Elfde college algoritmiek

24 april 2024

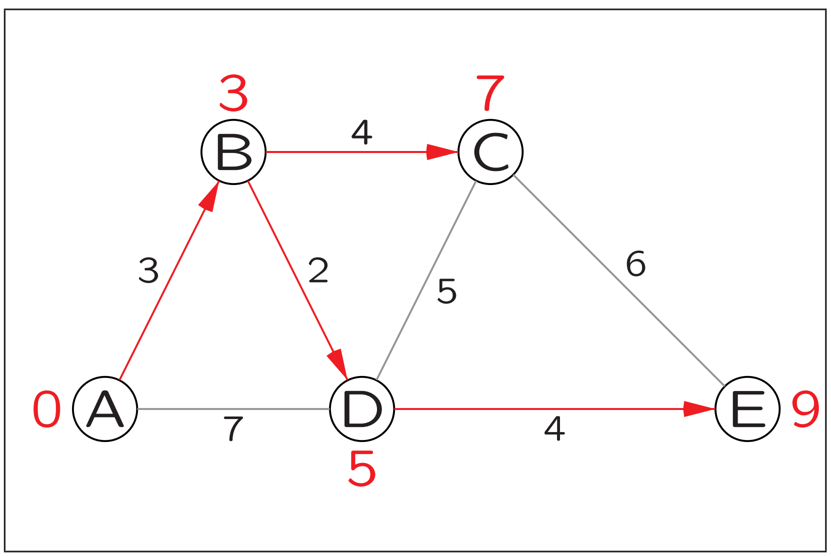
Gretige Algoritmen,
Algoritme van Dijkstra

- nakijken opdracht 1
- deadline opdracht 2

Gegeven een graaf G met gewichten op de takken, en een beginknoop s. We nemen aan dat alle gewichten ≥ 0 zijn.

Gevraagd: voor elke willekeurige knoop v in de graaf (de lengte van) het/een kortste pad van s naar v.

Merk op dat al deze kortste paden vanuit s samen een boomstructuur vormen.



De kortste paden vanuit A zijn:

A $\longrightarrow$ B: lengte 3

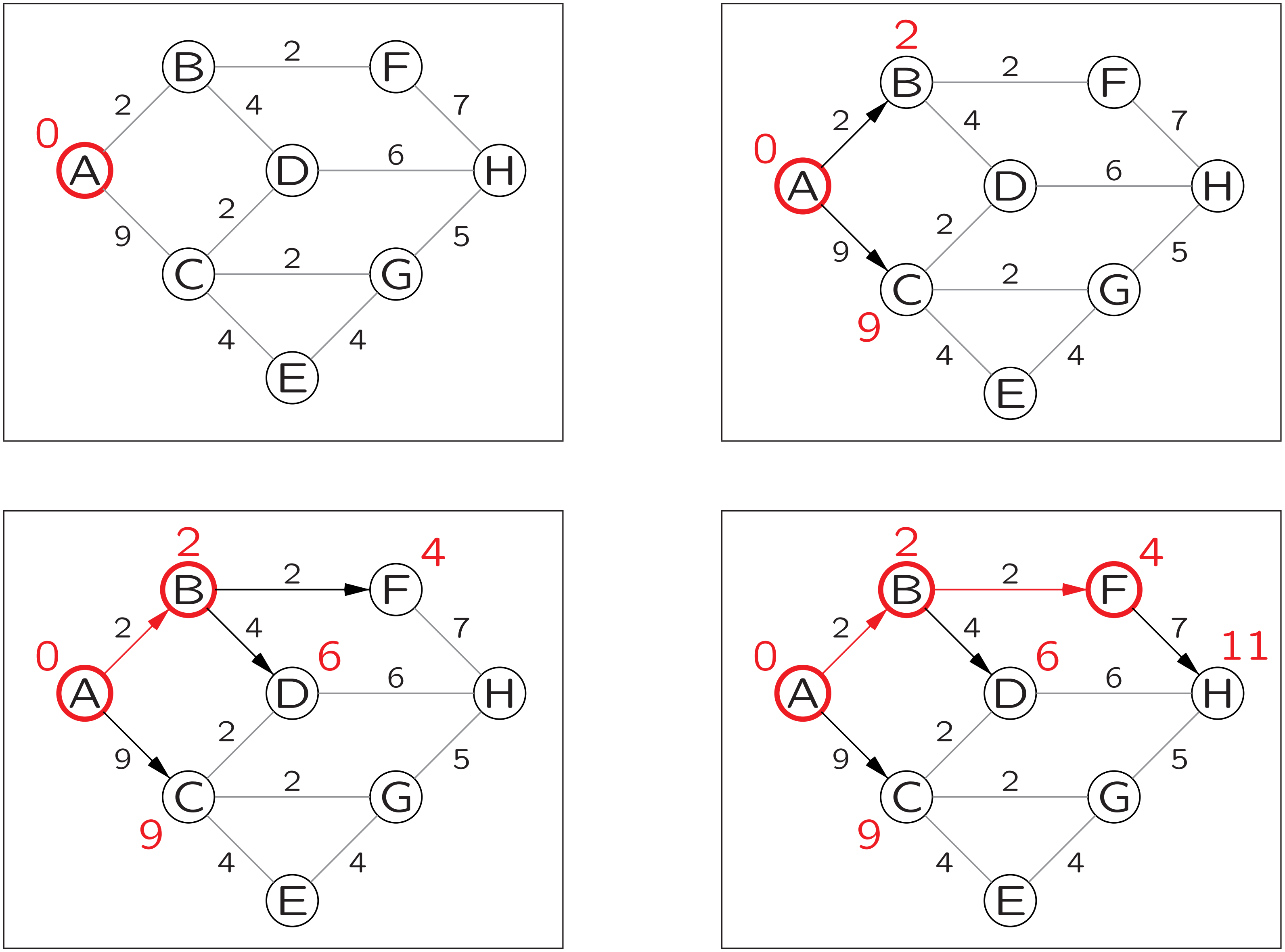
A $\longrightarrow$ B $\longrightarrow$ D: lengte 5

