

# Tweede college algoritmiek

14 februari 2014

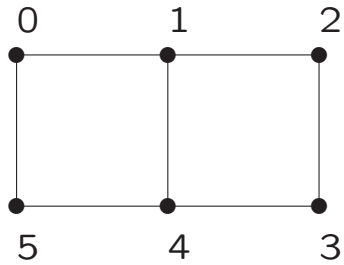
Grafen en bomen

Intro toestand-actie-ruimte

Een **graaf**  $G$  wordt gedefinieerd als een paar  $(V, E)$ , waarbij  $V$  een eindige verzameling is van **knopen** (vertices) en  $E$  een verzameling van paren van knopen: de **takken/pijlen** (edges). Een tak/pijl  $(u, v)$  verbindt de knopen  $u$  en  $v$  met elkaar.

### Terminologie:

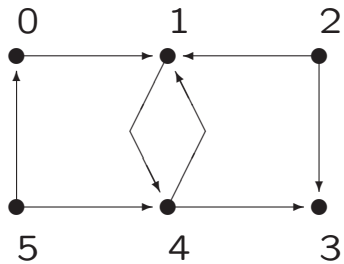
- ongerichte/gerichte graaf
- gewogen graaf
- paden en cykels/kringen
- samenhangend
- acyclisch



1.

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

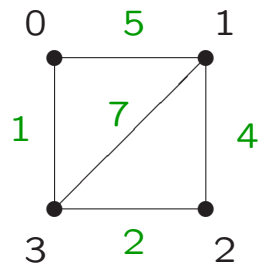
$$E = \{(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 0), (4, 1)\}$$



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.

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

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

**Adjacency matrix:** de **gerichte** graaf wordt gerepresenteerd door een tweedimensionaal array `int inhoud[n][n]` ( $n$  het aantal knopen), waarbij `inhoud[i][j]` aangeeft of er een pijl van  $i$  naar  $j$  gaat. (Analoog ongerichte grafen; zie college 1.)

**Adjacency list:** de **gerichte** graaf wordt gerepresenteerd door een eendimensionaal array `buur* inhoud[n]` ( $n$  het aantal knopen), waarbij `inhoud[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  $i$ . (Analoog ongerichte grafen; zie college 1.)

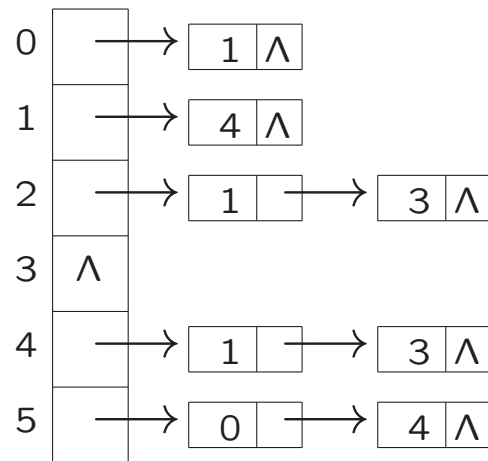
Adjacency list representatie in C++:

```
struct buur {
    int knoopnummer;
    buur* volgende;
} // misschien meer velden, bv. char info of int gewicht
buur* inhoud[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. Neem aan dat de omgekeerde pijl  $(j, i)$  nog niet bestaat.

