

Inleiding tot Programmeren

Toon Calders

toon.calders@uantwerpen.be

Les 5: recursieve functies

Wat zagen we vorige keer?

- dict, set en tuple
- Composite data types (samengestelde data types)
- dict, set en list zijn *mutable*
- tuple, string, int, bool zijn *immutable*

Snelle quiz

- Wat is het resultaat van het volgende code-fragment:

```
L=[{1,2},{(3,4),(3,4)},{7}]  
K=L  
K[0]={3,4}  
print(L)
```

- (A) $[\{1,2\}, \{(3,4), (3,4)\}, \{7\}]$
- (B) $[\{3,4\}, \{(3,4)\}, \{7\}]$
- (C) $[\{1,2\}, \{(3,4)\}, \{7\}]$

Snelle quiz

- Wat is het resultaat van het volgende code-fragment:

```
L=[{1,2},{(3,4),(3,4)},{7}]  
K=L  
K[0]={3,4}  
print(L)
```

- (A) [{1,2}, {(3,4), (3,4)}, {7}]
- (B) [{3,4}, {(3,4)}, {7}]**
- (C) [{1,2}, {(3,4)}, {7}]

Snelle quiz

- Wat is het resultaat van het volgende code-fragment:

```
L=[ {1, 2}, { (3, 4) , (3, 4) }, {7} ]  
K=L[:]  
K[0]={3, 4}  
print(L)
```

- (A) [{1,2}, {(3, 4), (3, 4)}, {7}]
- (B) [{3, 4}, {(3, 4)}, {7}]
- (C) [{1, 2}, {(3, 4)}, {7}]

Snelle quiz

- Wat is het resultaat van het volgende code-fragment:

```
L=[ {1, 2}, { (3, 4) , (3, 4) }, {7} ]  
K=L[:]  
K[0]={3, 4}  
print(L)
```

- (A) [[{1,2}, {(3, 4), (3, 4)}, {7}]]
- (B) [[{3, 4}, {(3, 4)}, {7}]]
- (C) [[{1, 2}, {(3, 4)}, {7}]]**

Snelle quiz

- Wat is het resultaat van het volgende code-fragment:

```
L=[ {1,2}, { (3,4) , (3,4) }, {7} ]  
K=L[:]  
K[0].add(2)  
K[0].add(3)  
print(L)
```

- (A) [{1, 2, 2, 3}, {(3, 4), (3, 4)}, {7}]
- (B) [{1, 2}, {(3, 4)}, {7}]
- (C) [{1, 2, 3}, {(3, 4)}, {7}]

Snelle quiz

- Wat is het resultaat van het volgende code-fragment:

```
L=[ {1,2}, { (3,4) , (3,4) }, {7} ]  
K=L[:]  
K[0].add(2)  
K[0].add(3)  
print(L)
```

- (A) [{1, 2, 2, 3}, {(3, 4), (3, 4)}, {7}]
- (B) [{1, 2}, {(3, 4)}, {7}]
- (C) [{1, 2, 3}, {(3, 4)}, {7}]**

Deze les

- Recursieve functies = functies die zichzelf aanroepen
- Wederzijdse recursie (mutual recursion) = twee of meerdere functies die elkaar aanroepen
- Vergeet niet een basisgeval op te stellen!
- Recursie brengt veel overhead met zich mee: opbouwen en afbreken stack frames
- Als f zichzelf oproept, dan hebben de twee uitvoeringen geen toegang tot elkaars variabelen!

Voorbeeld: recursieve functie

- Bij het oplopen van een trap kan je telkens 1 of 2 treden tegelijk nemen; op hoeveel manieren kan je een trap met n treden oplopen?
- We maken een functie $\text{manieren}(n)$
 - We starten met 1 trede: $\text{manieren}(n-1)$ voor de rest
 - We starten met 2 treden: $\text{manieren}(n-2)$ voor de rest
- Basisgeval vergeten?
 - Oneindige recursie

Stack Frame

```
def manieren(n):  
    if n<=1:  
        return 1  
    return manieren(n-1)+manieren(n-2)  
  
print(manieren(3))
```

- Recursieve aanroepen zijn onafhankelijk van elkaar

```
manieren(3)                                n=3  
    if n<=1:  
        return 1  
    return manieren(n-1)+manieren(n-2)  
        manieren(2)                        n=2  
            if n<=1:  
                return 1  
            return manieren(n-1)+manieren(n-2)  
                manieren(1)                n=1  
                    if n<=1:  
                        return 1  
                    return manieren(n-1)+manieren(n-2)
```

