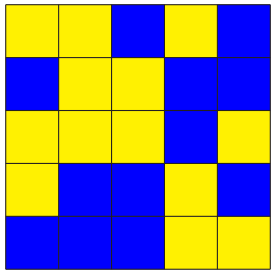
Vierde college algoritmiek

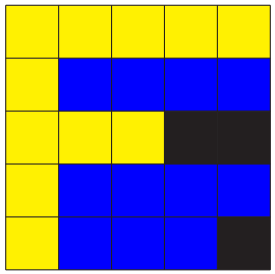
28 februari 2023

Toestand-actie-ruimte

Brute Force

recursie





eind deze week / komend weekend

Voorbeeld 3: Torens van Hanoi

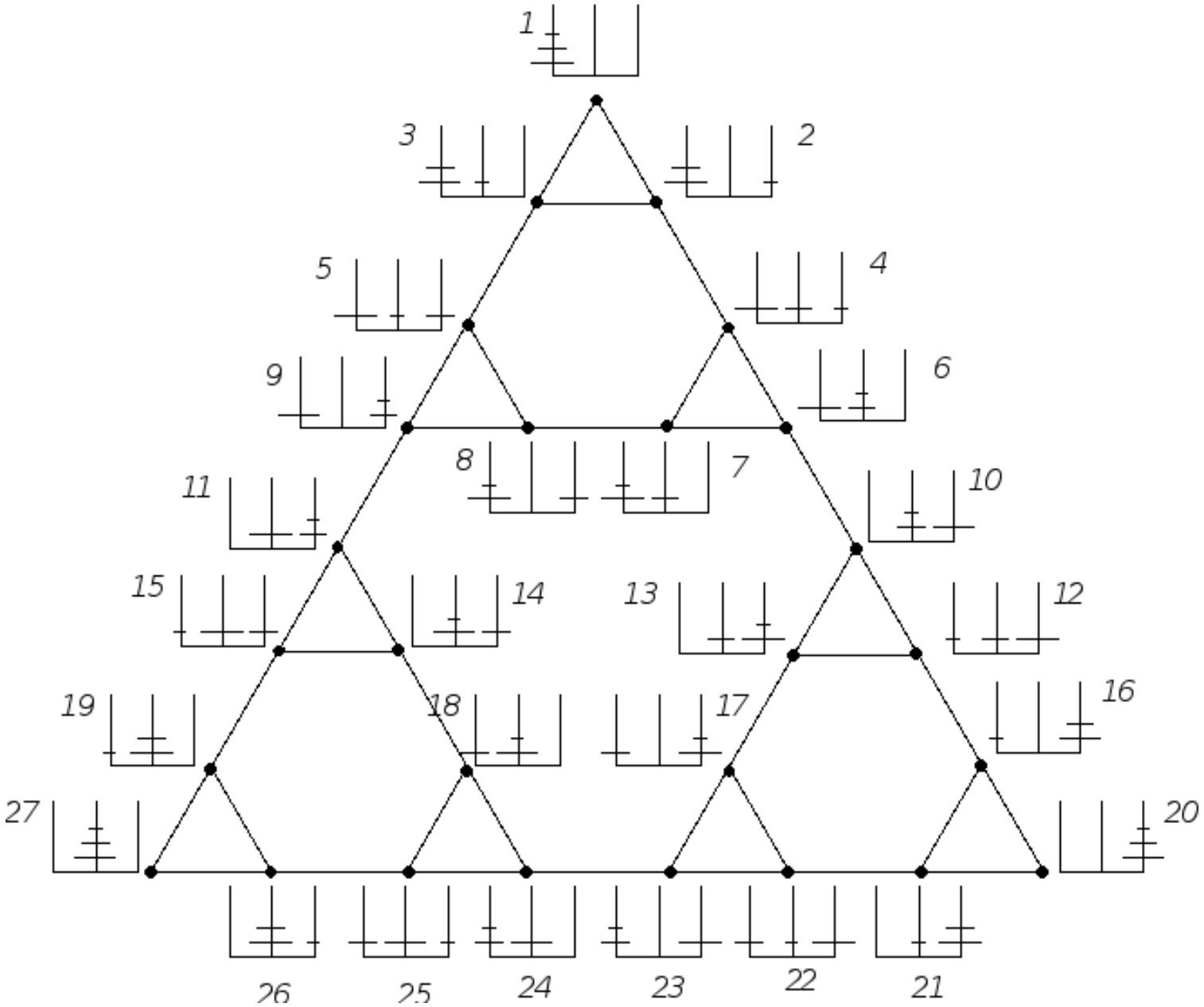
Gegeven n ($n \geq 1$) schijven, alle verschillend in grootte, en 3 palen. In de beginsituatie liggen alle schijven boven op elkaar om één paal, waarbij er geen grotere schijf op een kleinere ligt. De andere 2 palen zijn leeg.

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.

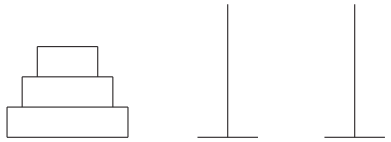
Regels:

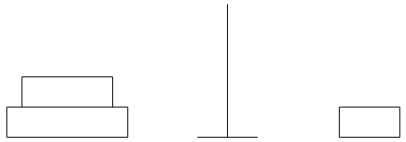
- Alleen de *bovenste* schijf van een stapel kan verzet worden
- deze mag alleen *bovenop* een andere stapel gelegd worden.
- *Restrictie*: er mag nooit een grotere schijf op een kleinere gelegd worden.

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.

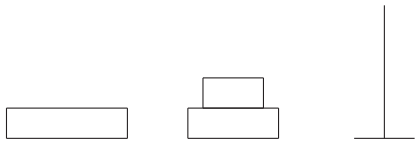

1
3
2
5
4
9
6
8
7
11
10
15
14
13
12
19
18
17
16
27
20
26
25
24
23
22
21

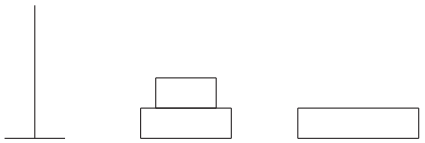
Optimale oplossing voor $n = 3$.