```
// het array graaf (buur* graaf[n]) bevat de gerichte graaf
buur* hulp = graaf[i]; buur* vorige = NULL;
while ( hulp->knoopnummer != j ) { // knoop j zoeken
    vorige = hulp; hulp = hulp->volgende;
}
// haal buurknoop j uit lijst
if ( vorige == NULL ) { // j is de eerste buur van i
    graaf[i] = hulp->volgende;
}
else { // j is niet de eerste buur van i
    vorige->volgende = hulp->volgende;
}
delete hulp; // gooi buur j weg
// voeg nu i vooraan in de buurlijst van j toe
hulp = new buur;
hulp->knoopnummer = i; hulp->volgende = graaf[j];
graaf[j] = hulp;
```

Gegeven een adjacency list representatie van een gerichte graaf: hoeveel stappen kost het om het aantal uitgaande pijlen van iedere knoop te vinden? En het aantal inkomende pijlen?

En nu dezelfde vraag, maar voor een adjacency matrix representatie?

Hoeveel stappen kost het om te testen of van de ene knoop een pijl loopt naar de andere bij een adjacency list respectievelijk een adjacency matrix representatie?

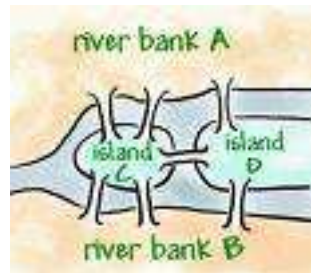
Hoeveel ruimte neemt een adjacency list in beslag? En een matrix?

- 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).
- De lengte van een pad = het aantal takken op dat pad = het aantal knopen - 1.
