



# Constructieve Algoritmen

Bas Terwijn <[b.terwijn@uva.nl](mailto:b.terwijn@uva.nl)>

Algoritmen en Heuristieken

Minor Programmeren

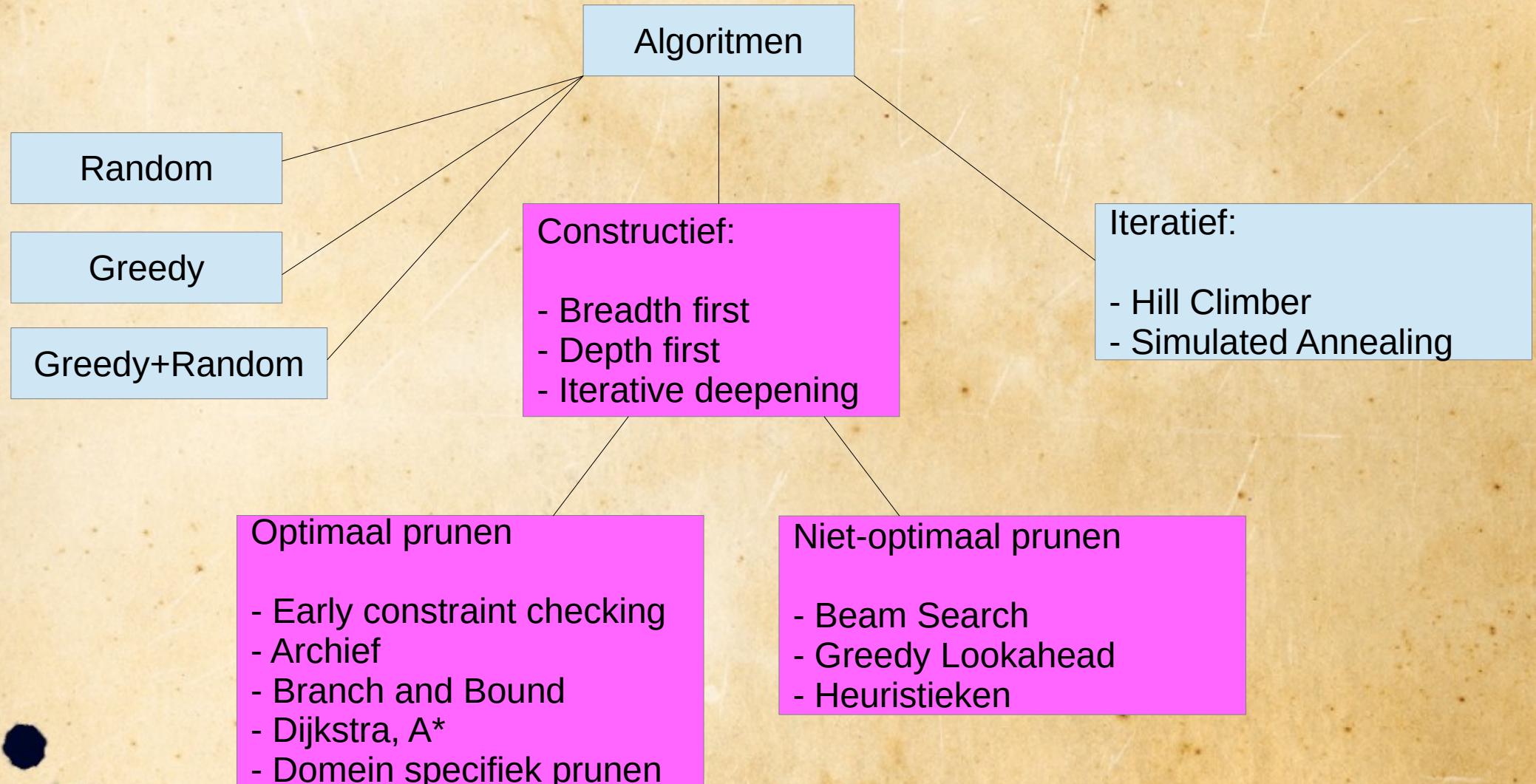
Universiteit van Amsterdam

# Wat ging vooraf

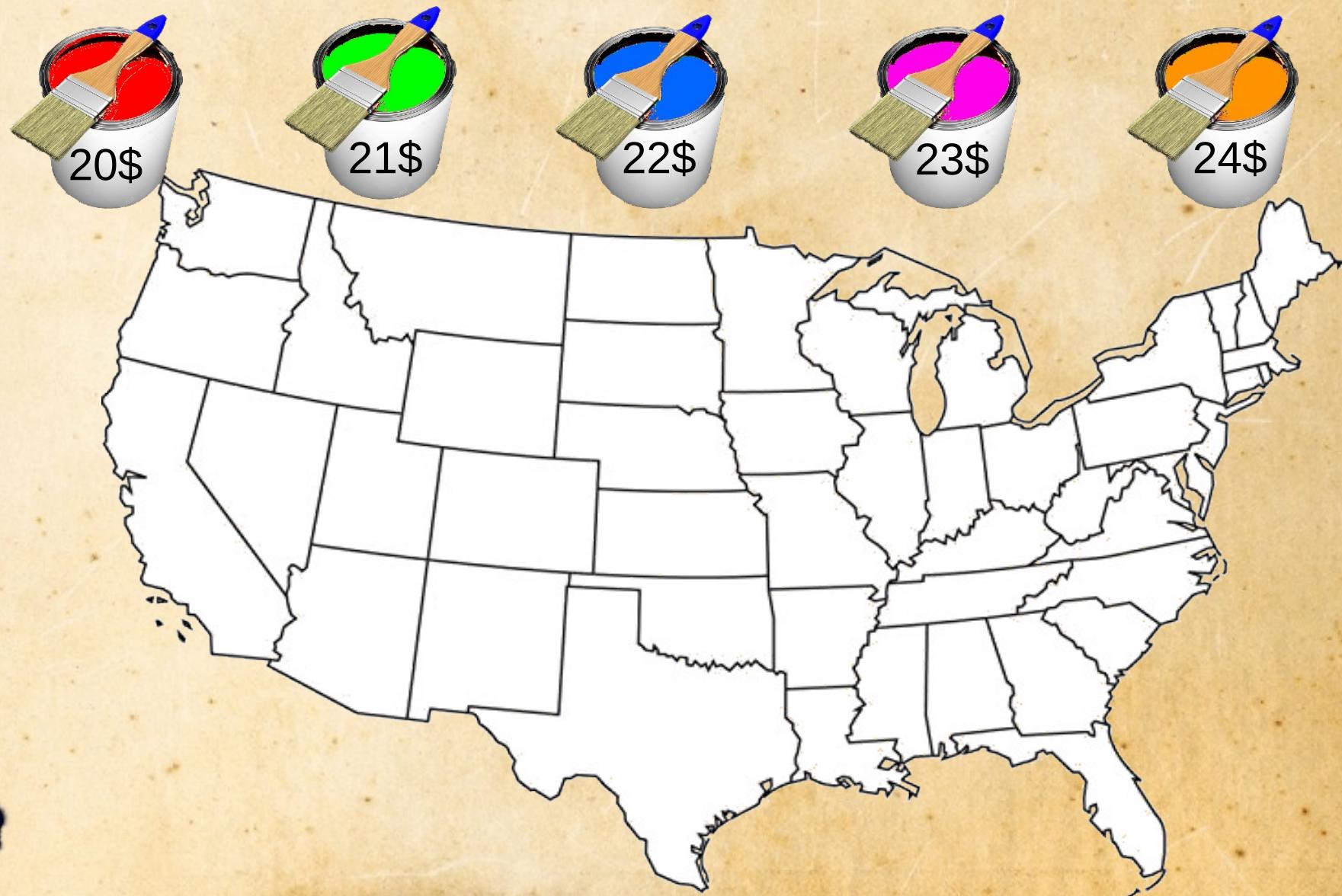


- Kick Off: Wouter Vrieling
- State Space, Representatie: Wouter Vrieling
- Live Coding: Wouter Vrieling, Quinten van der Post

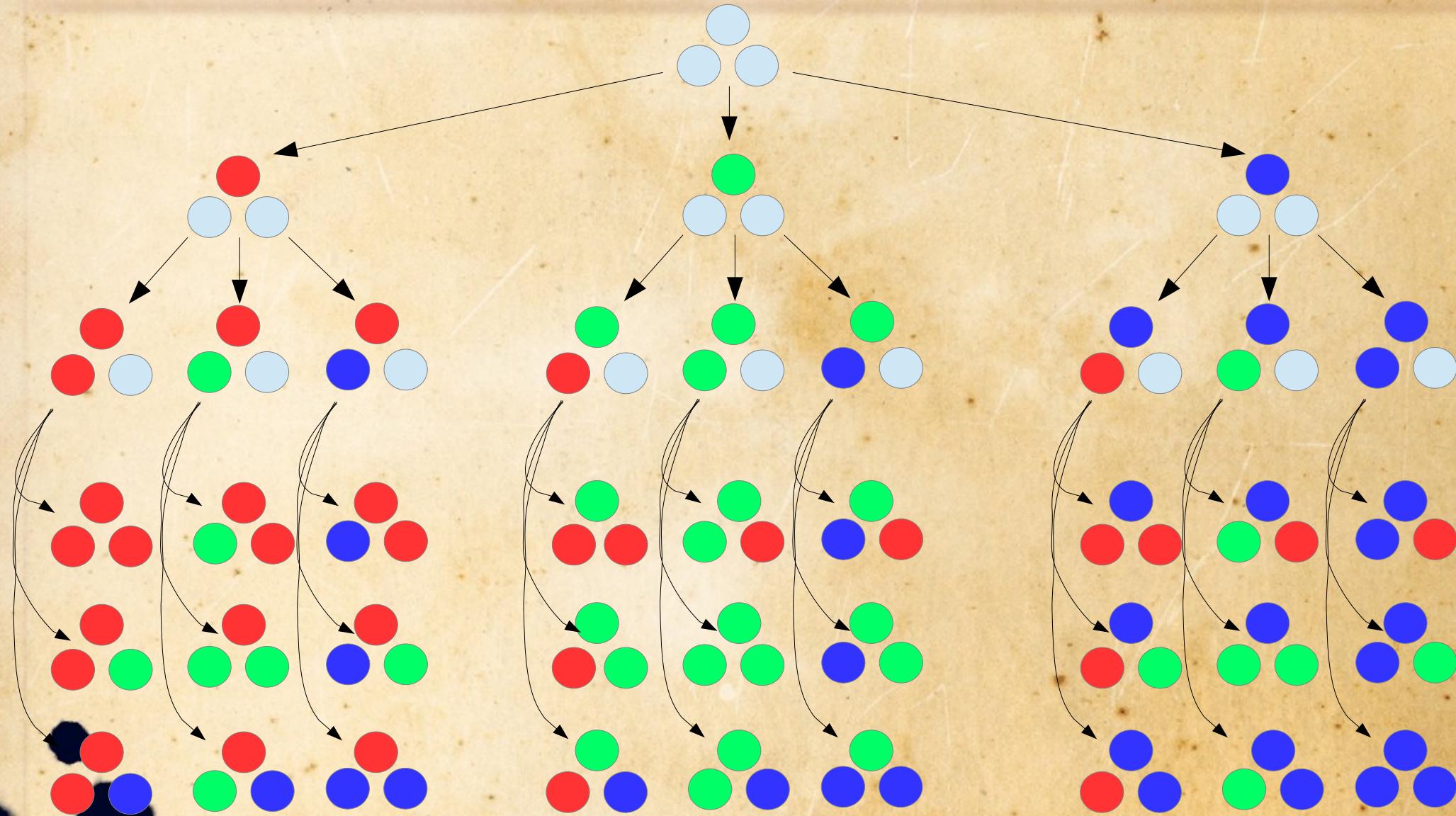
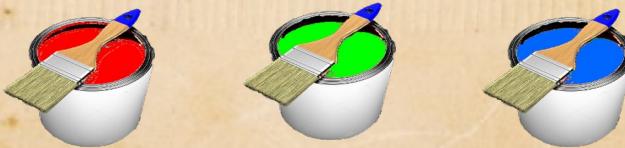
# Algoritmen