1. 

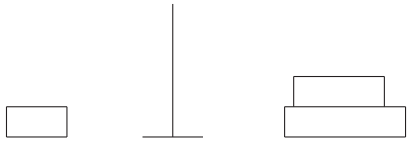
2. 

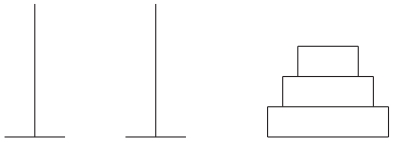
3. 

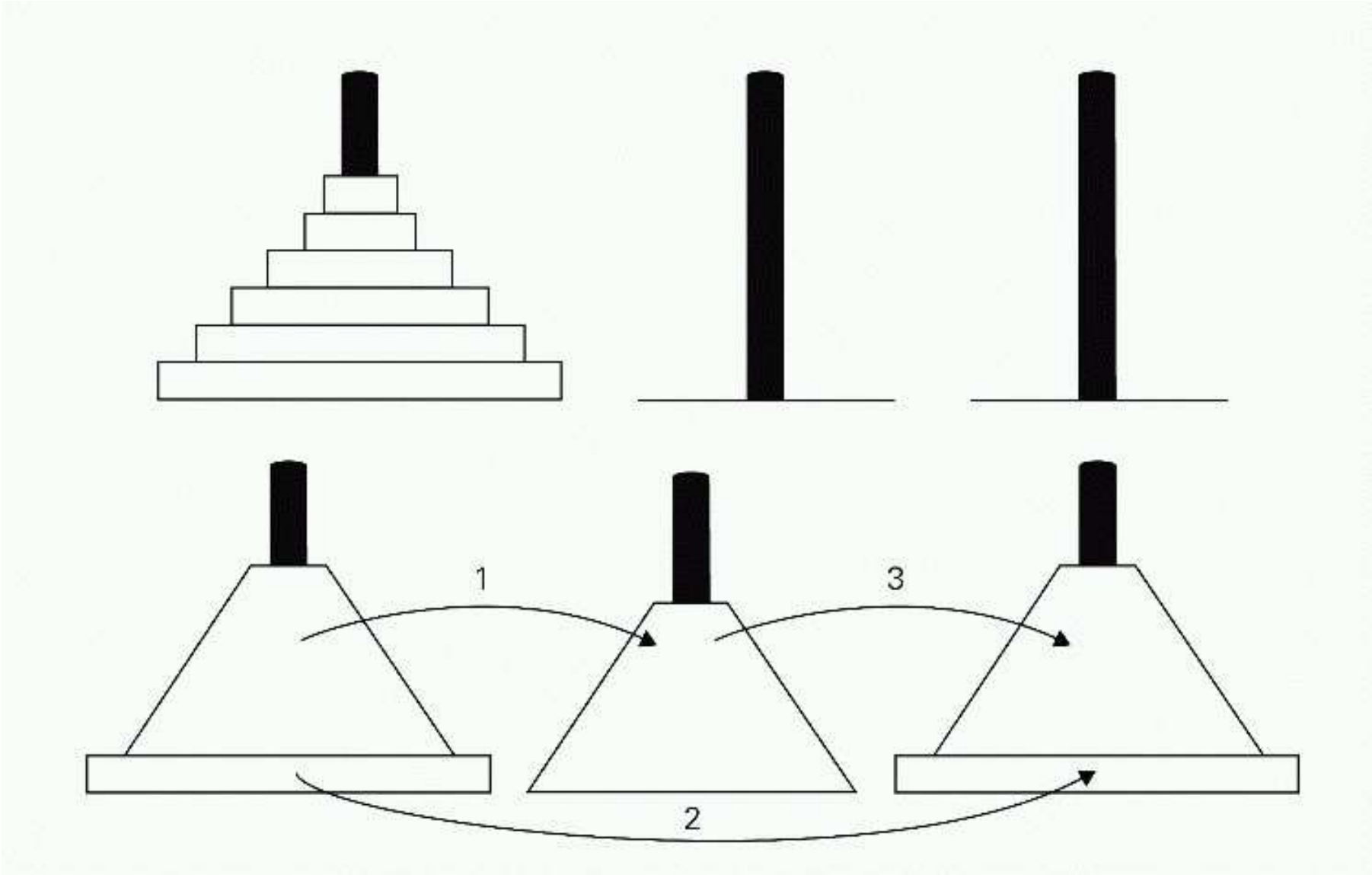
4. 

5. 

6. 

7. 

8. 



Recursieve oplossing van de Torens van Hanoi

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.

Een Intermezzo

Die Hard

We onderscheiden toestanden en zinvolle (!) acties:

