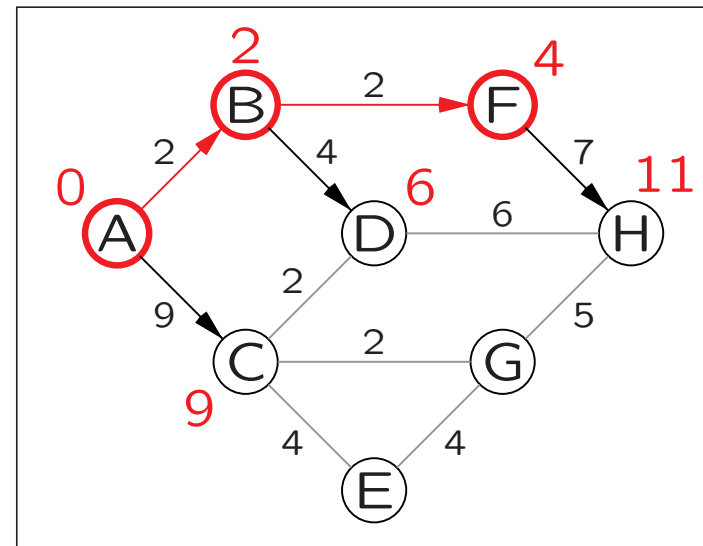
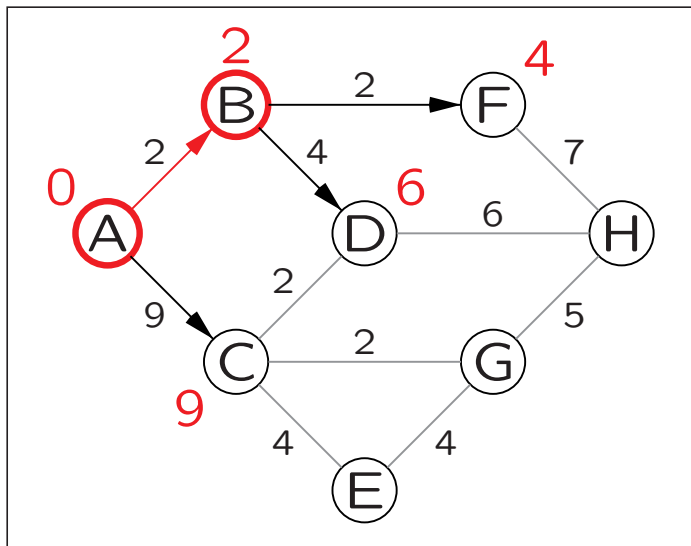
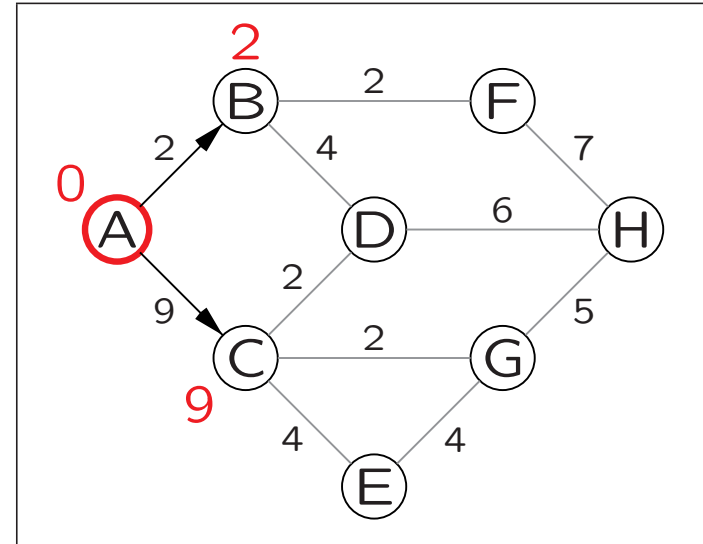
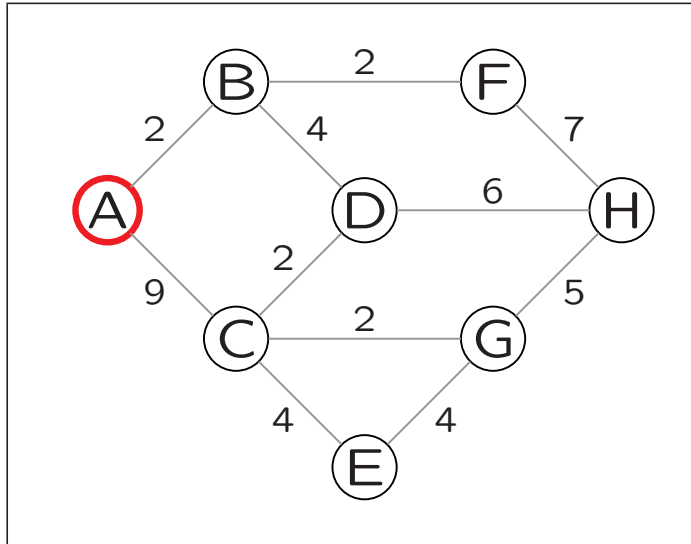
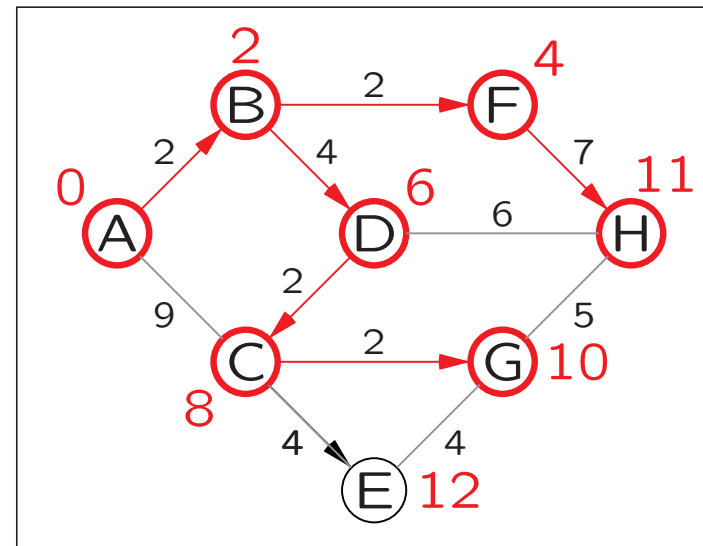
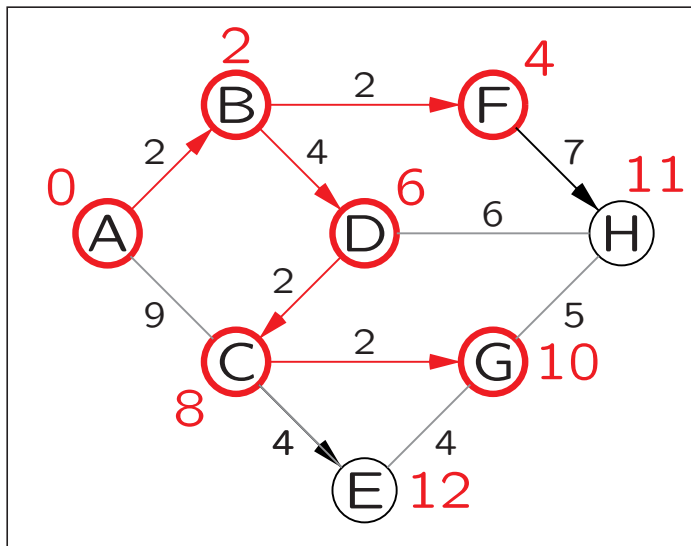
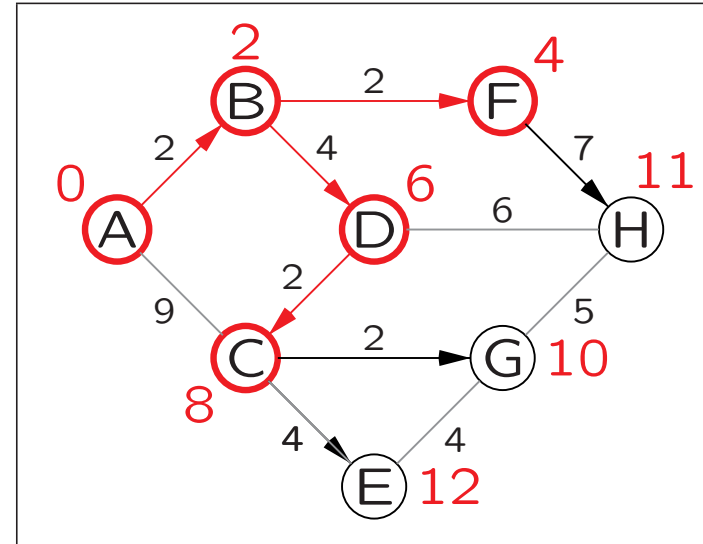
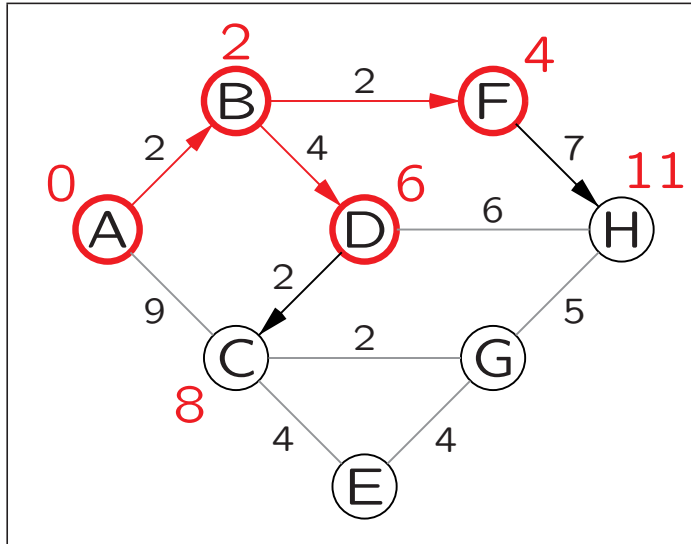


