nO(n) + mO(1) = O(n^2 + m)$
 - ▶ Sorterad lista:
 - ▶ $nO(n) + nO(1) + nO(1) + mO(n) = O(n^2 + mn)$
 - ▶ Heap:
 - ▶ $nO(\log n) + nO(1) + nO(\log n) + mO(\log n) = O((n + m) \log n)$
- ▶ Heap är snabbast!

Komplexitet, kortaste vägen

- ▶ En-till-alla:
 - ▶ Floyd: $O(n^3)$ (finns ej i en-till-alla-version)
 - ▶ Dijkstra: $O((n + m) \log n)$
- ▶ Alla-till-alla:
 - ▶ Floyd: $O(n^3)$
 - ▶ Dijkstra: $O((n + m) \log n)$ för en-till-alla
 - ▶ Måste köras n gånger för att få alla-till-alla:
 - ▶ $nO((n + m) \log n) = O(n^2 \log n + mn \log n)$
 - ▶ För gles graf $m \approx n$: $O(n^2 \log n)$
 - ▶ För tät graf $m \approx n^2$: $O(n^3 \log n)$
 - ▶ Dijkstra snabbare på stora, glesa grafer

Blank

Blank

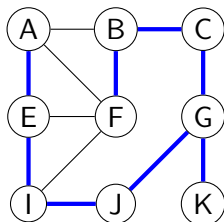
Blank

3. Minsta uppspännande träd

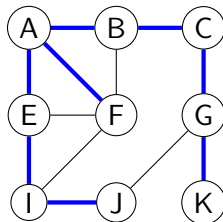
Uppspännande träd, oviktad graf

- ▶ Både bredden-först och djupet-först-traverseringarna gav oss uppspännande träd:

- ▶ Djupet-först:



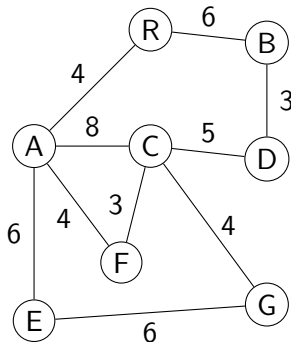
- ▶ Bredden-först:



- ▶ Har träden **minimal längd**?
 - ▶ För oviktade grafer — ja!
 - ▶ Längd = $n - 1$
 - ▶ Om varje kant har samma vikt är **alla** uppspännande träd minimala

Uppspännande träd, viktad graf

- ▶ Hur hanterar man grafer med **vikter**?
 - ▶ Exempel: Bygga fibernät mellan byar
 - ▶ Vikten på bågen motsvarar **kostnaden** att dra fiber mellan grannbyarna
 - ▶ Man söker ett uppspännande träd med minsta möjliga **totala** längd
 - ▶ Det är alltså **inte** en kortaste-vägen-algoritm
 - ▶ För **mängdororienterad** specifikation finns **Kruskals** algoritm
 - ▶ För **navigeringsorienterad** specifikation finns **Prims** algoritm



Blank

Kruskals algorithm

Kruskals algoritm för minsta uppspännande träd

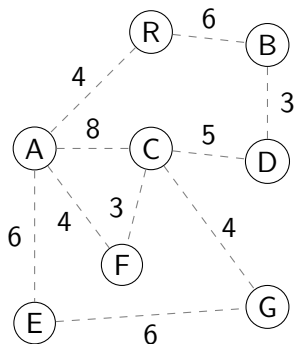
- ▶ Utgå från en prioritetskö av **alla** bågar
- ▶ I varje steg, plocka **kortaste** bågen från kön
 - ▶ Fyra alternativ:
 1. Bilda **nytt** träd
 2. **Bygg ut** ett träd
 3. **Ignorera** bågen
 4. **Slå ihop** två träd
- ▶ Under algoritmens gång kan vi ha en **skog**
- ▶ Till slut har vi bara ett **träd** (för sammanhängande gra)fc
- ▶ Vår beskrivning använder **färger** för att hålla i sär träden

Kruskals algoritm för minsta uppspännande träd, algoritm

- ▶ Låt alla noder sakna färg
- ▶ Stoppa in alla bågarna i en prioritetskö q , sorterade efter vikt
- ▶ Upprepa tills q är tom:
 0. Ta första bågen ur q
 1. Om ingen av noderna är färgade:
 - ▶ Färglägg med **ny** färg (bilda nytt träd)
 2. Om endast en nod är färgad:
 - ▶ Färglägg den **ofärgade** noden (utöka trädet)
 3. Om bägge noderna har **samma** färg:
 - ▶ **Ignorera** bågen (den skulle skapa en cykel)
 4. Om noderna har **olika** färg
 - ▶ Välj en av färgerna och **färga om** det nya gemensamma trädet (slå ihop träden)

Kruskals algoritm för minsta uppspannande träd, exempel

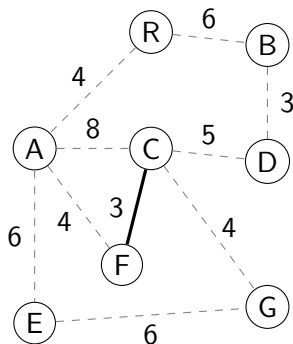
- Upprepa tills kön är tom:



$$q = \{ (C,F,3), (B,D,3), (C,G,4), (A,F,4), (A,R,4), (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspännande träd, exempel

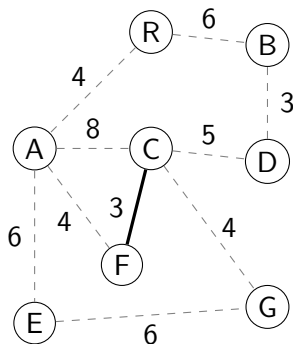
- Upprepa tills kön är tom:
 - Ta första bågen $(C,F,3)$ ur kön



$$q = \{ (B,D,3), (C,G,4), (A,F,4), (A,R,4), (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

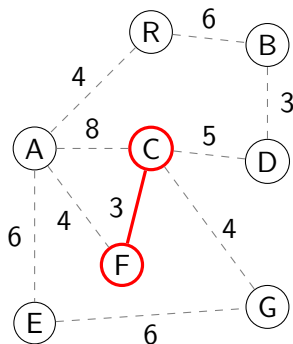
- Upprepa tills kön är tom:
 - Ta första bågen $(C,F,3)$ ur kön
 - Ingen av (C,F) är färgade:



$$q = \{ (B,D,3), (C,G,4), (A,F,4), (A,R,4), (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

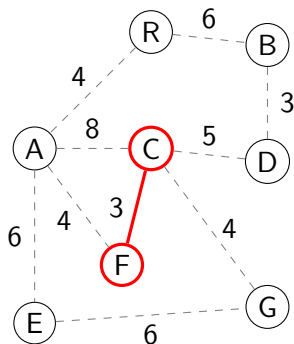
- ▶ Upprepa tills kön är tom:
 - ▶ Ta första bågen $(C,F,3)$ ur kön
 - ▶ Ingen av (C,F) är färgade:
 - ▶ Färglägg med ny färg (fall 1)



$$q = \{ (B,D,3), (C,G,4), (A,F,4), (A,R,4), (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

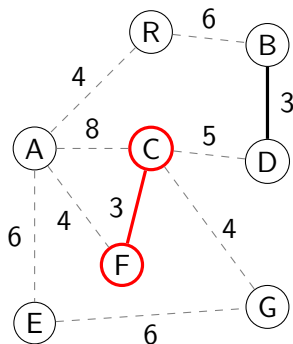
- Upprepa tills kön är tom:



$$q = \{ (B,D,3), (C,G,4), (A,F,4), (A,R,4), (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

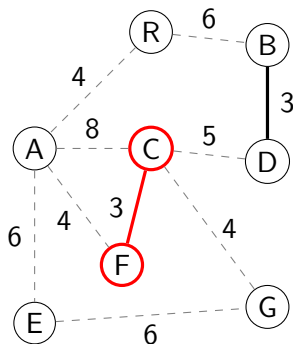
- Upprepa tills kön är tom:
 - Ta första bågen (B,D,3) ur kön



$$q = \{ (C,G,4), (A,F,4), (A,R,4), (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspännande träd, exempel

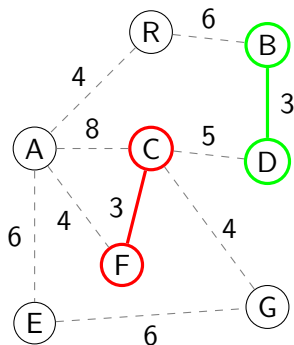
- Upprepa tills kön är tom:
 - Ta första bågen $(B,D,3)$ ur kön
 - Ingen av (B,D) är färgade:



$$q = \{ (C,G,4), (A,F,4), (A,R,4), (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

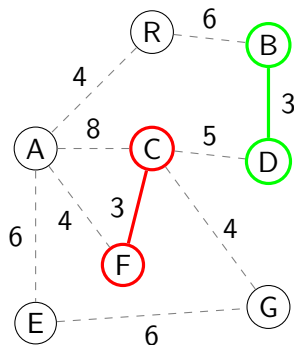
- Upprepa tills kön är tom:
 - Ta första bågen $(B,D,3)$ ur kön
 - Ingen av (B,D) är färgade:
 - Färglägg med ny färg (fall 1)



$$q = \{ (C,G,4), (A,F,4), (A,R,4), (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

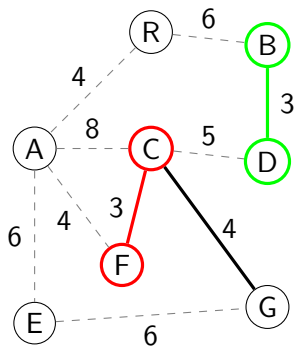
- Upprepa tills kön är tom:



$$q = \{ (C,G,4), (A,F,4), (A,R,4), (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

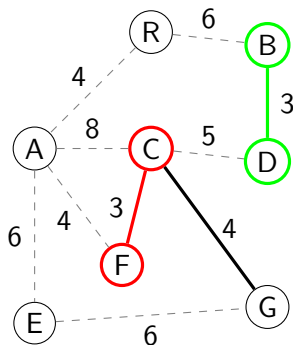
- Upprepa tills kön är tom:
 - Ta första bågen $(C,G,4)$ ur kön



$$q = \{ (A,F,4), (A,R,4), (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

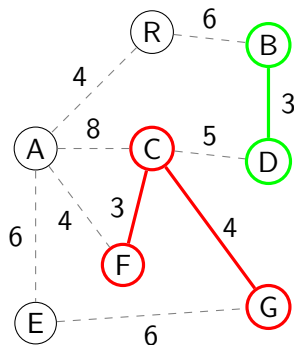
- Upprepa tills kön är tom:
 - Ta första bågen (C,G,4) ur kön
 - C är färgad



$$q = \{ (A,F,4), (A,R,4), (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

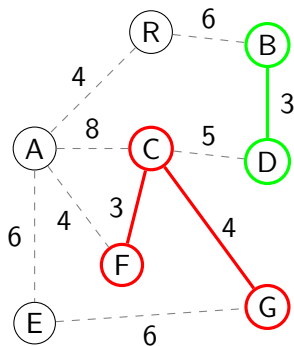
- ▶ Upprepa tills kön är tom:
 - ▶ Ta första bågen (C,G,4) ur kön
 - ▶ C är färgad
 - ▶ Färglägg med C:s färg (fall 2)



$$q = \{ (A,F,4), (A,R,4), (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