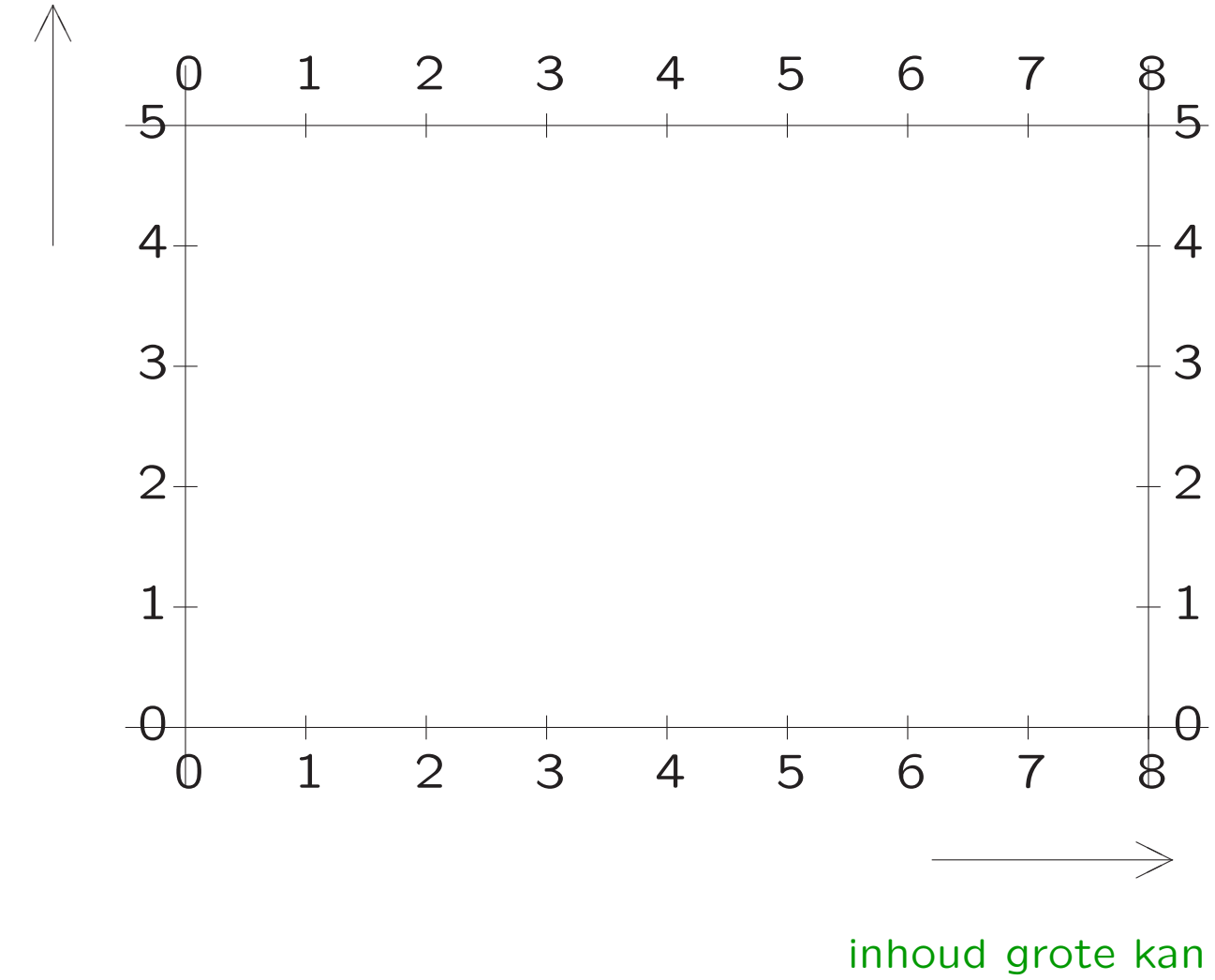
Toestand: Een paar (x, y) met $0 \leq x \leq 8$ en $0 \leq y \leq 5$. Hierin is x de inhoud van de grote kan en y de inhoud van de kleine kan.

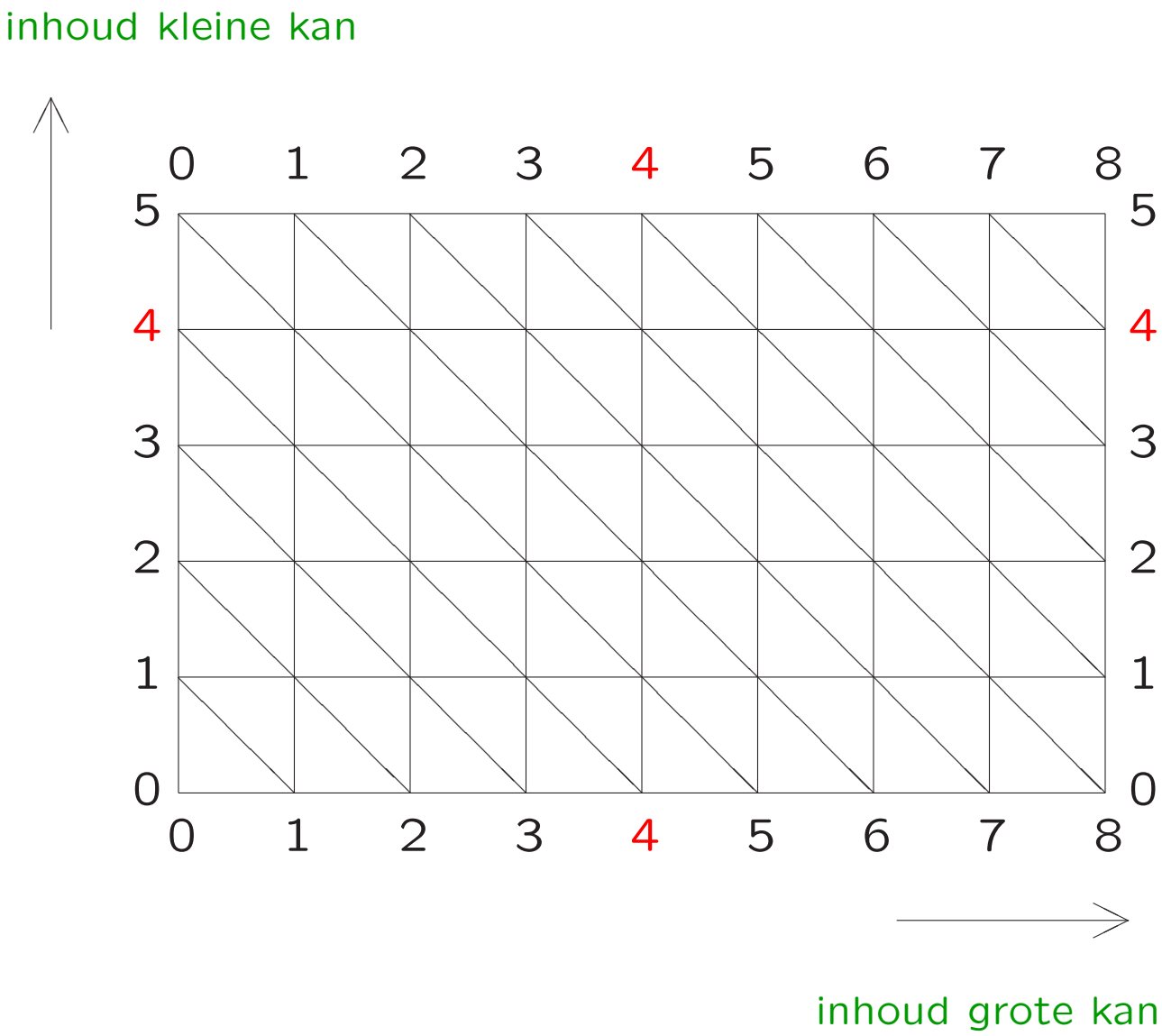
Begintoestand: beide kannen leeg, dus (0,0)

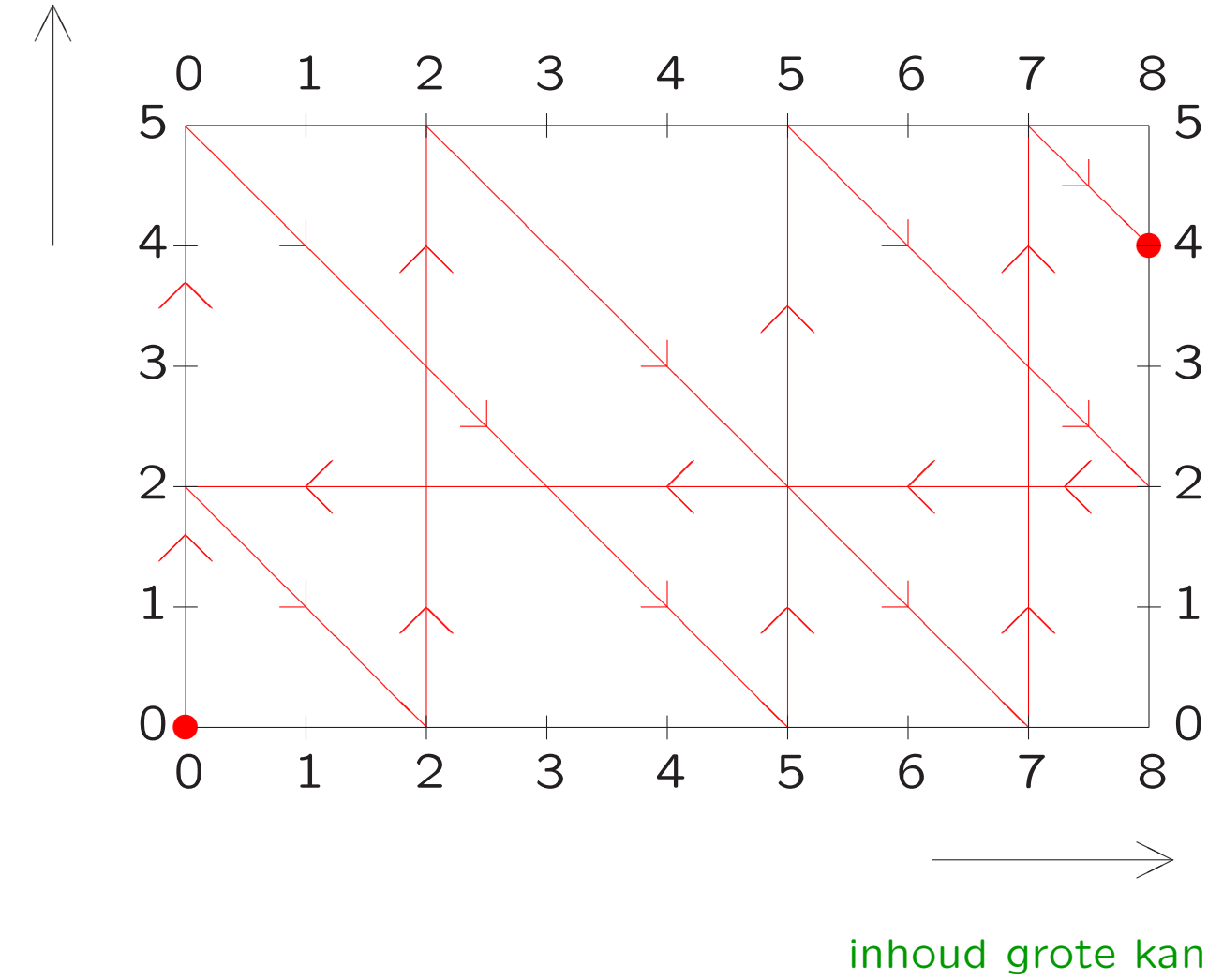
Eindtoestanden: alle toestanden met 4 liter in een van beide kannen, dus (4, y) en (x, 4)

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
0 1 2 3 4 5 6 7 8
5
4
3
2
1
0
inhoud grote kan


inhoud kleine kan
0 1 2 3 4 5 6 7 8
5
4
3
2
1
0
inhoud grote kan


inhoud kleine kan
0 1 2 3 4 5 6 7 8
5
4
3
2
1
0
inhoud grote 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) of (oplossing gevonden)

    Als nog geen oplossing gevonden
        Grote kan leeggooien;
        Giet uit de kleine kan over in de grote kan;
totdat oplossing gevonden
```

Wanneer oplossing mogelijk?

Brute Force

Brute force: a straightforward approach, usually directly based on the problem statement and definitions.

Ofwel: los een probleem op via de meest voor de hand liggende (recht-toe-recht-aan) methode, meestal door eenvoudigweg de definitie van een oplossing te gebruiken. Vaak ook: alle mogelijkheden proberen.

Voorbeeld 1: vind de grootste gemene deler van twee getallen m en n door van alle mogelijke integers ≥ 2 (en $\leq \min(m, n)$) te proberen of ze zowel m als n delen. (Zie college 1.)

Voorbeeld 2: los de DONALD + GERALD = ROBERT puzzel op door alle 9! (er was al gegeven dat D = 5) mogelijke antwoorden te proberen. (Zie college 1.)

Voorbeeld 3: zoek een gegeven X in een array van n stuks door er van links naar rechts doorheen te lopen en X met alle n te vergelijken. (Zie college 3.)

Zoek herhaald de kleinste en zet die op de juiste positie in het array.