- Als alle knopen verschillend zijn heet het pad **enkelvoudig**.

- Een **kring** in een ongerichte graaf is een pad dat begint en eindigt in dezelfde knoop en dat geen enkele tak meer dan één keer bevat.
- 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**.
- Een graaf die samenhangend en acyclisch is noemen we een boom
- In een boom is er precies één pad tussen elk tweetal knopen  $u$  en  $v$ .

Twee zeer bekende graafproblemen:

Koningsberger bruggen probleem Vierkleurenprobleem

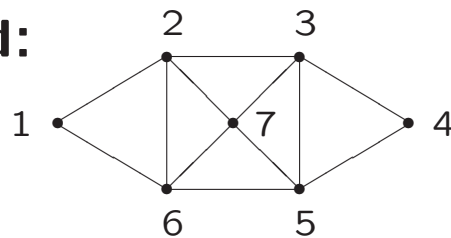


Hoezo  
graafproblemen?

**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?

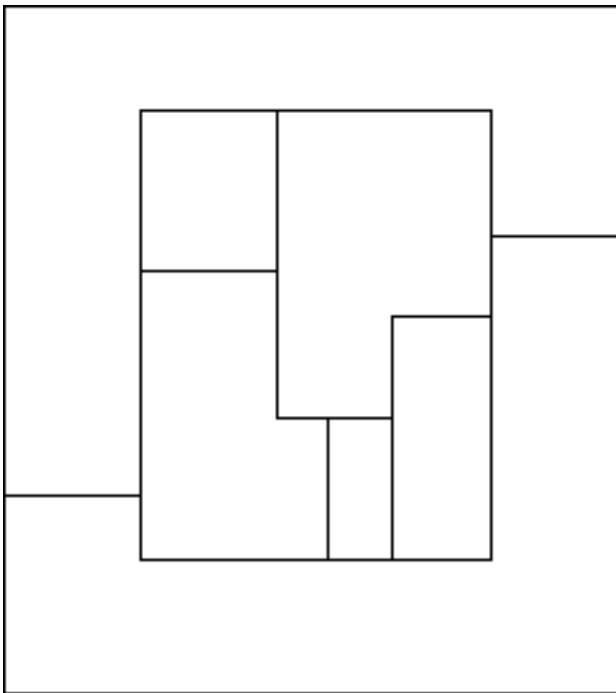
**Voorbeeld:**



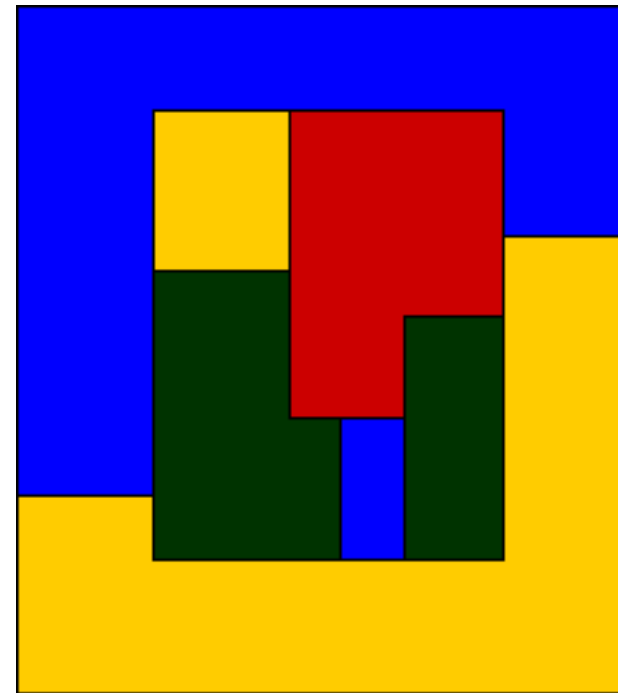
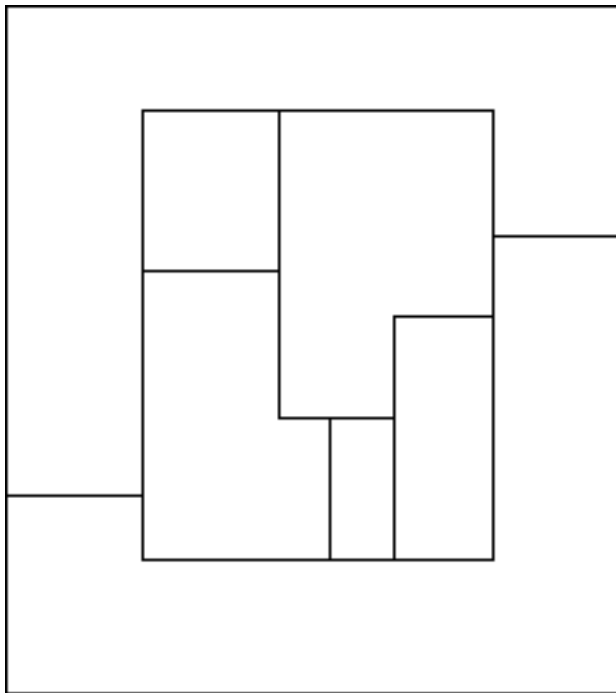
Voor deze graaf is 1 2 3 4 5 3 7 5 6 7 2 6 1 een Eulerkring.

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

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:

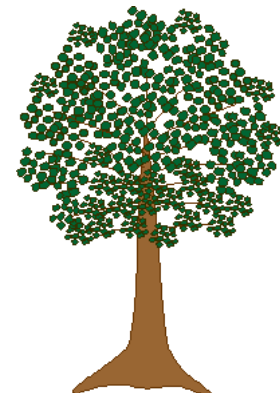


Definitie: een **boom** is een samenhangende (= uit één stuk bestaande) ongerichte graaf zonder cykels (= kringen).

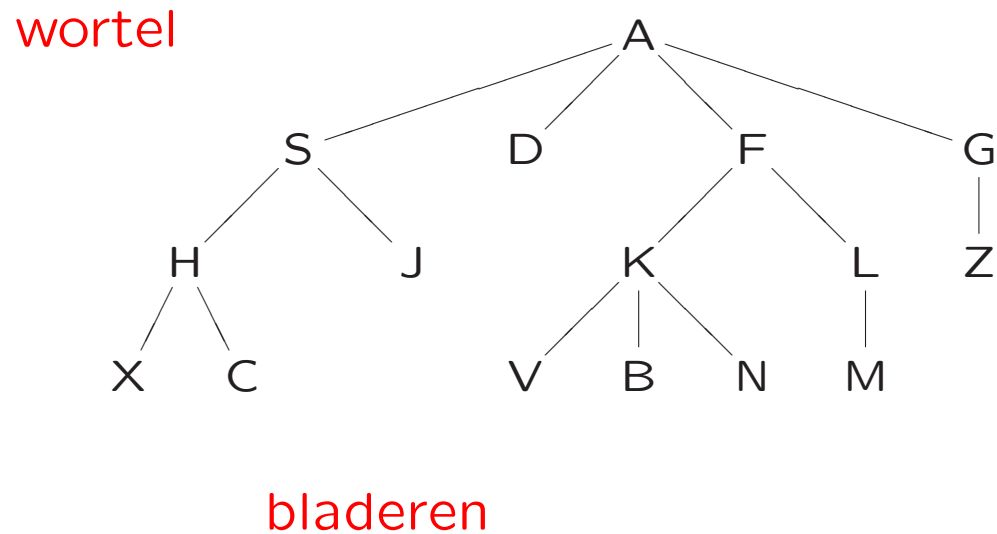