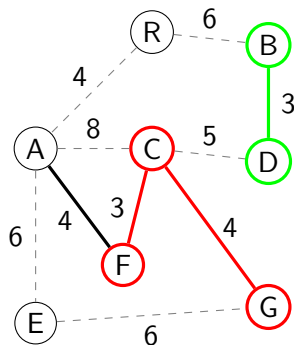
- Upprepa tills kön är tom:



$$q = \{ (A,F,4), (A,R,4), (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

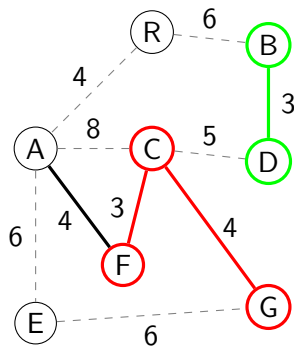
- Upprepa tills kön är tom:
 - Ta första bågen (A,F,4) ur kön



$$q = \{ (A,R,4), (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspännande träd, exempel

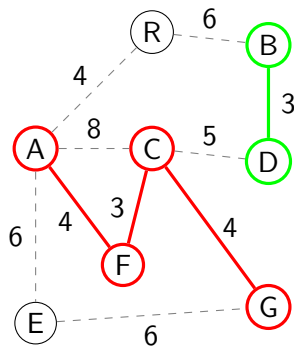
- ▶ Upprepa tills kön är tom:
 - ▶ Ta första bågen (A,F,4) ur kön
 - ▶ F är färgad



$$q = \{ (A,R,4), (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

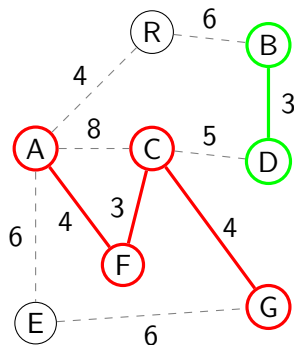
- ▶ Upprepa tills kön är tom:
 - ▶ Ta första bågen (A,F,4) ur kön
 - ▶ F är färgad
 - ▶ Färglägg med F:s färg (fall 2)



$$q = \{ (A,R,4), (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

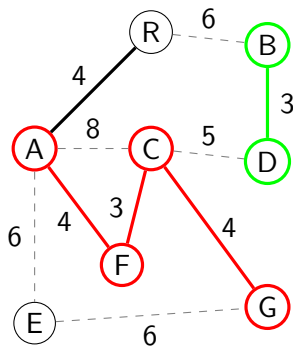
- Upprepa tills kön är tom:



$$q = \{ (A,R,4), (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

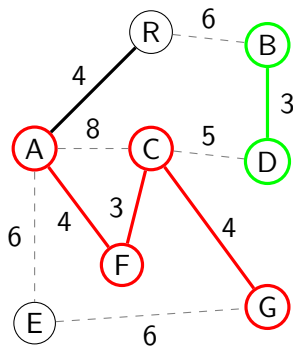
- Upprepa tills kön är tom:
 - Ta första bågen $(A,R,4)$ ur kön



$$q = \{ (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

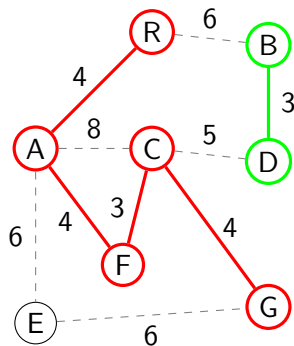
- Upprepa tills kön är tom:
 - Ta första bågen (A,R,4) ur kön
 - A är färgad



$$q = \{ (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

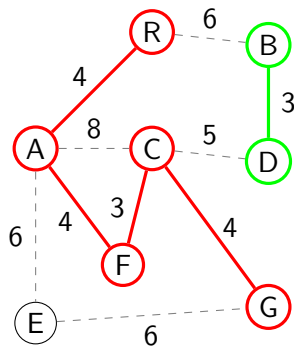
- ▶ Upprepa tills kön är tom:
 - ▶ Ta första bågen (A,R,4) ur kön
 - ▶ A är färgad
 - ▶ Färglägg med A:s färg (fall 2)



$$q = \{ (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

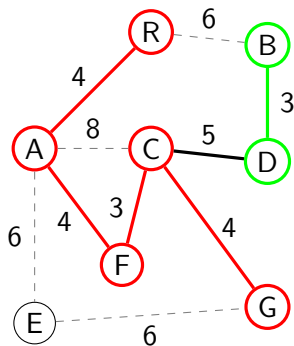
- Upprepa tills kön är tom:



$$q = \{ (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

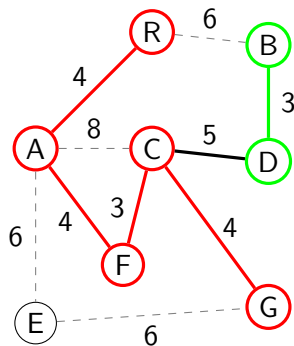
- Upprepa tills kön är tom:
 - Ta första bågen (C,D,5) ur kön



$$q = \{ (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspännande träd, exempel

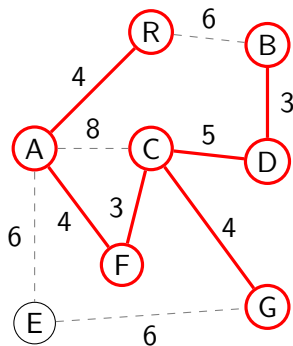
- ▶ Upprepa tills kön är tom:
 - ▶ Ta första bågen (C,D,5) ur kön
 - ▶ C och D färgade med olika färg



$$q = \{ (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

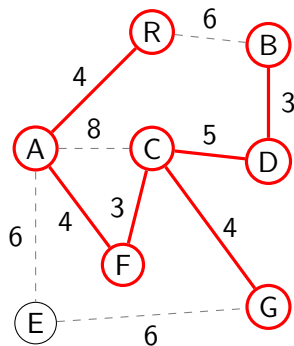
- ▶ Upprepa tills kön är tom:
 - ▶ Ta första bågen (C,D,5) ur kön
 - ▶ C och D färgade med olika färg
 - ▶ Färglägg bägge graferna med C:s färg (fall 4)



$$q = \{ (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

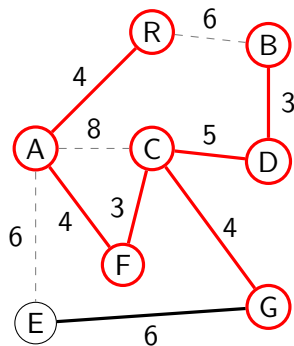
- Upprepa tills kön är tom:



$$q = \{ (E,G,6), (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

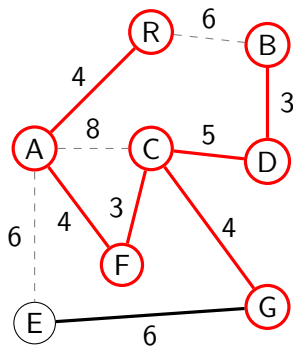
- Upprepa tills kön är tom:
 - Ta första bågen (E,G,6) ur kön



$$q = \{ (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspännande träd, exempel

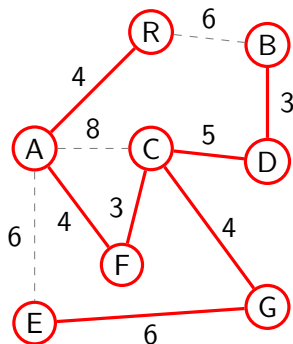
- ▶ Upprepa tills kön är tom:
 - ▶ Ta första bågen (E,G,6) ur kön
 - ▶ G är färgad



$$q = \{ (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

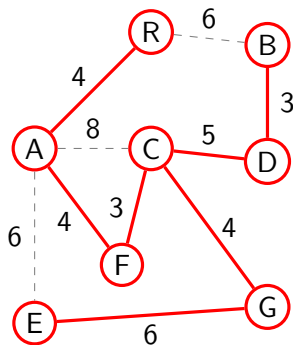
- ▶ Upprepa tills kön är tom:
 - ▶ Ta första bågen (E,G,6) ur kön
 - ▶ G är färgad
 - ▶ Färglägg med G:s färg (fall 2)



$$q = \{ (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspännande träd, exempel

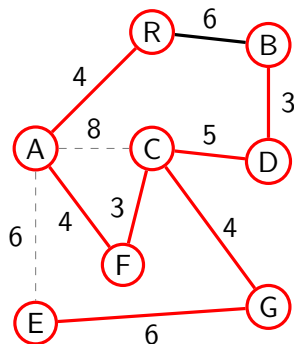
- Upprepa tills kön är tom:



$$q = \{ (B,R,6), (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspännande träd, exempel

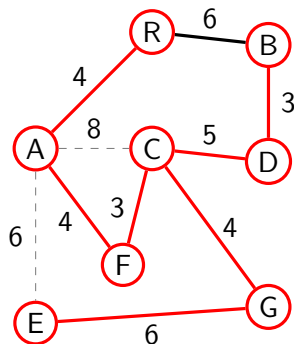
- ▶ Upprepa tills kön är tom:
 - ▶ Ta första bågen $(B,R,6)$ ur kön



$$q = \{ (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

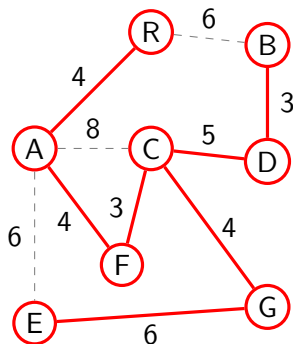
- ▶ Upprepa tills kön är tom:
 - ▶ Ta första bågen $(B,R,6)$ ur kön
 - ▶ Bägge färgade med samma färg



$$q = \{ (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspännande träd, exempel

