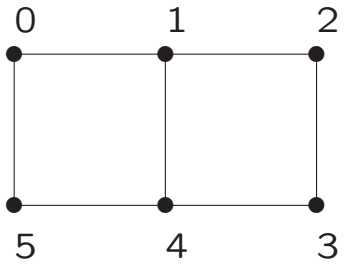
Algoritmiek

12 februari 2025

Grafen en bomen

- vrijdag 14 februari geen werkcollege

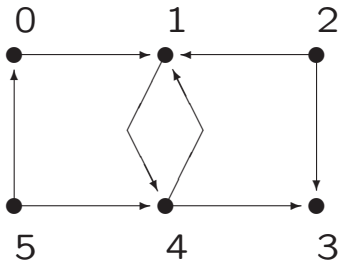
- bomenpracticum, dinsdag 18 februari, 09.00-10.45 in computerzalen DM.0.09, DM.0.13, DM.0.21



1.

$V = \{0, 1, 2, 3, 4, 5\}$;

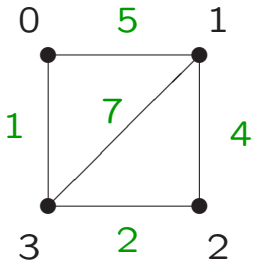
$E = \{(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 0), (4, 1)\}$



2.

$V = \{0, 1, 2, 3, 4, 5\}$;

$E = \{(0, 1), (2, 1), (2, 3), (4, 3), (5, 4), (5, 0), (4, 1), (1, 4)\}$



3.

$V = \{0, 1, 2, 3\}$;

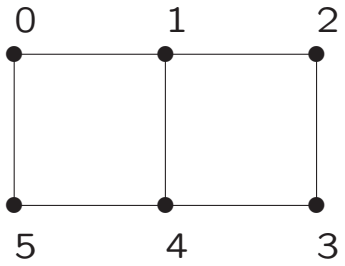
$E = \{(0, 1), (1, 2), (2, 3), (0, 3), (1, 3)\}$

Een pad van u naar v is een rij knopen waarvoor geldt dat tussen elk tweetal opeenvolgende knopen uit die rij een tak zit (resp. een pijl loopt van de ene naar de volgende knoop; dan: gericht pad).
Lengte van een pad = aantal takken op dat pad = aantal knopen - 1.
Als alle knopen verschillend zijn heet het pad **enkelvoudig**.

Een kring in een ongerichte graaf is een pad met minstens 3 knopen, dat begint en eindigt in dezelfde knoop en dat geen enkele tak meer dan één keer bevat. Analoog gerichte graaf.

Een ongerichte graaf heet samenhangend als er tussen elk tweetal knopen u en v een pad loopt.

Een graaf die geen kringen bevat heet acyclisch.



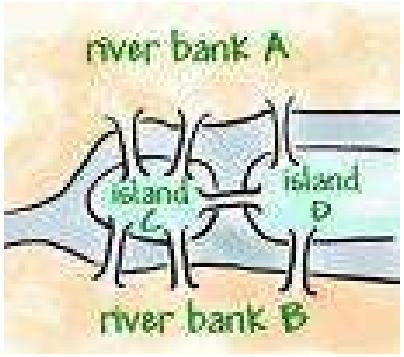
1.

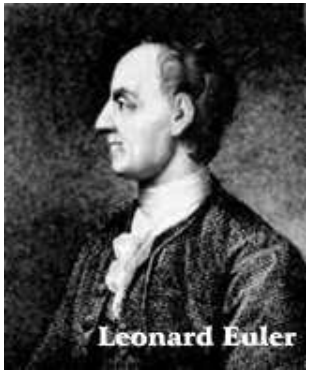
$V = \{0, 1, 2, 3, 4, 5\}$;

$E = \{(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 0), (4, 1)\}$

Een zeer bekend graafprobleem:

Koningsberger bruggen probleem



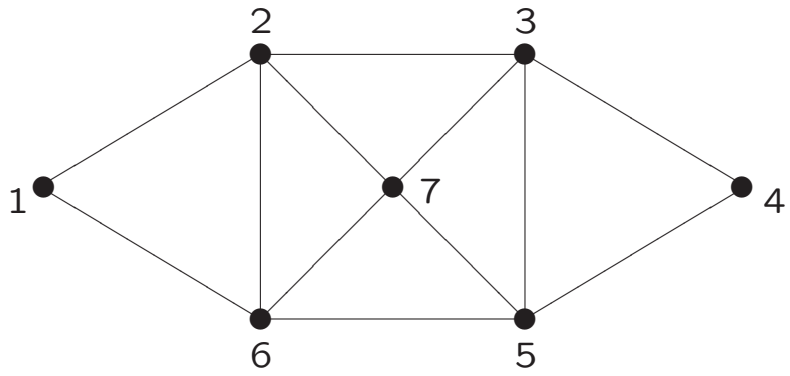


Hoezo **graaf**probleem?

Definitie: een wandeling (pad) in een ongerichte graaf die terugkeert in zijn beginpunt en die *alle* takken precies één keer doorloopt heet een Eulerkring*. Analoog: Eulerpad.

Probleem. Gegeven een ongerichte graaf $\mathcal{G}$. Heeft $\mathcal{G}$ een Eulerkring?

