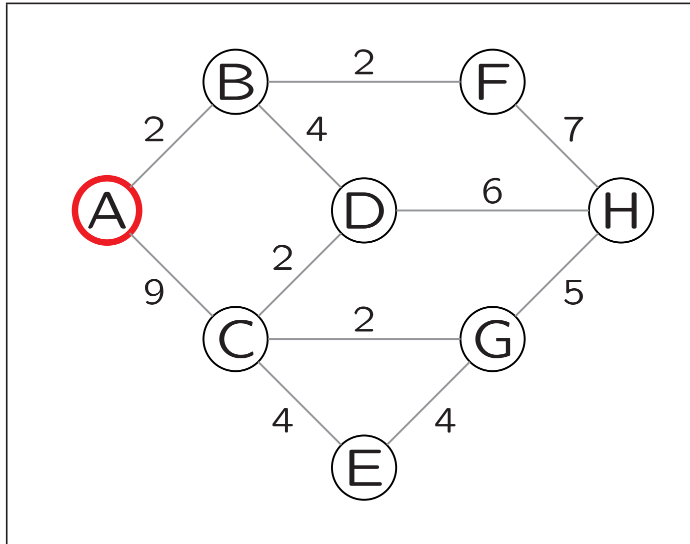
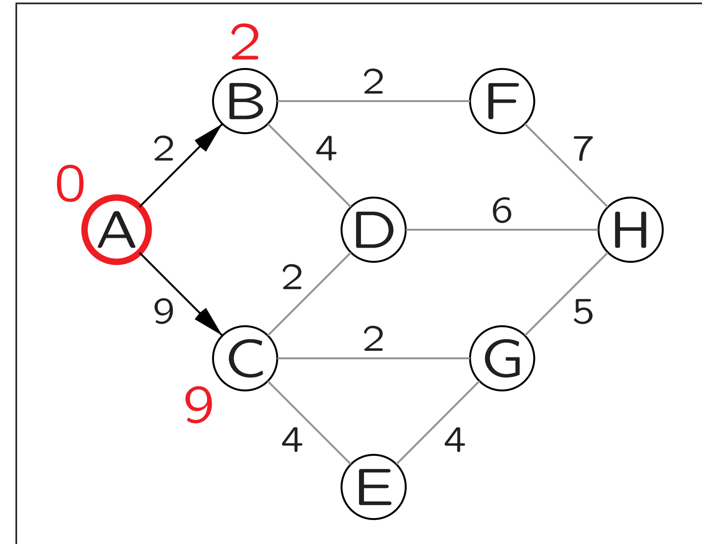
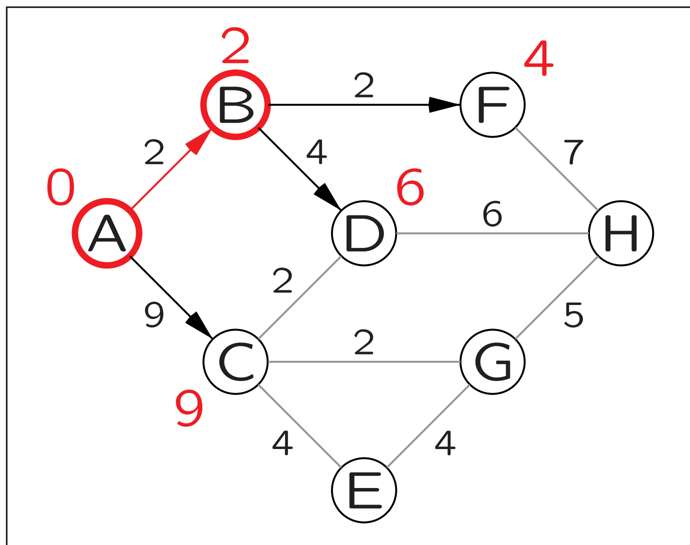
Elfde college algoritmiek

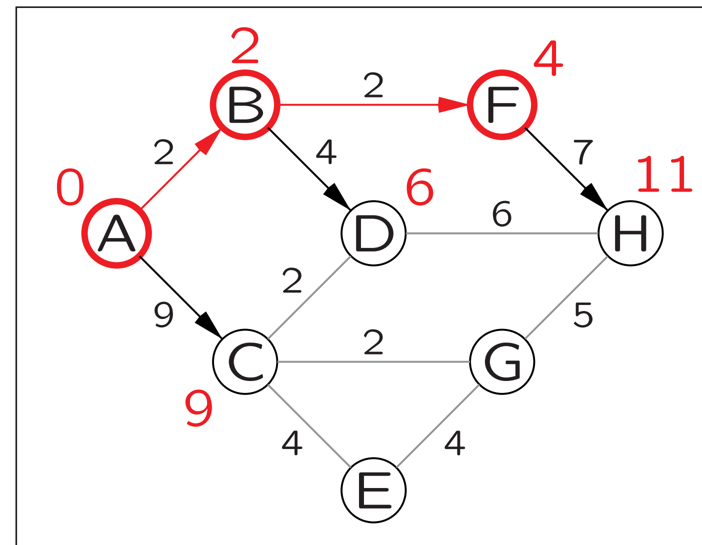
28 april / 4 mei 2017

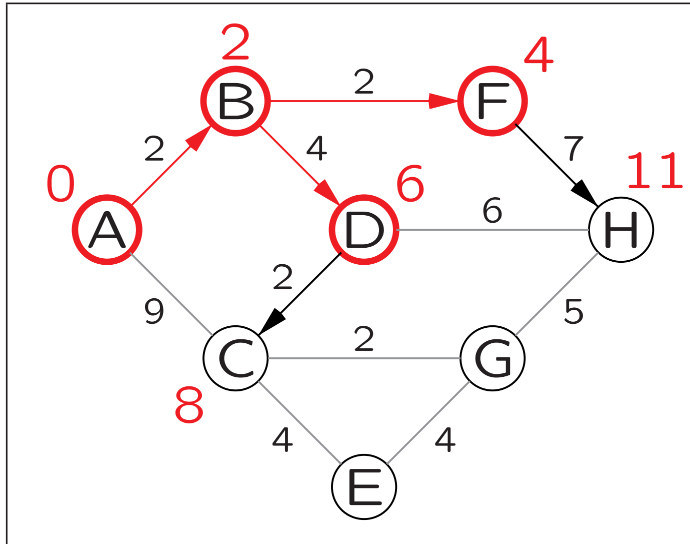
Algoritme van Dijkstra, Heap, Heapify & Heapsort


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

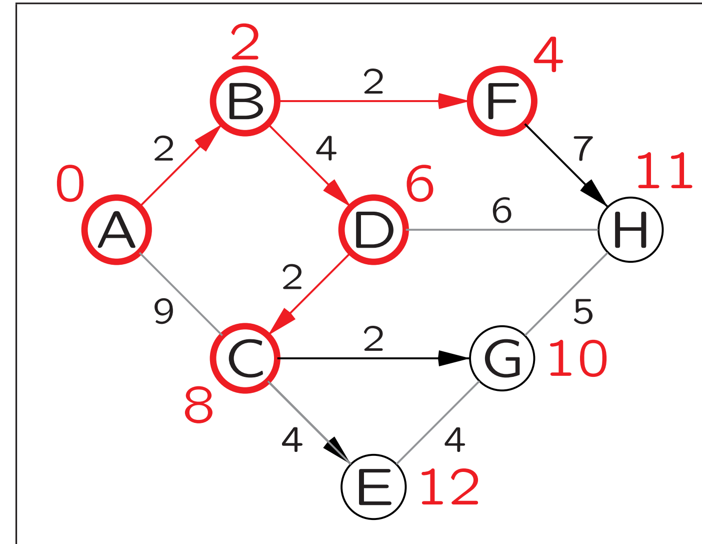

0
2
9
A
B
C
D
E
F
G
H
2
9
2
4
2
2
6
4
4
7
5


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

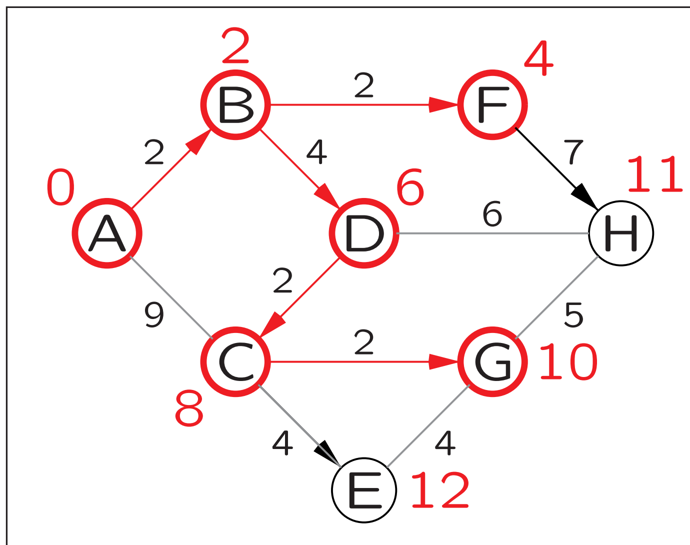

0
2
9
4
6
11
A
B
C
D
E
F
G
H
2
9
2
4
2
2
6
4
4
7
5



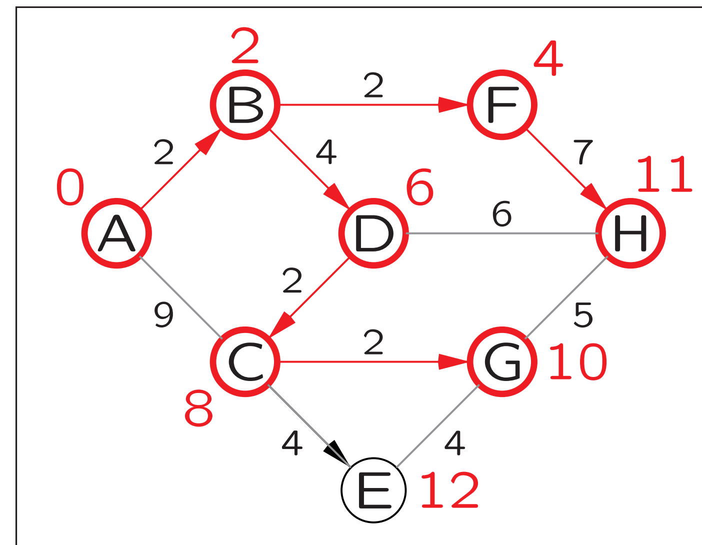
A
B
C
D
E
F
G
H
0
2
4
6
8
11
2
2
4
7
6
2
9
2
5
4
4



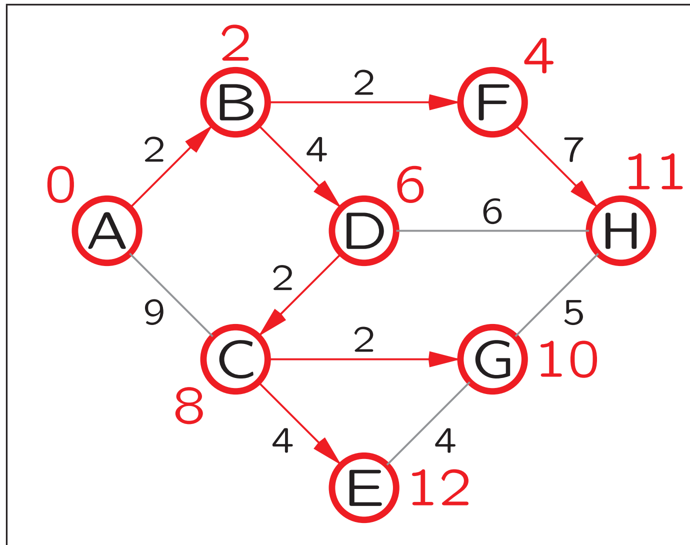
A
B
C
D
E
F
G
H
0
2
4
6
8
10
11
12
2
2
4
7
6
2
9
2
5
4
4

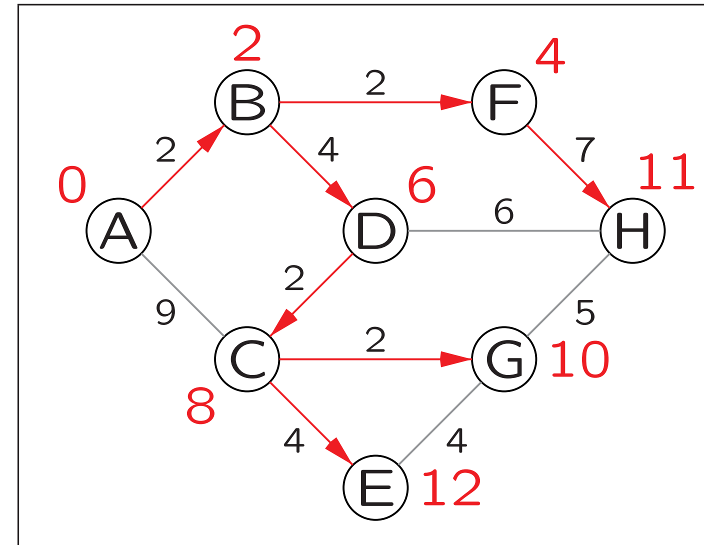


A
B
C
D
E
F
G
H
0
2
4
6
8
10
11
12
2
2
4
7
6
2
9
2
5
4
4



A
B
C
D
E
F
G
H
0
2
4
6
8
10
11
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

Uit college 10: Pseudo-code

// 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$[v]$:= ∞;
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):
$\Theta(n + 1 + n(n + 1 + n)) = \Theta(n^2)$

- 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.

