

# Derde college algoritmiek

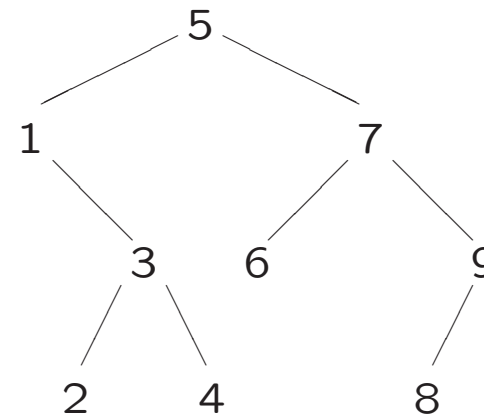
20 februari 2015

Toestand-actie-ruimte

- 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.

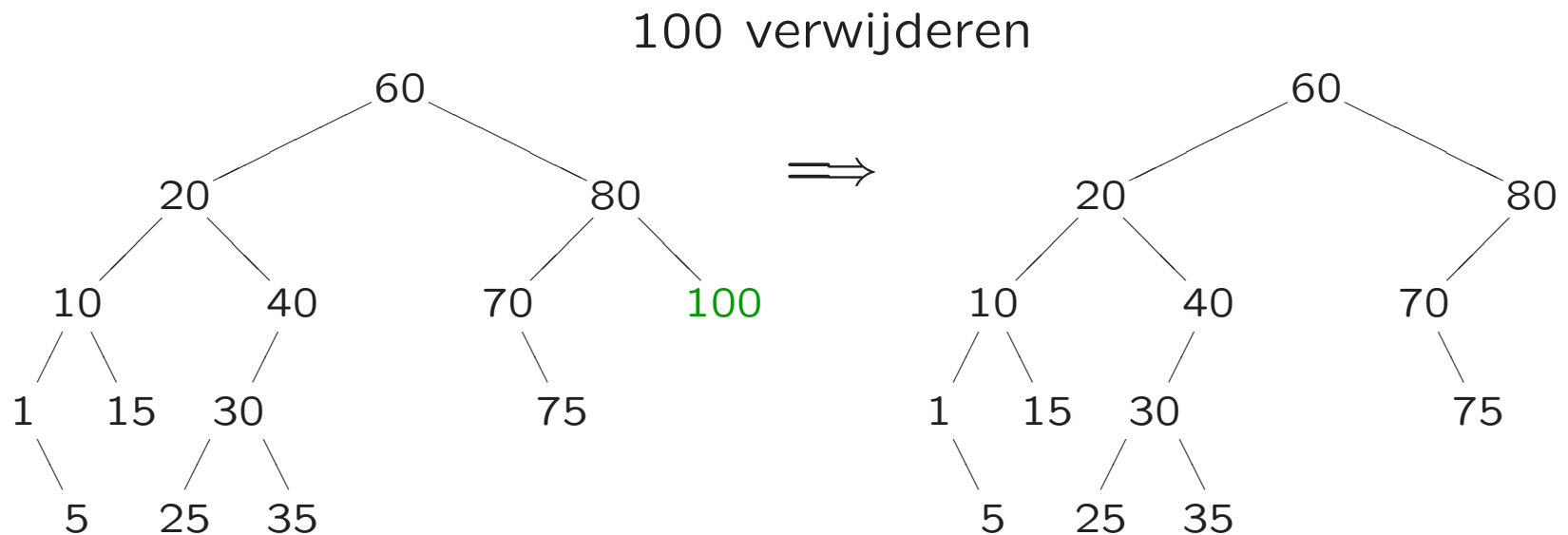
**LWR**

1 2 3 4 5 6 7 8 9



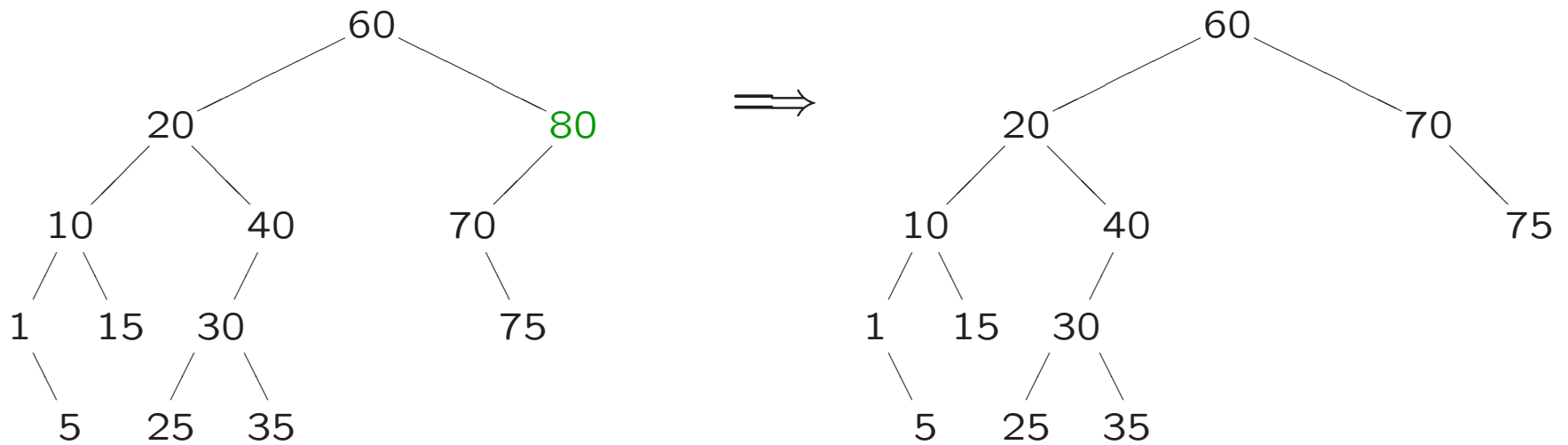
\*zie ook werkgroep 2 en Levitin, p.60/189/190

Eerst de weg te gooien waarde/knoop zoeken en deze vervolgens netjes verwijderen. Hierbij is de ouder van de te verwijderen knoop nodig. We onderscheiden drie gevallen: de knoop is een blad of heeft één kind of heeft twee kinderen.