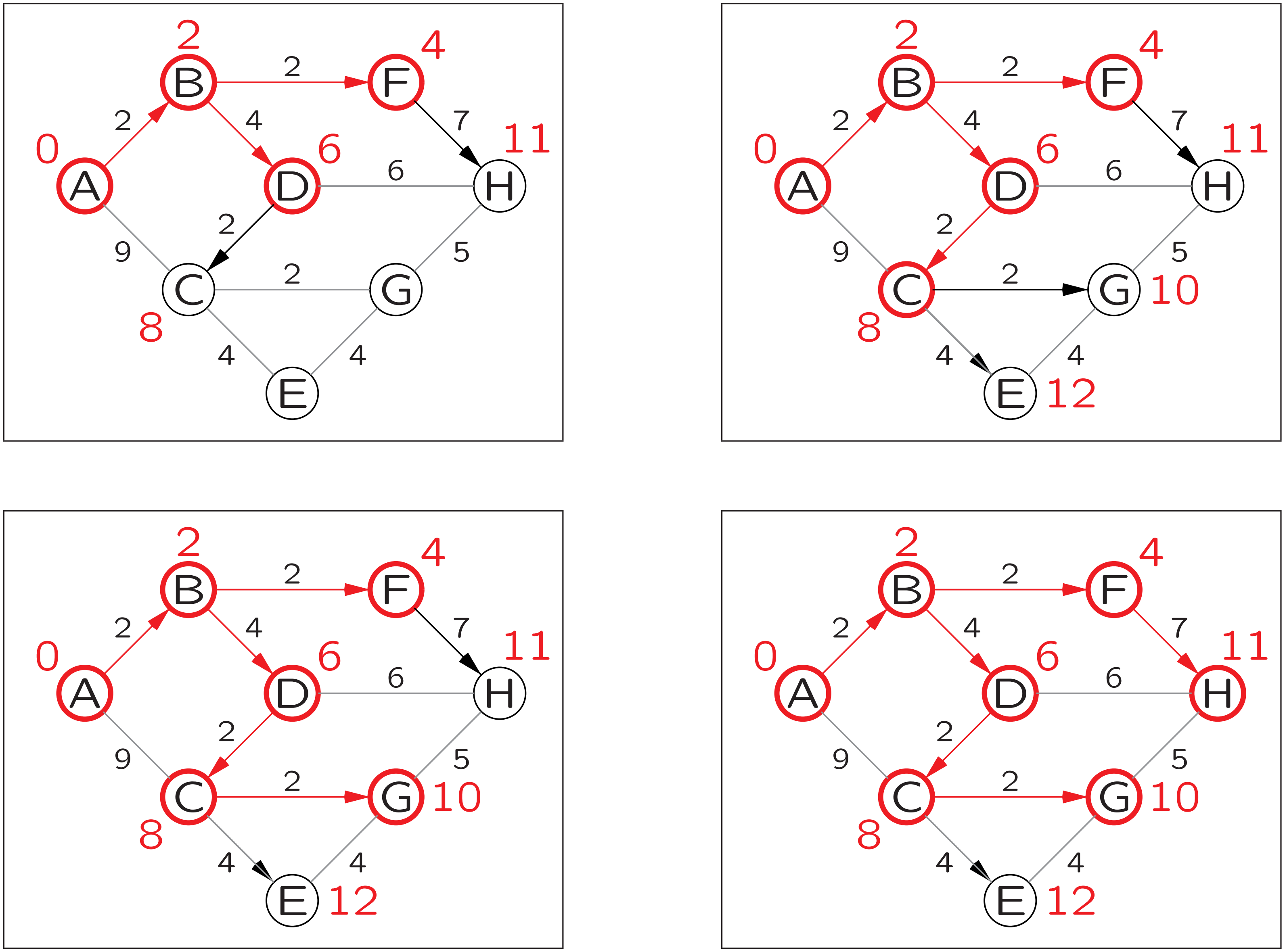
A $\longrightarrow$ B $\longrightarrow$ C: lengte 7

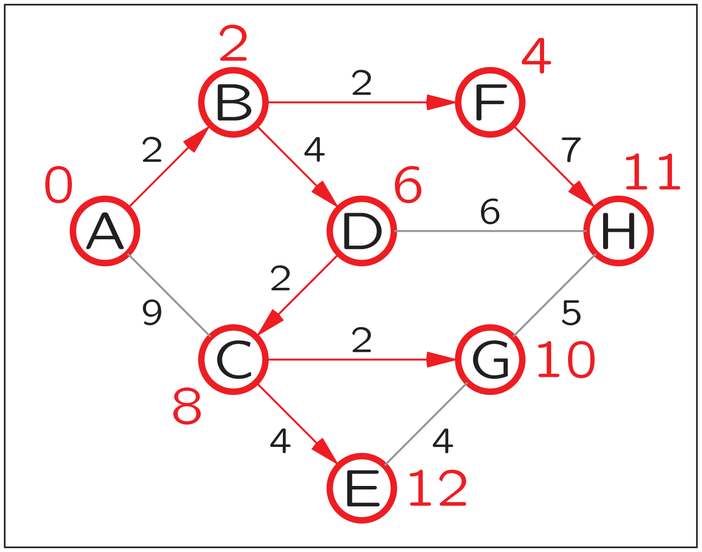
A $\longrightarrow$ B $\longrightarrow$ D $\longrightarrow$ E: lengte 9