Voorbeeld:

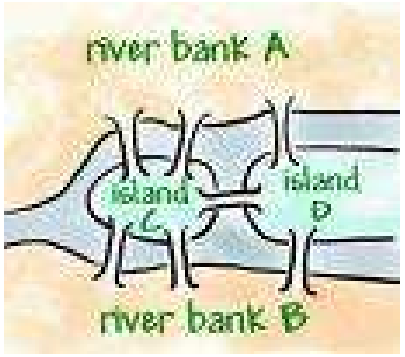


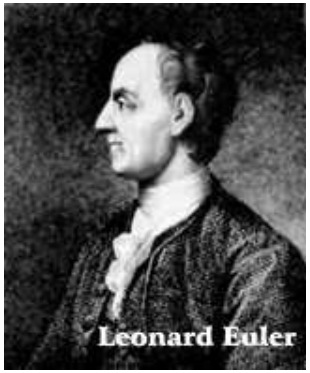
Voor deze graaf is 1 2 3 4 5 3 7 5 6 7 2 6 1 een Eulerkring.

* Een kring is een pad met minstens 3 knopen dat begint en eindigt in dezelfde knoop en dat geen enkele tak meer dan één keer doorloopt.

Nog een zeer bekend graafprobleem:

Koningsberger bruggen probleem

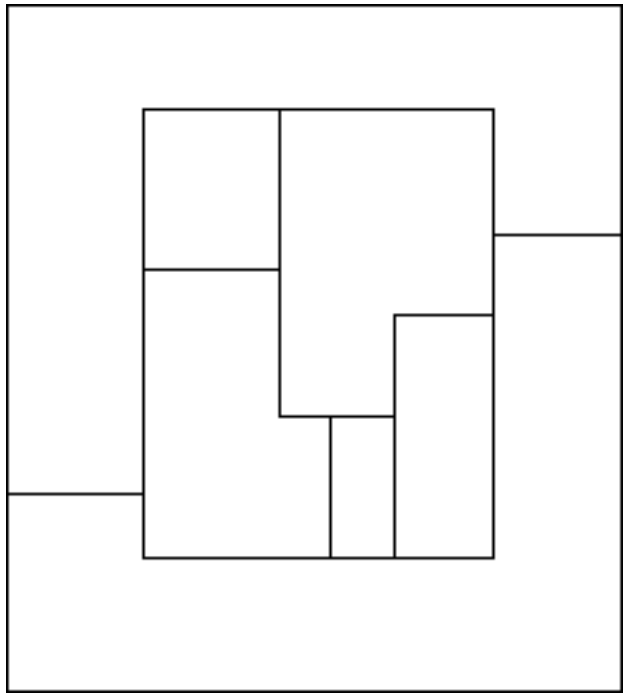




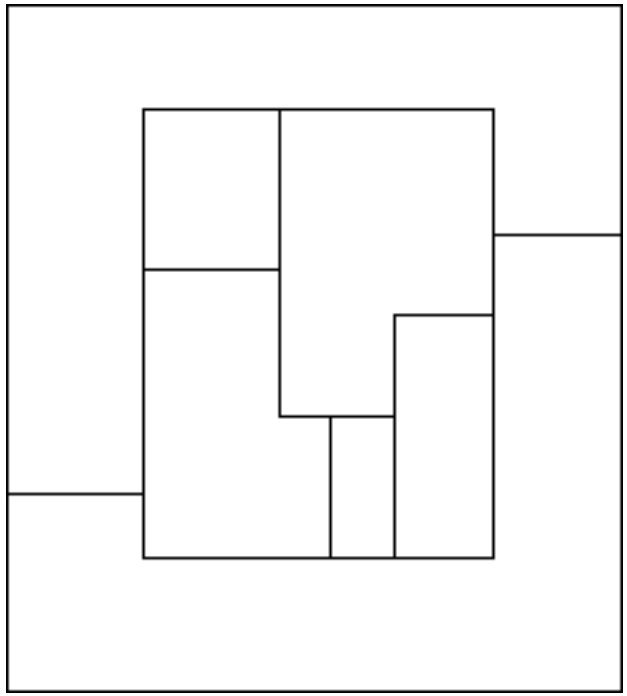
Vierkleurenprobleem

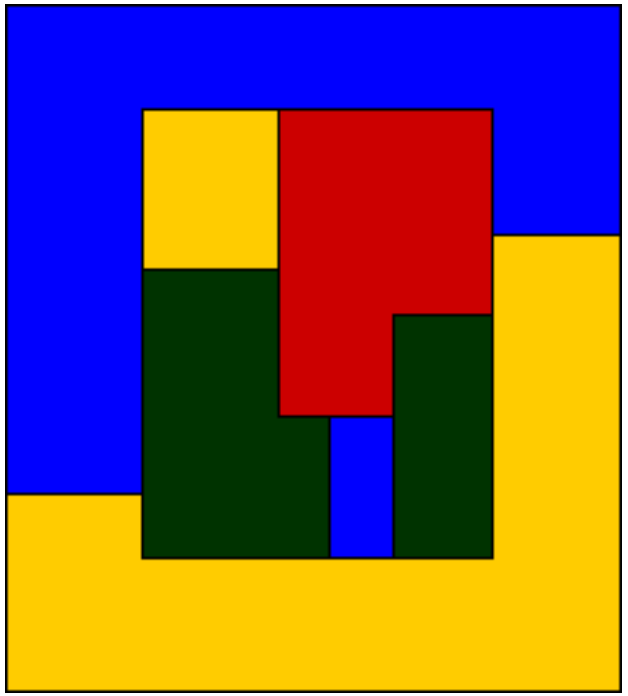
Hoezo **graaf**probleem?