# Kaart kleuren



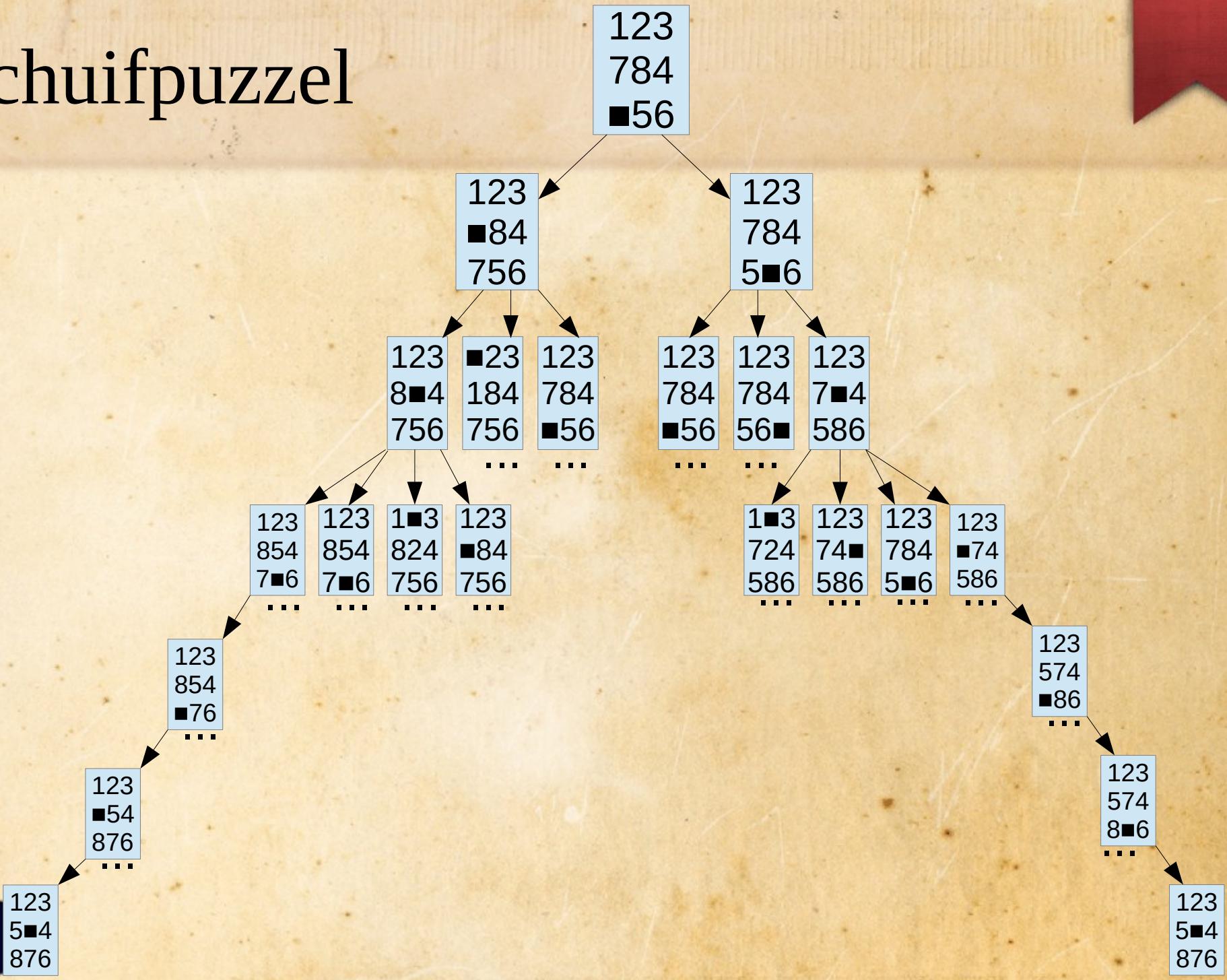
# Kaart kleuren



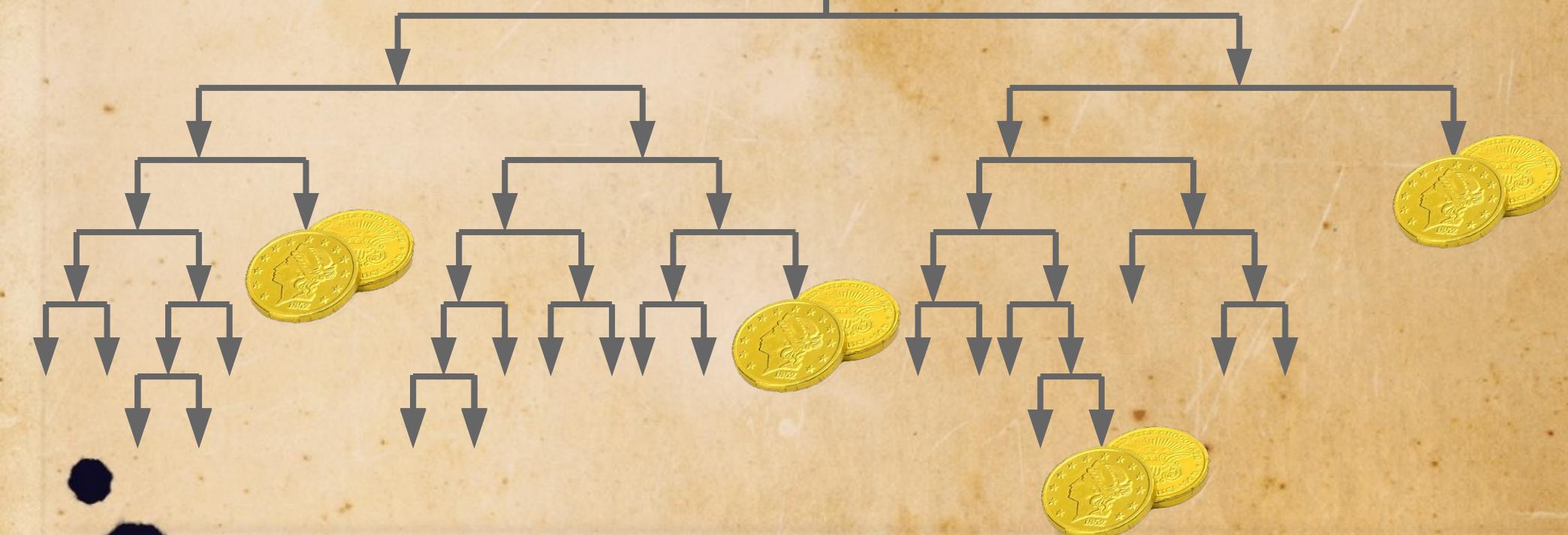
# Schuifpuzzel



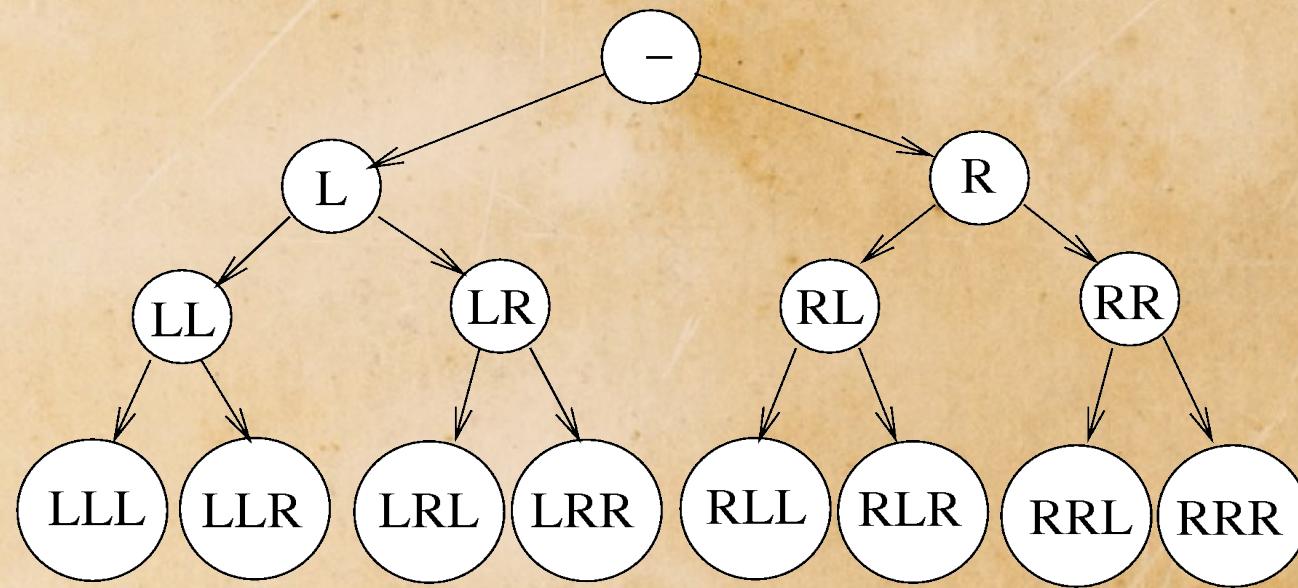
# Schuifpuzzel



# Doolhof



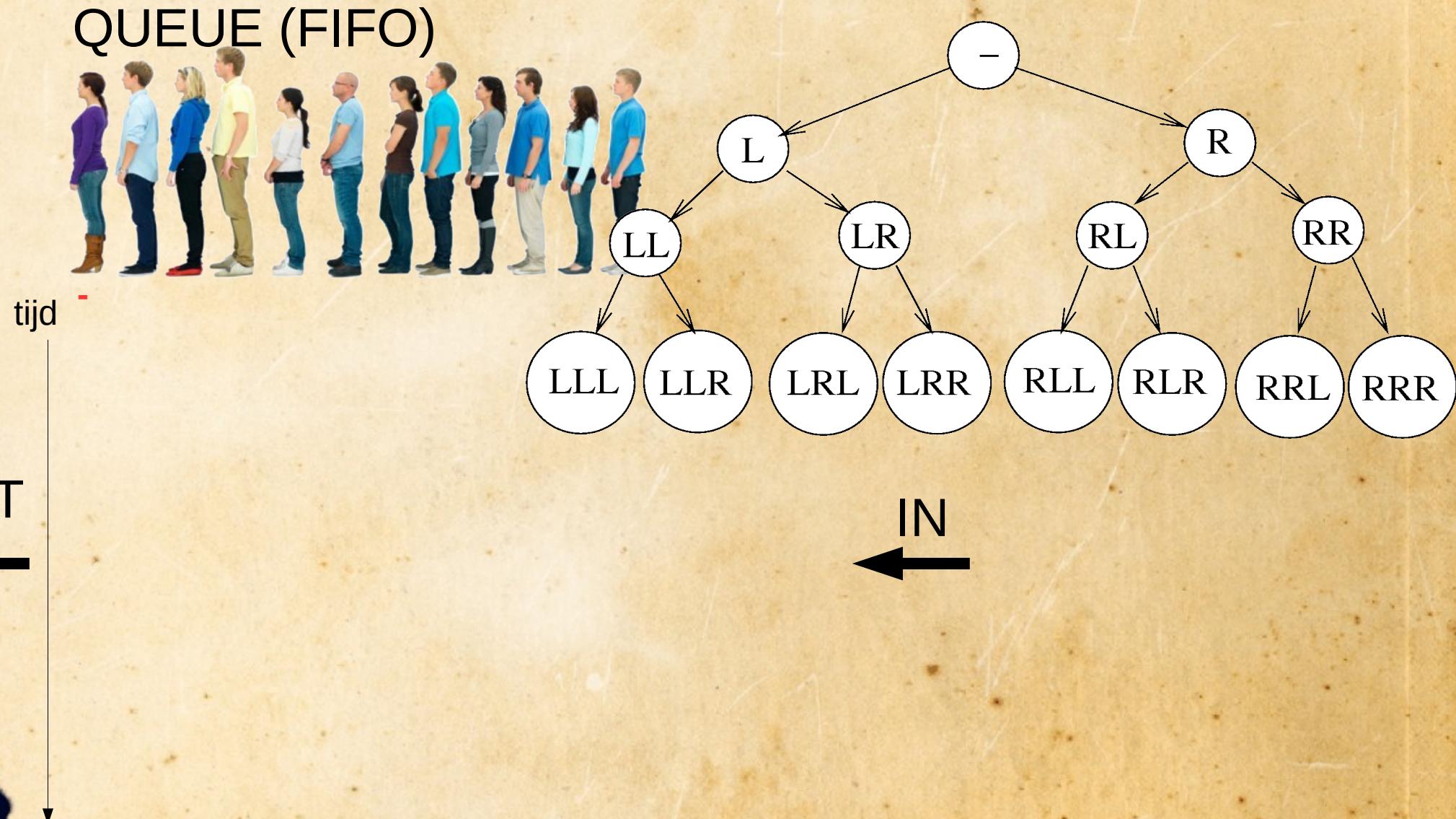
# Doolhof



# Constructieve Algoritmen

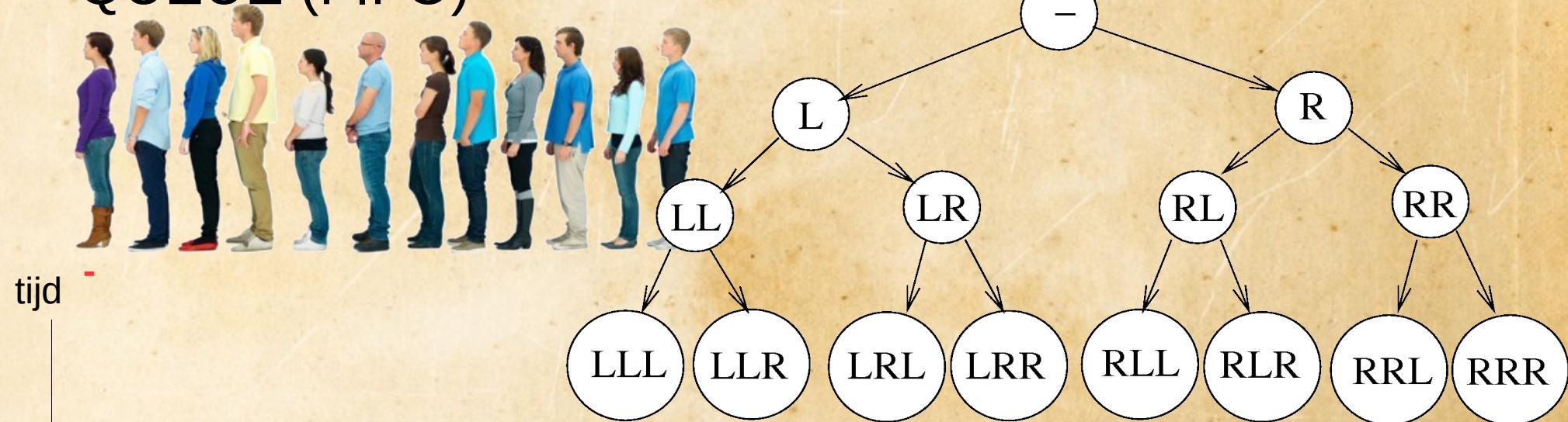


# Breadth first



# Breadth first

QUEUE (FIFO)