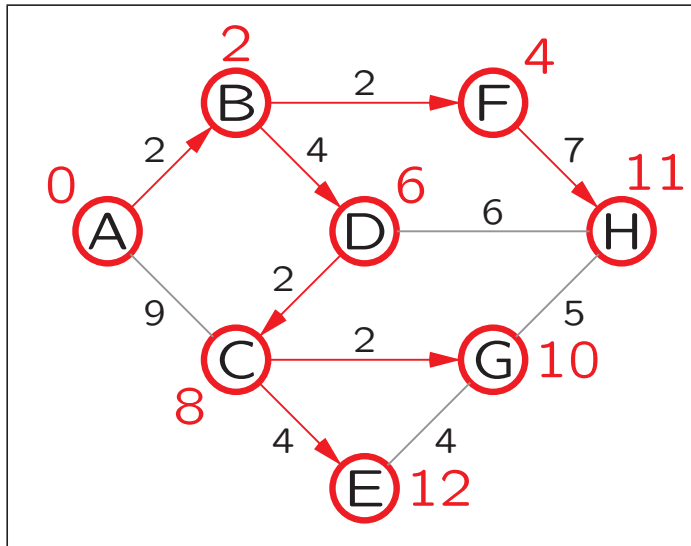
Elfde college algoritmiek

21 april 2021

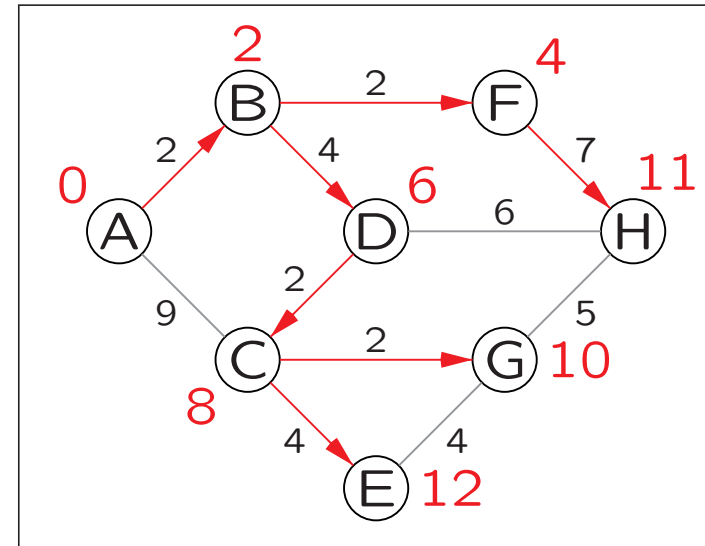
Algoritme van Dijkstra,
Gretige Algoritmen