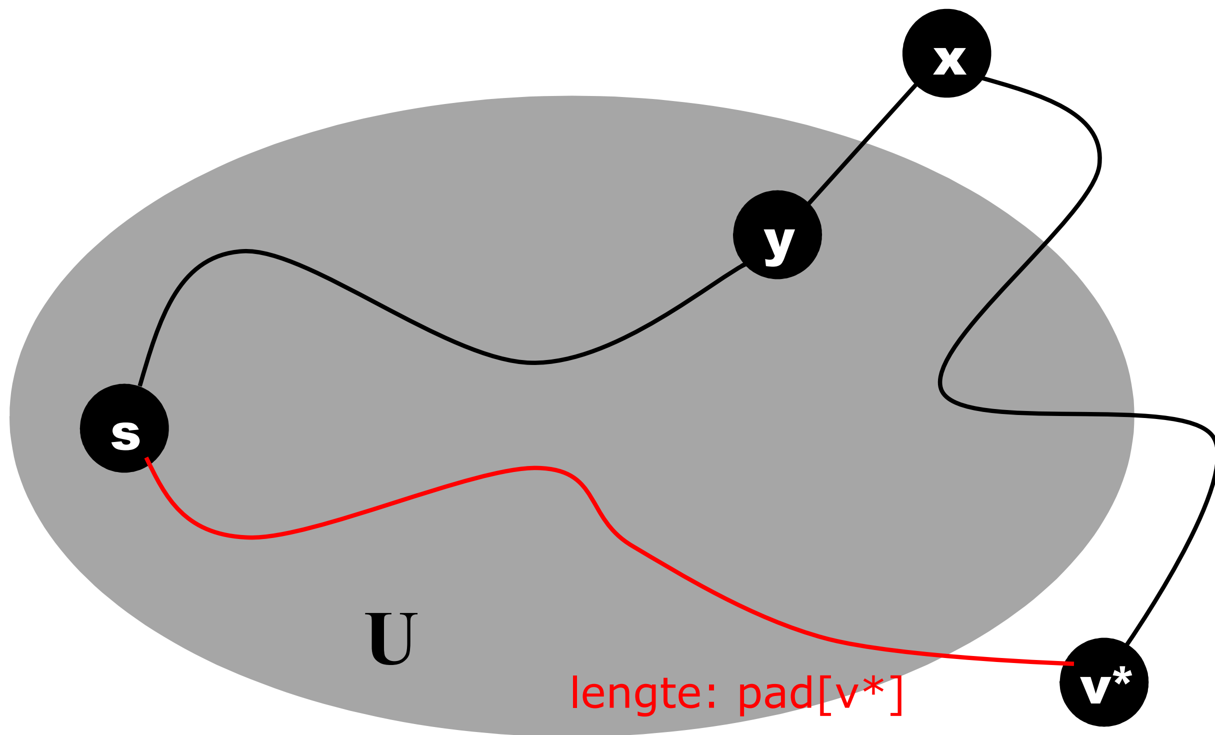
Na elke ronde (dus ook na de laatste, wanneer $U = V$) van het algoritme van Dijkstra geldt:

- 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.

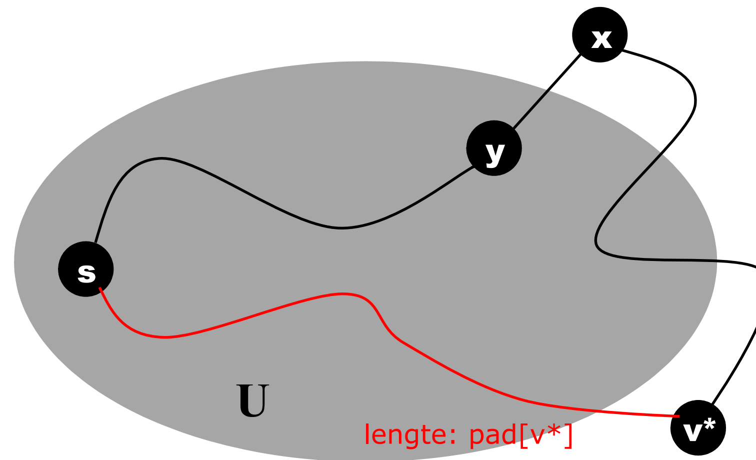
Om dit te bewijzen moet je laten zien dat:

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

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.

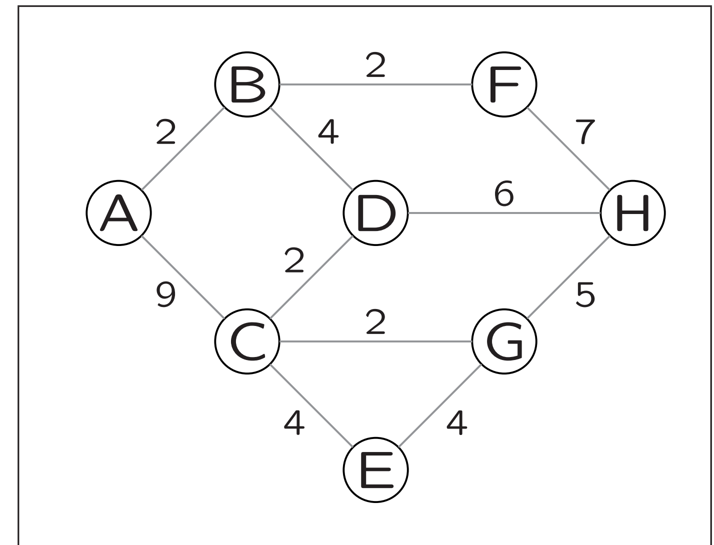


Dan geldt: $d(s, y, x, v^*) \geq d(s, y, x) = d(s, y) +$ gewicht$(y, x) \geq$ pad$[y]$ + gewicht$(y, x) \geq$ pad$[x] \geq$ pad$[v^*]$ omdat respectievelijk alle gewichten van de takken ≥ 0 zijn, pad$[y]$ volgens inductiehypothese kortste afstand is naar y, (#) geldt voor x, en v^* gekozen was in deze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

QED

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.

