F12 - Grafalgoritmer

5DV149/5DV150 Datastrukturer och algoritmer
Kapitel 17

Niclas Börlin
niclas.borlin@cs.umu.se
Anna Jonsson
aj@cs.umu.se

2020-02-26 Wed

Innehåll

- Grafalgoritmer
  1. Traversering
     - *Djupet-först*
     - *Bredden-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*
     - Prims algoritm.
     - Kruskal algoritm.

1. Traversering av grafer

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 labyrint genom att markera de vägar man gått med färg.
- Endast de noder man kan nå från utgångsnoden kommer att besökas.

Algoritm för djupet-först-traversering av graf

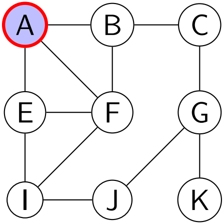
```
Algorithm g=depthFirst(Node n, Graph g)
  input: A node n in a graph g to be traversed
n.visited ← true; // Mark the node as visited.
neighbourSet ← neighbours(n,g); // All neighbours
for each neighbour b in neighbourSet do
  if not isVisited(b,g) then
    // Visit unless visited
    g ← depthFirst(b,g);
```

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.

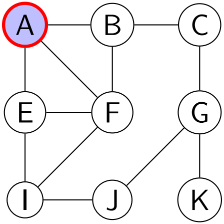
g=depthFirst(A,g) för oriktad graf

- $n \leftarrow$ A. Markera n som besökt.



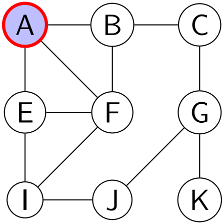
g=depthFirst(A,g) för oriktad graf

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



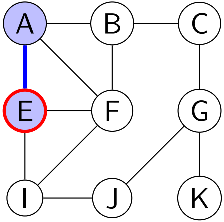
g=depthFirst(A,g) för oriktad graf

- $n \leftarrow$ A. Markera n som besökt.
- Grannar: {E,F,B}.
- E ej besökt $\rightarrow$ anropa depthFirst(E,g).



g=depthFirst(E,g) för oriktad graf

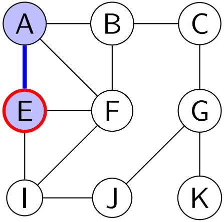
- $n \leftarrow$ E. Markera n som besökt.



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

g=depthFirst(E,g) för oriktad graf

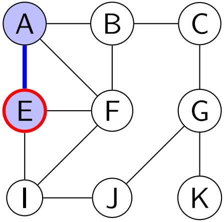
- $n \leftarrow$ E. Markera n som besökt.
- Grannar: $\{$I,F,A$\}$.



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

g=depthFirst(E,g) för oriktad graf

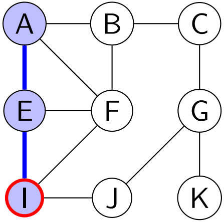
- $n \leftarrow$ E. Markera n som besökt.
- Grannar: {I,F,A}.
- I ej besökt $\rightarrow$ anropa depthFirst(I,g).



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

g=depthFirst(I,g) för oriktad graf

- $n \leftarrow$ I. Markera n som besökt.

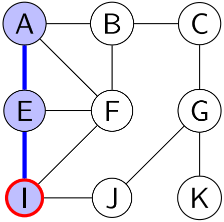


(n=E, {I,F,A})

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

g=depthFirst(I,g) för oriktad graf

- $n \leftarrow$ I. Markera n som besökt.
- Grannar: {E,J,F}.

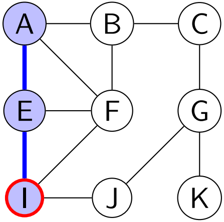


(n=E, {I,F,A})

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

g=depthFirst(I,g) för oriktad graf

- $n \leftarrow$ I. Markera n som besökt.
- Grannar: {E,J,F}.
- E redan besökt $\rightarrow$ Grannar: {~~E~~,J,F}.

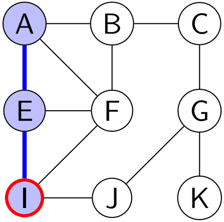


(n=E, {I,F,A})

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

g=depthFirst(I,g) för oriktad graf

- $n \leftarrow$ I. Markera n som besökt.
- Grannar: {E,J,F}.
- E redan besökt $\rightarrow$ Grannar: {~~E~~,J,F}.
- J ej besökt $\rightarrow$ anropa depthFirst(J,g).

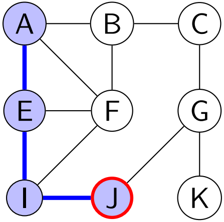


(n=E, {I,F,A})

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

g=depthFirst(J,g) för oriktad graf

- $n \leftarrow$ J. Markera n som besökt.



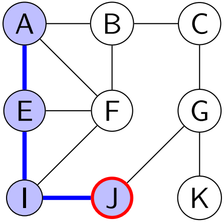
(n=I, {$\cancel{\text{E}}$,J,F})

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

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

g=depthFirst(J,g) för oriktad graf

- $n \leftarrow$ J. Markera n som besökt.
- Grannar: {G,I}.



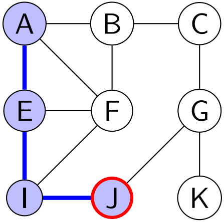
(n=I, {$\cancel{\text{E}}$,J,F})

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

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

g=depthFirst(J,g) för oriktad graf

- $n \leftarrow$ J. Markera n som besökt.
- Grannar: {G,I}.
- G ej besökt $\rightarrow$ anropa depthFirst(G,g).



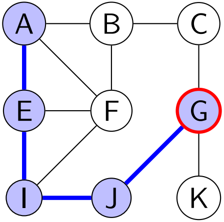
(n=I, $\{\cancel{E}$,J,F$\}$)

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

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

g=depthFirst(G,g) för oriktad graf

- $n \leftarrow$ G. Markera n som besökt.



(n=J, {G,I})

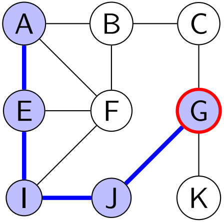
(n=I, {$\cancel{E}$,J,F})

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

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

g=depthFirst(G,g) för oriktad graf

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



(n=J, {G,I})

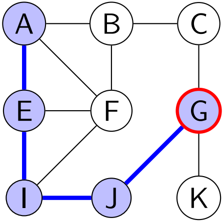
(n=I, {~~E~~,J,F})

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

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

g=depthFirst(G,g) för oriktad graf

- $n \leftarrow$ G. Markera n som besökt.
- Grannar: {C,K,J}.
- C ej besökt → anropa depthFirst(C,g).



(n=J, {G,I})

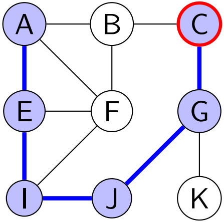
(n=I, {~~E~~,J,F})

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

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

g=depthFirst(C,g) för oriktad graf

- $n \leftarrow$ C. Markera n som besökt.



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

(n=J, {G,I})

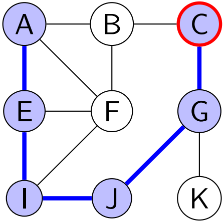
(n=I, {$\cancel{\text{E}}$,J,F})

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

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

g=depthFirst(C,g) för oriktad graf

- $n \leftarrow$ C. Markera n som besökt.
- Grannar: {G,B}.



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

(n=J, {G,I})

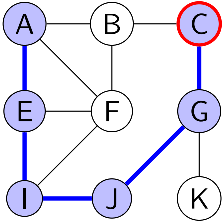
(n=I, {~~E~~,J,F})

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

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

g=depthFirst(C,g) för oriktad graf

- $n \leftarrow$ C. Markera n som besökt.
- Grannar: {G,B}.
- G redan besökt $\rightarrow$ Grannar: {~~G~~,B}.



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

(n=J, {G,I})

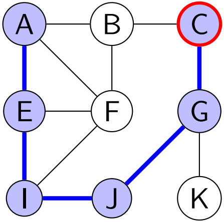
(n=I, {~~E~~,J,F})

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

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

g=depthFirst(C,g) för oriktad graf

- $n \leftarrow$ C. Markera n som besökt.
- Grannar: {G,B}.
- G redan besökt $\rightarrow$ Grannar: {$\cancel{G}$,B}.
- B ej besökt $\rightarrow$ anropa depthFirst(B,g).



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

(n=J, {G,I})

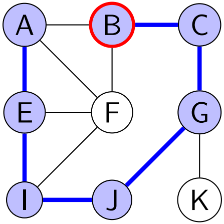
(n=I, {$\cancel{E}$,J,F})

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

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

g=depthFirst(B,g) för oriktad graf

- $n \leftarrow$ B. Markera n som besökt.



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

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

(n=J, {G,I})

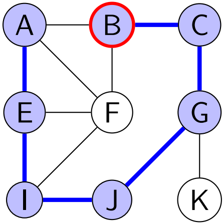
(n=I, {$\cancel{E}$,J,F})

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

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

g=depthFirst(B,g) för oriktad graf

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



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

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

(n=J, {G,I})

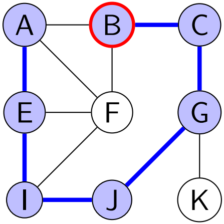
(n=I, {~~E~~,J,F})

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

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

g=depthFirst(B,g) för oriktad graf

- $n \leftarrow$ B. Markera n som besökt.
- Grannar: {A,F,C}.
- A redan besökt $\rightarrow$ Grannar: {$\cancel{\text{A}}$,F,C}.



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

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

(n=J, {G,I})

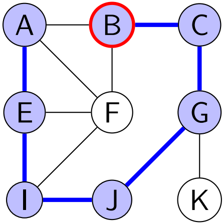
(n=I, {$\cancel{\text{E}}$,J,F})

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

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

g=depthFirst(B,g) för oriktad graf

- $n \leftarrow$ B. Markera n som besökt.
- Grannar: {A,F,C}.
- A redan besökt $\rightarrow$ Grannar: {~~A~~,F,C}.
- F ej besökt $\rightarrow$ anropa depthFirst(F,g).



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

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

(n=J, {G,I})

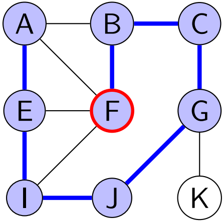
(n=I, {~~E~~,J,F})

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

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

g=depthFirst(F,g) för oriktad graf

- $n \leftarrow$ F. Markera n som besökt.



(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=depthFirst(F,g) för oriktad graf

- $n \leftarrow$ F. Markera n som besökt.
- Granner: {B,A,E,I}.

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

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

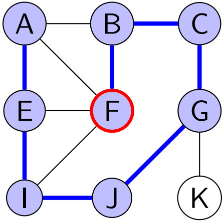
(*n*=G, {C,K,J})

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

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

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

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



g=depthFirst(F,g) för oriktad graf

- $n \leftarrow$ F. Markera n som besökt.
- Grannar: {B,A,E,I}.
- B besökt $\rightarrow$ Grannar: {~~B~~,A,E,I}.

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

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

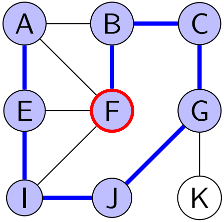
(n=G, {C,K,J})

(n=J, {G,I})

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

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

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



g=depthFirst(F,g) för oriktad graf

- $n \leftarrow$ F. Markera n som besökt.
- Grannar: {B,A,E,I}.
- B besökt → Grannar: {~~B~~,A,E,I}.
- A besökt → Grannar: {~~B~~,~~A~~,E,I}.

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

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

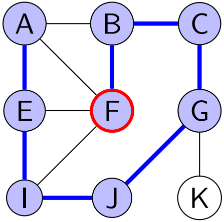
(n=G, {C,K,J})

(n=J, {G,I})

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

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

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



g=depthFirst(F,g) för oriktad graf

- $n \leftarrow$ F. Markera n som besökt.
- Grannar: {B,A,E,I}.
- B besökt → Grannar: {$\cancel{B}$,A,E,I}.
- A besökt → Grannar: {$\cancel{B}$,$\cancel{A}$,E,I}.
- E besökt → Grannar: {$\cancel{B}$,$\cancel{A}$,$\cancel{E}$,I}.

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

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

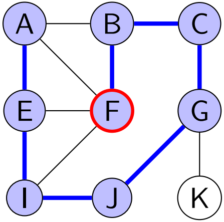
(n=G, {C,K,J})

(n=J, {G,I})

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

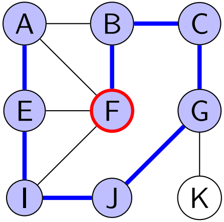
(n=E, {I,F,A})

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



g=depthFirst(F,g) för oriktad graf

- $n \leftarrow$ F. Markera n som besökt.
- Grannar: {B,A,E,I}.
- B besökt $\rightarrow$ Grannar: {$\cancel{\text{B}}$,A,E,I}.
- A besökt $\rightarrow$ Grannar: {$\cancel{\text{B}}$,$\cancel{\text{A}}$,E,I}.
- E besökt $\rightarrow$ Grannar: {$\cancel{\text{B}}$,$\cancel{\text{A}}$,$\cancel{\text{E}}$,I}.
- I besökt $\rightarrow$ Grannar: {$\cancel{\text{B}}$,$\cancel{\text{A}}$,$\cancel{\text{E}}$,$\cancel{\text{I}}$}.



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

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

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

(n=J, {G,I})

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

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

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

g=depthFirst(F,g) för oriktad graf

- $n \leftarrow$ F. Markera n 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=B, {$\cancel{A}$,F,C})

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

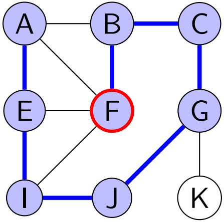
(n=G, {C,K,J})

(n=J, {G,I})

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

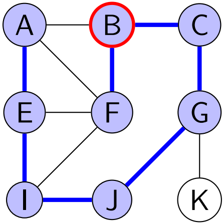
(n=E, {I,F,A})

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



g=depthFirst(B,g) för oriktad graf

- $n \leftarrow$ B. Markera n som besökt.
- Grannar: {A,F,C}.
- A redan besökt $\rightarrow$ Grannar: {~~A~~,F,C}.
- F ej besökt $\rightarrow$ anropa depthFirst(F,g).



(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=depthFirst(B,g) för oriktad graf

- $n \leftarrow$ B. Markera n som besökt.
- Grannar: {A,F,C}.
- A redan besökt $\rightarrow$ Grannar: {$\cancel{A}$,F,C}.
- F ej besökt $\rightarrow$ anropa depthFirst(F,g).
- F färdig $\rightarrow$ Grannar: {$\cancel{A}$,$\cancel{F}$,C}.

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

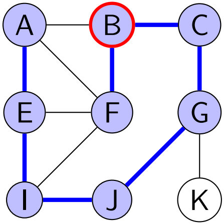
(n=G, {C,K,J})

(n=J, {G,I})

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

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

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



g=depthFirst(B,g) för oriktad graf

- $n \leftarrow$ B. Markera n som besökt.
- Grannar: {A,F,C}.
- A redan besökt $\rightarrow$ Grannar: {$\cancel{A}$,F,C}.
- F ej besökt $\rightarrow$ anropa depthFirst(F,g).
- F färdig $\rightarrow$ Grannar: {$\cancel{A}$,$\cancel{F}$,C}.
- C besökt $\rightarrow$ Grannar: {$\cancel{A}$,$\cancel{F}$,$\cancel{C}$}.

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

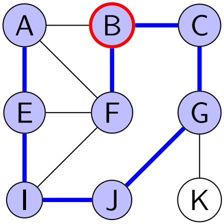
(n=G, {C,K,J})

(n=J, {G,I})

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

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

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



g=depthFirst(B,g) för oriktad graf

- $n \leftarrow$ B. Markera n som besökt.
- Grannar: {A,F,C}.
- A redan besökt $\rightarrow$ Grannar: {~~A~~,F,C}.
- F ej besökt $\rightarrow$ anropa depthFirst(F,g).
- F färdig $\rightarrow$ Grannar: {~~A~~,~~F~~,C}.
- C besökt $\rightarrow$ Grannar: {~~A~~,~~F~~,~~C~~}.
- Färdig med B, återvänd.

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

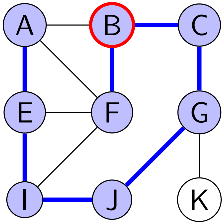
(n=G, {C,K,J})

(n=J, {G,I})

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

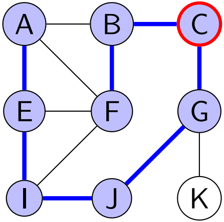
(n=E, {I,F,A})

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



g=depthFirst(C,g) för oriktad graf

- $n \leftarrow$ C. Markera n som besökt.
- Grannar: {G,B}.
- G redan besökt → Grannar: {~~G~~,B}.
- B ej besökt → anropa depthFirst(B,g).



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

(n=J, {G,I})

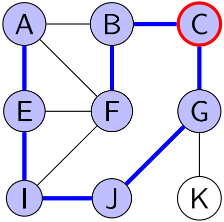
(n=I, {~~E~~,J,F})

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

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

g=depthFirst(C,g) för oriktad graf

- $n \leftarrow$ C. Markera n som besökt.
- Grannar: $\{$G,B$\}$.
- G redan besökt $\rightarrow$ Grannar: $\{\cancel{\text{G}}$,B$\}$.
- B ej besökt $\rightarrow$ anropa depthFirst(B,g).
- B färdig $\rightarrow$ Grannar: $\{\cancel{\text{G}},\cancel{\text{B}}\}$.



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

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

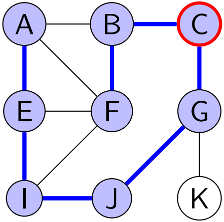
(n=I, $\{\cancel{\text{E}}$,J,F$\}$)

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

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

g=depthFirst(C,g) för oriktad graf

- $n \leftarrow$ C. Markera n som besökt.
- Grannar: {G,B}.
- G redan besökt → Grannar: {$\cancel{G}$,B}.
- B ej besökt → anropa depthFirst(B,g).
- B färdig → Grannar: {$\cancel{G}$,$\cancel{B}$}.
- Färdig med C, återvänd.



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

(n=J, {G,I})

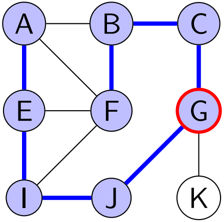
(n=I, {$\cancel{E}$,J,F})

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

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

g=depthFirst(G,g) för oriktad graf

- $n \leftarrow$ G. Markera n som besökt.
- Grannar: {C,K,J}.
- C ej besökt $\rightarrow$ anropa depthFirst(C,g).



(n=J, {G,I})

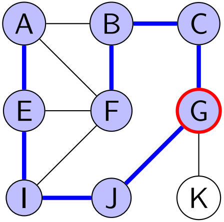
(n=I, {$\cancel{E}$,J,F})

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

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

g=depthFirst(G,g) för oriktad graf

- $n \leftarrow$ G. Markera n som besökt.
- Grannar: {C,K,J}.
- C ej besökt → anropa depthFirst(C,g).
- C färdig → Grannar: {$\cancel{C}$,K,J}.



(n=J, {G,I})

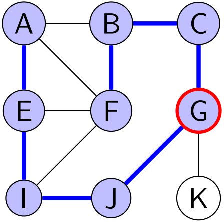
(n=I, {$\cancel{E}$,J,F})

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

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

g=depthFirst(G,g) för oriktad graf

- $n \leftarrow$ G. Markera n som besökt.
- Grannar: {C,K,J}.
- C ej besökt $\rightarrow$ anropa depthFirst(C,g).
- C färdig $\rightarrow$ Grannar: {~~C~~,K,J}.
- K ej besökt $\rightarrow$ anropa depthFirst(K,g).



(n=J, {G,I})

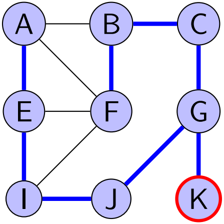
(n=I, {~~E~~,J,F})

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

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

g=depthFirst(K,g) för oriktad graf

- $n \leftarrow$ K. Markera n som besökt.



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

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

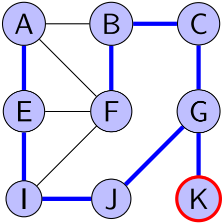
(n=I, $\{\cancel{E}$,J,F$\}$)

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

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

g=depthFirst(K,g) för oriktad graf

- $n \leftarrow$ K. Markera n som besökt.
- Grannar: {G}.



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

(n=J, {G,I})

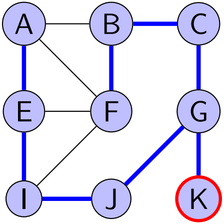
(n=I, {~~E~~,J,F})

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

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

g=depthFirst(K,g) för oriktad graf

- $n \leftarrow$ K. Markera n som besökt.
- Grannar: {G}.
- G besökt $\rightarrow$ Grannar: {$\cancel{G}$}.



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

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

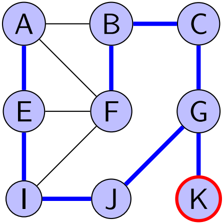
(*n*=I, {$\cancel{E}$,J,F})

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

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

g=depthFirst(K,g) för oriktad graf

- $n \leftarrow$ K. Markera n som besökt.
- Grannar: {G}.
- G besökt → Grannar: {~~G~~}.
- Färdig med K, återvänd.



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

(n=J, {G,I})

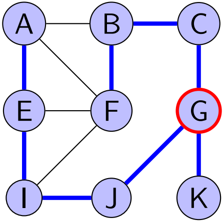
(n=I, {~~E~~,J,F})

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

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

g=depthFirst(G,g) för oriktad graf

- $n \leftarrow$ G. Markera n som besökt.
- Grannar: {C,K,J}.
- C ej besökt $\rightarrow$ anropa depthFirst(C,g).
- C färdig $\rightarrow$ Grannar: {~~C~~,K,J}.
- K ej besökt $\rightarrow$ anropa depthFirst(K,g).



(n=J, {G,I})

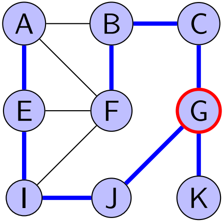
(n=I, {~~E~~,J,F})

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

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

g=depthFirst(G,g) för oriktad graf

- $n \leftarrow$ G. Markera n som besökt.
- Grannar: {C,K,J}.
- C ej besökt → anropa depthFirst(C,g).
- C färdig → Grannar: {$\cancel{C}$,K,J}.
- K ej besökt → anropa depthFirst(K,g).
- K färdig → Grannar: {$\cancel{C}$,$\cancel{K}$,J}.



(n=J, {G,I})

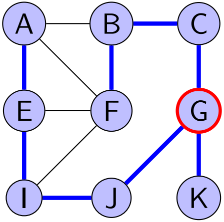
(n=I, {$\cancel{E}$,J,F})

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

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

g=depthFirst(G,g) för oriktad graf

- n ← G. Markera n som besökt.
- Grannar: {C,K,J}.
- C ej besökt → anropa depthFirst(C,g).
- C färdig → Grannar: {~~C~~,K,J}.
- K ej besökt → anropa depthFirst(K,g).
- K färdig → Grannar: {~~C~~,~~K~~,J}.
- J besökt → Grannar: {~~C~~,~~K~~,~~J~~}.



(n=J, {G,I})

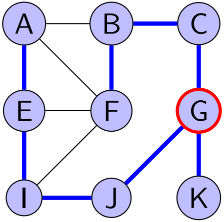
(n=I, {~~E~~,J,F})

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

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

g=depthFirst(G,g) för oriktad graf

- $n \leftarrow$ G. Markera n som besökt.
- Grannar: {C,K,J}.
- C ej besökt → anropa depthFirst(C,g).
- C färdig → Grannar: {~~C~~,K,J}.
- K ej besökt → anropa depthFirst(K,g).
- K färdig → Grannar: {~~C~~,~~K~~,J}.
- J besökt → Grannar: {~~C~~,~~K~~,~~J~~}.
- Färdig med G, återvänd.



(n=J, {G,I})

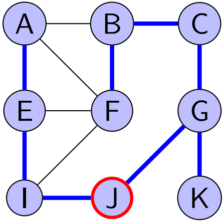
(n=I, {~~E~~,J,F})

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

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

g=depthFirst(J,g) för oriktad graf

- $n \leftarrow$ J. Markera n som besökt.
- Grannar: {G,I}.
- G ej besökt $\rightarrow$ anropa depthFirst(G,g).



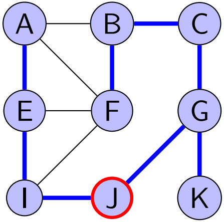
(n=I, {$\cancel{E}$,J,F})

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

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

g=depthFirst(J,g) för oriktad graf

- $n \leftarrow$ J. Markera n som besökt.
- Grannar: {G,I}.
- G ej besökt $\rightarrow$ anropa depthFirst(G,g).
- G färdig $\rightarrow$ Grannar: {~~G~~,I}.



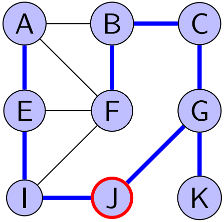
(n=I, {~~E~~,J,F})

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

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

g=depthFirst(J,g) för oriktad graf

- $n \leftarrow$ J. Markera n som besökt.
- Grannar: {G,I}.
- G ej besökt → anropa depthFirst(G,g).
- G färdig → Grannar: {$\cancel{\text{G}}$,I}.
- I besökt → Grannar: {$\cancel{\text{G}}$,$\cancel{\text{I}}$}.



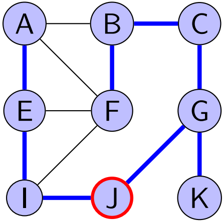
(n=I, {$\cancel{\text{E}}$,J,F})

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

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

g=depthFirst(J,g) för oriktad graf

- $n \leftarrow$ J. Markera n som besökt.
- Grannar: {G,I}.
- G ej besökt → anropa depthFirst(G,g).
- G färdig → Grannar: {~~G~~,I}.
- I besökt → Grannar: {~~G~~,~~I~~}.
- Färdig med J, återvänd.