Kleur de landkaart met maximaal vier kleuren onder de restrictie dat buurlanden een verschillende kleur hebben:



Kleuring van de landkaart met maximaal vier kleuren onder de restrictie dat buurlanden een verschillende kleur hebben:





Adjacency matrix: de **gerichte** graaf wordt gerepresenteerd door een tweedimensionaal array `int graaf[n][n]` (n het aantal knopen), waarbij `graaf`$[i][j]$ aangeeft of er een pijl van i naar j gaat.

Adjacency list: de **gerichte** graaf wordt gerepresenteerd door een eendimensionaal array `buur* graaf[n]` (n het aantal knopen), waarbij `graaf`$[i]$ de ingang is tot een lijst van knopen waarvoor er een pijl is van i naar die knoop. De buurlijst bevat dus alle uitgaande pijlen uit i.

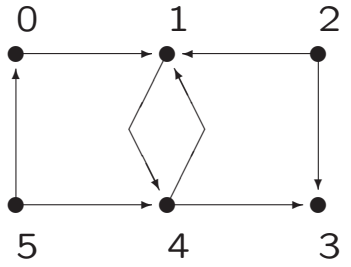
Adjacency list representatie in C++:

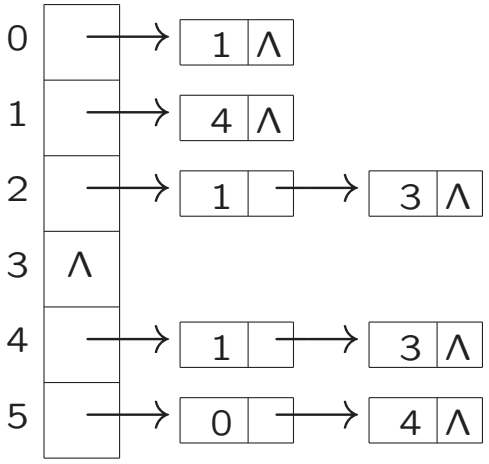
```
class buur
{ public:
    int knoopnummer;
    buur* volgende;
};  // buur
buur* graaf[n];
```

Adjacency matrix en adjacency list voor voorbeeldgraaf **2**:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

adjacency matrix





adjacency list

Geef een algoritme dat de pijl (i, j) in een gegeven gerichte graaf omdraait.

```
// het array graaf (buur* graaf[n]) bevat de gerichte graaf
```

Geef een algoritme dat de pijl (i, j) in een gegeven gerichte graaf omdraait.

```
// het array graaf (buur* graaf[n]) bevat de gerichte graaf

buur* hulp = graaf[i];
buur* vorige = nullptr;
while ( hulp->knoopnummer != j ) {       // knoop j zoeken
    vorige = hulp;
    hulp = hulp->volgende;
}
```

Geef een algoritme dat de pijl (i, j) in een gegeven gerichte graaf omdraait.

```
// het array graaf (buur* graaf[n]) bevat de gerichte graaf

buur* hulp = graaf[i];
buur* vorige = nullptr;
while ( hulp->knoopnummer != j ) {        // knoop j zoeken
    vorige = hulp;
    hulp = hulp->volgende;
}

if ( vorige == nullptr ) {                // j is eerste buur van i
    graaf[i] = hulp->volgende;
}
else {                                    // j is niet eerste buur van i
    vorige->volgende = hulp->volgende;    // haal buur j uit lijst
}
delete hulp;                              // gooi buur j weg
```

Geef een algoritme dat de pijl (i, j) in een gegeven gerichte graaf omdraait.

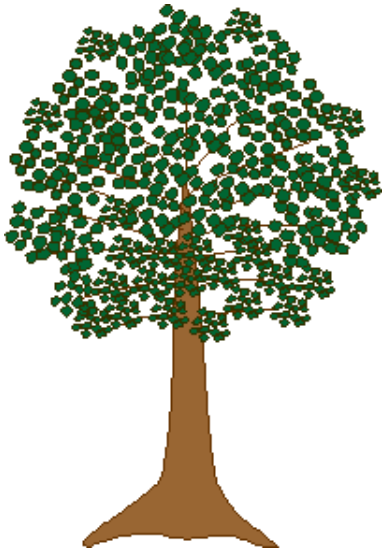
```
// het array graaf (buur* graaf[n]) bevat de gerichte graaf

buur* hulp = graaf[i];
buur* vorige = nullptr;
while ( hulp->knoopnummer != j ) {        // knoop j zoeken
    vorige = hulp;
    hulp = hulp->volgende;
}

if ( vorige == nullptr ) {                // j is eerste buur van i
    graaf[i] = hulp->volgende;
}
else {                                    // j is niet eerste buur van i
    vorige->volgende = hulp->volgende;    // haal buur j uit lijst
}
delete hulp;                              // gooi buur j weg

hulp = new buur;                          // voeg nu i vooraan in de buurlijst
hulp->knoopnummer = i;                    //  van j toe
hulp->volgende = graaf[j];
graaf[j] = hulp;
```

Definitie: een boom is een samenhangende (= uit één stuk bestaande) ongerichte graaf zonder cykels (= kringen).

