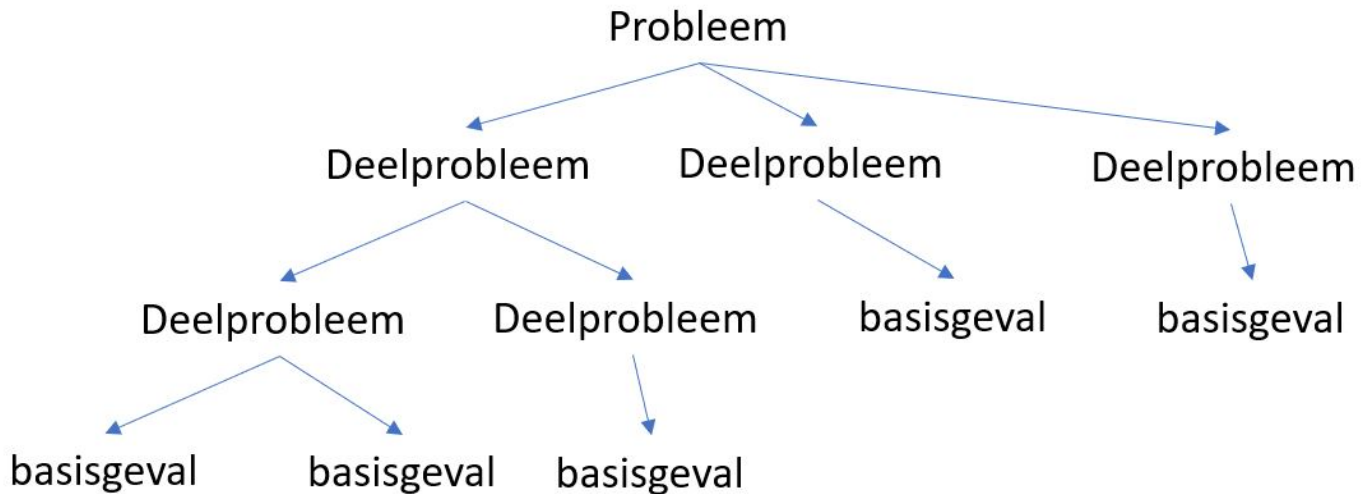


Divide and Conquer

Recursie wordt typisch gebruikt bij *divide-and-conquer* strategieën: een probleem opsplitsen in deelproblemen van hetzelfde type maar met een lagere complexiteit.



Divide and conquer = oplossingsstrategie

- Complex probleem wordt opgesplitst in deelproblemen
- Deelproblemen zijn eenvoudiger
- Dit doen we recursief tot basisgeval met triviale oplossing

Voorbeelden Divide-and-Conquer

Als we een lesrooster willen opstarten dan kunnen we de theorieles van Inleiding programmeren op maandag uren 1 en 2 zetten, en moeten we alle vakken behalve inleiding programmeren inplannen in alle uren behalve maandag 1 en 2. Of, IP moet op maandag uren 3 en 4 en alle andere vakken in een van de andere slots, etc. Hier splitsen we dus op in volgende deelproblemen:

- maak een rooster met IP om maandag 1&2
- maak een rooster met IP op maandag 3&4
- ...
- maak een rooster met IP op vrijdag 7&8

Het basisgeval is dat alle vakken correct ingepland zijn.

Je wil de kortste route van gebouw G naar jou thuis plannen. Een van de volgende deelproblemen zal de kortste route geven:

- verlaat gebouw G langs de achterkant en bereken de kortste route vanaf de achterkant van gebouw G tot bij je thuis,
- of: verlaat gebouw G langs de voorkant en bereken de kortste route vanaf de voorkant van gebouw G tot bij je thuis.

Het basisgeval is dat je thuis bent; dan is de kortste weg gewoon thuis blijven. Dit basisgeval komt ook wel eens voor bij lessen om 8:30 's morgens.

Betalen met minimum aantal munten

Volgende recursieve functie berekent of je een gegeven bedrag b kan betalen met minder dan n euromunten (1 cent, 2 cent, 5 cent, 10 cent, 20 cent, 50 cent, 1 euro, 2 euro).

```
In [1]: def betalen(b,n,munten):
        """
        Kan ik een bedrag van b cent betalen met minder dan n munten?
        Ik heb de keuze uit munten met bedrag in munten
        """
        if b==0: # basisgeval: exact bedrag bereikt
            return True
        elif b<0 or n==0 or len(munten)==0: # basisgeval: geen oplossing gevonden
            return False
        # algemeen geval
        m=munten[0]
        muntenzonderm=munten[1:]

        return betalen(b-m,n-1,munten) or betalen(b,n,muntenzonderm)

def betalenMetMax(b,n):
    return betalen(b,n,[1,2,5,10,20,50,100,200])

print(betalenMetMax(134,4))
print(betalenMetMax(134,5))
print(betalenMetMax(134,7))
print(betalenMetMax(135,4))
```

```
False
True
True
True
```