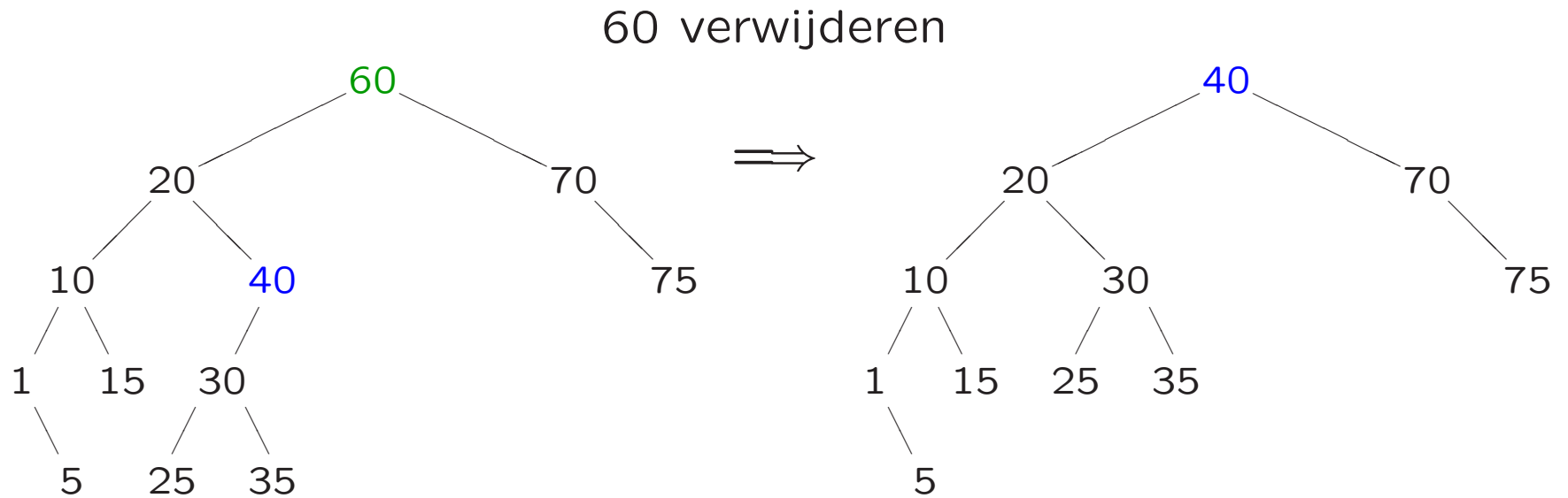


verwijderen van een blad

80 verwijderen



verwijderen van een knoop met 1 kind



**verwijderen van een knoop met 2 kinderen**

verwissel met de grootste kleinere of kleinste grotere

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

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

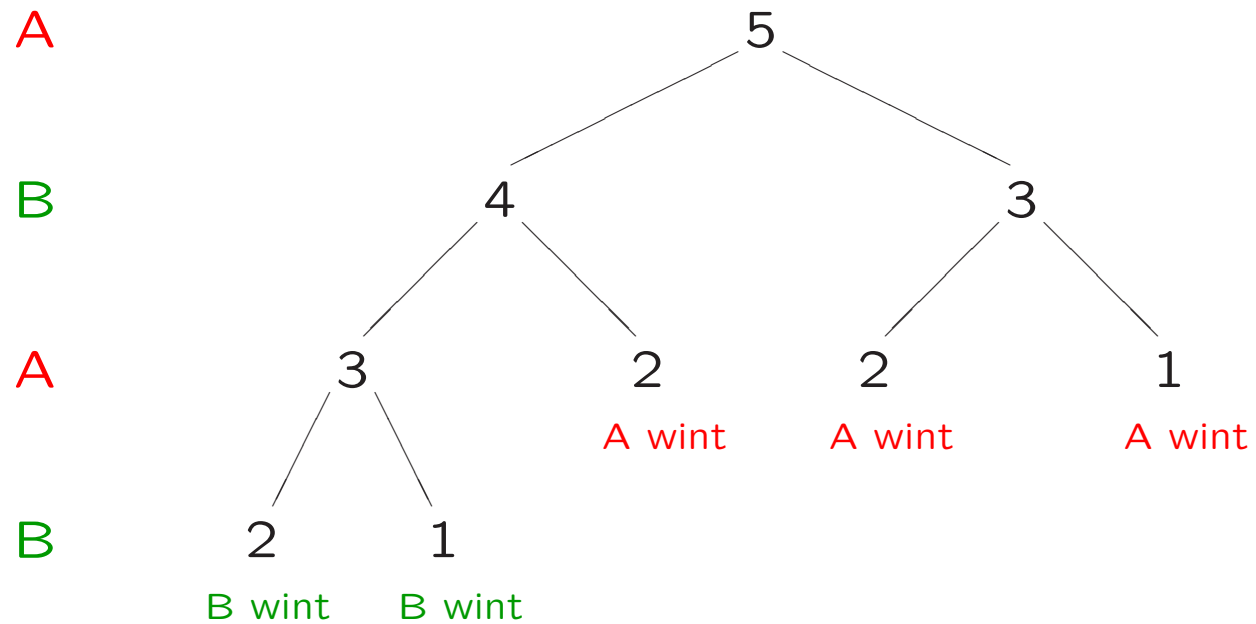
- *Bestaat uit* alle mogelijke **toestanden en acties**
- Begintoestand, eindtoestand(en)
- Een actie veroorzaakt een overgang van de ene (toegelaten) toestand naar een andere
- *Oplossing* van het probleem: een opeenvolging van acties die van de begintoestand naar een eindtoestand leiden

## Voorbeeld 1: NIM

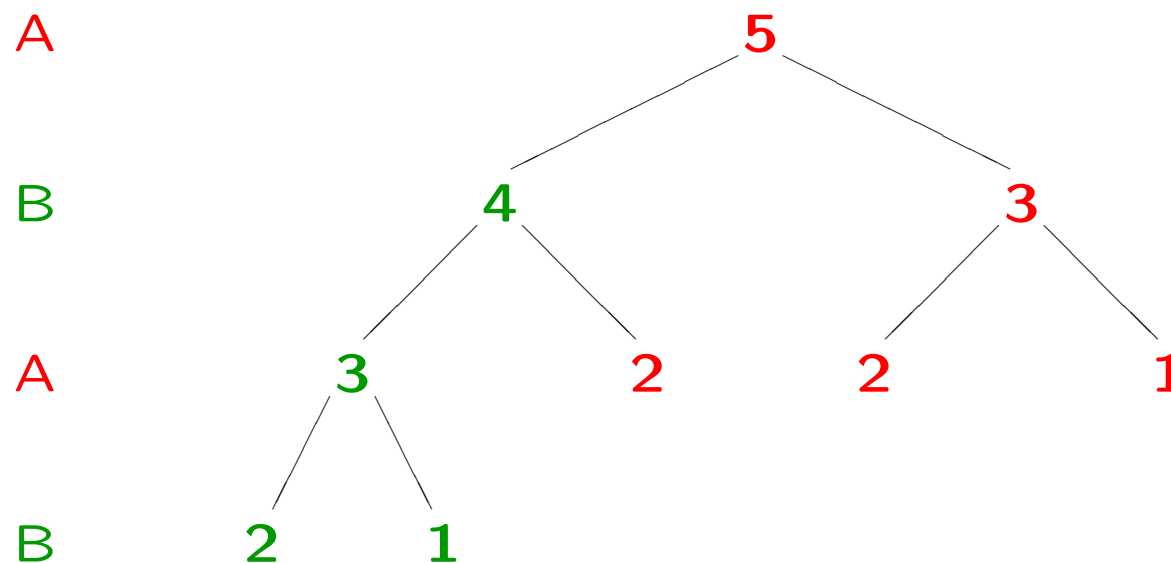
- We beginnen met één stapel van  $n$  lucifers (begintoestand)
- Er zijn twee spelers: **A** en **B**
- De spelers pakken om de beurt 1 of 2 lucifers (acties)
- Het spel is afgelopen als er geen lucifers meer op de stapel liggen (eindtoestand)
- De speler die de laatste lucifer(s) pakt heeft gewonnen

