Recursie vs Iteratie

2019

For, While in Python

Python kent
loop-constructies
zoals while en for

Deze zijn echter niet
noodzakelijk, omdat
ze altijd mbv een if en
recursie zijn te
realiseren.

Soms eleganter!

```
def fac(n):
    if n ==0:
        return 1
    else:
        return n * fac(n-1)

def facloop(n):
    res = 1
    k = 1
    while k <= n:
        res = res * k
        k = k+1
    return res

print(fac(7))
print(facloop(7))
```

Voorbeeld 2

Soms een beetje ingewikkelder.

Maar in principe kan het altijd!

Wat is het recept?

```
def smallest(a):
    s = a[0]
    n = len(a)
    k = 1
    while k < n:
        if a[k] < s:
            s = a[k]
        k = k+1
    return s

def smalrec(a,s,k):
    if k == len(a):
        return s
    elif a[k] < s:
        return smalrec(a,a[k],k+1)
    else:
        return smalrec(a,s,k+1)

def smallestrec(a):
    return smalrec(a,a[0],1)

a = [4,7,3,9,4,2,5,7]
print(smallest(a))
print(smallestrec(a))
```

Recursie

Voor fac en smallest waren de loop en recursie oplossing ongeveer even moeilijk te bedenken.

Als je een loop oplossing hebt is die eenvoudig om te zetten naar een recursieve oplossing.

Omgekeerd is het echter niet altijd (eenvoudig) mogelijk om een recursieve oplossing met een loop te realiseren

Torens van Hanoi

Tijdens Logica al behandeld.

Probleem: toren van n schijven

Verplaats schijven (1 voor 1) van paal 1
naar paal 2 waarbij paal 3 als hulp wordt gebruikt en waarbij een grotere schijf nooit op een kleinere mag komen.

Bij logica behandeld: er zijn 2^n - 1 verplaatsingen nodig

Maar nu willen de verplaatsingen printen: 1-3, 1-2, 3-2

Torens van Hanoi

Het probleem kan iteratief opgelost worden, maar dit is heel lastig!

Recursief is de oplossing veel eenvoudiger.

De aanpak werkt net als bij een inductiebewijs in de logica:

Je hebt een probleem van grootte n.

- wat moet je doen als n = 1 (of 0, dat hangt van het probleem af)?
- hoe reduceer je het probleem van grootte n tot een probleem van grootte n-1?

Voorbeeld Faculteit

Kijk nog eens naar fac(n)

De resursieve functie valt in 2 delen uit één:

Het geval n = 0 (of 1): dan is het resultaat 1

Het reduceren van fac(n) naar fac(n-1): `fac (n) = n * fac(n-1)`
Dit volgt direct uit de definitie van faculteit: `n! = 1*2*...*(n-1)*n`

Dus we lossen het probleem voor n op door de oplossing voor n-1 te nemen en deze te vermenigvuldigen met n.

Het n = 0 geval is nodig om dit proces te laten eindigen.

Torens van Hanoi

Als we deze techniek voor de Torens van Hanoi willen toepassen, moeten we dus 2 dingen doen:

Oplossing voor n = 0 bedenken

Gegeven een oplossing voor n-1, wat moet er gebeuren om hier een oplossing voor n van te maken?

n=0: dit is makkelijk, er hoeft helemaal niets te gebeuren!

Torens van Hanoi: `n-1 -> n`

In woorden:
Gegeven 3 palen a,b,c en je weet hoe je een toren van hoogte n-1 van a naar b moet verplaatsen met c als hulp, hoe verplaats je nu een toren van hoogte n?

De oplossing voor n wordt dan:

1. Verplaats eerst de bovenste n-1 schijven van paal a naar paal c met paal b als hulp.
2. Verplaats nu de onderste schijf van paal a naar b.
3. Verplaats nu de n-1 schijven van paal c naar paal b met paal a als hulp

```
def hanoi(n,a,b,c):
    if n > 0:
        hanoi(n-1,a,c,b)
        print(a,b)
        hanoi(n-1,c,b,a)

hanoi(5,1,2,3)
```

Iteratieve Oplossing Hanoi?

Analyseer het patroon van de verplaatsingen en bedenk dat het aantal verplaatsingen 2^n-1 is.

Veel succes!

Meer Recursie

Kleinste element in een array

Als lengte 1 dan waarde element

Als lengte n vergelijk de waarde van het eerste element van het minimum van de rest van het array.

```
def smallest(a):
    return smallrec(a,0,len(a))

def smallrec(a,start,n):
    if start == n-1:
        return a[start]
    else:
        k = smallrec(a,start+1,n)
        if a[start] < k:
            return a[start]
        else:
            return k

a = [4,2,7,4,3,9,8]

print(smallest(a))
```