

Tweede college algoritmiek

16 februari 2007

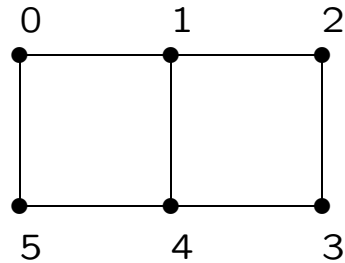
Bomen en toestand-actie-ruimte

Adjacency matrix: de **ongerichte** graaf wordt gerepresenteerd door een tweedimensionaal array `int inhoud[n][n]` (n het aantal knopen), waarbij `inhoud[i][j]` aangeeft of er een tak tussen i en j zit.

Adjacency list: de **ongerichte** graaf wordt gerepresenteerd door een eendimensionaal array `buur* inhoud[n]` (n het aantal knopen), waarbij `inhoud[i]` de ingang is tot een lijst van burens van knoop i .

Adjacency list representatie in C++:

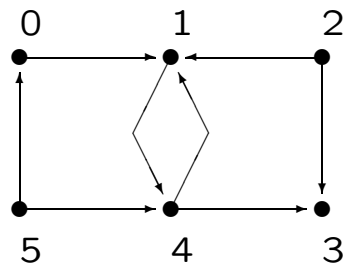
```
struct buur {  
    int knoopnummer;  
    buur* volgende;  
}  
buur* inhoud[n];
```



1.

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

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



2.

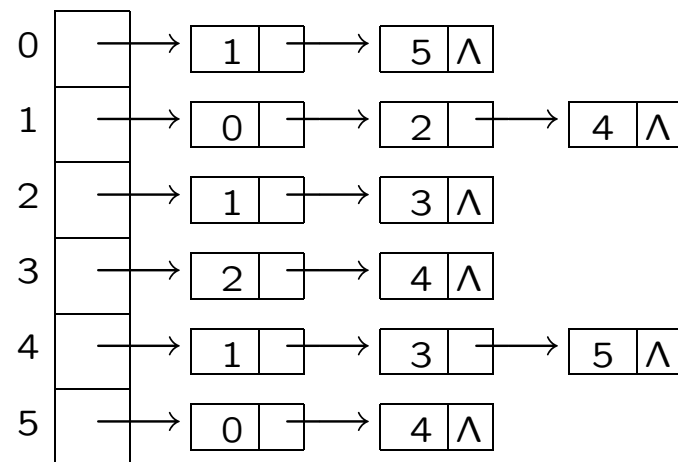
$$V = \{0, 1, 2, 3, 4, 5\};$$

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

Adjacency matrix en adjacency list voor voorbeeldgraaf **1**:

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

adjacency matrix



adjacency list

Geef een algoritme dat bepaalt of er in een gegeven ongerichte graaf hooguit twee knopen zijn met oneven graad.

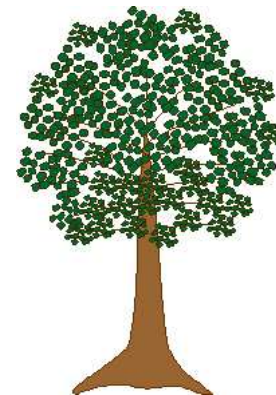
```
// het array graaf (buur* graaf[n]) bevat de ongerichte graaf
totaal = 0;
for ( knoop = 0; knoop < n; knoop++ ) {
    teller = 0;
    hulp = graaf[knoop];
    while ( hulp != null ) { // buurlijst aflopen
        teller++;
        hulp = hulp->volgende;
    }
    if ( teller%2 == 1 ) // oneven
        totaal++;
} // for
if ( totaal <= 2 )
    cout << " hooguit twee ... " << endl;
else
    cout << " meer dan twee ... " << endl;
```

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

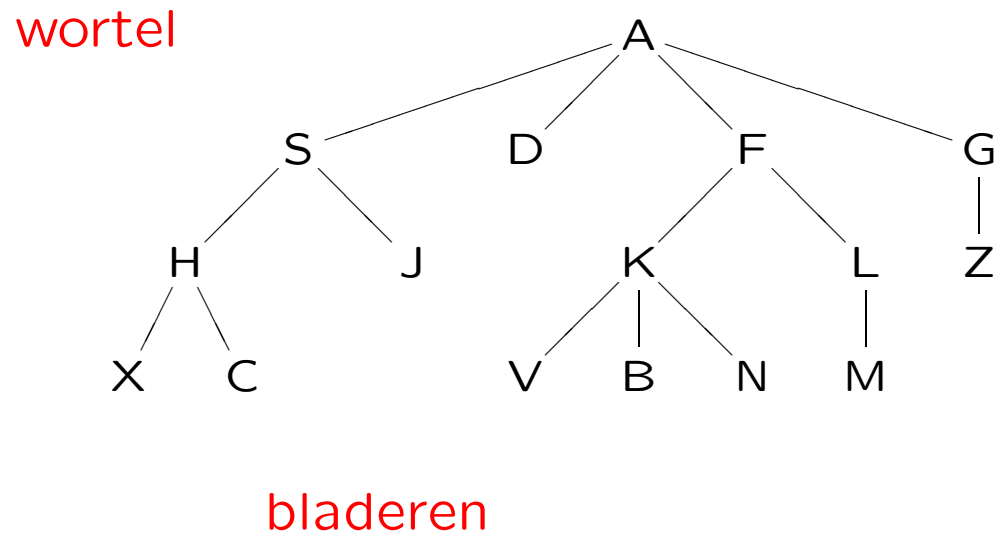
Wijs een speciale knoop aan, de *wortel*. Teken de wortel bovenaan en alle paden vanuit de wortel naar beneden: dit geeft een hiërarchische structuur die lijkt op een stamboom. Dit heet ook wel een **georiënteerde boom**. Meestal spreken we gewoon van een boom.