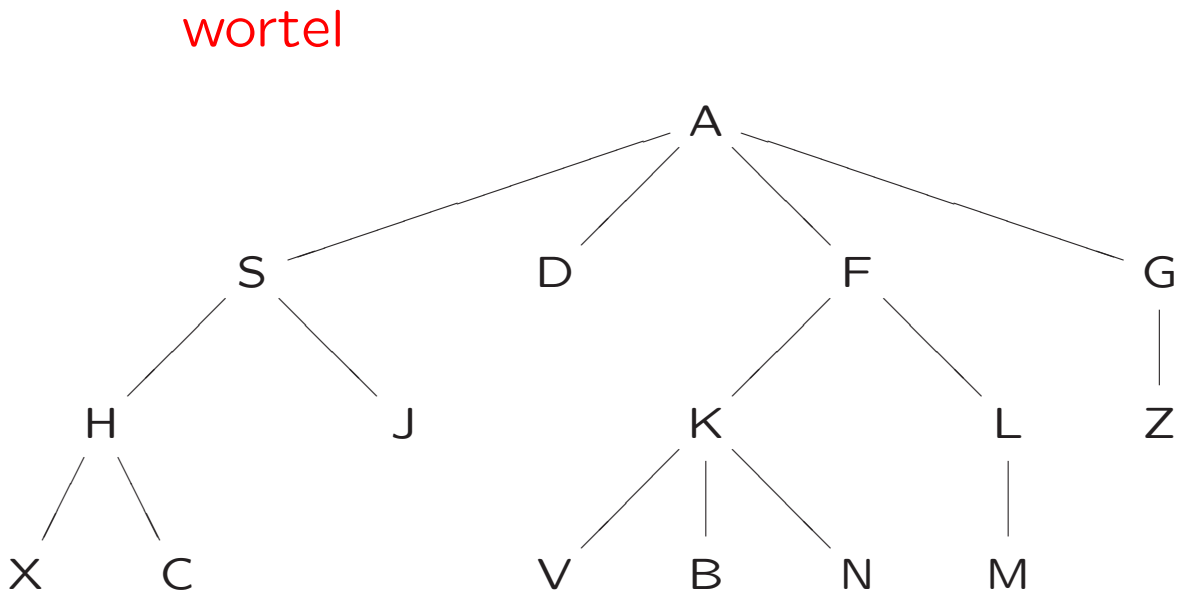
Wijs een speciale knoop aan, de *wortel*. Teken de wortel bovenaan en alle paden vanuit de wortel naar beneden: dit geeft een hiërarchische structuur die lijkt op een stamboom. Dit heet ook wel een georiënteerde boom. Meestal spreken we gewoon van een boom.

Stamboomterminologie:
kind $\longleftrightarrow$ ouder,
afstammeling $\longleftrightarrow$ voorouder.

In een georiënteerde boom hebben we dus ouder-kind relaties tussen knopen.



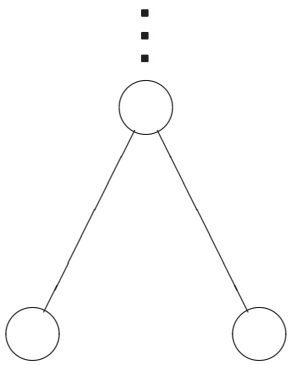
Terminologie:
takken, knopen, niveau, hoogte, wortel,
bladeren $\longleftrightarrow$ interne knopen



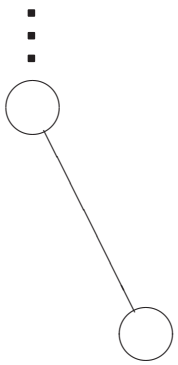
De wortel (hier A) is de *enige* ingang tot de boom.

- Een acyclische graaf die niet samenhangend is heet een bos. Het is namelijk een collectie bomen.
- Voor bomen geldt: aantal takken = aantal knopen - 1.
- Een geordende boom is een boom waarin van elke knoop de kinderen geordend zijn: oudste kind, een na oudste kind, ..., jongste kind.
- Onderstaande bomen zijn als georiënteerde bomen wel gelijk, maar als geordende georiënteerde bomen niet:

Een binaire boom is een boom waarin elke knoop ofwel nul, ofwel één ofwel twee kinderen heeft; als een knoop twee kinderen heeft dan is het ene kind het **linkerkind**, het andere het **rechterkind**; als een knoop één kind heeft, dan is dit ofwel een linkerkind, ofwel een rechterkind.