Algoritme van Euclides voor GCD

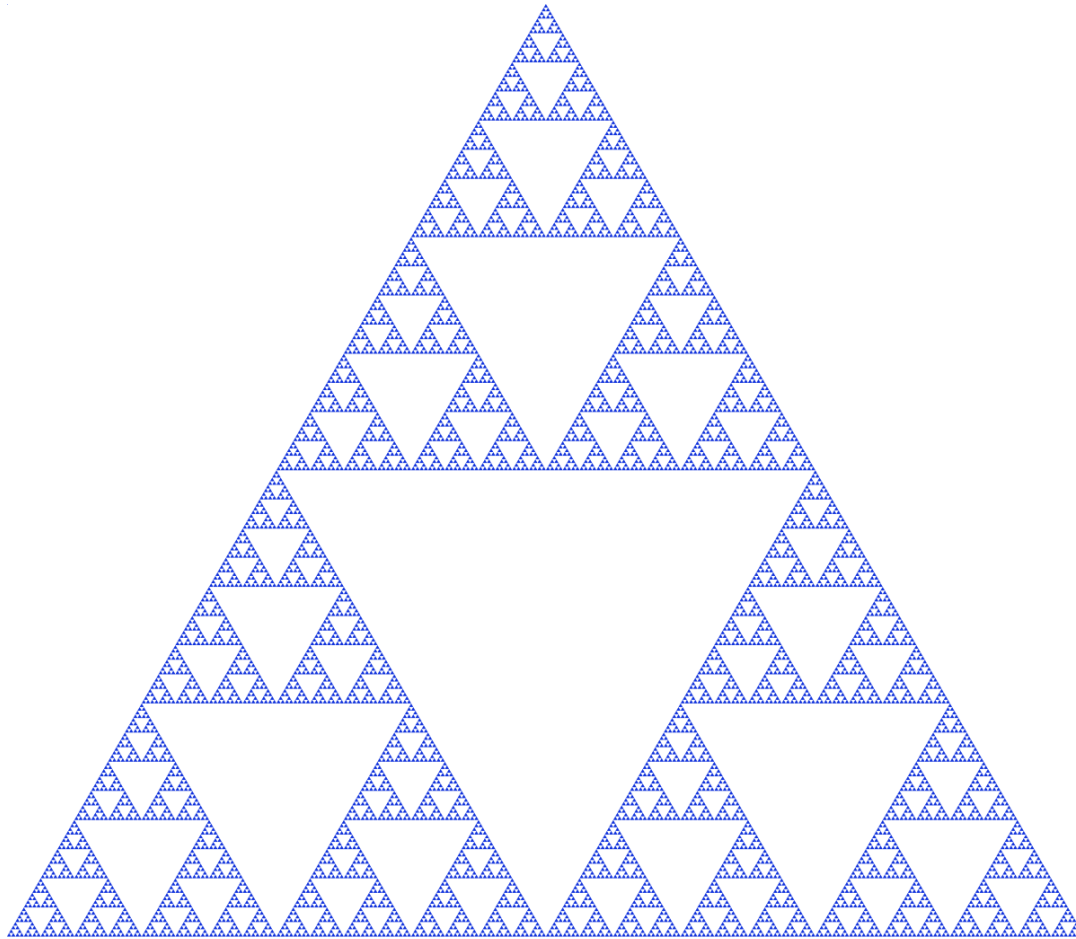
$$\text{gcd}(x, y) = \begin{cases} x & \text{if } y = 0 \\ \text{gcd}(y, \text{remainder}(x, y)) & \text{if } y > 0 \end{cases}$$

Pseudocode (recursive):

```
function gcd is:  
  input: integer x, integer y such that x > 0 and y >= 0  
  
    1. if y is 0, return x  
    2. otherwise, return [ gcd( y, (remainder of x/y) ) ]  
  
end gcd
```