```
for i := 0 to n − 2 do
    min := i;
    for j := i + 1 to n − 1 do
        if A[j] < A[min] then
            min := j;
        fi
    od
    wissel(A[i], A[min]);
od
```

Aantal vergelijkingen: $\frac{1}{2}n(n-1)$.

Probleem: bereken de waarde van het polynoom $p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ in het punt $x = x_0$ (Levitin, opgave 3.1.4)

Brute force algoritme (uit de definitie):

```
p := 0;
for i := n downto 0 do
    macht := 1;
    for j := 1 to i do
        macht := macht * x; // bereken x^i
    od
    p := p + a[i] * macht;
od
return p;
```

Efficiëntie (aantal $*/+$): $\Theta(n^2)$

Slimmer: we kunnen de efficiëntie eenvoudig flink verbeteren door van rechts naar links te evalueren en de x^i handiger te berekenen:

```
p := a[0];
macht := 1;
for i := 1 to n do
    macht := macht * x;
    p := p + a[i] * macht;
od
return p;
```

Efficiëntie: $\Theta(n)$;
Preciezer: $\#(*) = 2n$; $\#(+) = n$

Dit kan nog beter (methode van Horner), echter niet in orde van grootte.

Gegeven een patroon (= string van m karakters) en een tekst (= string van $n \geq m$ karakters). **Gevraagd** de index van de beginpositie in de tekst waar het patroon voorkomt.

Brute force algoritme: patroon v.l.n.r. langs de tekst schuiven en steeds de overeenkomstige karakters uit tekst en patroon vergelijken

```
for i := 0 to n − m do
    j := 0;
    while j < m and patroon[j] = tekst[i + j] do
        j := j + 1;
    od
    if j = m then
        return i;
    fi
od
return −1; // geen match gevonden
```

De werking van het algoritme geïllustreerd aan de hand van het volgende voorbeeld:

```
N  O  B  O  D  Y  -  N  O  T  I  C  E  D  -  H  I  M
N  O  T
   N  O  T
      N  O  T
         N  O  T
            N  O  T
               N  O  T
                  N  O  T
                     N  O  T
```

Het aantal vergelijkingen dat dit algoritme doet hangt af van de tekst en het patroon. In de worst case worden $m * (n - m + 1)$ vergelijkingen gedaan. Dit komt voor wanneer in elke i-stap het patroon helemaal (dus m vergelijkingen) vergeleken wordt met de tekst. De complexiteit van het algoritme is dus $O(n * m)$.

Opgave: geef een tekst en een patroon waarvoor het algoritme $m * (n - m + 1)$ vergelijkingen doet.

Opmerking: het kan beter (Boyer-Moore, Knuth-Morris-Prattt), maar voor "gewone-taal" teksten is het algoritme zo slecht nog niet.

Opmerking 2: `string::find (...)` gebruikt brute force

Opmerking 3: BAPC2006

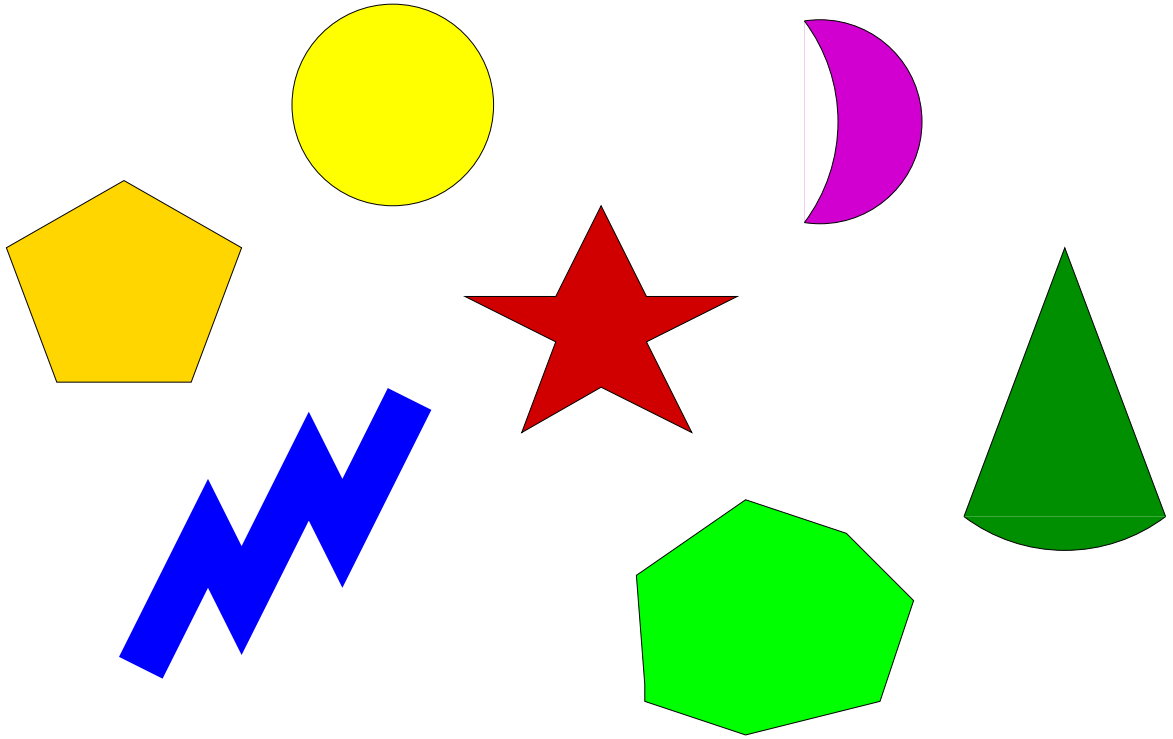
Gegeven n punten $p_1 = (x_1, y_1), \ldots, p_n = (x_n, y_n)$.
Gevraagd het/een tweetal punten dat het dichtst bij elkaar ligt. Afstandsmaat: $\mathrm{d}(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

Brute force algoritme: alle paren (p_i, p_j) (met $i < j$) aflopen en hun onderlinge afstanden $\mathrm{d}(p_i, p_j)$ vergelijken.