Sublijsten

Gegeven een lijst L , maak een lijst van alle deellijsten van L . Dit doen we als volgt:

- Als $L=[]$, dan is de lijst van deellijsten gelijk aan $[[]]$; enkel de lege lijst is een deellijst van de lege lijst
- In het andere geval nemen we 1 element uit L weg en krijgen we $L_{zondera}$. We genereren alle deellijsten van $L_{zondera}$. Elke deellijst van $L_{zondera}$ is een deellijst van L en bovendien is elke deellijst van $L_{zondera}$ waar we a aan toevoegen ook een deellijst.

```
In [3]: from copy import copy
```

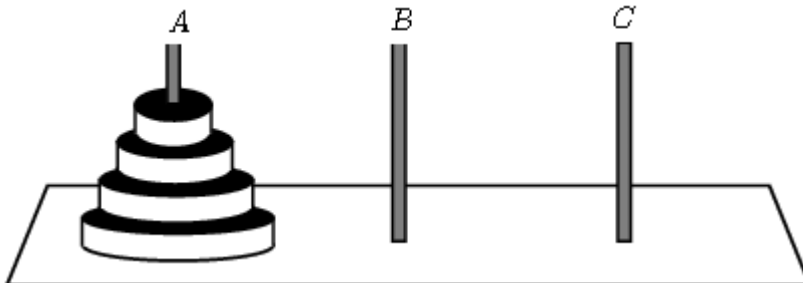
```
def deellijsten(L):  
    if L==[]:  
        return [ [] ]  
    else:  
        a=L[0]  
        Lzondera=L[1:]  
        deellijstenLzondera=deellijsten(Lzondera)  
        result=copy(deellijstenLzondera)  
        for dl in deellijstenLzondera:  
            result.append([a]+dl)  
        return result
```

```
L=[1,2,3,4]  
print(deellijsten(L))
```

```
[[], [4], [3], [3, 4], [2], [2, 4], [2, 3], [2, 3, 4], [1], [1, 4], [1, 3],  
[1, 3, 4], [1, 2], [1, 2, 4], [1, 2, 3], [1, 2, 3, 4]]
```

Torens van Hanoi

Bij het spel "[Torens van Hanoi \(https://nl.wikipedia.org/wiki/Torens_van_Hanoi\)](https://nl.wikipedia.org/wiki/Torens_van_Hanoi)", moet je schijven die van groot naar klein gestapeld zijn op een staak, verplaatsen naar een andere staak, zonder ooit een grotere schijf op een kleinere schijf te plaatsen. Hierbij kan gebruik gemaakt worden van één hulpstaak.



We moeten dus in dit geval 4 schijven verplaatsen van staak A naar staak C, waarbij we tussendoor staak B mogen gebruiken.

We raden je aan om eerst zelf even te proberen te bedenken hoe je dit probleem kan oplossen, vooraleer verder te lezen.

Je kan dit probleem als volgt oplossen:

hanoi met n schijven op staak A, te verplaatsen naar staak C, B is hulpstaak: - verplaats eerst de bovenste n-1 schijven van A naar B, je kan C als hulpstaak gebruiken - verplaats de grootste schijf van A naar C - verplaats het torentje met n-1 schijven van B naar C, je kan A als hulpstaak gebruiken

Je hebt dus het probleem (een toren van n schijven) herleidt tot twee kleinere problemen (twee maal een toren met $n - 1$ schijven verplaatsen, en tussendoor de grootste schijf verplaatsen). Elk van de torens met $n - 1$ schijven verplaatsen kan je dan weer opnieuw telkens opsplitsen 2 maal een toren met $n - 2$ schijven verplaatsen, ... Het basisgeval is een toren met 1 schijf verplaatsen, wat triviaal is: verplaats de schijf van A naar C. De code ziet er als volgt uit:

```
In [8]: def hanoi(n,A,B,C):  
        if n==1:  
            print("schijf 1 van",A,"naar",C)  
        else:  
            hanoi(n-1,A,C,B)  
            print("schijf",n,"van",A,"naar",C)  
            hanoi(n-1,B,A,C)  
  
        hanoi(3,"A","B","C")
```

```
schijf 1 van A naar C  
schijf 2 van A naar B  
schijf 1 van C naar B  
schijf 3 van A naar C  
schijf 1 van B naar A  
schijf 2 van B naar C  
schijf 1 van A naar C
```

In []: