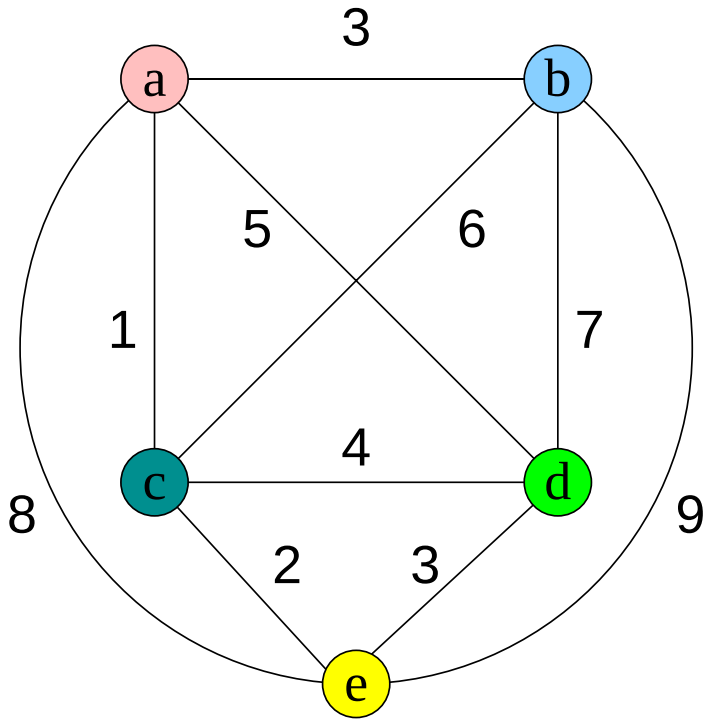
Twaalfde college algoritmiek

23 mei 2013

Branch & Bound, Heapsort

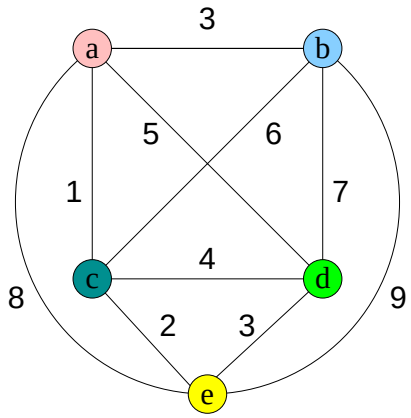
Traveling Salesman Problem (handelsreizigersprobleem)

Gegeven n steden waarvan alle onderlinge afstanden bekend zijn. **Gevraagd**: de/een kortste route die elke stad precies één keer aandoet, en weer terugkeert in het vertrekpunt.
Ofwel: vind de/een kortste Hamiltonkring in een samenhangende gewogen (complete) graaf. Het gaat hier om een ongerichte graaf.



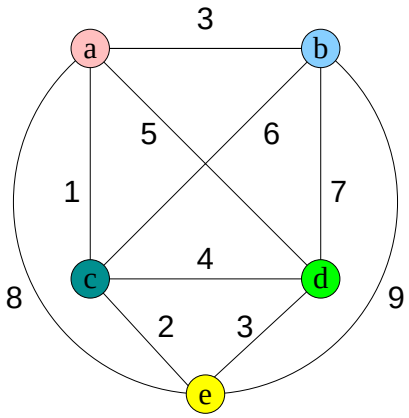
De optimale oplossing heeft lengte 16: a, b, d, e, c, a

Mogelijke ondergrenzen voor de kosten van een optimale oplossing. . .



Mogelijke ondergrenzen voor de kosten van een optimale oplossing:

- eenvoudig: $n\times$ kortste afstand tussen twee knopen; $5 \times 1 = 5$ bij aanvang (analoog als reeds takken gekozen zijn).
- iets beter: de lengtes van de n kortste takken gesommeerd; $1+2+3+3+4 = 13$ bij aanvang (analoog als reeds takken gekozen zijn).

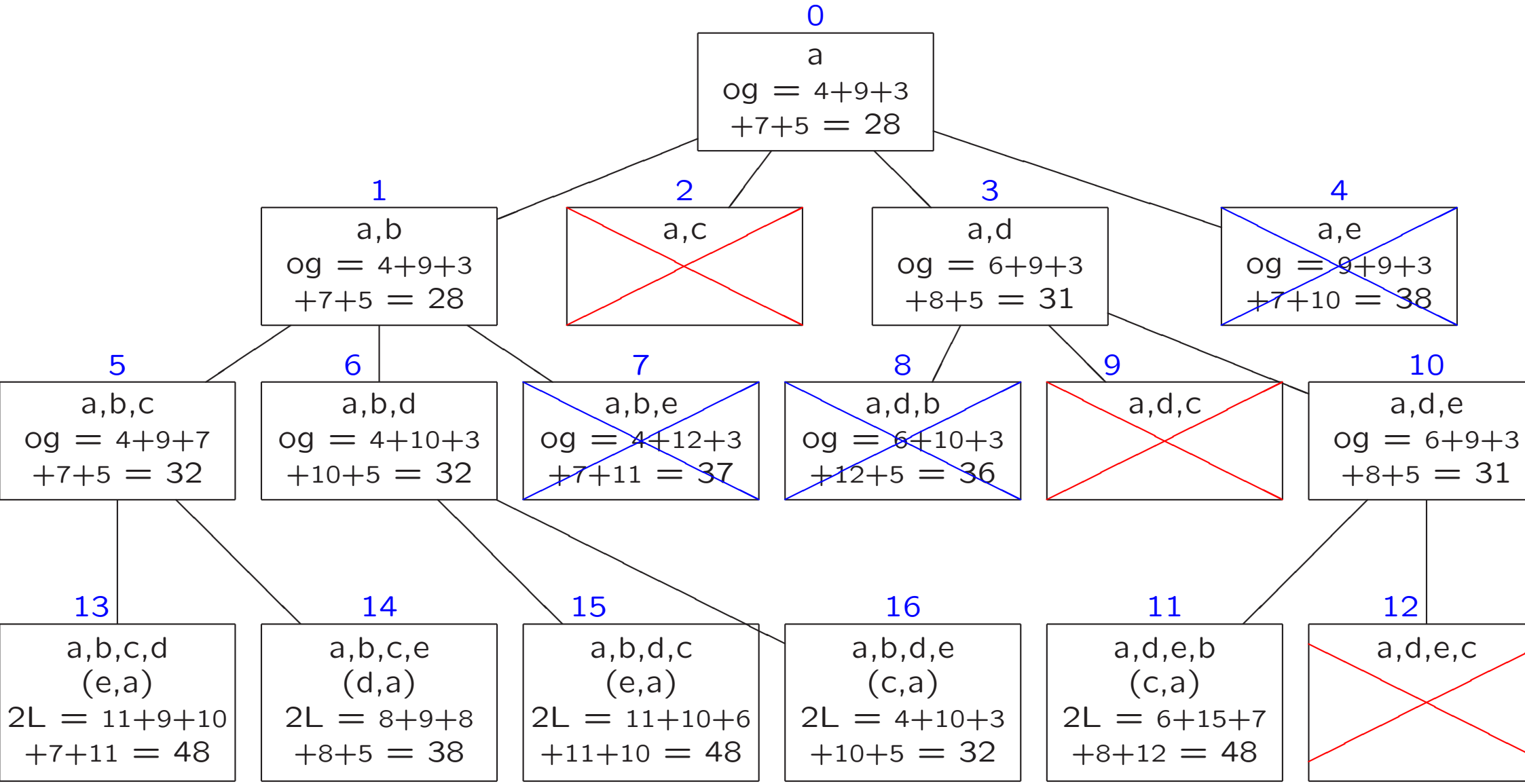


Mogelijke ondergrenzen voor de kosten van een optimale oplossing:

- nog beter: som over alle knopen i van de afstanden van knoop i tot de twee dichtstbijzijnde knopen (inclusief al gekozen takken), en dat gedeeld door 2(*); $((1+3)+(3+6)+(1+2)+(3+4)+(2+3))/2 = 14$ bij aanvang.

Opmerking: delen door 2 is niet nodig; je kunt ook 2×lengte minimaliseren (*).

B&B state space tree



Zonder beperking der algemeenheid kunnen we het volgende doen om werk uit te sparen:

- we bekijken alleen Hamiltonkringen die met a beginnen: immers, de kring c, d, e, a, b, c is hetzelfde als de kring a, b, c, d, e, a (en als d, e, a, b, c, d en e, a, b, c, d, e en als b, c, d, e, a, b). Je hoeft er hiervan maar een te bekijken.

- we bekijken alleen kringen waarbij b wordt bezocht voor c: immers, de kring a, c, d, b, e, a is dezelfde kring als a, e, b, d, c, a maar dan omgekeerd doorlopen. Beide hebben dezelfde lengte (want deze graaf is ongericht). We hoeven dus maar een van deze twee te bekijken.

In de state space tree van de vorige sheet is dus een deeloplossing beginnend met a, c *ontoelaatbaar* (infeasible) omdat hierin na uitbreiding c voor b komt. Deze knoop hoeft dus ook niet verder bekeken te worden; bijbehorende oplossingen komen we elders wel tegen. Bijv.: a, c, e, b, d, a vinden we terug als a, d, b, e, c, a, dus via knoop a, d.

We zeiden wel:

> delen door 2 is niet nodig; je kunt ook 2×lengte minimaliseren (*)

Delen door 2 geeft echter wel scherpere ondergrens, als je resultaat afrondt naar boven.

In vorige voorbeeld wordt ondergrens 31 van knoop a,d hiermee 16. Deze knoop hoef je nu niet uit te werken (zie boom in boek).

Branch & bound

- is alleen toepasbaar op optimalisatieproblemen
- genereert oplossingen stap voor stap en houdt de tot dusver gevonden beste oplossing bij
- gebruikt voor elke deeloplossing (= knoop in de state space tree) een of andere ondergrens (minimalisatie-probleem) resp. bovengrens (maximalisatieprobleem) op de te verwachten waarde van de objectfunctie, met als doel
  - van deeloplossingen te kunnen bepalen dat ze niet verder bekeken hoeven te worden: snoeien (*)
  - de zoekvolgorde in de zoekruimte (state space tree), dus de volgorde waarin knopen worden gegenereerd en verder bekeken, te leiden (**)

(*) zou bij backtracking ook kunnen, maar (**) niet of nauwelijks

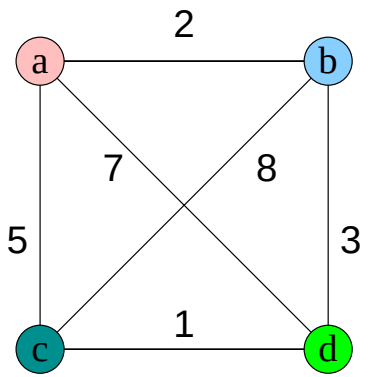
Een branch and bound algoritme breidt een knoop (deeloplossing) niet verder uit als

- de waarde van de ondergrens (bovengrens) bij die knoop niet beter is dan de waarde van de tot dusver gevonden beste oplossing: als ondergrens $\geq$ tot dusver gevonden minimale waarde, dan snoeien
- de deeloplossing niet meer voldoet aan de restricties (of niet meer uit te breiden is tot een toelaatbare oplossing)
- er nog maar één volledige, toelaatbare oplossing mogelijk is bij de deeloplossing (i.h.b. als deeloplossing zo'n volledige oplossing *is*); de waarde van deze ene oplossing wordt met beste oplossing tot nu toe vergeleken; update zonodig beste oplossing.

De volgorde waarin de knopen (deeloplossingen) worden uitgebreid hangt direct af van de berekende grenzen:

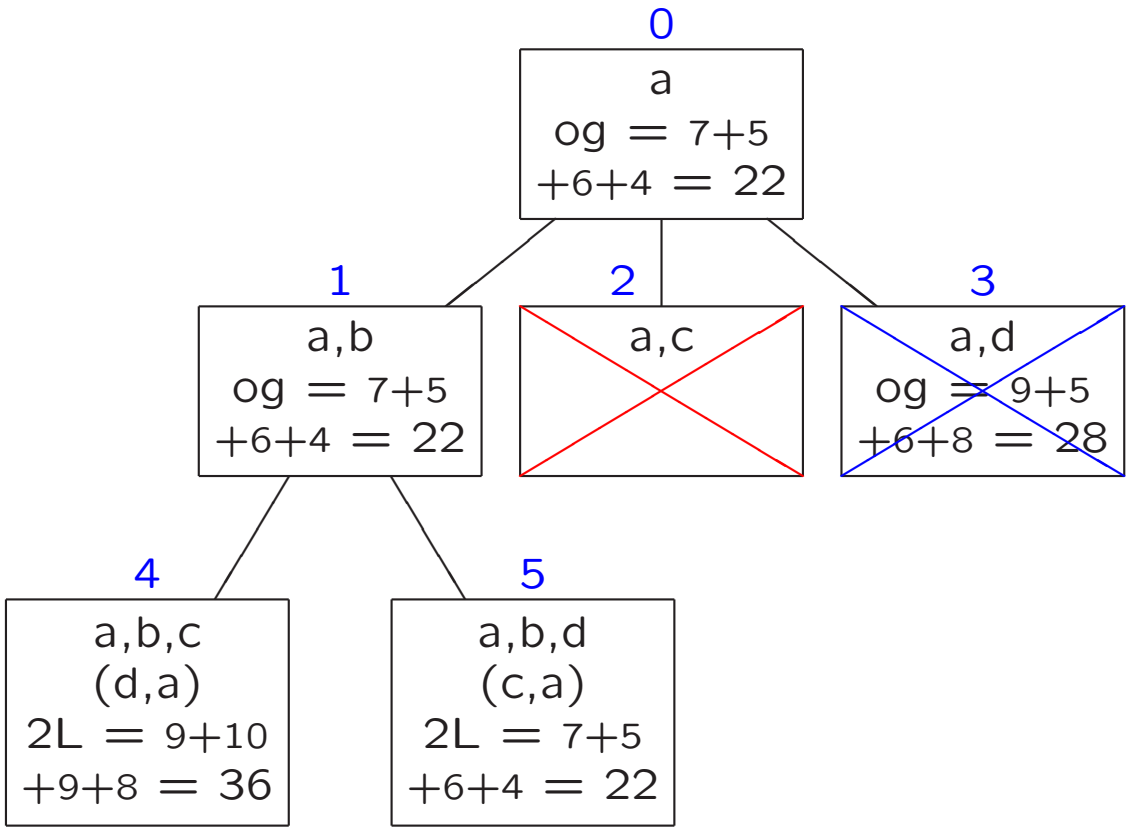
- er worden meerdere deeloplossingen tegelijk bijgehouden, dit in tegenstelling tot backtracking
- in elke stap wordt een van al deze deeloplossingen gekozen, en daarvan worden alle 1-staps-uitbreidingen (kinderen in de state space tree) bekeken en geëvalueerd (d.w.z. ondergrens/bovengrens bepaald)
- zinloze uitbreidingen worden meteen verworpen
- meestal wordt de deeloplossing gekozen die het meest veelbelovend lijkt: best-fit-first branch-and-bound
- bij minimalisatieproblemen (resp. maximalisatieproblemen) kiezen we de knoop met de laagste ondergrens (resp. hoogste bovengrens) als eerste

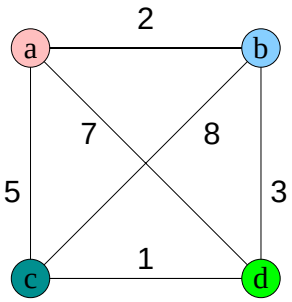
Nog een voorbeeld, met als ondergrens wederom de som van de twee kortste takken per knoop (eventueel gedeeld door 2, zoals in het boek).



De optimale oplossing heeft lengte 11: a, b, d, c, a.

Exercise 12.2.9, vervolg





Gegeven n objecten, met gewicht $w_1, \ldots, w_n$ en waarde $v_1, \ldots, v_n$, en een knapzak met capaciteit W. **Gevraagd**: de meest waardevolle deelverzameling der objecten die in de knapzak past (dus met totaalgewicht $\leq W$).

Voorbeeld:

item	w	v	v/w
1	4	\$40	10
2	7	\$42	6
3	5	\$25	5
4	3	\$12	4

Maximaal gewicht $W = 10$

item	w	v	v/w
1	4	\$40	10
2	7	\$42	6
3	5	\$25	5
4	3	\$12	4

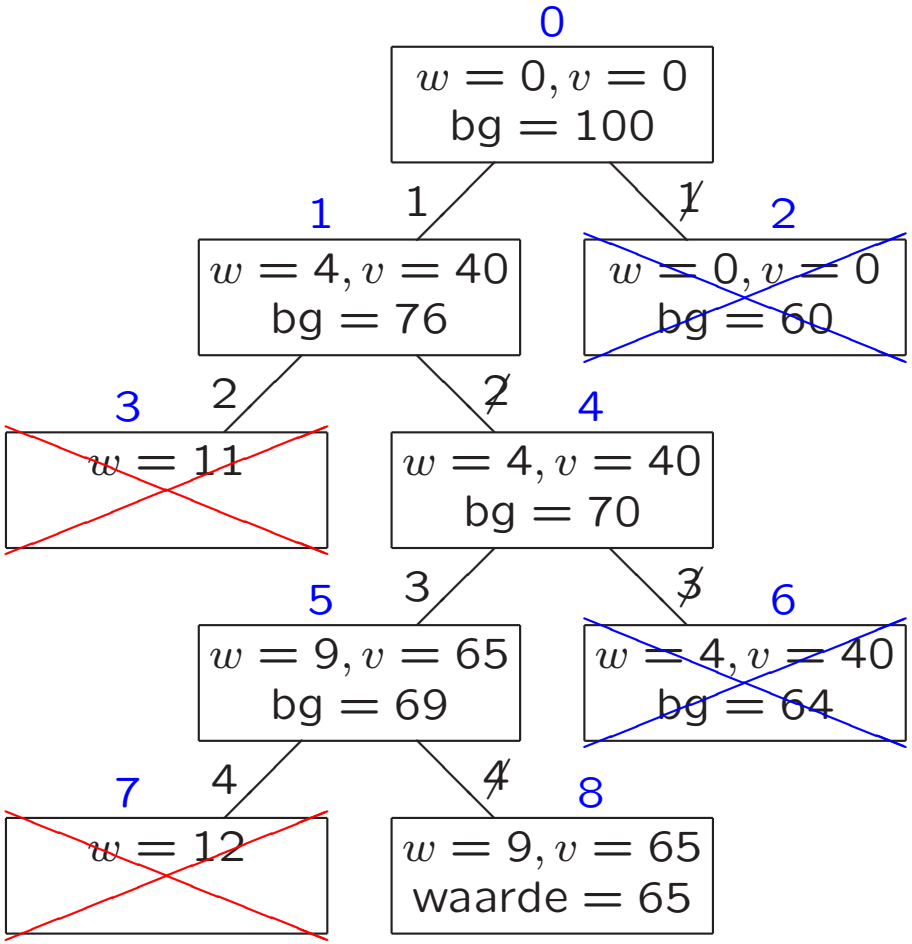
Opbouwen van de oplossing: in de i-de stap wordt object i wel of niet gekozen,

Een bovengrens voor de waarde van een optimale oplossing:

- Bij aanvang: $W * (v_1/w_1) = 100$;
- Na de i-de stap: $v + (W - w) * (v_{i+1}/w_{i+1})$, met v de totaalwaarde van de reeds gekozen objecten en w het totaalgewicht daarvan.

De optimale oplossing heeft gewicht 9 en waarde 65: $\{1, 3\}$.

B&B state space tree

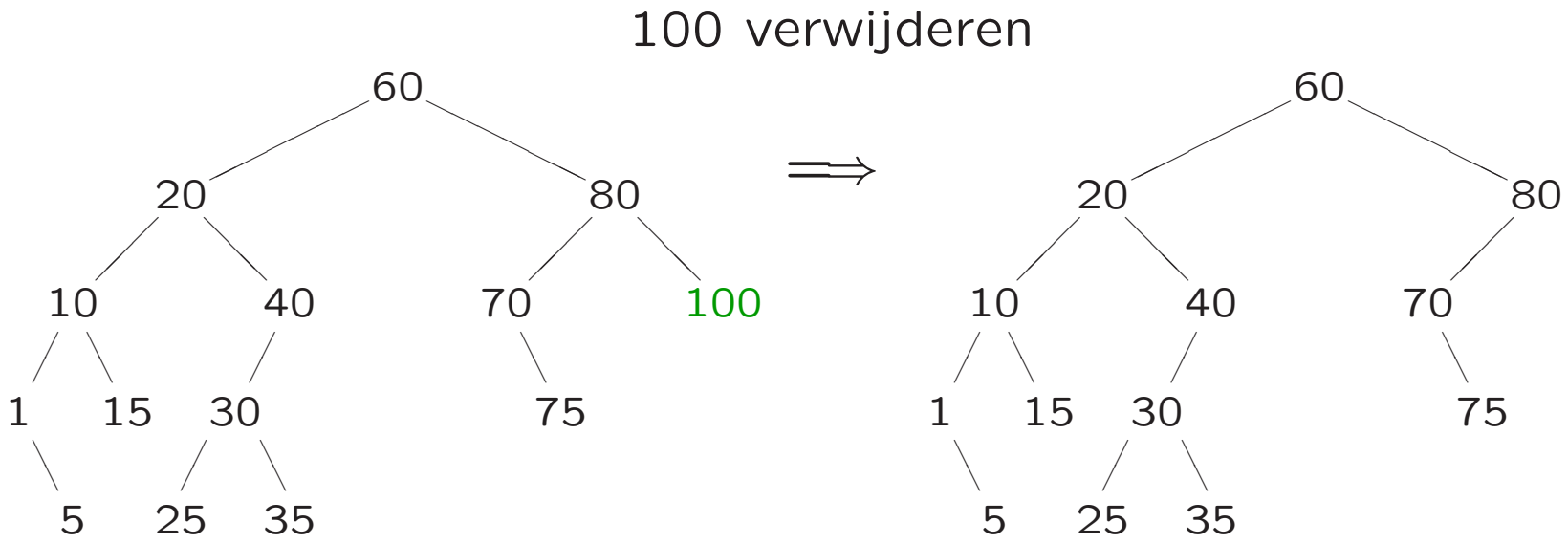


item	w	v	v/w
1	4	\$40	10
2	7	\$42	6
3	5	\$25	5
4	3	\$12	4

$W = 10$

Zie ook exercise 12.2.5, Levitin.