# Breadth first

QUEUE (FIFO)

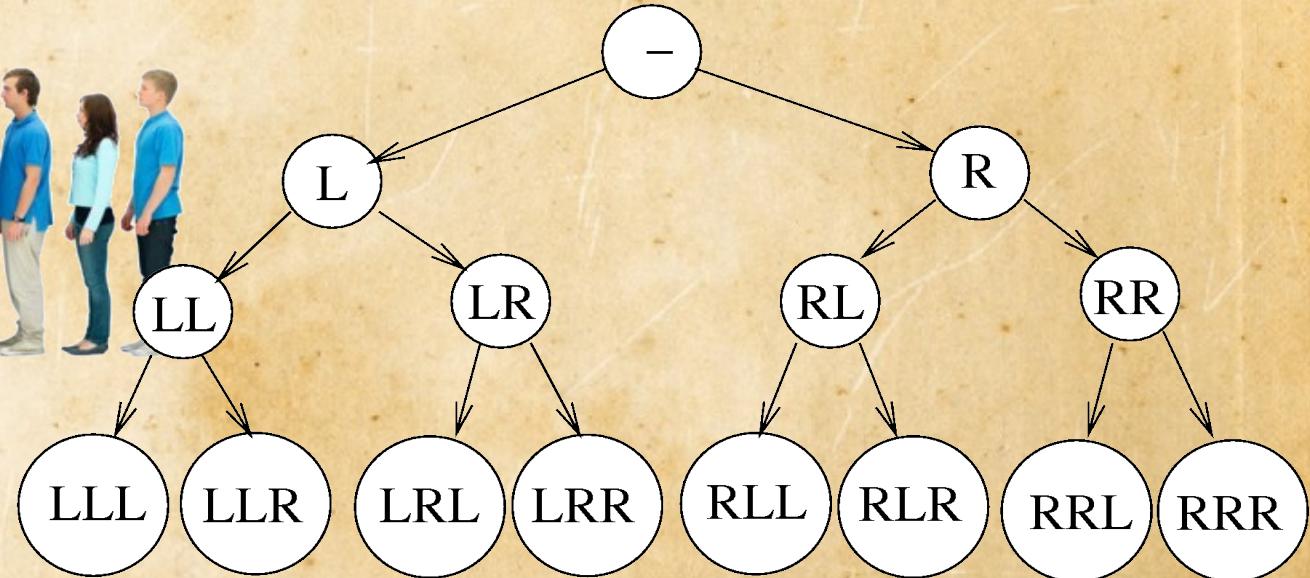


tijd  
- L

OUT



IN

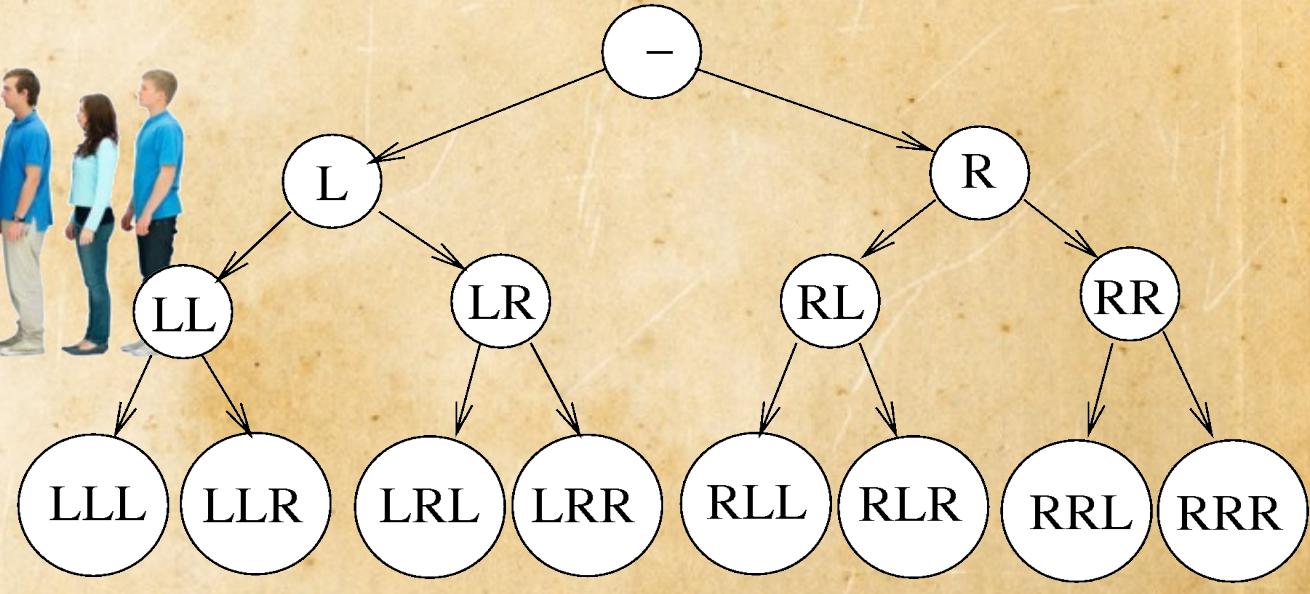


# Breadth first

QUEUE (FIFO)



tijd  
- L, R



OUT



IN

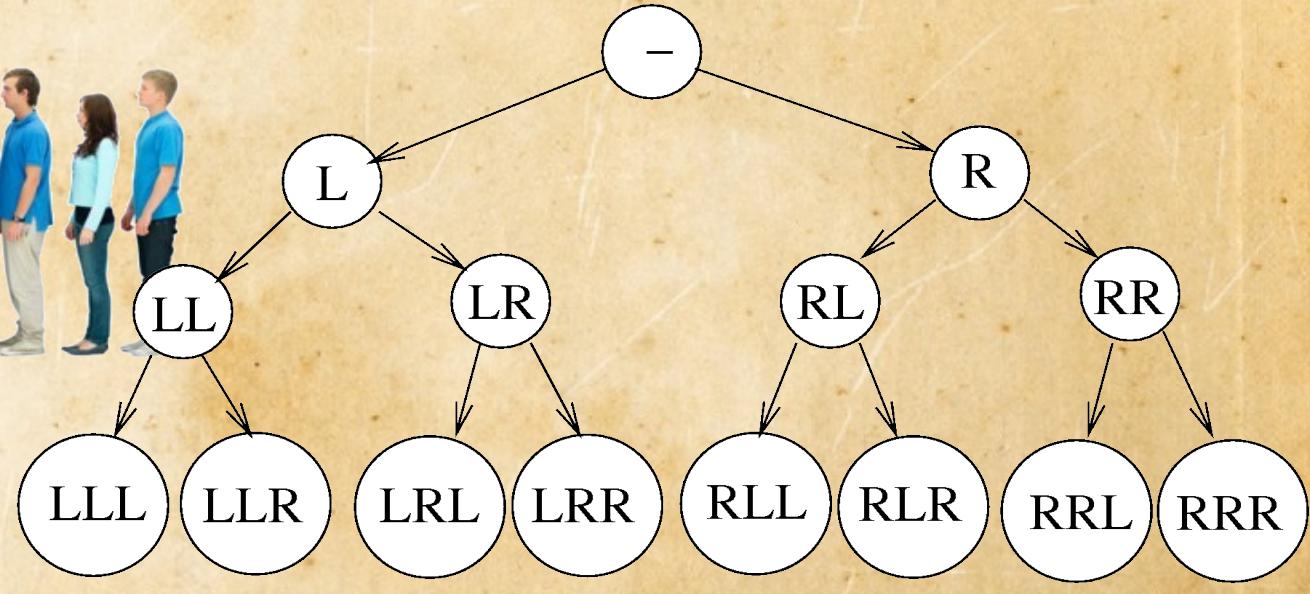


# Breadth first

QUEUE (FIFO)



tijd  
L  
R



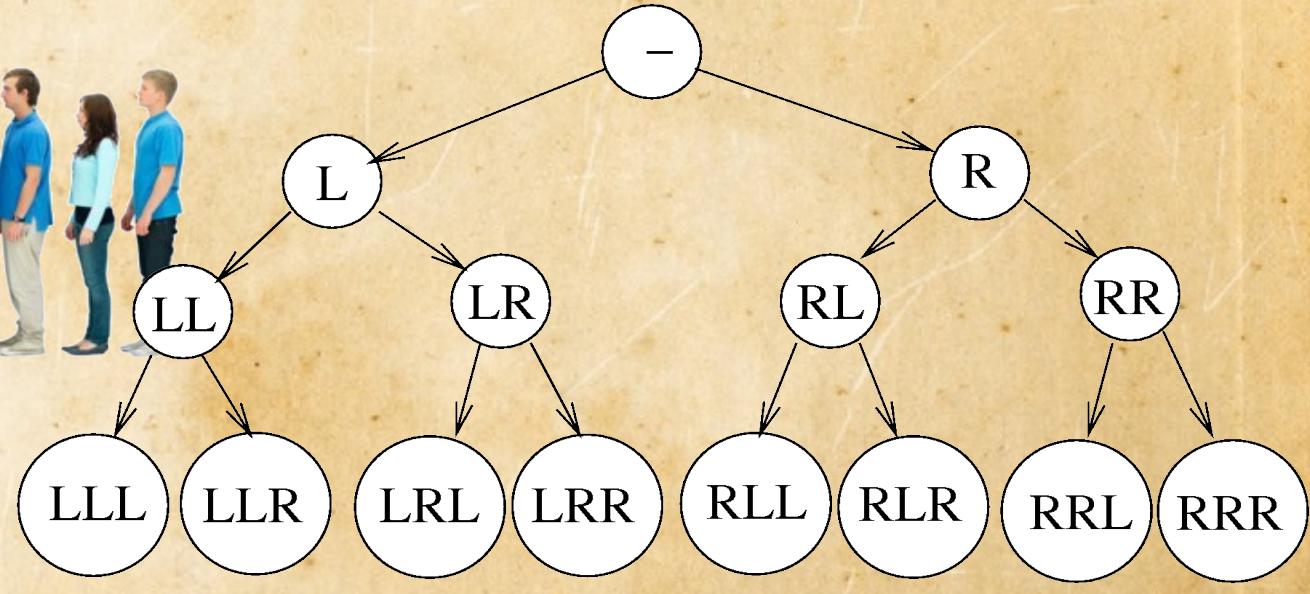
# Breadth first

QUEUE (FIFO)