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

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



Terminologie: takken, knopen, nivo, hoogte, wortel, bladeren \longleftrightarrow interne knopen

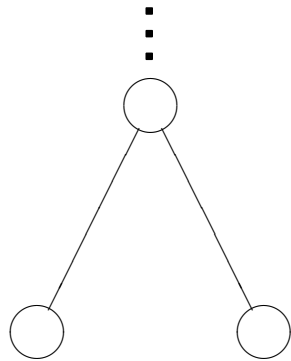


De **wortel** (hier A) is de *enige* ingang tot de boom.

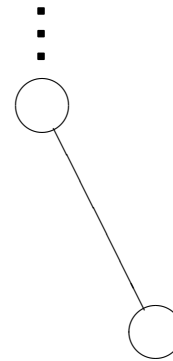
- Een acyclische graaf die niet samenhangend is heet een **bos**. Het is namelijk een collectie bomen.
- Voor bomen geldt: aantal takken = aantal knopen - 1.
- Een **geordende boom** is een boom waarin van elke knoop de kinderen geordend zijn: oudste kind, een na oudste kind, ..., jongste kind.
- Onderstaande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

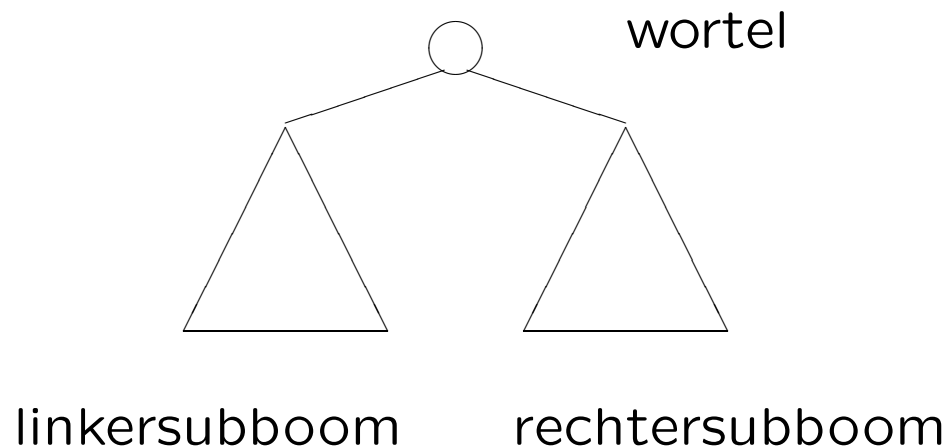


linkerkind rechterkind



één kind: rechterkind

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



```
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](#)

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
```

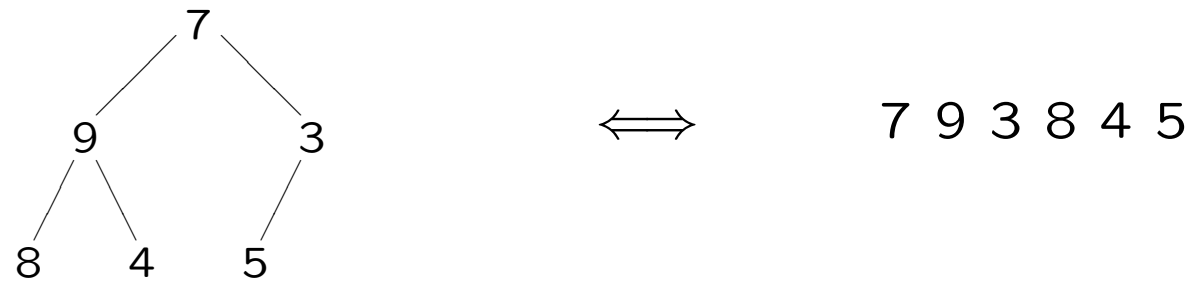
Merk op dat hier eigenlijk een preorde-wandeling wordt gedaan.