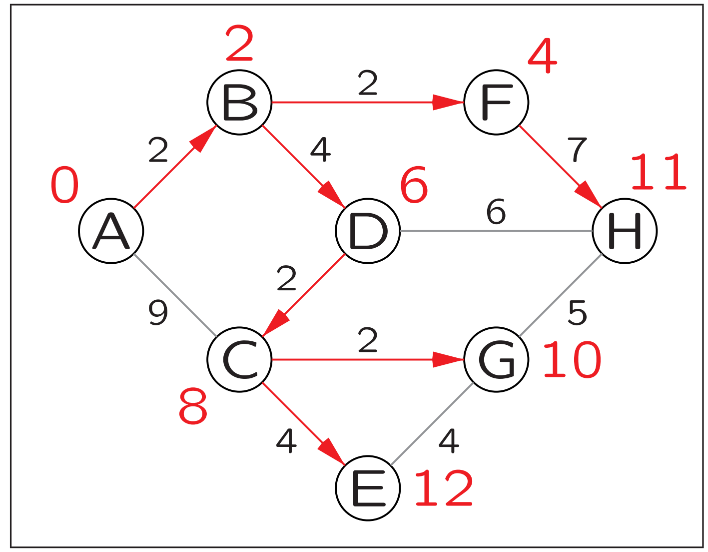
Oplossing: het algoritme van Dijkstra is een gretig algoritme, dat de kortste paden van s naar elk van de andere knopen vindt in volgorde van hun lengte. In elke stap wordt een knoop (en daarmee het pad van s naar die knoop) gekozen waarvoor het tot nu toe bekende pad vanaf s zo kort mogelijk is.

Edsger Wybe Dijkstra (Rotterdam, 11 mei 1930 — Nuenen, 6 augustus 2002) was een Nederlandse wiskundige en informaticus die veel voor de informatica heeft gedaan, met name op het gebied van gestructureerd programmeren. In 1972 werd hij onderscheiden met de Turing Award.


0
A
B
C
D
E
F
G
H
2
2
4
9
2
6
7
2
5
4
4
0
2
9
A
B
C
D
E
F
G
H
2
2
4
9
2
6
7
2
5
4
4
0
2
4
6
9
A
B
C
D
E
F
G
H
2
2
4
9
2
6
7
2
5
4
4
0
2
4
6
11
9
A
B
C
D
E
F
G
H
2
2
4
9
2
6
7
2
5
4
4


A B C D E F G H
0 2 4 6 11 8
2 2 4 7 6 2 9 2 5 4 4
A B C D E F G H
0 2 4 6 11 8 10 12
2 2 4 7 6 2 9 2 5 4 4
A B C D E F G H
0 2 4 6 11 8 10 12
2 2 4 7 6 2 9 2 5 4 4
A B C D E F G H
0 2 4 6 11 8 10 12
2 2 4 7 6 2 9 2 5 4 4





Het algoritme is klaar: alle knopen gehad

Alle kortste paden vanuit A met hun lengtes

Als je gewoon wilt doortekenen in één plaatje...

Begin met A, afstand 0.

A $\rightarrow$ B: $0 + 2 = 2$. OK.

A $\rightarrow$ C: $0 + 9 = 9$. OK.

Kies B, afstand 2, vanaf A.

B $\rightarrow$ D: $2 + 4 = 6$. OK.

B $\rightarrow$ F: $2 + 2 = 4$. OK.

Kies F, afstand 4, vanaf B.

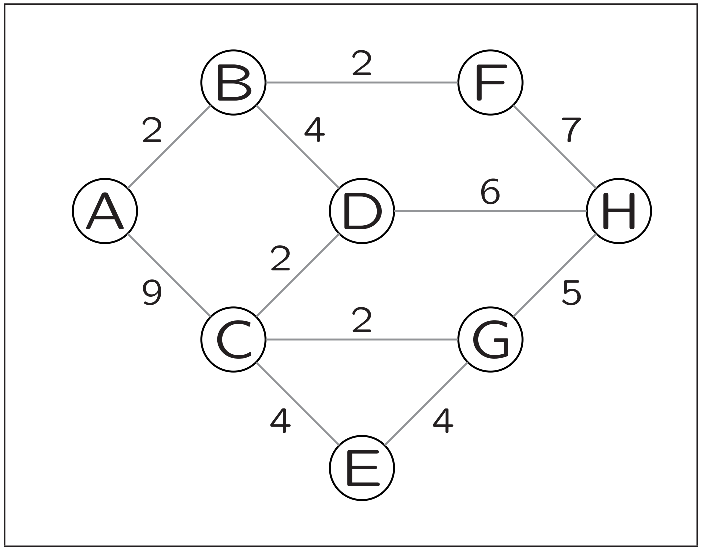
F $\rightarrow$ H: $4 + 7 = 11$. OK.

Kies D, afstand 6, vanaf B.

D $\rightarrow$ C: $6 + 2 = 8 < 9$. OK.

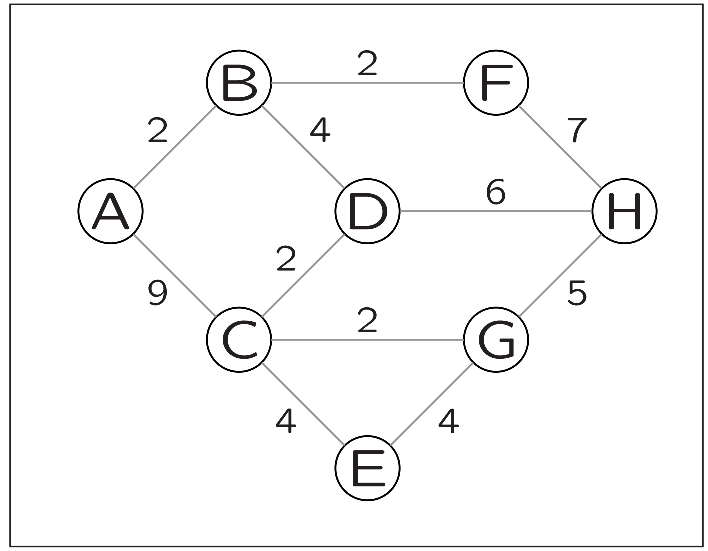
D $\rightarrow$ H: $6 + 6 = 12 > 11$. X.