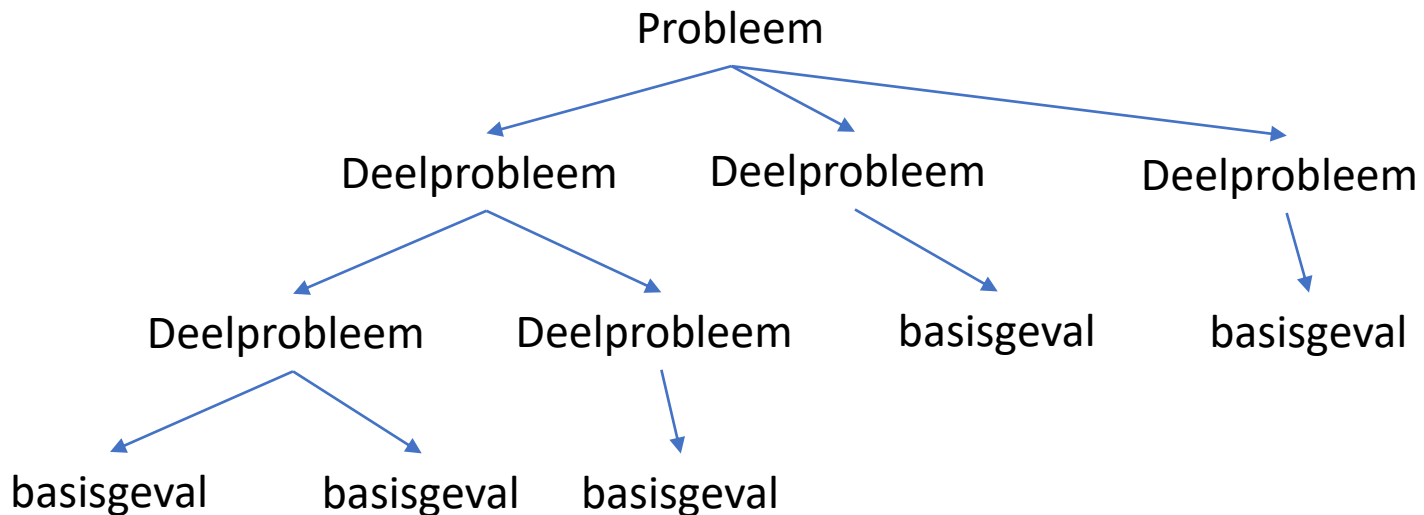
[https://en.wikipedia.org/wiki/Recursion_\(computer_science\)](https://en.wikipedia.org/wiki/Recursion_(computer_science))

Sierpinski driehoek



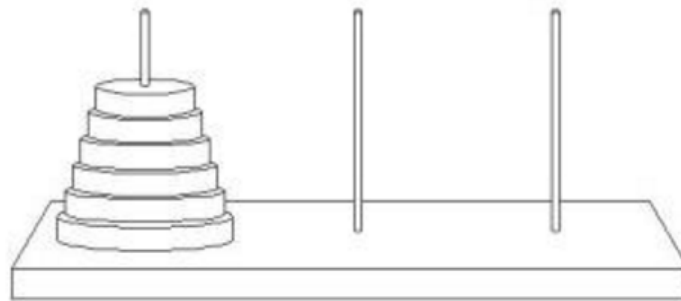
Divide and Conquer

- Divide and conquer = oplossingsstrategie
 - Complex probleem wordt opgesplitst in deelproblemen
 - Deelproblemen zijn eenvoudiger
 - Dit doen we recursief tot basisgeval met triviale oplossing