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

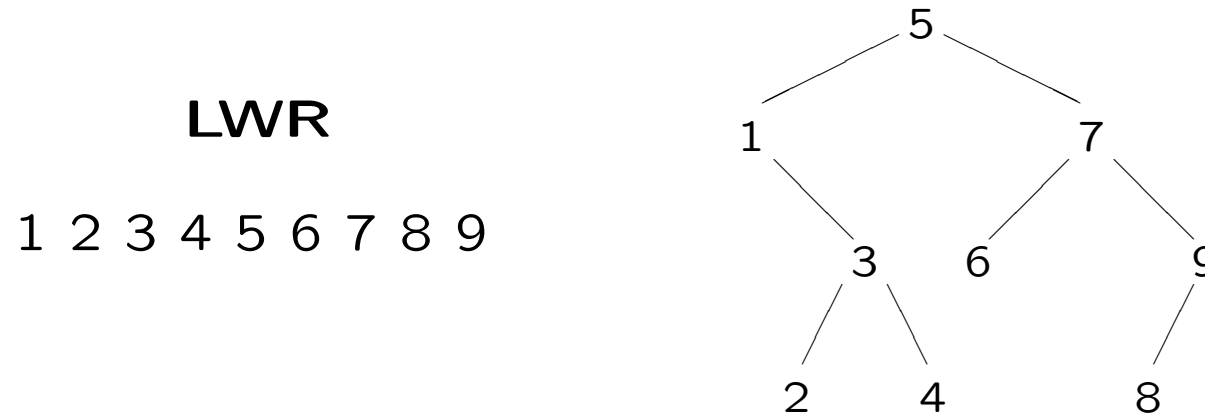
```
void breekaf (knoop* & root) {
    if ( root != NULL ) {
        breekaf (root->links);      L
        breekaf (root->rechts);    R
        delete root;              W
        root = NULL;
    } // if
} // breekaf
```

- Voor de hoogte h van een binaire boom met n knopen geldt: $\lfloor \log_2 n \rfloor \leq h \leq n - 1$
- 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.
- 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)

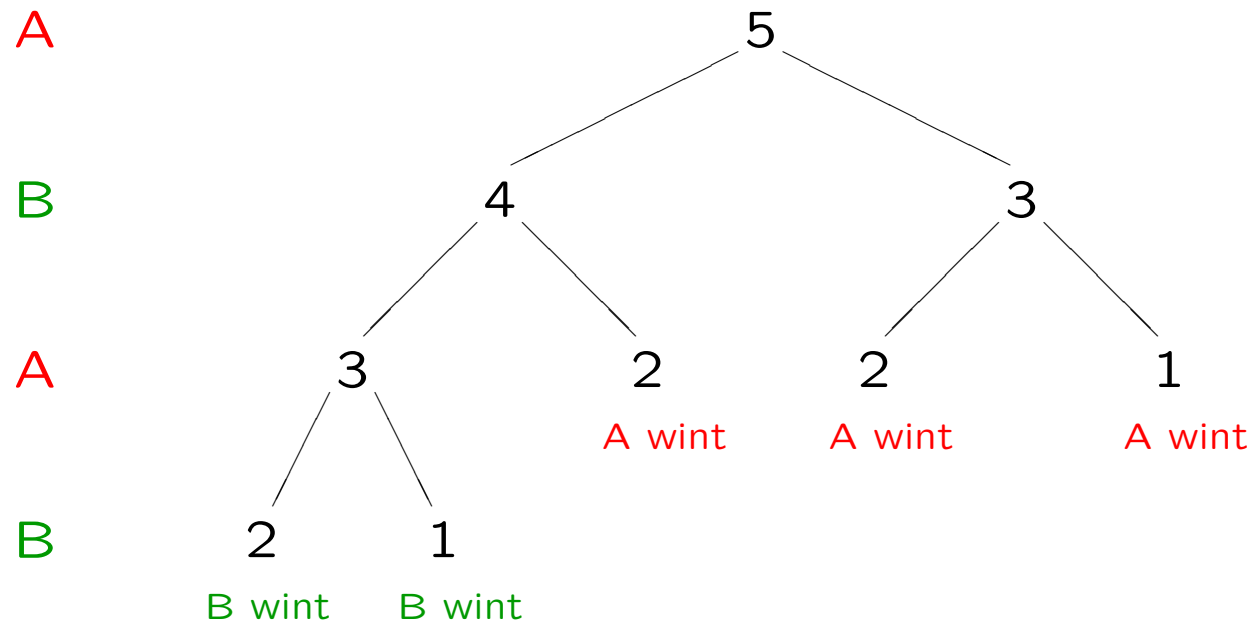
- *Bestaat uit* alle mogelijke **toestanden en acties**
- Begintoestand, eindtoestand(en)
- Een actie veroorzaakt een overgang van de ene (toegelaten) toestand naar een andere
- *Oplossing* van het probleem: een opeenvolging van acties die van de begintoestand naar een eindtoestand leiden