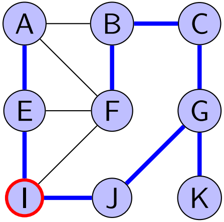
(n=I, {~~E~~,J,F})

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

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

g=depthFirst(I,g) för oriktad graf

- $n \leftarrow$ I. Markera n som besökt.
- Grannar: {E,J,F}.
- E redan besökt $\rightarrow$ Grannar: {~~E~~,J,F}.
- J ej besökt $\rightarrow$ anropa depthFirst(J,g).

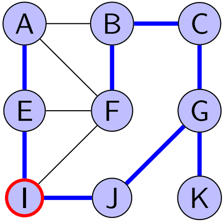


(n=E, {I,F,A})

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

g=depthFirst(I,g) för oriktad graf

- $n \leftarrow$ I. Markera n som besökt.
- Grannar: {E,J,F}.
- E redan besökt $\rightarrow$ Grannar: {~~E~~,J,F}.
- J ej besökt $\rightarrow$ anropa depthFirst(J,g).
- J färdig $\rightarrow$ Grannar: {~~E~~,~~J~~,F}.

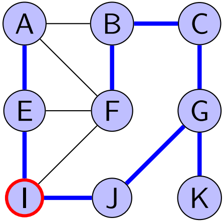


(n=E, {I,F,A})

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

g=depthFirst(I,g) för oriktad graf

- $n \leftarrow$ I. Markera n som besökt.
- Grannar: {E,J,F}.
- E redan besökt $\rightarrow$ Grannar: {$\cancel{E}$,J,F}.
- J ej besökt $\rightarrow$ anropa depthFirst(J,g).
- J färdig $\rightarrow$ Grannar: {$\cancel{E}$,$\cancel{J}$,F}.
- F besökt $\rightarrow$ Grannar: {$\cancel{E}$,$\cancel{J}$,$\cancel{F}$}

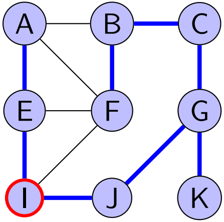


(n=E, {I,F,A})

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

g=depthFirst(I,g) för oriktad graf

- $n \leftarrow$ I. Markera n som besökt.
- Grannar: {E,J,F}.
- E redan besökt $\rightarrow$ Grannar: {$\cancel{E}$,J,F}.
- J ej besökt $\rightarrow$ anropa depthFirst(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.

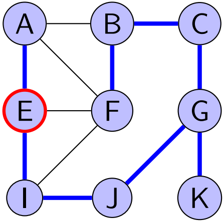


(n=E, {I,F,A})

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

g=depthFirst(E,g) för oriktad graf

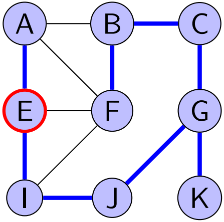
- $n \leftarrow$ E. Markera n som besökt.
- Grannar: {I,F,A}.
- I ej besökt $\rightarrow$ anropa depthFirst(I,g).



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

g=depthFirst(E,g) för oriktad graf

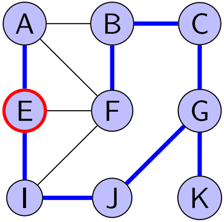
- $n \leftarrow$ E. Markera n som besökt.
- Grannar: {I,F,A}.
- I ej besökt $\rightarrow$ anropa depthFirst(I,g).
- I färdig $\rightarrow$ Grannar: {~~I~~,F,A}.



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

g=depthFirst(E,g) för oriktad graf

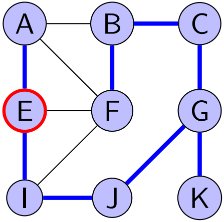
- $n \leftarrow$ E. Markera n som besökt.
- Grannar: {I,F,A}.
- I ej besökt → anropa depthFirst(I,g).
- I färdig → Grannar: {~~I~~,F,A}.
- F besökt → Grannar: {~~I~~,~~F~~,A}



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

g=depthFirst(E,g) för oriktad graf

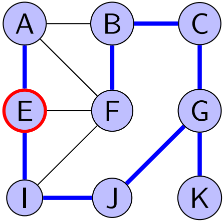
- $n \leftarrow$ E. Markera n som besökt.
- Grannar: {I,F,A}.
- I ej besökt $\rightarrow$ anropa depthFirst(I,g).
- I färdig $\rightarrow$ Grannar: {~~I~~,F,A}.
- F besökt $\rightarrow$ Grannar: {~~I~~,~~F~~,A}
- A besökt $\rightarrow$ Grannar: {~~I~~,~~F~~,~~A~~}.



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

g=depthFirst(E,g) för oriktad graf

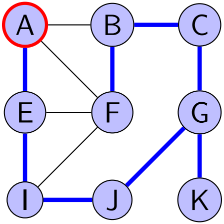
- $n \leftarrow$ E. Markera n som besökt.
- Grannar: {I,F,A}.
- I ej besökt → anropa depthFirst(I,g).
- I färdig → Grannar: {$\cancel{\text{I}}$,F,A}.
- F besökt → Grannar: {$\cancel{\text{I}}$,$\cancel{\text{F}}$,A}
- A besökt → Grannar: {$\cancel{\text{I}}$,$\cancel{\text{F}}$,$\cancel{\text{A}}$}.
- Färdig med E, återvänd.



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

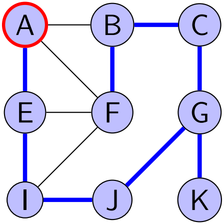
g=depthFirst(A,g) för oriktad graf

- $n \leftarrow$ A. Markera n som besökt.
- Grannar: {E,F,B}.
- E ej besökt $\rightarrow$ anropa depthFirst(E,g).



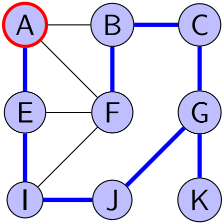
g=depthFirst(A,g) för oriktad graf

- $n \leftarrow$ A. Markera n som besökt.
- Grannar: {E,F,B}.
- E ej besökt $\rightarrow$ anropa depthFirst(E,g).
- E färdig $\rightarrow$ Grannar: {~~E~~,F,B}.



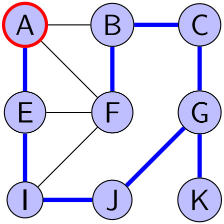
g=depthFirst(A,g) för oriktad graf

- $n \leftarrow$ A. Markera n som besökt.
- Grannar: {E,F,B}.
- E ej besökt → anropa depthFirst(E,g).
- E färdig → Grannar: {~~E~~,F,B}.
- F besökt → Grannar: {~~E~~,~~F~~,B}



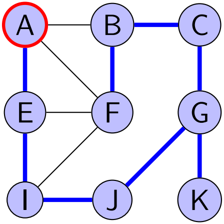
g=depthFirst(A,g) för oriktad graf

- $n \leftarrow$ A. Markera n som besökt.
- Grannar: {E,F,B}.
- E ej besökt $\rightarrow$ anropa depthFirst(E,g).
- E färdig $\rightarrow$ Grannar: {$\cancel{E}$,F,B}.
- F besökt $\rightarrow$ Grannar: {$\cancel{E}$,$\cancel{F}$,B}
- B besökt $\rightarrow$ Grannar: {$\cancel{E}$,$\cancel{F}$,$\cancel{B}$}.



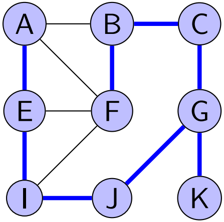
g=depthFirst(A,g) för oriktad graf

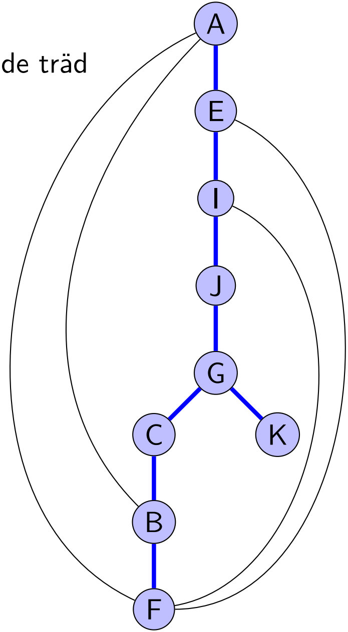
- $n \leftarrow$ A. Markera n som besökt.
- Grannar: {E,F,B}.
- E ej besökt $\rightarrow$ anropa depthFirst(E,g).
- E färdig $\rightarrow$ Grannar: {$\cancel{E}$,F,B}.
- F besökt $\rightarrow$ Grannar: {$\cancel{E}$,$\cancel{F}$,B}
- B besökt $\rightarrow$ Grannar: {$\cancel{E}$,$\cancel{F}$,$\cancel{B}$}.
- Färdig med A, återvänd.



Klart!

- Notera att vi fick ett uppspännande träd på samma gång.





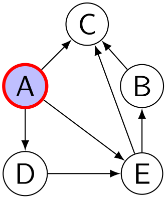
Hur behöver algoritmen modifieras för att fungera på en riktad graf?

Algoritm för djupet-först-traversering av graf

```
Algorithm g=depthFirst(Node n, Graph g)
  input: A node n in a graph g to be traversed
n.visited ← true; // Mark the node as visited.
neighbourSet ← neighbours(n,g); // All neighbours
for each neighbour b in neighbourSet do
  if not isVisited(b,g) then
    // Visit unless visited
    g ← depthFirst(b,g);
```

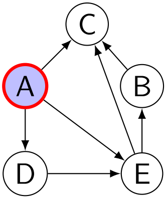
g=depthFirst(A,g) för riktad graf

- $n \leftarrow$ A. Markera n som besökt.



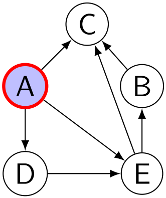
g=depthFirst(A,g) för riktad graf

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



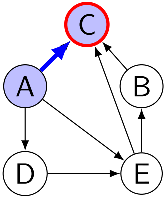
g=depthFirst(A,g) för riktad graf

- $n \leftarrow$ A. Markera n som besökt.
- Grannar: $\{$C,E,D$\}$.
- C ej besökt $\rightarrow$ anropa depthFirst(C,g).



g=depthFirst(C,g) för riktad graf

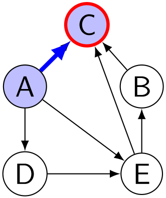
- $n \leftarrow$ C. Markera n som besökt.



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

g=depthFirst(C,g) för riktad graf

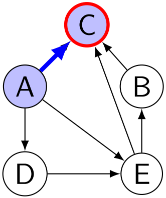
- $n \leftarrow$ C. Markera n som besökt.
- Inga grannar.



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

g=depthFirst(C,g) för riktad graf

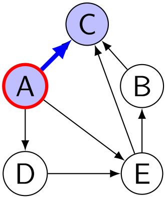
- $n \leftarrow$ C. Markera n som besökt.
- Inga grannar.
- Färdig med C, återvänd.



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

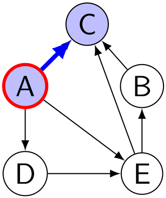
g=depthFirst(A,g) för riktad graf

- $n \leftarrow$ A. Markera n som besökt.
- Grannar: {C,E,D}.
- C ej besökt $\rightarrow$ anropa depthFirst(C,g).



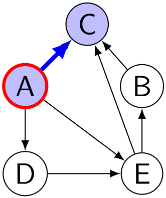
g=depthFirst(A,g) för riktad graf

- $n \leftarrow$ A. Markera n som besökt.
- Grannar: {C,E,D}.
- C ej besökt $\rightarrow$ anropa depthFirst(C,g).
- C färdig $\rightarrow$ Grannar: {$\cancel{C}$,E,D}.



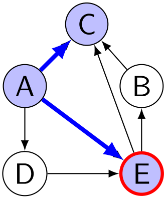
g=depthFirst(A,g) för riktad graf

- $n \leftarrow$ A. Markera n som besökt.
- Grannar: {C,E,D}.
- C ej besökt $\rightarrow$ anropa depthFirst(C,g).
- C färdig $\rightarrow$ Grannar: {$\cancel{C}$,E,D}.
- E ej besökt $\rightarrow$ anropa depthFirst(E,g).



g=depthFirst(E,g) för riktad graf

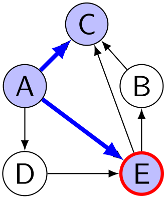
- $n \leftarrow$ E. Markera n som besökt.



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

g=depthFirst(E,g) för riktad graf

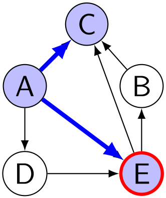
- $n \leftarrow$ E. Markera n som besökt.
- Grannar: $\{$B,C$\}$.



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

g=depthFirst(E,g) för riktad graf

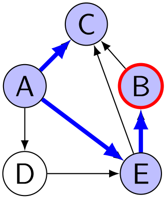
- $n \leftarrow$ E. Markera n som besökt.
- Grannar: $\{$B,C$\}$.
- B ej besökt $\rightarrow$ anropa depthFirst(B,g)



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

g=depthFirst(B,g) för riktad graf

- $n \leftarrow$ B. Markera n som besökt.

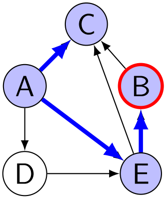


(n=E, {B,C})

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

g=depthFirst(B,g) för riktad graf

- $n \leftarrow$ B. Markera n som besökt.
- Grannar: $\{$C$\}$.

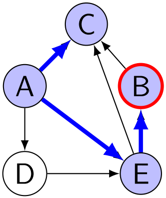


(n=E, $\{$B,C$\}$)

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

g=depthFirst(B,g) för riktad graf

- $n \leftarrow$ B. Markera n som besökt.
- Grannar: $\{$C$\}$.
- C besökt $\rightarrow$ Grannar: $\{\cancel{\text{C}}\}$.

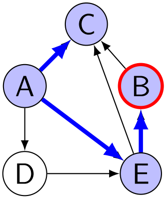


(n=E, $\{$B,C$\}$)

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

g=depthFirst(B,g) för riktad graf

- $n \leftarrow$ B. Markera n som besökt.
- Grannar: $\{C\}$.
- C besökt $\rightarrow$ Grannar: $\{\cancel{C}\}$.
- Färdig med B, återvänd.

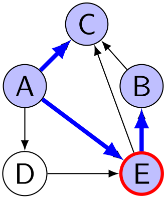


(n=E, $\{B,C\}$)

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

g=depthFirst(E,g) för riktad graf

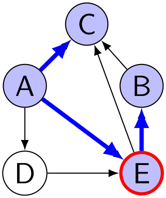
- $n \leftarrow$ E. Markera n som besökt.
- Grannar: {B,C}.
- B ej besökt $\rightarrow$ rekursivt anrop depthFirst(B,g)



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

g=depthFirst(E,g) för riktad graf

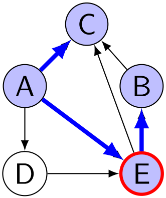
- $n \leftarrow$ E. Markera n som besökt.
- Grannar: {B,C}.
- B ej besökt → rekursivt anrop depthFirst(B,g)
- B färdig → Grannar: {~~B~~,C}.



(n=A, {~~C~~,E,D})

g=depthFirst(E,g) för riktad graf

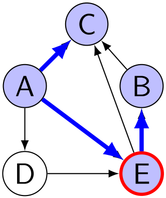
- $n \leftarrow$ E. Markera n som besökt.
- Grannar: {B,C}.
- B ej besökt $\rightarrow$ rekursivt anrop depthFirst(B,g)
- B färdig $\rightarrow$ Grannar: {~~B~~,C}.
- C besökt $\rightarrow$ Grannar: {~~B~~,~~C~~}.



(n=A, {~~C~~,E,D})

g=depthFirst(E,g) för riktad graf

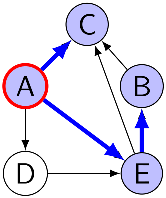
- $n \leftarrow$ E. Markera n som besökt.
- Grannar: {B,C}.
- B ej besökt $\rightarrow$ rekursivt anrop depthFirst(B,g)
- B färdig $\rightarrow$ Grannar: {$\cancel{\text{B}}$,C}.
- C besökt $\rightarrow$ Grannar: {$\cancel{\text{B}}$,$\cancel{\text{C}}$}.
- Färdig med E, återvänd.



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

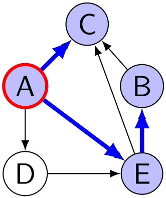
g=depthFirst(A,g) för riktad graf

- $n \leftarrow$ A. Markera n som besökt.
- Grannar: {C,E,D}.
- C ej besökt $\rightarrow$ anropa depthFirst(C,g).
- C färdig $\rightarrow$ Grannar: {$\cancel{C}$,E,D}.
- E ej besökt $\rightarrow$ rekursivt anrop depthFirst(E,g).



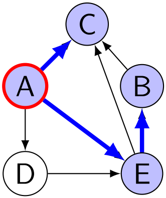
g=depthFirst(A,g) för riktad graf

- $n \leftarrow$ A. Markera n som besökt.
- Grannar: {C,E,D}.
- C ej besökt $\rightarrow$ anropa depthFirst(C,g).
- C färdig $\rightarrow$ Grannar: {$\cancel{C}$,E,D}.
- E ej besökt $\rightarrow$ rekursivt anrop depthFirst(E,g).
- E färdig $\rightarrow$ Grannar: {$\cancel{C}$,$\cancel{E}$,D}.



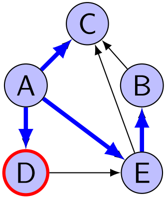
g=depthFirst(A,g) för riktad graf

- $n \leftarrow$ A. Markera n som besökt.
- Grannar: {C,E,D}.
- C ej besökt $\rightarrow$ anropa depthFirst(C,g).
- C färdig $\rightarrow$ Grannar: {$\cancel{C}$,E,D}.
- E ej besökt $\rightarrow$ rekursivt anrop depthFirst(E,g).
- E färdig $\rightarrow$ Grannar: {$\cancel{C}$,$\cancel{E}$,D}.
- D ej besökt $\rightarrow$ rekursivt anrop depthFirst(D,g).



g=depthFirst(D,g) för riktad graf

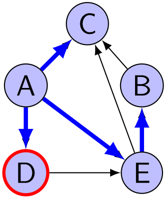
- $n \leftarrow$ D. Markera n som besökt.



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

g=depthFirst(D,g) för riktad graf

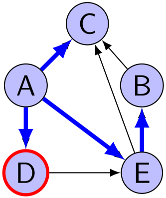
- $n \leftarrow$ D. Markera n som besökt.
- Grannar: $\{$E$\}$.



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

g=depthFirst(D,g) för riktad graf

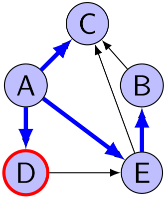
- $n \leftarrow$ D. Markera n som besökt.
- Grannar: {E}.
- E besökt $\rightarrow$ Grannar: {$\cancel{E}$}.



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

g=depthFirst(D,g) för riktad graf

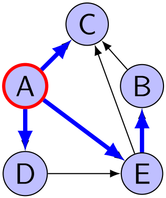
- $n \leftarrow$ D. Markera n som besökt.
- Grannar: {E}.
- E besökt $\rightarrow$ Grannar: {$\cancel{\text{E}}$}.
- Färdig med D, återvänd.



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

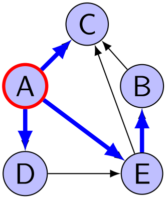
g=depthFirst(A,g) för riktad graf

- $n \leftarrow$ A. Markera n som besökt.
- Grannar: {C,E,D}.
- C ej besökt $\rightarrow$ anropa depthFirst(C,g).
- C färdig $\rightarrow$ Grannar: {$\cancel{C}$,E,D}.
- E ej besökt $\rightarrow$ rekursivt anrop depthFirst(E,g).
- E färdig $\rightarrow$ Grannar: {$\cancel{C}$,$\cancel{E}$,D}.
- D ej besökt $\rightarrow$ rekursivt anrop depthFirst(D,g).



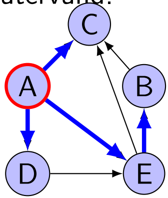
g=depthFirst(A,g) för riktad graf

- $n \leftarrow$ A. Markera n som besökt.
- Grannar: {C,E,D}.
- C ej besökt $\rightarrow$ anropa depthFirst(C,g).
- C färdig $\rightarrow$ Grannar: {$\cancel{C}$,E,D}.
- E ej besökt $\rightarrow$ rekursivt anrop depthFirst(E,g).
- E färdig $\rightarrow$ Grannar: {$\cancel{C}$,$\cancel{E}$,D}.
- D ej besökt $\rightarrow$ rekursivt anrop depthFirst(D,g).
- D färdig $\rightarrow$ Grannar: {$\cancel{C}$,$\cancel{E}$,$\cancel{D}$}.



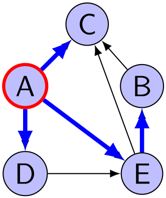
g=depthFirst(A,g) för riktad graf

- $n \leftarrow$ A. Markera n som besökt.
- Grannar: {C,E,D}.
- C ej besökt $\rightarrow$ anropa depthFirst(C,g).
- C färdig $\rightarrow$ Grannar: {$\cancel{C}$,E,D}.
- E ej besökt $\rightarrow$ rekursivt anrop depthFirst(E,g).
- E färdig $\rightarrow$ Grannar: {$\cancel{C}$,$\cancel{E}$,D}.
- D ej besökt $\rightarrow$ rekursivt anrop depthFirst(D,g).
- D färdig $\rightarrow$ Grannar: {$\cancel{C}$,$\cancel{E}$,$\cancel{D}$}.
- Färdig med A, återvänd.



Klar

- Fick också uppspännande träd.



Bredden-först-traversering

- Man undersöker först noden, sedan dess grannar, grannarnas grannar, osv.
- 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.

Algoritm, bredden-först-traversering av graf

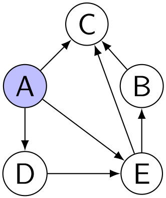
```
Algorithm g=breadthFirst(Node n, Graph g)
  input: A node n in a graph g to be traversed
Queue q ← empty();
(n,g) ← seen(n,g) // Mark the node as seen.
q ← enqueue(n,q); // Put node in queue.
while not isempty(q) do
  p ← front(q); // Pick first node from queue
  q ← dequeue(q);
  neighbourSet ← neighbours(p,g);
  for each neighbour b in neighbourSet do
    if not isSeen(b,g) then
        (b,g) ← seen(b,g) // Mark node as seen.
        q ← enqueue(b,q); // Put node in queue.
```

Visualiseringssymboler

- Aktuell nod markeras med röd ring.
- Ljusblå färg betyder sedd (*seen*) nod.
- Noder i kön markeras med grönstreckad cirkel.
- Bågarna som motsvarar rekursiva anrop markeras med tjock blå linje.

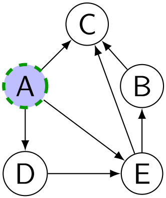
g=breadthFirst(A,g)

- seen(A,g);



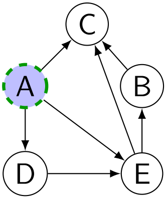
g=breadthFirst(A,g)

- seen(A,g);
- q={A};



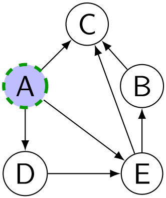
g=breadthFirst(A,g)

- seen(A,g);
- q={A};
- while not isempty(q)...



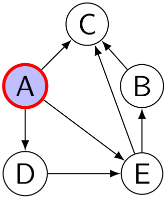
g=breadthFirst(A,g)

- while not isempty(q)...



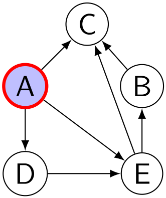
g=breadthFirst(A,g)

- while not isempty(q)...
  - p=A; q={};



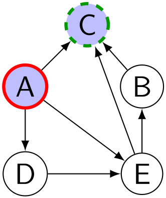
g=breadthFirst(A,g)

- while not isempty(q)...
  - p=A; q={};
  - neighbours={C,E,D}.



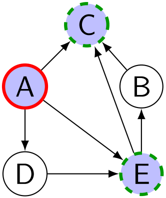
g=breadthFirst(A,g)

- while not isempty(q)...
  - p=A; q={};
  - neighbours={C,E,D}.
  - C not seen $\rightarrow$ seen(C,g); q={C};



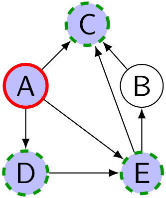
g=breadthFirst(A,g)

- while not isempty(q)...
  - p=A; q={};
  - neighbours={C,E,D}.
  - C not seen → seen(C,g); q={C};
  - E not seen → seen(E,g); q={C,E};



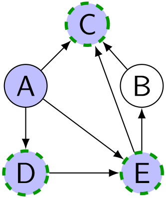
g=breadthFirst(A,g)

- while not isempty(q)...
  - p=A; q={};
  - neighbours={C,E,D}.
  - C not seen → seen(C,g); q={C};
  - E not seen → seen(E,g); q={C,E};
  - D not seen → seen(D,g); q={C,E,D};



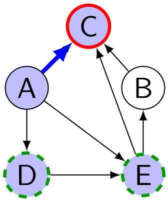
g=breadthFirst(A,g)

- while not isempty(q)...



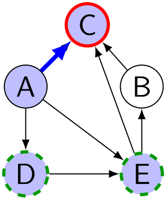
g=breadthFirst(A,g)

- while not isempty(q)...
  - p=C; q={E,D};



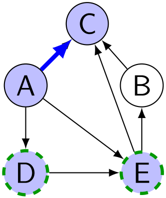
g=breadthFirst(A,g)

- while not isempty(q)...
  - p=C; q={E,D};
  - neighbours={}



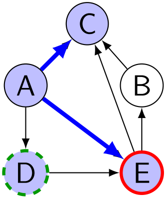
g=breadthFirst(A,g)

- while not isempty(q)...