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

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

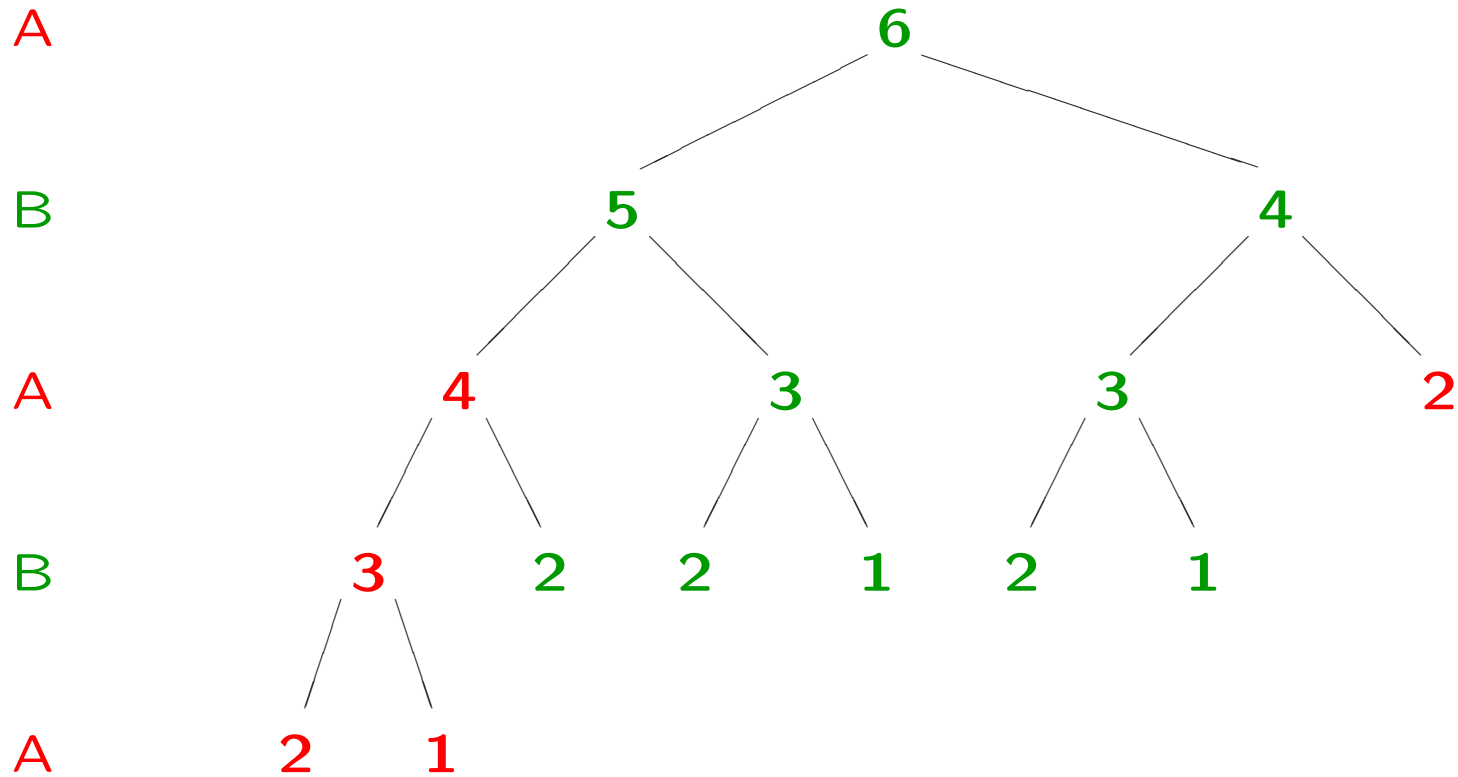


**Winnend voor A.**  
 Winnende zet: 2 lucifers wegnemen

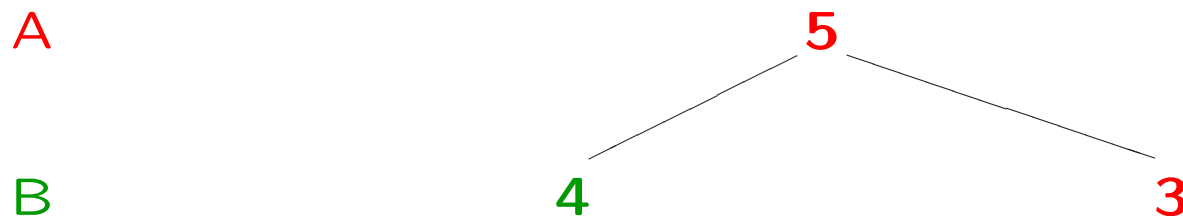




**A kan niet winnen** (bij perfect spel van B)

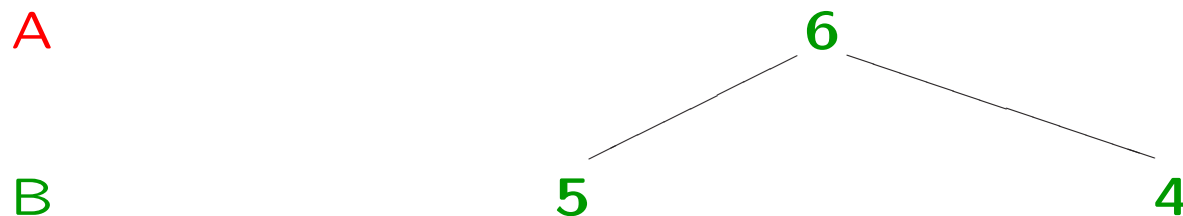


Een stand is **winnend** voor degene die aan de beurt is als een der directe (= in 1 zet te bereiken) vervolgstanden NIET **winnend** is voor de tegenstander.



Speler **A** is aan zet en een van de directe vervolgstanden (namelijk via de rechter zet, neem 2 lucifers) is niet winnend voor de tegenstander.

Een stand is **verliezend** (ofwel: NIET **winnend**) voor degene die aan de beurt is als ALLE directe (= in 1 zet te bereiken) vervolgstanden **winnend** zijn voor de tegenstander.



Speler **A** is aan zet en alle directe vervolgstanden (zowel via de linker als via de rechter zet) zijn winnend voor de tegenstander.

Een algoritme dat bepaalt of een stand **winnend** is, ziet er dus ruwweg zo uit:

Loop alle mogelijke directe vervolgstanden af:

kijk of je er een tegenkomt die not **winnend** is voor de tegenstander: **recursie**

zo ja, dan is de oorspronkelijke stand winnend (en heb je meteen een winnende zet) en hoef je niet verder te kijken

zo nee, dan de volgende vervolgstand proberen

Als alle vervolgstanden zijn geweest (en ook de laatste vervolgstand was winnend voor de tegenstander) is de oorspronkelijke stand niet winnend.

```
winnend(stand)::  
  
    if eindstand(stand) then  
        // makkelijk; bijv return false;  
    else  
        for alle mogelijke zetten i do  
            kopie := stand; // maak een kopie  
            doezet(kopie,i); // en doe de zet daarin  
            if not winnend(kopie) then  
                return true;  
            fi  
        od  
        return false;  
    fi
```

Zie ook Programmeermethoden (college over recursie)

```
winnend(stand)::  
  
    if eindstand(stand) then  
        // makkelijk; bijv return false;  
    else  
        for alle mogelijke zetten i do  
            // nu met terugzetten !  
            doezet(stand,i);  
            if not winnend(stand) then  
                neemzetterug(stand,i);  
                return true;  
            fi  
            neemzetterug(stand,i);  
        od  
        return false;  
    fi
```

Gebruik een kopie en doe daarin de zetten:

```
bool nimwinst (int stand) {
    int lucifer, kopie;
    if ( stand == 0 )
        return false;
    // tegenstander heeft zojuist de laatste lucifers gepakt
    else {
        // directe vervolgstanden aflopen
        for ( lucifer = 1; lucifer <= 2; lucifer++ ) {
            kopie = stand; // maak een kopie
            kopie -= lucifer; // doe een zet in de kopie
            if ( !nimwinst (kopie) ) {
                return true;
            }
        }
        return false;
    } // else
}
```

Met terugzetten:

```
bool nimwinst (int stand) {
    int lucifer;
    if ( stand == 0 )
        return false;
    // tegenstander heeft zojuist de laatste lucifers gepakt
    else {
        // directe vervolgstanden aflopen
        for ( lucifer = 1; lucifer <= 2; lucifer++ ) {
            stand -= lucifer; // doe een zet
            if ( !nimwinst (stand) ) {
                stand += lucifer; // terugzetten
                return true;
            }
            stand += lucifer; // terugzetten
        }
        return false;
    } // else
}
```

**Voorbeeld 2: Old world puzzle (Ex. 1.2.1.)**

We hebben een kool, een geit, een wolf en een boer. Deze moeten met een bootje van de ene kant van de rivier naar de andere. In het bootje kan alleen de boer met één ander iets. Als de boer er niet bij is zal de wolf de geit opeten en de geit de kool. De boer is de enige die de boot kan “besturen”.