Het algoritme is klaar:
alle knopen gehad



Alle kortste paden vanuit
A met hun lengtes

```
// invoer: samenhangende gewogen graaf  $G = (V, E)$  en startknoop  $s$   
// uitvoer: array dat de lengtes van de kortste paden vanuit  $s$  bevat;  
// na afloop is  $\text{pad}[v] =$  de lengte van een kortste pad van  $s$  naar  $v$ 
```

```
for  $v \in V$  do  
     $\text{pad}[v] := \infty$ ;  
od  
 $\text{pad}[s] := 0$ ;  
 $U := \emptyset$ ;  
  
while ( $U \neq V$ ) do  
    vind knoop  $v^* \in V \setminus U$  met  $\text{pad}[v^*]$  minimaal;  
     $U := U \cup \{v^*\}$ ;  
    for alle knopen  $v$  aangrenzend aan  $v^*$  do  
        if  $\text{pad}[v^*] + \text{gewicht}(v^*, v) < \text{pad}[v]$  then  
             $\text{pad}[v] := \text{pad}[v^*] + \text{gewicht}(v^*, v)$ ;  
        fi  
    od  
od
```

Complexiteit (met adjacency matrix): ...

```
// invoer: samenhangende gewogen graaf  $G = (V, E)$  en startknoop  $s$   
// uitvoer: array dat de lengtes van de kortste paden vanuit  $s$  bevat;  
// na afloop is  $\text{pad}[v] =$  de lengte van een kortste pad van  $s$  naar  $v$ 
```

```
for  $v \in V$  do  
     $\text{pad}[v] := \infty$ ;  
od  
 $\text{pad}[s] := 0$ ;  
 $U := \emptyset$ ;  
  
while (  $U \neq V$  ) do  
    vind knoop  $v^* \in V \setminus U$  met  $\text{pad}[v^*]$  minimaal;  
     $U := U \cup \{v^*\}$ ;  
    for alle knopen  $v$  aangrenzend aan  $v^*$  do  
        if  $\text{pad}[v^*] + \text{gewicht}(v^*, v) < \text{pad}[v]$  then  
             $\text{pad}[v] := \text{pad}[v^*] + \text{gewicht}(v^*, v)$ ;  
        fi  
    od  
od
```

Complexiteit (met adjacency matrix):
 $\Theta(n + 1 + n(n + 1 + n)) = \Theta(n^2)$

```
// invoer: samenhangende gewogen graaf  $G = (V, E)$  en startknoop  $s$   
// uitvoer: array dat de lengtes van de kortste paden vanuit  $s$  bevat;  
// na afloop is  $\text{pad}[v] =$  de lengte van een kortste pad van  $s$  naar  $v$ 
```

```
for  $v \in V$  do  
     $\text{pad}[v] := \infty$ ;  
od  
 $\text{pad}[s] := 0$ ;  
 $U := \emptyset$ ;  
  
while ( $U \neq V$ ) do  
    vind knoop  $v^* \in V \setminus U$  met  $\text{pad}[v^*]$  minimaal;  
     $U := U \cup \{v^*\}$ ;  
    for alle knopen  $v$  aangrenzend aan  $v^*$  do  
        if  $\text{pad}[v^*] + \text{gewicht}(v^*, v) < \text{pad}[v]$  then  
             $\text{pad}[v] := \text{pad}[v^*] + \text{gewicht}(v^*, v)$ ;  
        fi  
    od  
od
```

Complexiteit (met adjacency list en priority queue): ...

Bij Dijkstra:

kies knoop v^* buiten boom met laagste kandidaatwaarde $\text{pad}[v^*]$

Bij Branch & Bound (volgende week):

kies deeloplossing met beste ondergrens/bovengrens

Priority Queue:

- objecten met prioriteit (en info), b.v. $\{ (C,9), (D,6), (H,11) \}$
- insert
- findmax (findmin)
- deletemax (deletemin)
- (optioneel) changepriority

Bij Dijkstra: kies knoop buiten boom met laagste kandidaatwaarde

Bij Branch & Bound (volgende week):

kies deeloplossing met beste ondergrens/bovengrens

Priority Queue:

- objecten met prioriteit (en info), b.v. $\{(C, 9), (D, 6), (H, 11)\}$
- insert: heap: $O(\log n)$
- findmax (findmin): heap: $O(1)$
- deletemax (deletemin): heap: $O(\log n)$
- (optioneel) changepriority: heap: $O(\log n)$

```
// invoer: samenhangende gewogen graaf  $G = (V, E)$  en startknoop  $s$   
// uitvoer: array dat de lengtes van de kortste paden vanuit  $s$  bevat;  
// na afloop is  $\text{pad}[v] =$  de lengte van een kortste pad van  $s$  naar  $v$ 
```

```
for  $v \in V$  do  
     $\text{pad}[v] := \infty$ ;  
od  
 $\text{pad}[s] := 0$ ;  
 $U := \emptyset$ ;  
  
while ( $U \neq V$ ) do  
    vind knoop  $v^* \in V \setminus U$  met  $\text{pad}[v^*]$  minimaal;  
     $U := U \cup \{v^*\}$ ;  
    for alle knopen  $v$  aangrenzend aan  $v^*$  do  
        if  $\text{pad}[v^*] + \text{gewicht}(v^*, v) < \text{pad}[v]$  then  
             $\text{pad}[v] := \text{pad}[v^*] + \text{gewicht}(v^*, v)$ ;  
        fi  
    od  
od
```