Wijs een speciale knoop aan, de *wortel*. Teken de wortel bovenaan en alle paden vanuit de wortel naar beneden: dit geeft een hiërarchische structuur die lijkt op een stamboom. Dit heet ook wel een **georiënteerde boom**. Meestal spreken we gewoon van een boom.

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

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

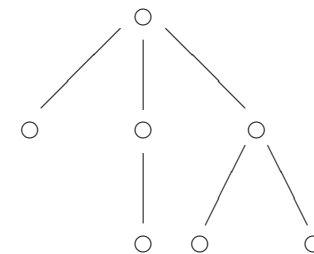
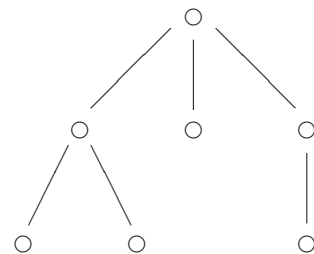


Terminologie: takken, knopen, niveau, hoogte, wortel, bladeren  $\longleftrightarrow$  interne knopen

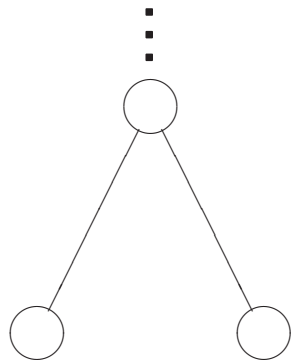


De **wortel** (hier A) is de **enige ingang** tot de boom.

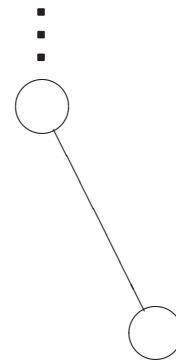
- Een acyclische graaf die niet samenhangend is heet een **bos**. Het is namelijk een collectie bomen.
- Voor bomen geldt: aantal takken = aantal knopen - 1.
- Een **geordende boom** is een boom waarin van elke knoop de kinderen geordend zijn: oudste kind, een na oudste kind, ..., jongste kind.
- Onderstaande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

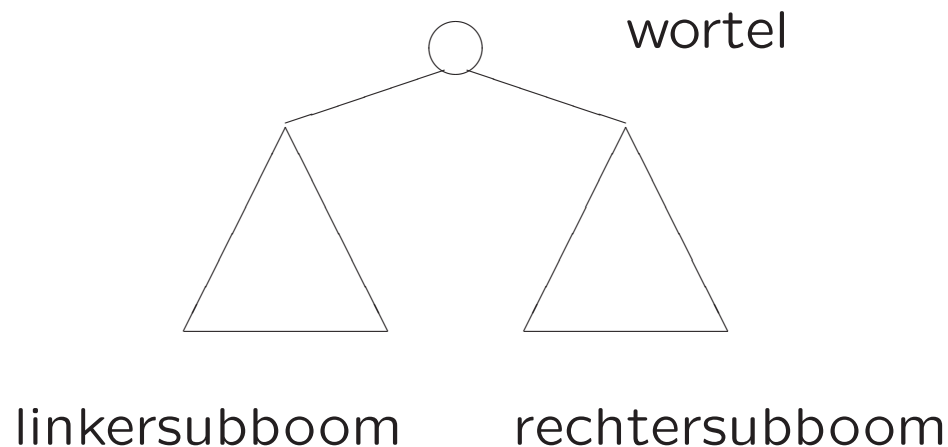


linkerkind    rechterkind



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 **rechtsubboom**.