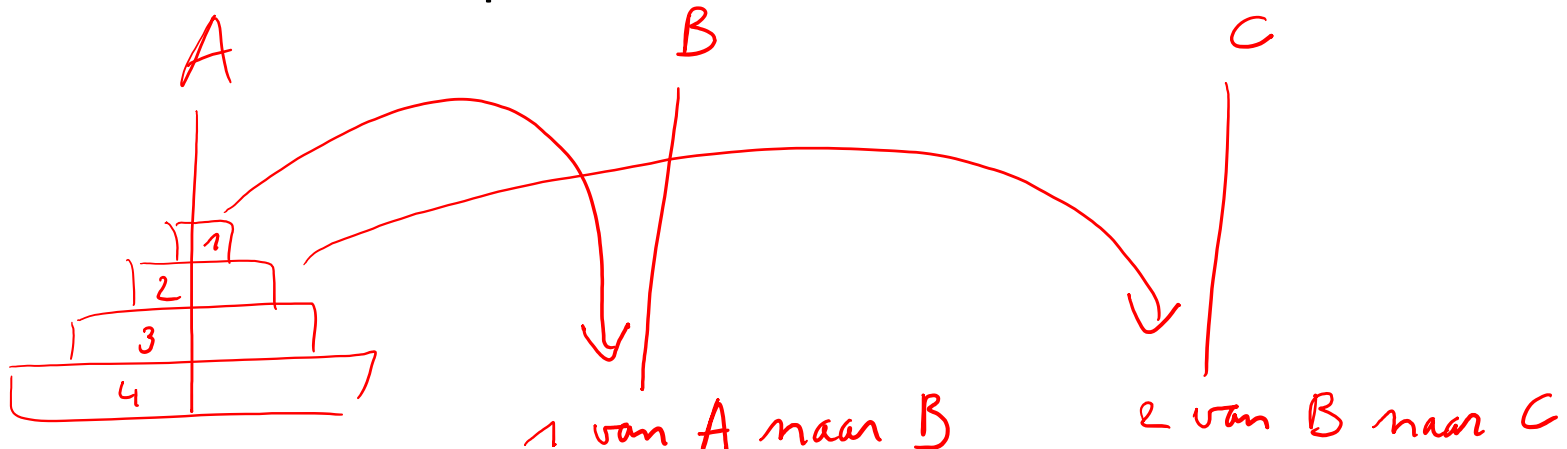
Torens van Hanoi

- Verplaats de toren van de linkerkant naar de rechterkant, met behulp van de middelste staak
 - Slechts 1 schijf per keer verplaatsen
 - Nooit een grotere schijf op een kleinere



Torens van Hanoi

- Om n schijven van staak 1 naar staak 3 te verplaatsen:
 - Verplaats bovenste $n-1$ schijven naar staak 2, gebruik staak 3 als hulpstaak
 - Verplaats laatste schijf naar staak 3
 - Verplaats $n-1$ schijven op staak 2 naar staak 3, gebruik staak 1 als hulpstaak.



Torens van Hanoi

- Input:
 - Startstaak
 - Eindstaak
 - Hulpstaak
 - Aantal n op startstaak die verplaatst moeten worden (altijd $1 \dots n$)
- Output:
 - Sequentie van bewegingen

Subset Sum Probleem

https://en.wikipedia.org/wiki/Subset_sum_problem

Gegeven een lijst getallen, kunnen we een niet-lege deellijst maken die som 0 heeft?

Bvb:

1,2,5,-3,6,-8 : JA $\rightarrow 1+2-3$

2,-3,2,-6 : NEEN

Subset Sum Probleem

- “Divide and conquer” benadering (verdeel en heers):
 - Alle mogelijkheden die het eerste getal van de lijst bevatten
 - Alle mogelijkheden die het eerste getal van de lijst NIET bevatten
- `subsetsum(L,som=0)`
 - True als we met L som kunnen maken
 - False anders

Subset Sum Probleem

Kunnen we 0 maken met 1,2,5,-3,6,-8 ?

Subset Sum Probleem

Kunnen we 0 maken met 1,2,5,-3,6,-8 ?

Ja, als we ofwel:

-1 kunnen maken met 2,5,-3,6,-8

0 kunnen maken met 2,5,-3,6,-8

Subset Sum Probleem

Kunnen we 0 maken met 1,2,5,-3,6,-8 ?

Ja, als we ofwel:

-1 kunnen maken met 2,5,-3,6,-8

Ja als we ofwel:

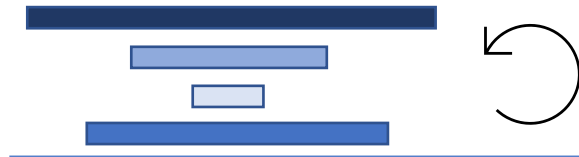
-3 kunnen maken met 5,-3, 6, -8

-1 kunnen maken met 5,-3, 6, -8

0 kunnen maken met 2,5,-3,6,-8

Pannenkoeken sorteren

- N pannenkoeken, allemaal verschillende grootte
- Sorteert van groot naar klein
- Enige operatie: schep tussen de stapel steken en omdraaien



- Kan je volgende stapel sorteren in 4 stappen?



* Hoeveel stappen heb je nodig om de moeilijkste stapel met 6 pannenkoeken te sorteren? Geef zulk een “moeilijkste stapel”.