

Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling

Ontwerpen van Algoritmen

Rob H. Bisseling

Mathematisch Instituut, Universiteit Utrecht

Cursus Programmeren voor Wiskunde (WISB152)

17 mei 2022



Universiteit Utrecht

Torens van Hanoi

Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling

Knapsackprobleem

Zelfmijdende wandeling



Universiteit Utrecht

De torens van Hanoi



Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling

- ▶ De torens van Hanoi bevatten een aantal schijven die van links naar rechts verplaatst moeten worden.
- ▶ Je mag een grotere schijf niet op een kleinere plaatsen.
- ▶ Oorsprong: legende over monniken in een klooster in Hanoi (Vietnam) die 64 schijven verplaatsen.
- ▶ Als ze klaar zijn is het **einde der wereld** gekomen.



Universiteit Utrecht

Snapshots oplossing 8 schijven: beginsituatie



Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling

Tussensituatie 1



Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling

7 schijven zijn verplaatst van links naar midden



Universiteit Utrecht

Tussensituatie 2



Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling

Grootste schijf is verplaatst naar eindpositie



Universiteit Utrecht

Eindsituatie



Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling

7 schijven verplaatst naar eindpositie



Universiteit Utrecht

Recursief algoritme: torens van Hanoi

function MOVE(n, i, j)

▷ Deze functie verplaatst n schijven van paal i naar j

$t \neq i, j$ is de andere paal

if $n = 1$ **then**

 print "verplaats schijf van paal i naar paal j "

else

 MOVE($n - 1, i, t$)

 MOVE($1, i, j$)

 MOVE($n - 1, t, j$)

Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Is het einde der wereld nabij?

- ▶ Rekentijd $T_n = 2T_{n-1} + 1$ voor $n > 1$. $T_1 = 1$.
- ▶ Master theorem niet toepasbaar, maar herhaalde substitutie wel:

$$\begin{aligned} T_n &= 2T_{n-1} + 1 \\ &= 2(2T_{n-2} + 1) + 1 \\ &= 4T_{n-2} + 3 \\ &= 4(2T_{n-3} + 1) + 3 \\ &= 8T_{n-3} + 7 \\ &= \dots \\ &= 2^{n-1}T_1 + 2^{n-1} - 1 \\ &= 2^{n-1} + 2^{n-1} - 1 \\ &= 2^n - 1. \end{aligned}$$

Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Is het einde der wereld nabij?



Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling

- ▶ Voor $n = 64$, moeten de monniken
$$2^{64} - 1 = 18\,446\,744\,073\,709\,551\,615$$
(18 triljoen) verplaatsingen uitvoeren.
- ▶ 1 verplaatsing per seconde: 585 miljard jaar
- ▶ We hebben nog de tijd!



Universiteit Utrecht

Recursieve algoritmen

- ▶ De oplossing van de Torens van Hanoi is een klassiek recursief algoritme, een algoritme dat zichzelf aanroeft.
- ▶ Dit is handig om een probleem op te splitsen in kleinere deelproblemen en is een voorbeeld van de verdeel en heers techniek (*divide and conquer*).
- ▶ Recursie geeft korte programmateksten, maar je moet wel goed nadenken.

Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Oplossing Doomsday I

- ▶ Bereken een doomsday voor elke maand:

```
def doomsday(date):
```

```
    """ deze functie berekent de weekdag  
        van een datum vanaf 1900 en voor 2100  
    """
```

```
    day_of_week = [ 'zondag' , 'maandag' , 'dinsdag' ,  
                    'woensdag' , 'donderdag' , 'vrijdag' , 'zaterdag' ]
```

```
# Lijst met doomsdays voor elke maand  
doomsday_in_month = [None, 3, 28, 14, 4, 9, 6,  
                      11, 8, 5, 10, 7, 12]
```

- ▶ Even maanden: 4-4, 6-6, 8-8, 10-10, 12-12
- ▶ Oneven maanden: 9-5 werk, supermarkt 7-11 (beide v.v.)
- ▶ Normaal jaar: 3-1, **28-2**, 14-3 (pi-dag)

Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Oplossing Doomsday II

- ▶ Bereken de weekdag van doomsday in het gegeven jaar

```
# lees de invoer  
s = date.split()  
day = int(s[0])  
month = int(s[1])  
year = int(s[2])
```

Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling

```
# bereken het aantal jaren sinds 1900  
y = year - 1900
```

```
# de Doomsday van dit jaar (1900 is woensdag)  
doomsday = (y // 12) + (y % 12) + (y % 12)//4 + 3
```

- ▶ Een cyclus is 12 jaar. Daarin schuift de doomsday $12 \times 365 + 3 \equiv 12 + 3 \equiv 1$ dag op (modulo 7).
- ▶ Binnen een cyclus schuif je elk jaar een dag op, plus het aantal schrikkeljaren.
- ▶ 1904: doomsday = $0 + 4 + 1 + 3 \equiv 1$ (maandag).



Universiteit Utrecht

Oplossing Doomsday III

- ▶ Bereken de weekdag van de gegeven datum

```
# is het een schrikkeljaar?
```

```
is_leap_year = (year % 4 == 0 and year % 100 != 0  
                ) or year % 400 == 0
```

Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling

```
# op welke datum is het doomsday deze maand?  
doomsday_date = doomsday_in_month[month]
```

```
# in een schrikkeljaar is dat een dag later
```

```
# in januari en februari
```

```
if is_leap_year and month <= 2:  
    doomsday_date += 1
```

```
# reken nu de weekdag uit als een getal modulo 7  
weekday = (doomsday + day - doomsday_date) % 7
```

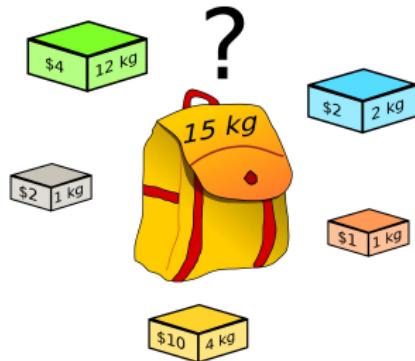
```
# vertaal dit naar de naam van de weekdag
```

```
return day_of_week[weekday]
```



Universiteit Utrecht

Knapsack probleem



Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling

- ▶ Gegeven zijn de **waarden** (*values*) $\{v_i\}_{i=0}^{n-1}$ en **gewichten** (*weights*) $\{w_i\}_{i=0}^{n-1}$ van een verzameling van n items.
- ▶ Selecteer een aantal items met een zo **groot mogelijke gezamenlijke waarde** en met een gezamenlijk gewicht onder een gegeven grens.



Universiteit Utrecht

Wiskundige abstractie

- ▶ De **toestandruimte** (*state space*) \mathcal{S} is de verzameling van mogelijke toestanden van het systeem.
- ▶ De **doelfunctie** c kent een waarde toe aan een gegeven toestand.
- ▶ Probleem: vind een toestand $t \in \mathcal{S}$ waarvoor $c(t)$ zo klein of zo groot mogelijk is.

Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Toestandsruimte voor knapsack

Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling

- De toestandsruimte bestaat uit alle mogelijke deelverzamelingen \mathcal{K} van $\{0, 1, \dots, n - 1\}$ waarvoor

$$\sum_{i \in \mathcal{K}} w_i \leq W.$$

- De doelfunctie is

$$c(\mathcal{K}) = \sum_{i \in \mathcal{K}} v_i.$$



Universiteit Utrecht

Binaire getallen

- ▶ We kunnen de toestanden weergeven als binaire getallen
 $t = (t_0, t_1, \dots, t_{n-1})$ waarbij

$$t_i = \begin{cases} 1 & \text{als } i \in \mathcal{K}, \\ 0 & \text{als } i \notin \mathcal{K}. \end{cases}$$

- ▶ Het aantal mogelijke toestanden is dus 2^n .
- ▶ We kunnen t ook zien als een geheel getal

$$t = \sum_{i=0}^{n-1} t_i 2^{n-1-i}.$$

- ▶ Hierbij is de eerste bit in de lijst, t_0 , het meest significant. Dit wordt wel een big-endian representatie genoemd.

Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Enumeratie

Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling

- ▶ De doelfunctie behorend bij toestand t is dan

$$c(t) = \sum_{i=0}^{n-1} t_i v_i.$$

- ▶ Het probleem kan opgelost worden door **enumeratie**: genereer alle mogelijke toestanden en kies de beste.



Universiteit Utrecht

Enumeratie in Python

- ▶ Pythonprogramma dat alle mogelijke zakken (lijsten van lengte n) genereert:

```
def enumerate(n, zak):  
  
    if n > 0:  
        enumerate(n-1,zak + [0])  
        enumerate(n-1,zak + [1])  
    else:  
        print(zak)
```

```
zak=[]  
enumerate(3,zak)
```

```
[0, 0, 0]  
[0, 0, 1]  
[0, 1, 0]  
[0, 1, 1]  
[1, 0, 0]  
[1, 0, 1]  
[1, 1, 0]  
[1, 1, 1]
```

Torens van
Hanoi

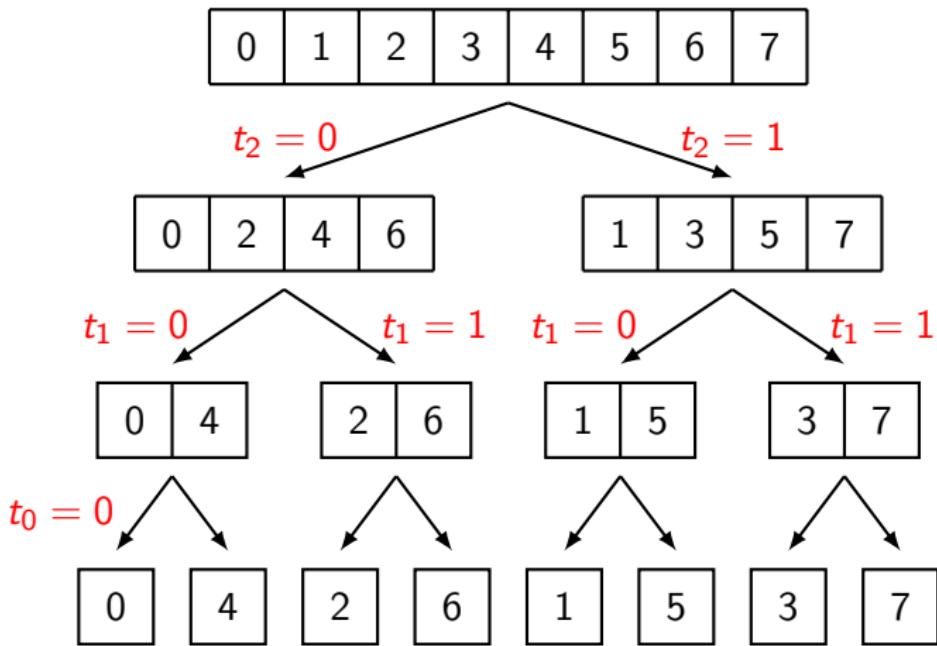
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Zoekboom: splits de toestandsruimte



- De **wortel** van de boom (*root, boven!*) representeert de hele toestandsruimte en de **bladeren** (*leaves, onder!*) de afzonderlijke toestanden.



Sneller zoeken

- ▶ Soms hoef je niet de hele boom te doorzoeken.
Bijvoorbeeld als je weet dat je al te veel gewicht in de knapzak hebt. Dit heet **snoeien** (*pruning*).
- ▶ De volgorde van beslissen maakt ook uit: begin met t_2 als item 2 het zwaarst is, dan kun je waarschijnlijk sneller snoeien.
- ▶ Een optimalisatie die vaak, maar niet altijd goed werkt heet een **heuristiek**.
- ▶ Je kunt ook tevreden zijn met een goede oplossing die niet de beste is: bijvoorbeeld neem telkens het item met de hoogste waarde die nog in de knapzak past. Dit heet een **greedy** algoritme.

Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Nieuwsgierig wandelen



Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling

Bron: my-new-york.com, nyc-architecture.com



Universiteit Utrecht

Definitie zelfmijdende wandeling

- ▶ Een **zelfmijdende wandeling** (*self-avoiding walk*, SAW) is een wandeling op een rooster waarbij wij nooit op de dezelfde plek terugkeren.
- ▶ We beginnen in de oorsprong.
- ▶ De lengte van een wandeling, d.w.z. het aantal stappen, noemen we N .

Torens van
Hanoi

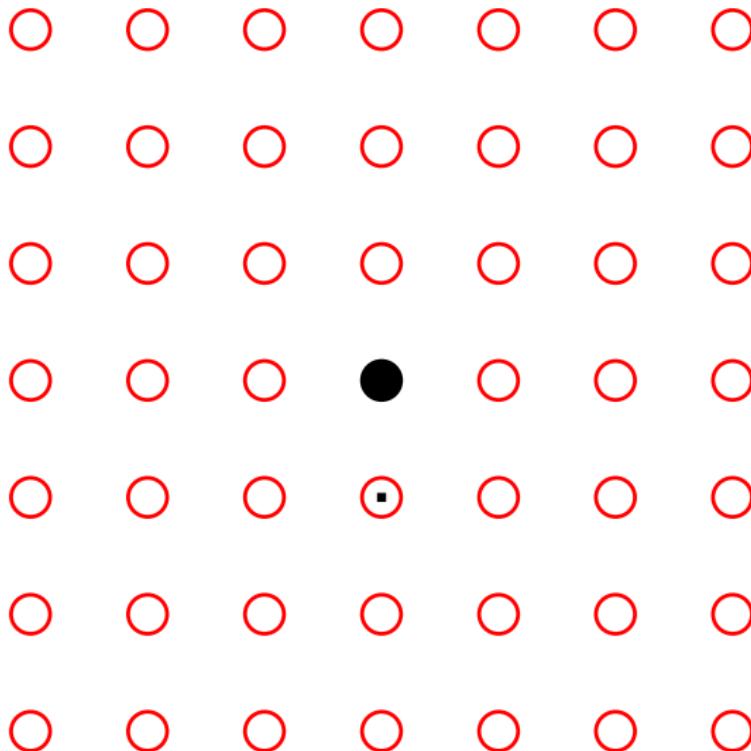
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Een zelfmijdende wandeling van lengte 0 in 2D



Torens van
Hanoi

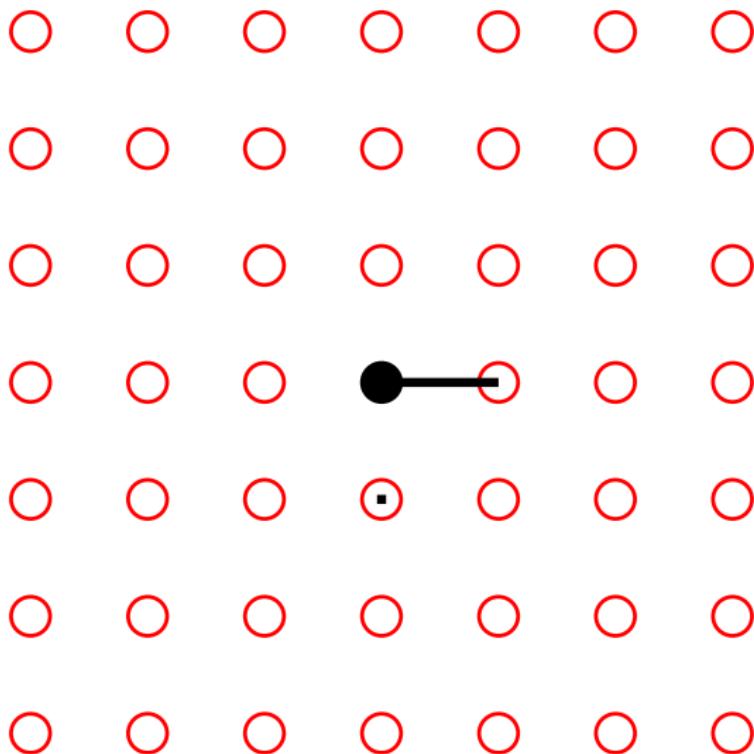
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Een zelfmijdende wandeling van lengte 1 in 2D