Voorbeeld 1: NIM

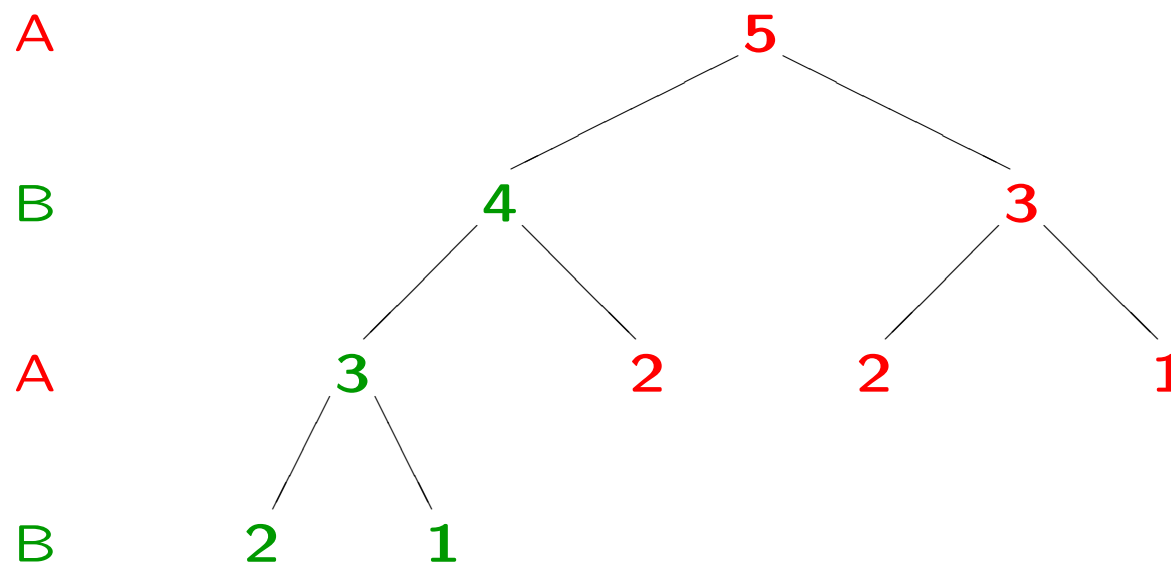
- We beginnen met één stapel van n lucifers (begintoestand)
- Er zijn twee spelers: **A** en **B**
- De spelers pakken om de beurt 1 of 2 lucifers
- Het spel is afgelopen als er geen lucifers meer op de stapel liggen
- De speler die de laatste lucifer(s) pakt heeft gewonnen

Gevraagd: is het spel winnend voor degene die begint?

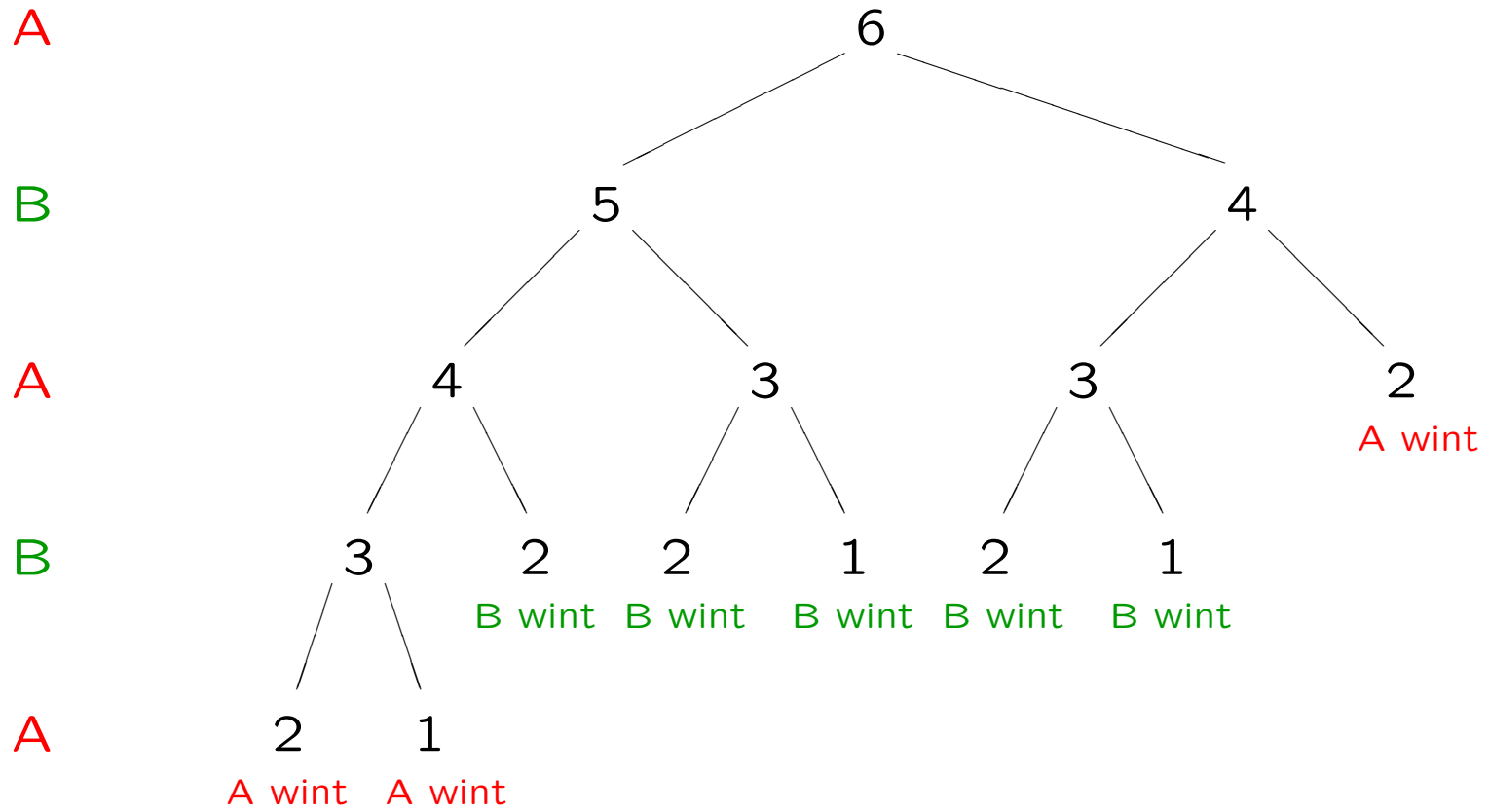
toestand-actie-ruimte (spelboom) $n = 5$



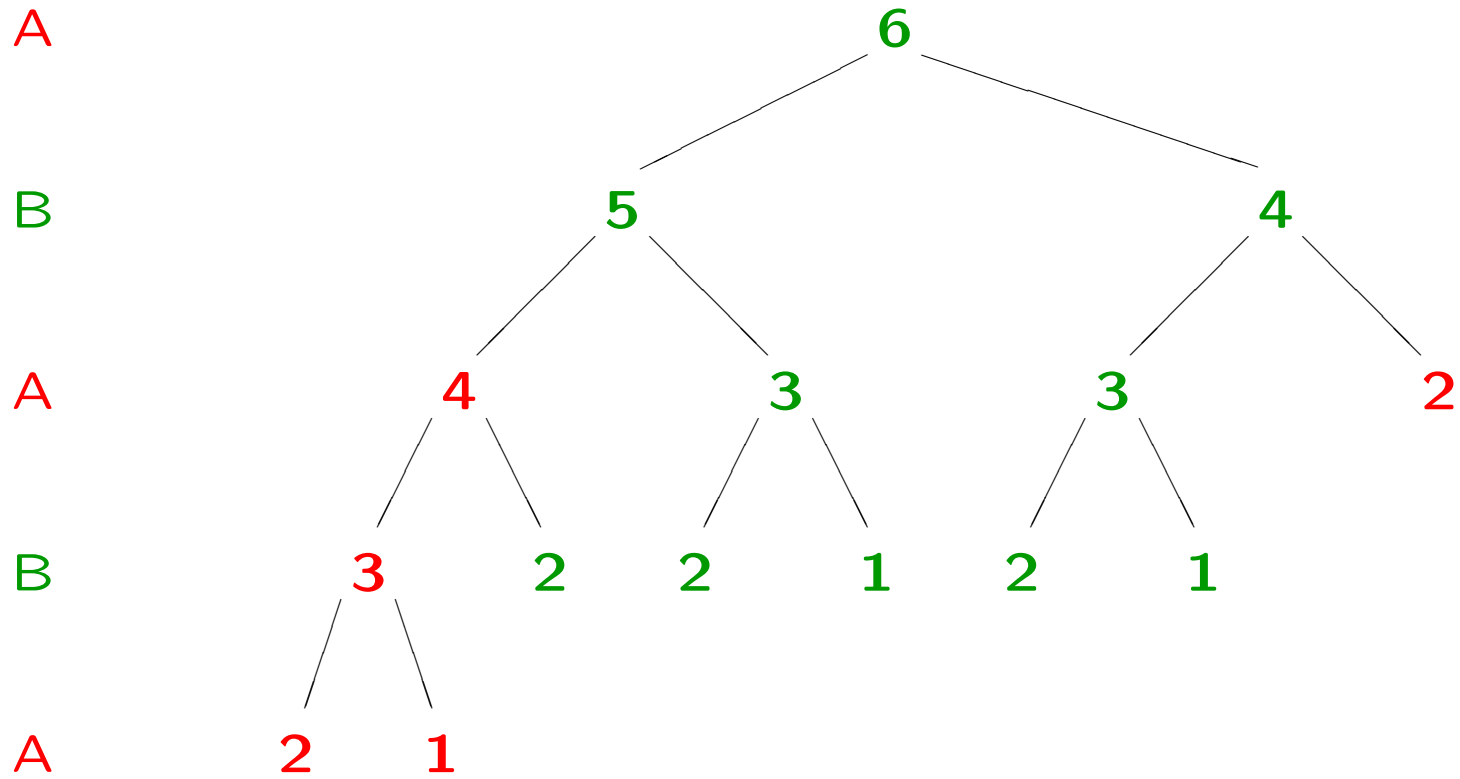
Winnend voor A.
 Winnende zet: 2 lucifers wegnemen



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



A kan niet winnen (bij perfect spel van B)

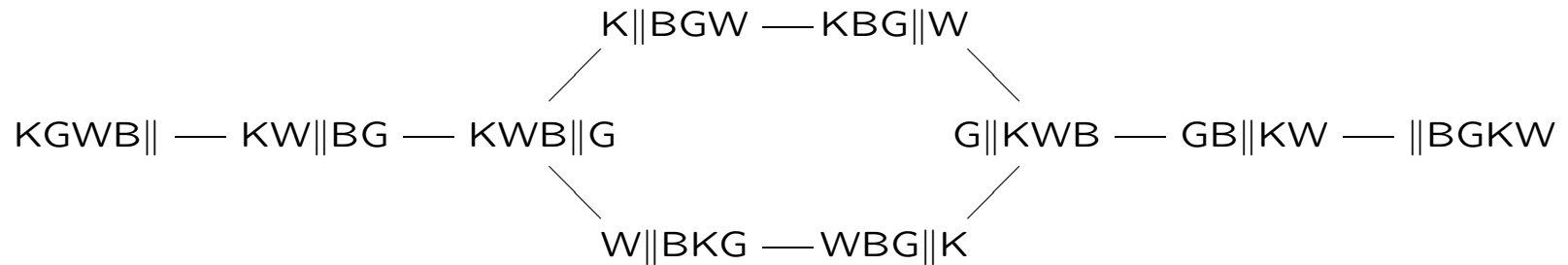


Voorbeeld 2: Old world puzzle (Ex. 1.2.1.)

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.

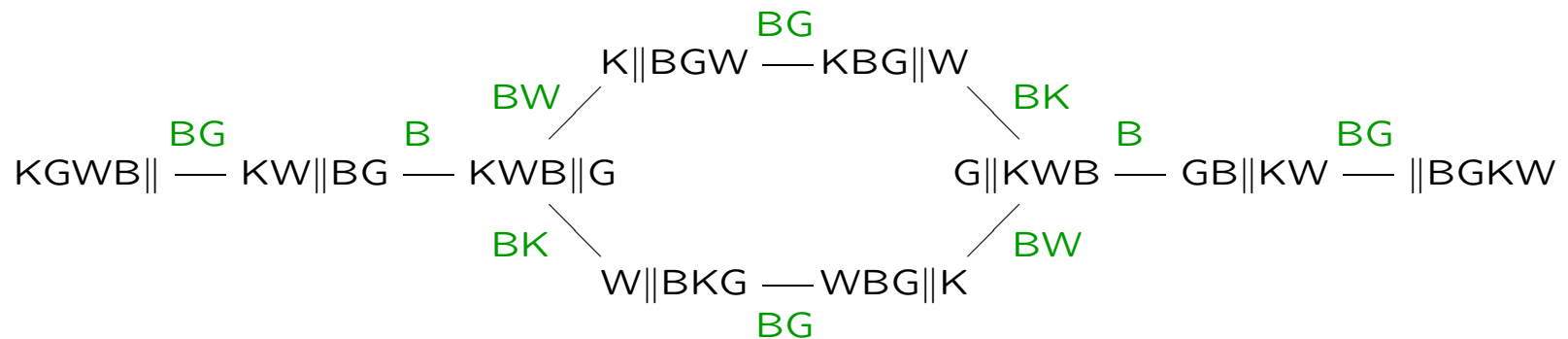
Vraag: Hoe kan alles naar de andere oever verplaatst worden?





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.