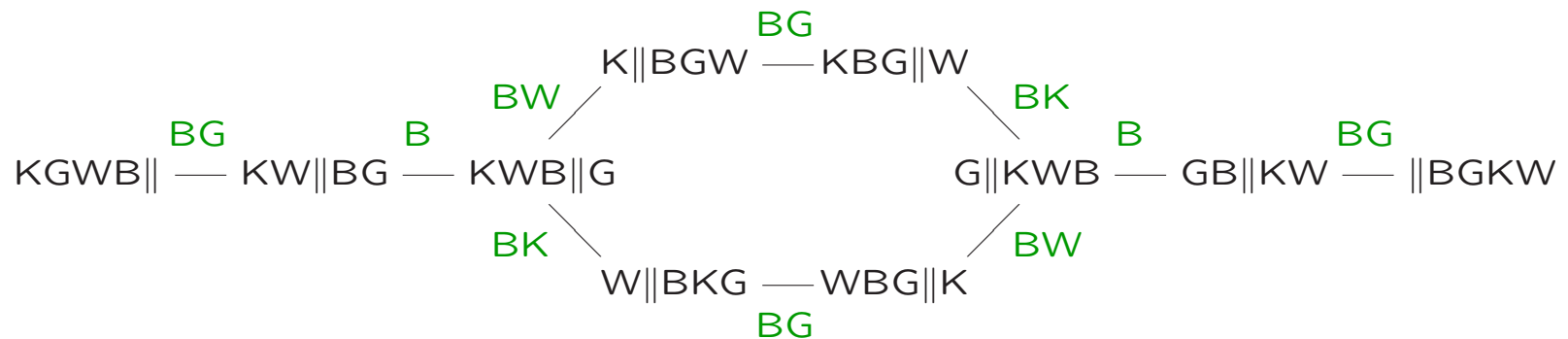
**Vraag:** Hoe kan alles (liefst zo snel mogelijk) naar de andere oever verplaatst worden?





Merk op: de boot ligt altijd aan de oever waar de boer zich bevindt.

De oplossing is een **kortste pad** van de begintoestand naar de eindtoestand: hier zijn er twee, bij beide moet de boer 7 keer de rivier oversteken.



Merk op: de boot ligt altijd aan de oever waar de boer zich bevindt.

De oplossing is een **kortste pad** van de begintoestand naar de eindtoestand: hier zijn er twee, bij beide moet de boer 7 keer de rivier oversteken.

Een leuke variant op dit probleem is het volgende:

We hebben drie professoren en drie studenten. Deze moeten allemaal met een bootje van de ene kant van de rivier naar de andere. In het bootje kunnen hooguit twee personen. Aan beide oevers mogen de professoren niet in de meerderheid zijn, anders worden de studenten nerveus.

**Vraag:** Hoe kan iedereen naar de andere oever verplaatst worden?

Merk op dat in tegenstelling tot het boer-wolf-geit-kool-probleem hier iedereen de boot kan “besturen”. Er moet nu dus in een toestand worden aangegeven waar de boot ligt.

Er zijn 4 verschillende oplossingen, elk met 11 keer overvaren.