Torens van
Hanoi

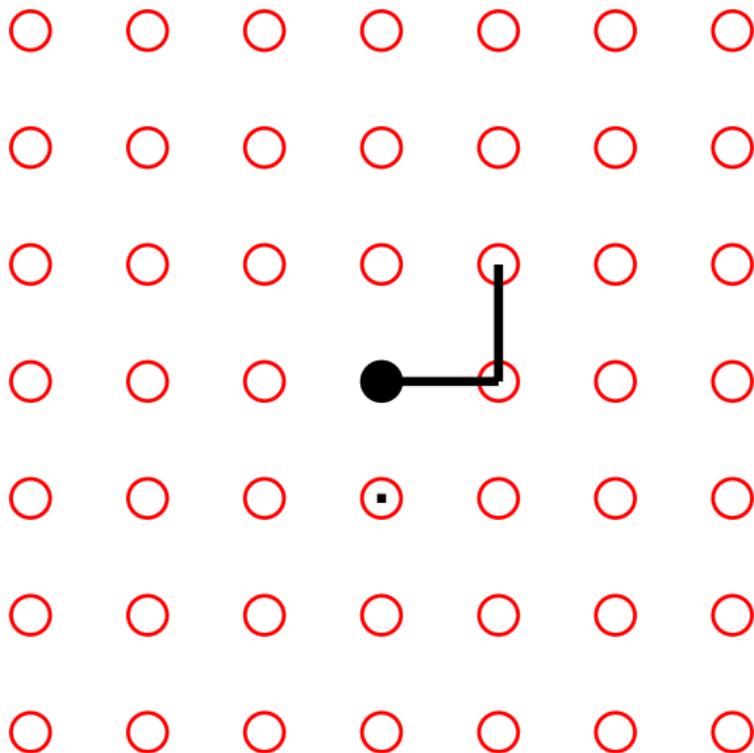
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Een zelfmijdende wandeling van lengte 2 in 2D



Torens van
Hanoi

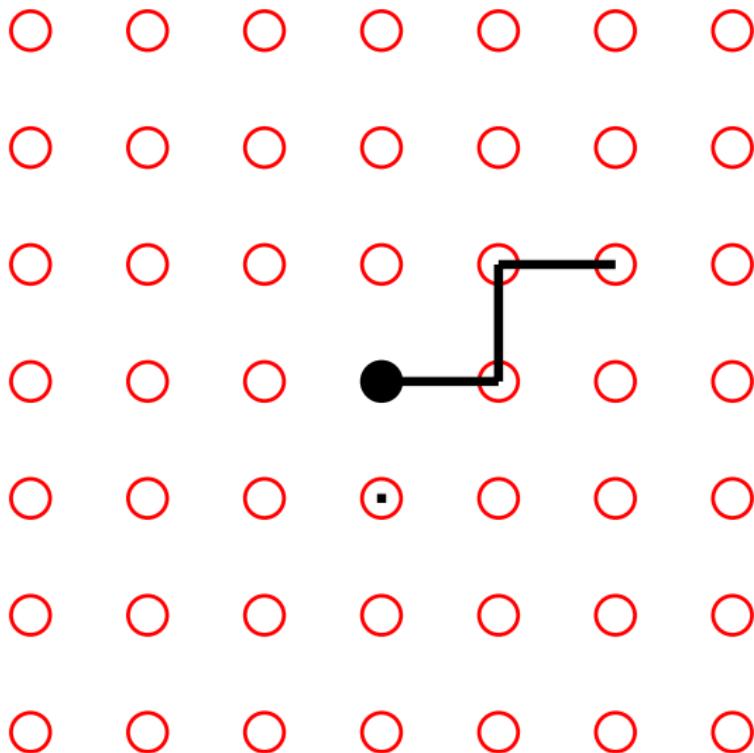
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Een zelfmijdende wandeling van lengte 3 in 2D



Torens van
Hanoi

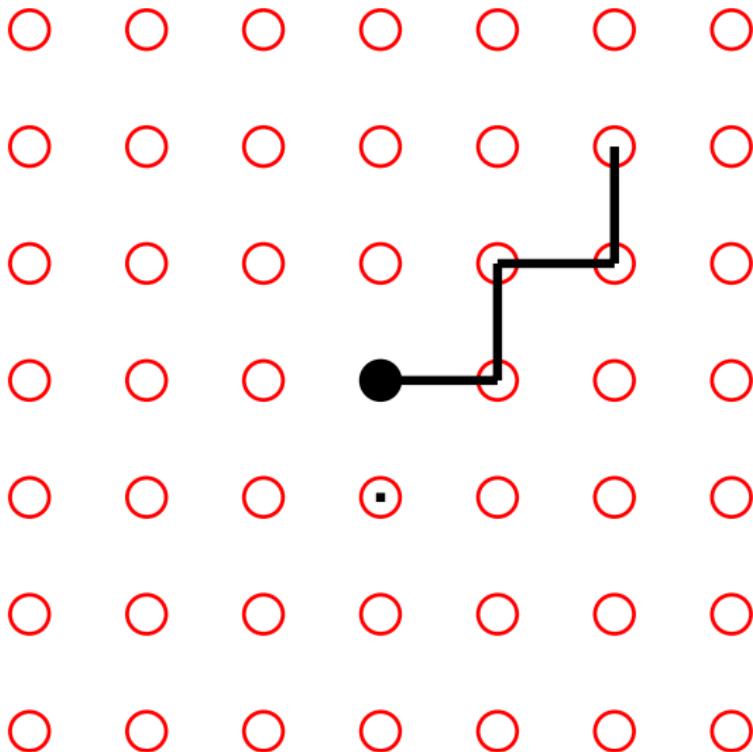
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Een zelfmijdende wandeling van lengte 4 in 2D



Torens van
Hanoi

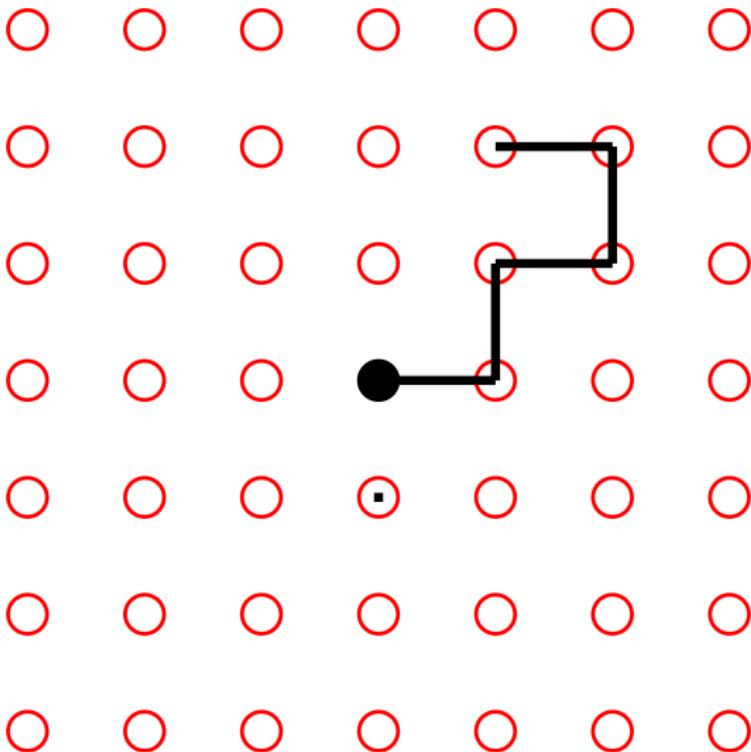
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Een zelfmijdende wandeling van lengte 5 in 2D