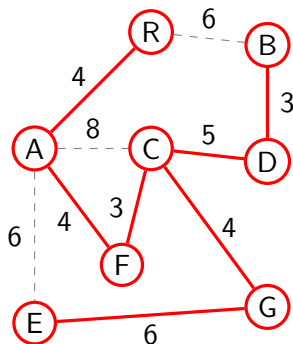
- ▶ Upprepa tills kön är tom:
 - ▶ Ta första bågen (B,R,6) ur kön
 - ▶ Bägge färgade med samma färg
 - ▶ Ignorera bågen (fall 3)



$$q = \{ (A,E,6), (A,C,8) \}$$

Kruskals algoritm för minsta uppspännande träd, exempel

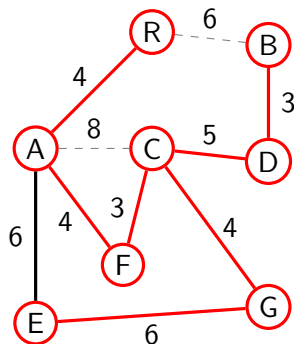
- Upprepa tills kön är tom:



$$q = \{ (A, E, 6), (A, C, 8) \}$$

Kruskals algoritm för minsta uppspännande träd, exempel

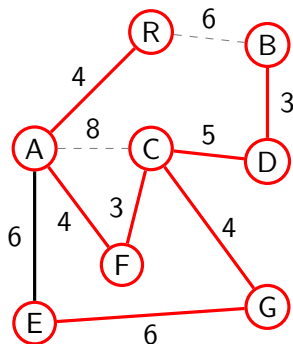
- ▶ Upprepa tills kön är tom:
 - ▶ Ta första bågen $(A,E,6)$ ur kön



$$q = \{ (A,C,8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

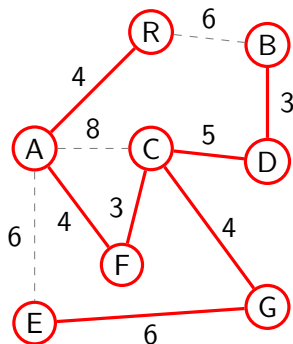
- ▶ Upprepa tills kön är tom:
 - ▶ Ta första bågen (A,E,6) ur kön
 - ▶ Bägge färgade med samma färg



$$q = \{ (A, C, 8) \}$$

Kruskals algoritm för minsta uppspännande träd, exempel

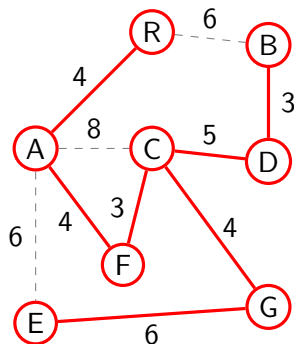
- ▶ Upprepa tills kön är tom:
 - ▶ Ta första bågen (A,E,6) ur kön
 - ▶ Bägge färgade med samma färg
 - ▶ Ignorera bågen (fall 3)



$$q = \{ (A, C, 8) \}$$

Kruskals algoritm för minsta uppspännande träd, exempel

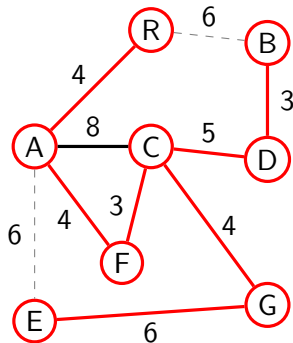
- Upprepa tills kön är tom:



$$q = \{ (A, C, 8) \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

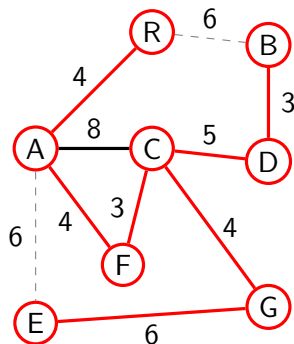
- ▶ Upprepa tills kön är tom:
 - ▶ Ta första bågen (A,C,8) ur kön



$$q = \{ \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

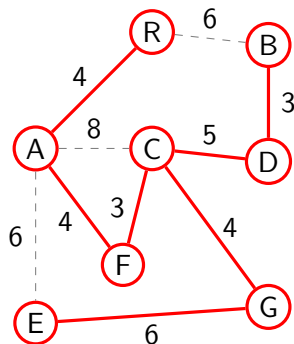
- Upprepa tills kön är tom:
 - Ta första bågen (A,C,8) ur kön
 - Bägge färgade med samma färg



$$q = \{ \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

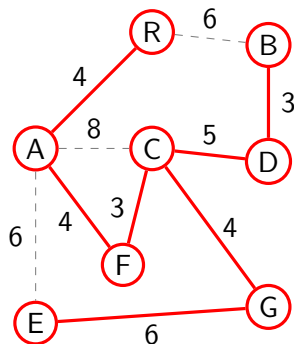
- ▶ Upprepa tills kön är tom:
 - ▶ Ta första bågen (A,C,8) ur kön
 - ▶ Bägge färgade med samma färg
 - ▶ Ignorera bågen (fall 3)



$$q = \{ \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

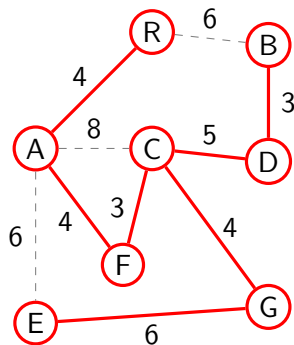
- Upprepa tills kön är tom:



$$q = \{ \}$$

Kruskals algoritm för minsta uppspannande träd, exempel

- ▶ Upprepa tills kön är tom:
- ▶ Klar!



Kruskals algoritm, komplexitet

- ▶ Bygg upp en prioritetskö utifrån en bågmängd
 - ▶ $O(m \log m)$ om heap
- ▶ Varje båge traverseras en gång: $O(m)$:
 - ▶ Hanteringen av bågen kan delas in i fyra fall:
 - ▶ Ingen nod färgad: $O(1)$
 - ▶ En nod färgad: $O(1)$
 - ▶ Noderna samma färg: $O(1)$
 - ▶ Noderna olika färg:
 - ▶ Naiv lösning: $O(n)$
 - ▶ Effektiv lösning $O(1)$
- ▶ Total komplexitet:
 - ▶ $O(m \log m) + O(m) = O(m \log m) = O(m \log n)$

Kruskals algorithm för minsta uppspännande träd, naiv

```
Algorithm Kruskal(g: Graph)
next-color  $\leftarrow$  1; q = Pqueue-empty()
for each node n in g do
    n.color  $\leftarrow$  0
for each edge e in g do
    q  $\leftarrow$  Insert(q,e)
while not Isempy(q) do
    e = (a,b)  $\leftarrow$  Inspect-first(q); q  $\leftarrow$  Delete-first(q)
    if a.color = b.color then // same color
        if a.color = 0 then // uncolored
            a.color  $\leftarrow$  next-color
            b.color  $\leftarrow$  next-color
            next-color  $\leftarrow$  next-color + 1
        else
            // same but color!=0, do nothing
    else // different colors
        if a.color = 0 then // b colored, not a
            a.color  $\leftarrow$  b.color
        else if b.color = 0 then // a colored, not b
            b.color  $\leftarrow$  a.color
        else // both colored with different colors
            for each node n in g do
                if n.color = b.color then
                    n.color  $\leftarrow$  a.color
```

"Omfärgning" av delgraf

- ▶ En naiv algoritm för omfärgning av ett träd/delgraf måste traversera **alla** noderna i delgrafen: $O(n)$
- ▶ Effektivare att definiera om **likhet** för färger
- ▶ Använd ett fält E med **ekvivalenta** färger

Kruskals algorithm för minsta uppspännande träd, effektiv

```
Algorithm Kruskal(g: Graph)
next-color  $\leftarrow$  1; q = Pqueue-empty(); E(0) = 0
for each node n in g do
    n.color  $\leftarrow$  0
for each edge e in g do
    q  $\leftarrow$  Insert(q,e)
while not Isempy(q) do
    e = (a,b)  $\leftarrow$  Inspect-first(q); q  $\leftarrow$  Delete-first(q)
    if E(a.color) = E(b.color) then // same color
        if a.color = 0 then // uncolored
            a.color  $\leftarrow$  next-color
            b.color  $\leftarrow$  next-color
            E(next-color)  $\leftarrow$  next-color
            next-color  $\leftarrow$  nextColor + 1
        else
            // same but color!=0, do nothing
    else // different colors
        if a.color = 0 then // b colored, not a
            a.color  $\leftarrow$  b.color
        else if b.color = 0 then // a colored, not b
            b.color  $\leftarrow$  a.color
        else // both colored with different colors
            E(a.color)  $\leftarrow$  min(E(a.color), E(b.color))
            E(b.color)  $\leftarrow$  min(E(a.color), E(b.color))
```

Fråga

- ▶ Hur fungerar Kruskals algoritm på en icke sammanhängande graf?

Fråga

- ▶ Hur fungerar Kruskals algoritm på en icke sammanhängande graf?
 - ▶ Resultatet blir en skog!

Prims algoritm

Prims algoritm för minsta uppspännande träd (1)

- ▶ Utgå från **godtycklig startnod**
- ▶ I varje steg, bygg på trädet med en båge med **minimal vikt**
- ▶ Använd en **prioritetskö** för att hålla reda på vilka bågar som kan vara aktuella
- ▶ Till slut spänner trädet upp grafen (eller en sammanhängande komponent av den)

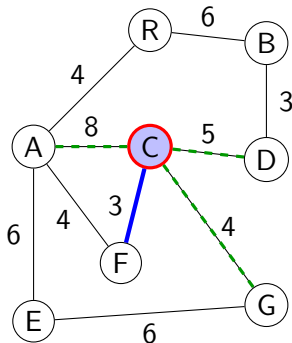
Prims algoritm för minsta uppspännande träd (2)

- ▶ Välj godtycklig **startnod** n ur grafen och låt n bli **rot** i trädet
- ▶ Skapa en tom **prioritetskö** q
- ▶ Upprepa:
 - ▶ Fas 0:
 - ▶ Markera n som **stängd**
 - ▶ Fas 1: Lägg till nya bågar till prioritetskön:
 - ▶ För var och en av de **öppna** (icke-stängda) grannarna w till n :
 - ▶ Lägg bågen (n, w, d) i prioritetskön q
 - ▶ Fas 2: Hitta **bästa** bågen att lägga till trädet:
 - ▶ Upprepa:
 - ▶ Ta första bågen (n, w, d) ur q
 - ▶ Om destinationsnoden w är **öppen**:
 - ▶ Lägg till bågen (n, w, d) till trädet
 - tills w öppen (lagt till en båge) eller q tom (klara)
 - ▶ Fas 3: Gå till den nya noden
 - ▶ Låt $n = w$

tills q är tom

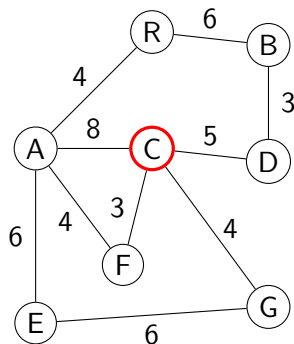
Symboler

- ▶ Stängda noder färgas **ljusblått**
- ▶ Aktuell nod ritas med **röd cirkel**
- ▶ Bågar i **prioritetskön** ritas **grönstreckade**
- ▶ Prioritetskön presenteras sorterad
- ▶ Bågar i den nuvarande **trädet** ritas i **mörkblått**



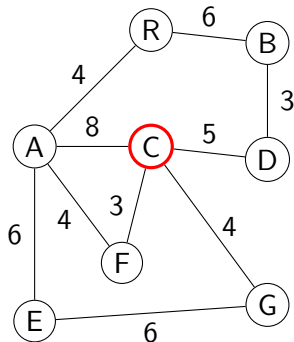
Prims minsta uppspännande träd, exempel

► $n \leftarrow C$.



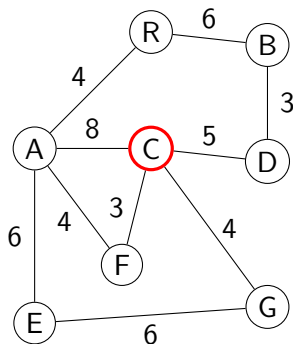
Prims minsta uppspännande träd, exempel

- ▶ $n \leftarrow C$.
- ▶ Låt n blir rot i trädet.