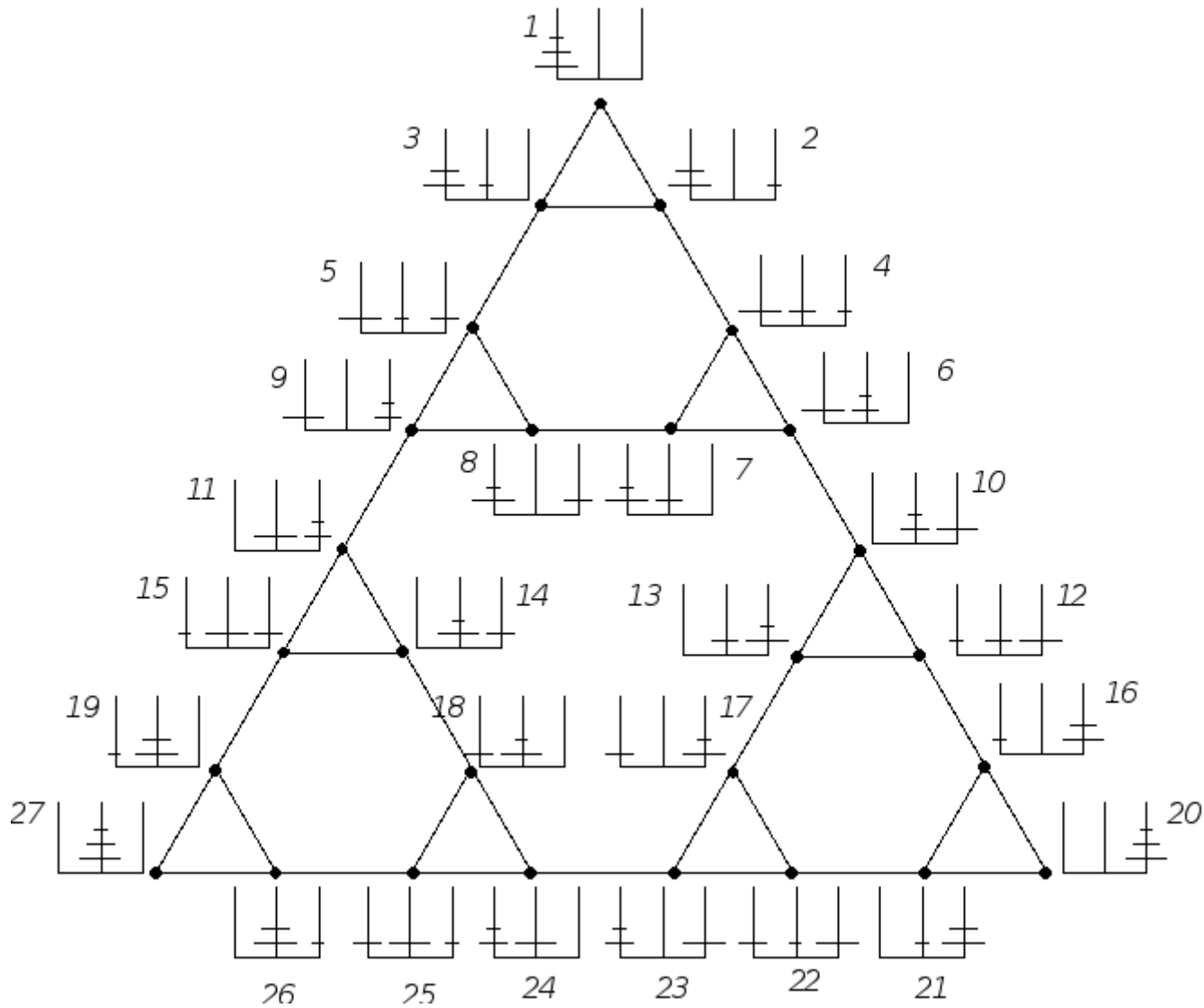
**Voorbeeld 3: Torens van Hanoi**

Gegeven  $n$  ( $n \geq 1$ ) schijven, alle verschillend in grootte, en 3 palen. In de beginsituatie liggen alle schijven boven op elkaar om één paal, waarbij er geen grotere schijf op een kleinere ligt. De andere 2 palen zijn leeg.

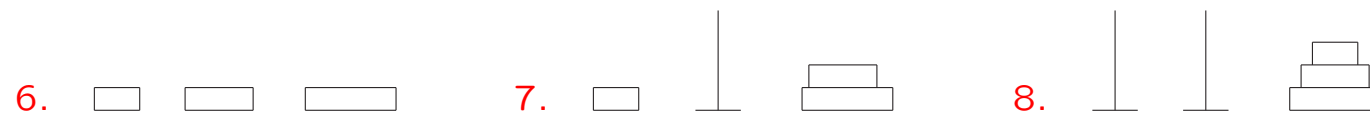
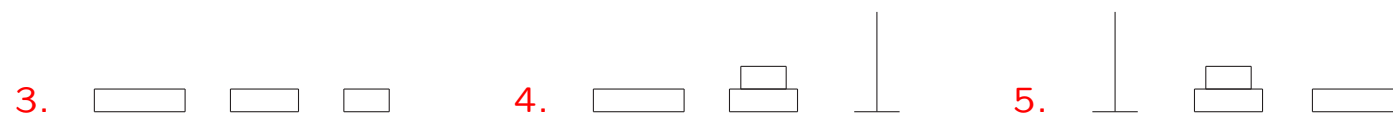
**Opdracht:** Breng de hele toren (zo snel mogelijk) naar een van de lege palen door het een voor een verplaatsen van schijven van de ene paal naar de andere. Alleen de *bovenste* schijf van een stapel kan verzet worden, en deze mag alleen *bovenop* een andere stapel gelegd worden. *Restrictie:* er mag nooit een grotere schijf op een kleinere gelegd worden.

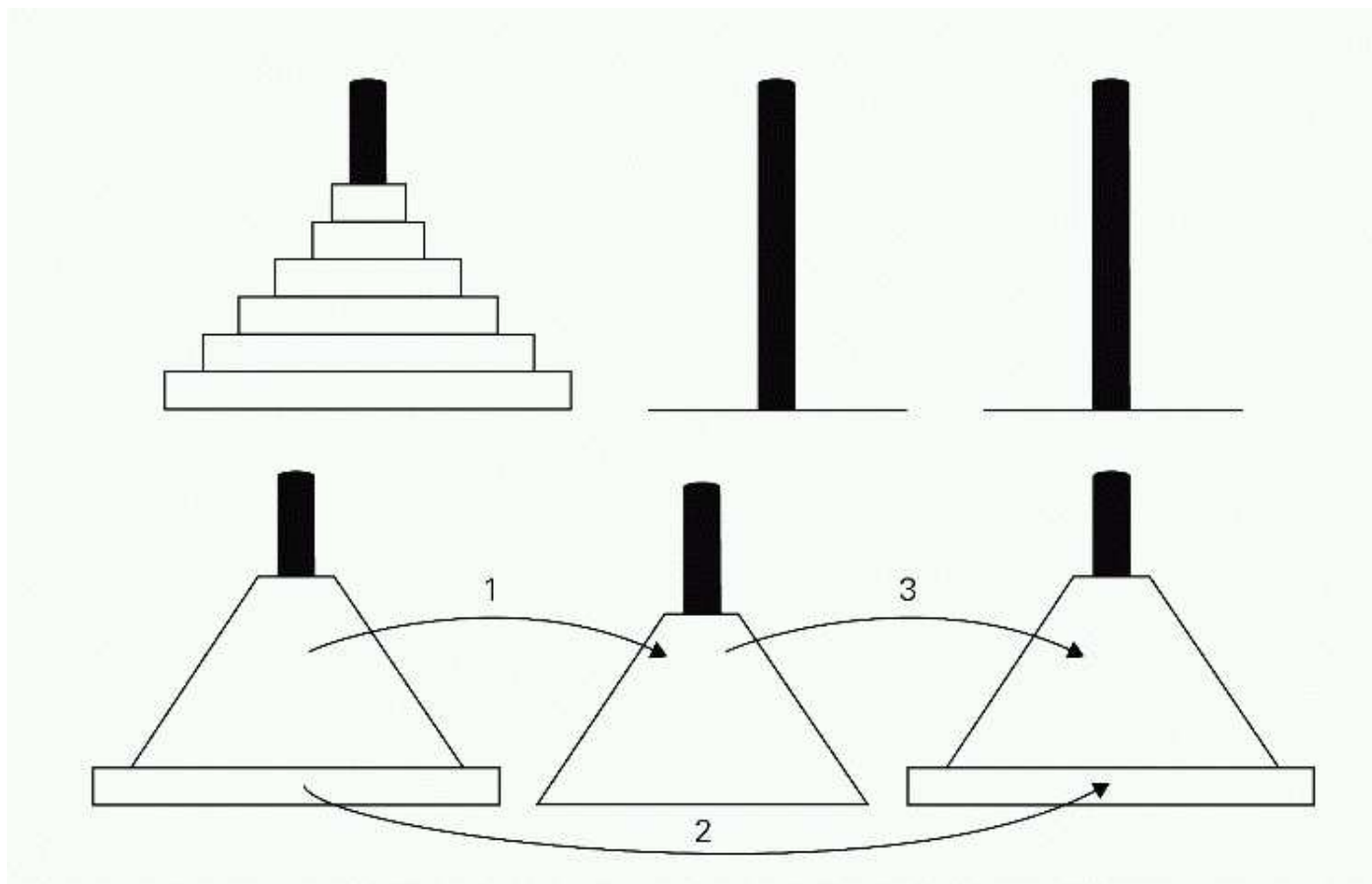
Een **toestand** is in dit geval een verdeling van de schijven over de palen, waarbij (als gevolg van de restrictie) geen grotere schijf op een kleinere ligt. Een **actie** is het verplaatsen van een schijf volgens de spelregels.