Torens van
Hanoi

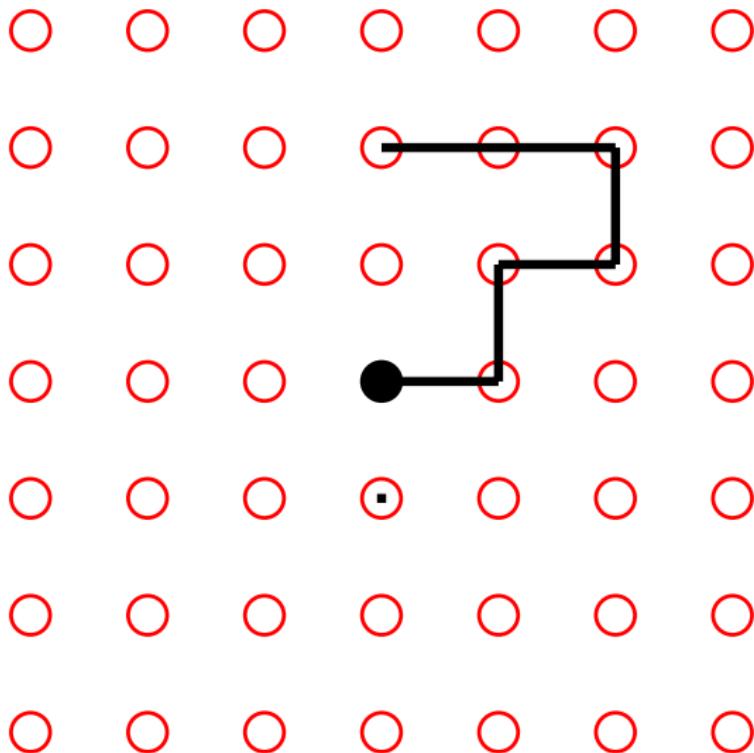
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Een zelfmijdende wandeling van lengte 6 in 2D



Torens van
Hanoi

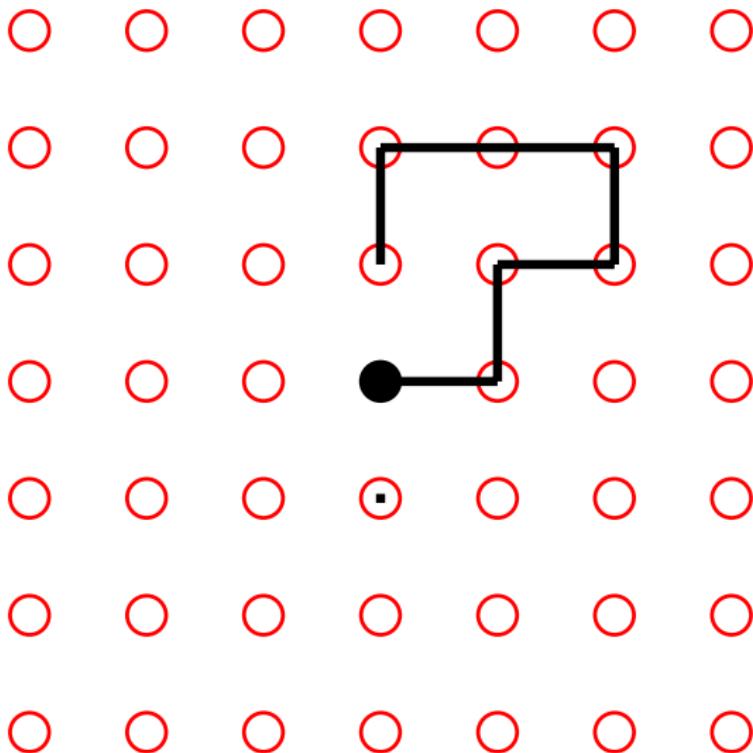
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Een zelfmijdende wandeling van lengte 7 in 2D



Torens van
Hanoi

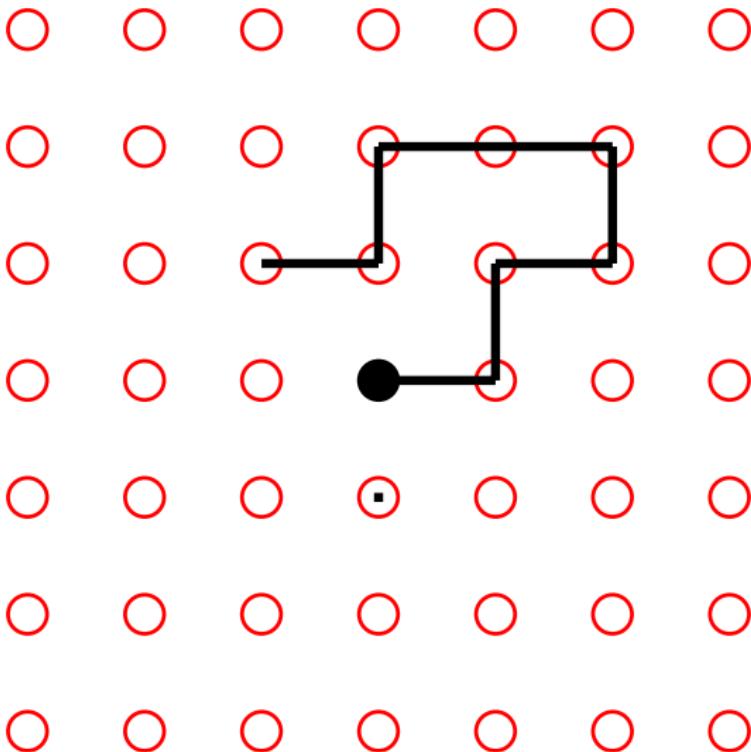
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Een zelfmijdende wandeling van lengte 8 in 2D



Torens van
Hanoi

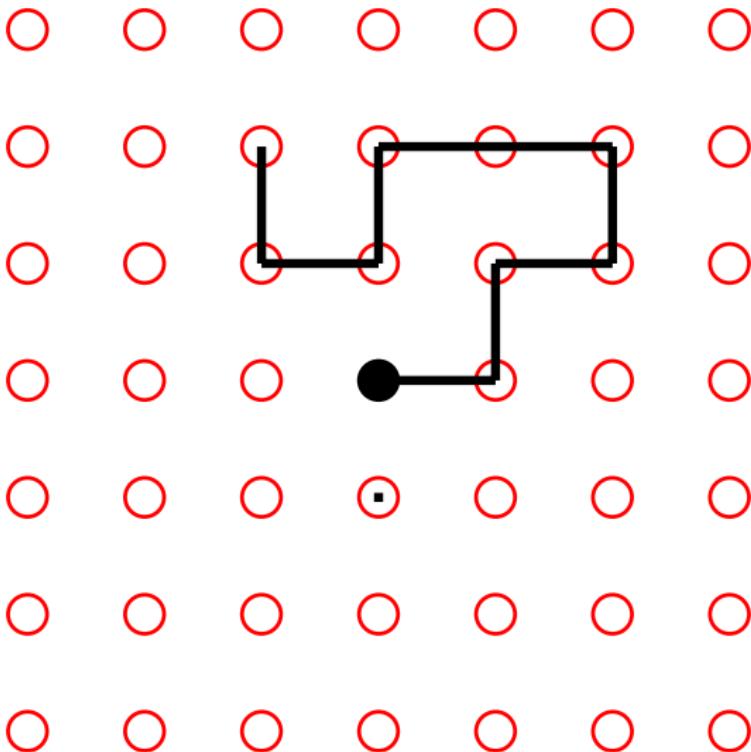
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Een zelfmijdende wandeling van lengte 9 in 2D



Torens van
Hanoi

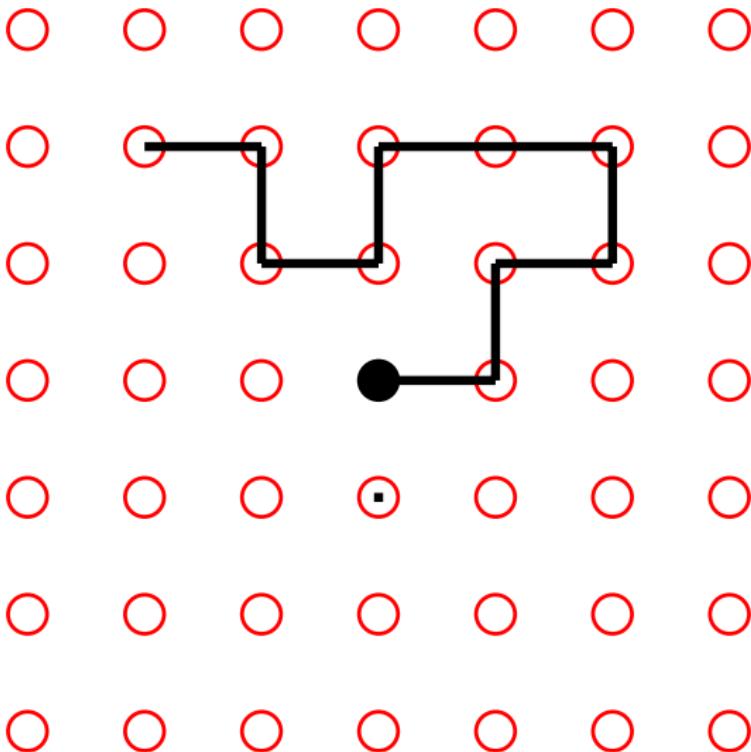
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Een zelfmijdende wandeling van lengte 10 in 2D



Torens van
Hanoi

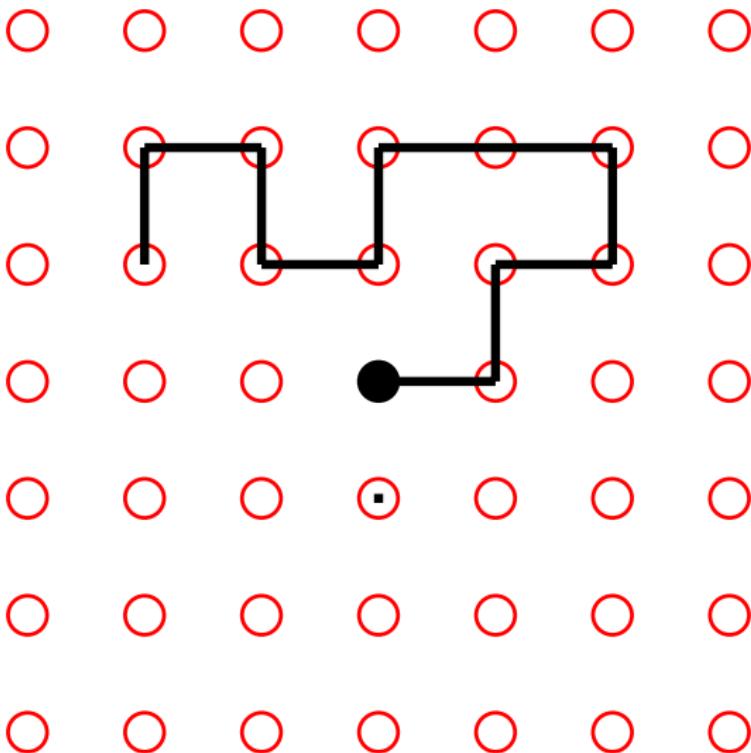
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Een zelfmijdende wandeling van lengte 11 in 2D



Torens van
Hanoi

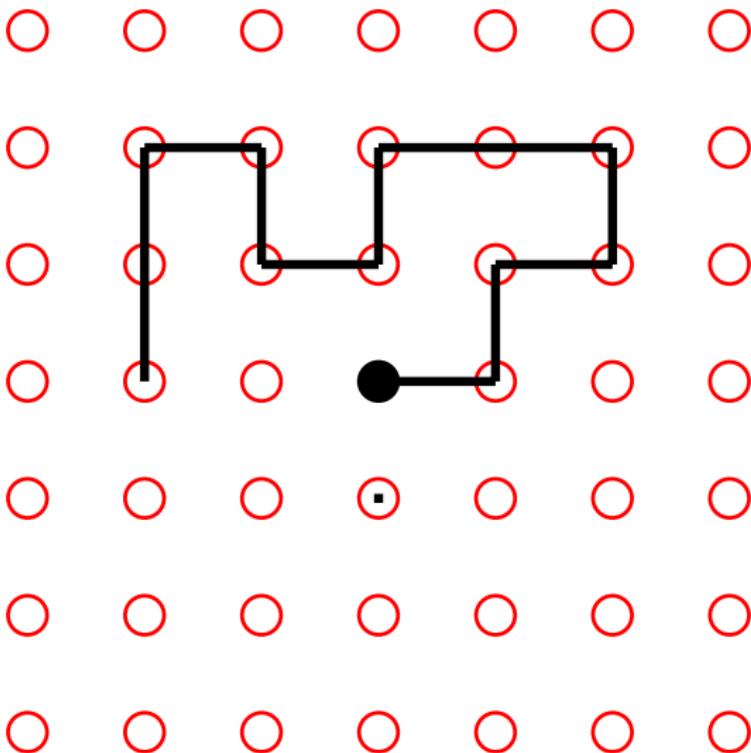
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Een zelfmijdende wandeling van lengte 12 in 2D



Torens van
Hanoi

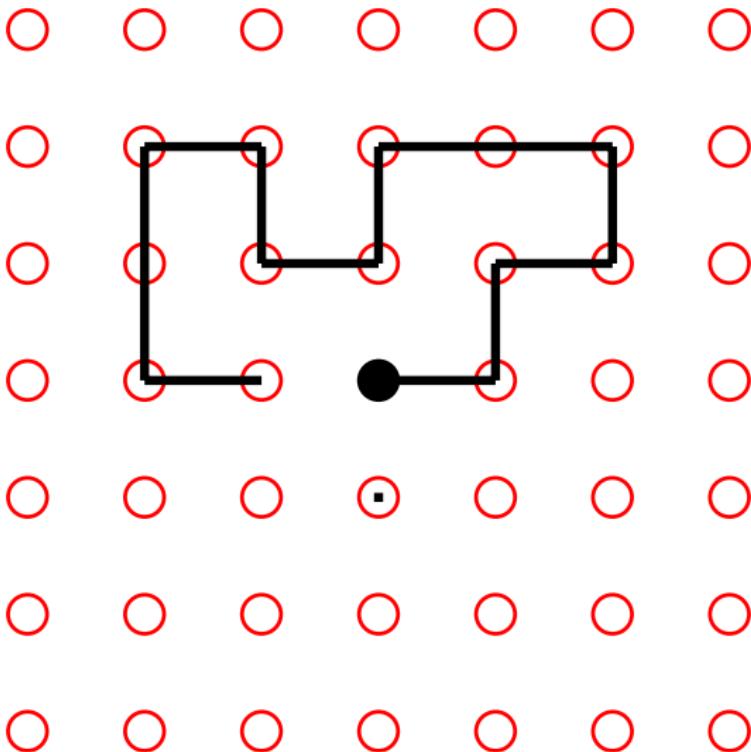
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Een zelfmijdende wandeling van lengte 13 in 2D



Torens van
Hanoi

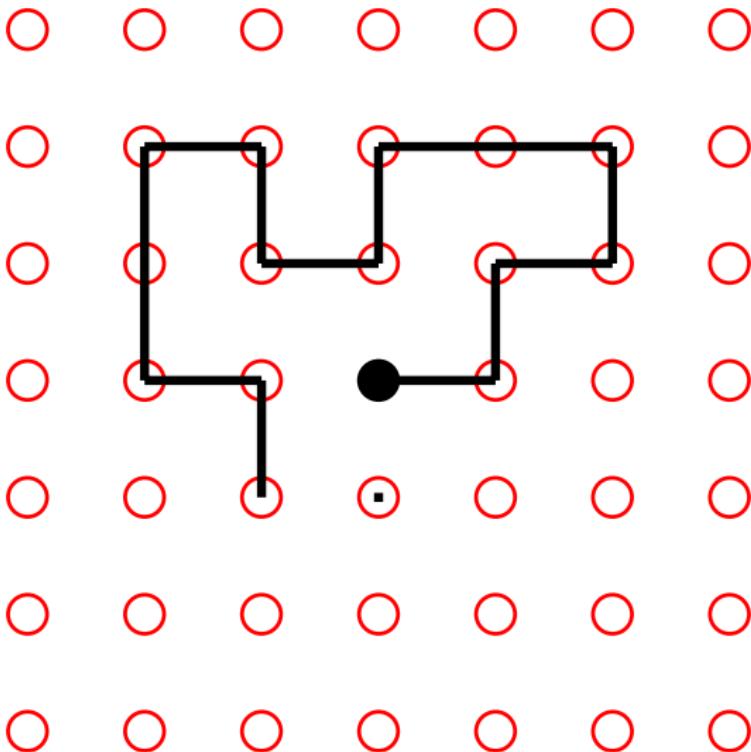
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Een zelfmijdende wandeling van lengte 14 in 2D