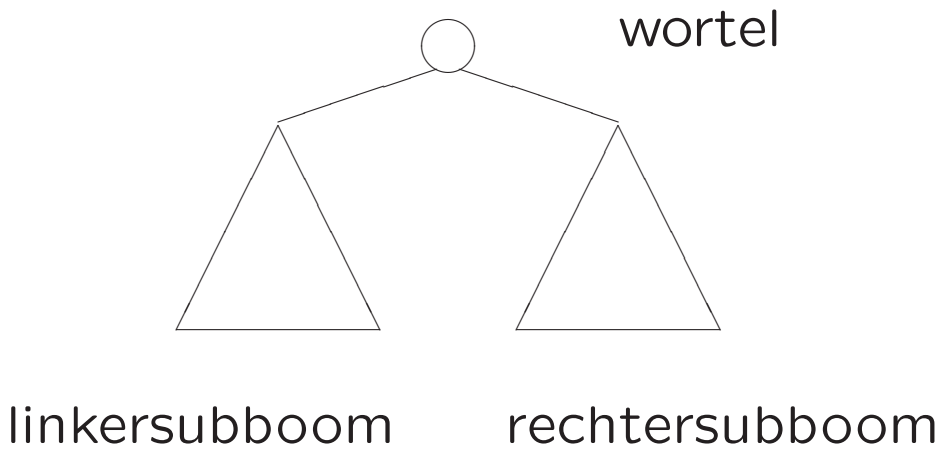


linkerkind rechterkind



één kind: rechterkind

Recursieve definitie: een binaire boom is een eindige verzameling knopen die ofwel leeg is, ofwel bestaat uit een speciale knoop (de wortel) en twee disjuncte verzamelingen knopen die samen de rest van alle knopen vormen. Die knoopverzamelingen vormen beide ook weer een binaire boom: de linkersubboom en de rechtersubboom.



```
class knoop {           // een struct mag ook
  public:
    knoop ( ) {         // constructor
      info = 0;
      links = nullptr;
      rechts = nullptr;
    }
    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 en ook bij bomenpracticum

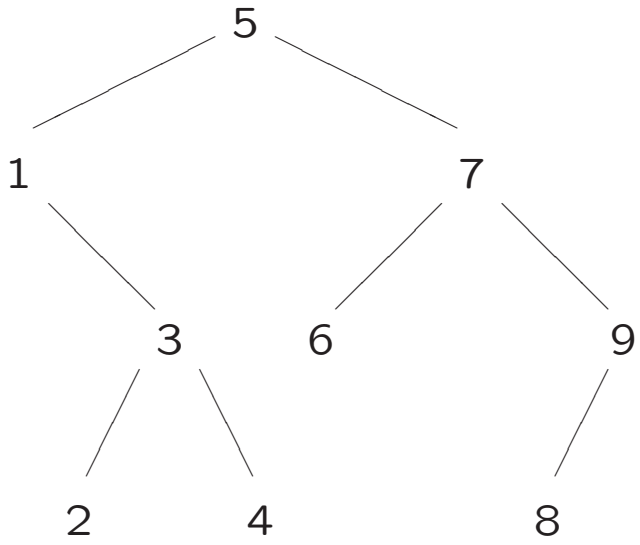
WLR (preorde):

bezoek wortel
doorloop linkersubboom WLR
doorloop rechtersubboom WLR

LWR (symmetrisch):

doorloop linkersubboom LWR
bezoek wortel
doorloop rechtersubboom LWR

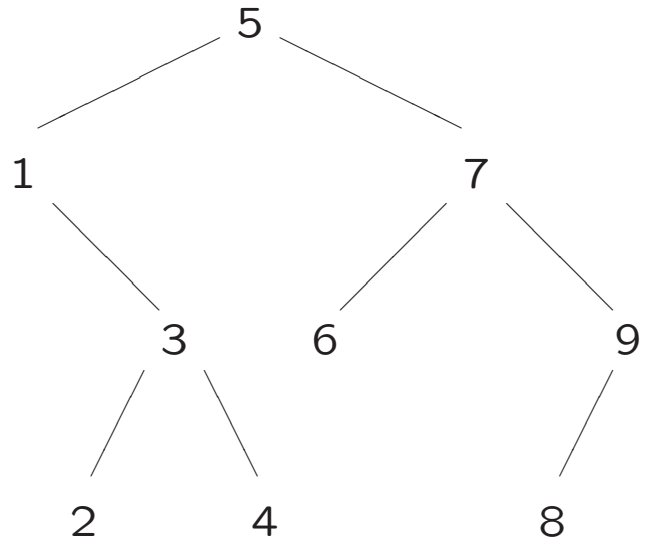
LRW (postorde): analoog



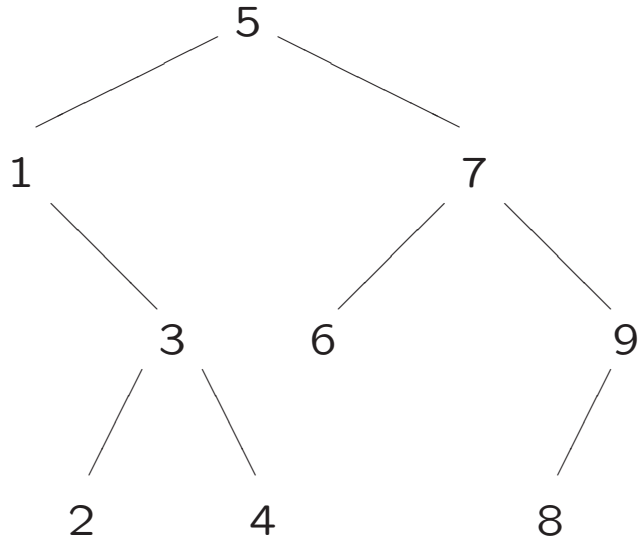
```
void preorde (knoop* root) {
  if ( root != nullptr ) {
    cout << root->info << endl;
    preorde (root->links);
    preorde (root->rechts);
  }  // if
}  // preorde