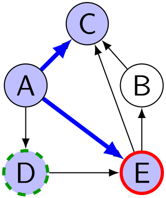
g=breadthFirst(A,g)

- while not isempty(q)...
  - p=E; q={D};



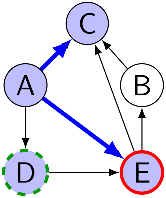
g=breadthFirst(A,g)

- while not isempty(q)...
  - p=E; q={D};
  - neighbours={C,B}



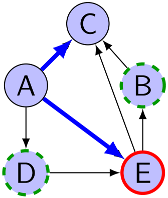
g=breadthFirst(A,g)

- while not isempty(q)...
  - p=E; q={D};
  - neighbours={C,B}
  - C seen.



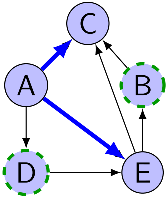
g=breadthFirst(A,g)

- while not isempty(q)...
  - p=E; q=$\{$D$\}$;
  - neighbours=$\{$C,B$\}$
  - C seen.
  - B not seen $\rightarrow$ seen(B,g); q=$\{$D,B$\}$;



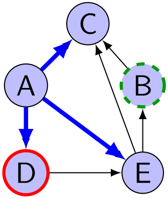
g=breadthFirst(A,g)

- while not isempty(q)...



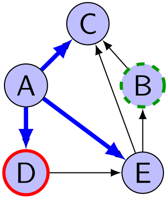
g=breadthFirst(A,g)

- while not isempty(q)...
  - p=D; q={B};



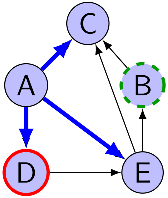
g=breadthFirst(A,g)

- while not isempty(q)...
  - p=D; q={B};
  - neighbours={E}



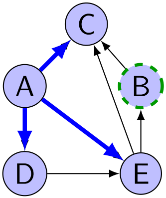
g=breadthFirst(A,g)

- while not isempty(q)...
  - p=D; q={B};
  - neighbours={E}
  - E seen.



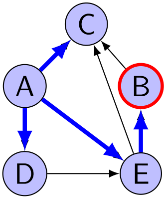
g=breadthFirst(A,g)

- while not isempty(q)...



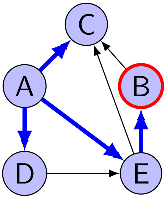
g=breadthFirst(A,g)

- while not isempty(q)...
  - p=B; q={};



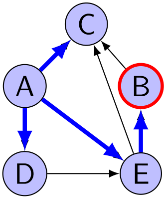
g=breadthFirst(A,g)

- while not isempty(q)...
  - p=B; q={};
  - neighbours={C}



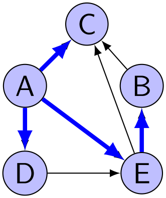
g=breadthFirst(A,g)

- while not isempty(q)...
  - p=B; q=$\{\}$;
  - neighbours=$\{$C$\}$
  - C seen.



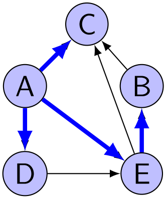
g=breadthFirst(A,g)

- while not isempty(q)...



g=breadthFirst(A,g)

- Klar!



Bredden-först, djupet-först-traversering, tidskomplexitet

- Givet en graf med n noder och m bågar.
- Varje nod besöks exakt en gång $O(n)$.
- 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:
    - $O(grad(v))$ per nod.
    - Totalt: $O(\sum grad(v)) = O(m)$ för alla bågar.
- Total komplexitet:

  Mängdorienterad $O(n) + O(mn) = O(mn)$.
  Navigeringsorienterad $O(n) + O(m) = O(m + n)$.

2. Kortaste-vägen-algoritmer

Kortaste vägen

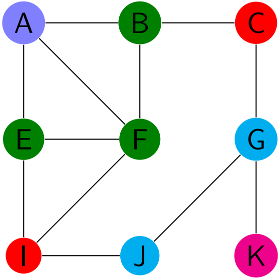
- Om grafen har *lika vikt på alla bågar* kan bredden-först-traversering användas för att beräkna kortaste vägen från en nod till alla andra noder.
- Krävs minimal modifiering av algoritmen:
  - Lägg till ett attribut avstånd (*distance*) till varje nod.
  - Avståndet från startnoden n till sig själv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

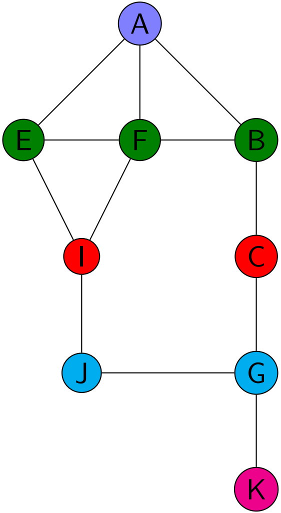
Kortaste-vägen-algoritm vid lika vikt

```
Algorithm g=shortestPath(Node n, Graph g)
  input: A node n in a graph g to be traversed
Queue q ← empty();
(n,g) ← seen(n,g) // Mark the node as seen.
setDist(n,0) // Distance from n to n.
q ← enqueue(n,q);
while not isempty(q) do
  p ← front(q); // Pick first node from queue
  q ← dequeue(q);
  neighbourSet ← neighbours(p,g);
  for each neighbour b in neighbourSet do
    if not isSeen(b,g) then
      (b,g) ← seen(b,g) // Mark node as seen.
      // Compute distance to new node.
      setDist(b,getDist(p)+1)
      q ← enqueue(b,q); // Put node in queue.
```

Kortaste vägen vid lika vikt/kostnad

- Startnod = A.





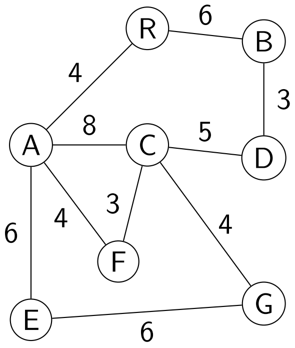
Kortaste vägen-algoritmer

- 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

Floyds shortest path

- Bygger på matrisrepresentation A av grafen.
- Vid starten innehåller A de direkta avstånden mellan noderna.
  - Avståndet till sig själv är 0.
  - Saknas båge 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, algoritm

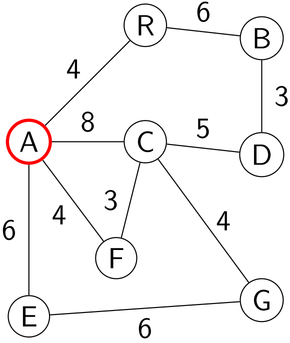
```
Algorithm floyd(Graph g)
  input: A graph g to find shortest path in
// Get matrix representation
A ← getMatrix(g)
n ← getNoOfNodes(g)
for k=1 to n do
  for i=1 to n do
    for j=1 to n do
        if A(i,j) > A(i,k) + A(k,j) then
            A(i,j) = A(i,k) + A(k,j)
```

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

Floyds shortest path

- Efter k=0 (vägar via A).

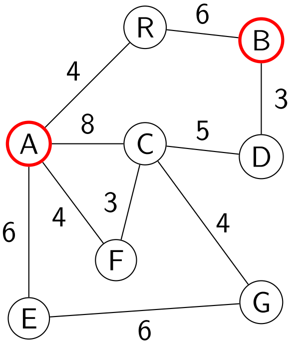
	A	B	C	D	E	F	G	R
A	0	∞	8	∞	6	4	∞	4
B	∞	0	∞	3	∞	∞	∞	6
C	8	∞	0	5	∞ 14	3	4	∞ 12
D	∞	3	5	0	∞	∞	∞	∞
E	6	∞	∞ 14	∞	0	∞ 10	6	∞ 10
F	4	∞	3	∞	∞ 10	0	∞	∞ 8
G	∞	∞	4	∞	6	∞	0	∞
R	4	6	∞ 12	∞	∞ 10	∞ 8	∞	0



Floyds shortest path

- Efter k=1 (vägar via B).

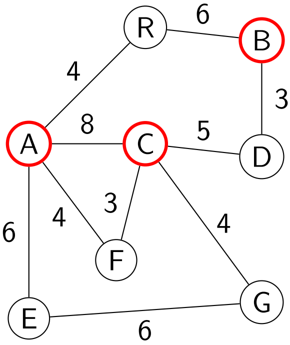
	A	B	C	D	E	F	G	R
A	0	∞	8	∞	6	4	∞	4
B	∞	0	∞	3	∞	∞	∞	6
C	8	∞	0	5	14	3	4	12
D	∞	3	5	0	∞	∞	∞	∞ 9
E	6	∞	14	∞	0	10	6	10
F	4	∞	3	∞	10	0	∞	8
G	∞	∞	4	∞	6	∞	0	∞
R	4	6	12	∞ 9	10	8	∞	0



Floyds shortest path

- Efter k=2 (vägar via C).

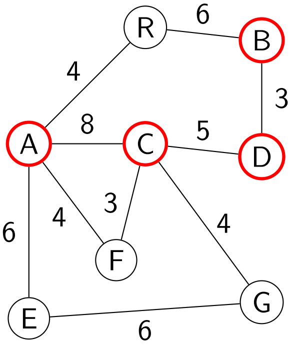
	A	B	C	D	E	F	G	R
A	0	∞	8	∞ 13	6	4	∞ 12	4
B	∞	0	∞	3	∞	∞	∞	6
C	8	∞	0	5	14	3	4	12
D	∞ 13	3	5	0	∞ 19	∞ 8	∞ 9	9
E	6	∞	14	∞ 19	0	10	6	10
F	4	∞	3	∞ 8	10	0	∞ 7	8
G	∞ 12	∞	4	∞ 9	6	∞ 7	0	∞ 16
R	4	6	12	9	10	8	∞ 16	0



Floyds shortest path

- Efter k=3 (vägar via D).

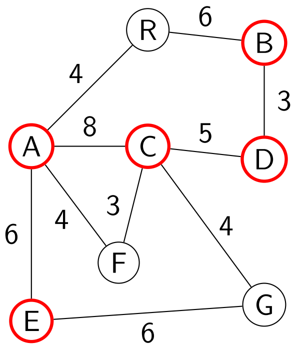
	A	B	C	D	E	F	G	R
A	0	∞ 16	8	13	6	4	12	4
B	∞ 16	0	∞ 8	3	∞ 22	∞ 11	∞ 12	6
C	8	∞ 8	0	5	14	3	4	12
D	13	3	5	0	19	8	9	9
E	6	∞ 22	14	19	0	10	6	10
F	4	∞ 11	3	8	10	0	7	8
G	12	∞ 12	4	9	6	7	0	16
R	4	6	12	9	10	8	16	0



Floyds shortest path

- Efter k=4 (vägar via E).

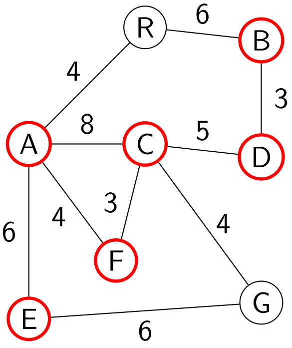
	A	B	C	D	E	F	G	R
A	0	16	8	13	6	4	12	4
B	16	0	8	3	22	11	12	6
C	8	8	0	5	14	3	4	12
D	13	3	5	0	19	8	9	9
E	6	22	14	19	0	10	6	10
F	4	11	3	8	10	0	7	8
G	12	12	4	9	6	7	0	16
R	4	6	12	9	10	8	16	0



Floyds shortest path

- Efter k=5 (vägar via F).

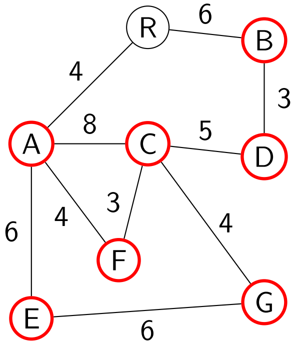
	A	B	C	D	E	F	G	R
A	0	16 15	8 7	13 12	6	4	12 11	4
B	16 15	0	8	3	22 21	11	12	6
C	8 7	8	0	5	14 13	3	4	12 11
D	13 12	3	5	0	19 18	8	9	9
E	6	22 21	14 13	19 18	0	10	6	10
F	4	11	3	8	10	0	7	8
G	12 11	12	4	9	6	7	0	16 15
R	4	6	12 11	9	10	8	16 15	0



Floyds shortest path

- Efter k=6 (vägar via G).

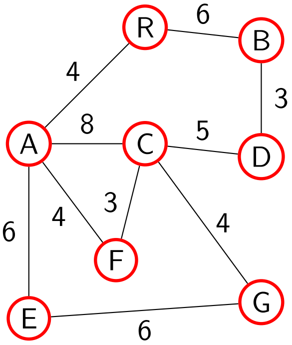
	A	B	C	D	E	F	G	R
A	0	15	7	12	6	4	11	4
B	15	0	8	3	21 / 18	11	12	6
C	7	8	0	5	13 / 10	3	4	11
D	12	3	5	0	18 / 15	8	9	9
E	6	21 / 18	13 / 10	18 / 15	0	10	6	10
F	4	11	3	8	10	0	7	8
G	11	12	4	9	6	7	0	15
R	4	6	11	9	10	8	15	0



Floyds shortest path

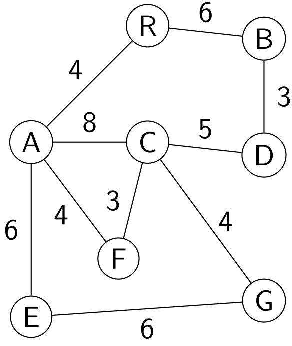
- Efter k=7 (vägar via R).

	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



Floyds shortest path, klar!

	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



Floyd, komplexitet

- ?

```
Algorithm floyd(Graph g)
  input: A graph g to find shortest path in
// Get matrix representation
A ← getMatrix(g)
n ← getNoOfNodes(g)
for k=1 to n do
  for i=1 to n do
    for j=1 to n do
        if A(i,j) > A(i,k) + A(k,j) then
            A(i,j) = A(i,k) + A(k,j)
```

Floyd, komplexitet

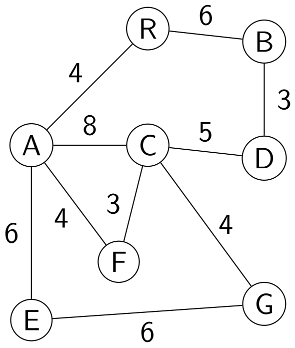
- ?
- Trippel-loop: $O(n^3)$

```
Algorithm floyd(Graph g)
  input: A graph g to find shortest path in
// Get matrix representation
A ← getMatrix(g)
n ← getNoOfNodes(g)
for k=1 to n do
  for i=1 to n do
    for j=1 to n do
      if A(i,j) > A(i,k) + A(k,j) then
        A(i,j) = A(i,k) + A(k,j)
```

Floyds shortest path, hitta kortaste vägen

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

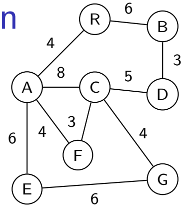
	A	B	C	D	E	F	G	R
A	0	10	7	12	6	4	11	4
B	10	0	8	3	16	11	12	6
C	7	8	0	5	10	3	4	11
D	12	3	5	0	15	8	9	9
E	6	16	10	15	0	10	6	10
F	4	11	3	8	10	0	7	8
G	11	12	4	9	6	7	0	15
R	4	6	11	9	10	8	15	0



Floyds algoritm, modifierad

```
Algorithm floyd(Graph g)
A ← getMatrix(g)
n ← getNoOfNodes(g)
for i=1 to n
  for j=1 to n
    if i==j or A(i,j)==inf then
        Path(i,j) = -1
    else
        Path(i,j) = i // We came to j from i
for k=1 to n
  for i=1 to n
    for j=1 to n
        if A(i,j) > A(i,k) + A(k,j) then
            // Remember new distance...
            A(i,j) = A(i,k) + A(k,j)
            // ...and how we came to j
            Path(i,j) = Path(k,j)
```

Floyds shortest path, hitta kortaste vägen

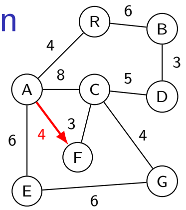


- Vad är kortaste vägen mellan A och C?

	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

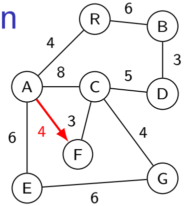


- Vad är kortaste vägen mellan A och C?

	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

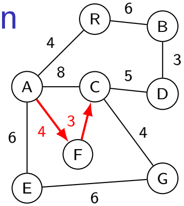


- Vad är kortaste vägen mellan A och C?

	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

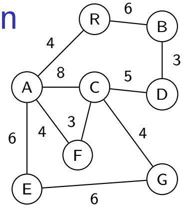


- Vad är kortaste vägen mellan A och C?

	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

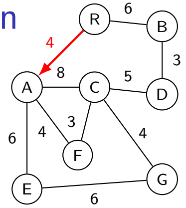


- Vad är kortaste vägen mellan R och G?

	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
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

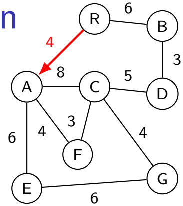


- Vad är kortaste vägen mellan R och G?

	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
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

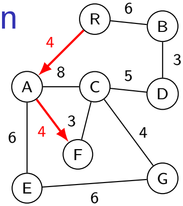


- Vad är kortaste vägen mellan R och G?

	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
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

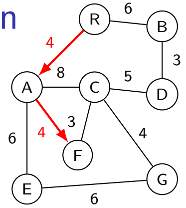


- Vad är kortaste vägen mellan R och G?

	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
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

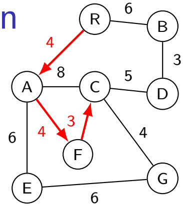


- Vad är kortaste vägen mellan R och G?

	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
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

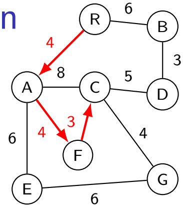


- Vad är kortaste vägen mellan R och G?

	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
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

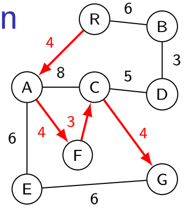


- Vad är kortaste vägen mellan R och G?

	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
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
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	-

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.
Distance	Värdet på den kortaste vägen fram till noden.
Parent	Referens till föregångaren längs vägen.

Dijkstras shortest path, algoritm

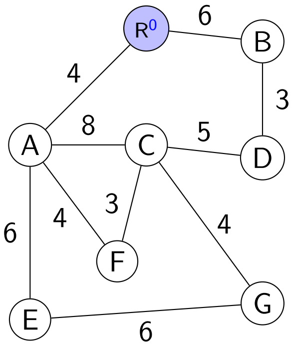
```
Algorithm dijkstra(Node n, Graph g)
  input: A graph g to find shortest path from node n
n.seen ← true; n.distance ← 0; n.parent ← null;
Pqueue q ← empty(); q ← insert(n,q);
while not isempty(q)
  v ← inspect-first(q); q ← delete-first(q);
  vd ← v.distance;
  neighbourSet ← neighbours(v, g);
  for each w in heighbourSet do
    d ← vd + getWeight(v,w);
    if not isSeen(w) then
      w.seen ← true;
      w.distance ← d;
      w.parent ← v;
      q ← insert(w,q);
    else if d < w.distance then
      w.distance ← d;
      w.parent ← v;
      q ← update(w,q)
```

Dijkstras shortest path, visualisering

- Symboler:
  - Aktuell nod har röd ring.
  - Sedda noder är ljusblåa.
  - Noder i prioritetskön har grönstreckad ring.
- Prioritetskön presenteras sorterad.

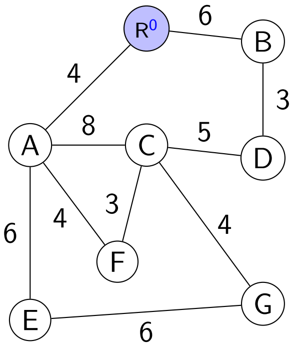
Dijkstras shortest path för oriktad graf

- R.seen = true;
- R.distance = 0;
- R.parent = null;



Dijkstras shortest path för oriktad graf

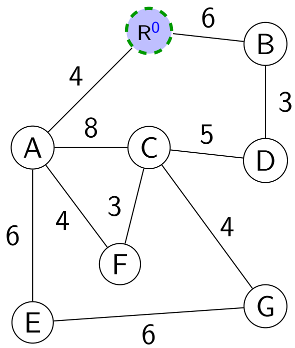
- R.seen = true;
- R.distance = 0;
- R.parent = null;
- Pqueue q = empty();



$q = \{ \}$

Dijkstras shortest path för oriktad graf

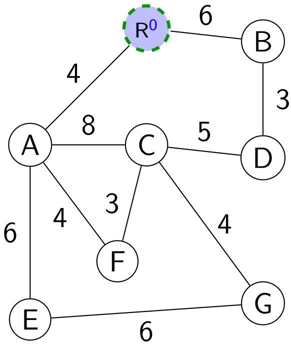
- R.seen = true;
- R.distance = 0;
- R.parent = null;
- Pqueue q = empty();
- q = insert(R(T,0,-),q);



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

Dijkstras shortest path för oriktad graf

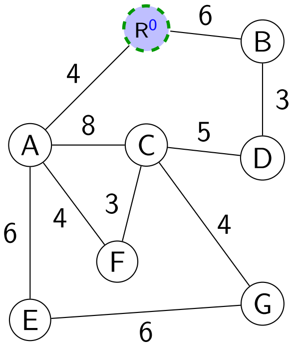
- R.seen = true;
- R.distance = 0;
- R.parent = null;
- Pqueue q = empty();
- q = insert(R(T,0,-),q);
- while not isempty(q)...



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

Dijkstras shortest path för oriktad graf

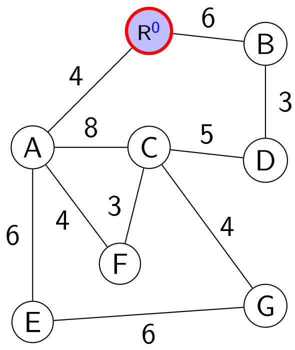
- while not isempty(q)...



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

Dijkstras shortest path för oriktad graf

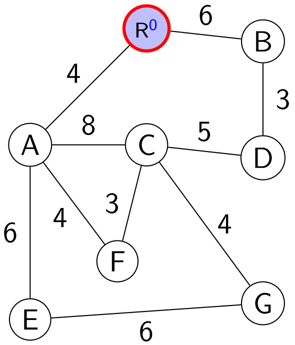
- while not isempty(q)...
  - v = R(T,0,-); q = delete-first(q);



$q = \{\ \}$

Dijkstras shortest path för oriktad graf

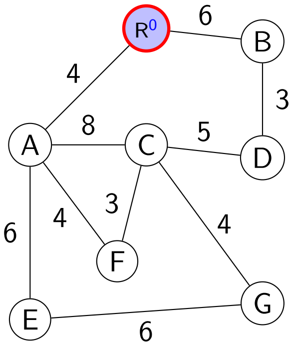
- while not isempty(q)...
  - v = R(T,0,-); q = delete-first(q);
  - v_d = v.distance = 0;
  - neighbourSet = {A,B};



$q = \{\ \}$

Dijkstras shortest path för oriktad graf

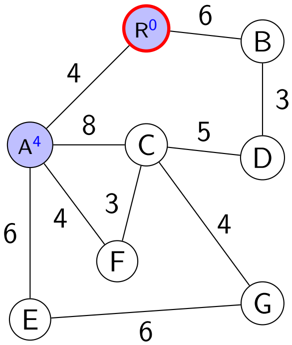
- while not isempty(q)...
  - v = R(T,0,-); q = delete-first(q);
  - v_d = v.distance = 0;
  - neighbourSet = {A,B};
  - A not seen



$q = \{\ \}$

Dijkstras shortest path för oriktad graf

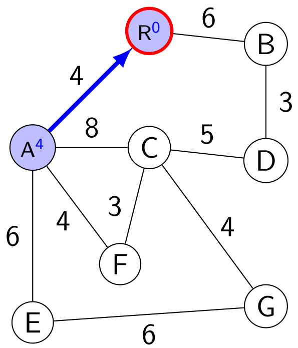
- while not isempty(q)...
  - v = R(T,0,-); q = delete-first(q);
  - $v_d = v$.distance = 0;
  - neighbourSet = {A,B};
  - A not seen
    - $d = v_d$ + getWeight(v,A) = 4;
    - A.seen = true;
    - A.distance = d;



$q = \{\ \}$

Dijkstras shortest path för oriktad graf

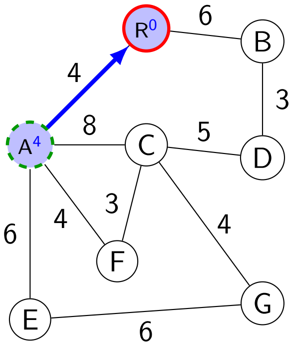
- while not isempty(q)...
  - v = R(T,0,-); q = delete-first(q);
  - v_d = v.distance = 0;
  - neighbourSet = {A,B};
  - A not seen
    - $d = v_d$ + getWeight(v,A) = 4;
    - A.seen = true;
    - A.distance = d;
    - A.parent = R;



$q = \{\ \}$

Dijkstras shortest path för oriktad graf

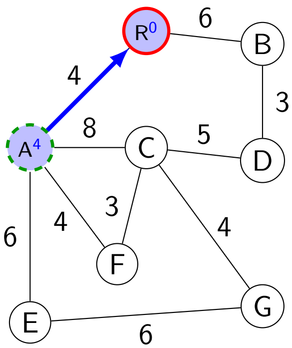
- while not isempty(q)...
  - v = R(T,0,-); q = delete-first(q);
  - v_d = v.distance = 0;
  - neighbourSet = {A,B};
  - A not seen
    - $d = v_d$ + getWeight(v,A) = 4;
    - A.seen = true;
    - A.distance = d;
    - A.parent = R;
    - q = insert(A(T,4,R),q);



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

Dijkstras shortest path för oriktad graf

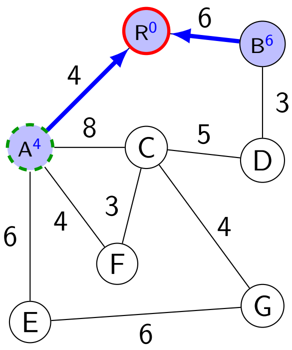
- while not isempty(q)...
  - v = R(T,0,-); q = delete-first(q);
  - v_d = v.distance = 0;
  - neighbourSet = {$\cancel{\text{A}}$,B};
  - B not seen



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

Dijkstras shortest path för oriktad graf

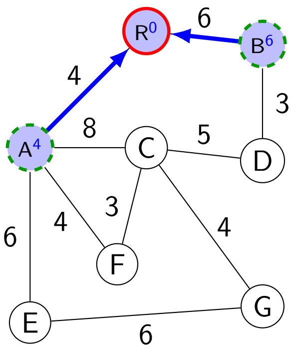
- while not isempty(q)...
  - v = R(T,0,-); q = delete-first(q);
  - v_d = v.distance = 0;
  - neighbourSet = {$\cancel{A}$,B};
  - B not seen
    - $d = v_d$ + getWeight(v,B) = 6;
    - B.seen = true;
    - B.distance = d;
    - B.parent = R;



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

Dijkstras shortest path för oriktad graf

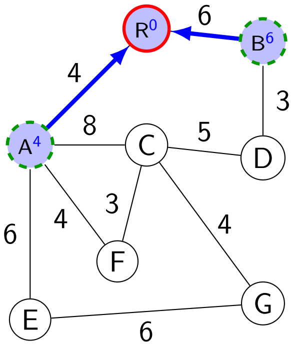
- while not isempty(q)...
  - v = R(T,0,-); q = delete-first(q);
  - $v_d = v$.distance = 0;
  - neighbourSet = {$\cancel{A}$,B};
  - B not seen
    - $d = v_d$ + getWeight(v,B) = 6;
    - B.seen = true;
    - B.distance = d;
    - B.parent = R;
    - q = insert(B(T,6,R),q);



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

Dijkstras shortest path för oriktad graf

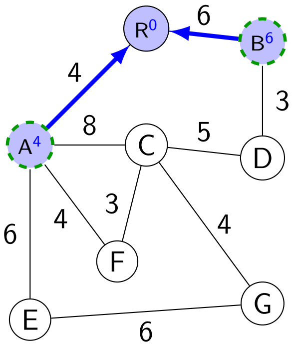
- while not isempty(q)...
  - v = R(T,0,-); q = delete-first(q);
  - v_d = v.distance = 0;
  - neighbourSet = {~~A~~,~~B~~};



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

Dijkstras shortest path för oriktad graf

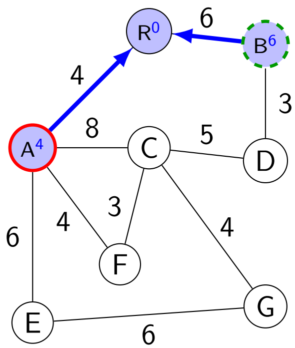
- while not isempty(q)...