Torens van
Hanoi

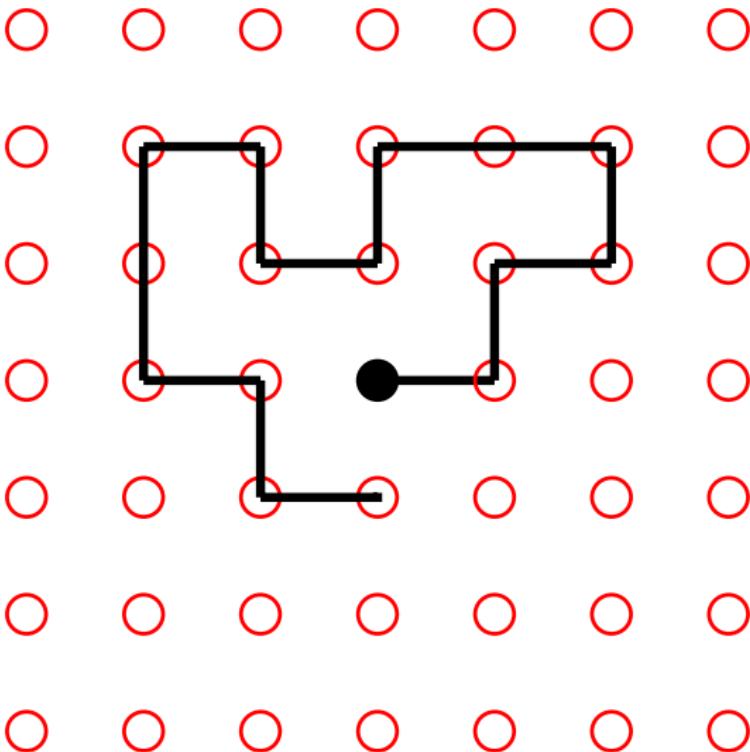
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Een zelfmijdende wandeling van lengte 15 in 2D



Torens van
Hanoi

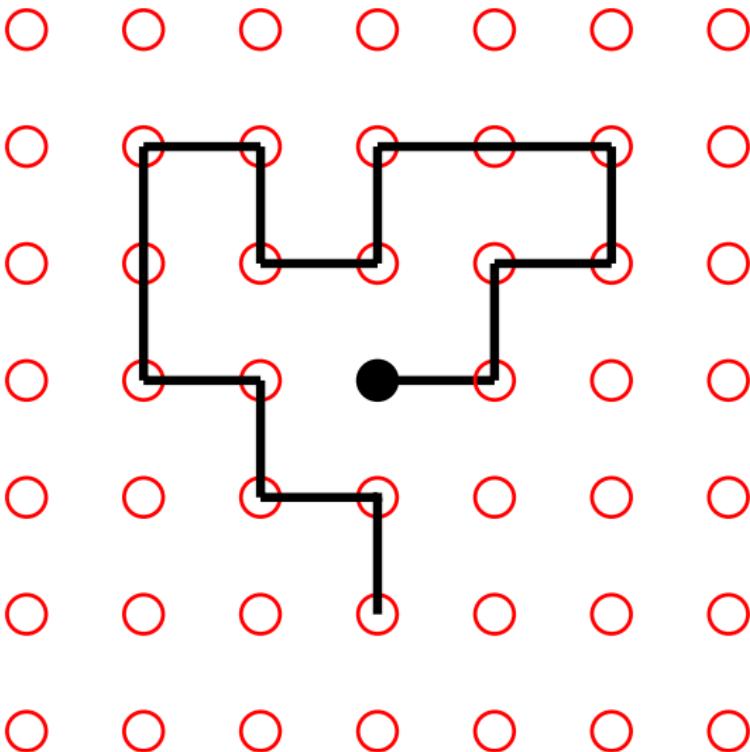
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Een zelfmijdende wandeling van lengte 16 in 2D



Torens van
Hanoi

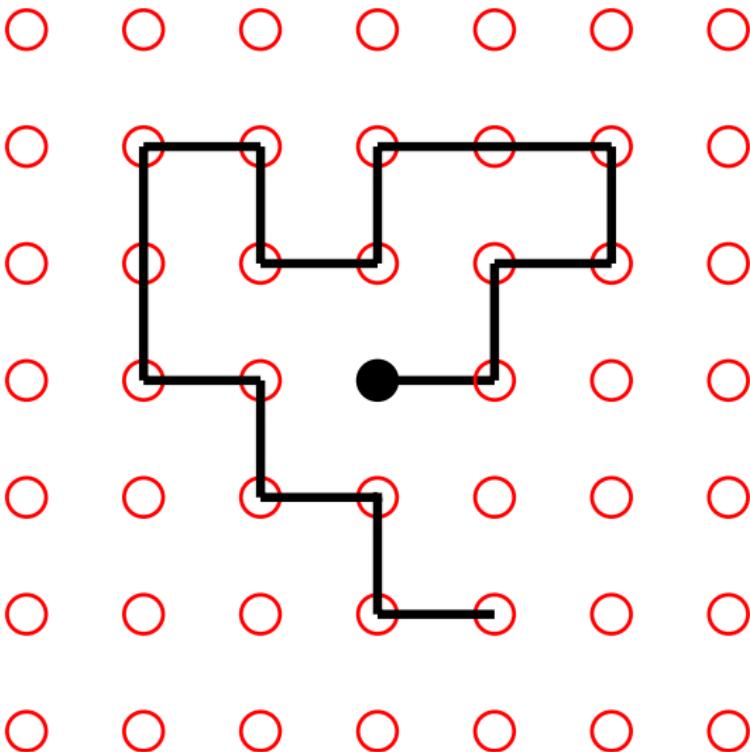
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Een zelfmijdende wandeling van lengte 17 in 2D



Torens van
Hanoi

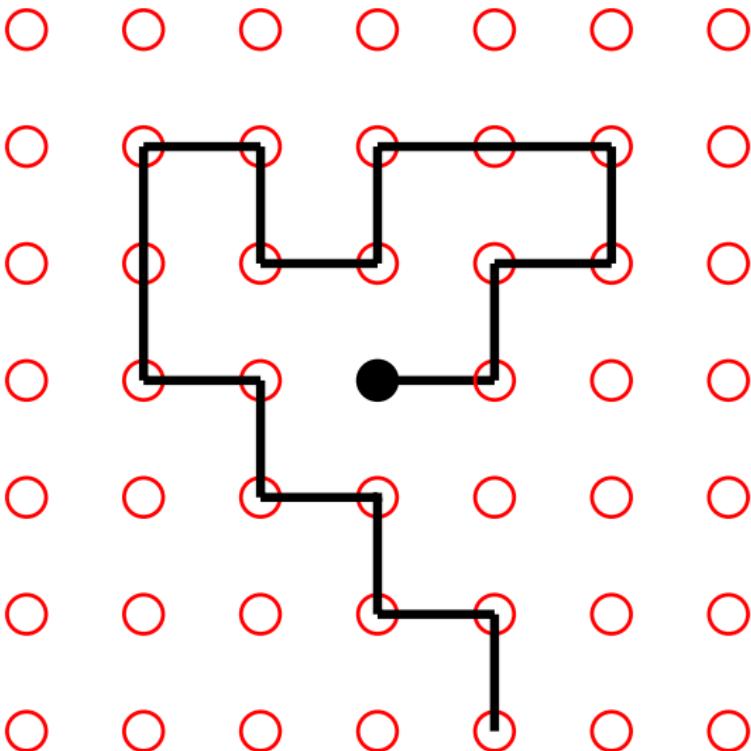
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Een zelfmijdende wandeling van lengte 18 in 2D



Torens van
Hanoi

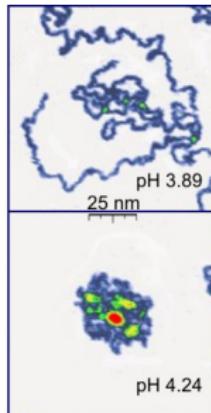
Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Waarom zijn zelfmijdende wandelingen nuttig?



Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling

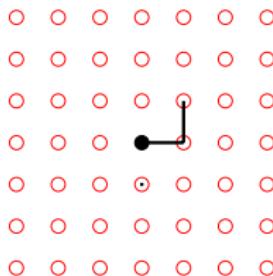
Poly(2-vinylpyridine) geobserveerd m.b.v. Atomic Force Microscope. Bron: Roiter en Minko (2007).