Complexiteit (met adjacency list en heap): ...

```
// invoer: samenhangende gewogen graaf  $G = (V, E)$  en startknoop  $s$   
// uitvoer: array dat de lengtes van de kortste paden vanuit  $s$  bevat;  
// na afloop is  $\text{pad}[v] =$  de lengte van een kortste pad van  $s$  naar  $v$ 
```

```
for  $v \in V$  do  
     $\text{pad}[v] := \infty$ ;  
od  
 $\text{pad}[s] := 0$ ;  
 $U := \emptyset$ ;  
  
while (  $U \neq V$  ) do  
    vind knoop  $v^* \in V \setminus U$  met  $\text{pad}[v^*]$  minimaal;  
     $U := U \cup \{v^*\}$ ;  
    for alle knopen  $v$  aangrenzend aan  $v^*$  do  
        if  $\text{pad}[v^*] + \text{gewicht}(v^*, v) < \text{pad}[v]$  then  
             $\text{pad}[v] := \text{pad}[v^*] + \text{gewicht}(v^*, v)$ ;  
        fi  
    od  
od
```

Complexiteit (met adjacency list en heap):
 $O(n + 1 + n + n \log n + m \log n) = O(m \log n)$

- In het algoritme bevat U steeds alle knopen waarvan de definitieve kortste afstand vanaf s reeds bepaald is. Voor deze knopen geeft het label $\text{pad}[v]$ al de definitieve kortste afstand aan. **Moet bewezen worden.**
- Voor de andere knopen w geldt na elke ronde (= doorgang door de while):

$$(\#) \text{pad}[w] = \min_{u \in U} \{ \text{pad}[u] + \text{gewicht}(u, w) \}^*$$

Dit volgt direct uit het algoritme.

- De volgende dichtstbijzijnde knoop v^* wordt gekozen uit de knopen uit $V \setminus U$ die direct grenzen aan U . Nadat deze gekozen is worden de labels aangepast, zodat $(\#)$ ook geldt voor de nieuwe U .
- Het is niet zo moeilijk dit algoritme aan te passen zodat ook de kortste paden zelf worden berekend. Sla direct na het aanpassen van het label van knoop v de nieuwe kandidaattak (v^*, v) op:

```
if  $\text{pad}[v^*] + \text{gewicht}(v^*, v) < \text{pad}[v]$  then  
     $\text{pad}[v] := \text{pad}[v^*] + \text{gewicht}(v^*, v);$   
    nieuwe kandidaattak voor  $v$ :  $(v^*, v)$   
fi
```

*Dit betekent dat $\text{pad}[w]$ voor deze knopen $w \notin U$ de lengte van een kortste pad van s naar w aangeeft via uitsluitend knopen van U .

Na elke ronde (dus ook na de laatste, wanneer $U = V$) van het algoritme van Dijkstra geldt:

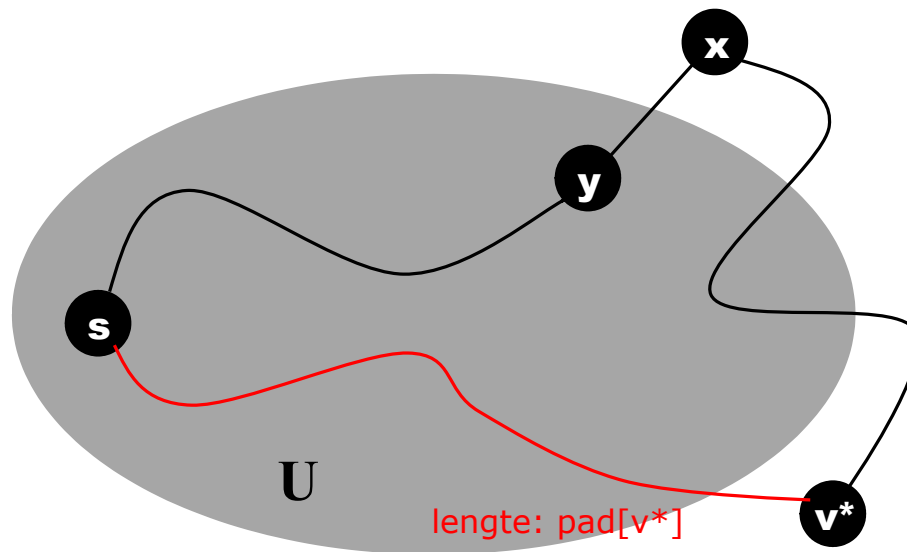
- U bevat alle knopen waarvan de definitieve kortste afstand vanaf s reeds bepaald is. Voor elke $v \in U$ geeft het label $\text{pad}[v]$ die kortste afstand aan.

Om dit te bewijzen moet je laten zien dat:

- wanneer v^* wordt toegevoegd aan U , $\text{pad}[v^*]$ inderdaad de lengte van het kortste pad van s naar v^* bevat

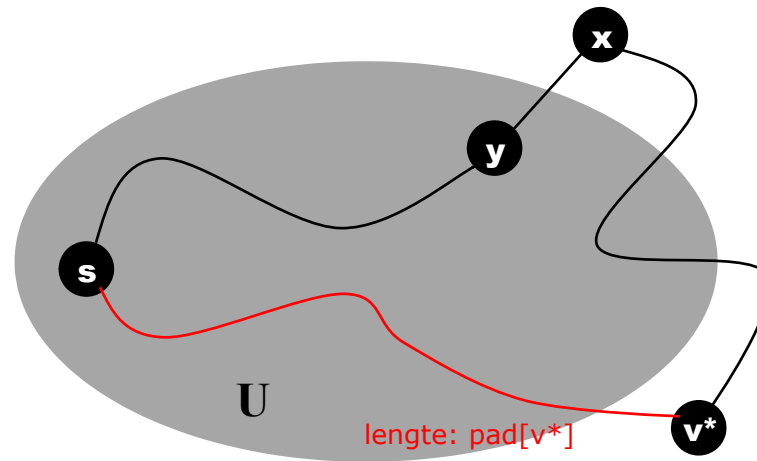
$\text{pad}[v^*]$ is daadwerkelijk lengte van een pad van s naar v^* :

$$(\#) \text{pad}[v^*] = \min_{u \in U} \{ \text{pad}[u] + \text{gewicht}(u, v^*) \}^*$$



Dit betekent dat $\text{pad}[v^]$ de lengte van een kortste pad van s naar v^* aangeeft via uitsluitend knopen van U .