(Incorrecte!) opsplitsing while-lus:

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

while ($U \neq V$) **do**
  vind knoop $v^* \in V \setminus U$ met pad[v^*] minimaal;
  $U := U \cup \{v^*\}$;
od
while ($U \neq V$) **do**
  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): $O(n + 1 + n \log n + m \log n)$

Bij Dijkstra: kies knoop buiten boom met laagste kandidaatwaarde

Bij Branch & Bound (later): kies deeloplossing met beste ondergrens/boveng

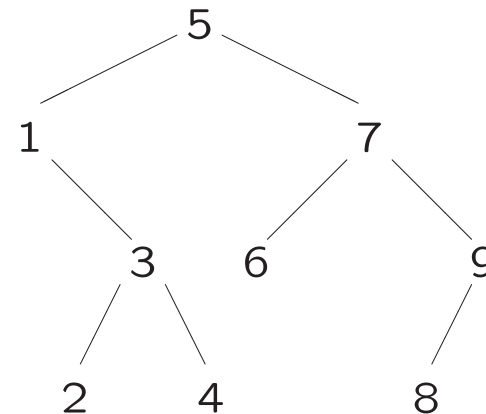
Priority Queue:

- objecten met prioriteit (en info)
- insert
- findmax (findmin)
- deletemax (deletemin)
- (optioneel) changepriority

Binaire zoekboom:

LWR

1 2 3 4 5 6 7 8 9



```
class knoop
{
  int info;
  knoop* links;
  knoop* rechts;
public:
  knoop ( ) // constructor
  {
    info = 0;
    links = NULL;
    rechts = NULL;
  }
};  // knoop
```

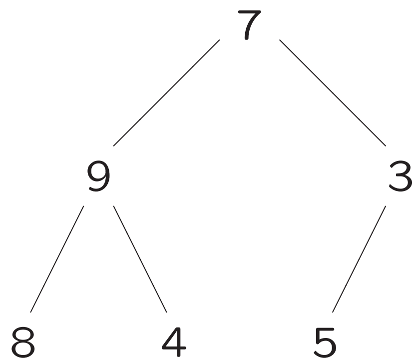
De binaire boom wordt gerepresenteerd door middel van een pointer naar de wortel:

```
knoop* wortel; // de ingang tot de binaire boom
```

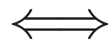
Netter om een klasse te gebruiken: zie Programmeermethoden

- Een complete binaire boom is een binaire boom waarbij alle nivo's geheel vol zitten, behalve eventueel het onderste. Op het onderste nivo mogen alleen de meest rechter knopen missen.

- Voorbeeld:

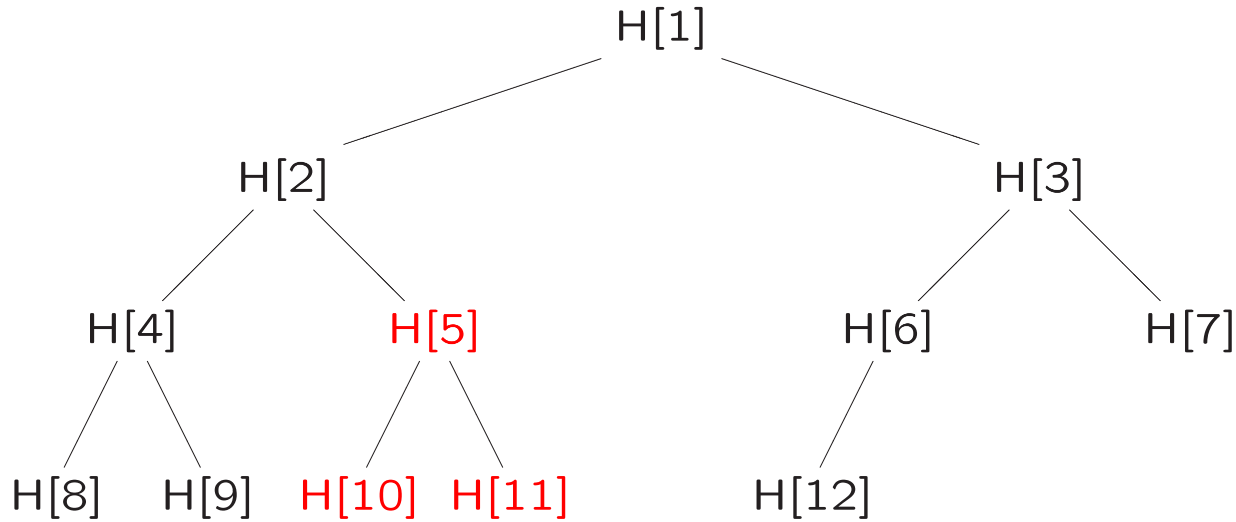


complete binaire boom

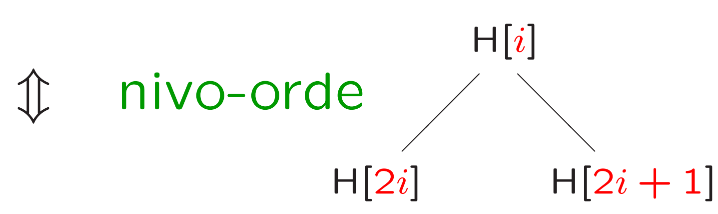


7 9 3 8 4 5

representatie als array



⇕ nivo-orde



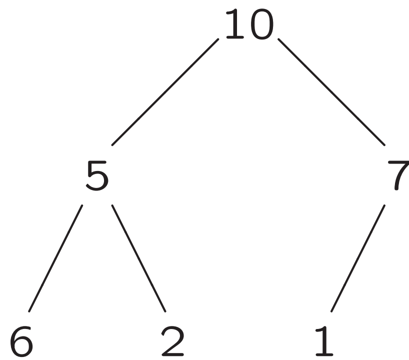
H[1] H[2] H[3] H[4] H[5] H[6] H[7] H[8] H[9] H[10] H[11] H[12]

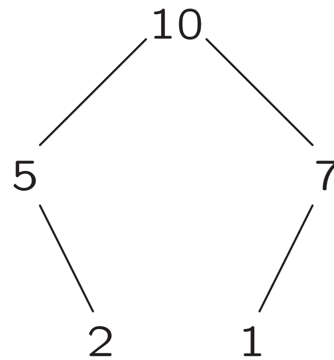
Definitie
Een heap (hoopstructuur) is een binaire boom met de volgende twee eigenschappen:

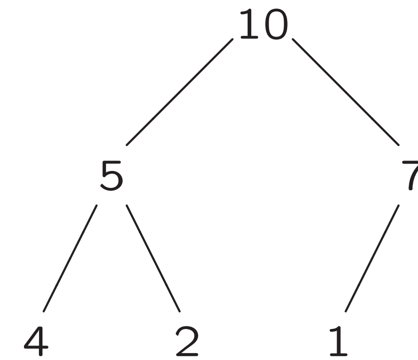
1. Vorm: de binaire boom is compleet
2. Inhoud: de heap-eigenschap geldt, d.w.z. in elke knoop geldt dat de waarde opgeslagen in die knoop groter dan of gelijk is aan(*) de waarde in zijn kinderen

Langs elk pad van de wortel tot een blad zijn de sleutels in de knopen dus van groot naar klein geordend.