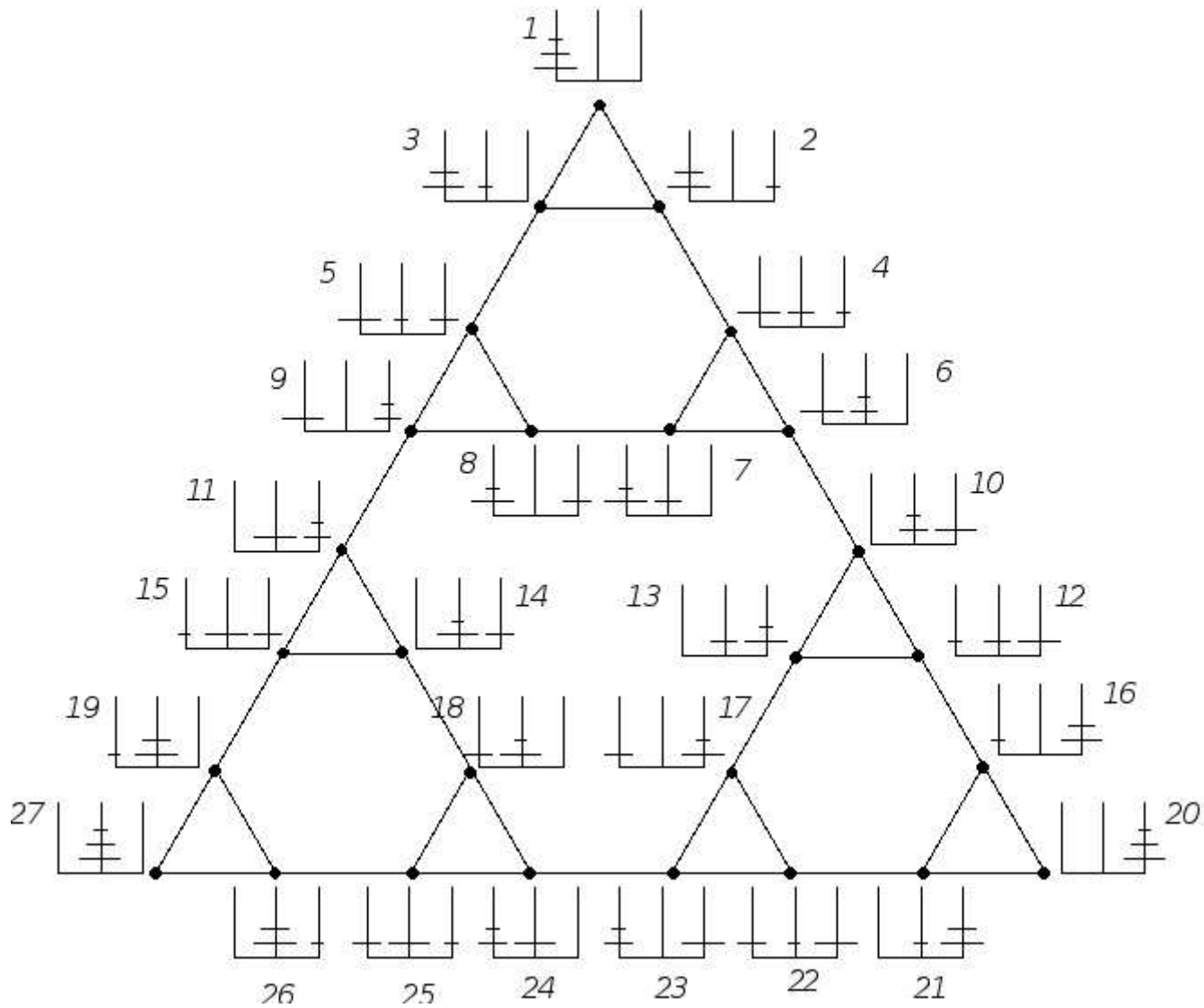
De oplossing is een **kortste pad** van de begintoestand naar de eindtoestand: hier zijn er twee, bij beide moet de boer 7 keer de rivier oversteken.

Voorbeeld 3: Torens van Hanoi

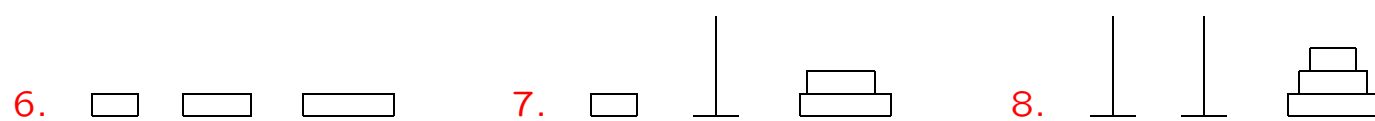
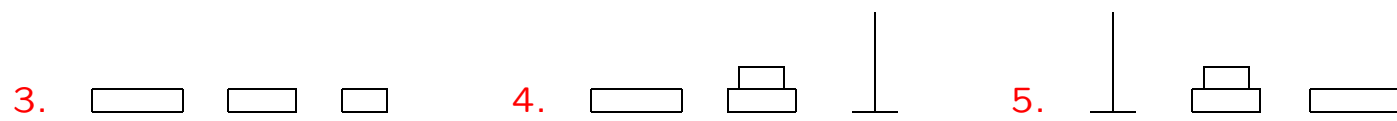
Gegeven n ($n \geq 1$) schijven, alle verschillend in grootte, en 3 palen. In de beginsituatie liggen alle schijven boven op elkaar om één paal, waarbij er geen grotere schijf op een kleinere ligt. De andere 2 palen zijn leeg.