Enzovoort.



Als je gewoon wilt doortekenen in één plaatje. . . (alternatief)

A	B	C	D	E	F	G	H	Actie
0	∞	∞	∞	∞	∞	∞	∞	Begin met A
–	2	9	∞	∞	∞	∞	∞	Kies B, vanaf A
–	–	9	6	∞	4	∞	∞	Kies F, vanaf B
–	–	9	6	∞	–	∞	11	Kies D, vanaf B
–	–	8	–	∞	–	∞	11	Kies C, vanaf D
–	–	–	–	12	–	10	11	Kies G, vanaf C
–	–	–	–	12	–	–	11	Kies H, vanaf F
–	–	–	–	12	–	–	–	Kies E, vanaf C



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

- gewichten ≥ 0

- kortste pad $A - B$ vs alle kortste paden vanuit A

// 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 pad$[v]$ = de lengte van een kortste pad van s naar v

for $v \in V$ **do**
    pad$[v] := \infty$;
od
pad$[s] := 0$;
$U := \emptyset$;

while ($U \neq V$) **do**
    vind knoop $v^* \in V \setminus U$ met pad$[v^*]$ minimaal;
    $U := U \cup \{v^*\}$;
    for alle knopen v aangrenzend aan v^* **do**
        if pad$[v^*]$ + gewicht$(v^*, v) <$ pad$[v]$ **then**
            pad$[v]$:= pad$[v^*]$ + 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 pad[v] = de lengte van een kortste pad van s naar v

for v ∈ V do
        pad[v] := ∞;
od
pad[s] := 0;
U := ∅;

while ( U ≠ V ) do
        vind knoop v* ∈ V \ U met pad[v*] minimaal;
        U := U ∪ {v*};
        for alle knopen v aangrenzend aan v* do
              if pad[v*] + gewicht(v*, v) < pad[v] then
                    pad[v] := pad[v*] + 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 pad[v] = de lengte van een kortste pad van s naar v

for v ∈ V do
        pad[v] := ∞;
od
pad[s] := 0;
U := ∅;