- ▶ Wandeling modelleert een **polymeer**, een lang molecuul, gebaseerd op een koolstofketen C–C–C–C··· C.
- ▶ DNA is een polymeer. Wij bestaan uit DNA.
- ▶ Zelfmijdend omdat je niet twee koolstofatomen op dezelfde plaats kunt zetten.



Universiteit Utrecht

Hoeveel zelfmijdende wandelingen zijn er?



Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling

- ▶ In 2D lengte $N = 2$: $Z_2 = 12$ wandelingen.
- ▶ Vraag: in 2D lengte $N = 3$?
- ▶ Vraag: in 2D lengte $N = 4$?
- ▶ Opgave met pen en papier. In 3D, wat is Z_1, Z_2, Z_3, Z_4 ?
- ▶ Opgave met pen en papier. Kun je een bovengrens geven voor Z_N als functie van N ? Dus $Z_N \leq \dots$
- ▶ En een ondergrens?



Universiteit Utrecht

Recursief SAW programma in Python met Numpy

- ▶ Registreer de wandeling t/m stap i als $(x[0], y[0]), \dots, (x[i], y[i])$, waarbij $0 \leq x[i], y[i] < 2N + 1$.

- ▶ **import** numpy

```
N = 5 # lengte van de wandeling
```

```
# Initialiseer visited: nog geen roosterpunten bezocht  
visited = numpy.zeros ((2*N+1,2*N+1), dtype = bool)
```

```
# Initialiseer wandeling en startpunt (N,N)
```

```
x = numpy.zeros ((N+1), dtype = int)
```

```
y = numpy.zeros ((N+1), dtype = int)
```

```
x[0] = N
```

```
y[0] = N
```

```
# Enumereer alle wandelingen
```

```
count = go(x, y, visited, 0, N)
```

```
print("het aantal wandelingen van lengte", N, "is", count)
```

Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Uitbreiden van stap $i - 1$ naar i

```
def go(x, y, visited, i, N):
    if visited[x[i],y[i]]:
        return 0 # ga terug: backtrack
    elif i==N:
        return 1 # een nieuwe SAW gevonden
    visited[x[i],y[i]] = True

    x[i+1] = x[i]+1
    y[i+1] = y[i]
    count = go(x, y, visited, i+1, N)
    x[i+1] = x[i]-1
    y[i+1] = y[i]
    count += go(x, y, visited, i+1, N)
    x[i+1] = x[i]
    y[i+1] = y[i]+1
    count += go(x, y, visited, i+1, N)
    x[i+1] = x[i]
    y[i+1] = y[i]-1
    count += go(x, y, visited, i+1, N)

    visited[x[i],y[i]] = False
    return count
```

Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Opknippen van een zelfmijdende wandeling

- ▶ Er geldt

$$Z_{M+N} \leq Z_M \cdot Z_N.$$

- ▶ Immers, een zelfmijdende wandeling van lengte $M + N$ kun je **opknippen** in een zelfmijdende wandeling van lengte M en een van lengte N .
- ▶ Er zijn in totaal hoogstens $Z_M \cdot Z_N$ verschillende mogelijkheden.
- ▶ Neem $M = N$, dan is $Z_{2N} \leq (Z_N)^2$.
- ▶ Dus $Z_N \geq (Z_{2N})^{1/2}$ voor alle N dus

$$Z_1 \geq (Z_2)^{1/2} \geq (Z_4)^{1/4} \geq (Z_8)^{1/8} \geq \dots$$

Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Aantal wandelingen in 2D

- $Z_N \geq (Z_{2N})^{1/2}$ voor alle N dus

$$Z_1 \geq (Z_2)^{1/2} \geq (Z_4)^{1/4} \geq (Z_8)^{1/8} \geq \dots$$

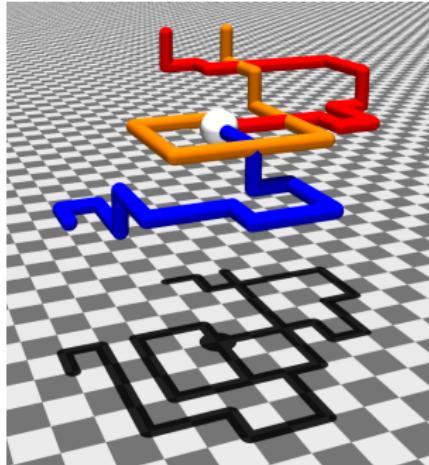
N	Z_N	$(Z_N)^{1/N}$
1	4	4.00
2	12	3.46
4	100	3.16
8	5916	2.96
16	17245332	2.83
32	119034997913020	2.75
64	4549252727304405545665901684	2.70

- Dalend rijtje getallen, van beneden begrensd, dus er is een limiet:

$$\lim_{N \rightarrow \infty} (Z_N)^{1/N} = \mu \approx 2.64, \quad \text{met } Z_N \sim \mu^N$$



Drie zelfmijdende wandelingen van lengte 18 in 3D



Torens van
Hanoi

Knapsack

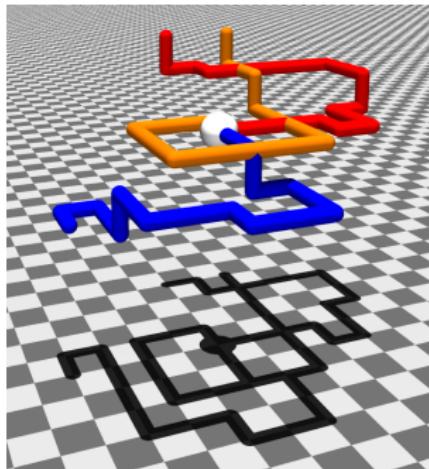
Zelfmijdende
wandeling

- ▶ Zelfmijdende wandelingen van lengte 18:
rood, oranje, blauw.
- ▶ Vraag: hoeveel combinaties van twee verschillende zelfmijdende wandelingen zijn er die je tot een wandeling van lengte 36 kunt aaneenklossen?
- ▶ Vraag: hetzelfde maar nu met zelfmijdend resultaat.



Universiteit Utrecht

Andere telmethode: kijk naar snijpunten



Torens van
Hanoi

Knapsack

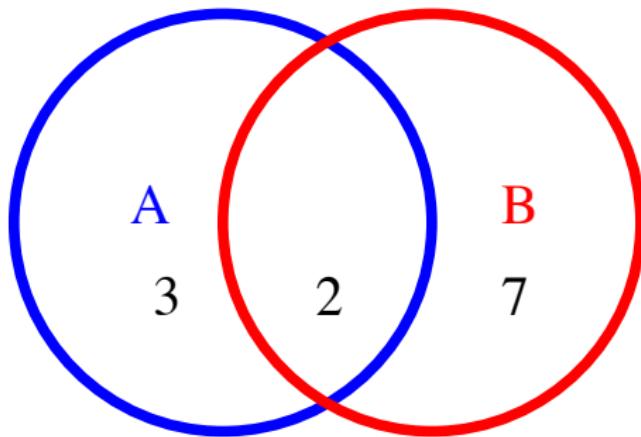
Zelfmijdende
wandeling

- ▶ Snijpunt $a = (2, 0, 0)$: rood/ oranje
- ▶ Snijpunt $b = (2, 3, 1)$: rood/ oranje
- ▶ Snijpunt $c = (0, -2, 0)$: blauw/ oranje
- ▶ Er zijn 3 paren $\{v, w\}$ met $v \neq w$.
- ▶ Voor elk snijpunt, verwijder het bijbehorende paar.
- ▶ Corrigeer: {rood, oranje} was dubbel verwijderd,
dus totaal $3-3+1 = 1$ combinatie blijft over.