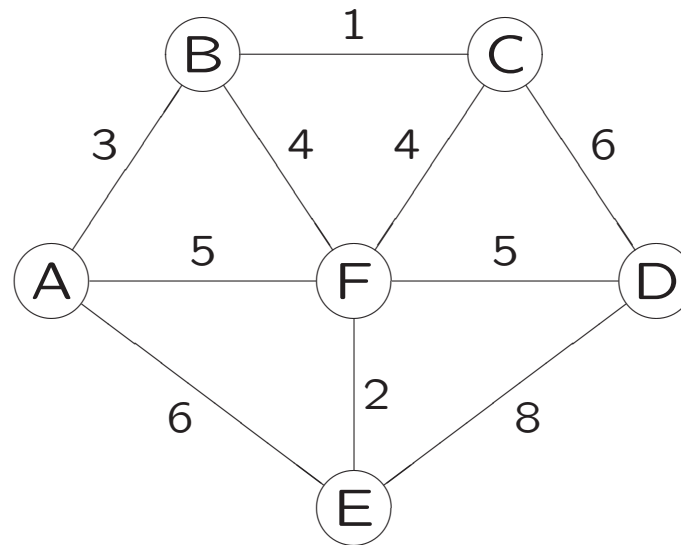
Bekijk, behalve het **pad ter lengte $\text{pad}[v^*]$** van s naar v^* via U een willekeurig ander pad van s naar v^* . Stel dat x de eerste knoop op dat pad is buiten U , en y de laatste knoop daarvóór. (Deze x kán gelijk zijn aan v^* .) Zij $d(s, y, x, v^*)$ de lengte van dat pad.

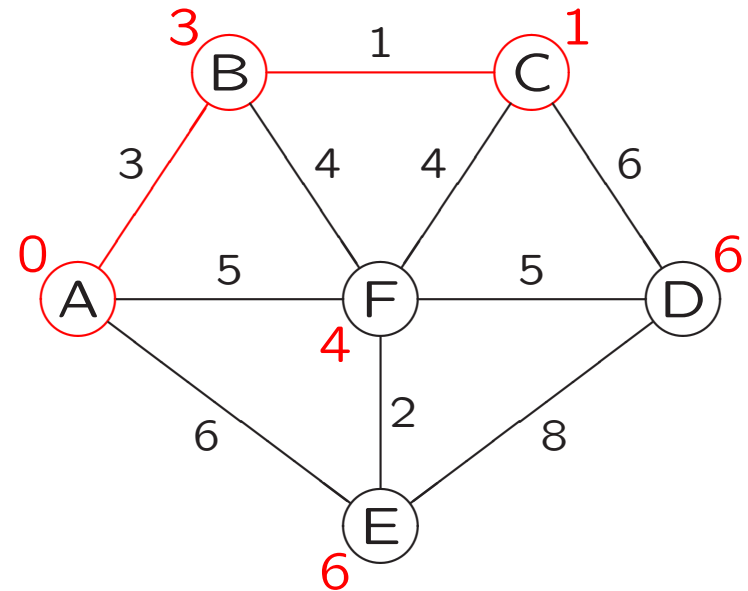
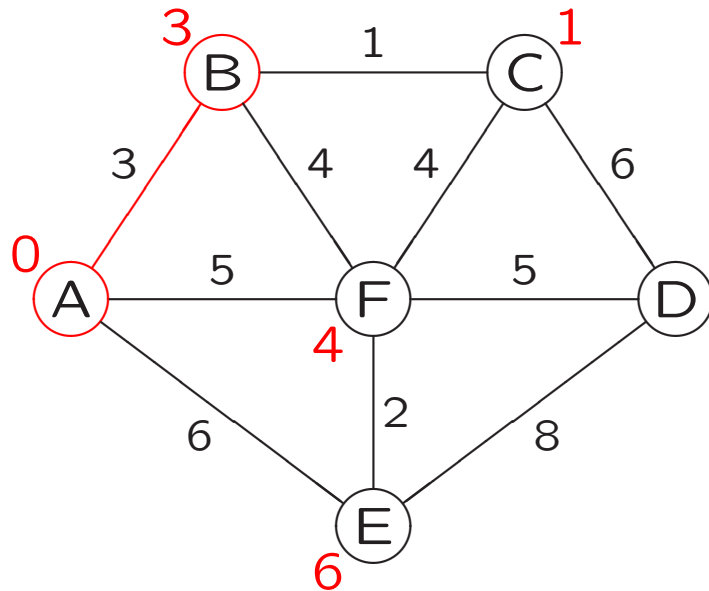
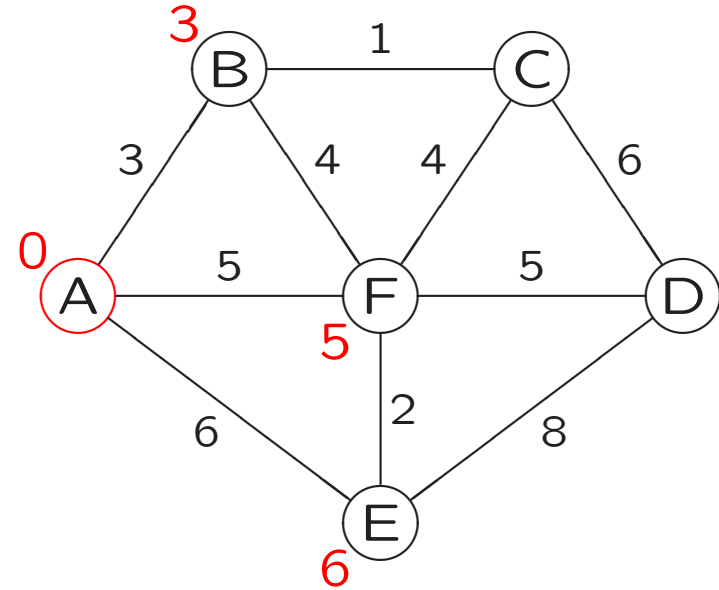
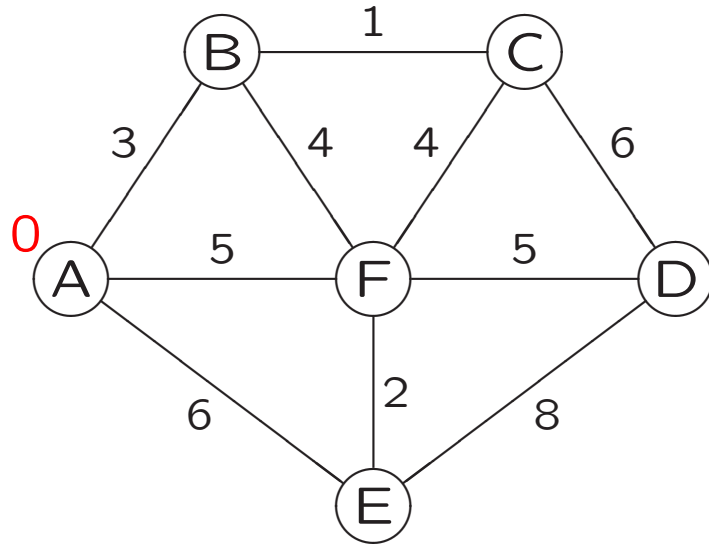