```
dmin := ∞;
for i := 1 to n − 1 do
    for j := i + 1 to n do
        d := (x_i − x_j)^2 + (y_i − y_j)^2;
        if d < dmin
            dmin := d; k := i; l := j;
        fi // (p_k, p_l) voorlopig closest pair
    od
od
```

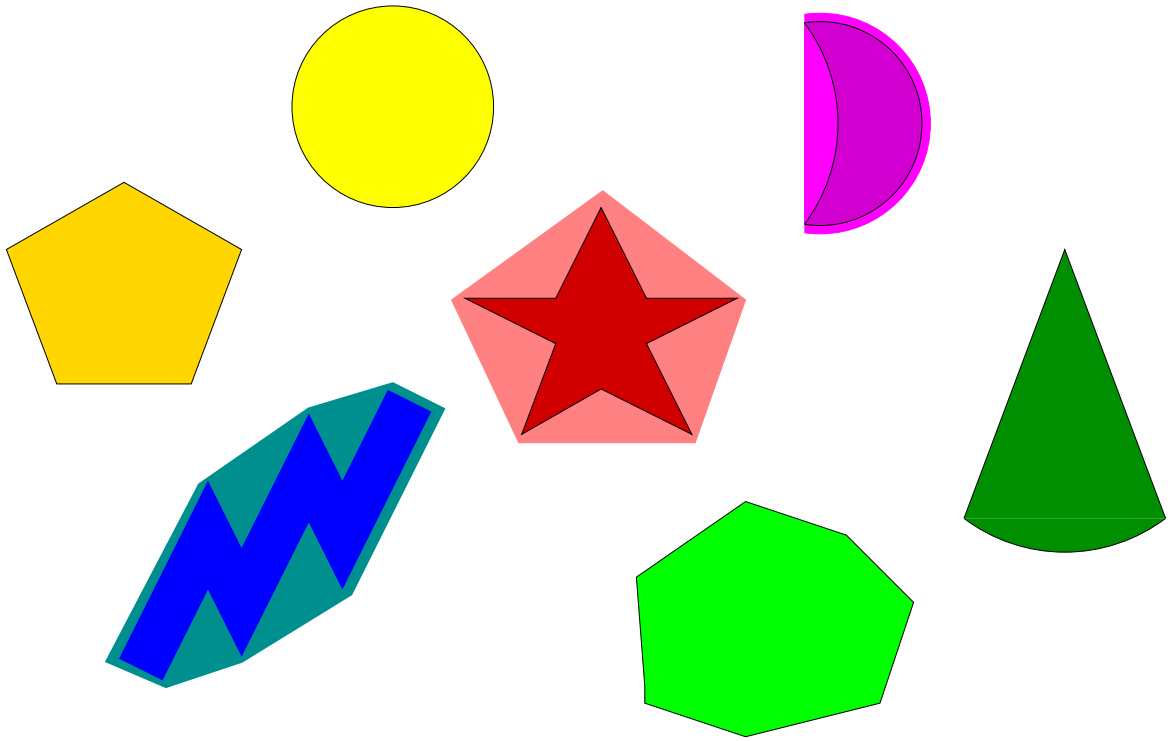
Complexiteit: $\frac{1}{2}n(n-1) = \Theta(n^2)$

Een verzameling punten in het platte vlak heet convex als voor elk tweetal punten uit die verzameling geldt dat het verbindend lijnstuk ook weer in die verzameling ligt.



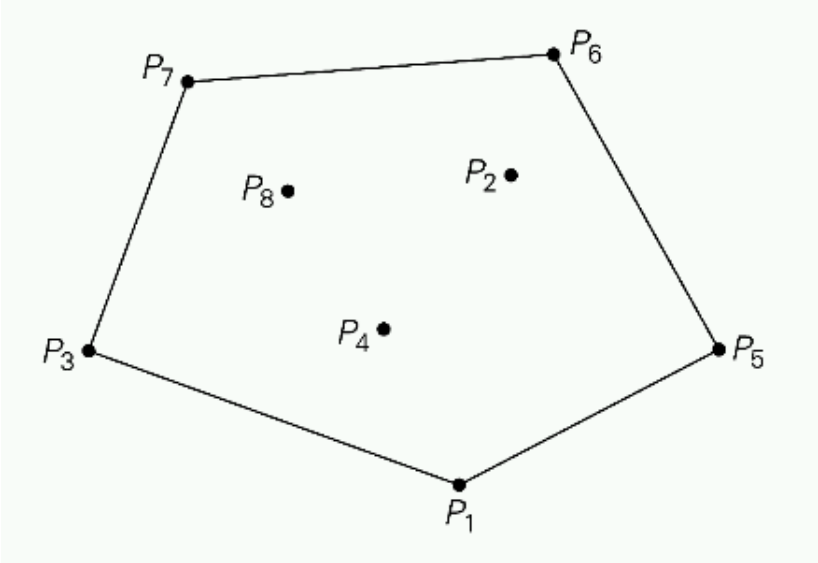
Convexe en niet-convexe vormen

De convex hull (convex omhulsel) van een verzameling S van punten in $\mathbf{R}^2$ is de kleinste convexe verzameling die S bevat.



Convexe omhulsels

Stelling: de convex hull van een verzameling S van $n > 2$ punten in $\mathbf{R}^2$ (niet alle op één lijn) is een convexe veelhoek (polygoon) met als hoekpunten enkele punten uit S.



De convex hull van de verzameling $\{P_1, P_2, \ldots, P_8\}$ is de convexe veelhoek met hoekpunten P_1, P_5, P_6, P_7 en P_3

We baseren ons brute force algoritme op de volgende observatie: een lijnstuk P_iP_j maakt deel uit van de rand van de convex hull van $\{P_1, P_2, \ldots, P_8\}$ d.e.s.d.a. alle andere punten van de verzameling aan een en dezelfde kant van de lijn door P_i en P_j liggen.

Brute force: Ga *voor elk tweetal* punten $P_i = (x_i, y_i)$ en $P_j = (x_j, y_j)$ na of *alle andere punten* aan dezelfde kant van de lijn $(y_j - y_i)x + (x_i - x_j)y = x_iy_j - y_ix_j$ liggen. Zo ja, dan is P_iP_j dus deel van de convex hull.

Complexiteit: $O(n^3)$

Opmerking: het kan veel beter, namelijk $O(n \lg n)$

Opmerking 2: WK 1992

Brute force:

- **Voordelen**:
  - algemeen toepasbaar
  - eenvoudig
  - levert voor een aantal belangrijke problemen (zoeken, patroon-herkenning) een zeer behoorlijk algoritme op
- **Nadelen**:
  - levert meestal geen efficiënt algoritme op
  - soms onacceptabel langzaam

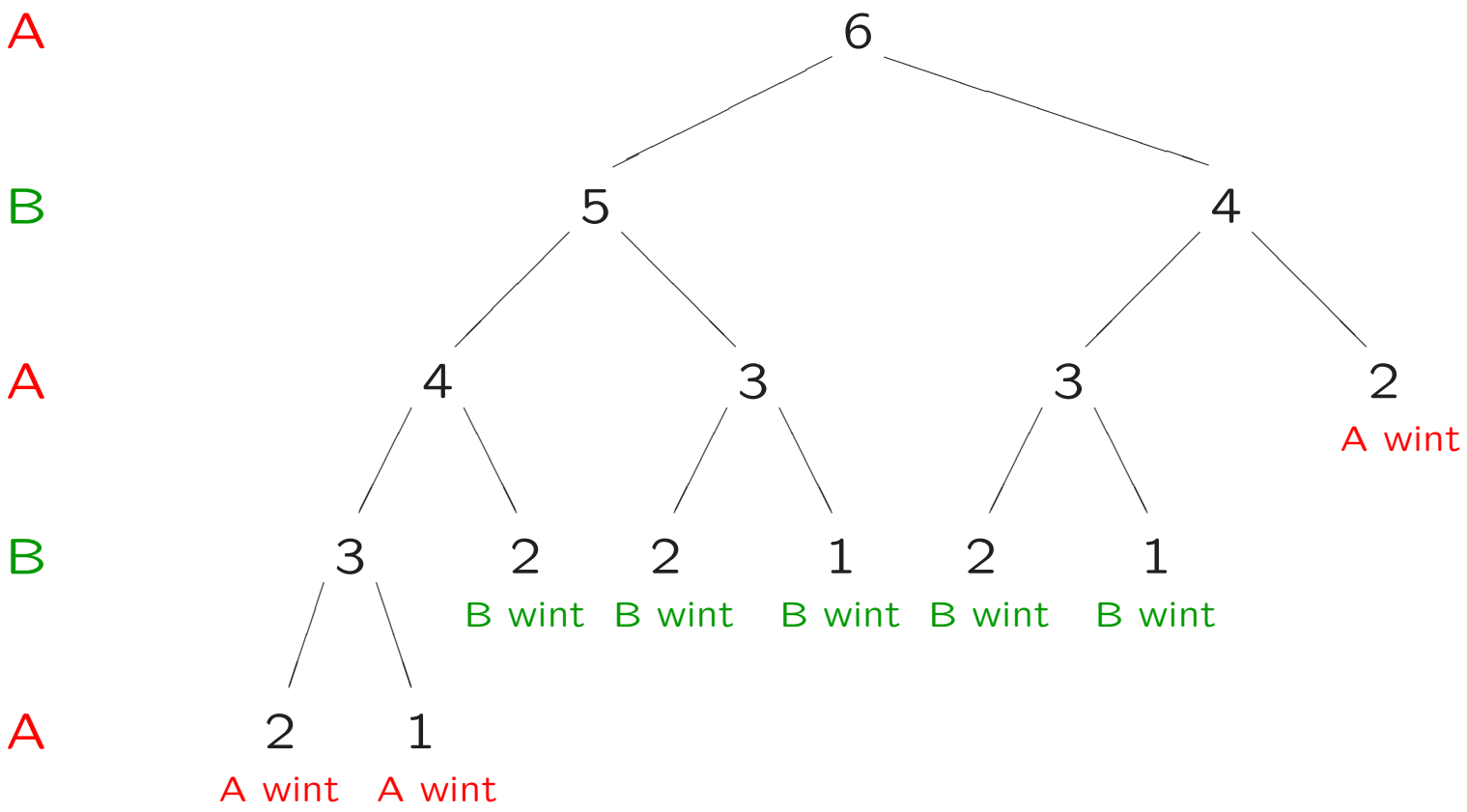