tijd

L  
R  
R, LL



# Breadth first

QUEUE (FIFO)



tijd

L

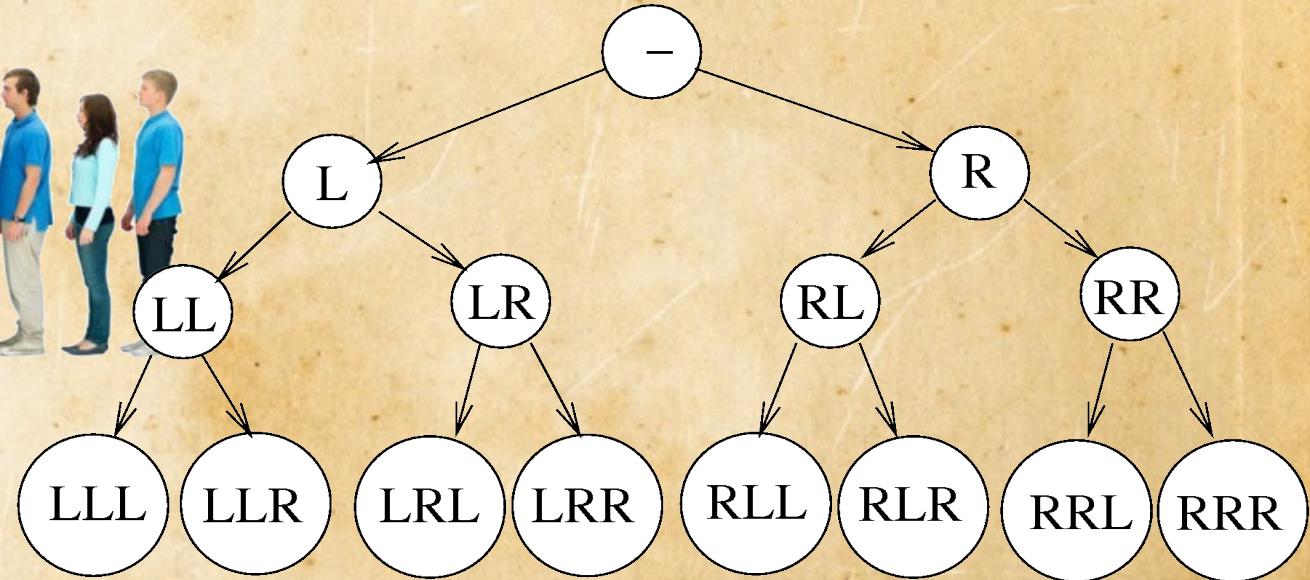
R, LL, LR

L

OUT



IN



# Breadth first

QUEUE (FIFO)



tijd

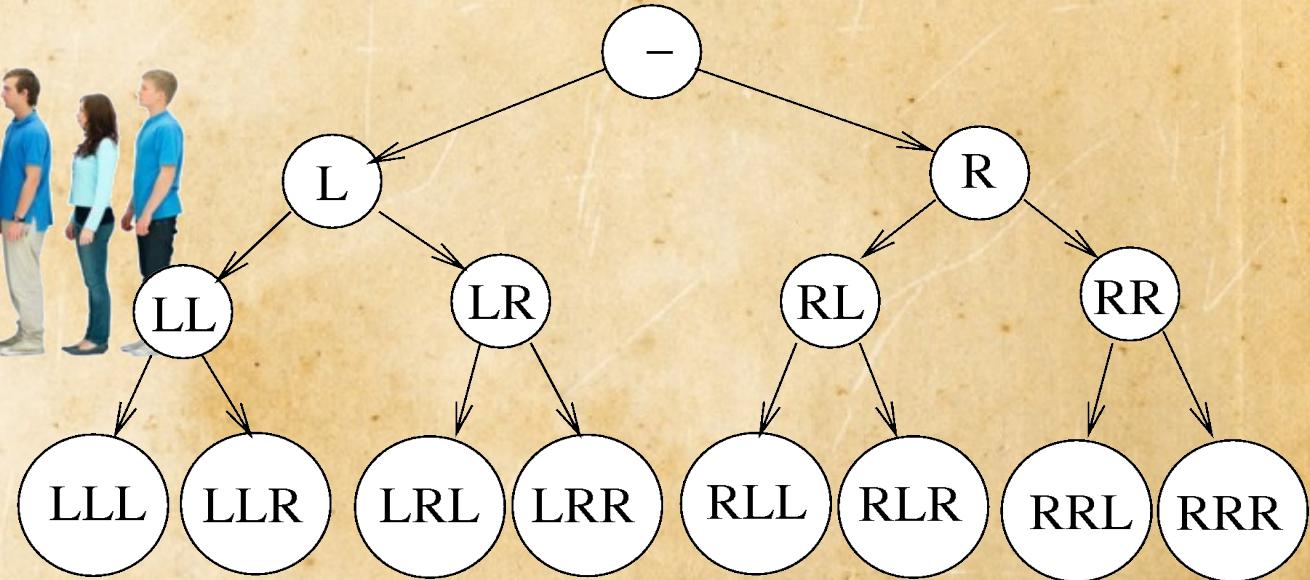
L, R  
R, LL, LR  
LL, LR

R

OUT



IN



# Breadth first

QUEUE (FIFO)



tijd

L, R

R, LL, LR

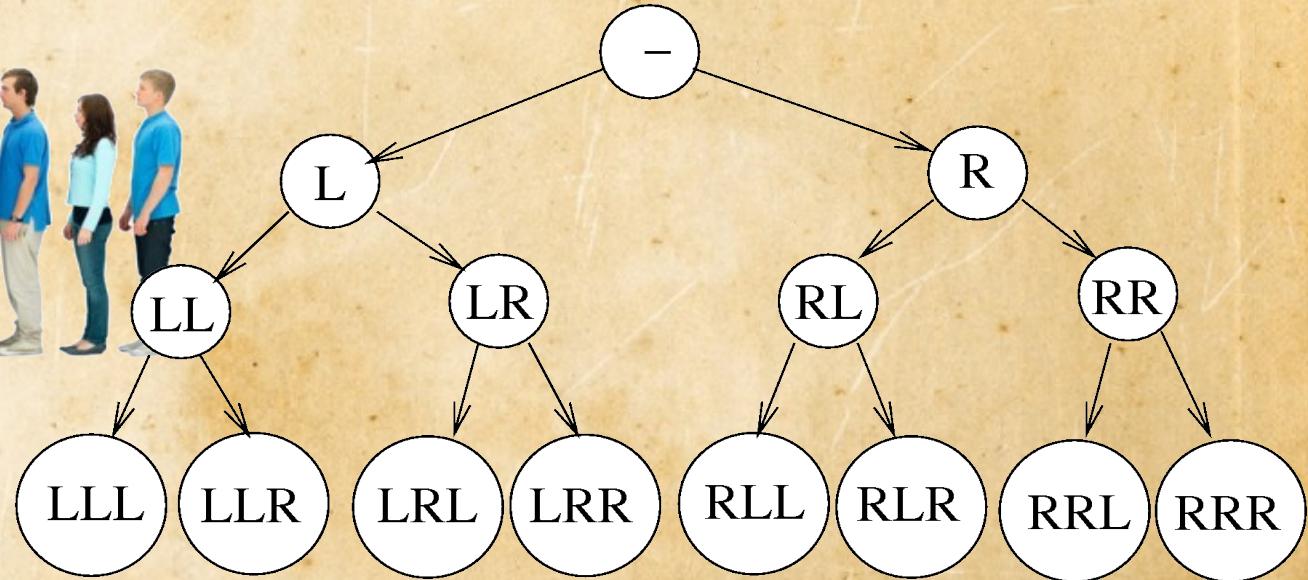
LL, LR, RL

R

OUT



IN



# Breadth first

QUEUE (FIFO)



tijd

L, R

R, LL, LR

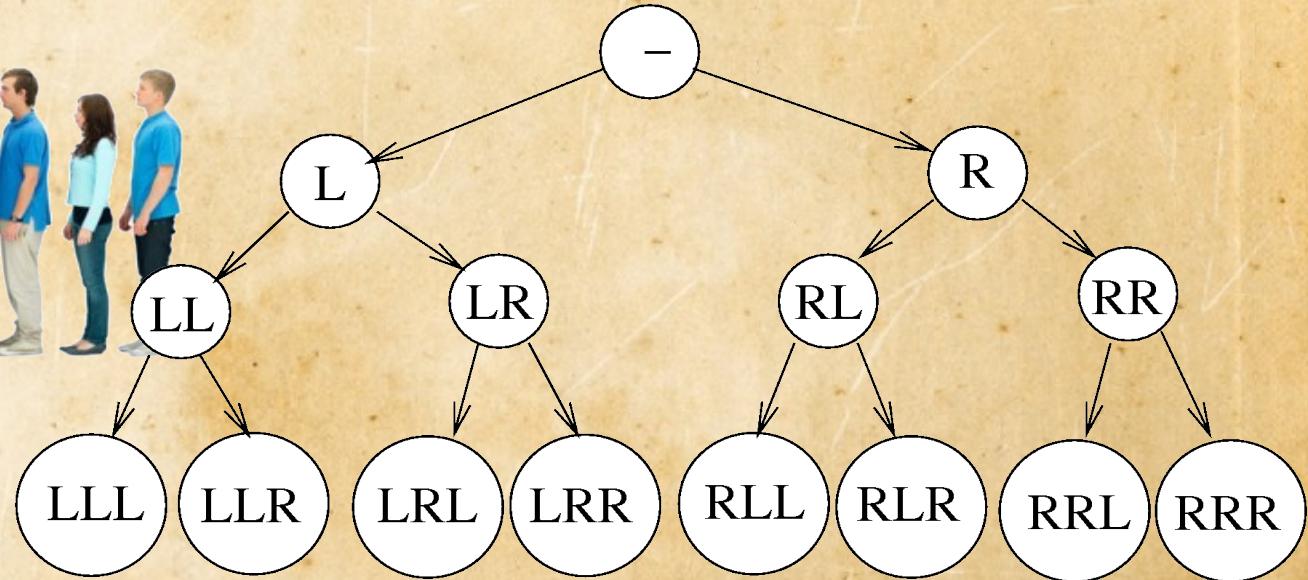
LL, LR, RL, RR

R

OUT



IN



# Breadth first

QUEUE (FIFO)



tijd

L, R

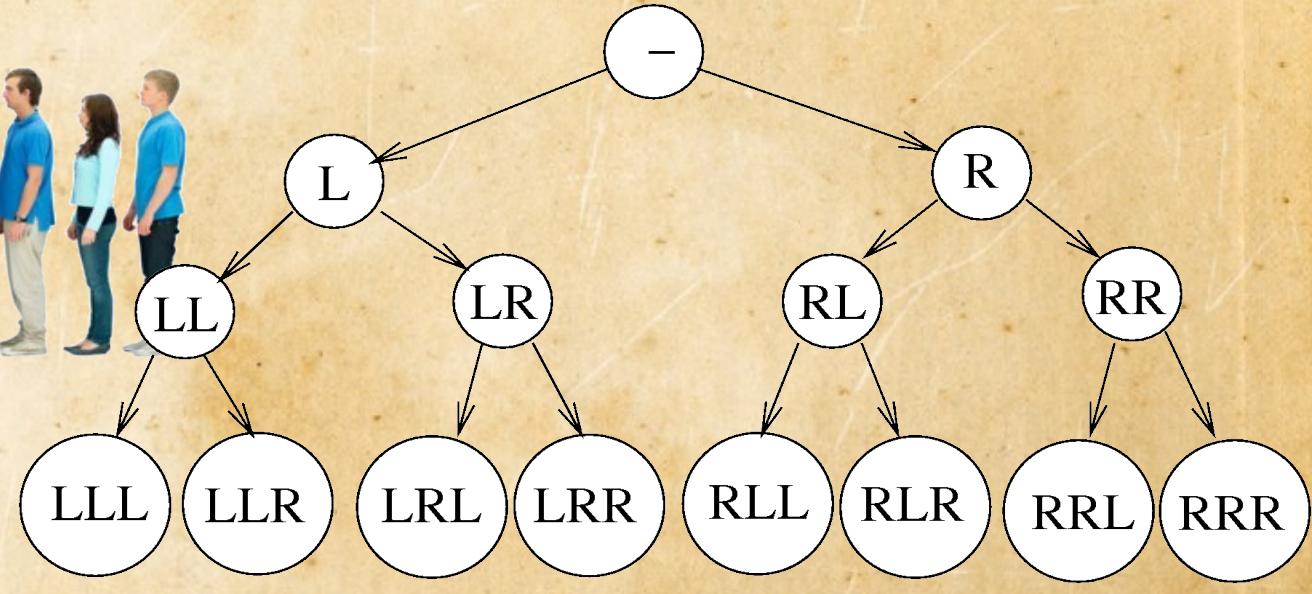
R, LL, LR

LL, LR, RL, RR

LR, RL, RR, LLL, LLR

LL

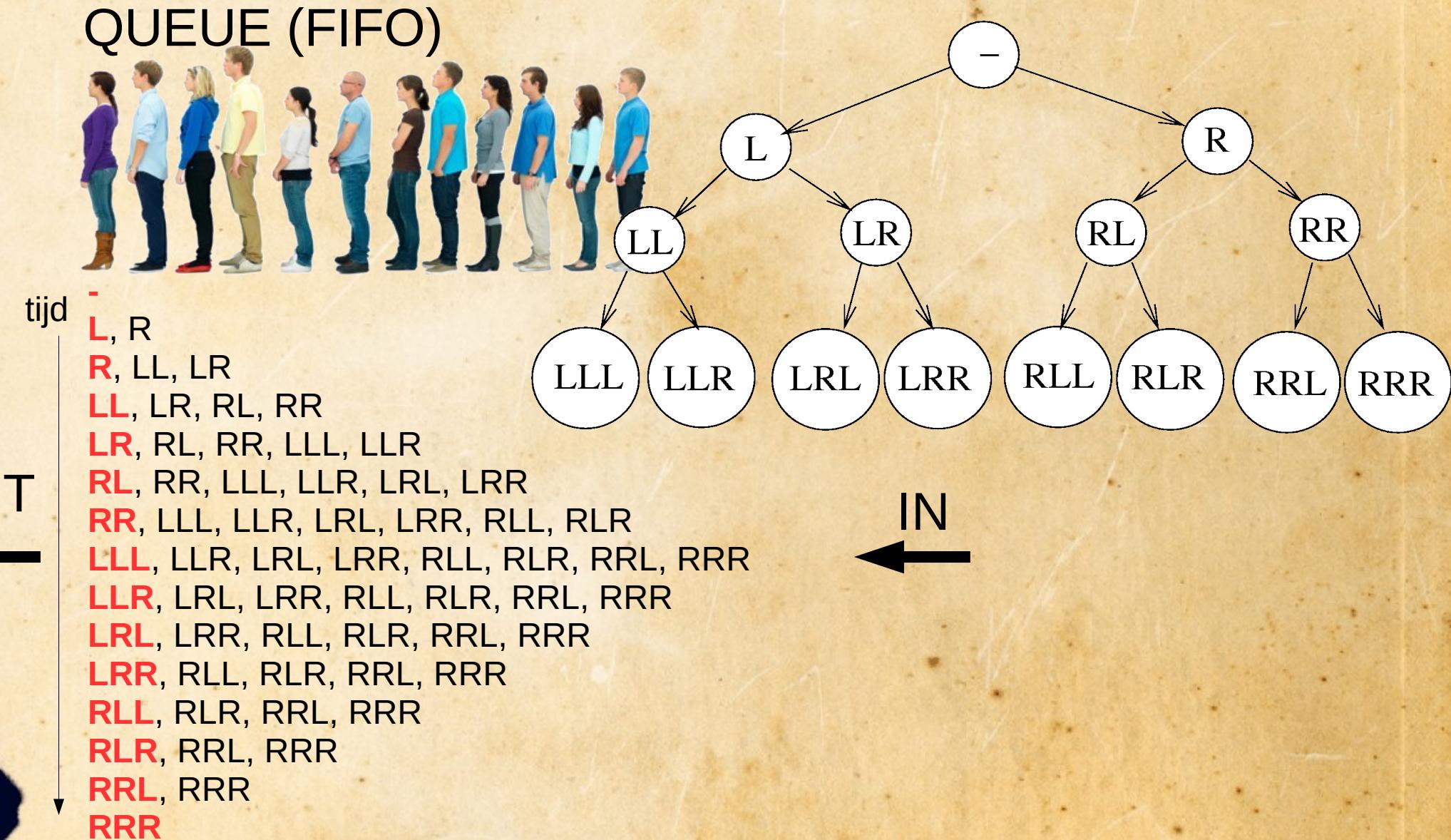
OUT



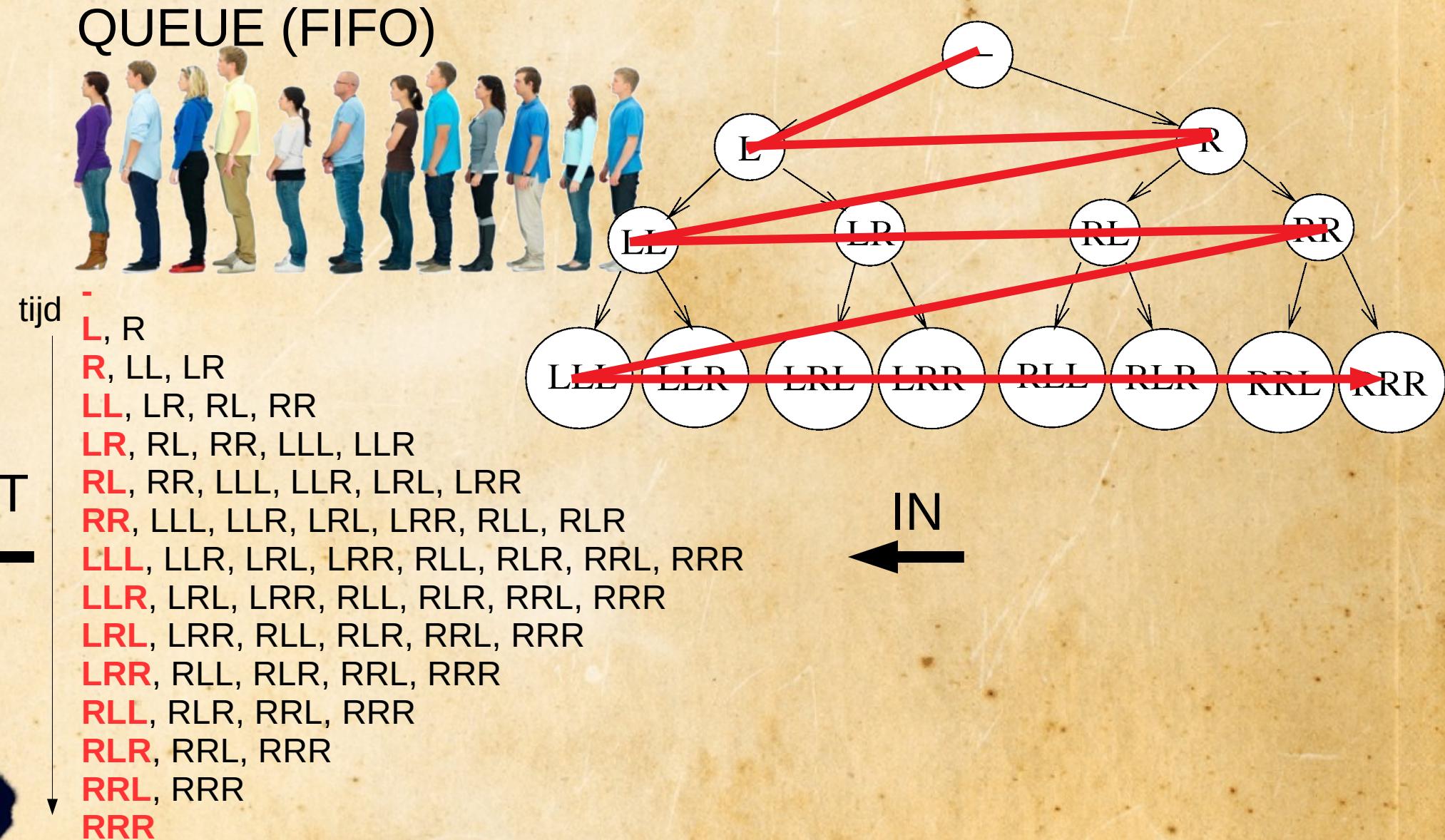
IN



# Breadth first



# Breadth first



# Breadth first in Python

```
import queue
import copy

depth = 3                                     # no deeper than 'depth'
queue = queue.Queue()
queue.put("")                                   # add begin state to queue
while not queue.empty():
    state = queue.get()                         # get first from queue
    print(state)
    if len(state) < depth:                     # stop condition
        for i in ['L', 'R']:                     # for each possible action:
            child = copy.deepcopy(state)         # deepcopy the state
            child += i                          # make new child
            queue.put(child)                   # add new child
```