while ( U ≠ V ) do
        vind knoop v* ∈ V \ U met pad[v*] minimaal;
        U := U ∪ {v*};
        for alle knopen v aangrenzend aan v* do
              if pad[v*] + gewicht(v*, v) < pad[v] then
                    pad[v] := pad[v*] + gewicht(v*, v);
              fi
        od
od
```

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

Bij Dijkstra:
kies knoop v^* buiten boom met laagste kandidaatwaarde 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 pad[v] = de lengte van een kortste pad van s naar v

for v ∈ V do
        pad[v] := ∞;
od
pad[s] := 0;
U := ∅;

while ( U ≠ V ) do
        vind knoop v* ∈ V \ U met pad[v*] minimaal;
        U := U ∪ {v*};
        for alle knopen v aangrenzend aan v* do
              if pad[v*] + gewicht(v*, v) < pad[v] then
                    pad[v] := pad[v*] + 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 pad[v] = de lengte van een kortste pad van s naar v

for v ∈ V do
        pad[v] := ∞;
od
pad[s] := 0;
U := ∅;

while ( U ≠ V ) do
        vind knoop v* ∈ V \ U met pad[v*] minimaal;
        U := U ∪ {v*};
        for alle knopen v aangrenzend aan v* do
              if pad[v*] + gewicht(v*, v) < pad[v] then
                    pad[v] := pad[v*] + 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 pad$[v]$ al de definitieve kortste afstand aan. Moet bewezen worden.

- Voor de andere knopen w geldt na elke ronde (= doorgang door de while):

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

  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 pad$[v^*]$ + gewicht$(v^*, v) <$ pad$[v]$ **then**
        pad$[v]$:= pad$[v^*]$ + gewicht(v^*, v);
        nieuwe kandidaattak voor v: (v^*, v)
fi

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

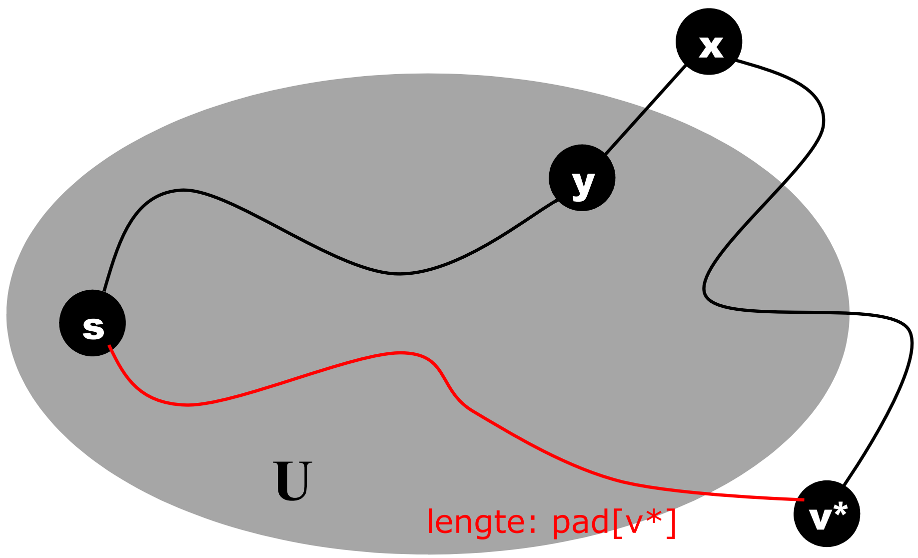
Na elke ronde (dus ook na de laatste, wanneer $U = V$) van het algoritme van Dijkstra geldt:

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

Om dit te bewijzen moet je laten zien dat:

- wanneer v^* wordt toegevoegd aan U, pad$[v^*]$ inderdaad de lengte van het kortste pad van s naar v^* bevat pad[v^*] is daadwerkelijk lengte van een pad van s naar v^*:

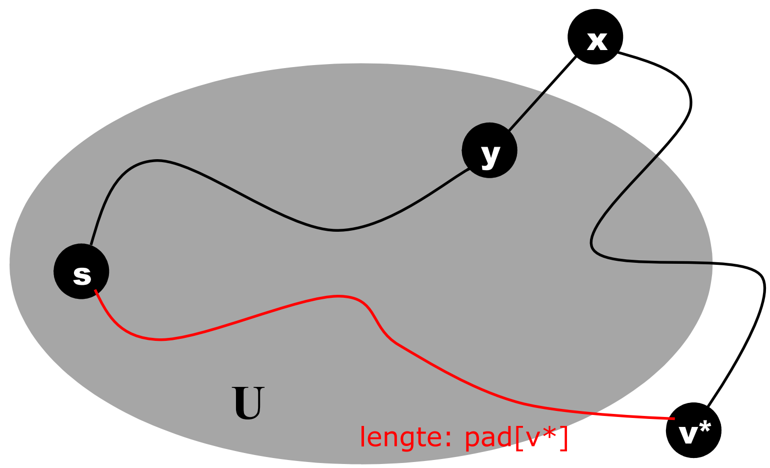
$$(\#)\ \mathsf{pad}[v^*] = \min_{u \in U}\{\mathsf{pad}[u] + \mathsf{gewicht}(u, v^*)\}^*$$



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

 Essentie van het bewijs (*)

Bekijk, behalve het pad ter lengte 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.

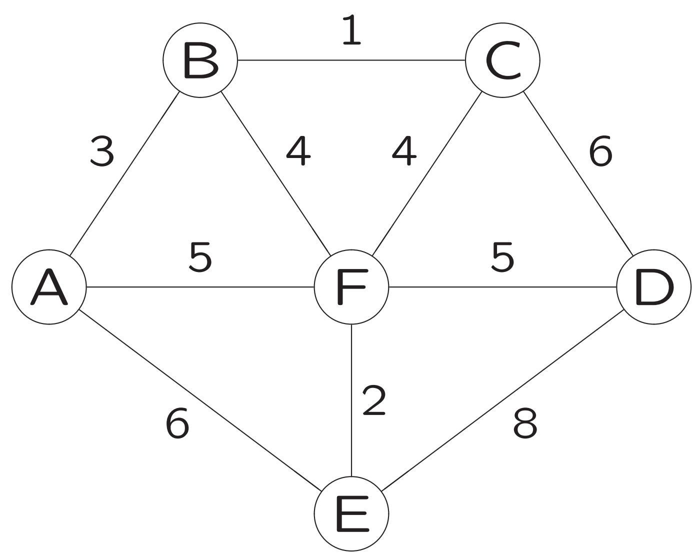


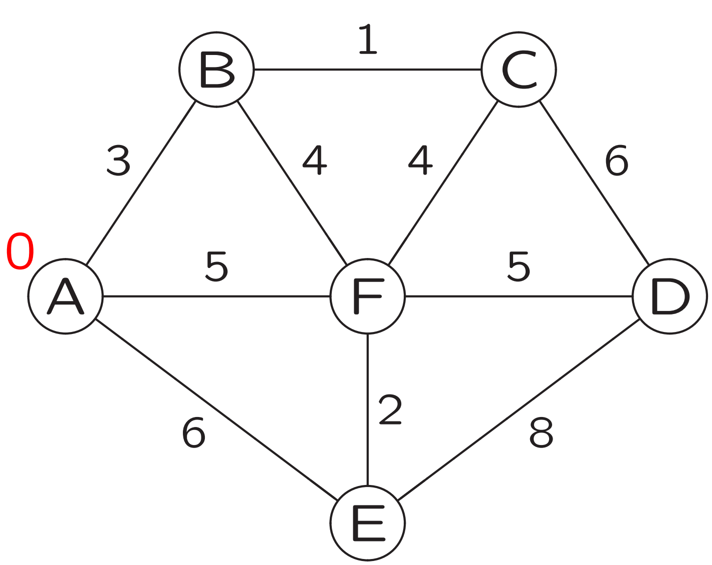
Dan geldt: $d(s, y, x, v^*) \geq d(s, y, x) = d(s, y) +$ gewicht$(y, x) \geq$ pad$[y] +$ gewicht$(y, x) \geq$ pad$[x] \geq$ pad$[v^*]$ omdat respectievelijk alle gewichten van de takken ≥ 0 zijn, pad$[y]$ volgens inductiehypothese kortste afstand is naar y, (#) geldt voor x, en v^* gekozen was in deze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

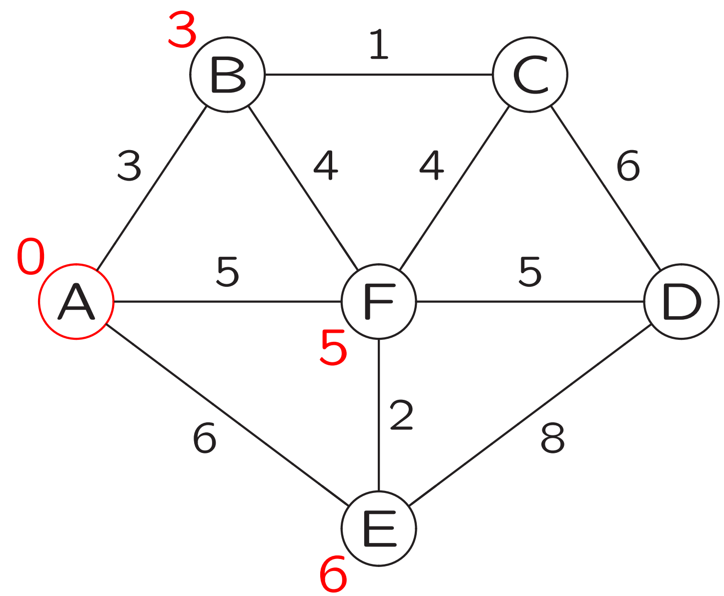
QED

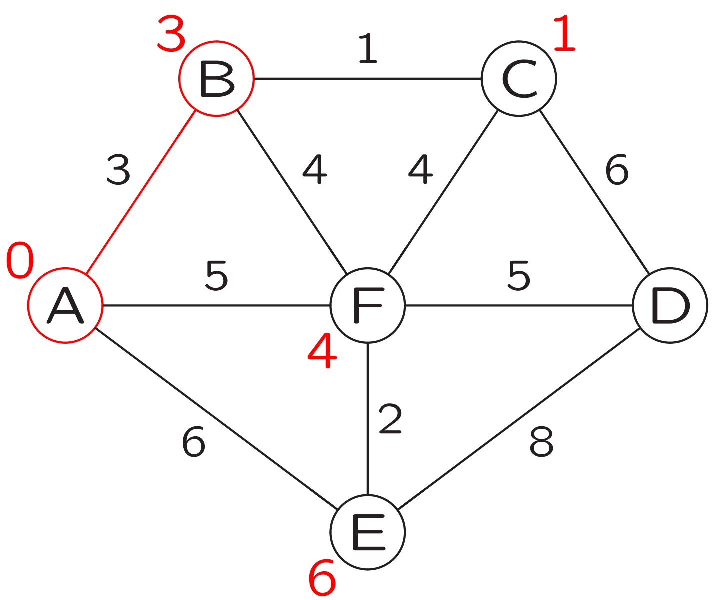
Gegeven een samenhangende, ongerichte graaf G met gewichten op de takken.

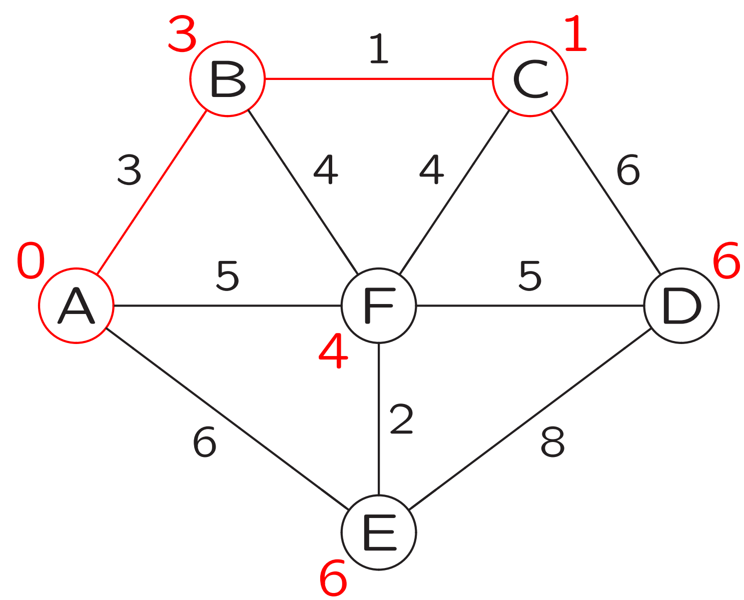
Gevraagd: een opspannende boom van G met minimaal totaal gewicht.

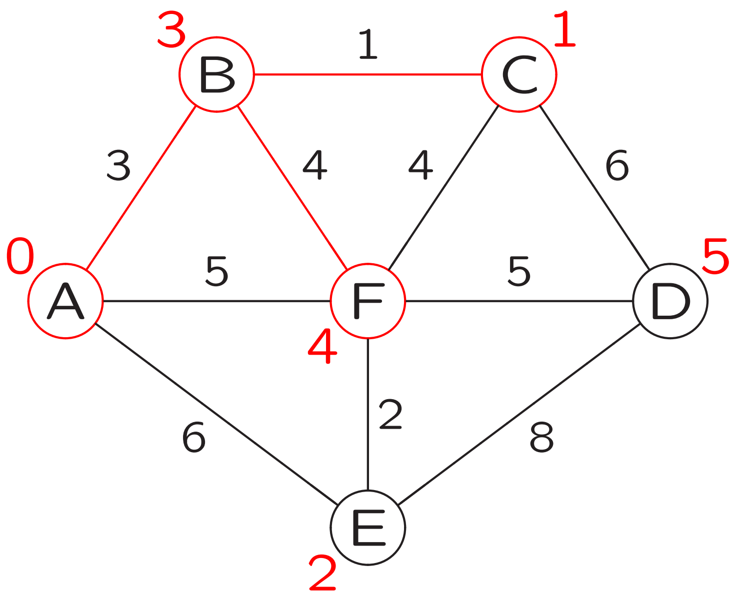


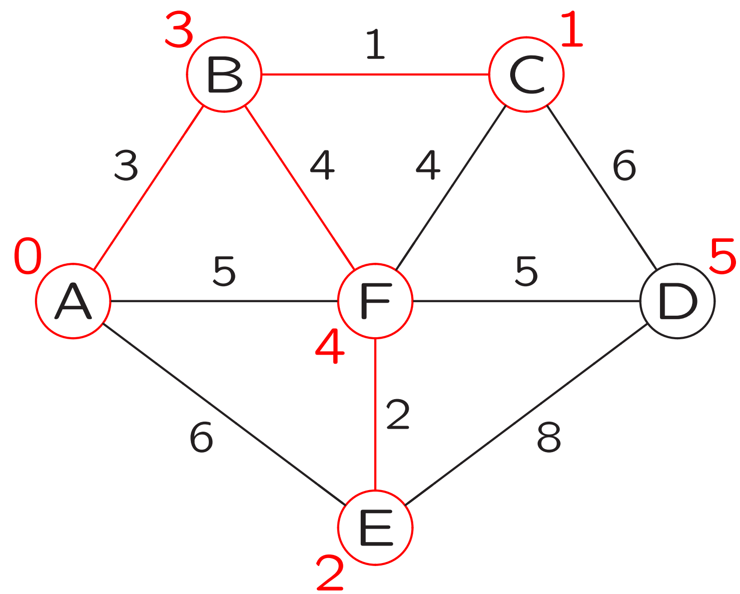

B
1
C
3
4
4
6
0
A
5
F
5
D
2
6
8
E

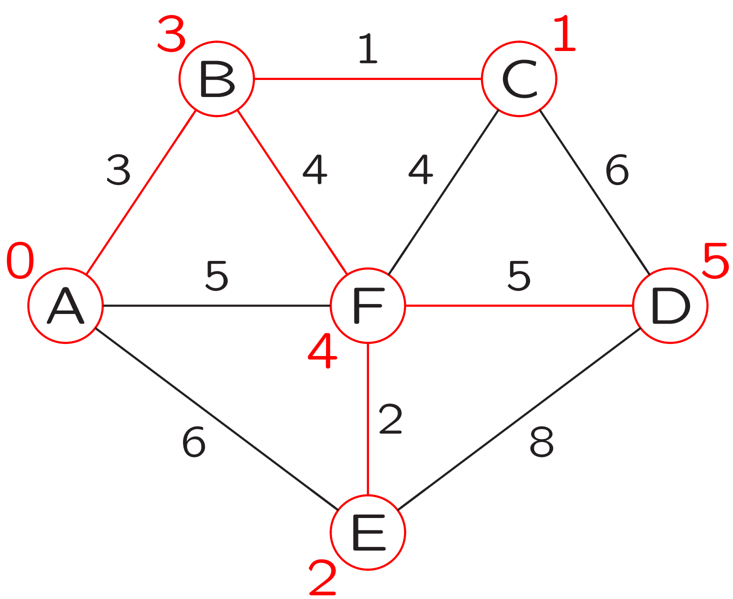

3
B
1
C
3
4
4
6
0
A
5
F
5
D
5
2
6
8
E
6


3
B
1
C
1
3
4
4
6
0
A
5
F
5
D
4
2
6
8
E
6


3
B
1
C
1
3
4
4
6
0
A
5
F
5
D
6
4
2
6
8
E
6


3
B
1
C
1
3
4
4
6
0
A
5
F
5
D
5
4
6
2
8
E
2


3
B
1
C
1
3
4
4
6
0
A
5
F
5
D
5
4
6
2
8
E
2


3
B
1
C
1
3
4
4
6
0
A
5
F
5
D
5
4
6
2
8
E
2

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

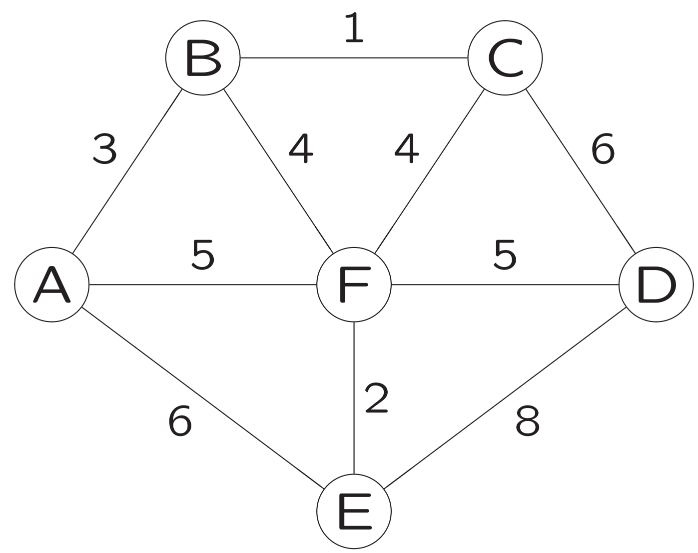
```
for v ∈ V do
        tak[v] := ∞;
od
tak[s] := 0;
U := ∅;
E_T := ∅;