Prims minsta uppspännande träd, exempel

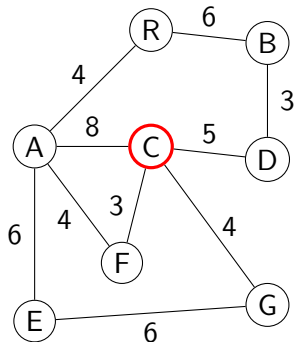
- ▶ $n \leftarrow C$.
- ▶ Låt n blir rot i trädet.
- ▶ Skapa en tom prioritetskö q .



$q = \{ \}$

Prims minsta uppspännande träd, exempel

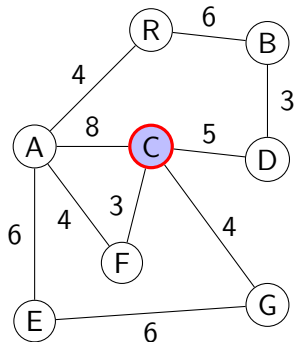
- ▶ $n \leftarrow C$.
- ▶ Låt n blir rot i trädet.
- ▶ Skapa en tom prioritetsskö q .
- ▶ Upprepa:



$q = \{ \}$

Prims minsta uppspännande träd, exempel

- Upprepa:
 - Fas 0: Markera C som stängd.

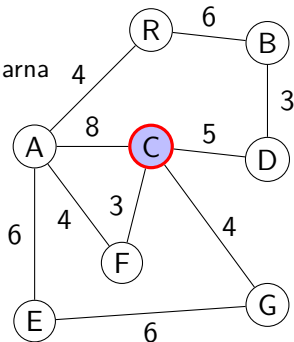


$q = \{ \}$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera C som stängd.
- Fas 1: För var och en av de öppna grannarna {A,F,G,D} till C:

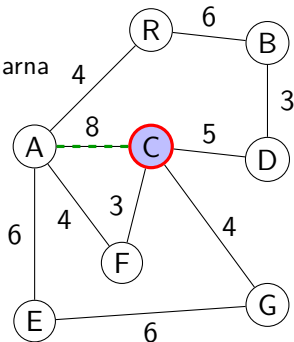


$q = \{ \}$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera C som stängd.
- Fas 1: För var och en av de öppna grannarna {A,F,G,D} till C:
 - Lägg (C,A,8) till q .

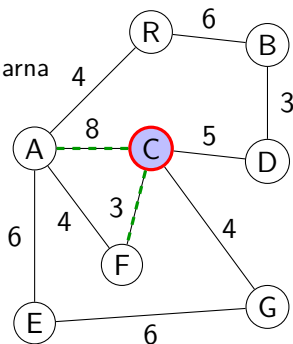


$$q = \{ (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera C som stängd.
- Fas 1: För var och en av de öppna grannarna {A,F,G,D} till C:
 - Lägg (C,F,3) till q .

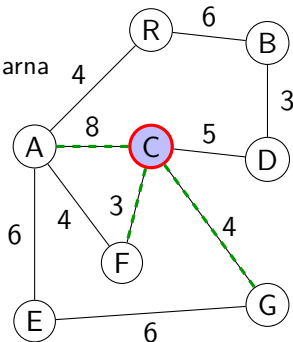


$$q = \{ (C,F,3), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera C som stängd.
- Fas 1: För var och en av de öppna grannarna {A,F,G,D} till C:
 - Lägg (C,G,4) till q .

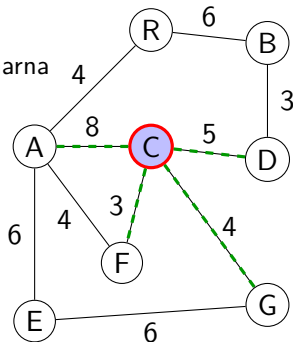


$$q = \{ (C,F,3), (C,G,4), (C,A,8) \}$$

Prims minsta uppspannande träd, exempel

► Upprepa:

- Fas 0: Markera C som stängd.
- Fas 1: För var och en av de öppna grannarna {A,F,G,D} till C:
 - Lägg (C,D,5) till q .

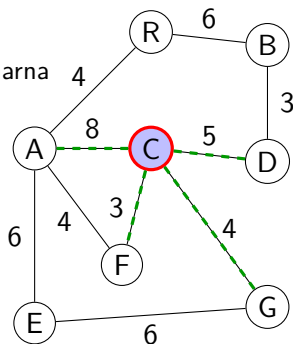


$$q = \{ (C,F,3), (C,G,4), (C,D,5), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera C som stängd.
- Fas 1: För var och en av de öppna grannarna {A,F,G,D} till C:
- Fas 2: Upprepa

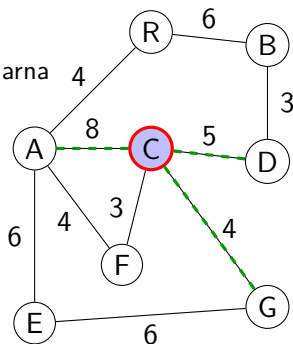


$$q = \{ (C,F,3), (C,G,4), (C,D,5), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera C som stängd.
- Fas 1: För var och en av de öppna grannarna {A,F,G,D} till C:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, F, 3)$ från q .

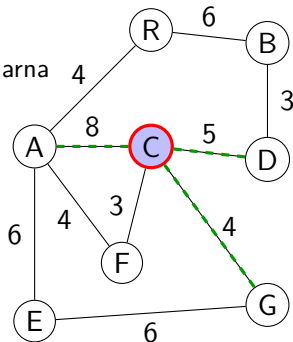


$$q = \{ (C, G, 4), (C, D, 5), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera C som stängd.
- Fas 1: För var och en av de öppna grannarna {A,F,G,D} till C:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, F, 3)$ från q .
 - F ej stängd.

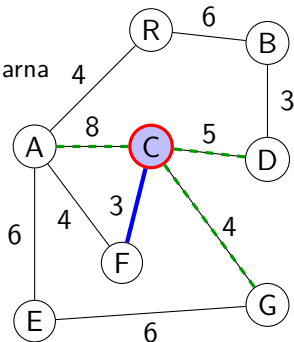


$$q = \{ (C, G, 4), (C, D, 5), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera C som stängd.
- Fas 1: För var och en av de öppna grannarna {A,F,G,D} till C:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, F, 3)$ från q .
 - F ej stängd.
 - Lägg $(C, F, 3)$ till trädet.

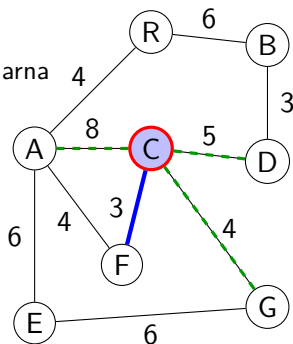


$$q = \{ (C, G, 4), (C, D, 5), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera C som stängd.
- Fas 1: För var och en av de öppna grannarna {A,F,G,D} till C:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, F, 3)$ från q .
 - F ej stängd.
 - Lägg $(C, F, 3)$ till träd.
- tills F ej stängd eller q är tom.

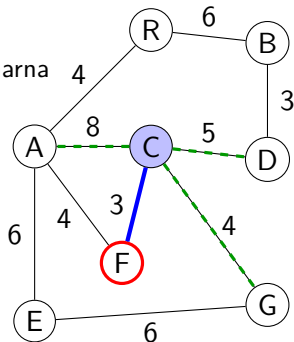


$$q = \{ (C, G, 4), (C, D, 5), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera C som stängd.
- Fas 1: För var och en av de öppna grannarna $\{A, F, G, D\}$ till C:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, F, 3)$ från q .
 - F ej stängd.
 - Lägg $(C, F, 3)$ till träd.
- tills F ej stängd eller q är tom.
- Fas 3: $n \leftarrow F$.



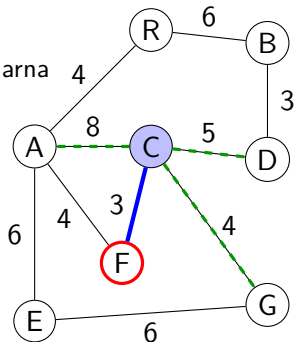
$$q = \{ (C, G, 4), (C, D, 5), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera C som stängd.
- Fas 1: För var och en av de öppna grannarna $\{A, F, G, D\}$ till C:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, F, 3)$ från q .
 - F ej stängd.
 - Lägg $(C, F, 3)$ till träd.
 - tills F ej stängd eller q är tom.
- Fas 3: $n \leftarrow F$.

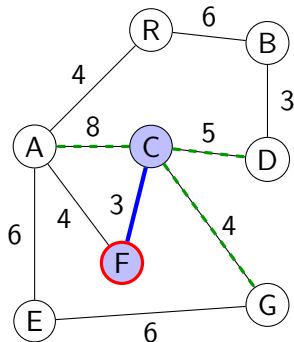
► tills q är tom.



$$q = \{ (C, G, 4), (C, D, 5), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

- ▶ Upprepa:
 - ▶ Fas 0: Markera F som stängd.

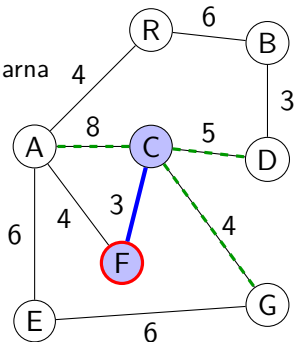


$$q = \{ (C,G,4), (C,D,5), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera F som stängd.
- Fas 1: För var och en av de öppna grannarna {A} till F:

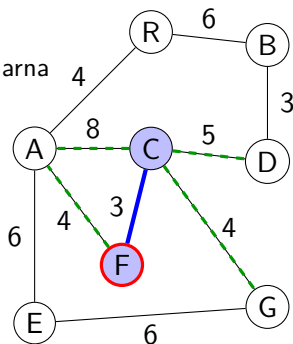


$$q = \{ (C,G,4), (C,D,5), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera F som stängd.
- Fas 1: För var och en av de öppna grannarna {A} till F:
 - Lägg (F,A,4) till q .

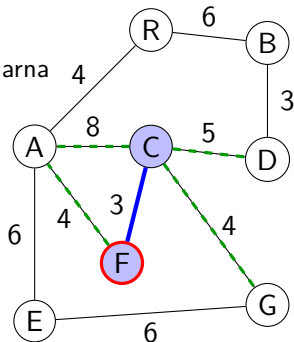


$$q = \{ (F,A,4), (C,G,4), (C,D,5), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera F som stängd.
- Fas 1: För var och en av de öppna grannarna {A} till F:
- Fas 2: Upprepa

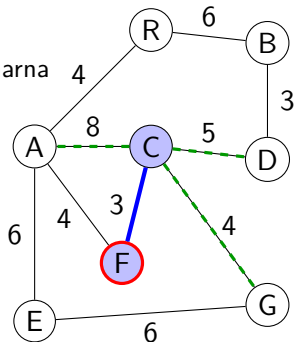


$$q = \{ (F,A,4), (C,G,4), (C,D,5), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera F som stängd.
- Fas 1: För var och en av de öppna grannarna {A} till F:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (F, A, 4)$ från q .