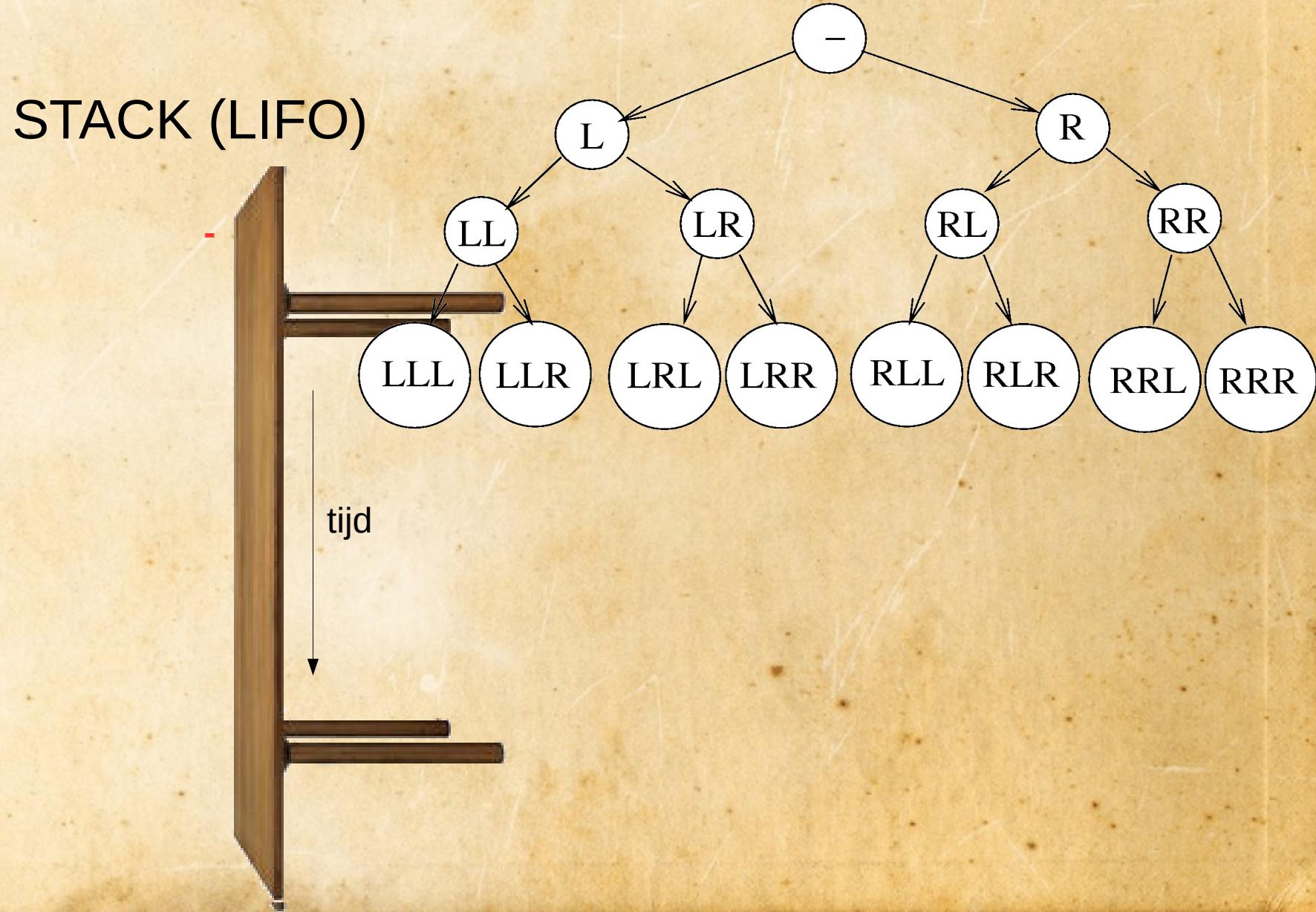
# Breadth first in Python

```
import queue
import copy

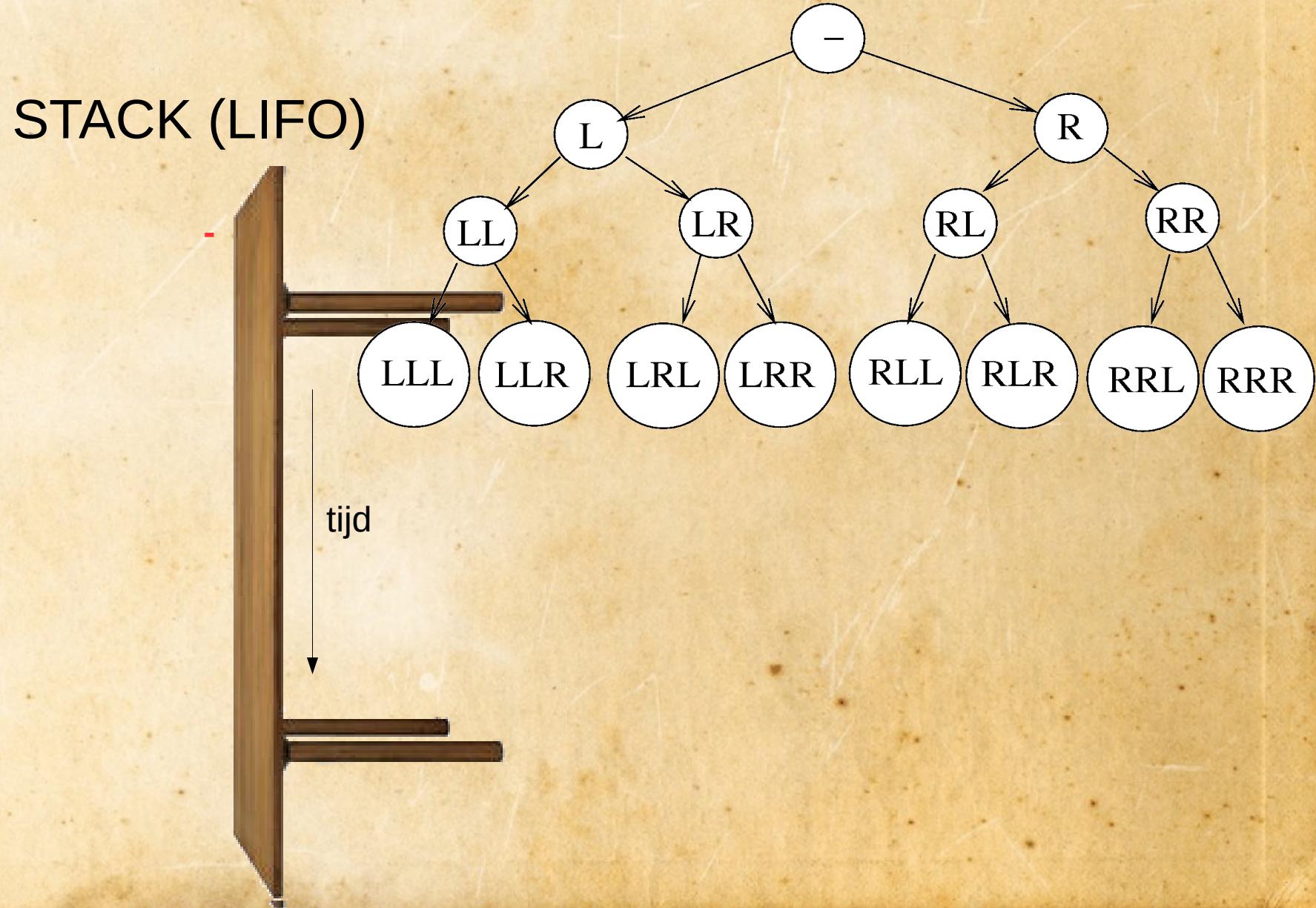
depth = 3
queue = queue.Queue()
queue.put("")
while not queue.empty():
    state = queue.get()                                # no deeper than
    print(state)                                       # add begin state
    if len(state) < depth:                            # get first from
        for i in ['L','R']:                            # stop condition
            child = copy.deepcopy(state)             # for each pc
            child += i                               # deepcopy
            queue.put(child)                         # make new
                                                # add new
```