BZboom: verwijderen



verwijderen van een blad

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

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

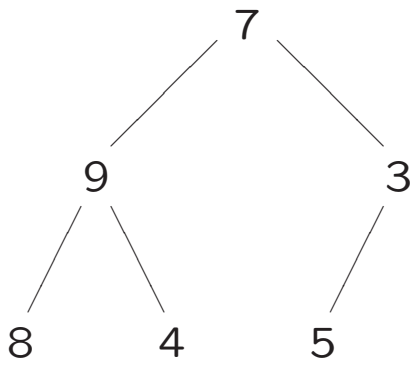
```
knoop* wortel; // de ingang tot de binaire boom
```

Netter om een klasse te gebruiken: zie Programmeermethoden

Complete binaire boom

- 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:



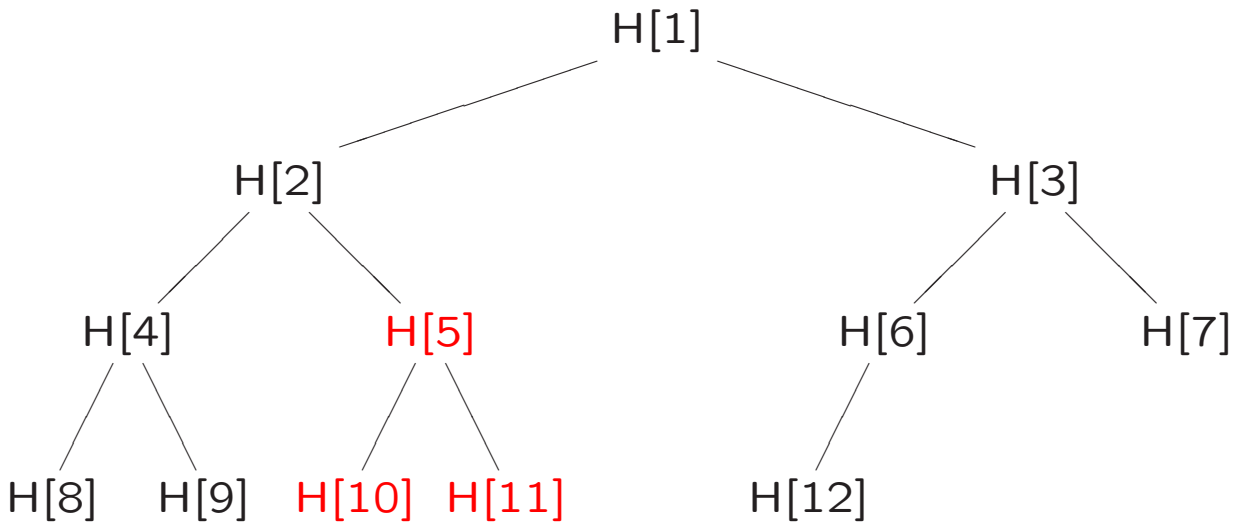
$\Longleftrightarrow$

7 9 3 8 4 5

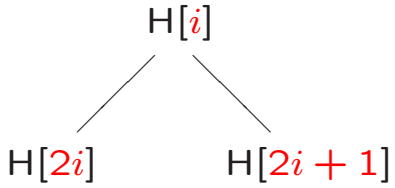
representatie als array

complete binaire boom

Boom $\leftrightarrow$ array



$\Updownarrow$ 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]

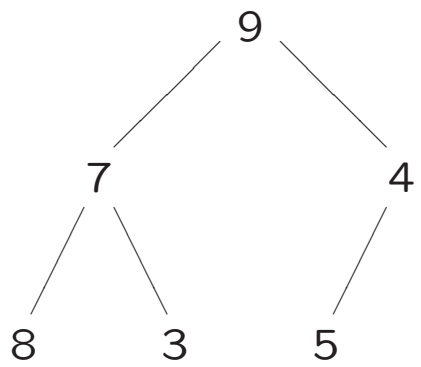
Definitie
Een heap (hoopstructuur) is een binaire boom met de volgende twee eigenschappen:

1. Vorm: de binaire boom is compleet

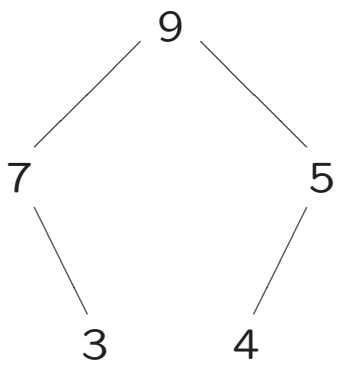
2. Inhoud: de heap-eigenschap geldt, d.w.z. in elke knoop geldt dat de waarde opgeslagen in die knoop groter dan of gelijk is aan(*) de waarde in zijn kinderen

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

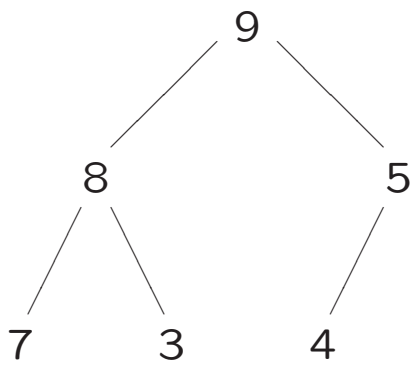
(*) we spreken dan wel van een max-heap; een min-heap wordt analoog gedefinieerd



1. geen heap



2. geen heap



3. wel heap

1. ouder $\geq$ kinderen geldt niet in elke knoop

2. niet compleet

3. compleet en 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 $H[\ \lfloor i/2 \rfloor\ ]$.