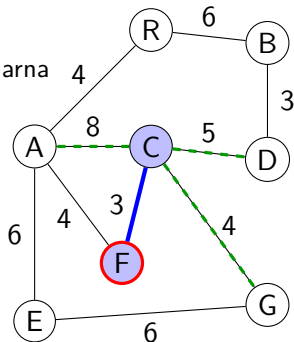


$$q = \{ (C, G, 4), (C, D, 5), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera F som stängd.
- Fas 1: För var och en av de öppna grannarna {A} till F:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (F, A, 4)$ från q .
 - A ej stängd.

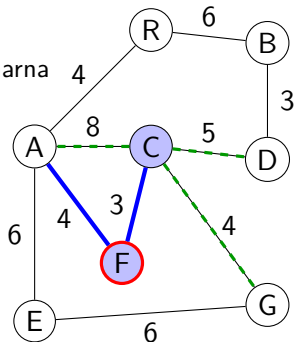


$$q = \{ (C, G, 4), (C, D, 5), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera F som stängd.
- Fas 1: För var och en av de öppna grannarna {A} till F:
 - Fas 2: Upprepa
 - Ta $(n, w, d) = (F, A, 4)$ från q .
 - A ej stängd.
 - Lägg $(F, A, 4)$ till trädet.

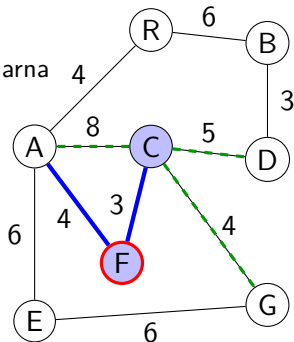


$$q = \{ (C, G, 4), (C, D, 5), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera F som stängd.
- Fas 1: För var och en av de öppna grannarna {A} till F:
 - Fas 2: Upprepa
 - Ta $(n, w, d) = (F, A, 4)$ från q .
 - A ej stängd.
 - Lägg $(F, A, 4)$ till träd.
- tills A ej stängd eller q är tom.

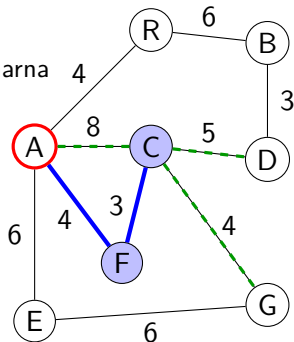


$$q = \{ (C, G, 4), (C, D, 5), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

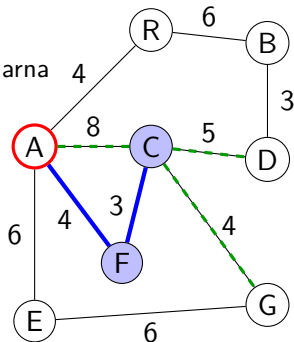
- Fas 0: Markera F som stängd.
- Fas 1: För var och en av de öppna grannarna {A} till F:
 - Fas 2: Upprepa
 - Ta $(n, w, d) = (F, A, 4)$ från q .
 - A ej stängd.
 - Lägg $(F, A, 4)$ till träd.
- tills A ej stängd eller q är tom.
- Fas 3: $n \leftarrow A$.



$$q = \{ (C, G, 4), (C, D, 5), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

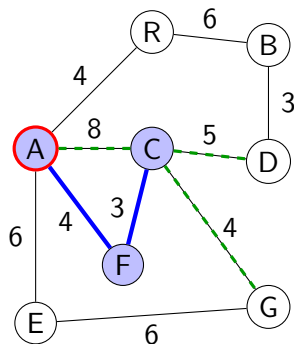
- ▶ Upprepa:
 - ▶ Fas 0: Markera F som stängd.
 - ▶ Fas 1: För var och en av de öppna grannarna {A} till F:
 - ▶ Fas 2: Upprepa
 - ▶ Ta $(n, w, d) = (F, A, 4)$ från q .
 - ▶ A ej stängd.
 - ▶ Lägg $(F, A, 4)$ till trädet.
 - ▶ tills A ej stängd eller q är tom.
 - ▶ Fas 3: $n \leftarrow A$.
 - ▶ tills q är tom.



$$q = \{ (C, G, 4), (C, D, 5), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

- ▶ Upprepa:
 - ▶ Fas 0: Markera A som stängd.

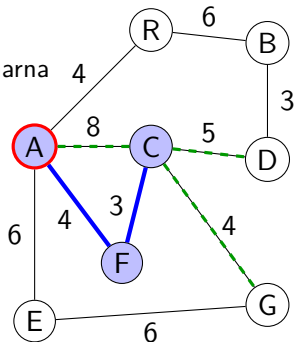


$$q = \{ (C,G,4), (C,D,5), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera A som stängd.
- Fas 1: För var och en av de öppna grannarna {R,E} till A:

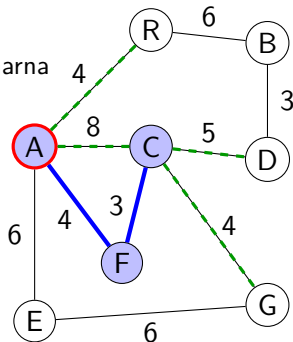


$$q = \{ (C,G,4), (C,D,5), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera A som stängd.
- Fas 1: För var och en av de öppna grannarna {R,E} till A:
 - Lägg (A,R,4) till q .

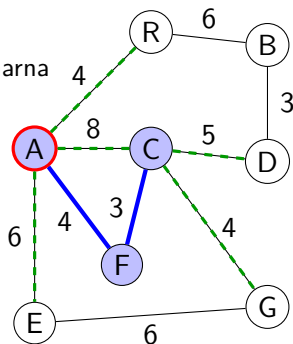


$$q = \{ (A,R,4), (C,G,4), (C,D,5), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera A som stängd.
- Fas 1: För var och en av de öppna grannarna {R,E} till A:
 - Lägg (A,E,6) till q .

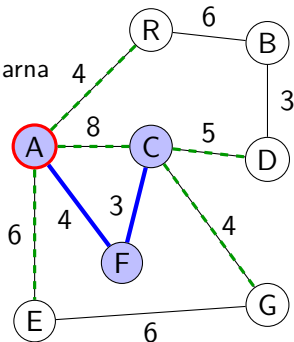


$$q = \{ (A,R,4), (C,G,4), (C,D,5), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera A som stängd.
- Fas 1: För var och en av de öppna grannarna {R,E} till A:
- Fas 2: Upprepa

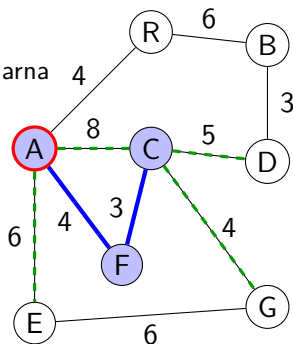


$$q = \{ (A,R,4), (C,G,4), (C,D,5), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera A som stängd.
- Fas 1: För var och en av de öppna grannarna $\{R,E\}$ till A:
 - Fas 2: Upprepa
 - Ta $(n, w, d) = (A, R, 4)$ från q .

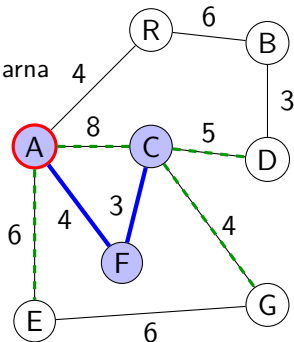


$$q = \{ (C, G, 4), (C, D, 5), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera A som stängd.
- Fas 1: För var och en av de öppna grannarna {R,E} till A:
 - Fas 2: Upprepa
 - Ta $(n, w, d) = (A, R, 4)$ från q .
 - R ej stängd.

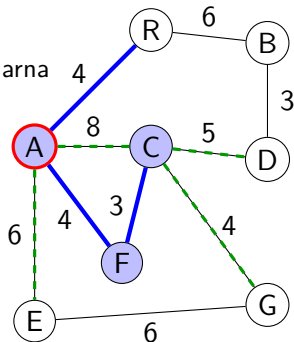


$$q = \{ (C, G, 4), (C, D, 5), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera A som stängd.
- Fas 1: För var och en av de öppna grannarna {R,E} till A:
 - Fas 2: Upprepa
 - Ta $(n, w, d) = (A, R, 4)$ från q .
 - R ej stängd.
 - Lägg $(A, R, 4)$ till trädet.

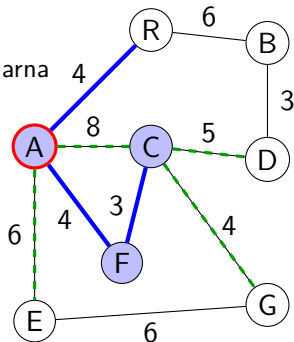


$$q = \{ (C, G, 4), (C, D, 5), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera A som stängd.
- Fas 1: För var och en av de öppna grannarna $\{R,E\}$ till A:
 - Fas 2: Upprepa
 - Ta $(n, w, d) = (A, R, 4)$ från q .
 - R ej stängd.
 - Lägg $(A, R, 4)$ till träd.
 - tills R ej stängd eller q är tom.

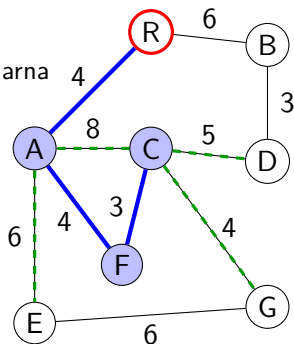


$$q = \{ (C, G, 4), (C, D, 5), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

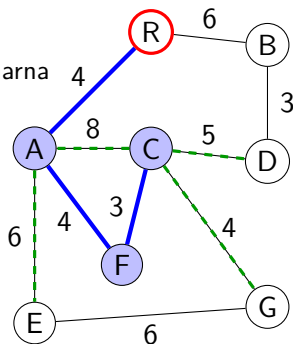
- Fas 0: Markera A som stängd.
- Fas 1: För var och en av de öppna grannarna {R,E} till A:
 - Fas 2: Upprepa
 - Ta $(n, w, d) = (A, R, 4)$ från q .
 - R ej stängd.
 - Lägg $(A, R, 4)$ till träd.
- tills R ej stängd eller q är tom.
- Fas 3: $n \leftarrow R$.



$$q = \{ (C, G, 4), (C, D, 5), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

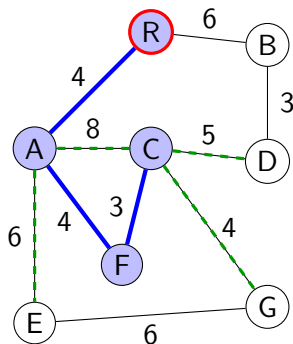
- ▶ Upprepa:
 - ▶ Fas 0: Markera A som stängd.
 - ▶ Fas 1: För var och en av de öppna grannarna {R,E} till A:
 - ▶ Fas 2: Upprepa
 - ▶ Ta $(n, w, d) = (A, R, 4)$ från q .
 - ▶ R ej stängd.
 - ▶ Lägg $(A, R, 4)$ till trädet.
 - ▶ tills R ej stängd eller q är tom.
 - ▶ Fas 3: $n \leftarrow R$.
 - ▶ tills q är tom.



$$q = \{ (C, G, 4), (C, D, 5), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

- Upprepa:
 - Fas 0: Markera R som stängd.