void symmetrisch (knoop* root) {
  if ( root != nullptr ) {
    symmetrisch (root->links);
    cout << root->info << endl;
    symmetrisch (root->rechts);
  }  // if
}  // symmetrisch
```

Werking recursie

```
1. void preorde (knoop* root) {
2.   if ( root != nullptr ) {
3.     cout << root->info << endl;
4.     preorde (root->links);
5.     preorde (root->rechts);
6.   }  // if
7. }  // preorde
```



```
1. void preorde (knoop* root) {
2.   if ( root != nullptr ) {
       int hulp = root->info;
3.     cout << hulp << endl;
4.     preorde (root->links);
5.     preorde (root->rechts);
6.   }  // if
7. }  // preorde
```



Lokale variabele: `int hulp`

We tellen *recursief* het aantal knopen van een binaire boom met ingang wortel.
Aanroep: `int tellen = aantal (wortel);`

```
int aantal (knoop* root) {
  if ( root == nullptr )      // lege boom
    return 0;
  else
    return ( 1 + aantal (root->links)
               + aantal (root->rechts) );
}  // aantal
```

Merk op dat hier eigenlijk een preorde-wandeling wordt gedaan.

We breken de binaire boom met ingang wortel helemaal af: hiertoe wordt eerst de linkersubboom (recursief) weggegooid, daarna de rechtersubboom en ten slotte de wortel zelf. Aanroep: `breekaf (wortel);`