Exhaustive search: brute force benadering voor problemen die te maken hebben met het vinden van een element met een speciale eigenschap binnen een verzameling van bijv. permutaties of deelverzamelingen of toestanden of ...

Methode:

. construeer op een systematische manier alle kandidaatoplossingen

. evalueer elk van deze mogelijke oplossingen

. retourneer een/de kandidaatoplossing met de gevraagde eigenschap (als die bestaat)



Exhaustive search: doorloop (*als het ware*) de hele spelboom om te bepalen of een stand winnend is. Alle toestanden/alle spelverlopen worden zo bekeken. Je kunt stoppen zodra je een winnende zet gevonden hebt.

Exhaustive search: brute force benadering voor problemen die te maken hebben met het vinden van een element met een speciale eigenschap binnen een verzameling van bijv. permutaties of deelverzamelingen of toestanden of ...

Methode:

. construeer op een systematische manier alle kandidaatoplossingen,
bijvoorbeeld alle permutaties van de getallen 1 t/m n

. evalueer elk van deze mogelijke oplossingen

. retourneer een/de kandidaatoplossing met de gevraagde eigenschap (als die bestaat) (*)

(*) soms, zoals bij optimalisatieproblemen, *moet* je daartoe alle kandidaatoplossingen gezien hebben

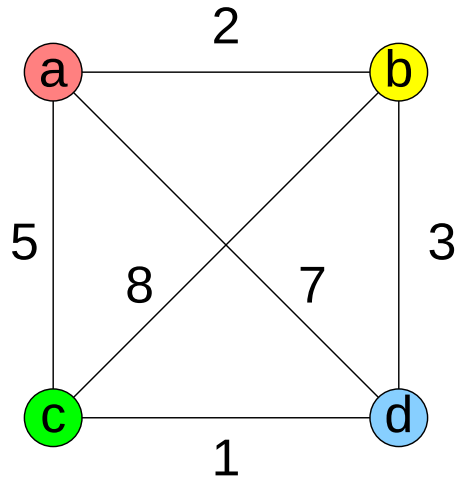
Traveling Salesman Problem (handelsreizigersprobleem)

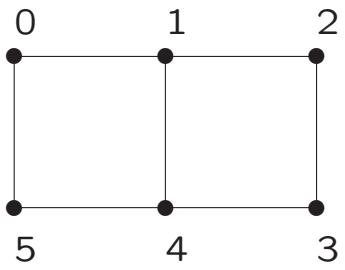
Gegeven n steden waarvan alle onderlinge afstanden bekend zijn.

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 (volledige) graaf.

Voorbeeld:





1.

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

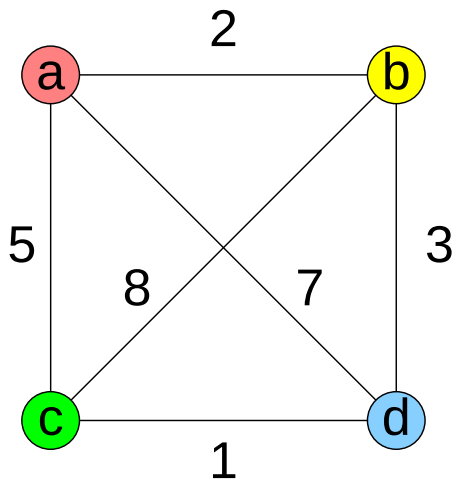
$E = \{(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 0), (4, 1)\}$

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.



Route	**Lengte**
$a \to b \to c \to d \to a$	$2+8+1+7=18$
$a \to b \to d \to c \to a$	$2+3+1+5=11$
$a \to c \to b \to d \to a$	$5+8+3+7=23$
$a \to c \to d \to b \to a$	$5+1+3+2=11$
$a \to d \to b \to c \to a$	$7+3+8+5=23$
$a \to d \to c \to b \to a$	$7+1+8+2=18$

Complexiteit: $\Omega((n-1)!)$,
immers alle $(n-1)!$ mogelijke Hamiltonkringen worden bekeken.

Code voor TSP met ES, 1