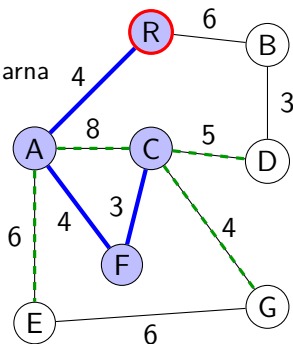


$$q = \{ (C,G,4), (C,D,5), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera R som stängd.
- Fas 1: För var och en av de öppna grannarna {B} till R:

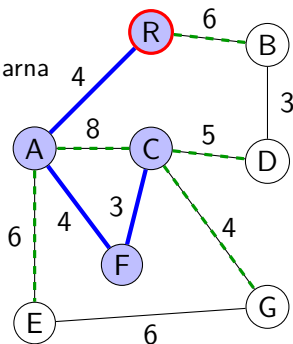


$$q = \{ (C, G, 4), (C, D, 5), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera R som stängd.
- Fas 1: För var och en av de öppna grannarna {B} till R:
 - Lägg (R,B,6) till q .

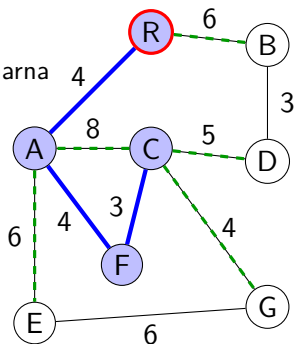


$$q = \{ (C,G,4), (C,D,5), (R,B,6), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera R som stängd.
- Fas 1: För var och en av de öppna grannarna {B} till R:
- Fas 2: Upprepa

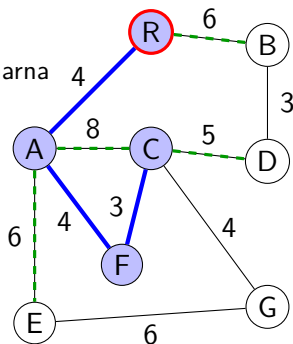


$$q = \{ (C,G,4), (C,D,5), (R,B,6), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera R som stängd.
- Fas 1: För var och en av de öppna grannarna $\{B\}$ till R:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, G, 4)$ från q .

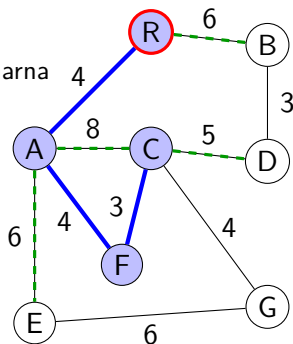


$$q = \{ (C, D, 5), (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera R som stängd.
- Fas 1: För var och en av de öppna grannarna {B} till R:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, G, 4)$ från q .
 - G ej stängd.

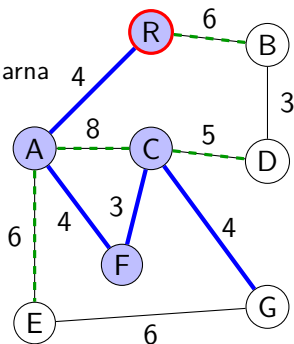


$$q = \{ (C, D, 5), (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera R som stängd.
- Fas 1: För var och en av de öppna grannarna $\{B\}$ till R:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, G, 4)$ från q .
 - G ej stängd.
 - Lägg $(C, G, 4)$ till träd.

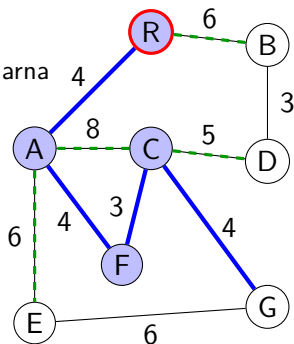


$$q = \{ (C, D, 5), (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera R som stängd.
- Fas 1: För var och en av de öppna grannarna {B} till R:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, G, 4)$ från q .
 - G ej stängd.
 - Lägg $(C, G, 4)$ till trädets.
- tills G ej stängd eller q är tom.

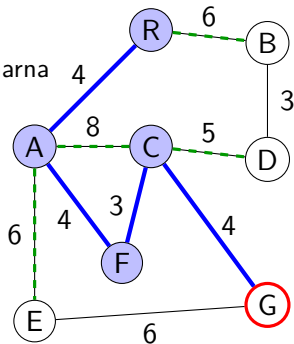


$$q = \{ (C, D, 5), (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera R som stängd.
- Fas 1: För var och en av de öppna grannarna {B} till R:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, G, 4)$ från q .
 - G ej stängd.
 - Lägg $(C, G, 4)$ till träd.
- tills G ej stängd eller q är tom.
- Fas 3: $n \leftarrow G$.



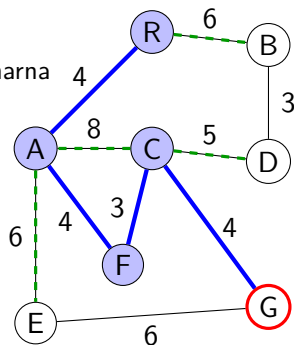
$$q = \{ (C, D, 5), (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera R som stängd.
- Fas 1: För var och en av de öppna grannarna {B} till R:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, G, 4)$ från q .
 - G ej stängd.
 - Lägg $(C, G, 4)$ till trädets.
- tills G ej stängd eller q är tom.
- Fas 3: $n \leftarrow G$.

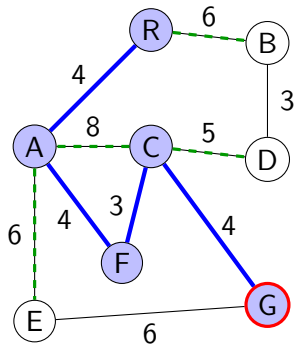
► tills q är tom.



$$q = \{ (C, D, 5), (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

- Upprepa:
 - Fas 0: Markera G som stängd.

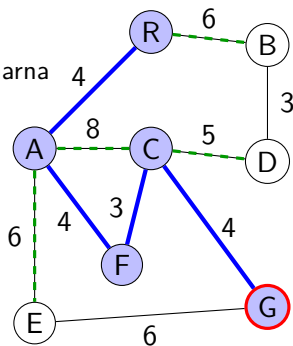


$$q = \{ (C,D,5), (R,B,6), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera G som stängd.
- Fas 1: För var och en av de öppna grannarna {E} till G:

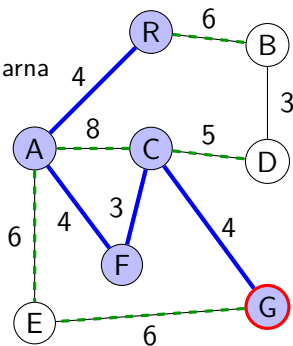


$$q = \{ (C,D,5), (R,B,6), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera G som stängd.
- Fas 1: För var och en av de öppna grannarna {E} till G:
 - Lägg (G,E,6) till q .

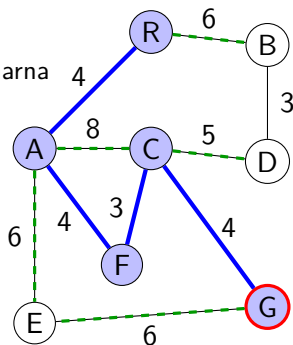


$$q = \{ (C,D,5), (G,E,6), (R,B,6), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera G som stängd.
- Fas 1: För var och en av de öppna grannarna {E} till G:
- Fas 2: Upprepa

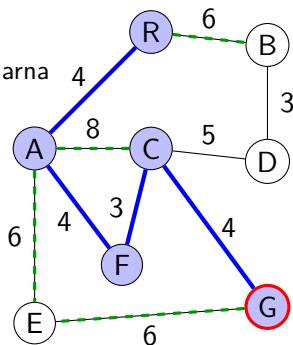


$$q = \{ (C,D,5), (G,E,6), (R,B,6), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera G som stängd.
- Fas 1: För var och en av de öppna grannarna $\{E\}$ till G:
- Fas 2: Upprepa
 - Ta $(n, w, d)=(C,D,5)$ från q .

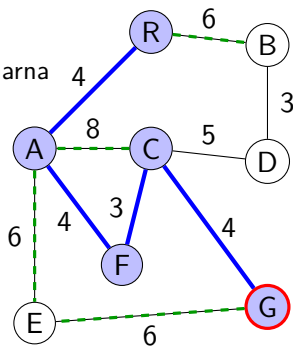


$$q = \{ (G,E,6), (R,B,6), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera G som stängd.
- Fas 1: För var och en av de öppna grannarna {E} till G:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, D, 5)$ från q .
 - D ej stängd.

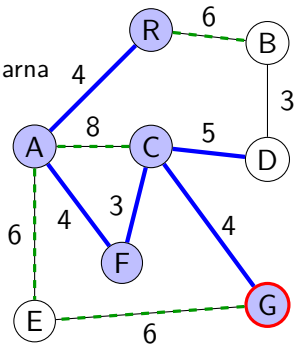


$$q = \{ (G, E, 6), (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera G som stängd.
- Fas 1: För var och en av de öppna grannarna {E} till G:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, D, 5)$ från q .
 - D ej stängd.
 - Lägg $(C, D, 5)$ till trädet.

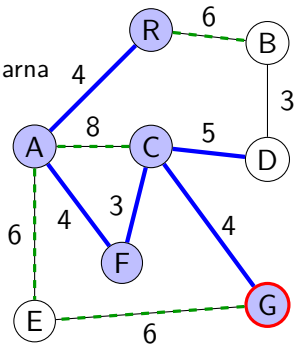


$$q = \{ (G, E, 6), (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera G som stängd.
- Fas 1: För var och en av de öppna grannarna {E} till G:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, D, 5)$ från q .
 - D ej stängd.
 - Lägg $(C, D, 5)$ till trädets.
- tills D ej stängd eller q är tom.

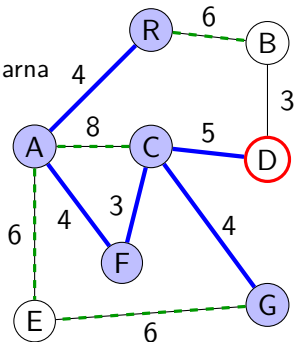


$$q = \{ (G, E, 6), (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera G som stängd.
- Fas 1: För var och en av de öppna grannarna {E} till G:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, D, 5)$ från q .
 - D ej stängd.
 - Lägg $(C, D, 5)$ till trädets.
- tills D ej stängd eller q är tom.
- Fas 3: $n \leftarrow D$.