```
void breekaf (knoop* & root) {
  if ( root != nullptr ) {
    breekaf (root->links);                    L
    breekaf (root->rechts);                   R
    delete root;                              W
    root = nullptr;
  }  // if
} // breekaf
```

- Voor de hoogte h van een binaire boom met n knopen geldt:
  $\ldots \leq h \leq \ldots$

- Voor de hoogte h van een binaire boom met n knopen geldt:
  $\ldots \leq h \leq n - 1$

- Voor de hoogte h van een binaire boom met n knopen geldt:
  $\lfloor \log_2 n \rfloor = \lceil \log_2(n+1) \rceil - 1 \leq h \leq n - 1$

- Hoeveel binaire bomen met minimale / maximale hoogte

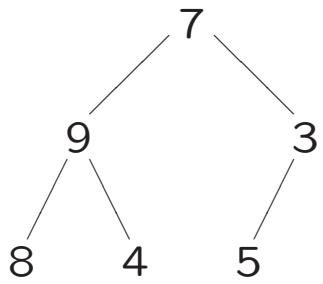
- Voor de hoogte h van een binaire boom met n knopen geldt: $\lfloor \log_2 n \rfloor = \lceil \log_2(n+1) \rceil - 1 \leq h \leq n - 1$

- Hoeveel binaire bomen met minimale vs maximale hoogte: 1 vs 2^{n-1}

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