(*) we spreken dan wel van een max-heap; een min-heap wordt analoog gedefinieerd







1. geen heap   2. geen heap   3. wel heap

1. ouder $\geq$ kinderen geldt niet in elke knoop
2. niet compleet
3. compleet <u>en</u> ouder $\geq$ kinderen

1. Gegeven n, dan bestaat er precies één **complete** binaire boom met n knopen. Deze heeft hoogte $\lfloor \lg n \rfloor$.

2. De wortel van een heap bevat altijd de grootste waarde.

3. Voor elke knoop van een heap geldt: de subboom met die knoop als wortel is weer een heap.

4. Een heap wordt gerepresenteerd door een eendimensionaal array H, met de inhoud van de n knopen op posities 1 t/m n.

- ouderknopen (interne knopen) corresponderen met de posities 1 t/m $\lfloor \frac{n}{2} \rfloor$; bladeren met $\lfloor \frac{n}{2} \rfloor + 1$ t/m n.

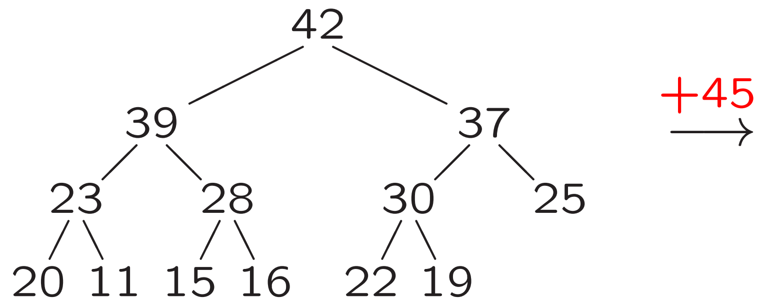
- de kinderen van $H[i]$ ($i = 1, \ldots, \lfloor \frac{n}{2} \rfloor$) zijn $H[2i]$ en $H[2i + 1]$; de ouder van $H[i]$ ($i = 2, \ldots, n$) is $H[\ \lfloor i/2 \rfloor\ ]$.

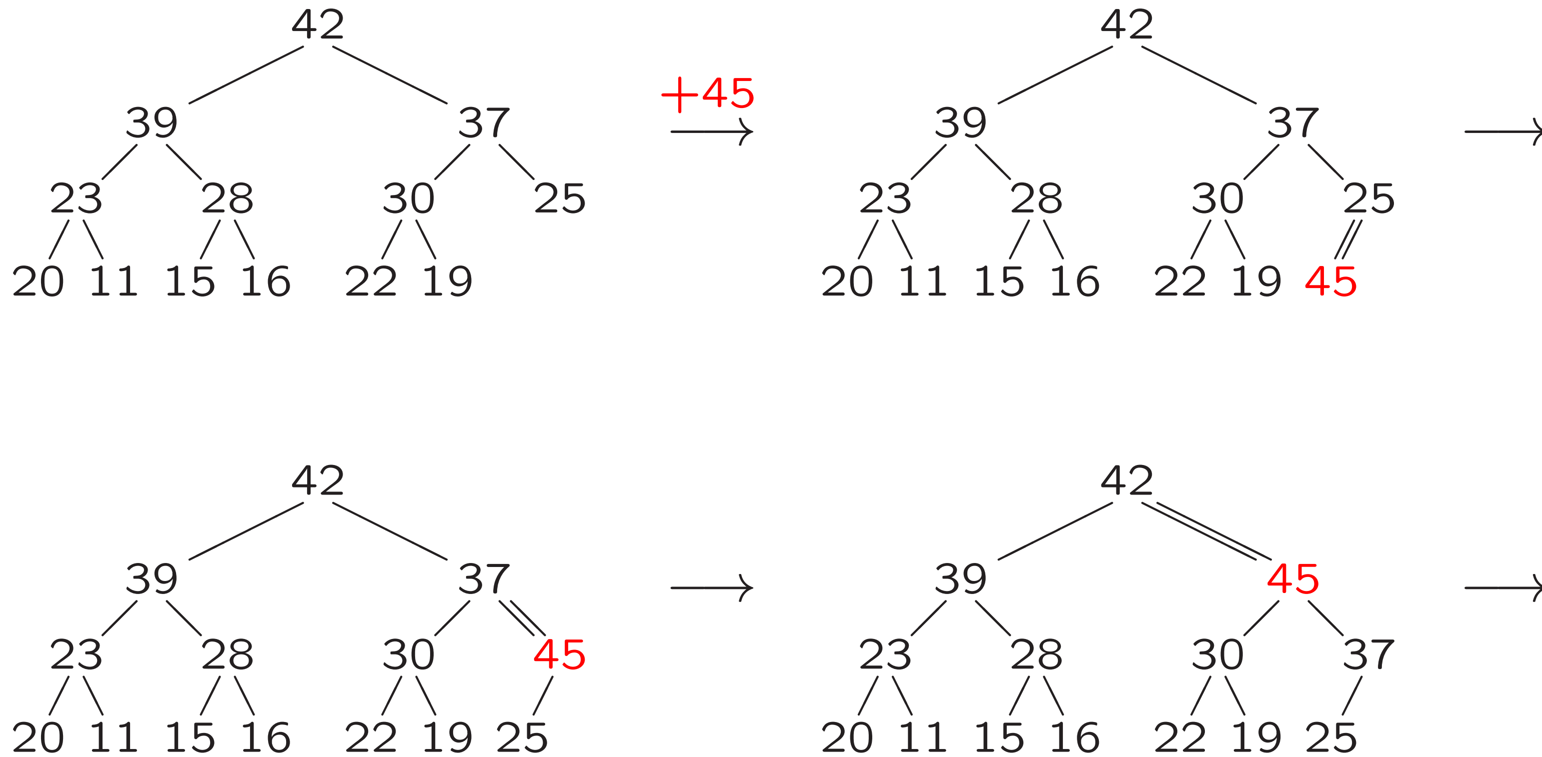
Wanneer de waarde in een knoop verandert (of een waarde wordt verwijderd/toegevoegd) zal i.h.a. de heap-eigenschap niet meer gelden. Er zijn twee manieren (beide $O(\lg n)$, met n het aantal knopen van de heap) om die weer te herstellen, afhankelijk van de situatie.