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

Dijkstras shortest path för oriktad graf

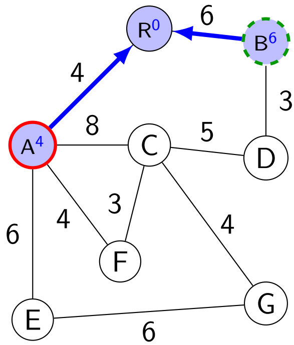
- while not isempty(q)...
  - v = A(T,4,R); q = delete-first(q);



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

Dijkstras shortest path för oriktad graf

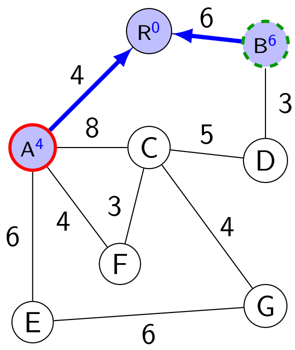
- while not isempty(q)...
  - v = A(T,4,R); q = delete-first(q);
  - $v_d = v$.distance = 4;
  - neighbourSet = {E,R,F,C};



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

Dijkstras shortest path för oriktad graf

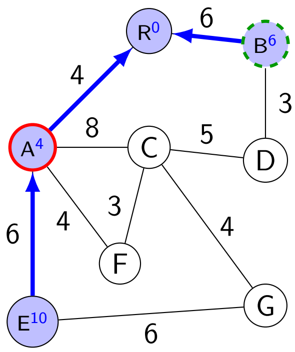
- while not isempty(q)...
  - v = A(T,4,R); q = delete-first(q);
  - $v_d = v$.distance = 4;
  - neighbourSet = {E,R,F,C};
  - E not seen



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

Dijkstras shortest path för oriktad graf

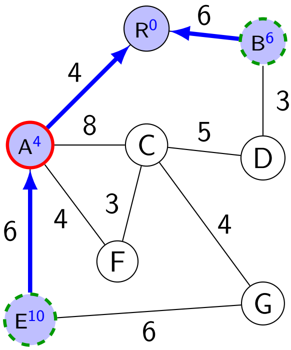
- while not isempty(q)...
  - v = A(T,4,R); q = delete-first(q);
  - v_d = v.distance = 4;
  - neighbourSet = {E,R,F,C};
  - E not seen
    - $d = v_d +$ getWeight(v,E) = 10;
    - E.seen = true;
    - E.distance = d;
    - E.parent = A;



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

Dijkstras shortest path för oriktad graf

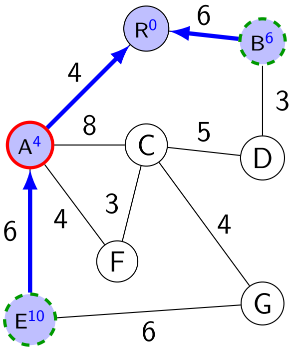
- while not isempty(q)...
  - v = A(T,4,R); q = delete-first(q);
  - $v_d = v$.distance = 4;
  - neighbourSet = {E,R,F,C};
  - E not seen
    - $d = v_d$ + getWeight(v,E) = 10;
    - E.seen = true;
    - E.distance = d;
    - E.parent = A;
    - q = insert(E(T,10,A),q);



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

Dijkstras shortest path för oriktad graf

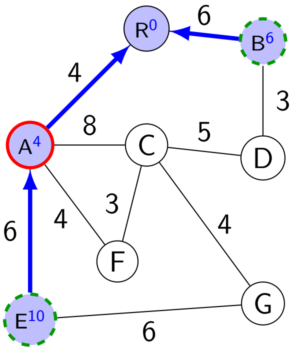
- while not isempty(q)...
  - v = A(T,4,R); q = delete-first(q);
  - v_d = v.distance = 4;
  - neighbourSet = {$\cancel{E}$,R,F,C};
  - R seen



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

Dijkstras shortest path för oriktad graf

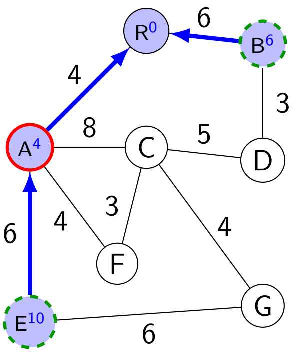
- while not isempty(q)...
  - v = A(T,4,R); q = delete-first(q);
  - v_d = v.distance = 4;
  - neighbourSet = {$\cancel{E}$,R,F,C};
  - R seen
    - $d = v_d$ + getWeight(v,R) = 8;
    - d **not** < R.distance



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

Dijkstras shortest path för oriktad graf

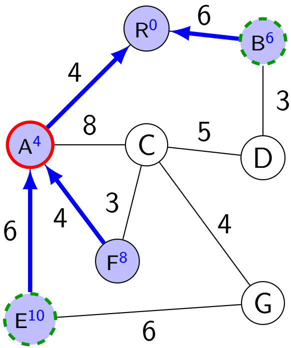
- while not isempty(q)...
  - v = A(T,4,R); q = delete-first(q);
  - v_d = v.distance = 4;
  - neighbourSet = {$\cancel{E}$,$\cancel{R}$,F,C};
  - F not seen



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

Dijkstras shortest path för oriktad graf

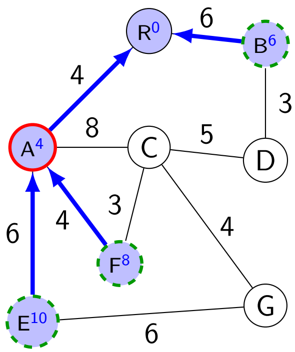
- while not isempty(q)...
  - v = A(T,4,R); q = delete-first(q);
  - v_d = v.distance = 4;
  - neighbourSet = {$\cancel{E}$,$\cancel{R}$,F,C};
  - F not seen
    - $d = v_d$ + getWeight(v,F) = 8;
    - F.seen = true;
    - F.distance = d;
    - F.parent = A;



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

Dijkstras shortest path för oriktad graf

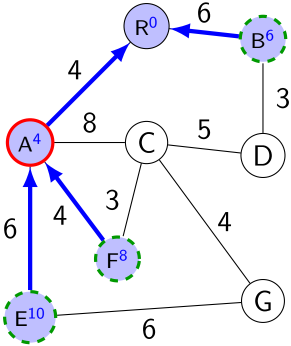
- while not isempty(q)...
  - v = A(T,4,R); q = delete-first(q);
  - v_d = v.distance = 4;
  - neighbourSet = {$\cancel{E}$,$\cancel{R}$,F,C};
  - F not seen
    - $d = v_d$ + getWeight(v,F) = 8;
    - F.seen = true;
    - F.distance = d;
    - F.parent = A;
    - q = insert(F(T,8,A),q);



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

Dijkstras shortest path för oriktad graf

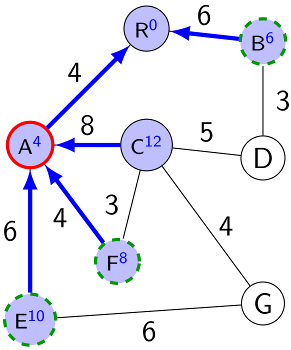
- while not isempty(q)...
  - v = A(T,4,R); q = delete-first(q);
  - v_d = v.distance = 4;
  - neighbourSet = {$\cancel{E}$,$\cancel{R}$,$\cancel{F}$,C};
  - C not seen



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

Dijkstras shortest path för oriktad graf

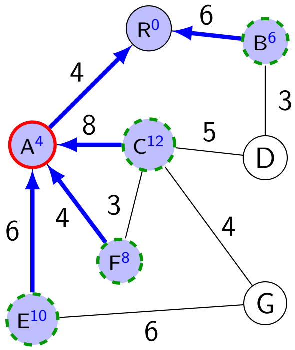
- while not isempty(q)...
  - v = A(T,4,R); q = delete-first(q);
  - $v_d = v$.distance = 4;
  - neighbourSet = {$\cancel{\text{E}}$,$\cancel{\text{R}}$,$\cancel{\text{F}}$,C};
  - C not seen
    - $d = v_d$ + getWeight(v,C) = 12;
    - C.seen = true;
    - C.distance = d;
    - C.parent = A;



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

Dijkstras shortest path för oriktad graf

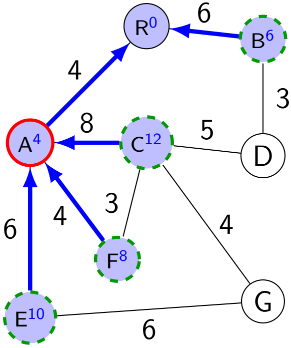
- while not isempty(q)...
  - v = A(T,4,R); q = delete-first(q);
  - v_d = v.distance = 4;
  - neighbourSet = {$\cancel{E}$,$\cancel{R}$,$\cancel{F}$,C};
  - C not seen
    - $d = v_d$ + getWeight(v,C) = 12;
    - C.seen = true;
    - C.distance = d;
    - C.parent = A;
    - q = insert(C(T,12,A),q);



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

Dijkstras shortest path för oriktad graf

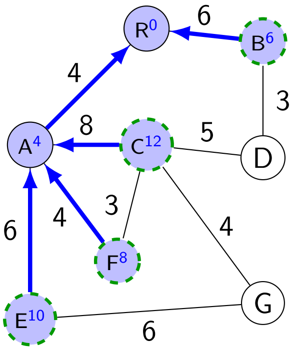
- while not isempty(q)...
  - v = A(T,4,R); q = delete-first(q);
  - $v_d = v$.distance = 4;
  - neighbourSet = {$\cancel{E}$,$\cancel{R}$,$\cancel{F}$,$\cancel{C}$};



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

Dijkstras shortest path för oriktad graf

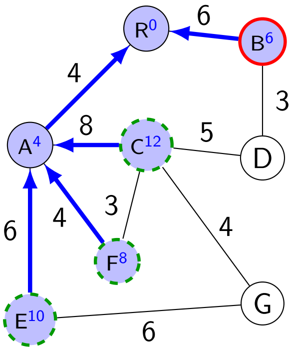
- while not isempty(q)...



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

Dijkstras shortest path för oriktad graf

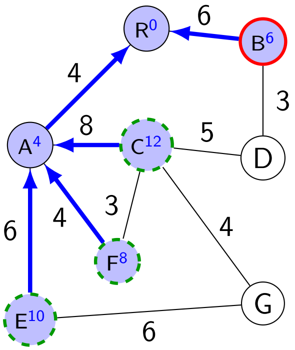
- while not isempty(q)...
  - v = B(T,6,R); q = delete-first(q);



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

Dijkstras shortest path för oriktad graf

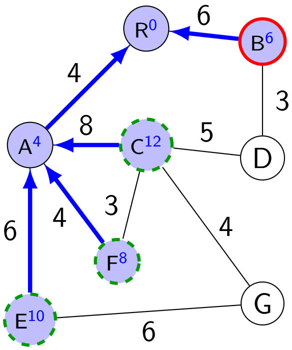
- while not isempty(q)...
  - v = B(T,6,R); q = delete-first(q);
  - v_d = v.distance = 6;
  - neighbourSet = {D,R};



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

Dijkstras shortest path för oriktad graf

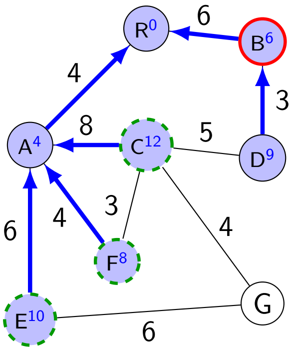
- while not isempty(q)...
  - v = B(T,6,R); q = delete-first(q);
  - $v_d = v$.distance = 6;
  - neighbourSet = {D,R};
  - D not seen



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

Dijkstras shortest path för oriktad graf

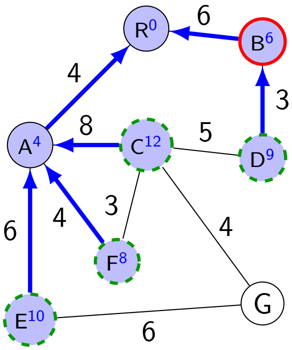
- while not isempty(q)...
  - v = B(T,6,R); q = delete-first(q);
  - $v_d = v$.distance = 6;
  - neighbourSet = {D,R};
  - D not seen
    - $d = v_d$ + getWeight(v,D) = 9;
    - D.seen = true;
    - D.distance = d;
    - D.parent = B;



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

Dijkstras shortest path för oriktad graf

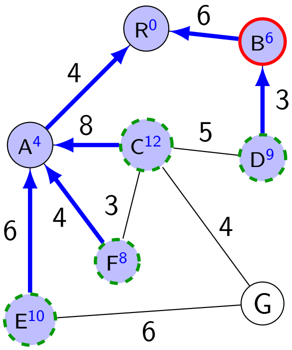
- while not isempty(q)...
  - v = B(T,6,R); q = delete-first(q);
  - $v_d = v$.distance = 6;
  - neighbourSet = {D,R};
  - D not seen
    - $d = v_d$ + getWeight(v,D) = 9;
    - D.seen = true;
    - D.distance = d;
    - D.parent = B;
    - q = insert(D(T,9,B),q);



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

Dijkstras shortest path för oriktad graf

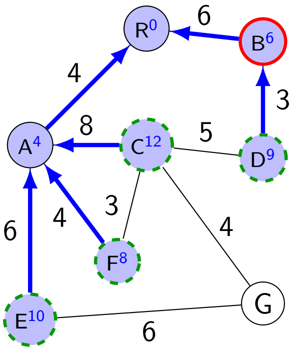
- while not isempty(q)...
  - v = B(T,6,R); q = delete-first(q);
  - $v_d = v$.distance = 6;
  - neighbourSet = {~~D~~,R};
  - R seen



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

Dijkstras shortest path för oriktad graf

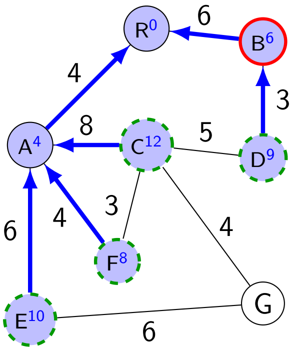
- while not isempty(q)...
  - v = B(T,6,R); q = delete-first(q);
  - $v_d = v$.distance = 6;
  - neighbourSet = {~~D~~,R};
  - R seen
    - $d = v_d$ + getWeight(v,R) = 12;
    - d **not** < R.distance



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

Dijkstras shortest path för oriktad graf

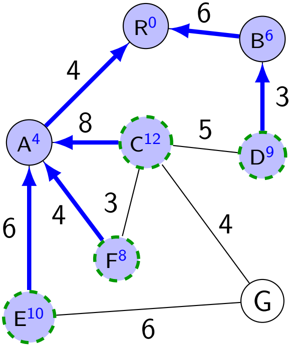
- while not isempty(q)...
  - v = B(T,6,R); q = delete-first(q);
  - $v_d = v$.distance = 6;
  - neighbourSet = {$\cancel{D}$,$\cancel{R}$};



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

Dijkstras shortest path för oriktad graf

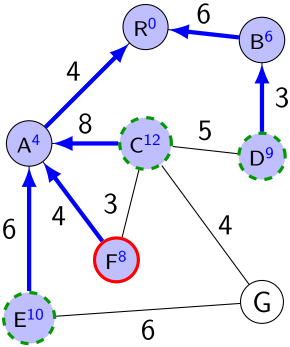
- while not isempty(q)...



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

Dijkstras shortest path för oriktad graf

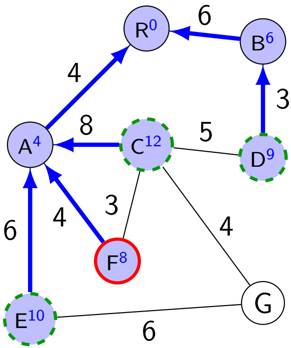
- while not isempty(q)...
  - v = F(T,8,A); q = delete-first(q);



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

Dijkstras shortest path för oriktad graf

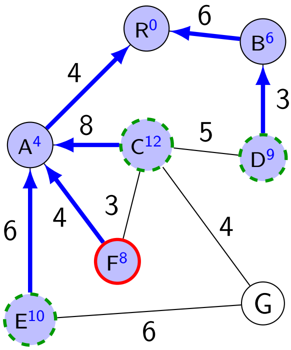
- while not isempty(q)...
  - v = F(T,8,A); q = delete-first(q);
  - v_d = v.distance = 8;
  - neighbourSet = {A,C};



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

Dijkstras shortest path för oriktad graf

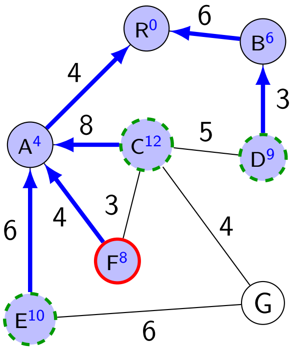
- while not isempty(q)...
  - v = F(T,8,A); q = delete-first(q);
  - v_d = v.distance = 8;
  - neighbourSet = {A,C};
  - A seen



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

Dijkstras shortest path för oriktad graf

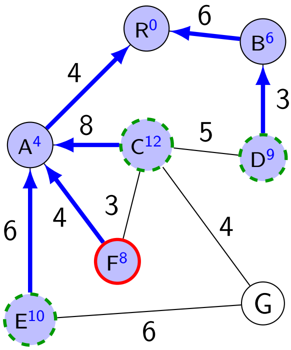
- while not isempty(q)...
  - v = F(T,8,A); q = delete-first(q);
  - $v_d = v$.distance = 8;
  - neighbourSet = {A,C};
  - A seen
    - $d = v_d$ + getWeight(v,A) = 12;
    - d **not** < A.distance



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

Dijkstras shortest path för oriktad graf

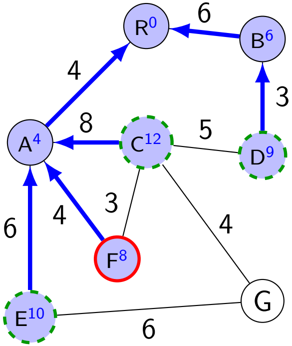
- while not isempty(q)...
  - v = F(T,8,A); q = delete-first(q);
  - v_d = v.distance = 8;
  - neighbourSet = {$\cancel{A}$,C};
  - C seen



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

Dijkstras shortest path för oriktad graf

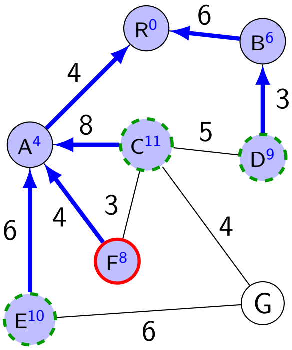
- while not isempty(q)...
  - v = F(T,8,A); q = delete-first(q);
  - $v_d = v$.distance = 8;
  - neighbourSet = {~~A~~,C};
  - C seen
    - $d = v_d$ + getWeight(v,C) = 11;
    - d **is** < C.distance



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

Dijkstras shortest path för oriktad graf

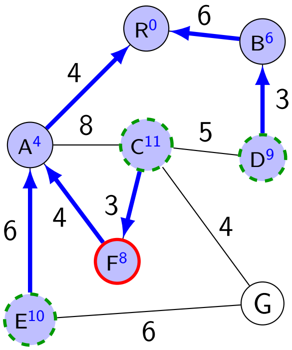
- while not isempty(q)...
  - v = F(T,8,A); q = delete-first(q);
  - v_d = v.distance = 8;
  - neighbourSet = {$\cancel{A}$,C};
  - C seen
    - $d = v_d$ + getWeight(v,C) = 11;
    - d **is** < C.distance
      - C.distance = d;



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

Dijkstras shortest path för oriktad graf

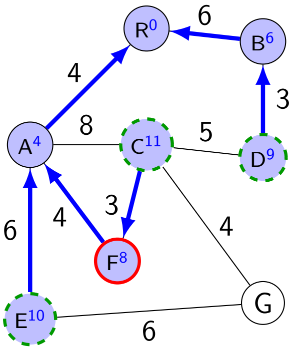
- while not isempty(q)...
  - v = F(T,8,A); q = delete-first(q);
  - v_d = v.distance = 8;
  - neighbourSet = {$\cancel{A}$,C};
  - C seen
    - $d = v_d +$ getWeight(v,C) = 11;
    - d **is** < C.distance
      - C.distance = d;
      - C.parent = F;



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

Dijkstras shortest path för oriktad graf

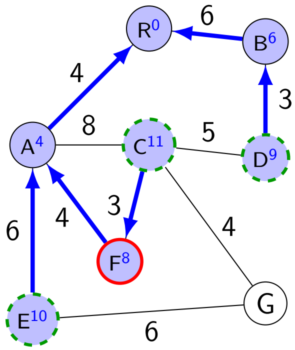
- while not isempty(q)...
  - v = F(T,8,A); q = delete-first(q);
  - v_d = v.distance = 8;
  - neighbourSet = {~~A~~,C};
  - C seen
    - $d = v_d$ + getWeight(v,C) = 11;
    - d **is** $<$ C.distance
      - C.distance = d;
      - C.parent = F;
      - q = update(C,q);



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

Dijkstras shortest path för oriktad graf

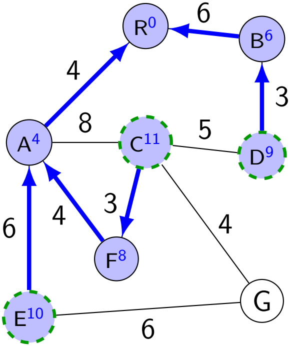
- while not isempty(q)...
  - v = F(T,8,A); q = delete-first(q);
  - v_d = v.distance = 8;
  - neighbourSet = $\{\cancel{A},\cancel{C}\}$;



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

Dijkstras shortest path för oriktad graf

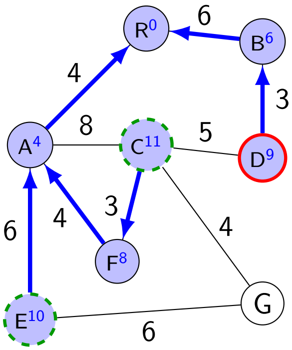
- while not isempty(q)...