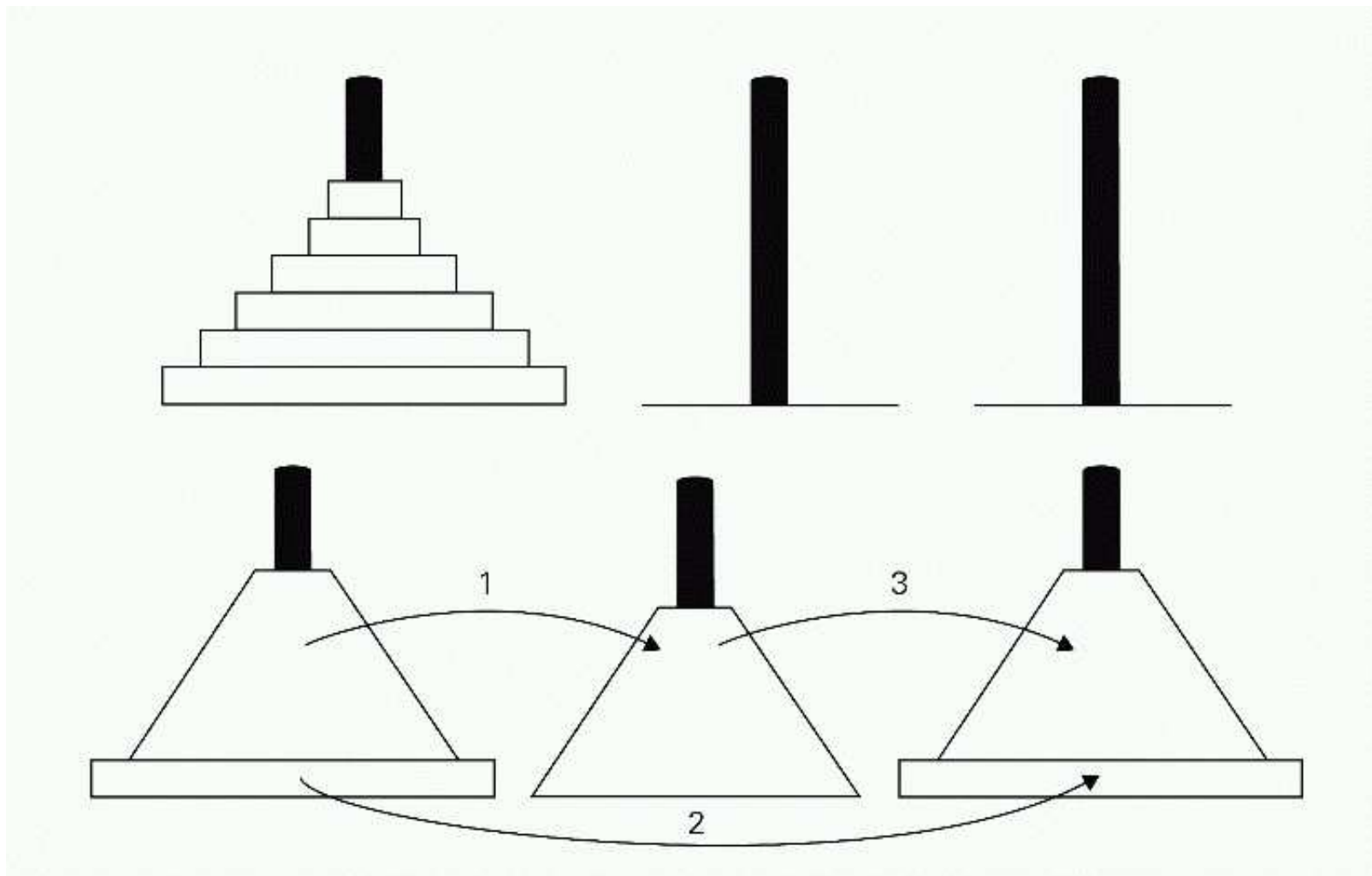
Opdracht: Breng de hele toren (zo snel mogelijk) naar een van de lege palen door het een voor een verplaatsen van schijven van de ene paal naar de andere. Alleen de *bovenste* schijf van een stapel kan verzet worden, en deze mag alleen *bovenop* een andere stapel gelegd worden. *Restrictie:* er mag nooit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



Optimale oplossing voor $n = 3$.





Recursieve oplossing van de Torens van Hanoi

- **Werkcollege:**

donderdag 22 februari 2007 in computerzaal 306/308

- **Opgaven:**

zie <http://www.liacs.nl/home/graaf/ALGO/algo2007.html>

- **Volgend college:**

vrijdag 23 februari 2007