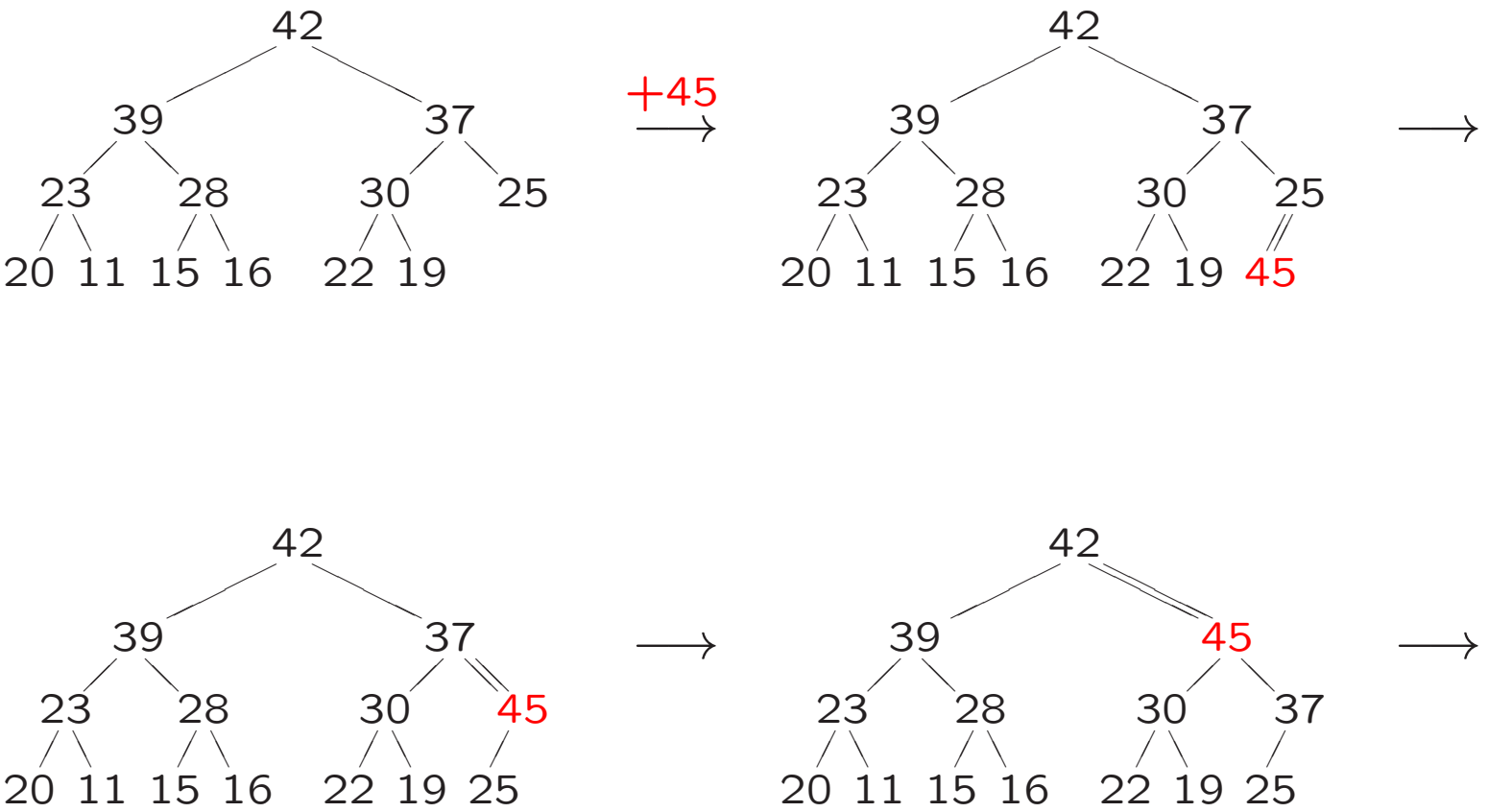
Wanneer de waarde in een knoop verandert (of een waarde wordt verwijderd/toegevoegd) zal i.h.a. de heap-eigenschap niet meer gelden. Er zijn twee manieren (beide $O(\lg n)$, met n het aantal knopen van de heap) om die weer te herstellen, afhankelijk van de situatie.

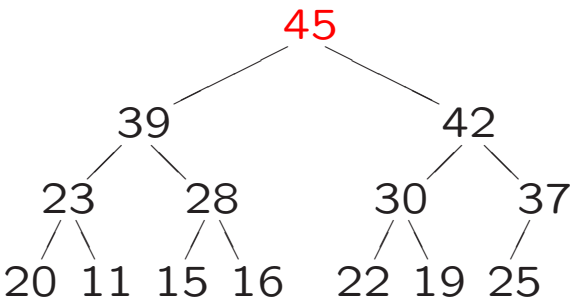
Stel nu dat de waarde in één knoop veranderd wordt. Dan zijn er twee mogelijkheden:

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)