Universiteit Utrecht

Principe van inclusie-exclusie: 2 verzamelingen



Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling

- ▶ Aantal elementen van A is $|A| = 5$.
- ▶ $|B| = 9$. Aantal elementen van de vereniging is

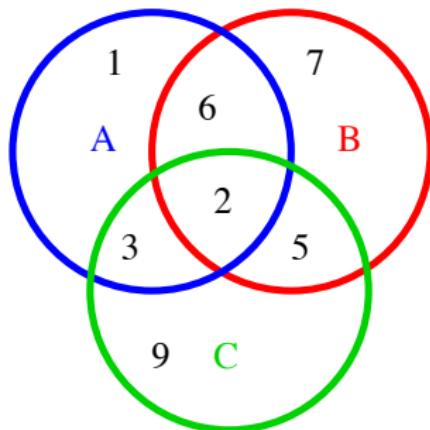
$$|A \cup B| = |A| + |B| - |A \cap B|$$

- ▶ Hier: $|A \cup B| = 5 + 9 - 2 = 12$.
- ▶ Andere telwijze: $3+2+7 = 12$.



Universiteit Utrecht

Principe van inclusie-exclusie: 3 verzamelingen



Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling

- Aantal elementen van de vereniging is

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |A \cap C| + |A \cap B \cap C|$$

- Hier: $|A \cup B \cup C| = 12 + 20 + 19 - 8 - 7 - 5 + 2 = 33$.
- Andere telwijze: $1+6+2+3+7+5+9 = 33$.



Universiteit Utrecht

Tellen van paren wandelingen

- ▶ Pas het inclusie-exclusie principe uit de **combinatoriek** toe met A_i de verzameling van paren zelfmijdende wandelingen $\{v, w\}$ van lengte N vanuit de oorsprong die beide door roosterpunt i gaan.
- ▶ De roosterpunten zijn nu genummerd van 1 tot n ; de oorsprong $i = 0$ doet niet mee.
- ▶ De verzameling

$$\bigcup_{i=1}^n A_i = A_1 \cup A_2 \cup \cdots \cup A_n$$

bevat alle paren die elkaar snijden.

Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Berekening van Z_{2N}

- ▶ Er zijn evenveel zelfmijdende wandelingen van lengte $2N$ als er niet-snijdende paren van lengte N zijn, want je kunt twee wandelingen **aaneenklossen**.
- ▶ Er geldt dus:

$$Z_{2N} = Z_N^2 - \left| \bigcup_i A_i \right|.$$

- ▶ Dit kunnen we **efficiënt uitrekenen** door alleen naar wandelingen van lengte N te kijken.

Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling



Universiteit Utrecht

Aantal zelfmijdende wandelingen in 3D

<i>N</i>	Z_N	Jaar Auteur	
1	6		Torens van Hanoi
2	30		Knapsack
3	150		Zelfmijdende wandeling
4	726		
5	3 534		
6	16 926	1947 Orr	
7	81 390		
8	387 966		
9	1 853 886	1959 Fisher, Sykes	
10	8 809 878		
11	41 934 150		
12	198 842 742		
13	943 974 510		
14	4 468 911 678		
15	21 175 146 054		
16	100 121 875 974		
17	473 730 252 102		
18	2 237 723 684 094		



Aantal zelfmijdende wandelingen in 3D

N	Z_N	Jaar Auteur	
19	10 576 033 219 614		Torens van Hanoi
20	49 917 327 838 734	1987 Guttmann	Knapsack
21	235 710 090 502 158	1989 Guttmann	Zelfmijdende wandeling
22	1 111 781 983 442 406		
23	5 245 988 215 191 414	1992 MacDonald e.a.	
24	24 730 180 885 580 790		
25	116 618 841 700 433 358		
26	549 493 796 867 100 942	2000 MacDonald e.a.	
27	2 589 874 864 863 200 574		
28	12 198 184 788 179 866 902		
29	57 466 913 094 951 837 030		
30	270 569 905 525 454 674 614	2007 Clisby, Liang, Slade	
31	1 274 191 064 726 416 905 966		
32	5 997 359 460 809 616 886 494		
33	28 233 744 272 563 685 150 118		
34	132 853 629 626 823 234 210 582		
35	625 248 129 452 557 974 777 990		
36	2 941 370 856 334 701 726 560 670	2011 Schram, Barkema, Bisseling	

Publicatie in *Journal of Statistical Mechanics*

Torens van
Hanoi

Knapsack

Zelfmijdende
wandeling

Exact enumeration of self-avoiding walks

R. D. Schram,^{1,2} G. T. Barkema,¹ and R. H. Bisseling,²

¹Institute for Theoretical Physics, Utrecht University,
P.O. Box 80195, 3508 TD Utrecht, The Netherlands

²Mathematical Institute, Utrecht University, P.O. Box 80010, 3508 TA Utrecht, The Netherlands
(Dated: May 9, 2011)

A prototypical problem on which techniques for exact enumeration are tested and compared is the enumeration of self-avoiding walks. Here, we show an advance in the methodology of enumeration, making the process thousands or millions of times faster. This allowed us to enumerate self-avoiding walks on the simple cubic lattice up to a length of 36 steps.

I. INTRODUCTION

According to renormalization group theory, the scaling properties of critical systems are insensitive to microscopic details and are governed by a small set of universal exponents. Polymers can be considered as critical systems in the limit where their length N (the number of chained monomers) grows [1]. For instance, the free energy F_N of an isolated polymer in a swollen phase behaves asymptotically as $\exp(-F_N) \equiv Z_N \approx A\mu^N N^\theta$. Here, the connectivity constant μ and the amplitude A are non-universal (model-dependent) quantities. The exponent θ , however, characterizing the leading correction to the scaling behavior, is believed to be universal; it is related to the entropic exponent $\gamma = \theta + 1 \approx 1.157$. The average squared distance between the end points of such polymers scales as $N^{2\nu}$, where $\nu \approx 0.588$ in three dimensions is also a universal critical exponent.

Universal exponents such as θ and ν can be measured most accurately in computer simulations of the most rudimentary models in the universality class of swollen polymers, which arguably is that of self-avoiding walks (SAWs) on a lattice. Estimates of these exponents can be obtained by counting the number Z_N of SAWs of all lengths up to N_{\max} , and calculating the sum P_N of their squared end-to-end extensions, which scales as $P_N \sim Z_N N^{2\nu}$. The exponents can then be obtained from

$$\theta = \frac{N^2 - 4}{4} \left[\log \frac{Z_N^2}{Z_{N+2} Z_{N-2}} \right] \quad (1)$$

and

$$\nu = \frac{N - 1}{4} \left[\log \frac{P_{N+1}}{Z_{N+1}} - \log \frac{P_{N-1}}{Z_{N-1}} \right], \quad (2)$$

respectively, in the limit of increasing N . In Eq. (1), the values of N are taken a distance two apart, so that the formula involves either only even N , or only odd N ; this is more accurate than mixing even and odd values. Similar considerations lead to Eq. (2). The accuracy of the estimates improves significantly with increasing N_{\max} , but unfortunately at the expense of an exponentially growing number of walks. In two dimensions, various algorithmic improvements have allowed for the enumeration of all SAWs up to $N_{\max} = 71$ steps [2], but these methods cannot be used effectively in three dimensions, which is the most relevant dimensionality for practical purposes. Hence, to date, the enumeration of three-dimensional SAWs stops at $N_{\max} = 30$ steps [3].

Counting SAWs has a long history, see e.g. [4]. In a paper by Orr [5] from 1947, Z_N was given for all N up to $N_{\max} = 6$; these values were calculated by hand. In 1959, Fisher and Sykes [6] enumerated all SAWs in 3D up to $N_{\max} = 9$. More recently, in 1987 Guttmann [7] enumerated longer SAWs up to $N_{\max} = 20$, and extended this by one step in 1989 [8]. In 1992, MacDonald et al. [9] reached $N_{\max} = 23$, and in 2000 MacDonald et al. [10] reached $N_{\max} = 26$. In 2007, Cldy et al. [3] reached $N_{\max} = 30$, which is currently the best result.

Here, we present the new length-doubling method which allowed us to reach $N_{\max} = 36$ using 50,000 hours of computing time, a result that would have taken roughly fifty million hours with traditional methods, or alternatively we would have to wait another 20 years by Moore's law (which states that the number of transistors on a computer chip doubles every two years) before we could undertake the computation.

II. LENGTH-DOUBLING METHOD

In the length-doubling method, we determine for each non-empty subset S of lattice sites the number $Z_N(S)$ of SAWs with length N and originating in the origin, that visit the complete subset. Let $|S|$ denote the number of sites in S . The number Z_{2N} of SAWs of length $2N$ can then be obtained by the length-doubling formula



Universiteit Utrecht