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

Dijkstras shortest path för oriktad graf

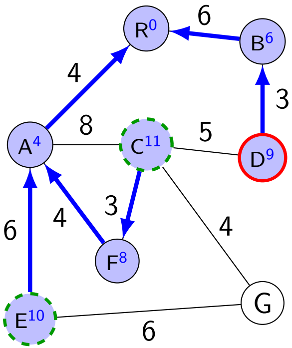
- while not isempty(q)...
  - v = D(T,9,B); q = delete-first(q);



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

Dijkstras shortest path för oriktad graf

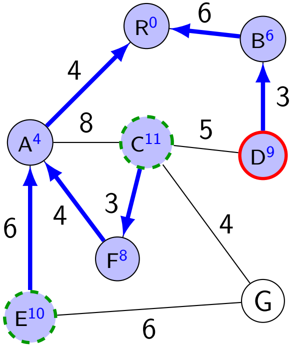
- while not isempty(q)...
  - v = D(T,9,B); q = delete-first(q);
  - v_d = v.distance = 9;
  - neighbourSet = {B,C};



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

Dijkstras shortest path för oriktad graf

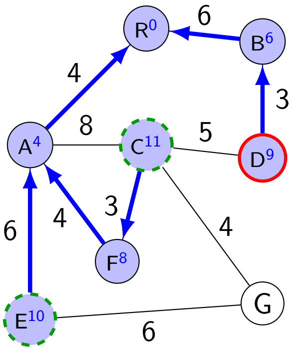
- while not isempty(q)...
  - v = D(T,9,B); q = delete-first(q);
  - v_d = v.distance = 9;
  - neighbourSet = {B,C};
  - B seen



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

Dijkstras shortest path för oriktad graf

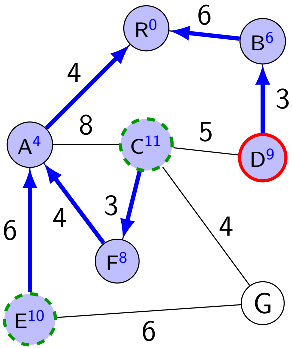
- while not isempty(q)...
  - v = D(T,9,B); q = delete-first(q);
  - v_d = v.distance = 9;
  - neighbourSet = {B,C};
  - B seen
    - $d = v_d + \text{getWeight}(v,\text{B}) = 12$;
    - d **not** < B.distance



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

Dijkstras shortest path för oriktad graf

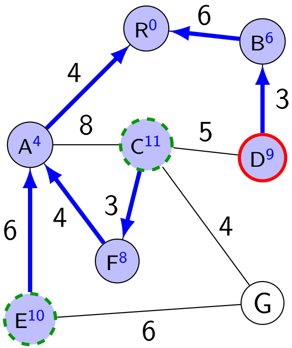
- while not isempty(q)...
  - v = D(T,9,B); q = delete-first(q);
  - $v_d = v$.distance = 9;
  - neighbourSet = {~~B~~,C};
  - C seen



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

Dijkstras shortest path för oriktad graf

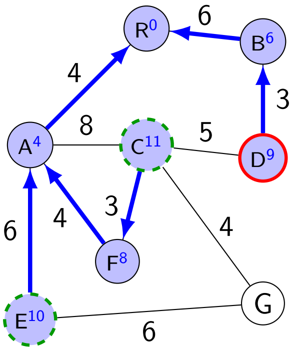
- while not isempty(q)...
  - v = D(T,9,B); q = delete-first(q);
  - $v_d = v$.distance = 9;
  - neighbourSet = {~~B~~,C};
  - C seen
    - $d = v_d$ + getWeight(v,C) = 14;
    - d **not** < C.distance



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

Dijkstras shortest path för oriktad graf

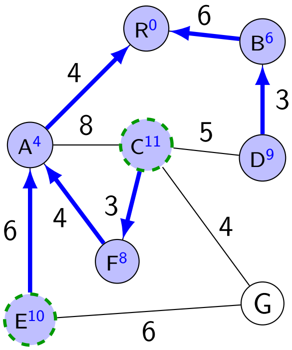
- while not isempty(q)...
  - v = D(T,9,B); q = delete-first(q);
  - v_d = v.distance = 9;
  - neighbourSet = {$\cancel{B}$,$\cancel{C}$};



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

Dijkstras shortest path för oriktad graf

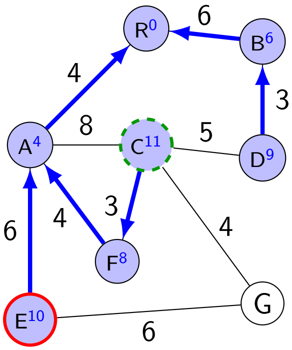
- while not isempty(q)...



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

Dijkstras shortest path för oriktad graf

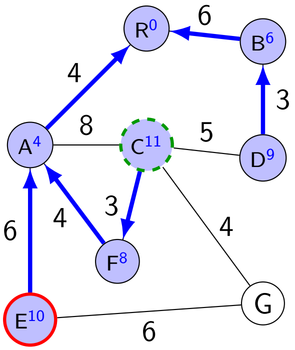
- while not isempty(q)...
  - v = E(T,10,A); q = delete-first(q);



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

Dijkstras shortest path för oriktad graf

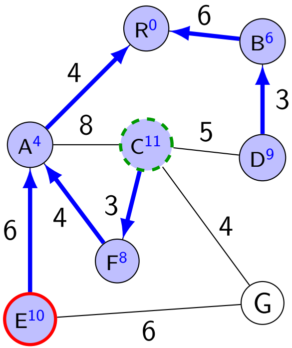
- while not isempty(q)...
  - v = E(T,10,A); q = delete-first(q);
  - v_d = v.distance = 10;
  - neighbourSet = {A,G};



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

Dijkstras shortest path för oriktad graf

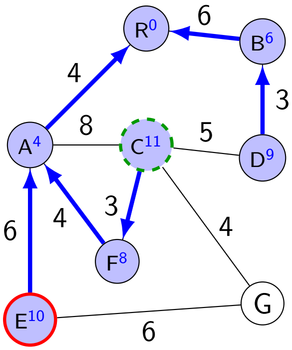
- while not isempty(q)...
  - v = E(T,10,A); q = delete-first(q);
  - $v_d = v$.distance = 10;
  - neighbourSet = {A,G};
  - A seen



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

Dijkstras shortest path för oriktad graf

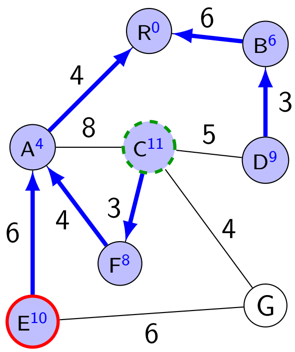
- while not isempty(q)...
  - v = E(T,10,A); q = delete-first(q);
  - v_d = v.distance = 10;
  - neighbourSet = {A,G};
  - A seen
    - $d = v_d$ + getWeight(v,A) = 16;
    - d **not** < A.distance



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

Dijkstras shortest path för oriktad graf

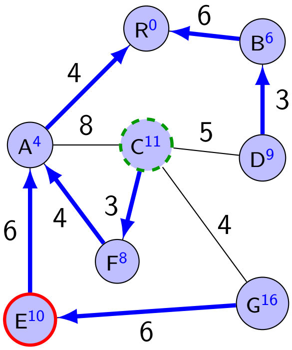
- while not isempty(q)...
  - v = E(T,10,A); q = delete-first(q);
  - v_d = v.distance = 10;
  - neighbourSet = {~~A~~,G};
  - G not seen



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

Dijkstras shortest path för oriktad graf

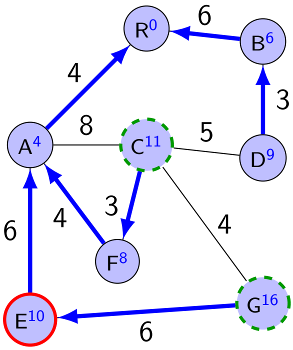
- while not isempty(q)...
  - v = E(T,10,A); q = delete-first(q);
  - v_d = v.distance = 10;
  - neighbourSet = {$\cancel{A}$,G};
  - G not seen
    - $d = v_d$ + getWeight(v,G) = 16;
    - G.seen = true;
    - G.distance = d;
    - G.parent = E;



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

Dijkstras shortest path för oriktad graf

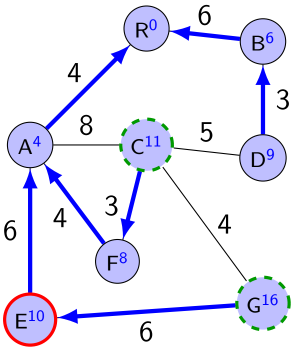
- while not isempty(q)...
  - v = E(T,10,A); q = delete-first(q);
  - v_d = v.distance = 10;
  - neighbourSet = {$\cancel{A}$,G};
  - G not seen
    - $d = v_d$ + getWeight(v,G) = 16;
    - G.seen = true;
    - G.distance = d;
    - G.parent = E;
    - q = insert(G(T,16,E),q);



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

Dijkstras shortest path för oriktad graf

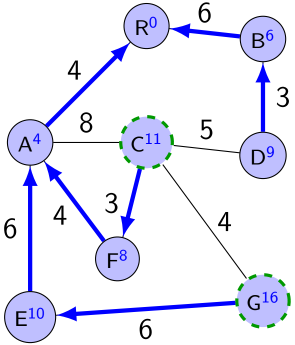
- while not isempty(q)...
  - v = E(T,10,A); q = delete-first(q);
  - v_d = v.distance = 10;
  - neighbourSet = {$\cancel{A}$,$\cancel{G}$};



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

Dijkstras shortest path för oriktad graf

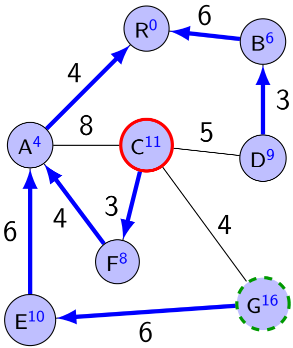
- while not isempty(q)...



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

Dijkstras shortest path för oriktad graf

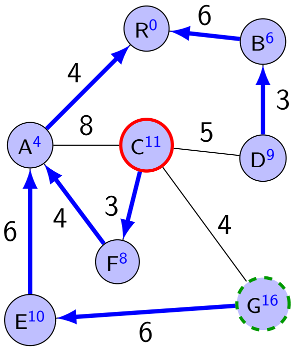
- while not isempty(q)...
  - v = C(T,11,F); q = delete-first(q);



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

Dijkstras shortest path för oriktad graf

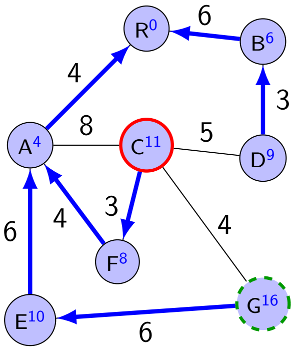
- while not isempty(q)...
  - v = C(T,11,F); q = delete-first(q);
  - v_d = v.distance = 11;
  - neighbourSet = {A,F,G,D};



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

Dijkstras shortest path för oriktad graf

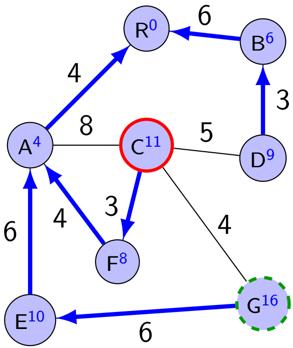
- while not isempty(q)...
  - v = C(T,11,F); q = delete-first(q);
  - v_d = v.distance = 11;
  - neighbourSet = {A,F,G,D};
  - A seen



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

Dijkstras shortest path för oriktad graf

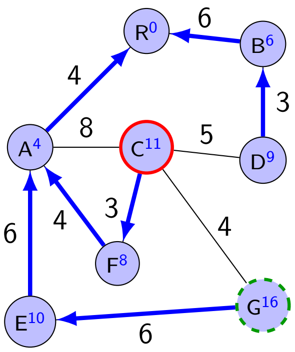
- while not isempty(q)...
  - v = C(T,11,F); q = delete-first(q);
  - $v_d = v$.distance = 11;
  - neighbourSet = {A,F,G,D};
  - A seen
    - $d = v_d$ + getWeight(v,A) = 19;
    - d **not** < A.distance



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

Dijkstras shortest path för oriktad graf

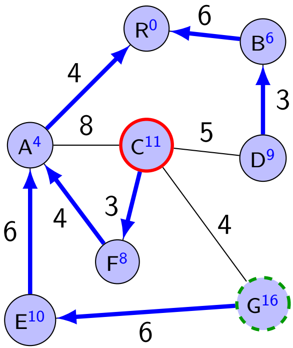
- while not isempty(q)...
  - v = C(T,11,F); q = delete-first(q);
  - v_d = v.distance = 11;
  - neighbourSet = {~~A~~,F,G,D};
  - F seen



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

Dijkstras shortest path för oriktad graf

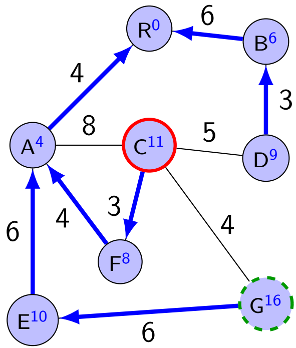
- while not isempty(q)...
  - v = C(T,11,F); q = delete-first(q);
  - $v_d = v$.distance = 11;
  - neighbourSet = {$\cancel{A}$,F,G,D};
  - F seen
    - $d = v_d$ + getWeight(v,F) = 14;
    - d **not** < F.distance



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

Dijkstras shortest path för oriktad graf

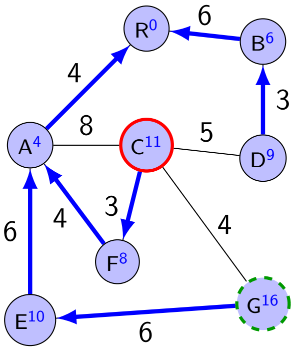
- while not isempty(q)...
  - v = C(T,11,F); q = delete-first(q);
  - v_d = v.distance = 11;
  - neighbourSet = $\{\cancel{A},\cancel{F},G,D\}$;
  - G seen



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

Dijkstras shortest path för oriktad graf

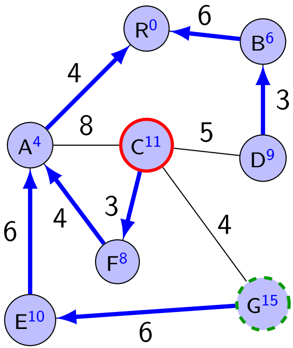
- while not isempty(q)...
  - v = C(T,11,F); q = delete-first(q);
  - $v_d = v$.distance = 11;
  - neighbourSet = {$\cancel{\text{A}}$,$\cancel{\text{F}}$,G,D};
  - G seen
    - $d = v_d$ + getWeight(v,G) = 15;
    - d **is** < G.distance



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

Dijkstras shortest path för oriktad graf

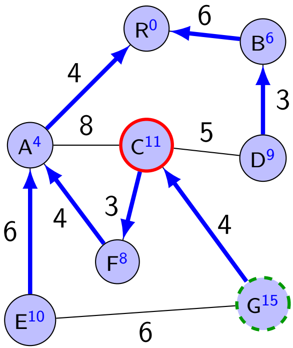
- while not isempty(q)...
  - v = C(T,11,F); q = delete-first(q);
  - $v_d = v$.distance = 11;
  - neighbourSet = {$\cancel{\text{A}}$,$\cancel{\text{F}}$,G,D};
  - G seen
    - $d = v_d$ + getWeight(v,G) = 15;
    - d **is** < G.distance
      - G.distance = d;



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

Dijkstras shortest path för oriktad graf

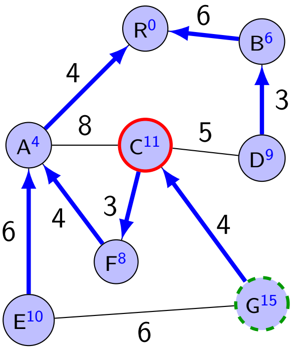
- while not isempty(q)...
  - v = C(T,11,F); q = delete-first(q);
  - $v_d = v$.distance = 11;
  - neighbourSet = {$\cancel{\text{A}}$,$\cancel{\text{F}}$,G,D};
  - G seen
    - $d = v_d$ + getWeight(v,G) = 15;
    - d **is** < G.distance
      - G.distance = d;
      - G.parent = C;



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

Dijkstras shortest path för oriktad graf

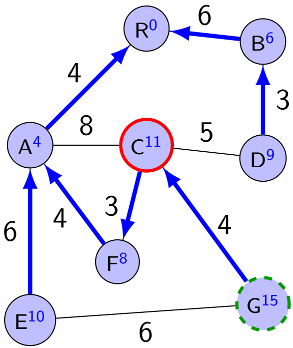
- while not isempty(q)...
  - v = C(T,11,F); q = delete-first(q);
  - v_d = v.distance = 11;
  - neighbourSet = {$\cancel{A}$,$\cancel{F}$,G,D};
  - G seen
    - $d = v_d$ + getWeight(v,G) = 15;
    - d **is** < G.distance
      - G.distance = d;
      - G.parent = C;
      - q = update(G,q);



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

Dijkstras shortest path för oriktad graf

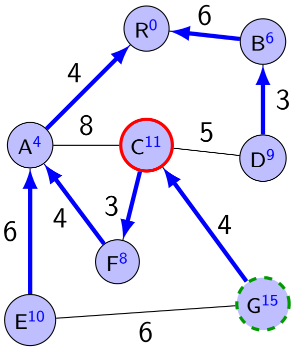
- while not isempty(q)...
  - v = C(T,11,F); q = delete-first(q);
  - $v_d = v$.distance $= 11$;
  - neighbourSet = {~~A~~,~~F~~,~~G~~,D};
  - D seen



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

Dijkstras shortest path för oriktad graf

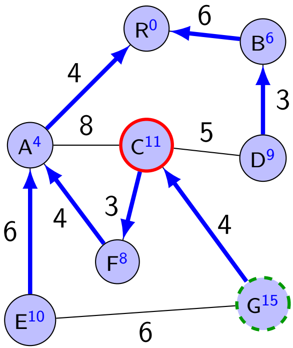
- while not isempty(q)...
  - v = C(T,11,F); q = delete-first(q);
  - v_d = v.distance = 11;
  - neighbourSet = {~~A~~,~~F~~,~~G~~,D};
  - D seen
    - $d = v_d$ + getWeight(v,D) = 16;
    - d **not** < D.distance



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

Dijkstras shortest path för oriktad graf

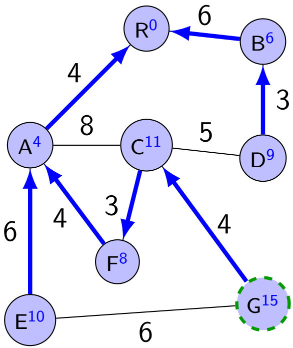
- while not isempty(q)...
  - v = C(T,11,F); q = delete-first(q);
  - v_d = v.distance = 11;
  - neighbourSet = {$\cancel{A}$,$\cancel{F}$,$\cancel{G}$,$\cancel{D}$};



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

Dijkstras shortest path för oriktad graf

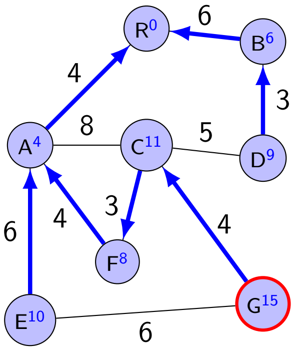
- while not isempty(q)...



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

Dijkstras shortest path för oriktad graf

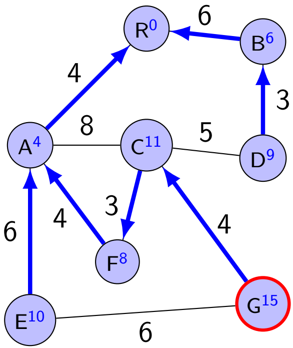
- while not isempty(q)...
  - v = G(T,15,C); q = delete-first(q);



$q = \{\ \}$

Dijkstras shortest path för oriktad graf

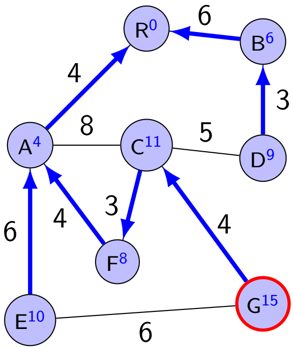
- while not isempty(q)...
  - v = G(T,15,C); q = delete-first(q);
  - v_d = v.distance = 15;
  - neighbourSet = {E,C};



$q = \{\ \}$

Dijkstras shortest path för oriktad graf

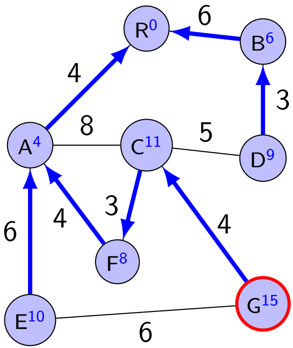
- while not isempty(q)...
  - v = G(T,15,C); q = delete-first(q);
  - v_d = v.distance = 15;
  - neighbourSet = {E,C};
  - E seen



$q = \{\ \}$

Dijkstras shortest path för oriktad graf

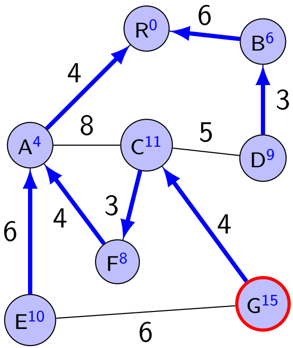
- while not isempty(q)...
  - v = G(T,15,C); q = delete-first(q);
  - v_d = v.distance = 15;
  - neighbourSet = {E,C};
  - E seen
    - $d = v_d$ + getWeight(v,E) = 21;
    - d **not** < E.distance



$q = \{\ \}$

Dijkstras shortest path för oriktad graf

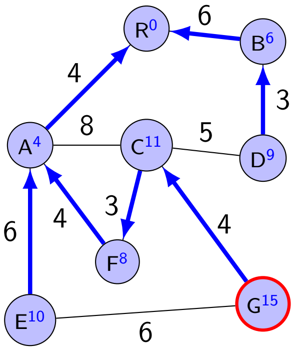
- while not isempty(q)...
  - v = G(T,15,C); q = delete-first(q);
  - v_d = v.distance = 15;
  - neighbourSet = {~~E~~,C};
  - C seen



$q = \{\ \}$

Dijkstras shortest path för oriktad graf

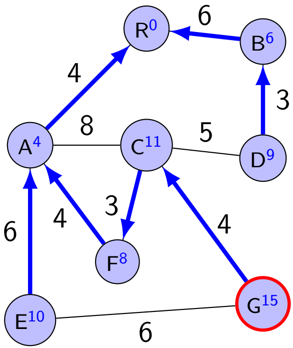
- while not isempty(q)...
  - v = G(T,15,C); q = delete-first(q);
  - v_d = v.distance = 15;
  - neighbourSet = {$\cancel{E}$,C};
  - C seen
    - $d = v_d$ + getWeight(v,C) = 19;
    - d **not** < C.distance



$q = \{ \}$

Dijkstras shortest path för oriktad graf

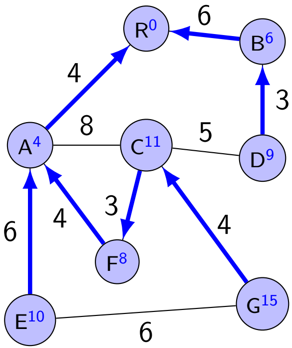
- while not isempty(q)...
  - v = G(T,15,C); q = delete-first(q);
  - v_d = v.distance = 15;
  - neighbourSet = {$\cancel{E}$,$\cancel{C}$};



$q = \{\ \}$

Dijkstras shortest path för oriktad graf

- while not isempty(q)...



$q = \{\ \}$

Dijkstras shortest path, algoritm

```
Algorithm dijkstra(Node n, Graph g)
  input: A graph g to find shortest path from node n
n.seen ← true; n.distance ← 0; n.parent ← null;
Pqueue q ← empty(); q ← insert(n,q);
while not isempty(q)
  v ← inspect-first(q); q ← delete-first(q);
  vd ← v.distance;
  neighbourSet ← neighbours(v, g);
  for each w in heighbourSet do
    d ← vd + getWeight(v,w);
    if not isSeen(w)
      w.seen ← true;
      w.distance ← d;
      w.parent ← v;
      q ← insert(w,q);
    else if d < w.distance
      w.distance ← d;
      w.parent ← v;
      q ← update(w,q)
```

Dijkstras shortest path, komplexitet

- Vi sätter in varje nod i prioritetskön en gång.
  - Totalt $n \cdot O(\text{insert})$.
- Vi tar ut varje nod ur prioritetskön en gång.
  - Totalt $n \cdot O(\text{delete-first})$.
- Vi kan behöva uppdatera element i prioritetskön.
  - Maximalt m gånger: $m \cdot O(\text{update})$.
- Osorterad lista (via referens till noden):
  - $nO(1) + nO(n) + mO(1) = O(n^2 + m)$.
- Sorterad lista:
  - $nO(n) + nO(1) + mO(n) = O(n^2 + mn)$.
- Heap:
  - $nO(\log n) + nO(\log n) + mO(\log n) = O((n + m)\log n)$.

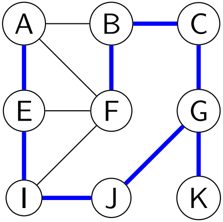
Komplexitet alla-till-alla: Floyd vs. Dijkstra

- Floyd: $O(n^3)$.
- Snabbaste Dijkstra: $O((n+m)\log n)$ för en-till-alla.
  - Måste köras n gånger för att få alla-till-alla:
    - $O(n(n+m)\log n) = O(n^2 \log n + mn \log n)$.
    - För gles graf $m \approx n$: $O(n^2 \log n)$.
    - För tät graf $m \approx n^2$: $O(n^3 \log n)$.
- Djikstra snabbare på stora, glesa grafer.

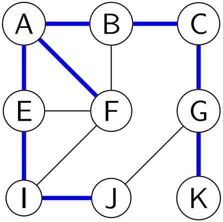
3. Minsta uppspännande träd

Uppspännande träd, oviktad graf

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



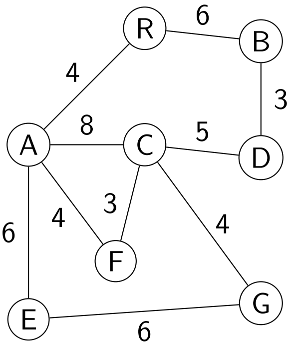
  - Bredden-först:



- 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?
  - Man söker ett uppspännande träd med minsta möjliga totala längd.
    - Det är alltså *inte* en kortaste-vägen-algoritm.
  - För navigeringsorienterad specifikation finns Prims algoritm.
  - För mängdorienterad specifikation finns Kruskals algoritm.