- Een binaire zoekboom is een binaire boom waarbij voor elke knoop geldt dat de waarde in die knoop groter is dan alle waarden in zijn linkersubboom, en kleiner dan alle waarden in zijn rechtersubboom.

Complete binaire boom:

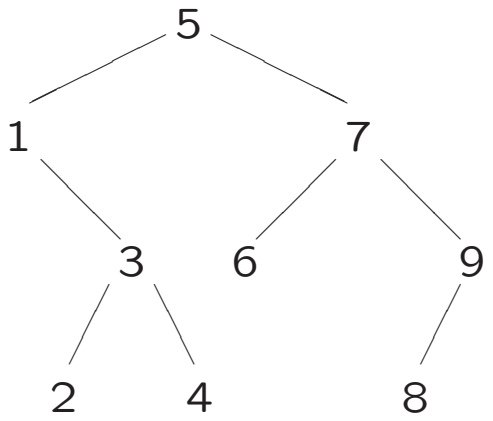


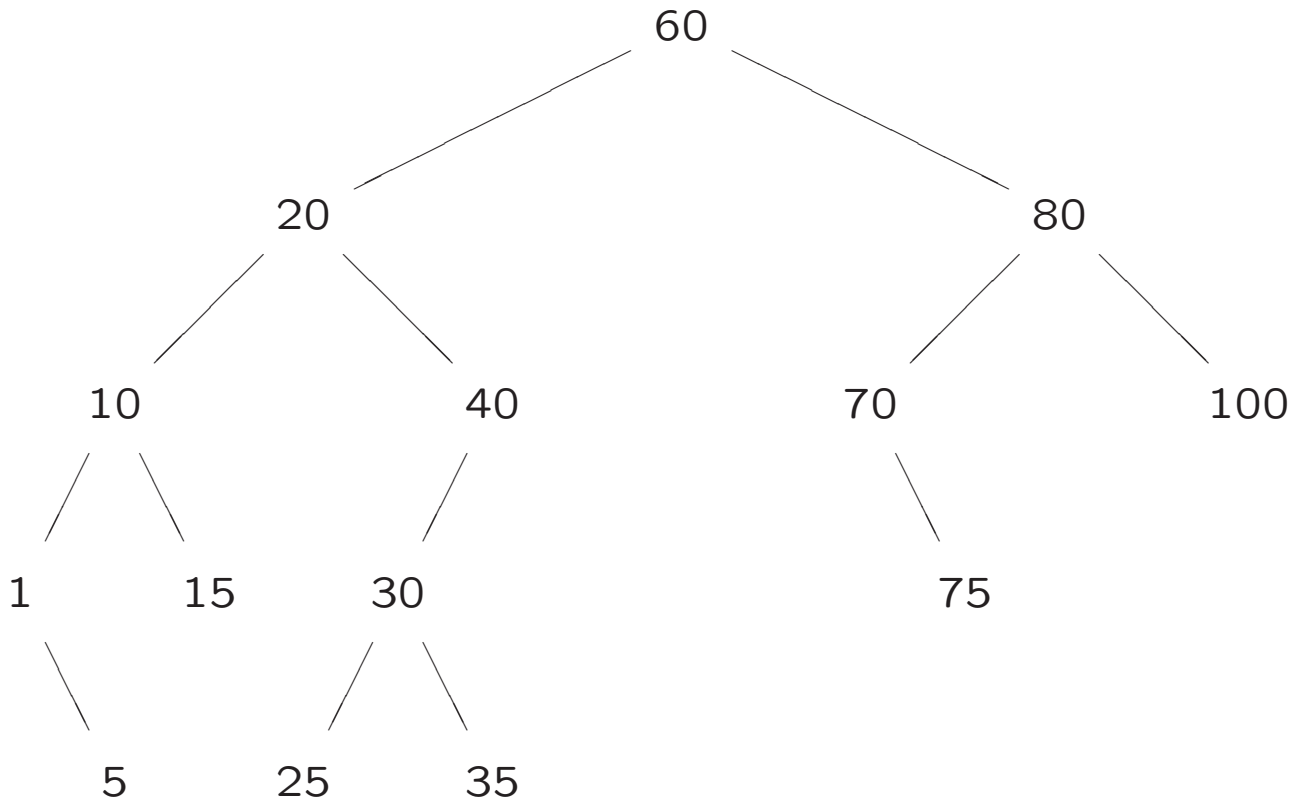
$\Longleftrightarrow$ 7 9 3 8 4 5

Binaire zoekboom:

LWR

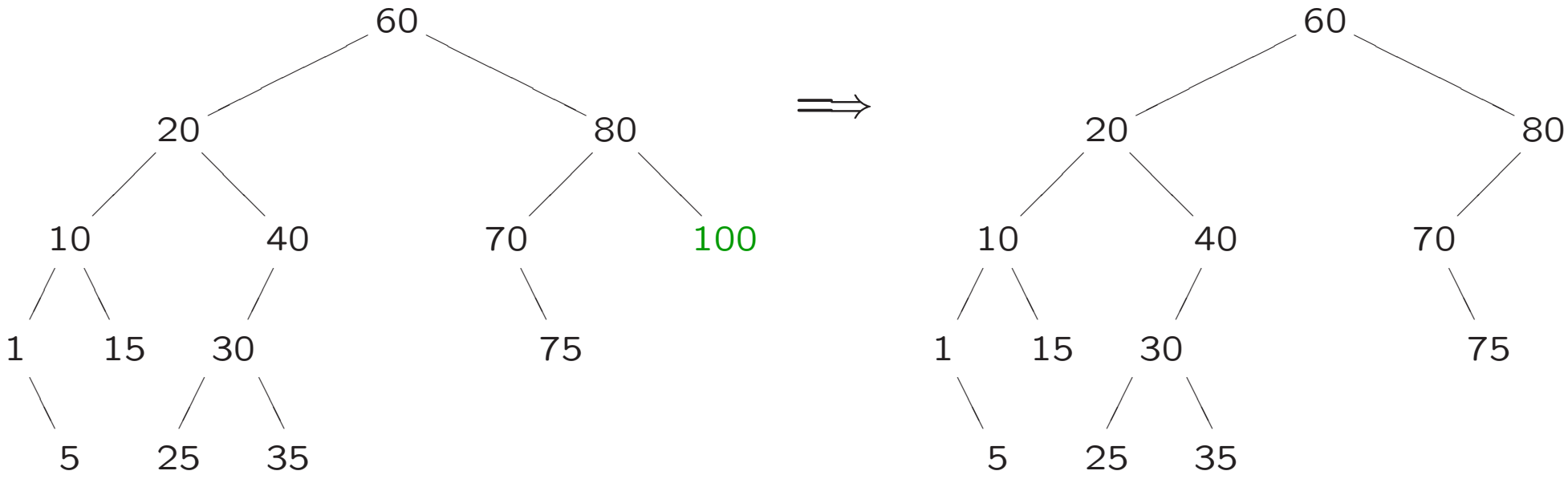
1 2 3 4 5 6 7 8 9





toevoegen. . .

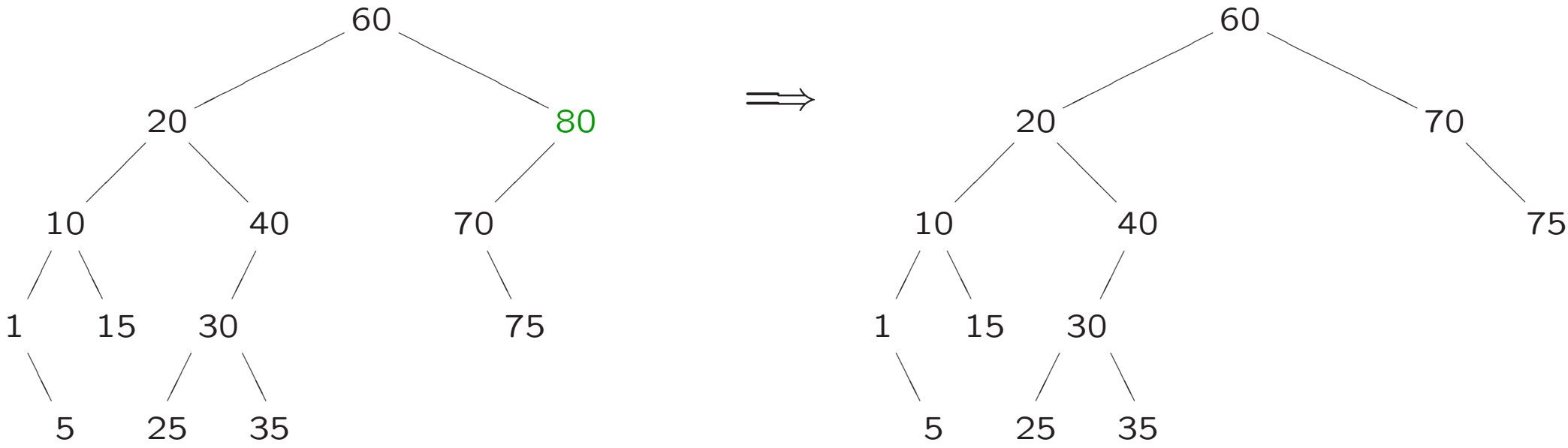
100 verwijderen



verwijderen van een blad

vervolgens 80

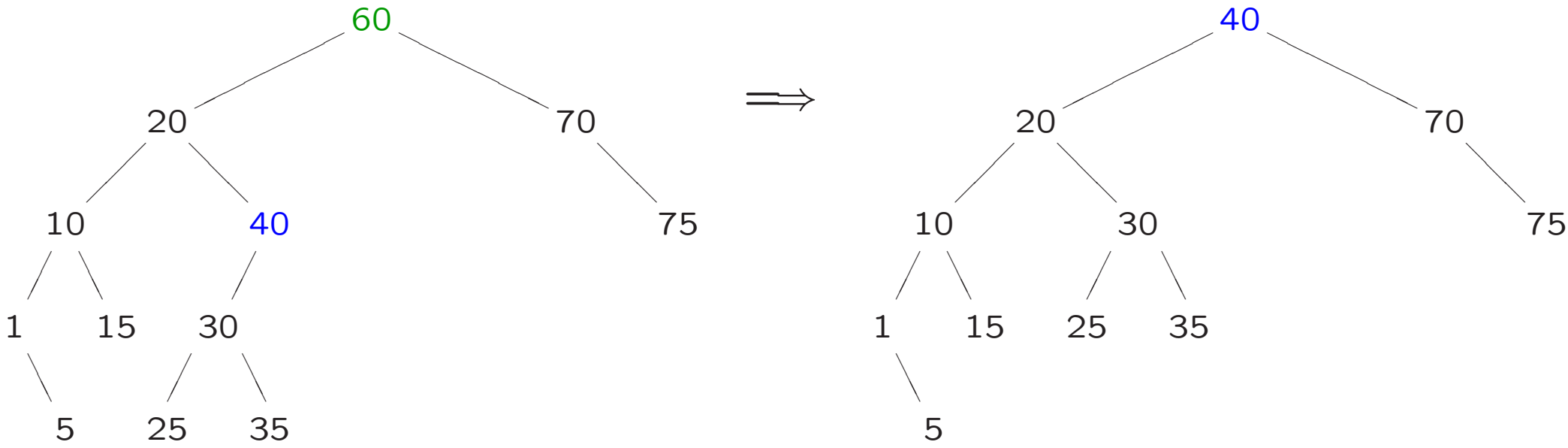
80 verwijderen



verwijderen van een knoop met 1 kind

vervolgens 60

60 verwijderen



verwijderen van een knoop met 2 kinderen

verwissel met de grootste kleinere of kleinste grotere

- **Lezen/leren bij dit college:**
  Paragraaf 1.4 (vanaf/na graafrepresentaties)

- **Werkcollege**: vrijdagochtend geen werkcollege

- **Practicumbijeenkomst:**
  dinsdag 18 februari, 09.00–10.45, bomenpracticum, computerzalen

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

- **Volgend college:**
  Woensdag 19 februari 2025, 15.15–17.00