Toevoegen -1-


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

⇓

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

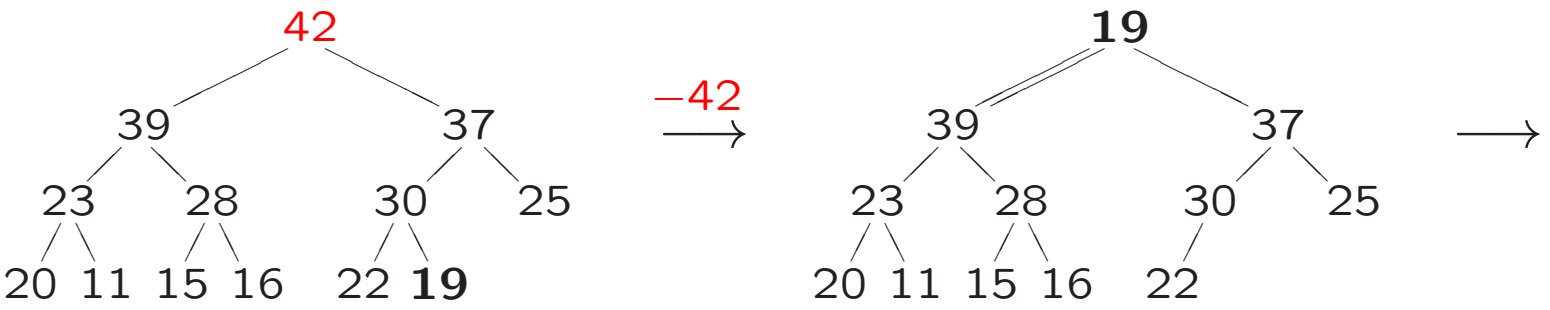
⇓

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

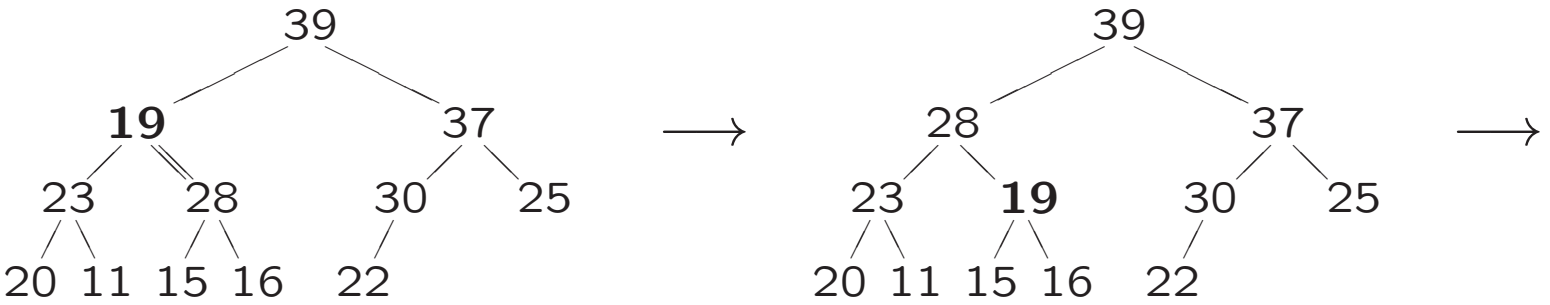
⇓

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

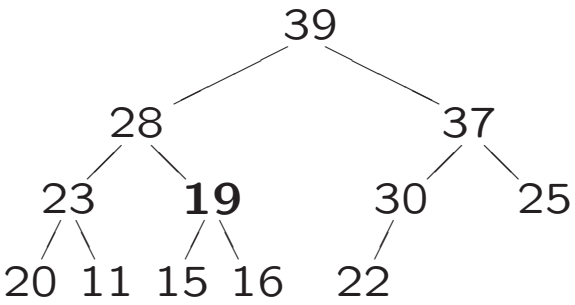
Verwijderen maximum -1-



links heap, rechts heap



Verwijderen maximum -2-



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

⇓

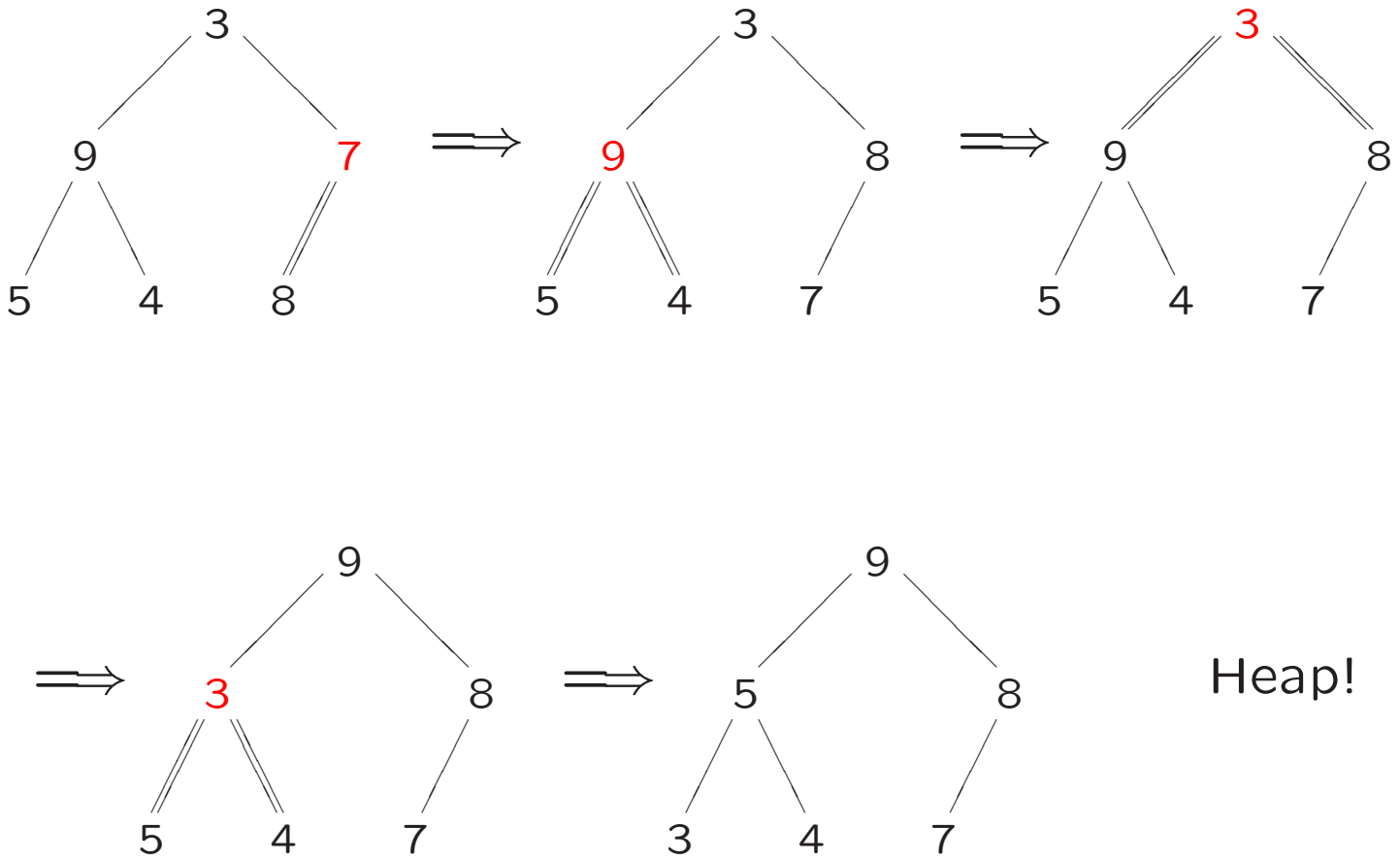
39 28 37 23 **19** 30 25 20 11 15 16 22

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

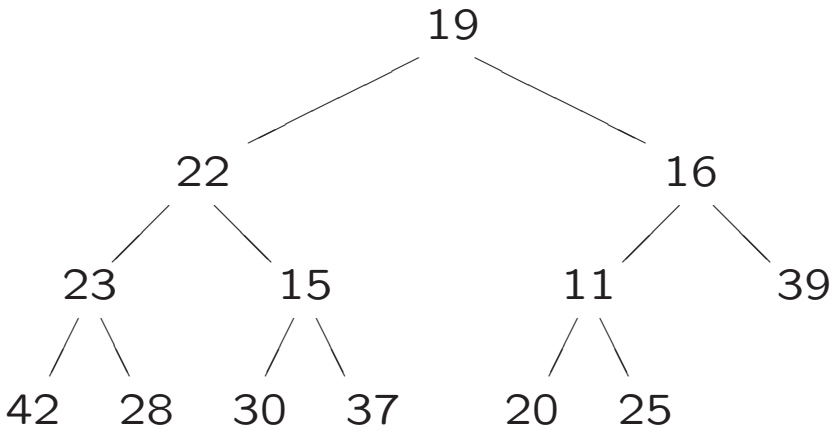
- Bottom up (heapify): beginnend bij $H[\lfloor \frac{n}{2} \rfloor]$, de laatste ouderknoop, en zo teruglopend, wordt in alle subbomen met de ouderknopen als wortel de heap-eigenschap hersteld via omlaag zakken van de (inhoud van die) ouderknoop

- 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