**Vraag:** Hoeveel toestanden zijn er in dit geval eigenlijk?



Optimale oplossing voor  $n = 3$ .





**Recursieve** oplossing van de Torens van Hanoi

Zie verder: Programmeermethoden, college over recursie.

# Een Intermezzo

## Voorbeeld 4: Kannenprobleem

We hebben twee kannen: een grote met een inhoud van 8 liter, en een kleine met een inhoud van 5 liter. Op de kannen staat geen maatverdeling. Verder hebben we de beschikking over een waterkraan en een afvoer. Bij aanvang zijn beide kannen leeg.

**Vraag:** Hoe krijgen we precies 4 liter water in een van de twee kannen? En liefst zo snel mogelijk.



We onderscheiden toestanden en zinvolle (!) acties:

**Toestand:** Een paar  $(x, y)$  met  $0 \leq x \leq 8$  en  $0 \leq y \leq 5$ . Hierin is  $x$  de inhoud van de grote kan en  $y$  de inhoud van de kleine kan.

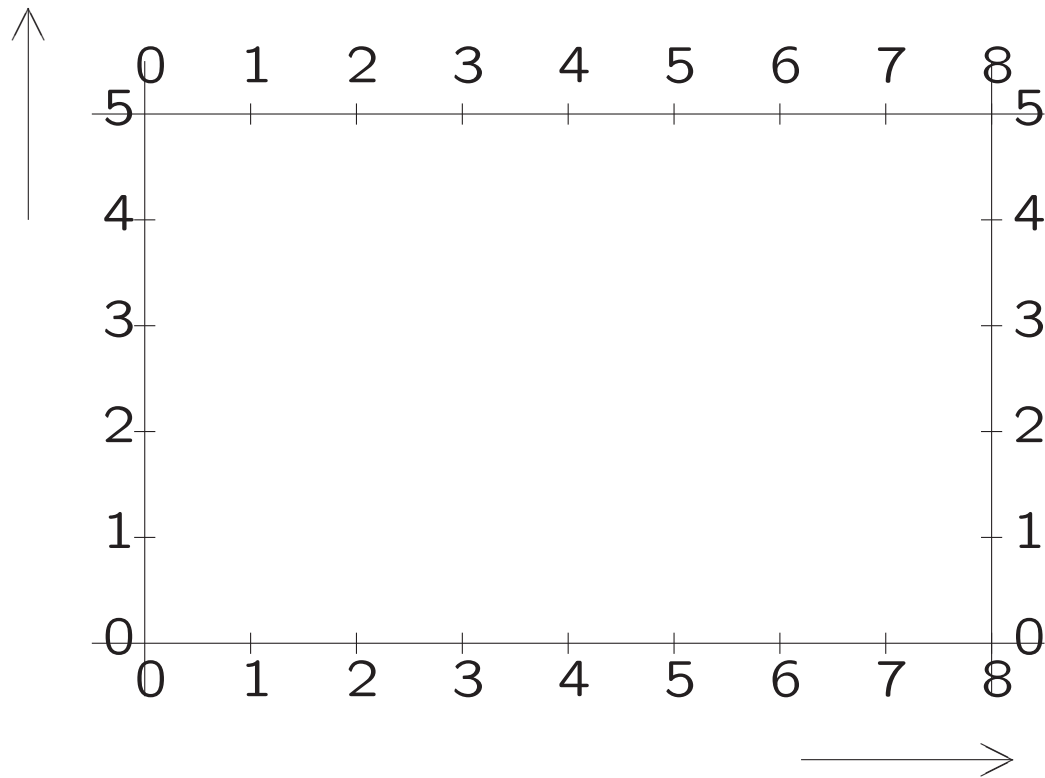
**Begintoestand:** beide kannen leeg, dus  $(0,0)$