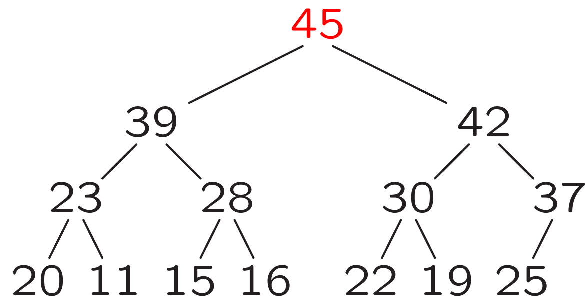
Stel nu dat de waarde in één knoop veranderd wordt. Dan zijn er twee mogelijkheden:

1. waarde in knoop $>$ waarde in ouder: herhaald verwisselen met ouder totdat de heap-eigenschap hersteld is (waarde borrelt omhoog)

2. waarde knoop $<$ waarde van (ten minste een der) kinderen: herhaald verwisselen met grootste kind totdat de heap-eigenschap hersteld is (waarde zakt omlaag)


42
39
37
23
28
30
25
20
11
15
16
22
19
+45


42
39
37
23
28
30
25
20 11 15 16 22 19
+45
42
39
37
23
28
30
25
20 11 15 16 22 19 45
42
39
37
23
28
30
45
20 11 15 16 22 19 25
42
39
45
23
28
30
37
20 11 15 16 22 19 25



Heap-eigenschap weer hersteld

42 39 37 23 28 30 25 20 11 15 16 22 19

$\Downarrow$

42 39 37 23 28 30 25 20 11 15 16 22 19 45

$\Downarrow$

42 39 37 23 28 30 45 20 11 15 16 22 19 25

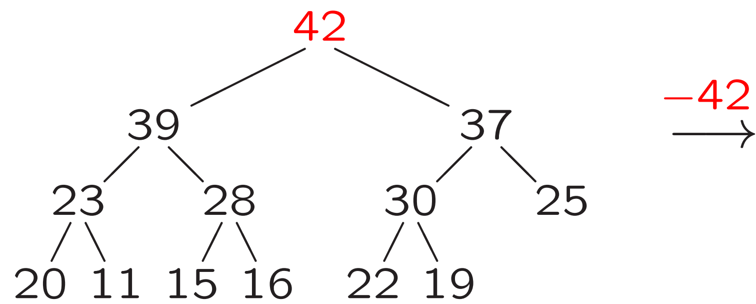
$\Downarrow$

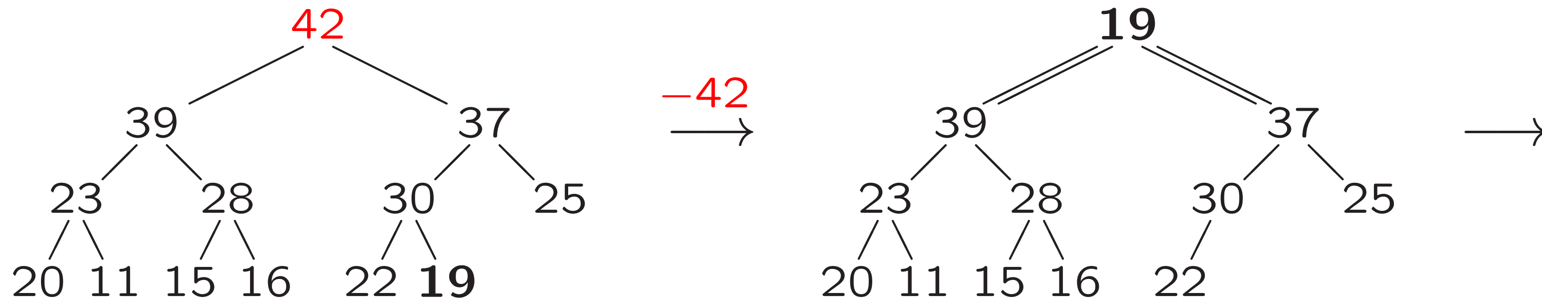
42 39 45 23 28 30 37 20 11 15 16 22 19 25

$\Downarrow$

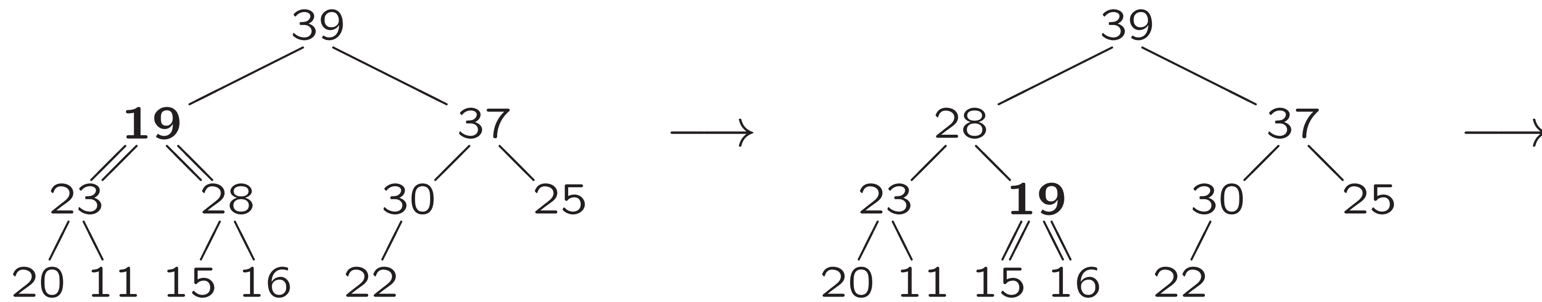
45 39 42 23 28 30 37 20 11 15 16 22 19 25