while ( U ≠ V ) do
        vind knoop v* ∈ V \ U met tak[v*] minimaal;
        U := U ∪ {v*};
        voeg bijbehorende tak naar v* toe aan E_T (als v* ≠ 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)

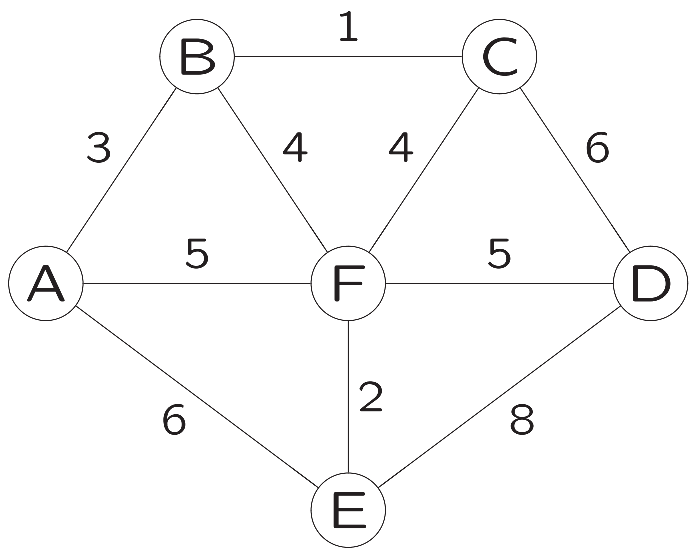
Als je gewoon wilt doortekenen in één plaatje. . .