Prims algoritm för minsta uppspännande träd (1)

- Utgå från godtycklig startnod.
- Bygg upp ett större och större träd som till slut spänner upp grafen eller en sammanhängande komponent av den.
- I varje steg, bygg på trädet med en båge med minimal vikt.

Prims algoritm för minsta uppspännande träd (2)

- Välj godtycklig startnod n ur grafen. 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 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).
  - Låt $n = w$. (Byt till den nya noden.)

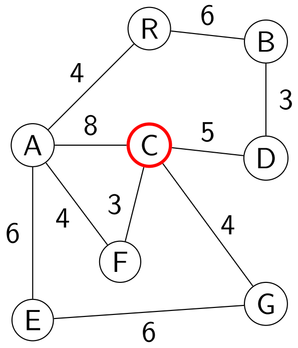
  tills q är tom.

Symboler

- Stängda noder färgas ljusblått.
- Aktuell nod ritas med röd cirkel.
- Bågar i prioritetskön ritas grönstreckade.

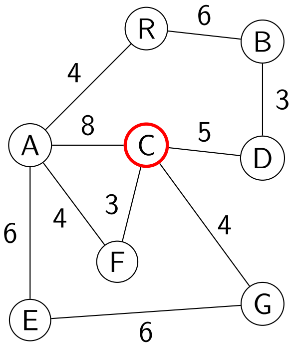
Prims minsta uppspännande träd, exempel

- $n \leftarrow$ C.



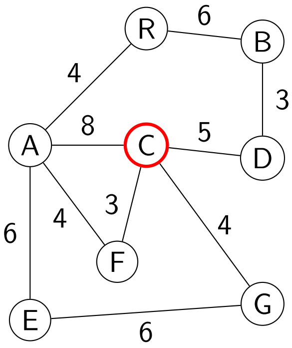
Prims minsta uppspännande träd, exempel

- $n \leftarrow$ C.
- Låt n blir rot i trädet.



Prims minsta uppspännande träd, exempel

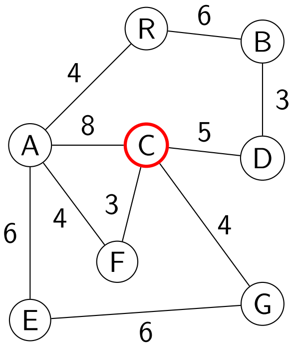
- $n \leftarrow$ C.
- Låt n blir rot i trädet.
- Skapa en tom prioritetskö q.



$q = \{\ \}$

Prims minsta uppspännande träd, exempel

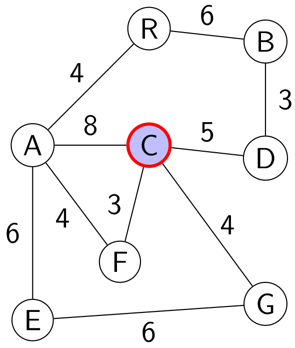
- $n \leftarrow$ C.
- Låt n blir rot i trädet.
- Skapa en tom prioritetskö q.
- Upprepa:



$q = \{\ \}$

Prims minsta uppspännande träd, exempel

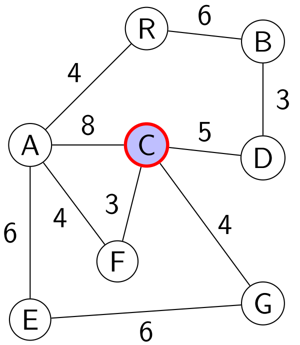
- Upprepa:
  - Markera C som stängd.



$q = \{\ \}$

Prims minsta uppspännande träd, exempel

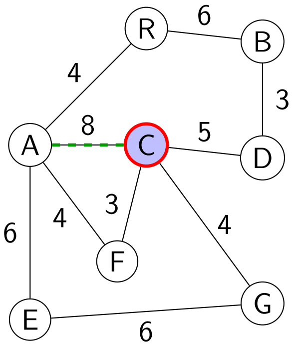
- Upprepa:
  - Markera C som stängd.
  - För var och en av de icke-stängda grannarna {A,F,G,D} till C:



q={ }

Prims minsta uppspännande träd, exempel

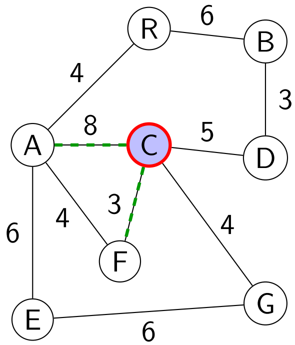
- Upprepa:
  - Markera C som stängd.
  - För var och en av de icke-stängda grannarna {A,F,G,D} till C:
    - Lägg (C,A,8) till q.



q={ (C,A,8) }

Prims minsta uppspännande träd, exempel

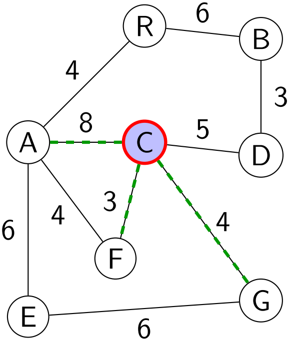
- Upprepa:
  - Markera C som stängd.
  - För var och en av de icke-stängda grannarna {A,F,G,D} till C:
    - Lägg (C,F,3) till q.



q={ (C,F,3), (C,A,8) }

Prims minsta uppspännande träd, exempel

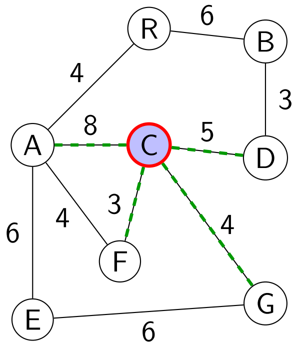
- Upprepa:
  - Markera C som stängd.
  - För var och en av de icke-stängda grannarna {A,F,G,D} till C:
    - Lägg (C,G,4) till q.



q={ (C,F,3), (C,G,4), (C,A,8) }

Prims minsta uppspännande träd, exempel

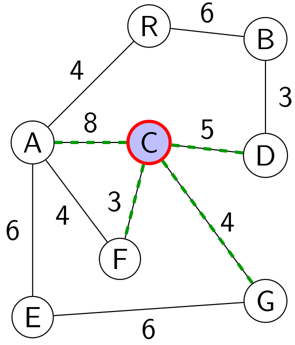
- Upprepa:
  - Markera C som stängd.
  - För var och en av de icke-stängda grannarna {A,F,G,D} till C:
    - Lägg (C,D,5) till q.



q={ (C,F,3), (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

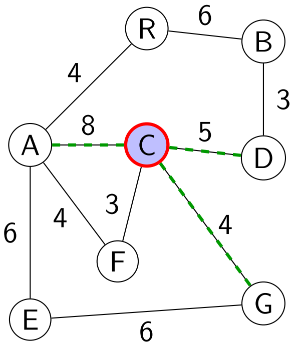
- Upprepa:
  - Markera C som stängd.
  - För var och en av de icke-stängda grannarna {A,F,G,D} till C:
  - Upprepa:



q={ (C,F,3), (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

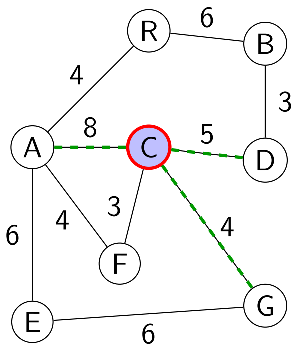
- Upprepa:
  - Markera C som stängd.
  - För var och en av de icke-stängda grannarna {A,F,G,D} till C:
  - Upprepa:
    - Ta (n, w, d)=(C,F,3) från q.



q={ (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

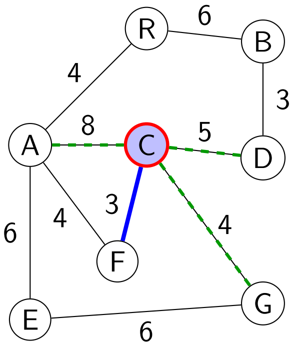
- Upprepa:
  - Markera C som stängd.
  - För var och en av de icke-stängda grannarna {A,F,G,D} till C:
  - Upprepa:
    - Ta (n, w, d)=(C,F,3) från q.
    - F ej stängd.



q={ (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

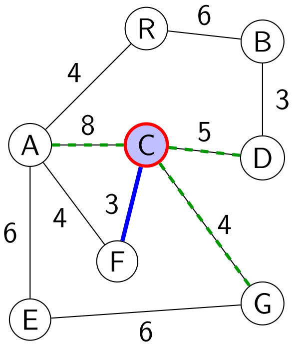
- Upprepa:
  - Markera C som stängd.
  - För var och en av de icke-stängda grannarna {A,F,G,D} till C:
  - Upprepa:
    - Ta (n, w, d)=(C,F,3) från q.
    - F ej stängd.
      - Lägg (C,F,3) till trädet.



q={ (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

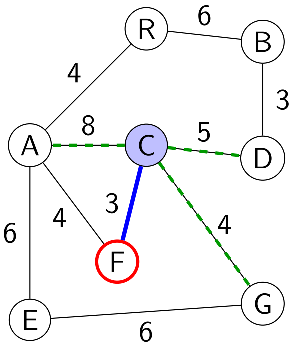
- Upprepa:
  - Markera C som stängd.
  - För var och en av de icke-stängda grannarna {A,F,G,D} till C:
  - Upprepa:
    - Ta (n, w, d)=(C,F,3) från q.
    - F ej stängd.
      - Lägg (C,F,3) till trädet.
  - tills F ej stängd eller q är tom.



q={ (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

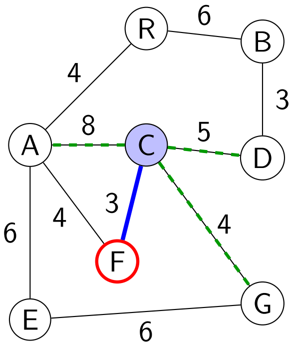
- Upprepa:
  - Markera C som stängd.
  - För var och en av de icke-stängda grannarna {A,F,G,D} till C:
  - Upprepa:
    - Ta (n, w, d)=(C,F,3) från q.
    - F ej stängd.
      - Lägg (C,F,3) till trädet.
  - tills F ej stängd eller q är tom.
  - $n \leftarrow$ F.



q={ (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

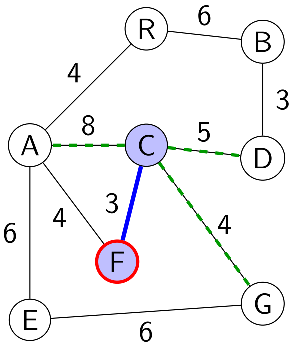
- Upprepa:
  - Markera C som stängd.
  - För var och en av de icke-stängda grannarna {A,F,G,D} till C:
  - Upprepa:
    - Ta (n, w, d)=(C,F,3) från q.
    - F ej stängd.
      - Lägg (C,F,3) till trädet.
  - tills F ej stängd eller q är tom.
  - $n \leftarrow$ F.
- tills q är tom.



q={ (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

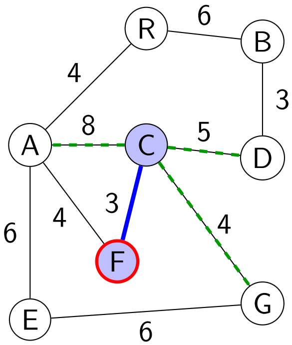
- Upprepa:
  - Markera F som stängd.



q={ (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

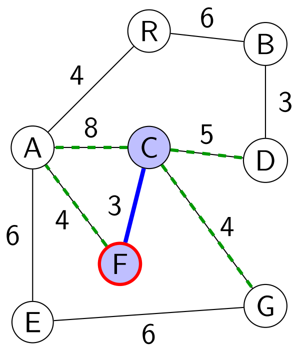
- Upprepa:
  - Markera F som stängd.
  - För var och en av de icke-stängda grannarna {A} till F:



q={ (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

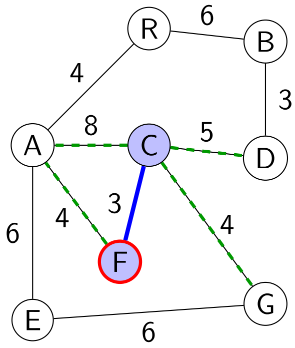
- Upprepa:
  - Markera F som stängd.
  - För var och en av de icke-stängda grannarna {A} till F:
    - Lägg (F,A,4) till q.



q={ (F,A,4), (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

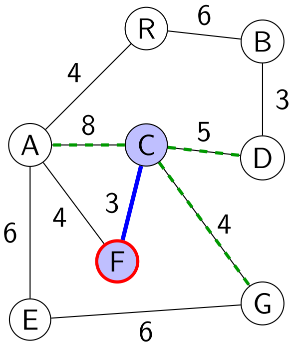
- Upprepa:
  - Markera F som stängd.
  - För var och en av de icke-stängda grannarna {A} till F:
  - Upprepa:



q={ (F,A,4), (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

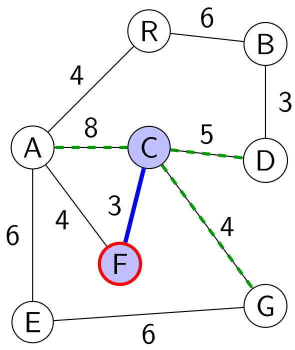
- Upprepa:
  - Markera F som stängd.
  - För var och en av de icke-stängda grannarna {A} till F:
  - Upprepa:
    - Ta (n, w, d)=(F,A,4) från q.



q={ (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

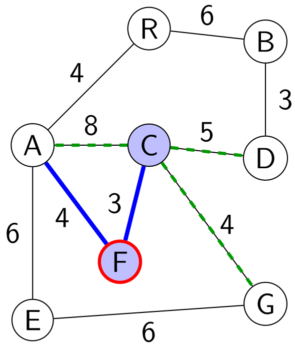
- Upprepa:
  - Markera F som stängd.
  - För var och en av de icke-stängda grannarna {A} till F:
  - Upprepa:
    - Ta (n, w, d)=(F,A,4) från q.
    - A ej stängd.



q={ (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

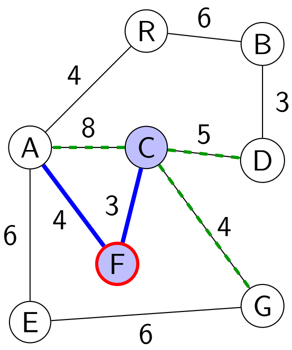
- Upprepa:
  - Markera F som stängd.
  - För var och en av de icke-stängda grannarna {A} till F:
  - Upprepa:
    - Ta (n, w, d)=(F,A,4) från q.
    - A ej stängd.
      - Lägg (F,A,4) till trädet.



q={ (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

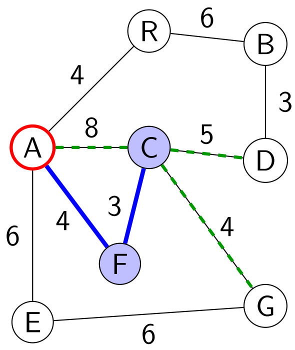
- Upprepa:
  - Markera F som stängd.
  - För var och en av de icke-stängda grannarna {A} till F:
  - Upprepa:
    - Ta (n, w, d)=(F,A,4) från q.
    - A ej stängd.
      - Lägg (F,A,4) till trädet.
  - tills A ej stängd eller q är tom.



q={ (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

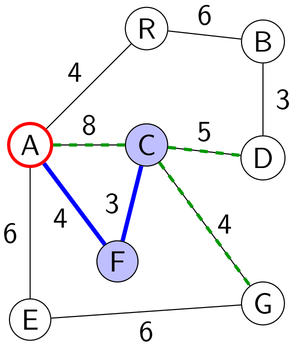
- Upprepa:
  - Markera F som stängd.
  - För var och en av de icke-stängda grannarna {A} till F:
  - Upprepa:
    - Ta (n, w, d)=(F,A,4) från q.
    - A ej stängd.
      - Lägg (F,A,4) till trädet.
  - tills A ej stängd eller q är tom.
  - $n \leftarrow$ A.



q={ (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

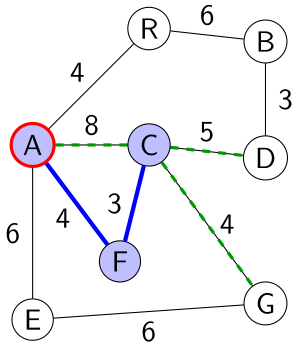
- Upprepa:
  - Markera F som stängd.
  - För var och en av de icke-stängda grannarna {A} till F:
  - Upprepa:
    - Ta (n, w, d)=(F,A,4) från q.
    - A ej stängd.
      - Lägg (F,A,4) till trädet.
  - tills A ej stängd eller q är tom.
  - $n \leftarrow$ A.
- tills q är tom.



q={ (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

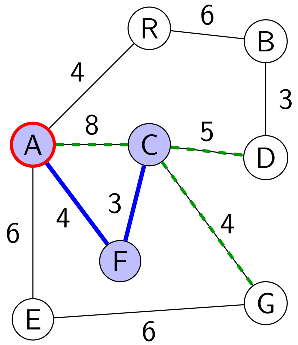
- Upprepa:
  - Markera A som stängd.



q={ (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

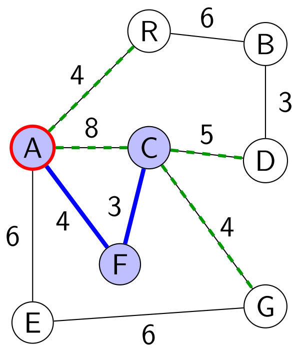
- Upprepa:
  - Markera A som stängd.
  - För var och en av de icke-stängda grannarna {R,E} till A:



q={ (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

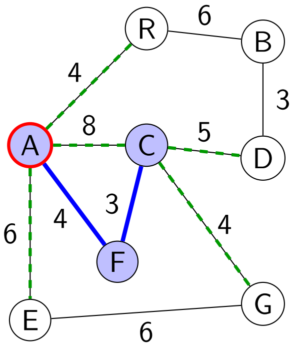
- Upprepa:
  - Markera A som stängd.
  - För var och en av de icke-stängda grannarna {R,E} till A:
    - Lägg (A,R,4) till q.



q={ (A,R,4), (C,G,4), (C,D,5), (C,A,8) }

Prims minsta uppspännande träd, exempel

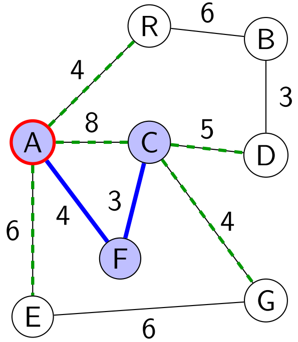
- Upprepa:
  - Markera A som stängd.
  - För var och en av de icke-stängda grannarna {R,E} till A:
    - Lägg (A,E,6) till q.



q={ (A,R,4), (C,G,4), (C,D,5), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

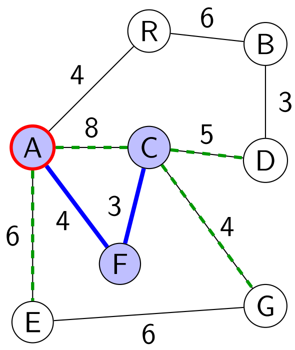
- Upprepa:
  - Markera A som stängd.
  - För var och en av de icke-stängda grannarna $\{R,E\}$ till A:
  - Upprepa:



q={ (A,R,4), (C,G,4), (C,D,5), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

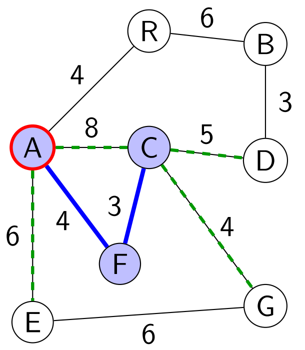
- Upprepa:
  - Markera A som stängd.
  - För var och en av de icke-stängda grannarna {R,E} till A:
  - Upprepa:
    - Ta (n, w, d)=(A,R,4) från q.



q={ (C,G,4), (C,D,5), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

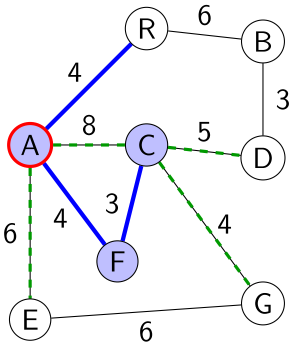
- Upprepa:
  - Markera A som stängd.
  - För var och en av de icke-stängda grannarna {R,E} till A:
  - Upprepa:
    - Ta (n, w, d)=(A,R,4) från q.
    - R ej stängd.



q={ (C,G,4), (C,D,5), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

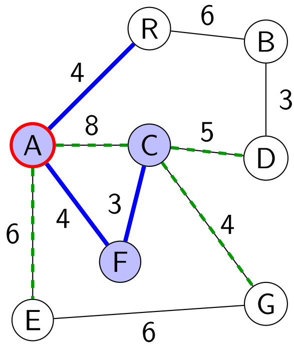
- Upprepa:
  - Markera A som stängd.
  - För var och en av de icke-stängda grannarna {R,E} till A:
  - Upprepa:
    - Ta (n, w, d)=(A,R,4) från q.
    - R ej stängd.
      - Lägg (A,R,4) till trädet.



q={ (C,G,4), (C,D,5), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

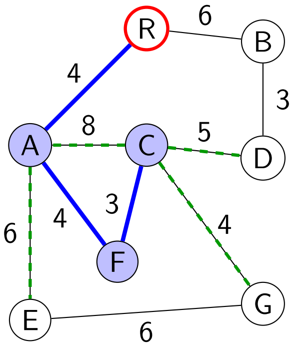
- Upprepa:
  - Markera A som stängd.
  - För var och en av de icke-stängda grannarna {R,E} till A:
  - Upprepa:
    - Ta (n, w, d)=(A,R,4) från q.
    - R ej stängd.
      - Lägg (A,R,4) till trädet.
  - tills R ej stängd eller q är tom.



q={ (C,G,4), (C,D,5), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

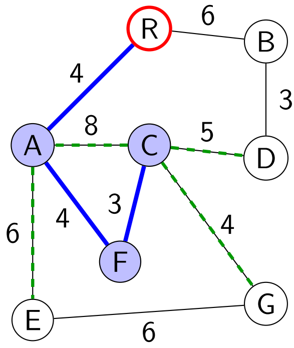
- Upprepa:
  - Markera A som stängd.
  - För var och en av de icke-stängda grannarna {R,E} till A:
  - Upprepa:
    - Ta (n, w, d)=(A,R,4) från q.
    - R ej stängd.
      - Lägg (A,R,4) till trädet.
  - tills R ej stängd eller q är tom.
  - $n \leftarrow$ R.



q={ (C,G,4), (C,D,5), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

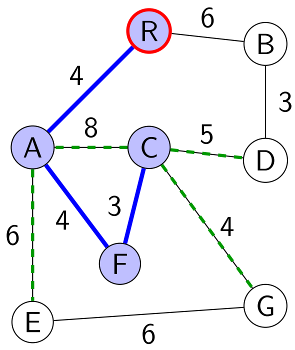
- Upprepa:
  - Markera A som stängd.
  - För var och en av de icke-stängda grannarna {R,E} till A:
  - Upprepa:
    - Ta (n, w, d)=(A,R,4) från q.
    - R ej stängd.
      - Lägg (A,R,4) till trädet.
  - tills R ej stängd eller q är tom.
  - $n \leftarrow$ R.
- tills q är tom.



q={ (C,G,4), (C,D,5), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

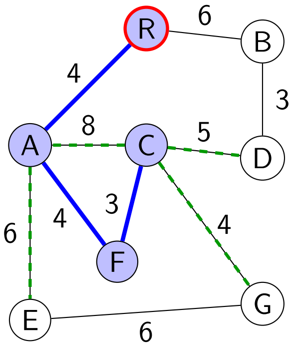
- Upprepa:
  - Markera R som stängd.



q={ (C,G,4), (C,D,5), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

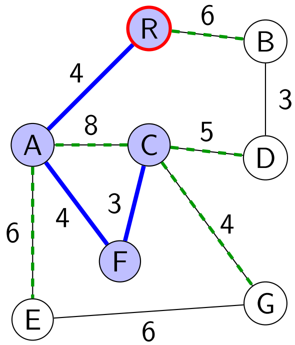
- Upprepa:
  - Markera R som stängd.
  - För var och en av de icke-stängda grannarna {B} till R:



q={ (C,G,4), (C,D,5), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

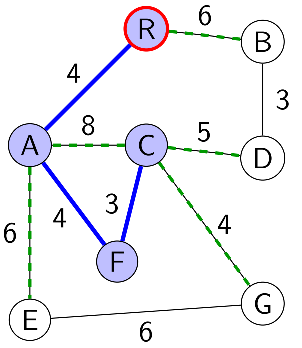
- Upprepa:
  - Markera R som stängd.
  - För var och en av de icke-stängda grannarna {B} till R:
    - Lägg (R,B,6) till q.



q={ (C,G,4), (C,D,5), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

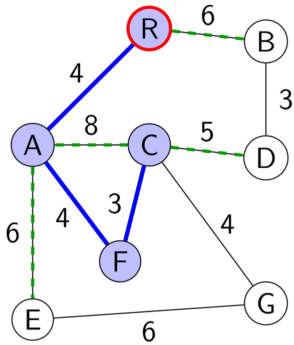
- Upprepa:
  - Markera R som stängd.
  - För var och en av de icke-stängda grannarna {B} till R:
  - Upprepa:



q={ (C,G,4), (C,D,5), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

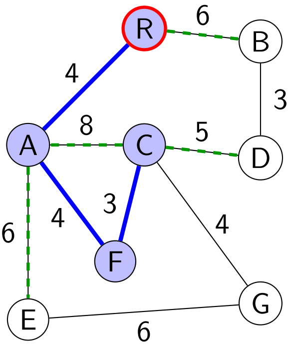
- Upprepa:
  - Markera R som stängd.
  - För var och en av de icke-stängda grannarna {B} till R:
  - Upprepa:
    - Ta (n, w, d)=(C,G,4) från q.