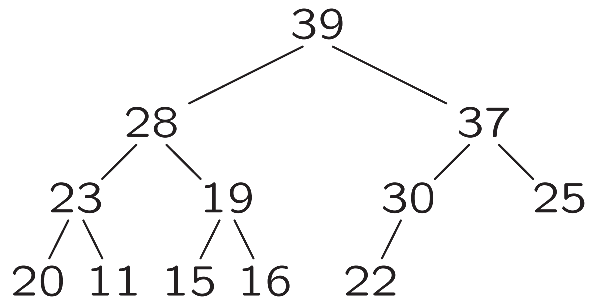
Verwijderen max. -1-





links heap, rechts heap





Heap-eigenschap weer hersteld

42 39 37 23 28 30 25 20 11 15 16 22 **19**

⇓

19 39 37 23 28 30 25 20 11 15 16 22

⇓

39 **19** 37 23 28 30 25 20 11 15 16 22

⇓

39 28 37 23 **19** 30 25 20 11 15 16 22

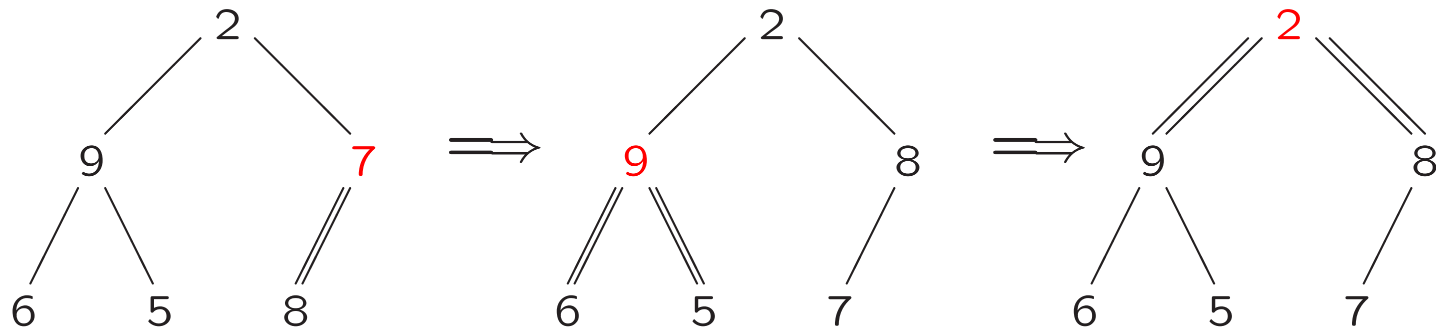
⇓

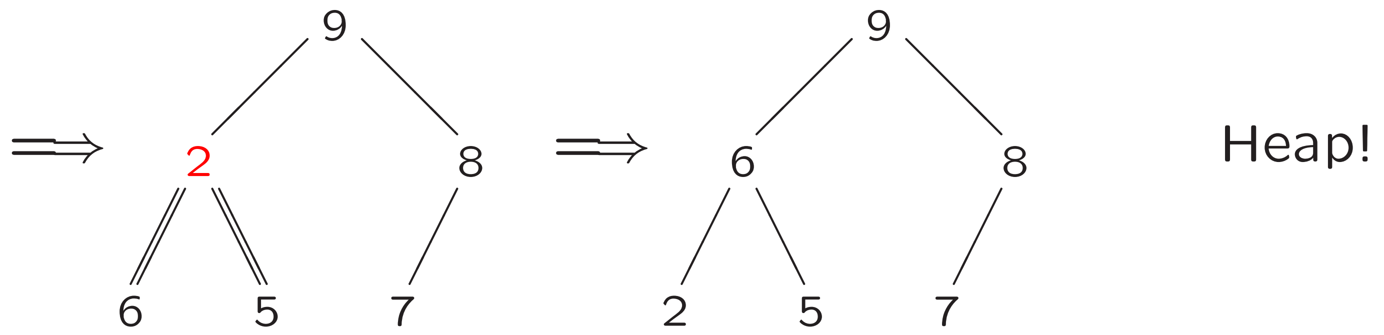
39 28 37 23 19 30 25 20 11 15 16 22

Constructie van een heap uit een gegeven rij (array H) sleutels (getallen bijv.):

- Bottom up (heapify): beginnend bij $H[\lfloor \frac{n}{2} \rfloor]$, de laatste ouderknoop, en zo teruglopend, wordt in alle subbomen met de ouderknopen als wortel de heap-eigenschap hersteld via omlaag zakken van de (inhoud van die) ouder-knoop

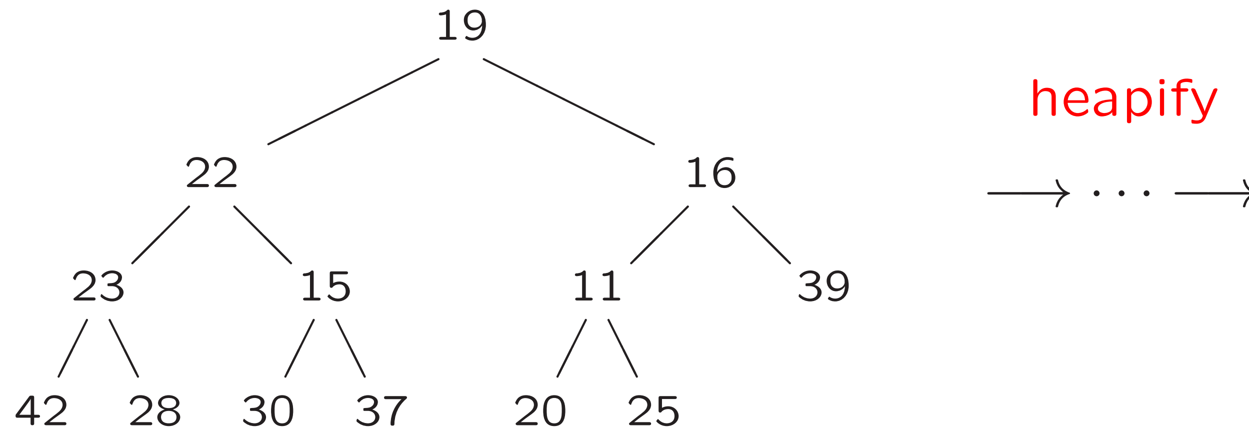
- Top down: . . .





Heap!