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.

Algoritme van Kruskal



// 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): $e_{i_1}, e_{i_2}, \ldots, e_{i_m}$;
$E_T = \emptyset$;
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): $e_{i_1}, e_{i_2}, \ldots, e_{i_m}$;
$E_T = \emptyset$;
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

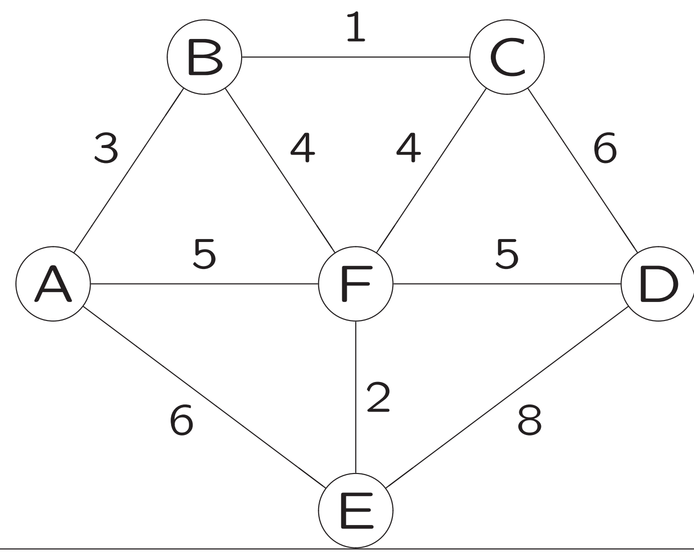
sorteer E op gewicht (in oplopende volgorde): $e_{i_1}, e_{i_2}, \ldots, e_{i_m}$;
$E_T = \emptyset$;
takteller = 0;
k = 0;
while takteller $< |V| - 1$ **do**
    k = k+1;
    if $E_T \cup \{e_{i_k}\}$ is acyclisch **then**
        $E_T = E_T \cup \{e_{i_k}\}$;
        takteller = takteller+1;
    fi
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

- makeset(x)
- find(x)
- union(x, y)



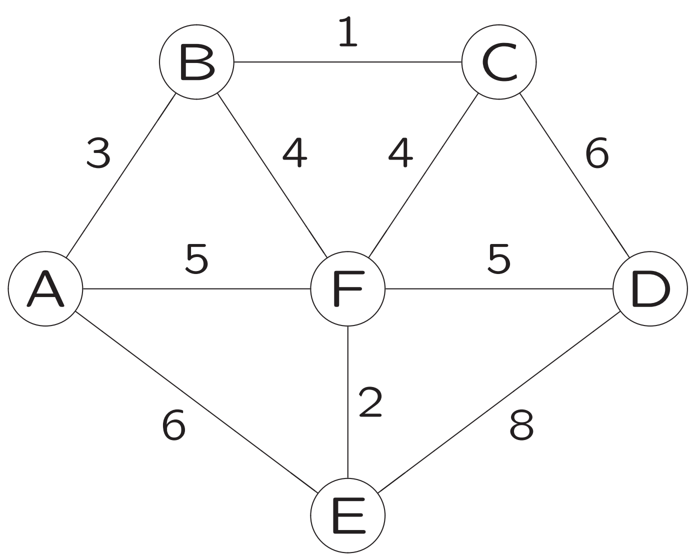
Drie implementaties Union-Find (zie ook werkcollege 12):

1. alleen representant per deelverzameling

2. representant per deelverzameling en linked-list met elementen

3. (omgekeerd gerichte) boom per deelverzameling

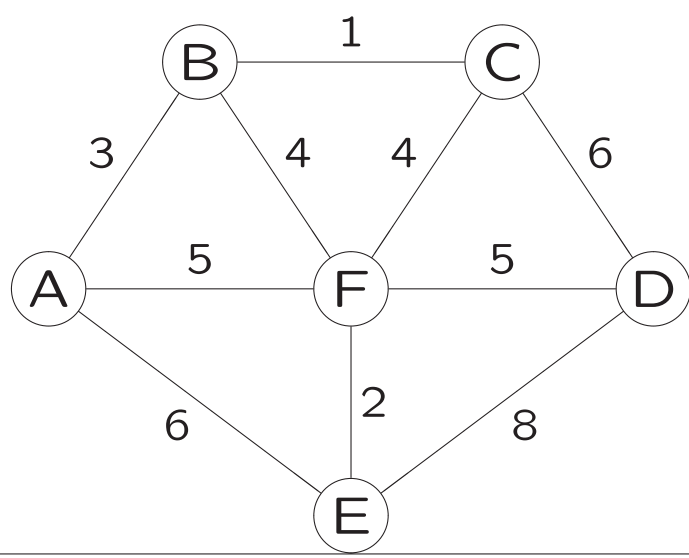
- makeset(x)
- find(x)
- union(x, y)



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

Abstract data type van een collectie deelverzamelingen van een eindige verzameling

- makeset(x): $O(1)$
- find(x): $O(\log n)$
- 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): $e_{i_1}, e_{i_2}, \ldots, e_{i_m}$;
$E_T = \emptyset$;
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): $e_{i_1}, e_{i_2}, \ldots, e_{i_m}$;
$E_T = \emptyset$;
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

- **Werkcollege** gretige algoritmen / Dijkstra:
  vrijdag 26 april 2024, 13.15–15.00, 407-409

- **Opgaven:**
  zie `http://www.liacs.leidenuniv.nl/~vlietrvan1/algoritmiek/`

- **Practicumbijeenkomst** programmeeropdracht 2:
  Maandag 29 april 2024, 13.15–15.00, computerzalen Snellius

- **Volgend (laatste!) hoorcollege:**
  Woensdag 1 mei 2024, 15.15–17.00

- **Daarna:**
  15 mei 2024: tentamen oefenen