```
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](#)

- » [Klimpark](#)
- » [Boomwandeling](#)
- » [Schnitzas houtpark](#)
- » [Duivelswater](#)
- » [De Lindingalm](#)

- » [Schnappsbranderij](#)
- » [Tennis](#)
- » [Bergkristalmuseum](#)
- » [Reiterkogellift](#)

## De boomwandeling & de Golden Gate Bridge van de Alpen



De boomwandelweg is de hoogst gelegen boomwandeling van Europa, samen met de Golden Gate Bridge van de Alpen is de combinatie, bestaande uit 1 km wandeling in boomkruinen en een 200 m lange hangbrug op rond 40 m hoogte, een wereldwijd uniek project. Door de unieke constructie met trappen, torentjes, bruggen en platformen ontstonden verschillende prachtige uitzichtspunten in het adembenemende bergdecor van Saalbach Hinterglemm.

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 != NULL ) {
        cout << root->info << endl;
        preorde (root->links);
        preorde (root->rechts);
    } // if
} // preorde

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

We tellen *recursief* het aantal knopen van een binaire boom met ingang wortel.

Aanroep: `int tellen = aantal (wortel);`

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

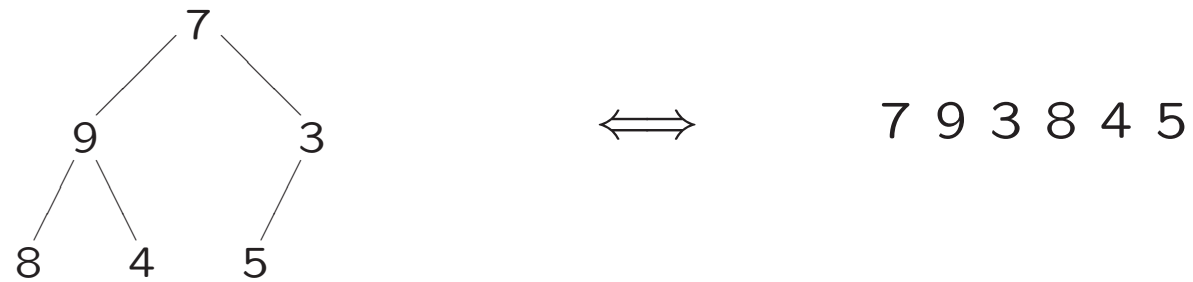
Merk op dat hier eigenlijk een preorde-wandeling wordt gedaan.

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);`

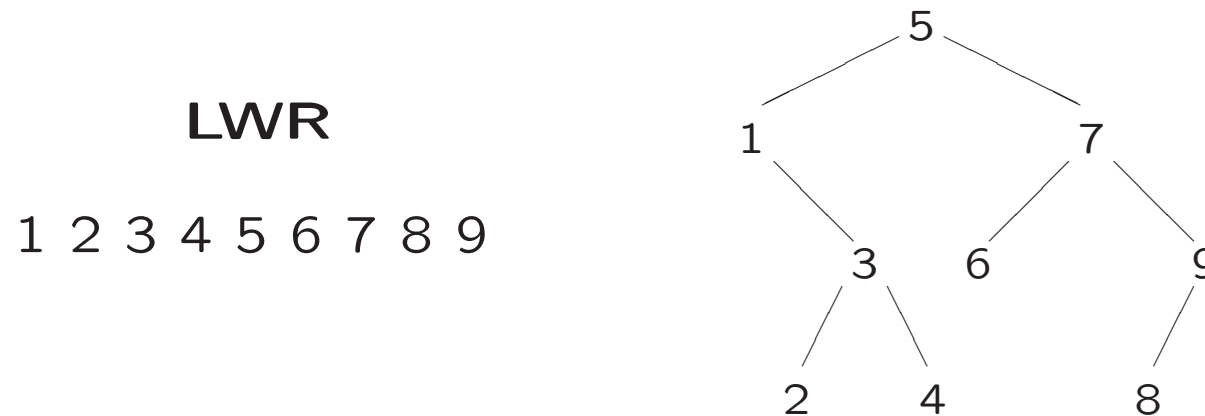
```
void breekaf (knoop* & root) {  
    if ( root != NULL ) {  
        breekaf (root->links);    L  
        breekaf (root->rechts);   R  
        delete root;             W  
        root = NULL;  
    } // if  
} // breekaf
```

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



Binaire zoekboom:



**Probleem  $\longrightarrow$  Toestand-actie-ruimte**

Een **toestand-actie-ruimte** (toestand-actie-diagram, state transition diagram, toestandsruimte, state space)