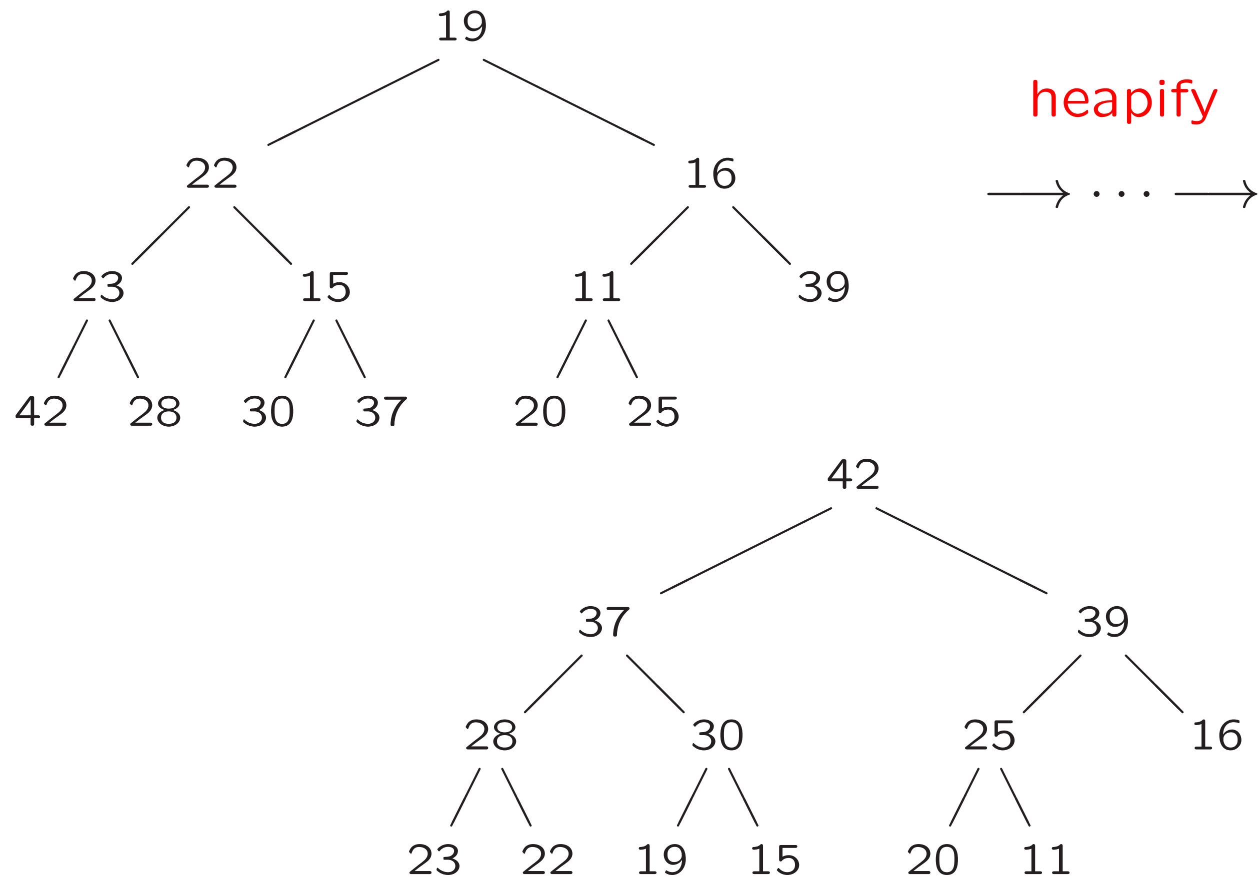
Groter voorbeeld



heapify

$\longrightarrow \cdots \longrightarrow$

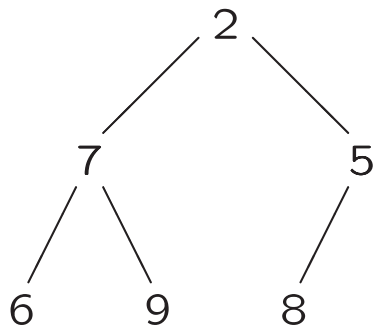
Groter voorbeeld



```
for i := ⌊ n/2 ⌋ downto 1 do
      k := i; v := H[k];
      heap := false;
      while not heap and 2 * k ≤ n do // geen blad
         j := 2 * k; // linkerkind
         if j < n then // 2 kinderen
            if H[j] < H[j + 1] then
               j := j + 1; fi // selecteer grootste kind
         fi
         if  v ≥ H[j] then
            heap := true;
         else
            H[k] := H[j]; k := j; fi
      od
      H[k] := v;
od
```

- Bottom up (heapify): . . .

- Top down: beginnend bij de wortel en door telkens een knoop meer bij de heap te betrekken wordt de heap-eigenschap steeds hersteld via omhoog borrelen van de (inhoud van de) nieuwe knoop



- Naam bottom-up – top-down

- Worst case bottom-up (heapify). . .

  Worst case complexiteit. . .

- Worst case top-down. . .

  Worst case complexiteit. . .

- Naam bottom-up – top-down

- Worst case bottom-up (heapify)...

  Worst case complexiteit: $O(n)$

  `http://oeis.org/A000295`

- Worst case top-down...

  Worst case complexiteit: $O(n \lg n)$

  `http://oeis.org/A036799`

1. Maak een heap van het gegeven array
2. Verwijder nu herhaald ($n-1$ keer) de grootste waarde uit de wortel:
   - verwissel deze met de laatste waarde uit de heap
   - verlaag de grootte van de heap met 1
   - zorg dat overal de heap-eigenschap weer geldt door de nieuwe waarde uit de wortel te laten zakken

Het array wordt zo oplopend gesorteerd.
Complexiteit: $O(n \lg n)$.

Sorteer het array 3 9 7 5 4 8 met heapsort.

Fase 1

2 9 **7** 6 5 8 $\longrightarrow$
2 **9** 8 6 5 7 $\longrightarrow$
2 9 8 6 5 7 $\longrightarrow$
9 **2** 8 6 5 7 $\longrightarrow$
9 6 8 2 5 7

Fase 2

9 5 8 3 4 7 $\longrightarrow$
7 5 8 3 4 |9 $\longrightarrow$
8 5 7 3 4 |9 $\longrightarrow$
4 5 7 3 |8 9 $\longrightarrow$
7 5 4 3 |8 9 $\longrightarrow$
3 5 4 |7 8 9 $\longrightarrow$
5 3 4 |7 8 9 $\longrightarrow$
4 3 |5 7 8 9 $\longrightarrow$
3 |4 5 7 8 9 $\longrightarrow$
|3 4 5 7 8 9

- **Lezen/leren bij dit college:**
  paragraaf 9.3, paragraaf 6.4, slides

- **Werkcollege:**
  vanmiddag, 13.45–15.30, in computerzaal:
  derde programmeeropdracht (dynamisch programmeren)

- **Volgend (laatste!) college:**
  11/12 mei 2017

- **Daarna:**
  18/19 mei 2017: tentamen oefenen