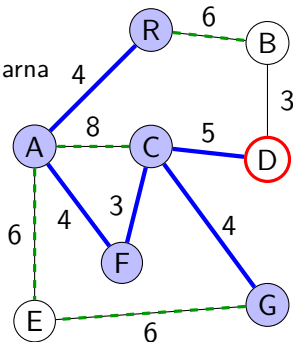
$$q = \{ (G, E, 6), (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera G som stängd.
- Fas 1: För var och en av de öppna grannarna {E} till G:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, D, 5)$ från q .
 - D ej stängd.
 - Lägg $(C, D, 5)$ till trädets.
 - tills D ej stängd eller q är tom.
 - Fas 3: $n \leftarrow D$.

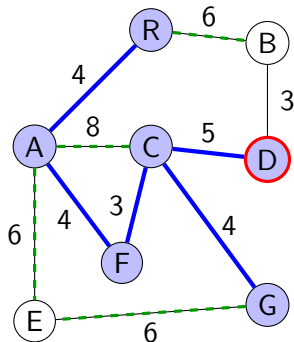
► tills q är tom.



$$q = \{ (G, E, 6), (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

- Upprepa:
 - Fas 0: Markera D som stängd.

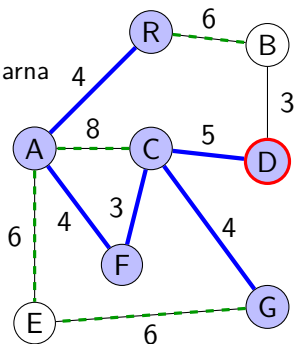


$$q = \{ (G,E,6), (R,B,6), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera D som stängd.
- Fas 1: För var och en av de öppna grannarna {B} till D:

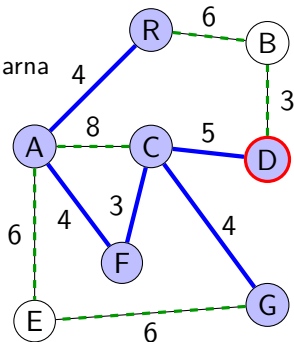


$$q = \{ (G,E,6), (R,B,6), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera D som stängd.
- Fas 1: För var och en av de öppna grannarna {B} till D:
 - Lägg $(D,B,3)$ till q .

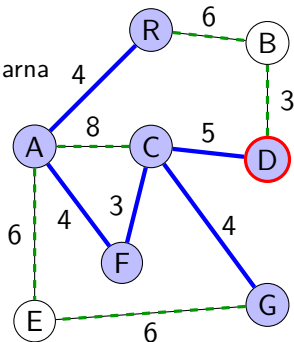


$$q = \{ (D,B,3), (G,E,6), (R,B,6), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera D som stängd.
- Fas 1: För var och en av de öppna grannarna {B} till D:
- Fas 2: Upprepa

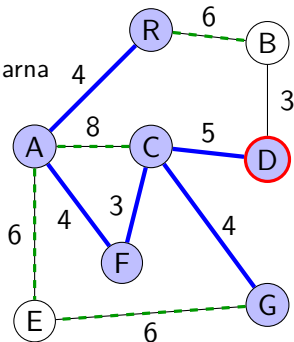


$$q = \{ (D,B,3), (G,E,6), (R,B,6), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera D som stängd.
- Fas 1: För var och en av de öppna grannarna $\{B\}$ till D:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (D, B, 3)$ från q .

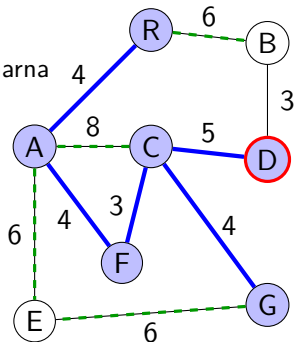


$$q = \{ (G, E, 6), (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera D som stängd.
- Fas 1: För var och en av de öppna grannarna $\{B\}$ till D:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (D, B, 3)$ från q .
 - B ej stängd.

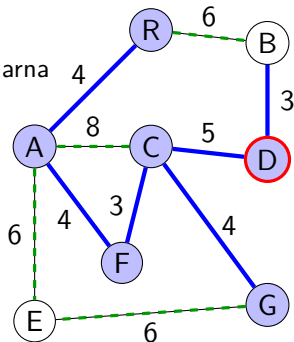


$$q = \{ (G, E, 6), (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera D som stängd.
- Fas 1: För var och en av de öppna grannarna $\{B\}$ till D:
 - Fas 2: Upprepa
 - Ta $(n, w, d) = (D, B, 3)$ från q .
 - B ej stängd.
 - Lägg $(D, B, 3)$ till trädet.

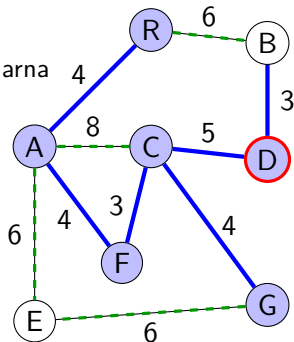


$$q = \{ (G, E, 6), (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera D som stängd.
- Fas 1: För var och en av de öppna grannarna $\{B\}$ till D:
 - Fas 2: Upprepa
 - Ta $(n, w, d) = (D, B, 3)$ från q .
 - B ej stängd.
 - Lägg $(D, B, 3)$ till trädets.
- tills B ej stängd eller q är tom.

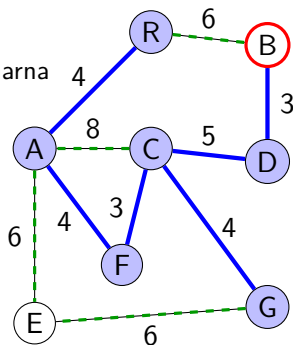


$$q = \{ (G, E, 6), (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera D som stängd.
- Fas 1: För var och en av de öppna grannarna $\{B\}$ till D:
 - Fas 2: Upprepa
 - Ta $(n, w, d) = (D, B, 3)$ från q .
 - B ej stängd.
 - Lägg $(D, B, 3)$ till trädets.
- tills B ej stängd eller q är tom.
- Fas 3: $n \leftarrow B$.



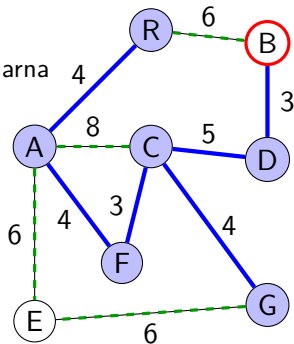
$$q = \{ (G, E, 6), (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera D som stängd.
- Fas 1: För var och en av de öppna grannarna $\{B\}$ till D:
 - Fas 2: Upprepa
 - Ta $(n, w, d) = (D, B, 3)$ från q .
 - B ej stängd.
 - Lägg $(D, B, 3)$ till trädets.
 - tills B ej stängd eller q är tom.
 - Fas 3: $n \leftarrow B$.

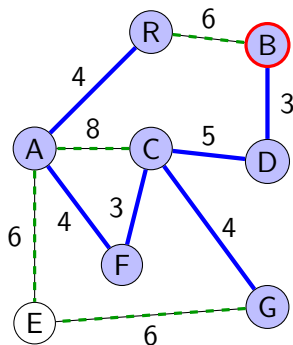
► tills q är tom.



$$q = \{ (G, E, 6), (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

- Upprepa:
 - Fas 0: Markera B som stängd.

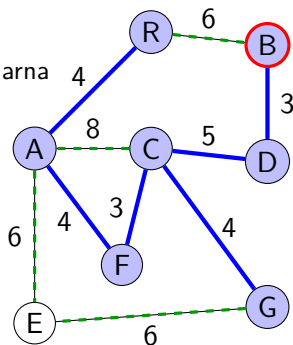


$$q = \{ (G,E,6), (R,B,6), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera B som stängd.
- Fas 1: För var och en av de öppna grannarna { } till B:

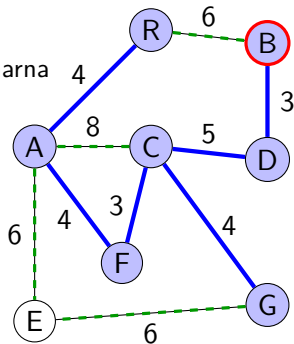


$$q = \{ (G,E,6), (R,B,6), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera B som stängd.
- Fas 1: För var och en av de öppna grannarna $\{ \}$ till B:
- Fas 2: Upprepa

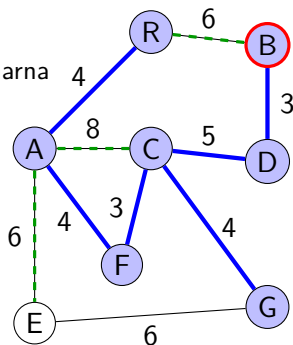


$$q = \{ (G,E,6), (R,B,6), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera B som stängd.
- Fas 1: För var och en av de öppna grannarna $\{ \}$ till B:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (G, E, 6)$ från q .

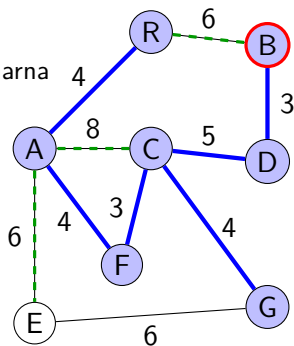


$$q = \{ (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera B som stängd.
- Fas 1: För var och en av de öppna grannarna $\{ \}$ till B:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (G, E, 6)$ från q .
 - E ej stängd.

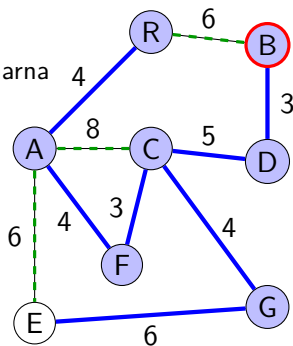


$$q = \{ (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera B som stängd.
- Fas 1: För var och en av de öppna grannarna $\{ \}$ till B:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (G, E, 6)$ från q .
 - E ej stängd.
 - Lägg $(G, E, 6)$ till träd.

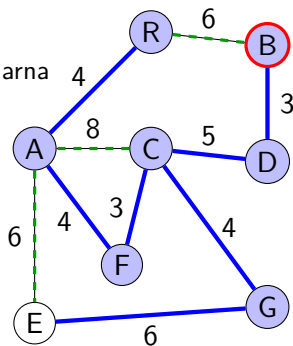


$$q = \{ (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera B som stängd.
- Fas 1: För var och en av de öppna grannarna $\{ \}$ till B:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (G, E, 6)$ från q .
 - E ej stängd.
 - Lägg $(G, E, 6)$ till trädet.
- tills E ej stängd eller q är tom.

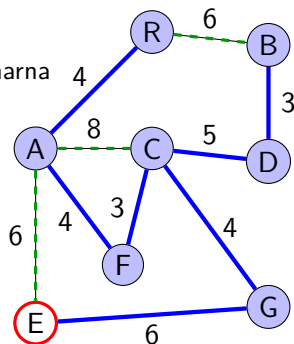


$$q = \{ (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera B som stängd.
- Fas 1: För var och en av de öppna grannarna $\{ \}$ till B:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (G, E, 6)$ från q .
 - E ej stängd.
 - Lägg $(G, E, 6)$ till träd.
- tills E ej stängd eller q är tom.
- Fas 3: $n \leftarrow E$.