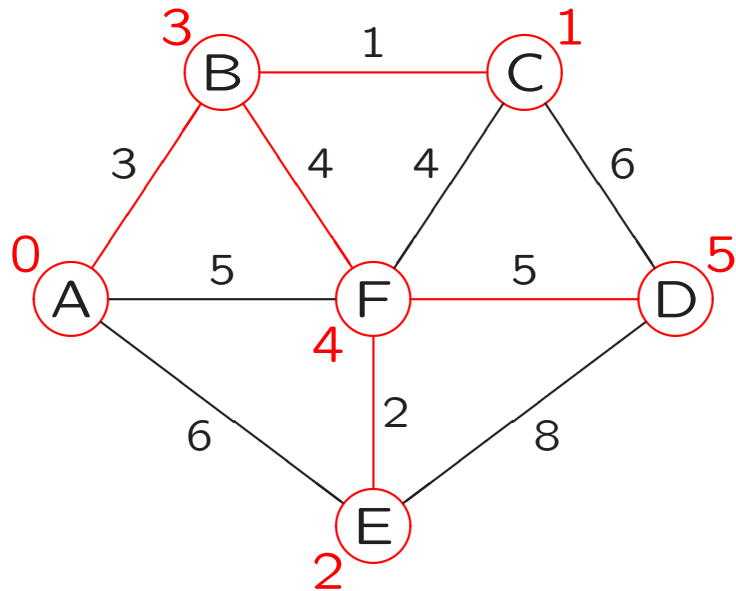
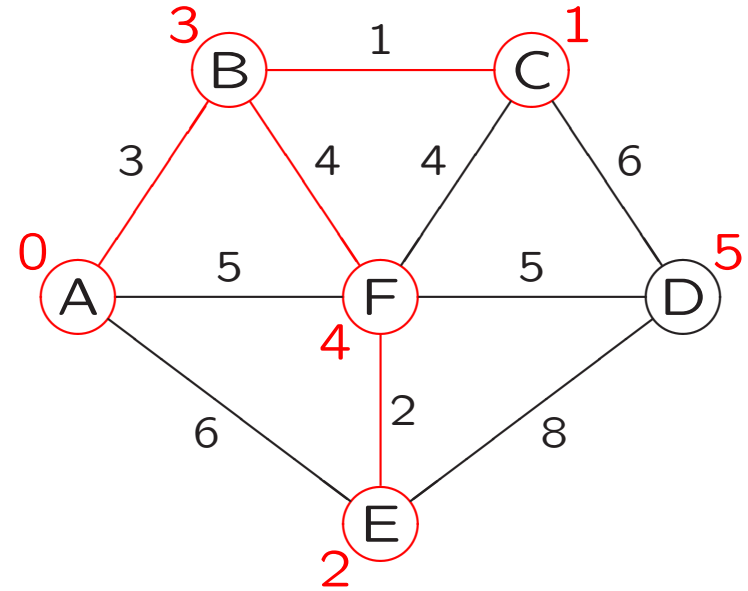
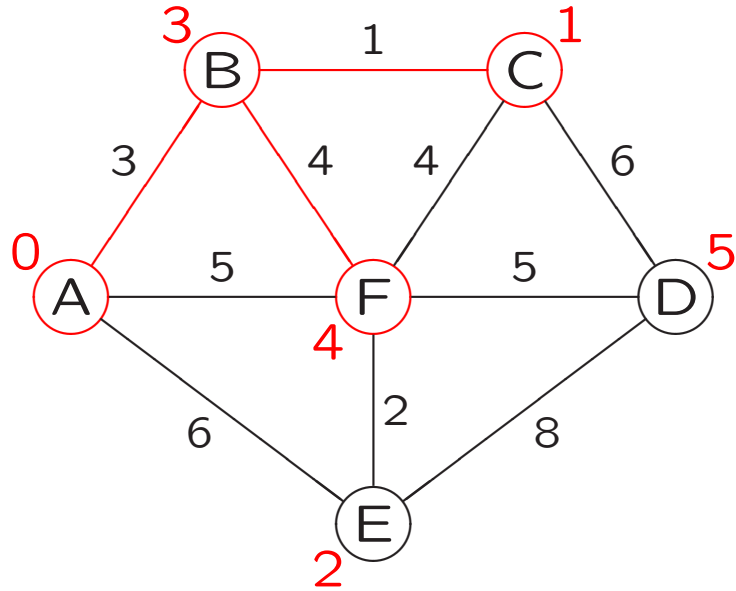
Dan geldt: $d(s, y, x, v^*) \geq d(s, y, x) = d(s, y) + \text{gewicht}(y, x) \geq \text{pad}[y] + \text{gewicht}(y, x) \geq \text{pad}[x] \geq \text{pad}[v^*]$ omdat respectievelijk alle gewichten van de takken ≥ 0 zijn, $\text{pad}[y]$ volgens inductiehypothese kortste afstand is naar y , ($\#$) geldt voor x , en v^* gekozen was in deze ronde als 'minimale' knoop.

Gegeven een **samenhangende, ongerichte** graaf G met gewichten op de takken.

Gevraagd: een **opspannende boom** van G met minimaal totaal gewicht.