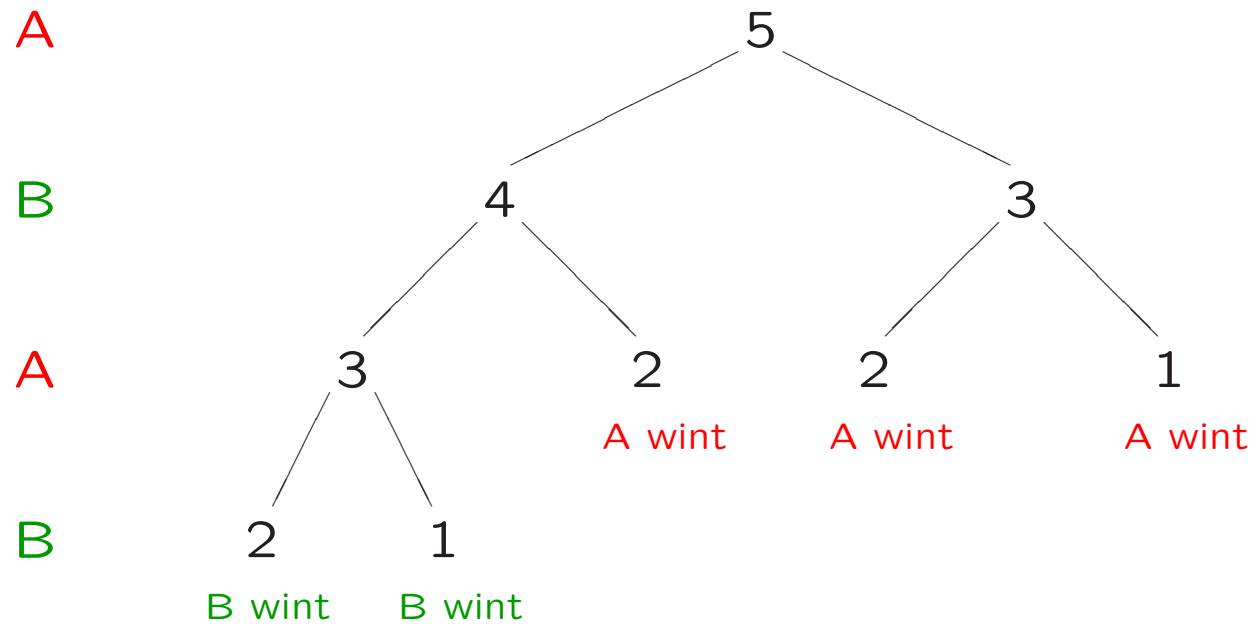
- *Bestaat uit* alle mogelijke **toestanden en acties**
- Begintoestand, eindtoestand(en)
- Een actie veroorzaakt een overgang van de ene (toegelaten) toestand naar een andere
- *Oplossing* van het probleem: een opeenvolging van acties die van de begintoestand naar een eindtoestand leiden

**Voorbeeld 1: NIM**

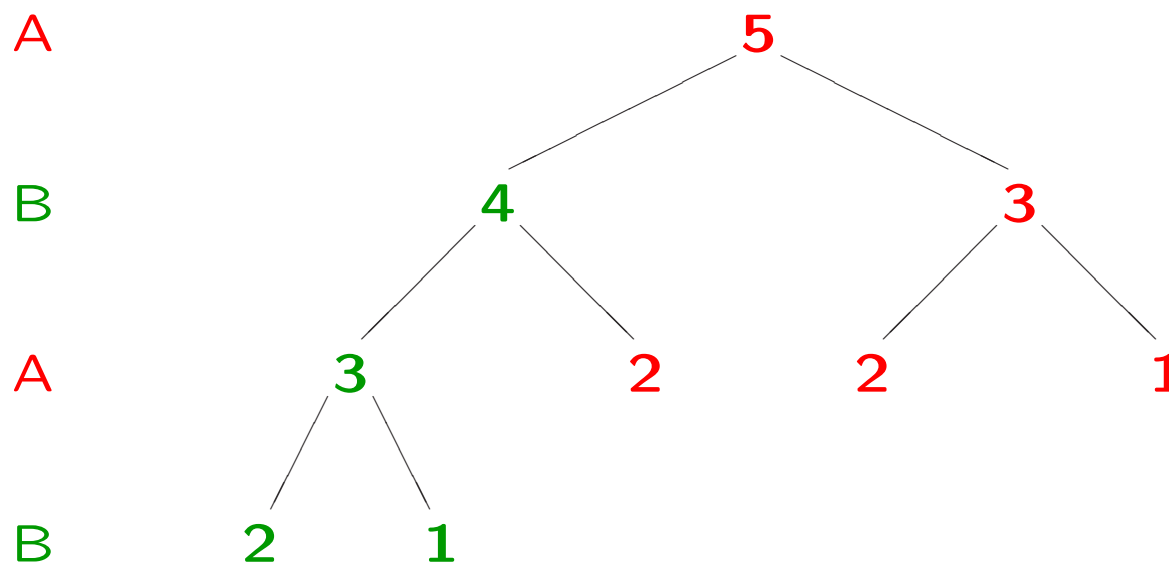
- Begintoestand: een stapel van  $n$  lucifers
- Er zijn twee spelers: **A** en **B**
- De spelers pakken om de beurt 1 of 2 lucifers (acties)
- Het spel is afgelopen als er geen lucifers meer op de stapel liggen (eindtoestand)
- De speler die de laatste lucifer(s) pakt heeft gewonnen