- Heapify is $O(n)$; de andere methode is $O(n \lg n)$

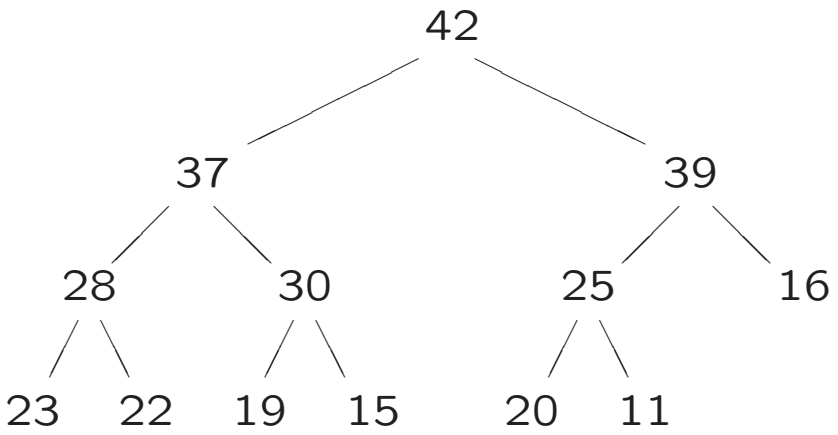

3
9
7
5
4
8
3
9
8
5
4
7
3
9
8
5
4
7
9
3
8
5
4
7
9
5
8
3
4
7
Heap!

Groter voorbeeld



heapify

$\longrightarrow \cdots \longrightarrow$



```
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
```

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

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

- **Lezen/leren bij dit college:** Paragraaf 12.2, paragraaf 6.4
- **Werkcollege:**
  donderdag 23 mei 2013, 13:45–15:30, in zaal Noordeinde
- **Laatste college:**
  donderdag 30 mei 2013, zaal Noordeinde
  11:15–13:00: oefenen oud tentamen
  13:45–15:30: bespreken oud tentamen
- **Opgaven/oude tentamens:**
  http://www.liacs.nl/home/rvvliet/algoritmiek/
- **Vragenuur:**
  donderdag 6 juni 2013, 13:45–15:30 uur, zaal Noordeinde
- **Tentamen:**
  dinsdag 11 juni 2013, 10:00–13:00 uur, zaal Bezuidenhout
- Borrel
  vanmiddag, 15:30 –