// invoer: **samenhangende, ongerichte** gewogen graaf $G = (V, E)$ en startknoop s
// uitvoer: E_T , verzameling takken van minimale opspannende boom T van G

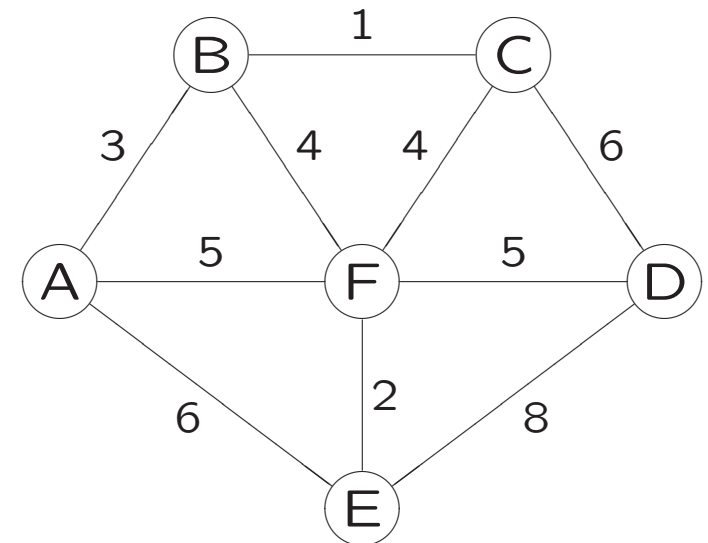
```
for  $v \in V$  do  
    tak[ $v$ ] :=  $\infty$ ;  
od  
tak[ $s$ ] := 0;  
 $U$  :=  $\emptyset$ ;  
 $E_T$  :=  $\emptyset$ ;  
  
while ( $U \neq V$ ) do  
    vind knoop  $v^* \in V \setminus U$  met tak[ $v^*$ ] minimaal;  
     $U$  :=  $U \cup \{v^*\}$ ;  
    voeg bijbehorende tak naar  $v^*$  toe aan  $E_T$  (als  $v^* \neq s$ );  
    for alle knopen  $v$  aangrenzend aan  $v^*$  do  
        if gewicht( $v^*, v$ ) < tak[ $v$ ] then  
            tak[ $v$ ] := gewicht( $v^*, v$ );  
        fi  
    od  
od
```

Complexiteit (met adjacency list en heap):

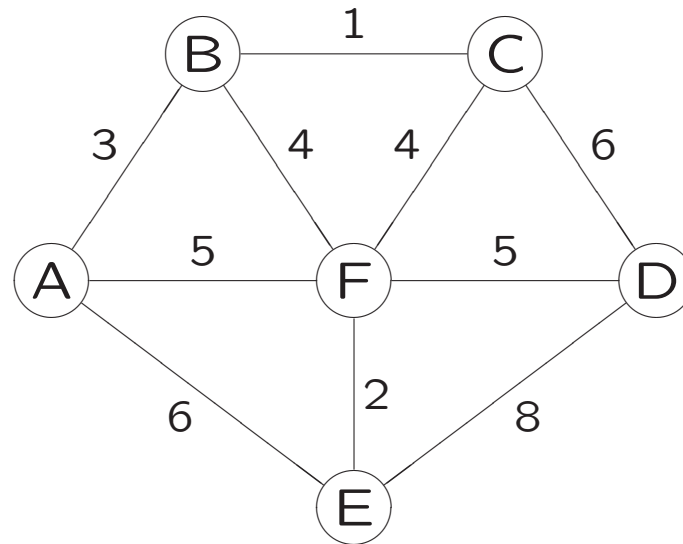
$O(n + 1 + n + n \log n + n + m \log n) = O(m \log n)$ (zie Dijkstra)

Als je gewoon wilt doortekenen in één plaatje...

A	B	C	D	E	F	Actie
0	∞	∞	∞	∞	∞	Begin met A
–	3	∞	∞	6	5	Kies B, vanaf A
–	–	1	∞	6	4	Kies C, vanaf B
–	–	–	6	6	4	Kies F, vanaf B
–	–	–	5	2	–	Kies E, vanaf F
–	–	–	5	–	–	Kies D, vanaf F



Een rij in de tabel komt overeen met het array `tak` in pseudo-code op vorige slide.



// invoer: **samenhangende, ongerichte** gewogen graaf $G = (V, E)$

// uitvoer: E_T , verzameling takken van minimale opspannende boom T van G

sorteer E op gewicht (in oplopende volgorde): $e_{i_1}, e_{i_2}, \dots, e_{i_m}$;

$E_T = \emptyset$;

takteller = 0;

k = 0;

while takteller < $|V| - 1$ **do**

 k = k+1;

if $E_T \cup \{e_{i_k}\}$ is acyclisch **then**

$E_T = E_T \cup \{e_{i_k}\}$;

 takteller = takteller+1;

fi

do

// invoer: **samenhangende, ongerichte** gewogen graaf $G = (V, E)$

// uitvoer: E_T , verzameling takken van minimale opspannende boom T van G

sorteer E op gewicht (in oplopende volgorde): $e_{i_1}, e_{i_2}, \dots, e_{i_m}$;

$E_T = \emptyset$;

takteller = 0;

k = 0;

while takteller < $|V| - 1$ **do**

 k = k+1;

if $E_T \cup \{e_{i_k}\}$ is acyclisch **then**

$E_T = E_T \cup \{e_{i_k}\}$;

 takteller = takteller+1;

fi

do

Complexiteit...

// invoer: **samenhangende, ongerichte** gewogen graaf $G = (V, E)$

// uitvoer: E_T , verzameling takken van minimale opspannende boom T van G

sorteer E op gewicht (in oplopende volgorde): $e_{i_1}, e_{i_2}, \dots, e_{i_m}$;