**Gevraagd:** is het spel winnend voor degene die begint?

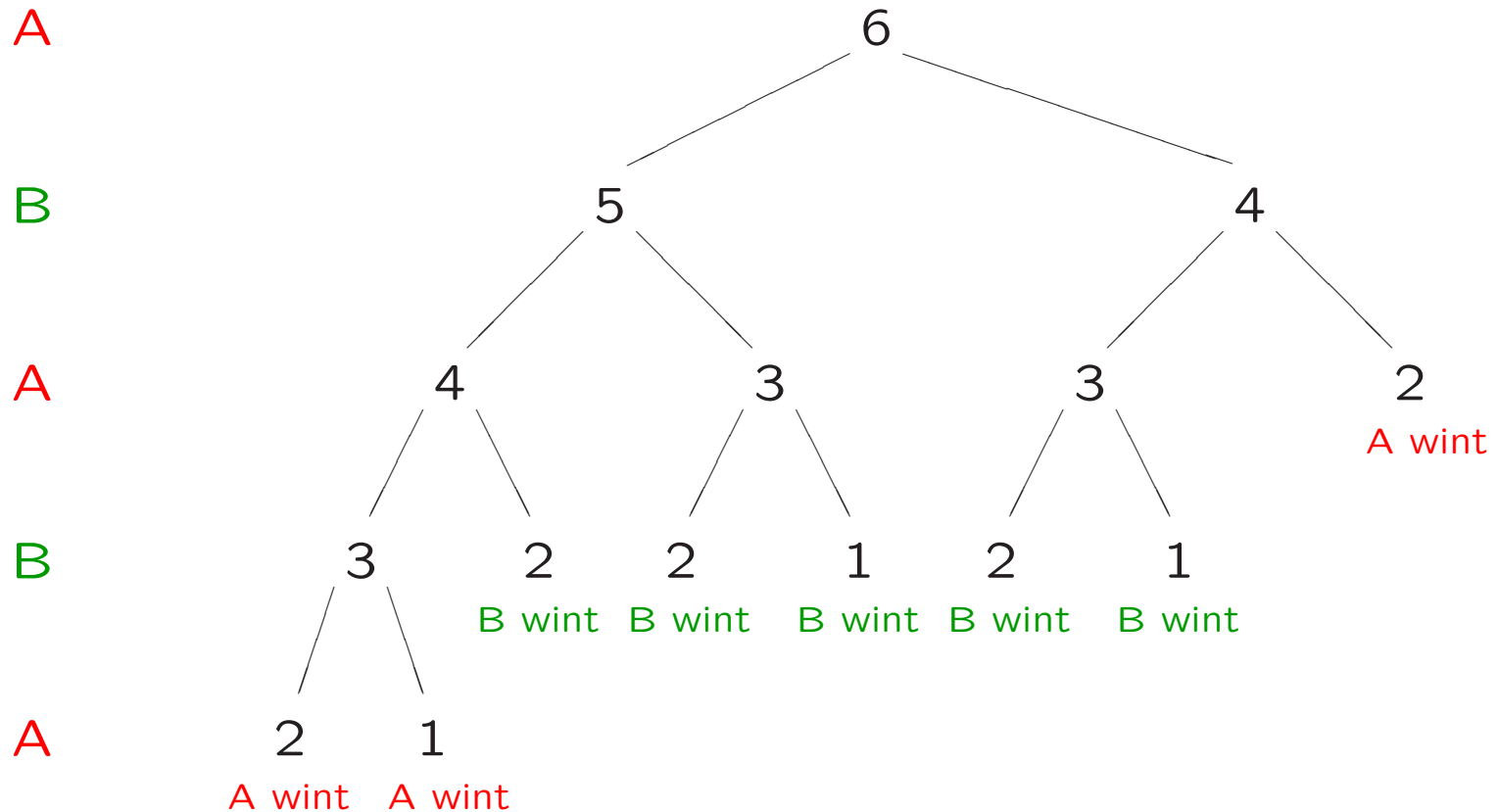
toestand-actie-ruimte (**spelboom**)  $n = 5$