q={ (C,D,5), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

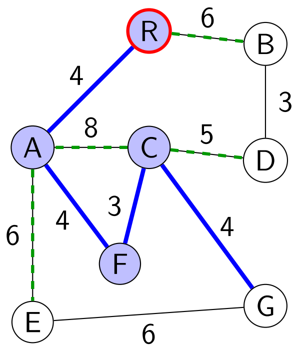
- Upprepa:
  - Markera R som stängd.
  - För var och en av de icke-stängda grannarna {B} till R:
  - Upprepa:
    - Ta (n, w, d)=(C,G,4) från q.
    - G ej stängd.



q={ (C,D,5), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

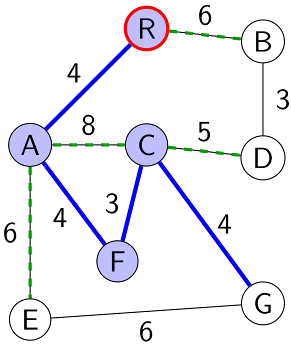
- Upprepa:
  - Markera R som stängd.
  - För var och en av de icke-stängda grannarna {B} till R:
  - Upprepa:
    - Ta (n, w, d)=(C,G,4) från q.
    - G ej stängd.
      - Lägg (C,G,4) till trädet.



q={ (C,D,5), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

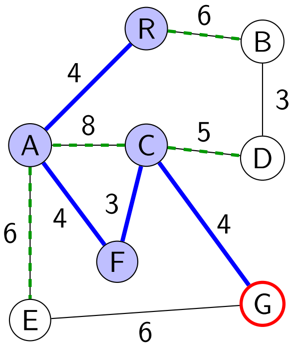
- Upprepa:
  - Markera R som stängd.
  - För var och en av de icke-stängda grannarna {B} till R:
  - Upprepa:
    - Ta (n, w, d)=(C,G,4) från q.
    - G ej stängd.
      - Lägg (C,G,4) till trädet.
  - tills G ej stängd eller q är tom.



q={ (C,D,5), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

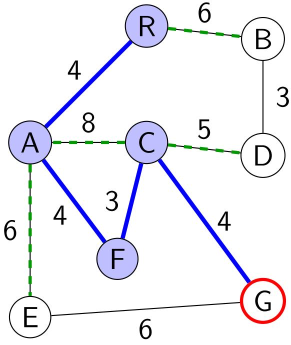
- Upprepa:
  - Markera R som stängd.
  - För var och en av de icke-stängda grannarna {B} till R:
  - Upprepa:
    - Ta (n, w, d)=(C,G,4) från q.
    - G ej stängd.
      - Lägg (C,G,4) till trädet.
  - tills G ej stängd eller q är tom.
  - $n \leftarrow$ G.



q={ (C,D,5), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

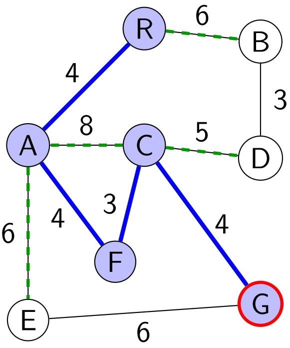
- Upprepa:
  - Markera R som stängd.
  - För var och en av de icke-stängda grannarna {B} till R:
  - Upprepa:
    - Ta (n, w, d)=(C,G,4) från q.
    - G ej stängd.
      - Lägg (C,G,4) till trädet.
  - tills G ej stängd eller q är tom.
  - $n \leftarrow$ G.
- tills q är tom.



q={ (C,D,5), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

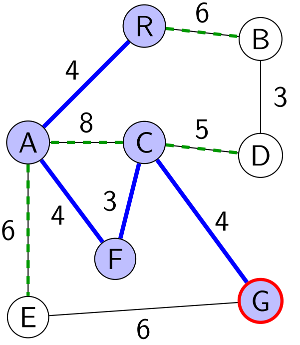
- Upprepa:
  - Markera G som stängd.



q={ (C,D,5), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

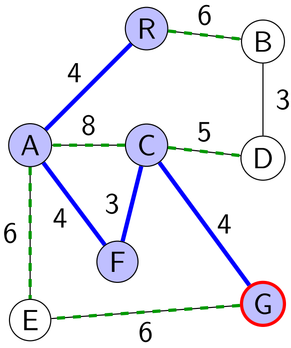
- Upprepa:
  - Markera G som stängd.
  - För var och en av de icke-stängda grannarna {E} till G:



q={ (C,D,5), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

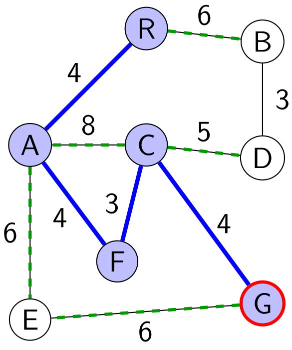
- Upprepa:
  - Markera G som stängd.
  - För var och en av de icke-stängda grannarna {E} till G:
    - Lägg (G,E,6) till q.



q={ (C,D,5), (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

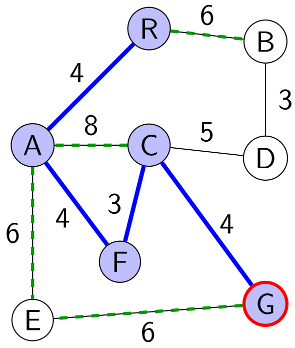
- Upprepa:
  - Markera G som stängd.
  - För var och en av de icke-stängda grannarna {E} till G:
  - Upprepa:



q={ (C,D,5), (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

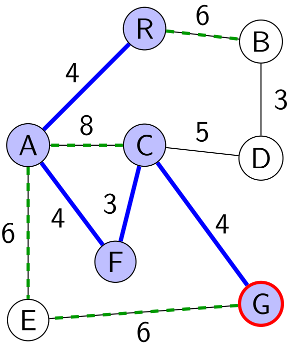
- Upprepa:
  - Markera G som stängd.
  - För var och en av de icke-stängda grannarna {E} till G:
  - Upprepa:
    - Ta (n, w, d)=(C,D,5) från q.



q={ (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

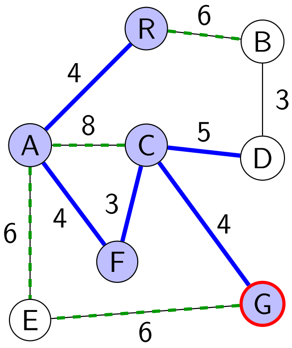
- Upprepa:
  - Markera G som stängd.
  - För var och en av de icke-stängda grannarna {E} till G:
  - Upprepa:
    - Ta (n, w, d)=(C,D,5) från q.
    - D ej stängd.



q={ (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

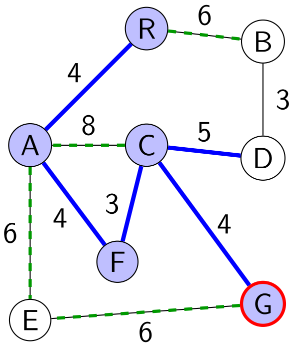
- Upprepa:
  - Markera G som stängd.
  - För var och en av de icke-stängda grannarna {E} till G:
  - Upprepa:
    - Ta (n, w, d)=(C,D,5) från q.
    - D ej stängd.
      - Lägg (C,D,5) till trädet.



q={ (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

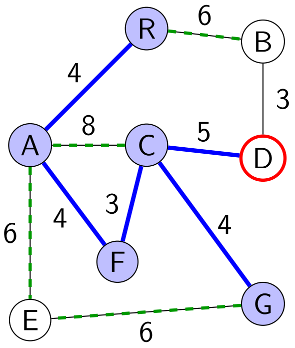
- Upprepa:
  - Markera G som stängd.
  - För var och en av de icke-stängda grannarna {E} till G:
  - Upprepa:
    - Ta (n, w, d)=(C,D,5) från q.
    - D ej stängd.
      - Lägg (C,D,5) till trädet.
  - tills D ej stängd eller q är tom.



q={ (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

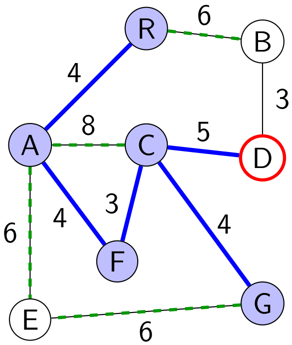
- Upprepa:
  - Markera G som stängd.
  - För var och en av de icke-stängda grannarna {E} till G:
  - Upprepa:
    - Ta (n, w, d)=(C,D,5) från q.
    - D ej stängd.
      - Lägg (C,D,5) till trädet.
  - tills D ej stängd eller q är tom.
  - $n \leftarrow$ D.



q={ (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

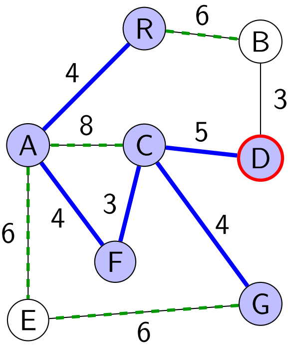
- Upprepa:
  - Markera G som stängd.
  - För var och en av de icke-stängda grannarna {E} till G:
  - Upprepa:
    - Ta (n, w, d)=(C,D,5) från q.
    - D ej stängd.
      - Lägg (C,D,5) till trädet.
  - tills D ej stängd eller q är tom.
  - $n \leftarrow$ D.
- tills q är tom.



q={ (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

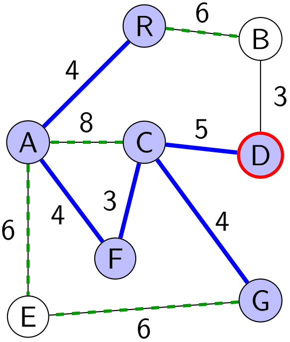
- Upprepa:
  - Markera D som stängd.



q={ (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

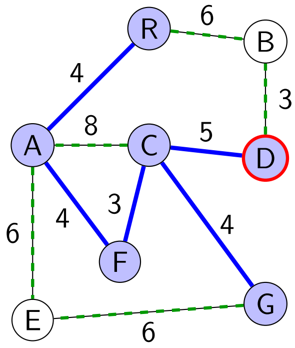
- Upprepa:
  - Markera D som stängd.
  - För var och en av de icke-stängda grannarna {B} till D:



q={ (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

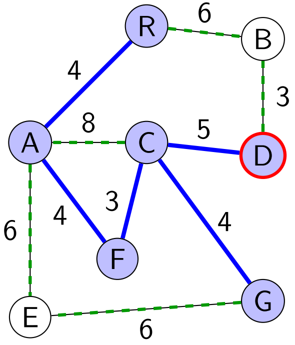
- Upprepa:
  - Markera D som stängd.
  - För var och en av de icke-stängda grannarna {B} till D:
    - Lägg (D,B,3) till q.



q={ (D,B,3), (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

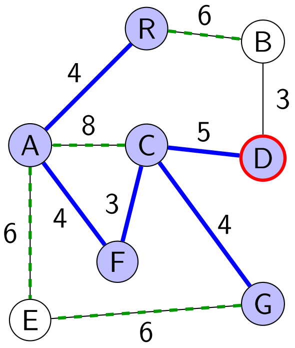
- Upprepa:
  - Markera D som stängd.
  - För var och en av de icke-stängda grannarna {B} till D:
  - Upprepa:



q={ (D,B,3), (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

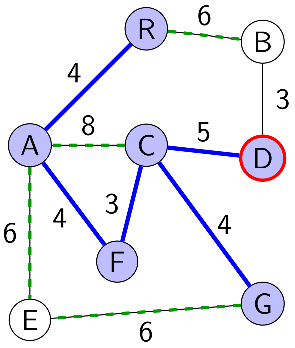
- Upprepa:
  - Markera D som stängd.
  - För var och en av de icke-stängda grannarna {B} till D:
  - Upprepa:
    - Ta (n, w, d)=(D,B,3) från q.



q={ (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

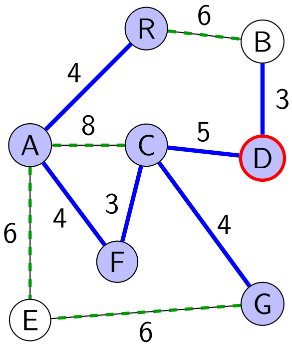
- Upprepa:
  - Markera D som stängd.
  - För var och en av de icke-stängda grannarna {B} till D:
  - Upprepa:
    - Ta (n, w, d)=(D,B,3) från q.
    - B ej stängd.



q={ (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

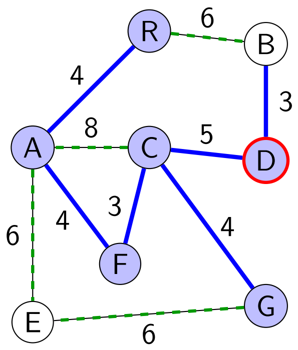
- Upprepa:
  - Markera D som stängd.
  - För var och en av de icke-stängda grannarna {B} till D:
  - Upprepa:
    - Ta (n, w, d)=(D,B,3) från q.
    - B ej stängd.
      - Lägg (D,B,3) till trädet.



q={ (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

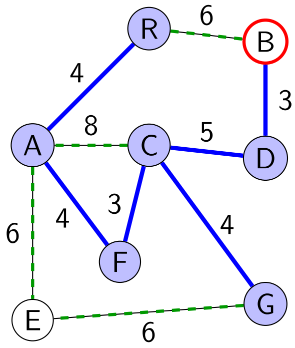
- Upprepa:
  - Markera D som stängd.
  - För var och en av de icke-stängda grannarna {B} till D:
  - Upprepa:
    - Ta (n, w, d)=(D,B,3) från q.
    - B ej stängd.
      - Lägg (D,B,3) till trädet.
  - tills B ej stängd eller q är tom.



q={ (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

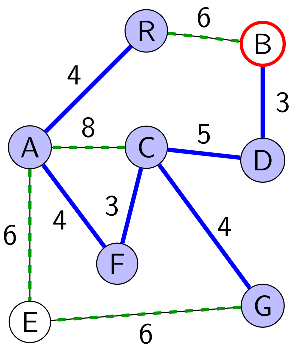
- Upprepa:
  - Markera D som stängd.
  - För var och en av de icke-stängda grannarna {B} till D:
  - Upprepa:
    - Ta (n, w, d)=(D,B,3) från q.
    - B ej stängd.
      - Lägg (D,B,3) till trädet.
  - tills B ej stängd eller q är tom.
  - $n \leftarrow$ B.



q={ (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

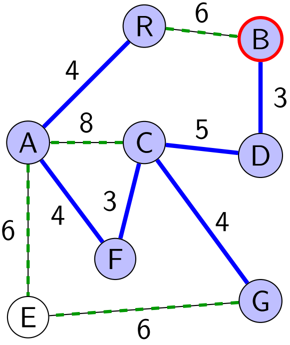
- Upprepa:
  - Markera D som stängd.
  - För var och en av de icke-stängda grannarna {B} till D:
  - Upprepa:
    - Ta (n, w, d)=(D,B,3) från q.
    - B ej stängd.
      - Lägg (D,B,3) till trädet.
  - tills B ej stängd eller q är tom.
  - $n \leftarrow$ B.
- tills q är tom.



q={ (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

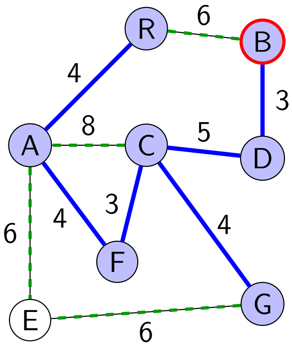
- Upprepa:
  - Markera B som stängd.



q={ (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

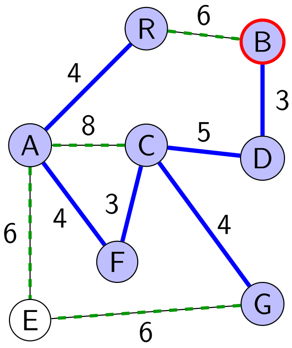
- Upprepa:
  - Markera B som stängd.
  - För var och en av de icke-stängda grannarna { } till B:



q={ (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

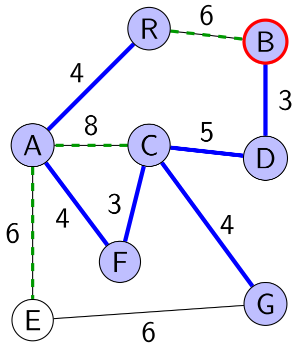
- Upprepa:
  - Markera B som stängd.
  - För var och en av de icke-stängda grannarna { } till B:
  - Upprepa:



q={ (G,E,6), (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

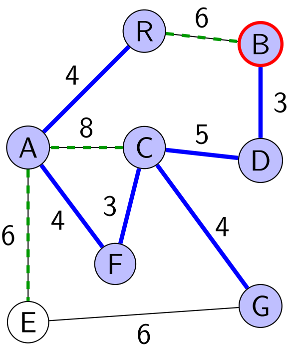
- Upprepa:
  - Markera B som stängd.
  - För var och en av de icke-stängda grannarna { } till B:
  - Upprepa:
    - Ta (n, w, d)=(G,E,6) från q.



q={ (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

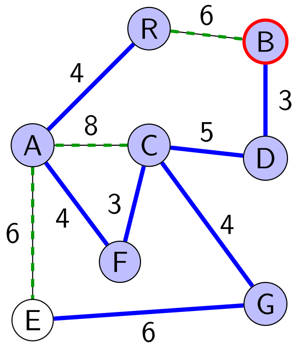
- Upprepa:
  - Markera B som stängd.
  - För var och en av de icke-stängda grannarna { } till B:
  - Upprepa:
    - Ta (n, w, d)=(G,E,6) från q.
    - E ej stängd.



q={ (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

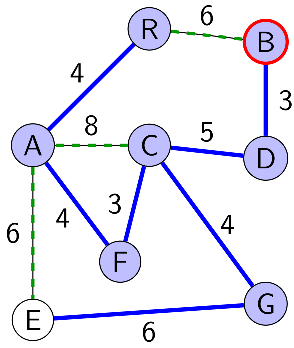
- Upprepa:
  - Markera B som stängd.
  - För var och en av de icke-stängda grannarna { } till B:
  - Upprepa:
    - Ta (n, w, d)=(G,E,6) från q.
    - E ej stängd.
      - Lägg (G,E,6) till trädet.



q={ (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

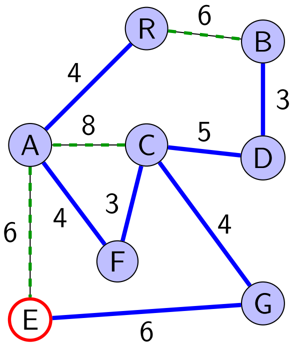
- Upprepa:
  - Markera B som stängd.
  - För var och en av de icke-stängda grannarna { } till B:
  - Upprepa:
    - Ta (n, w, d)=(G,E,6) från q.
    - E ej stängd.
      - Lägg (G,E,6) till trädet.
  - tills E ej stängd eller q är tom.



q={ (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

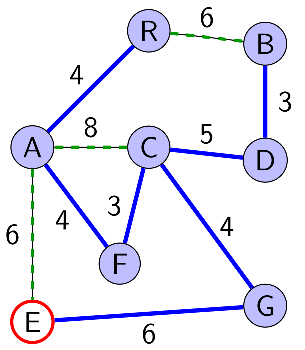
- Upprepa:
  - Markera B som stängd.
  - För var och en av de icke-stängda grannarna { } till B:
  - Upprepa:
    - Ta (n, w, d)=(G,E,6) från q.
    - E ej stängd.
      - Lägg (G,E,6) till trädet.
  - tills E ej stängd eller q är tom.
  - $n \leftarrow$ E.



q={ (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

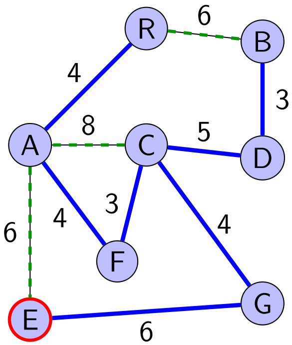
- Upprepa:
  - Markera B som stängd.
  - För var och en av de icke-stängda grannarna { } till B:
  - Upprepa:
    - Ta (n, w, d)=(G,E,6) från q.
    - E ej stängd.
      - Lägg (G,E,6) till trädet.
  - tills E ej stängd eller q är tom.
  - $n \leftarrow$ E.
- tills q är tom.



q={ (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

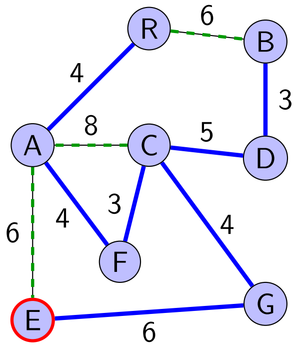
- Upprepa:
  - Markera E som stängd.



q={ (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

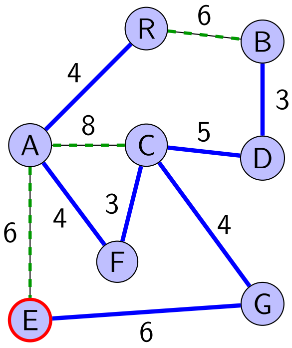
- Upprepa:
  - Markera E som stängd.
  - För var och en av de icke-stängda grannarna { } till E:



q={ (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

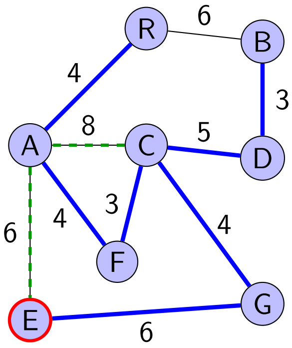
- Upprepa:
  - Markera E som stängd.
  - För var och en av de icke-stängda grannarna { } till E:
  - Upprepa:



q={ (R,B,6), (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

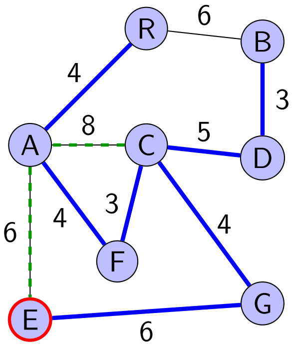
- Upprepa:
  - Markera E som stängd.
  - För var och en av de icke-stängda grannarna { } till E:
  - Upprepa:
    - Ta (n, w, d)=(R,B,6) från q.



q={ (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

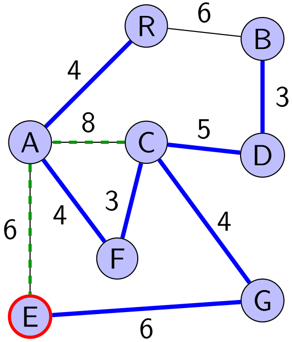
- Upprepa:
  - Markera E som stängd.
  - För var och en av de icke-stängda grannarna { } till E:
  - Upprepa:
    - Ta (n, w, d)=(R,B,6) från q.
    - B stängd.



q={ (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

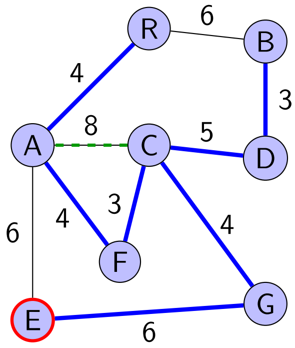
- Upprepa:
  - Markera E som stängd.
  - För var och en av de icke-stängda grannarna { } till E:
  - Upprepa:
    - Ta (n, w, d)=(R,B,6) från q.
    - B stängd.
  - tills B ej stängd eller q är tom.



q={ (A,E,6), (C,A,8) }

Prims minsta uppspännande träd, exempel

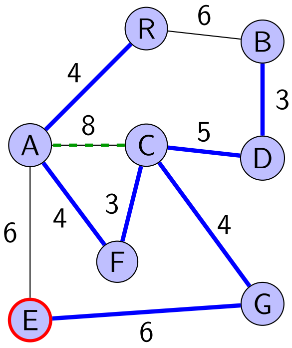
- Upprepa:
  - Markera E som stängd.
  - För var och en av de icke-stängda grannarna { } till E:
  - Upprepa:
    - Ta (n, w, d)=(A,E,6) från q.



q={ (C,A,8) }

Prims minsta uppspännande träd, exempel

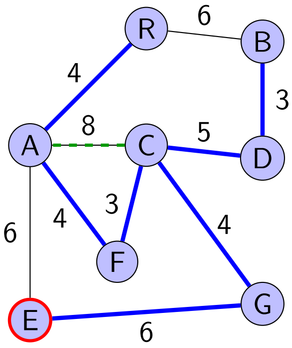
- Upprepa:
  - Markera E som stängd.
  - För var och en av de icke-stängda grannarna { } till E:
  - Upprepa:
    - Ta (n, w, d)=(A,E,6) från q.
    - E stängd.



q={ (C,A,8) }

Prims minsta uppspännande träd, exempel

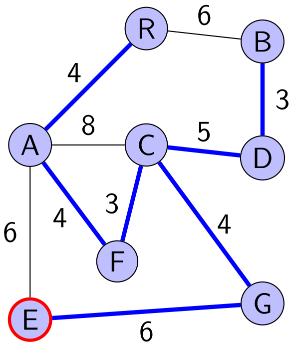
- Upprepa:
  - Markera E som stängd.
  - För var och en av de icke-stängda grannarna { } till E:
  - Upprepa:
    - Ta (n, w, d)=(A,E,6) från q.
    - E stängd.
  - tills E ej stängd eller q är tom.



q={ (C,A,8) }

Prims minsta uppspännande träd, exempel

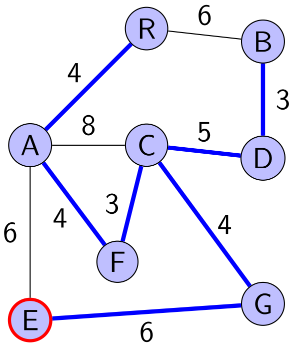
- Upprepa:
  - Markera E som stängd.
  - För var och en av de icke-stängda grannarna { } till E:
  - Upprepa:
    - Ta (n, w, d)=(C,A,8) från q.



q={ }

Prims minsta uppspännande träd, exempel

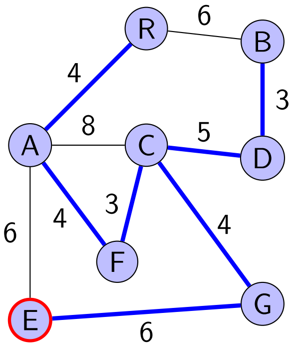
- Upprepa:
  - Markera E som stängd.
  - För var och en av de icke-stängda grannarna { } till E:
  - Upprepa:
    - Ta (n, w, d)=(C,A,8) från q.
    - A stängd.



q={ }

Prims minsta uppspännande träd, exempel

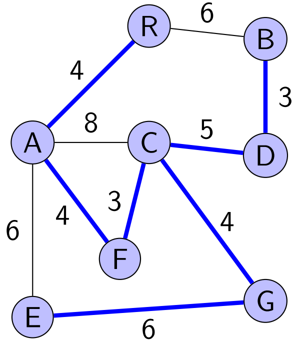
- Upprepa:
  - Markera E som stängd.
  - För var och en av de icke-stängda grannarna { } till E:
  - Upprepa:
    - Ta (n, w, d)=(C,A,8) från q.
    - A stängd.
  - tills A ej stängd eller q är tom.



q={ }

Prims minsta uppspännande träd, exempel

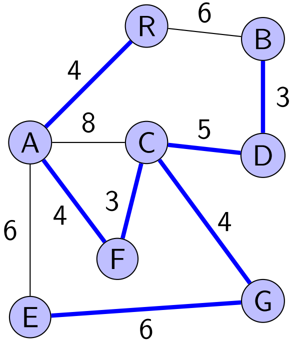
- Upprepa:
  - Markera E som stängd.
  - För var och en av de icke-stängda grannarna { } till E:
  - Upprepa:
  - tills A ej stängd eller q är tom.
  - $n \leftarrow$ A.



q={ }

Prims minsta uppspännande träd, exempel

- Upprepa:
  - Markera E som stängd.
  - För var och en av de icke-stängda grannarna { } till E:
  - Upprepa:
  - tills A ej stängd eller q är tom.
  - $n \leftarrow$ A.
- tills q är tom.



q={ }

Prims algoritm för minsta uppspännande träd

- Välj godtycklig startnod n ur grafen.
- Låt n bli rot i trädet.
- Skapa en tom prioritetskö q.
- Upprepa:
  - Markera n som stängd.
  - För var och en av de icke-stängda grannarna w till n:
    - Lägg bågen (n, w, d) i prioritetskön q.
  - Upprepa:
    - Ta första bågen (n, w, d) ur q.
    - Om destinationsnoden w ej är stängd:
      - Lägg till bågen (n, w, d) till trädet.

    tills w ej stängd eller q är tom.
  - Låt $n = w$.

  tills q är tom.