```
void bepaalroute(int n, int pos, int route[], int &best) {
  int lengte;

  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
}

int main ( ) {
  int best = MAXINT;

  route[0] = 1;
  bepaalroute (n, 1, route, best);
  cout << "Kortste afstand: " << best << endl;
}
```

```
void bepaalroute(int n, int pos, int route[], int &best) {
  int lengte;

  if (pos == n) {  // route afsluiten ('eindstand')
    route[pos] = route[0];
    lengte = berekenlengte (n, route)
    if (lengte < best)
      best = lengte;
  }
  else {  // pos < n
    ...
    ...
    ...
    ...
    ...
  }
}

int main ( ) {
  int best = MAXINT;

  route[0] = 1;
  bepaalroute (n, 1, route, best);
  cout << "Kortste afstand: " << best << endl;
}
```

```
void bepaalroute(int n, int pos, int route[], int &best) {
  int lengte;

  if (pos == n) {  // route afsluiten ('eindstand')
    route[pos] = route[0];
    lengte = berekenlengte (n, route)
    if (lengte < best)
      best = lengte;
  }
  else {  // pos < n
    for (int i=1; i<=n; i++)  // 'for alle mogelijke zetten'
      if ('i nog niet in route') {
        route [pos] = i;
        bepaalroute (n, pos+1, route, best);
      }
  }
}

int main ( ) {
  int best = MAXINT;

  route[0] = 1;
  bepaalroute (n, 1, route, best);
  cout << "Kortste afstand: " << best << endl;
}
```

```
void bepaalroute(int n, int pos, int route[], bool gehad[], int &best) {
  int lengte;
  if (pos == n) {  // route afsluiten ('eindstand')
    route[pos] = route[0];
    lengte = berekenlengte (n, route)
    if (lengte < best)
      best = lengte;
  }
  else {  // pos < n
    for (int i=1; i<=n; i++)  // 'for alle mogelijke zetten'
      if (! gehad[i]) {
        route [pos] = i;      gehad[i] = true;
        bepaalroute (n, pos+1, route, gehad, best);
        gehad[i] = false;  // 'undoezet'
      }
  }
}

int main ( ) {
  bool gehad[n+1];  // TODO: false initialiseren
  int best = MAXINT;
  route[0] = 1;      gehad[1] = true;
  bepaalroute (n, 1, route, gehad, best);
  cout << "Kortste afstand: " << best << endl;
}
```

- **Lezen/leren bij dit college:**
  slides
  Paragraaf 3 inl., 3.1–3

- Geen werkcollege (opgaven op papier)

- Geen practicumbijeenkomst

- **Volgend college:**
  dinsdag 7 maart 2023, 11.00–12.45