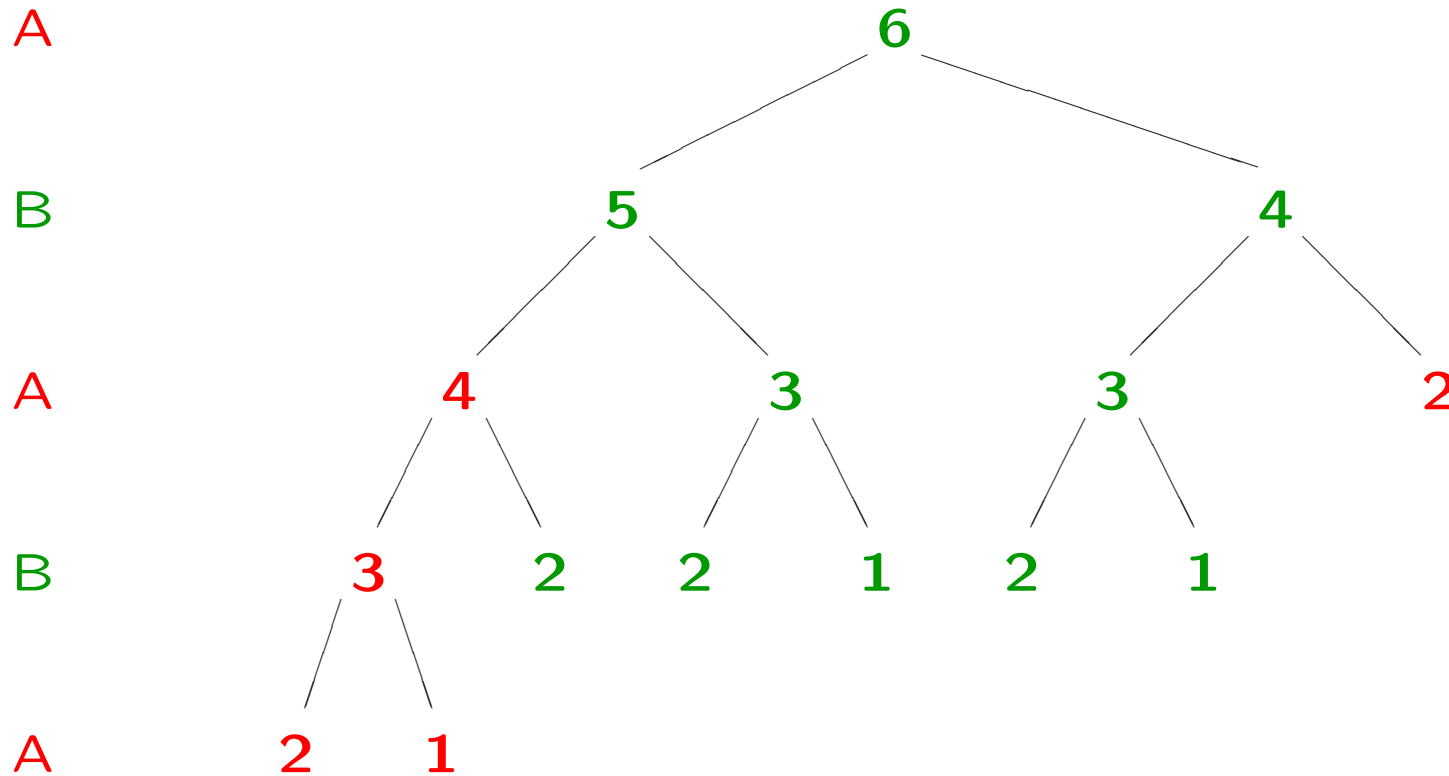
**Winnend voor A.**  
 Winnende zet: 2 lucifers wegnemen



toestand-actie-ruimte (spelboom)  $n = 6$



**A kan niet winnen** (bij perfect spel van B)



- **Lezen/leren uit Levitin bij dit college:**

Paragraaf 1.4 (vanaf graafrepresentaties)

- **Werkcollege:** Bomenpracticum

donderdag 20 februari 2014, 11:15—13:00, **computer-  
zalen 306/308, 303**

- **Opgaven:**

zie <http://www.liacs.nl/home/graaf/ALGO/>

- Zie de website voor de eerste programmeeropdracht

- **Volgend college:**

vrijdag 21 februari 2014, 11:15—13:00, zaal B2

Stel, je hebt een zak met 20 zwarte ballen en 16 witte ballen (en wat extra witte en zwarte ballen tot je beschikking). Je herhaalt nu de volgende operatie totdat er nog maar één bal in de zak over is.

Je haalt zonder te kijken twee ballen tegelijk uit de zak. Als deze dezelfde kleur blijken te hebben, doe je een zwarte bal in de zak; als ze verschillend van kleur zijn, doe je een witte bal in de zak.

De vraag is: wat is de kleur van de bal die als laatste overblijft in de zak?

Dezelfde vraag, maar nu begin je met 20 zwarte en 15 witte ballen in de zak.