$E_T = \emptyset$;

takteller = 0;

k = 0;

while takteller < $|V| - 1$ **do**

 k = k+1;

if $E_T \cup \{e_{i_k}\}$ is acyclisch **then**

$E_T = E_T \cup \{e_{i_k}\}$;

 takteller = takteller+1;

fi

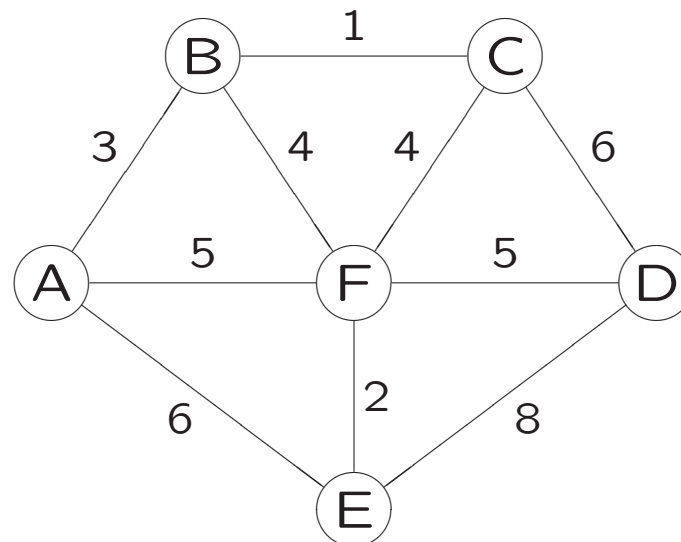
do

Complexiteit (met herhaald DFS of BFS): $\Theta(m * \log m + 1 + m * n)$

Complexiteit Kruskal, m.b.v. **Union-Find**

Abstract data type van een collectie disjuncte deelverzamelingen van een eindige verzameling

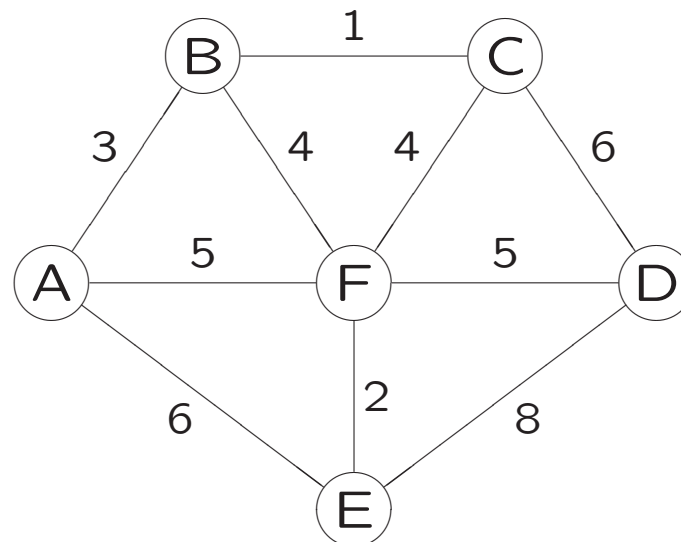
- $\text{makeset}(x)$
- $\text{find}(x)$
- $\text{union}(x, y)$



Twee implementaties Union-Find... (zie werkcollege 12)

Abstract data type van een collectie disjuncte deelverzamelingen van een eindige verzameling

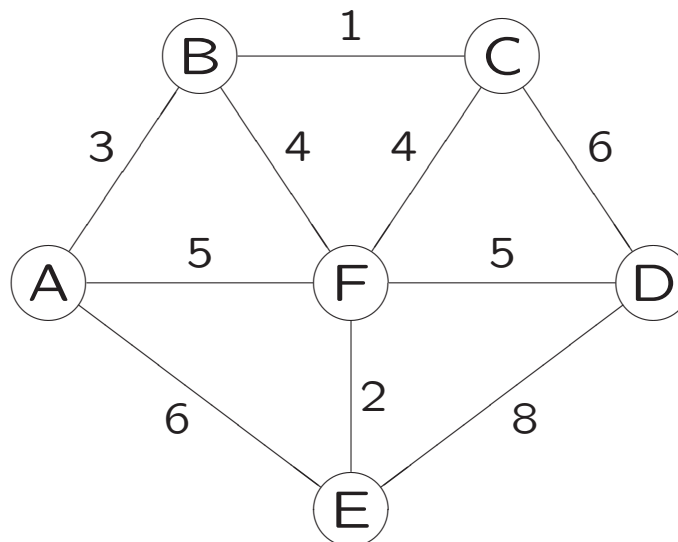
- $\text{makeset}(x)$
- $\text{find}(x)$
- $\text{union}(x, y)$



Complexiteit Kruskal, m.b.v. **Union-Find** (boomimplementatie)

Abstract data type van een collectie deelverzamelingen van een eindige verzameling

- $\text{makeset}(x)$: $O(1)$
- $\text{find}(x)$: $O(\log n)$
- $\text{union}(x, y)$: $O(1)$, als ...



// invoer: **samenhangende, ongerichte** gewogen graaf $G = (V, E)$

// uitvoer: E_T , verzameling takken van minimale opspannende boom T van G

sorteer E op gewicht (in oplopende volgorde): $e_{i_1}, e_{i_2}, \dots, e_{i_m}$;

$E_T = \emptyset$;

takteller = 0;

k = 0;

while takteller < $|V| - 1$ **do**

 k = k+1;

if $E_T \cup \{e_{i_k}\}$ is acyclisch **then**

$E_T = E_T \cup \{e_{i_k}\}$;

 takteller = takteller+1;

fi

do

Complexiteit...

// invoer: **samenhangende, ongerichte** gewogen graaf $G = (V, E)$

// uitvoer: E_T , verzameling takken van minimale opspannende boom T van G

sorteer E op gewicht (in oplopende volgorde): $e_{i_1}, e_{i_2}, \dots, e_{i_m}$;

$E_T = \emptyset$;

takteller = 0;

k = 0;

while takteller < $|V| - 1$ **do**

 k = k+1;

if $E_T \cup \{e_{i_k}\}$ is acyclisch **then**

$E_T = E_T \cup \{e_{i_k}\}$;

 takteller = takteller+1;

fi

do

Complexiteit: $O(m \log m + 1 + m \log n)$

Correctheid:

Voor de eerste iteratie, en na elke iteratie van de while-lus is E_T nog bevat in (uit te breiden tot) een minimale opspannende boom.

Bewijs: ...

- **Lezen/leren bij dit college:**
paragraaf 9.1, 9.2 (inclusief implementatie van Union Find), 9.3, slides
- **Geen werkcollege** deze week
- **Practicumbijeenkomst** programmeeropdracht 2:
donderdag 22 april 2021, 14.15-16.00, Kaltura liveroom
- **Volgend (laatste!) hoorcollege:**
woensdag 28 april 2021, 14.15–16.00 (weblecture) ?