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 kön.
    - 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 sammanhängade graf?

Kruskals algoritm för minsta uppspännande träd

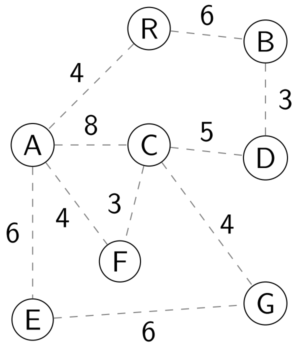
- Utgå från en prioritetskö av alla bågar.
- I varje steg, plocka kortaste bågen från kön.
  - Fyra alternativ:
    - Bygg nytt träd.
    - Bygg ut ett träd.
    - Slå ihop två träd.
    - Ignorera bågen.
- Under algoritmens gång kan vi ha en skog.
- Till slut har vi bara ett träd.
- Vår beskrivning 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. Sortera 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 uppspännande träd, exempel

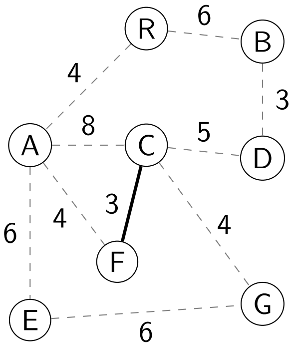
- Upprepa tills kön är tom:



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

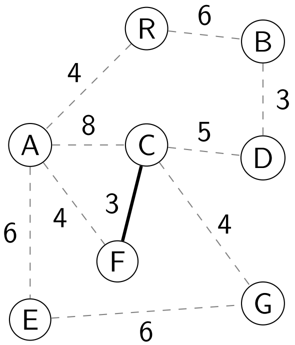
- Upprepa tills kön är tom:
  - Ta första bågen (C,F,3) ur kön.



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 uppspännande träd, exempel

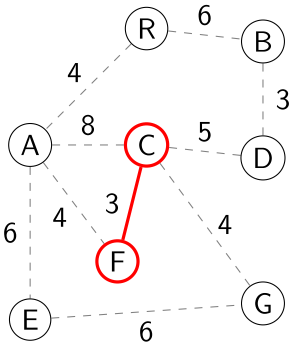
- Upprepa tills kön är tom:
  - Ta första bågen (C,F,3) ur kön.
  - Ingen av (C,F) är färgade:



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 uppspännande träd, exempel

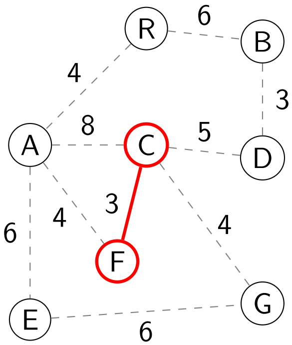
- Upprepa tills kön är tom:
  - Ta första bågen (C,F,3) ur kön.
  - Ingen av (C,F) är färgade:
    - Färglägg med ny färg.



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 uppspännande träd, exempel

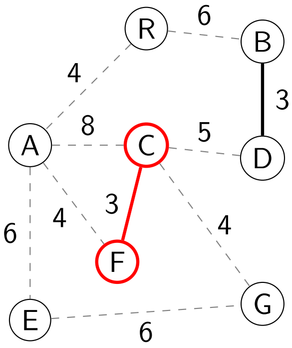
- Upprepa tills kön är tom:



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 uppspännande träd, exempel

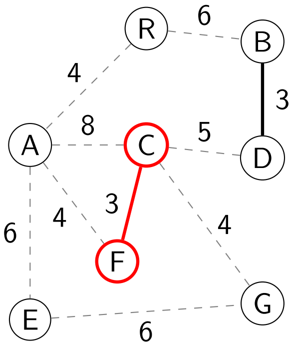
- Upprepa tills kön är tom:
  - Ta första bågen (B,D,3) ur kön.



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

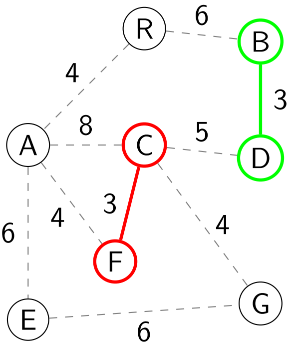
- Upprepa tills kön är tom:
  - Ta första bågen (B,D,3) ur kön.
  - Ingen av (B,D) är färgade:



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 uppspännande träd, exempel

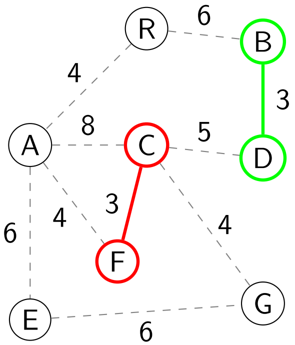
- Upprepa tills kön är tom:
  - Ta första bågen (B,D,3) ur kön.
  - Ingen av (B,D) är färgade:
    - Färglägg med ny färg.



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

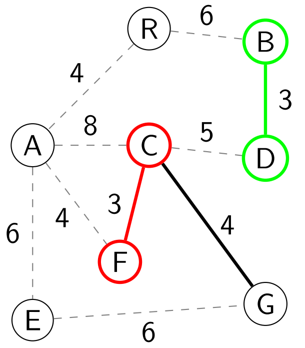
- Upprepa tills kön är tom:



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

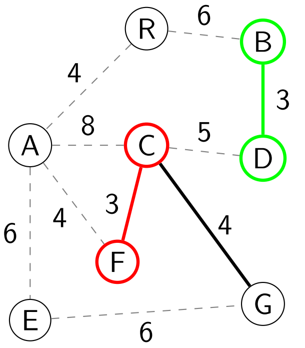
- Upprepa tills kön är tom:
  - Ta första bågen (C,G,4) ur kön.



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 uppspännande träd, exempel

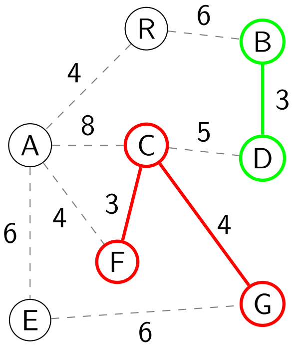
- Upprepa tills kön är tom:
  - Ta första bågen (C,G,4) ur kön.
  - C är färgad.



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 uppspännande träd, exempel

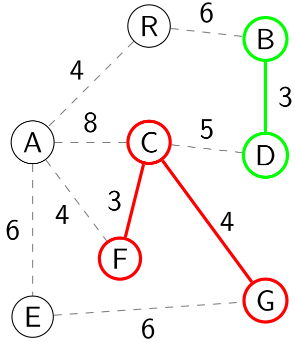
- Upprepa tills kön är tom:
  - Ta första bågen (C,G,4) ur kön.
  - C är färgad.
    - Färglägg med C:s färg.



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 uppspännande träd, exempel

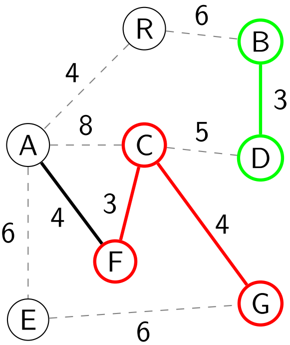
- Upprepa tills kön är tom:



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 uppspännande träd, exempel

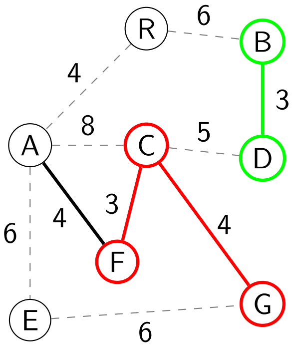
- Upprepa tills kön är tom:
  - Ta första bågen (A,F,4) ur kön.



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

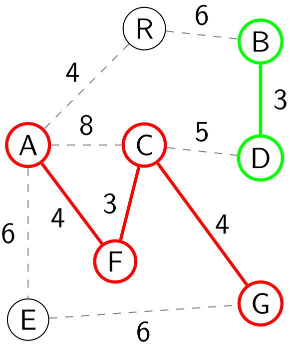
- Upprepa tills kön är tom:
  - Ta första bågen (A,F,4) ur kön.
  - F är färgad.



q={ (A,R,4), (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

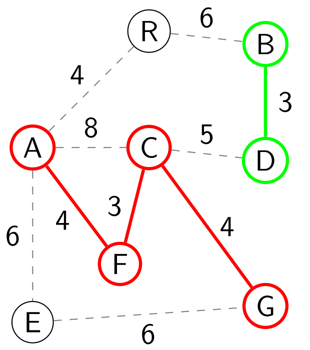
- Upprepa tills kön är tom:
  - Ta första bågen (A,F,4) ur kön.
  - F är färgad.
    - Färglägg med F:s färg.



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

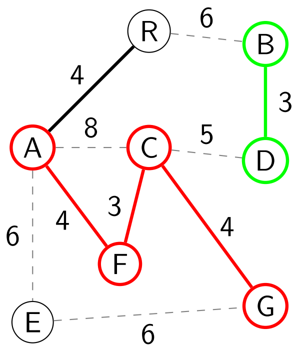
- Upprepa tills kön är tom:



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

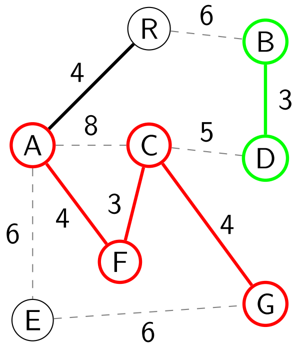
- Upprepa tills kön är tom:
  - Ta första bågen (A,R,4) ur kön.



q={ (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

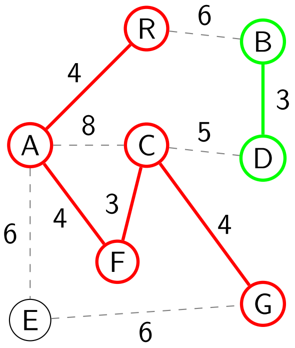
- Upprepa tills kön är tom:
  - Ta första bågen (A,R,4) ur kön.
  - A är färgad.



q={ (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

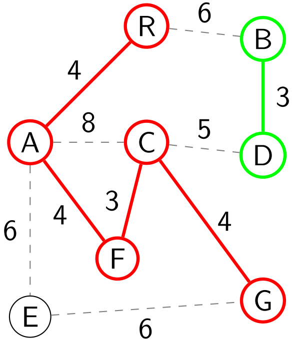
- Upprepa tills kön är tom:
  - Ta första bågen (A,R,4) ur kön.
  - A är färgad.
    - Färglägg med A:s färg.



q={ (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

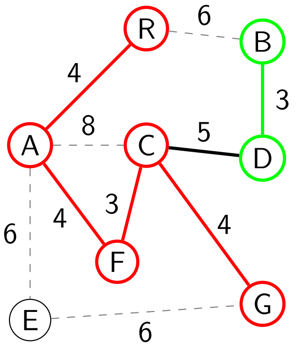
- Upprepa tills kön är tom:



q={ (C,D,5), (E,G,6), (B,R,6), (A,E,6), (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

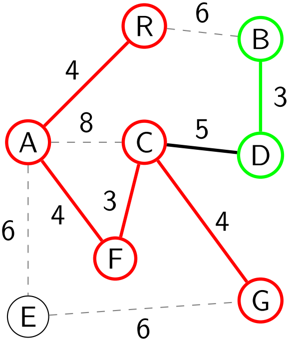
- Upprepa tills kön är tom:
  - Ta första bågen (C,D,5) ur kön.



q={ (E,G,6), (B,R,6), (A,E,6), (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

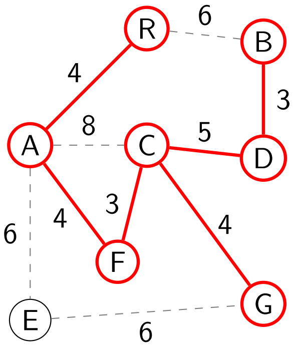
- Upprepa tills kön är tom:
  - Ta första bågen (C,D,5) ur kön.
  - C och D färgade med olika färg.



q={ (E,G,6), (B,R,6), (A,E,6), (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

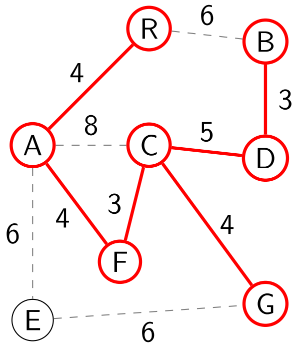
- Upprepa tills kön är tom:
  - Ta första bågen (C,D,5) ur kön.
  - C och D färgade med olika färg.
    - Färglägg bägge graferna med C:s färg.



q={ (E,G,6), (B,R,6), (A,E,6), (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

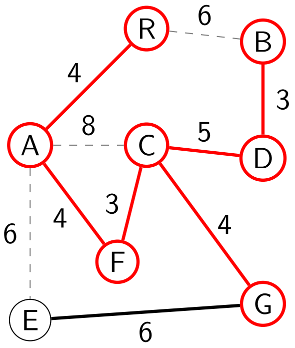
- Upprepa tills kön är tom:



q={ (E,G,6), (B,R,6), (A,E,6), (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

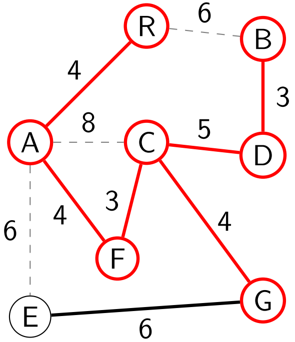
- Upprepa tills kön är tom:
  - Ta första bågen (E,G,6) ur kön.



q={ (B,R,6), (A,E,6), (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

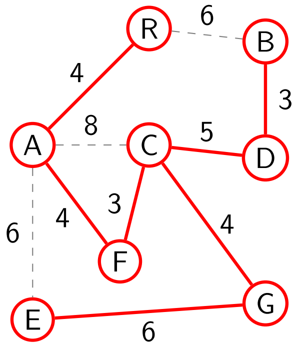
- Upprepa tills kön är tom:
  - Ta första bågen (E,G,6) ur kön.
  - G är färgad.



q={ (B,R,6), (A,E,6), (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

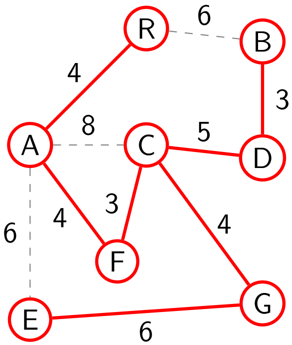
- Upprepa tills kön är tom:
  - Ta första bågen (E,G,6) ur kön.
  - G är färgad.
    - Färglägg med G:s färg.



q={ (B,R,6), (A,E,6), (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

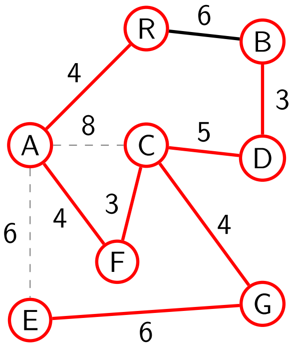
- Upprepa tills kön är tom:



q={ (B,R,6), (A,E,6), (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

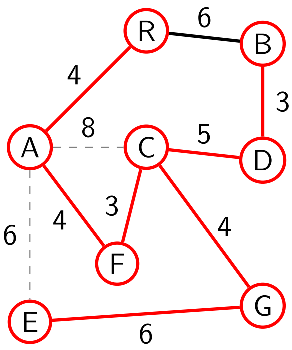
- Upprepa tills kön är tom:
  - Ta första bågen (B,R,6) ur kön.



q={ (A,E,6), (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

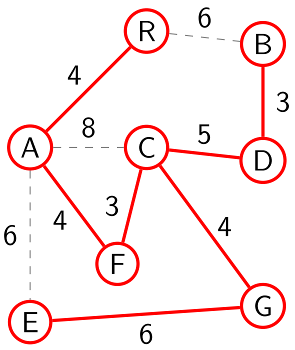
- Upprepa tills kön är tom:
  - Ta första bågen (B,R,6) ur kön.
  - Bägge färgade med samma färg.



q={ (A,E,6), (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

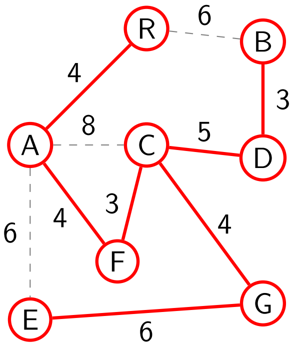
- Upprepa tills kön är tom:
  - Ta första bågen (B,R,6) ur kön.
  - Bägge färgade med samma färg.
    - Ignorera bågen.



q={ (A,E,6), (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

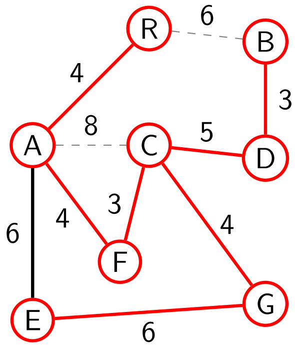
- Upprepa tills kön är tom:



q={ (A,E,6), (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

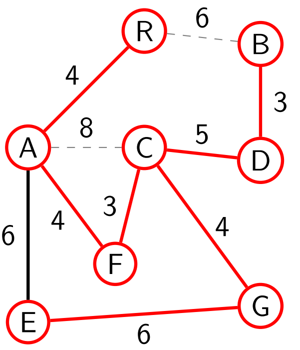
- Upprepa tills kön är tom:
  - Ta första bågen (A,E,6) ur kön.



q={ (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

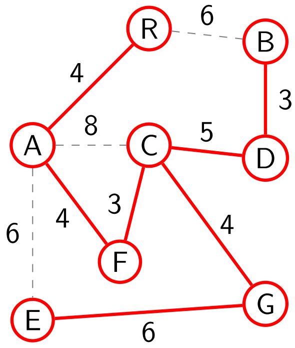
- Upprepa tills kön är tom:
  - Ta första bågen (A,E,6) ur kön.
  - Bägge färgade med samma färg.



q={ (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

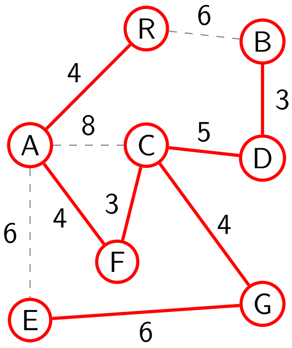
- Upprepa tills kön är tom:
  - Ta första bågen (A,E,6) ur kön.
  - Bägge färgade med samma färg.
    - Ignorera bågen.



q={ (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

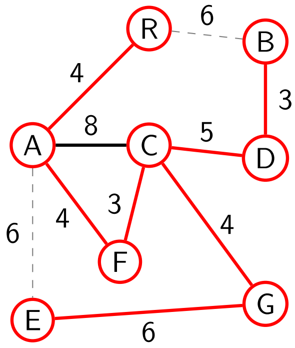
- Upprepa tills kön är tom:



q={ (A,C,8) }

Kruskals algoritm för minsta uppspännande träd, exempel

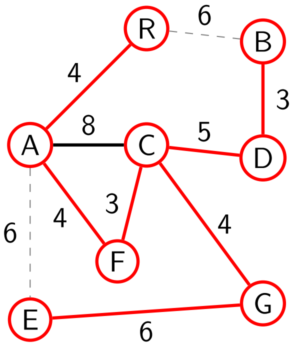
- Upprepa tills kön är tom:
  - Ta första bågen (A,C,8) ur kön.



q={ }

Kruskals algoritm för minsta uppspännande träd, exempel

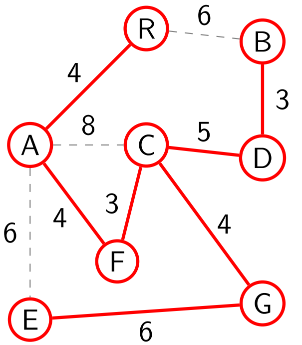
- Upprepa tills kön är tom:
  - Ta första bågen (A,C,8) ur kön.
  - Bägge färgade med samma färg.



q={ }

Kruskals algoritm för minsta uppspännande träd, exempel

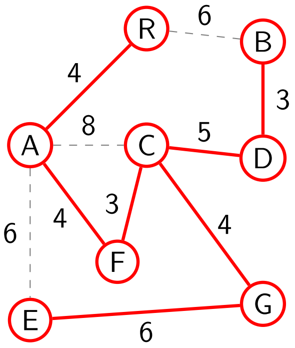
- Upprepa tills kön är tom:
  - Ta första bågen (A,C,8) ur kön.
  - Bägge färgade med samma färg.
    - Ignorera bågen.



$q = \{\ \}$

Kruskals algoritm för minsta uppspännande träd, exempel

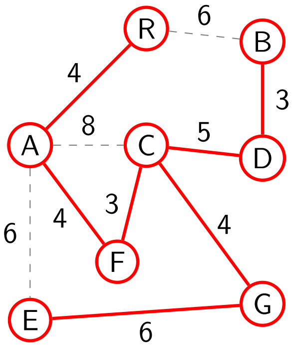
- Upprepa tills kön är tom:



q={ }

Kruskals algoritm för minsta uppspännande 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 algoritm för minsta uppspännande träd, naiv

```
Algorithm Kruskal(Graph g)
nextColor ← 1; Pqueue q = empty();
for each node n in g
  n.color ← 0;
for each edge e in g
  q ← insert(q,e);
while not isempty(q) do
  e = (a,b) ← inspect-first(q); q ← delete-first(q);
  if a.color = b.color then
    if a.color = 0 then
      a.color ← nextColor
      b.color ← nextColor
      nextColor ← nextColor+1
    else
      // same color!=0, do nothing
  else // different color
    if a.color = 0 then // b colored
      b.color ← a.color
    elseif b.color = 0 then // a colored
      a.color ← b.color
    else // colored with different colors
      for each node n in g
        if n.color = b.color then
          n.color ← 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 algoritm för minsta uppspännande träd, effektiv

```
Algorithm Kruskal(Graph g)
nextColor ← 1; Pqueue q = empty(); E(0)=0;
for each node n in g
  n.color ← 0;
for each edge e in g
  q ← insert(q,e);
while not isempty(q) do
  e = (a,b) ← inspect-first(q); q ← delete-first(q);
  if E(a.color) = E(b.color) then
    if a.color = 0 then
      a.color ← nextColor
      b.color ← nextColor
      E(nextColor) ← nextColor
      nextColor ← nextColor+1
    else
      // same color!=0, do nothing
  else // different color
    if a.color = 0 then // b colored
      b.color ← a.color
    elseif b.color = 0 then // a colored
      a.color ← b.color
    else // colored with different colors
      E(a.color) ← min(E(a.color), E(b.color))
      E(b.color) ← min(E(a.color), E(b.color))
```

Fråga

- Hur fungerar Kruskals algoritm på en icke sammanhängade graf?