**Eindtoestanden:** alle toestanden met 4 liter in een van beide kannen, dus  $(4, y)$  en  $(x, 4)$

**Acties:** vullen, legen en overgieten

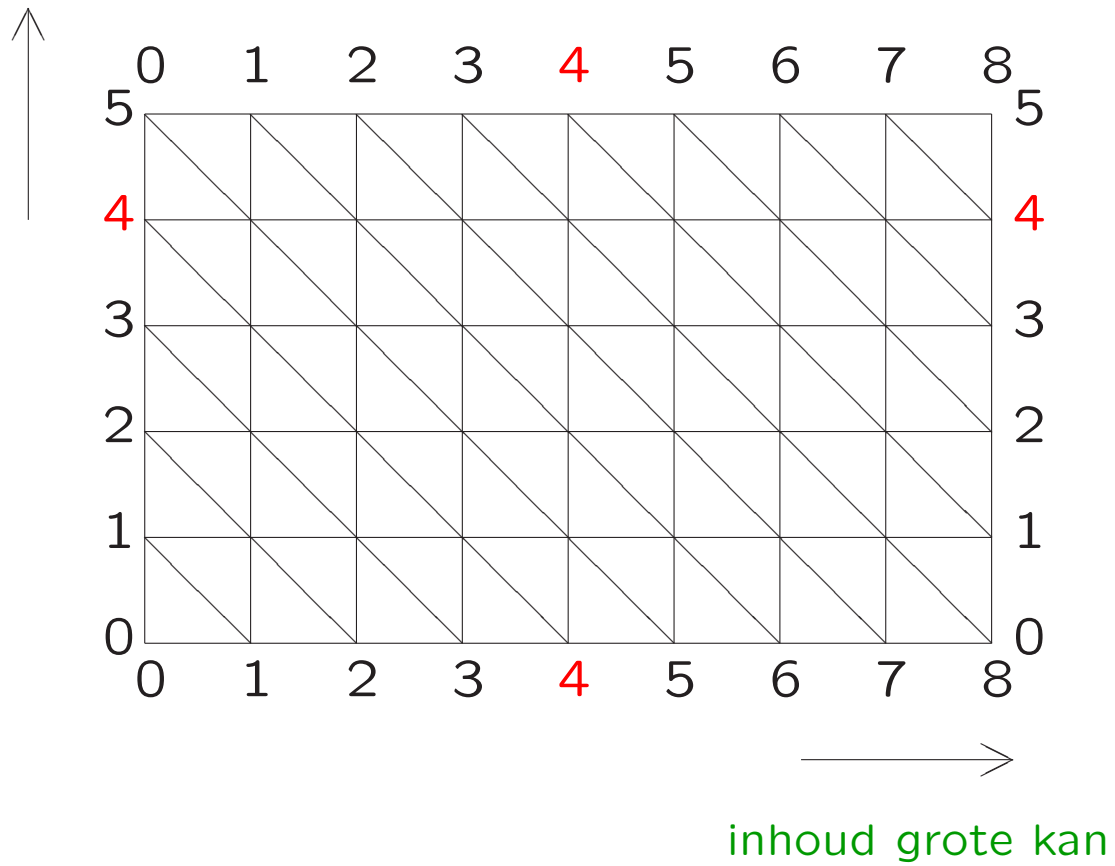
- een kan geheel (aan)vullen
- een kan geheel leeggooien
- de ene kan leeggooien in de andere
- van de ene kan in de andere gieten totdat deze vol is

inhoud kleine kan



inhoud grote kan

inhoud kleine kan





De snelste oplossing gebruikt de volgende **strategie** en zorgt voor 4 liter in de kleine kan. Er is overigens ook een (iets) langere oplossing, die 4 liter in de grote kan achterlaat.

Herhaal

Herhaal

Vul de kleine kan;

Giet over in de grote kan;

totdat de grote kan vol is

Grote kan leeggooien;

Giet uit de kleine kan over in de grote kan;

totdat oplossing gevonden

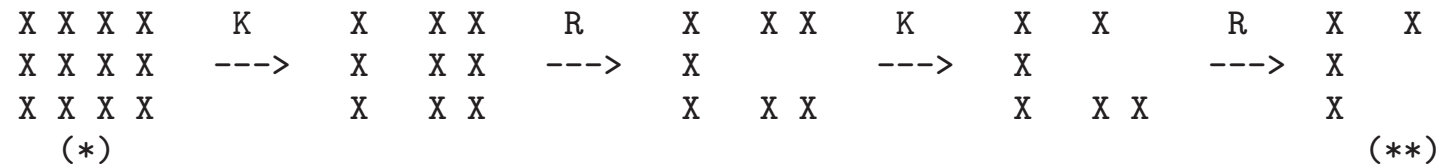
Zie verder het college.

- **Lezen/leren bij dit college uit Levitin:**  
Paragraaf 6.6 (Reduction to Graph Problems)  
Paragraaf 4.5 (Searching and Inserting in BST, The Game of Nim)
- Programmeeropdracht: maandag op de website
- **Volgend werkcollege:**  
donderdag 26 februari 2015; 13:45–15:30; zaal B1, B2
- **Uitwerking bomenpracticum:**  
binnenkort op <http://keep.liacs.leidenuniv.nl/~graafjmde/ALGO/>
- **Opgaven:**  
zie <http://keep.liacs.leidenuniv.nl/~graafjmde/ALGO/>
- **Volgend college:**  
vrijdag 27 februari 2015; 11:15–13:00; zaal B2

We bekijken een tweepersoonsspel dat wordt gespeeld met  $m * n$  dezelfde stenen, die in een  $m$  bij  $n$  rechthoek ( $m$  rijen en  $n$  kolommen) worden gelegd. Er zijn twee spelers,  $K$  (Kolom) en  $R$  (Rij), die om de beurt een kolom ( $K$ ) of een rij ( $R$ ) stenen wegnemen. We spreken af dat  $K$  begint.

Als speler  $K$  een zet doet neemt deze steeds een hele kolom weg, speler  $R$  een hele rij. Door het wegnemen van een rij of kolom uit een rechthoek zal in het algemeen die rechthoek in twee kleinere rechthoeken worden verdeeld. Indien van de rand wordt weggehaald, resteeert één kleinere rechthoek. Bij aanvang van het spel is er slechts één rechthoek, maar tijdens het spel worden dit er meer. Als een speler aan de beurt is neemt deze een kolom ( $K$ ) of rij ( $R$ ) weg uit precies één van die rechthoeken. Een zet bestaat dus uit het aanwijzen van een kolom of rij in een van de rechthoeken, en die daar vervolgens weghalen. De speler die de laatste steen/stenen weghaalt heeft gewonnen.

Een voorbeeld van een mogelijk spelverloop, met  $m = 3$  en  $n = 4$ :



In het spelverloop hierboven geven de X-en de stenen aan. In toestand/spelsituatie (\*\*) is  $K$  aan de beurt. Als deze in de eerstvolgende zet de meest linker kolom weghaalt verliest hij. Veronderstel dus dat hij de derde kolom (één steen dus) weghaalt. Dan blijft er een toestand over die winnend is voor  $R$ . (Zij neemt namelijk de middelste rij (één steen) en haalt dan in haar volgende zet de laatste steen weg.) Stand (\*\*) is derhalve verliezend voor  $K$  (ofwel winnend voor  $R$ ), want zijn twee mogelijke zetten leiden allebei tot verlies.