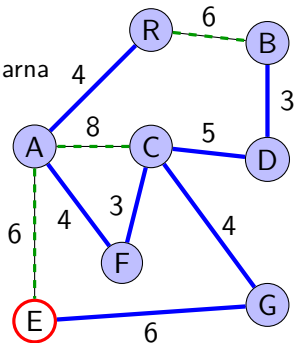
$$q = \{ (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera B som stängd.
- Fas 1: För var och en av de öppna grannarna $\{ \}$ till B:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (G, E, 6)$ från q .
 - E ej stängd.
 - Lägg $(G, E, 6)$ till trädets.
 - tills E ej stängd eller q är tom.
 - Fas 3: $n \leftarrow E$.

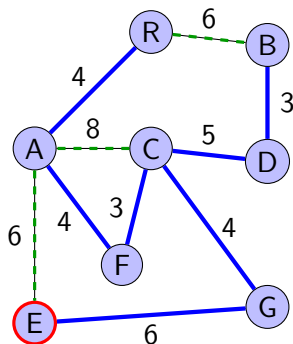
► tills q är tom.



$$q = \{ (R, B, 6), (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

- Upprepa:
 - Fas 0: Markera E som stängd.

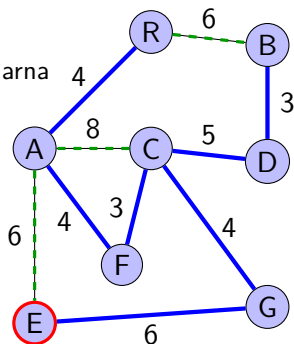


$$q = \{ (R,B,6), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera E som stängd.
- Fas 1: För var och en av de öppna grannarna { } till E:

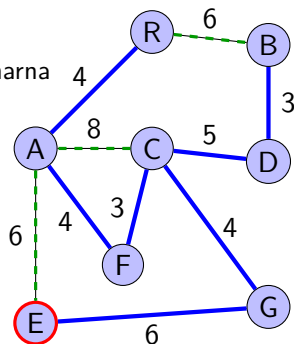


$$q = \{ (R,B,6), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera E som stängd.
- Fas 1: För var och en av de öppna grannarna { } till E:
- Fas 2: Upprepa

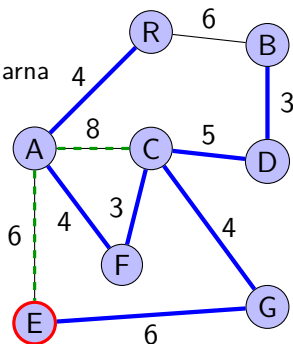


$$q = \{ (R,B,6), (A,E,6), (C,A,8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera E som stängd.
- Fas 1: För var och en av de öppna grannarna $\{ \}$ till E:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (R, B, 6)$ från q .

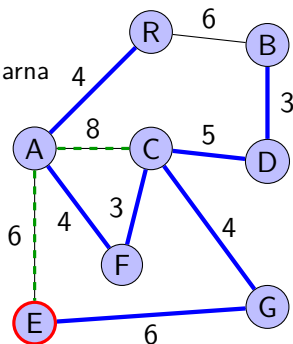


$$q = \{ (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera E som stängd.
- Fas 1: För var och en av de öppna grannarna $\{ \}$ till E:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (R, B, 6)$ från q .
 - B stängd.

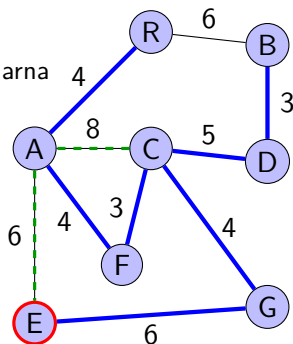


$$q = \{ (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera E som stängd.
- Fas 1: För var och en av de öppna grannarna $\{ \}$ till E:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (R, B, 6)$ från q .
 - B stängd.
- tills B ej stängd eller q är tom.

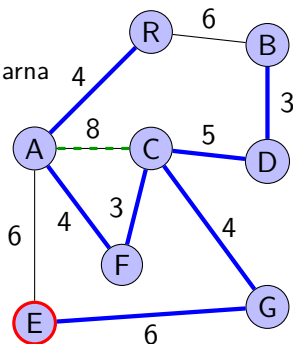


$$q = \{ (A, E, 6), (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera E som stängd.
- Fas 1: För var och en av de öppna grannarna $\{ \}$ till E:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (A, E, 6)$ från q .

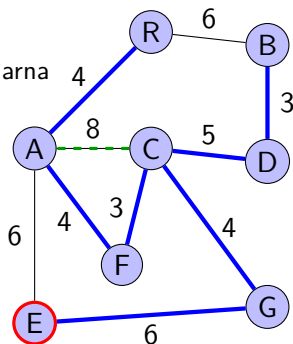


$$q = \{ (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera E som stängd.
- Fas 1: För var och en av de öppna grannarna $\{ \}$ till E:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (A, E, 6)$ från q .
 - E stängd.

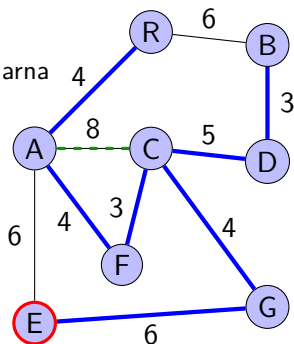


$$q = \{ (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera E som stängd.
- Fas 1: För var och en av de öppna grannarna $\{ \}$ till E:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (A, E, 6)$ från q .
 - E stängd.
- tills E ej stängd eller q är tom.

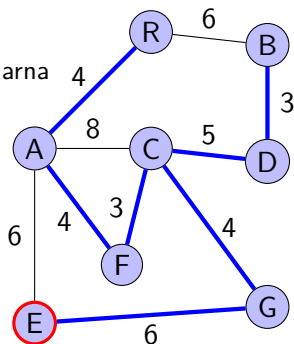


$$q = \{ (C, A, 8) \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera E som stängd.
- Fas 1: För var och en av de öppna grannarna $\{ \}$ till E:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, A, 8)$ från q .

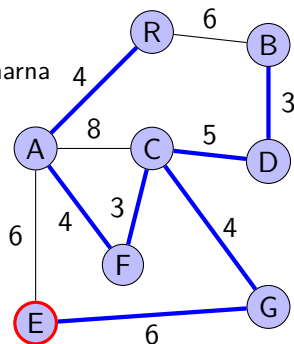


$$q = \{ \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera E som stängd.
- Fas 1: För var och en av de öppna grannarna $\{ \}$ till E:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, A, 8)$ från q .
 - A stängd.

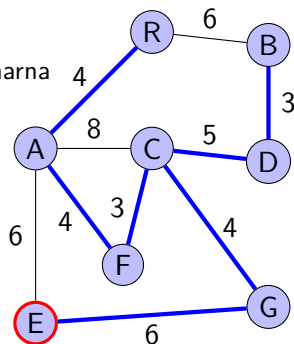


$$q = \{ \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

- Fas 0: Markera E som stängd.
- Fas 1: För var och en av de öppna grannarna $\{ \}$ till E:
- Fas 2: Upprepa
 - Ta $(n, w, d) = (C, A, 8)$ från q .
 - A stängd.
- tills A ej stängd eller q är tom.

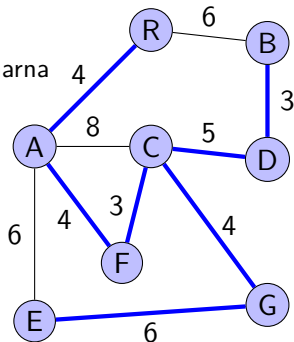


$$q = \{ \}$$

Prims minsta uppspännande träd, exempel

► Upprepa:

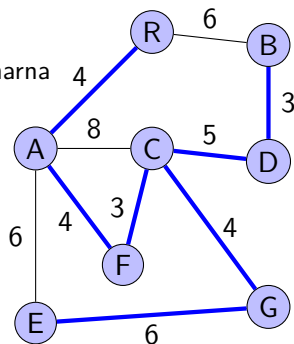
- Fas 0: Markera E som stängd.
- Fas 1: För var och en av de öppna grannarna $\{ \}$ till E:
- Fas 2: Upprepa
- tills A ej stängd eller q är tom.
- Fas 3: $n \leftarrow A$.



$$q = \{ \}$$

Prims minsta uppspännande träd, exempel

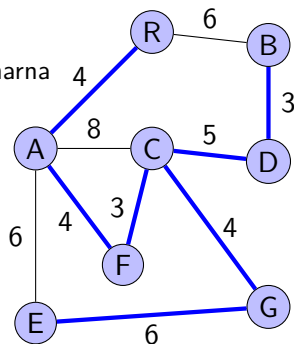
- ▶ Upprepa:
 - ▶ Fas 0: Markera E som stängd.
 - ▶ Fas 1: För var och en av de öppna grannarna $\{ \}$ till E:
 - ▶ Fas 2: Upprepa
 - ▶ tills A ej stängd eller q är tom.
 - ▶ Fas 3: $n \leftarrow A$.
- ▶ tills q är tom.



$$q = \{ \}$$

Prims minsta uppspännande träd, exempel

- ▶ Upprepa:
 - ▶ Fas 0: Markera E som stängd.
 - ▶ Fas 1: För var och en av de öppna grannarna $\{ \}$ till E:
 - ▶ Fas 2: Upprepa
 - ▶ tills A ej stängd eller q är tom.
 - ▶ Fas 3: $n \leftarrow A$.
- ▶ tills q är tom.
- ▶ Klar!



$q = \{ \}$

Prims algoritm för minsta uppspännande träd (igen)

- ▶ Välj godtycklig startnod n ur grafen och låt n bli rot i trädet
 - ▶ Skapa en tom prioritetskö q
 - ▶ Upprepa:
 - ▶ Markera n som stängd
 - ▶ För var och en av de öppna grannarna w till n :
 - ▶ Lägg bågen (n, w, d) i prioritetkön q
 - ▶ Upprepa:
 - ▶ Ta första bågen (n, w, d) ur q
 - ▶ Om destinationsnoden w ej är stängd:
 - ▶ Lägg till bågen (n, w, d) till trädettills w ej stängd eller q är tom
 - ▶ Låt $n = w$
- tills
- q
- är tom
- ▶ Vad blir komplexiteten?

Prims algoritm, komplexitet

- ▶ Man gör en traversering av grafen, dvs. $O(m) + O(n)$
- ▶ Sen tillkommer köoperationer:
 - ▶ För varje båge:
 - ▶ Sätt in ett element i prioritetsskön
 - ▶ Inspektera elementet
 - ▶ Ta ut elementet
 - ▶ Komplexitet: $O(m)$ (lista) eller $O(\log m)$ (heap).
- ▶ Totalt: $O(n) + O(m^2)$ eller $O(n) + O(m \log m)$

Fråga

- ▶ Hur fungerar Prims algoritm på en icke sammanhängande graf?

Fråga

- ▶ Hur fungerar Prims algoritm på en icke sammanhängande graf?
 - ▶ Vi får ett träd som spänner upp den sammanhängande komponent som startnoden ingick i