L  
R  
LL  
LR  
RL  
RR  
LLL  
LLR  
LRL  
LRR  
RLL  
RLR  
RRL  
RRR

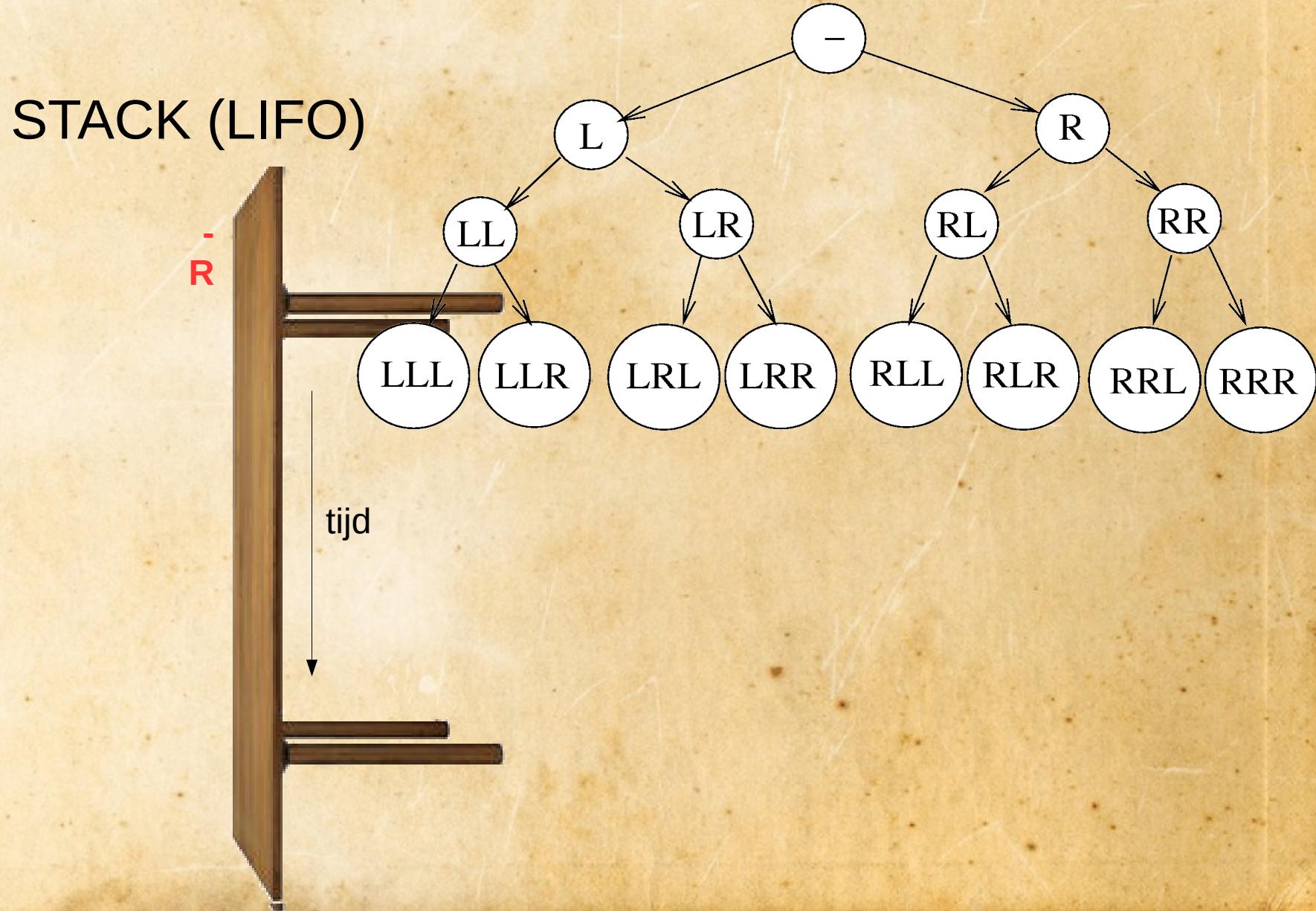
# Depth first



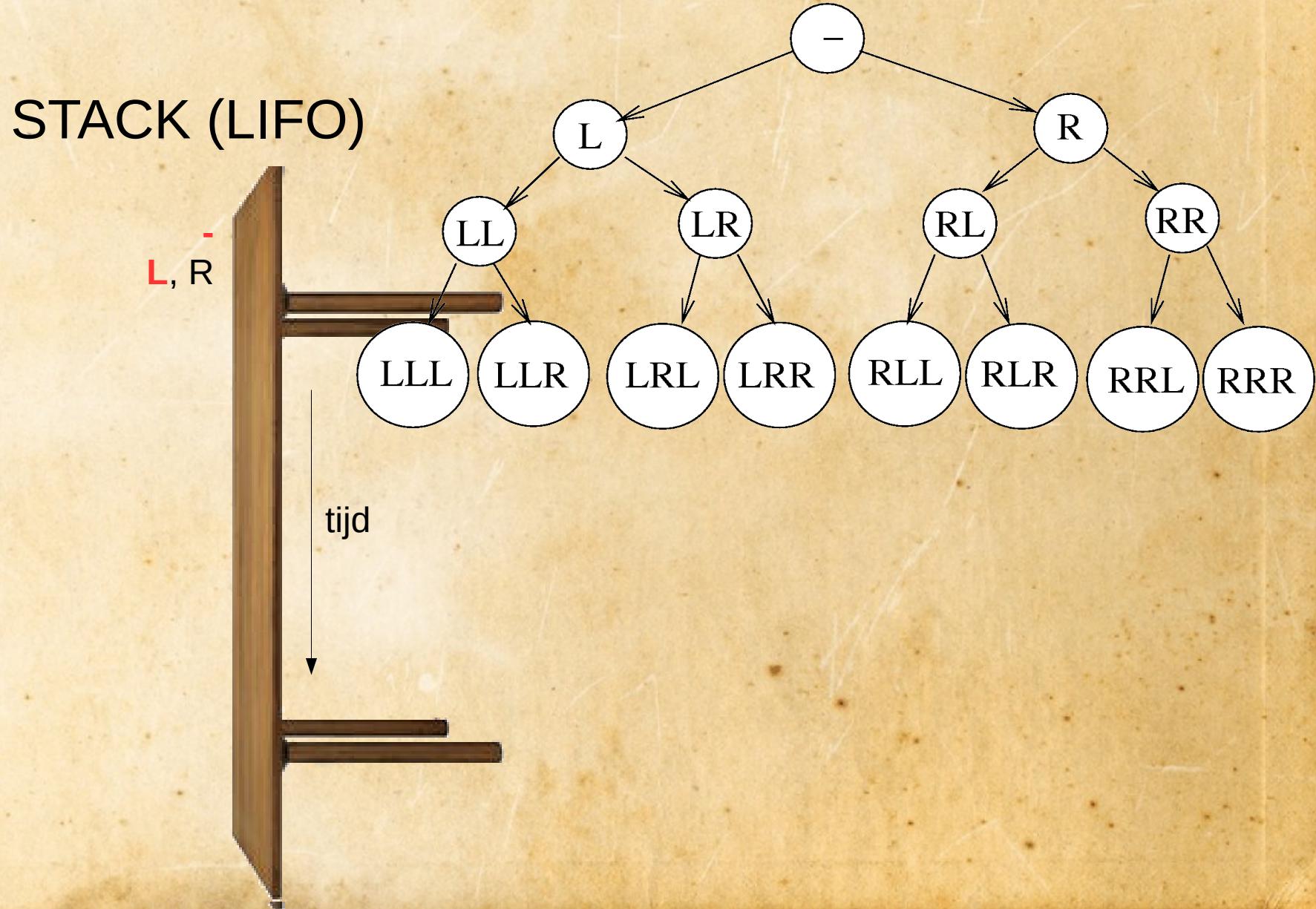
# Depth first



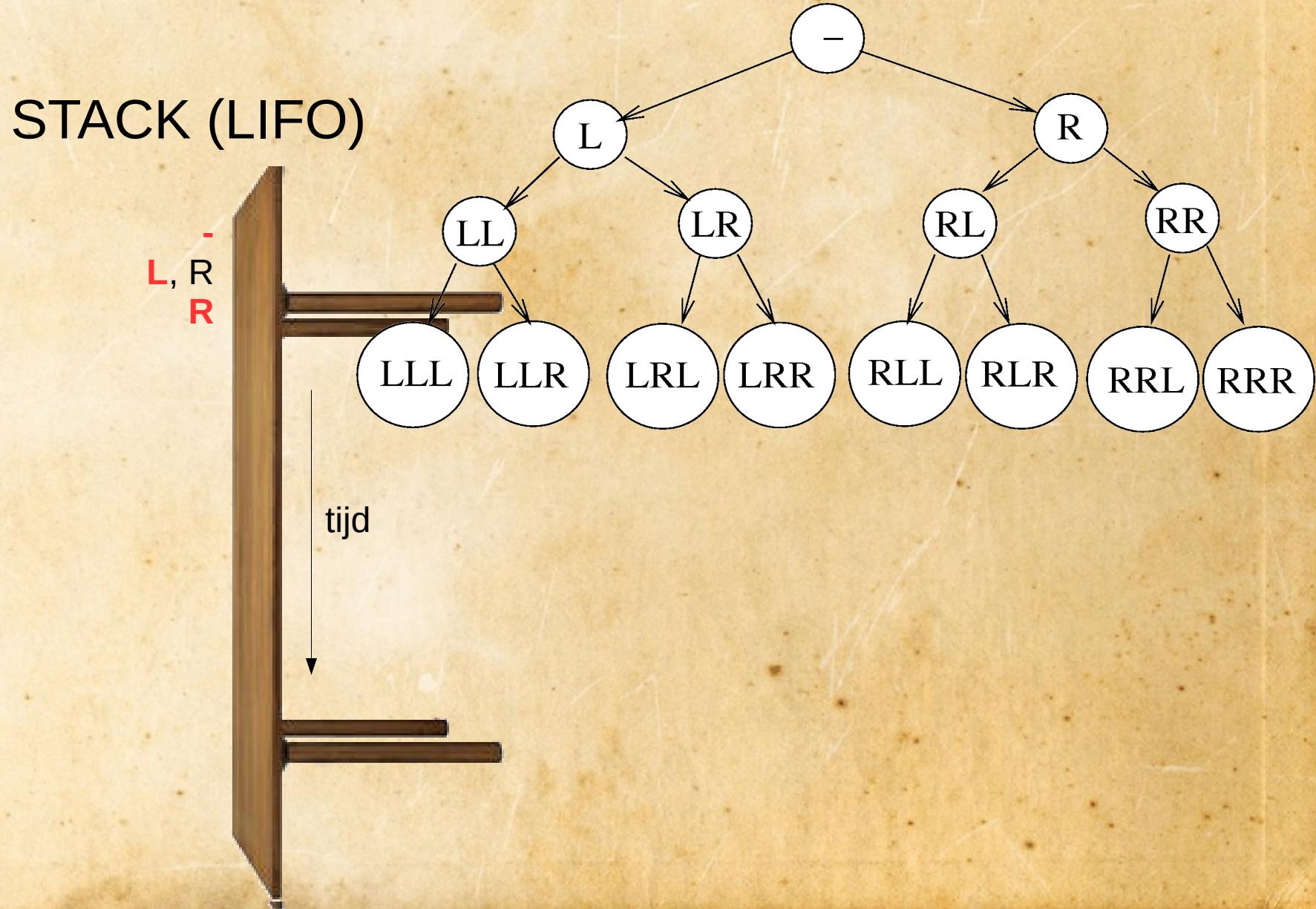
# Depth first



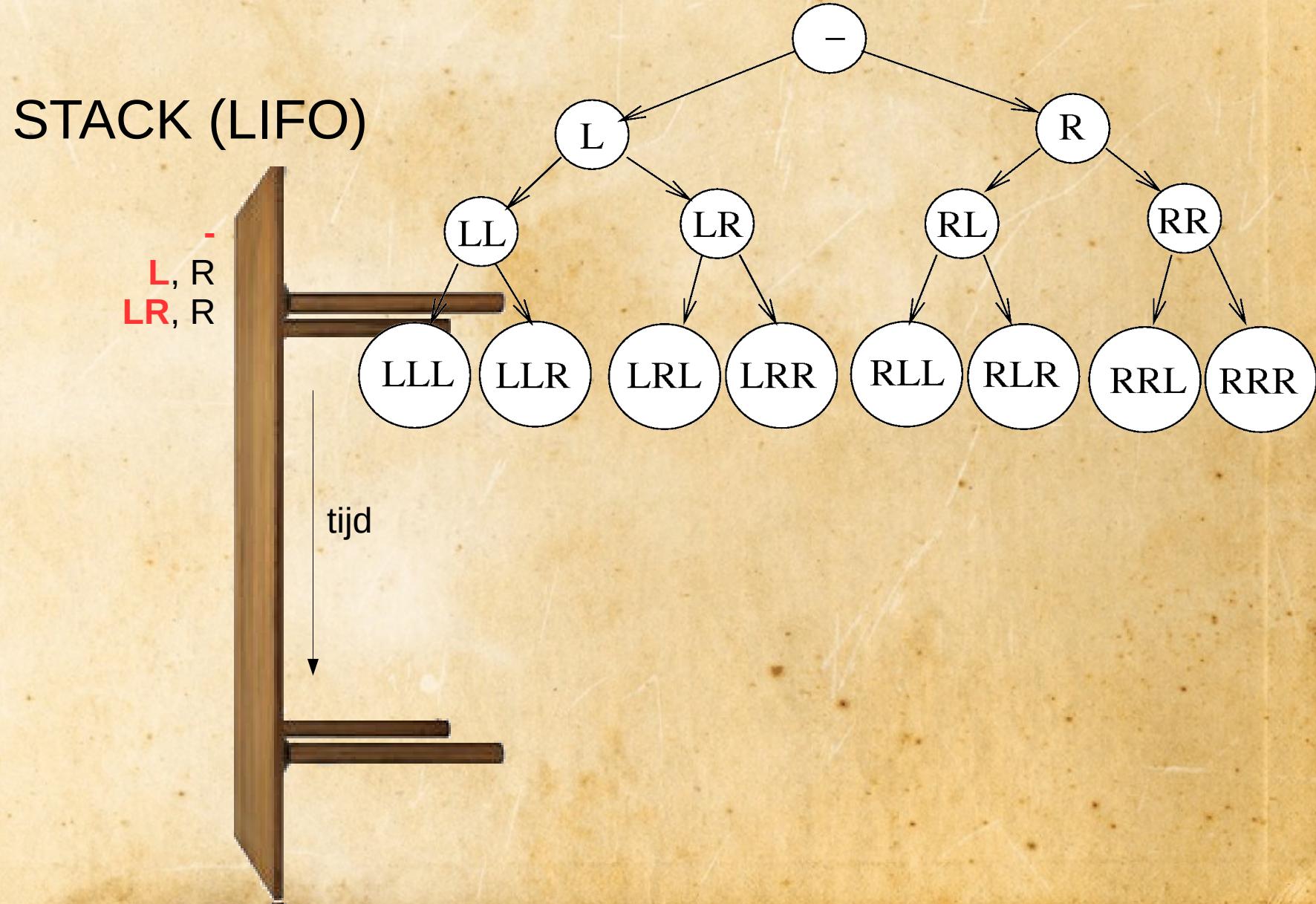
# Depth first



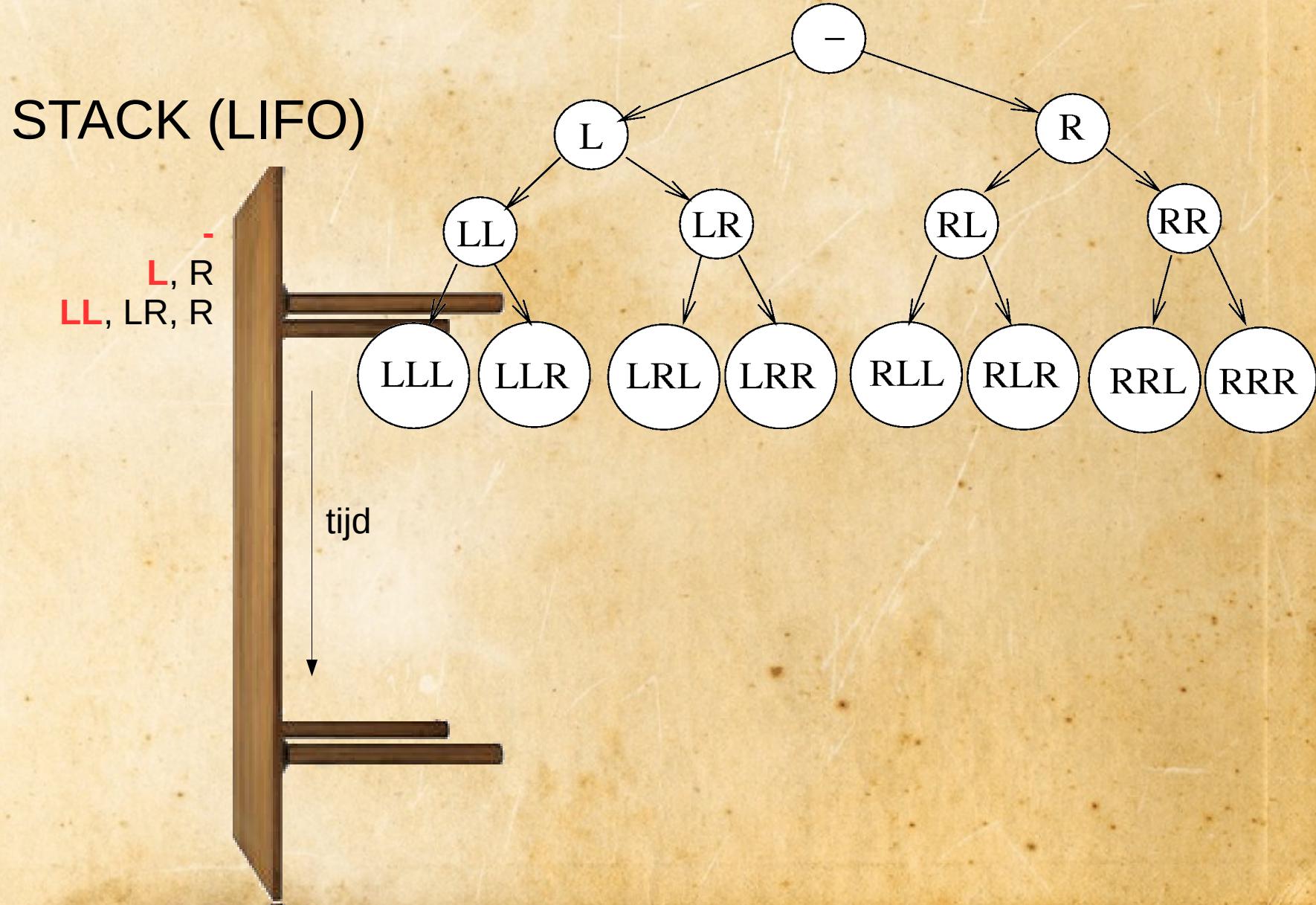
# Depth first



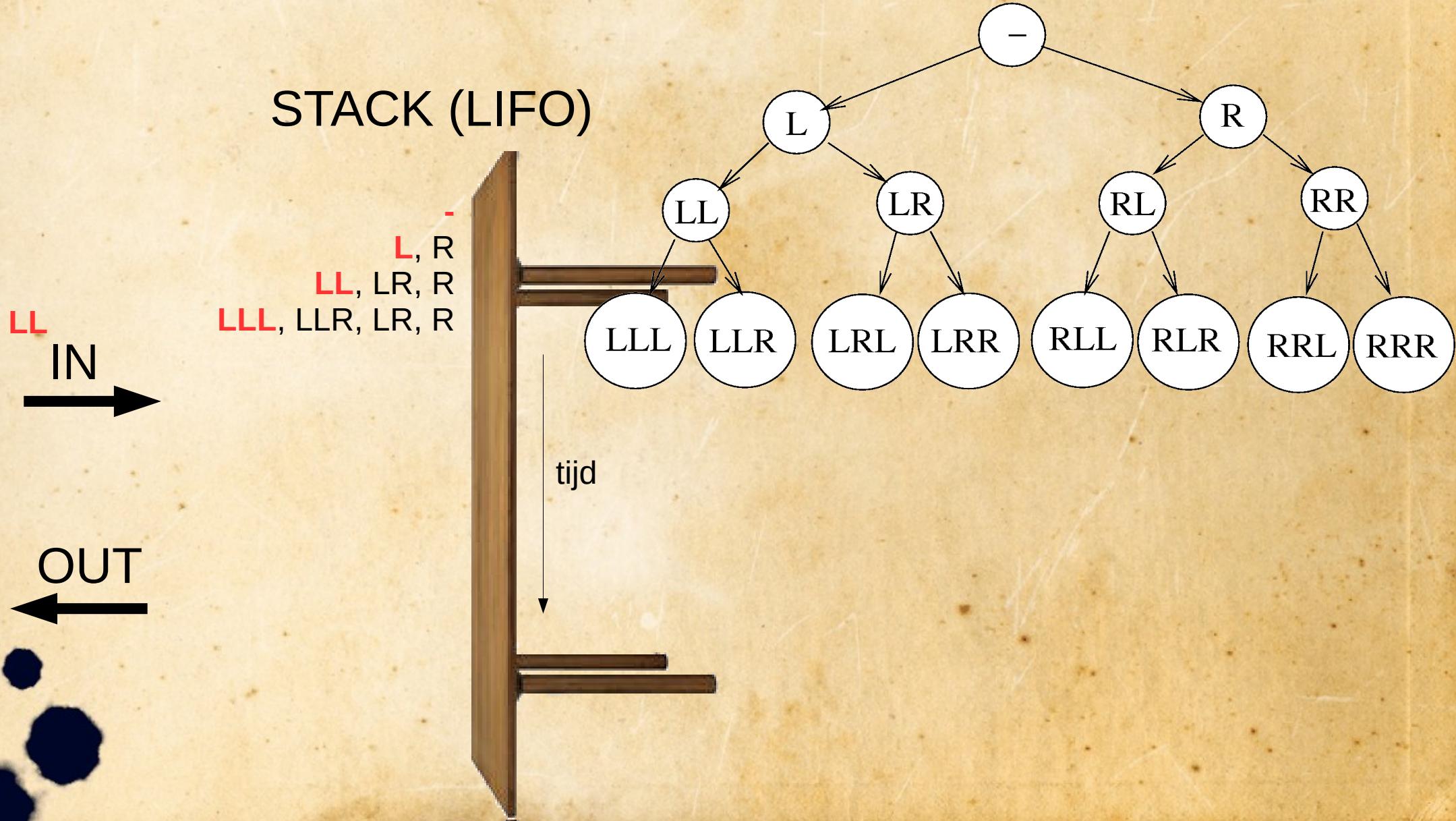
# Depth first



# Depth first



# Depth first



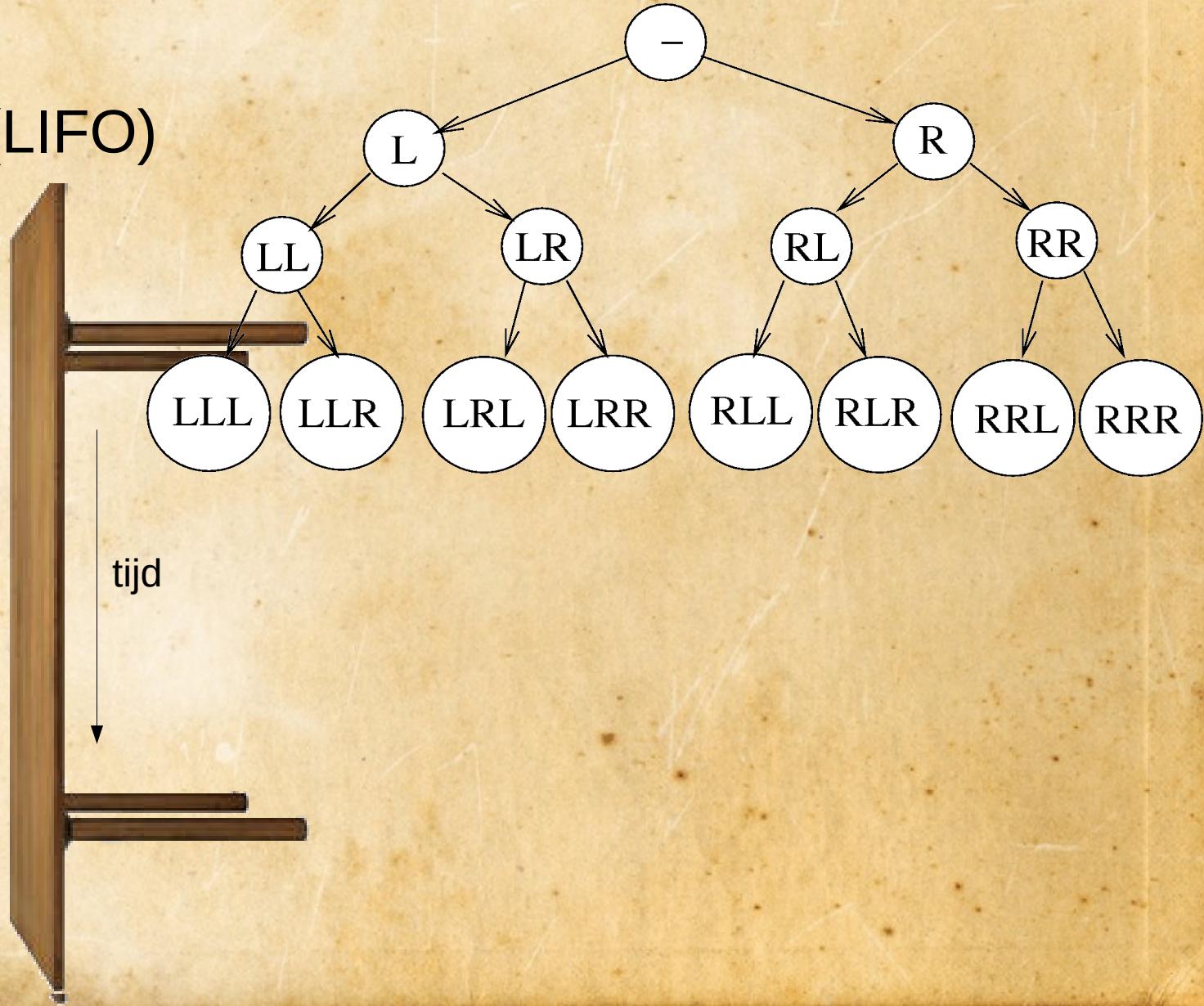
# Depth first

STACK (LIFO)

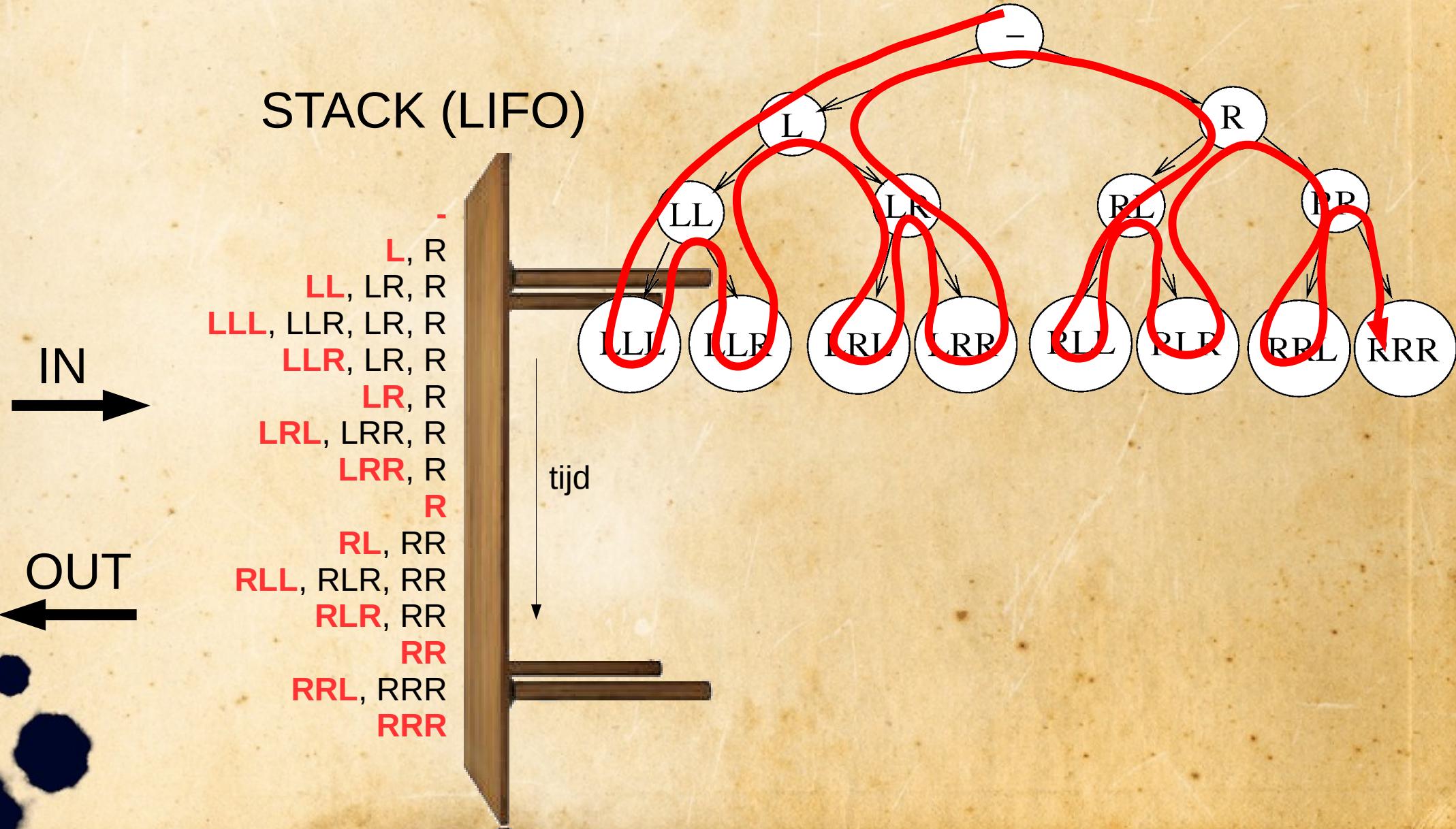
IN →

← OUT

-  
L, R  
LL, LR, R  
LLL, LLR, LR, R  
LLR, LR, R  
LR, R  
LRL, LRR, R  
LRR, R  
R  
RL, RR  
RLL, RLR, RR  
RLR, RR  
RR  
RRL, RRR  
RRR



# Depth first



# Depth first in Python

```
import copy

depth = 3
stack = []
while len(stack)>0:
    state=stack.pop()
    print(state)
    if len(state) < depth:
        for i in ['R','L']:
            child = copy.deepcopy(state)
            child += i
            stack.append(child)
# no deeper than 'depth'
# add begin state to stack
# get top from stack
# stop condition
# for each possible action:
#     deepcopy the state
#     make new child
#     put on stack
```

# Depth first in Python

```
import copy

depth = 3
stack = [""] # no deeper than
# add begin stat
while len(stack)>0: # get top from s
    state=stack.pop() # stop condition
    print(state) # for each p
    if len(state) < depth: # deepcop
        for i in ['R','L']: # make ne
            child = copy.deepcopy(state) # put on
            child += i
            stack.append(child)
```

L  
LL  
LLL  
LLR  
LR  
LRL  
LRR  
RR  
RL  
RLL  
RLR  
RR  
RRL  
RRR

# Depth first, kan ook recursief

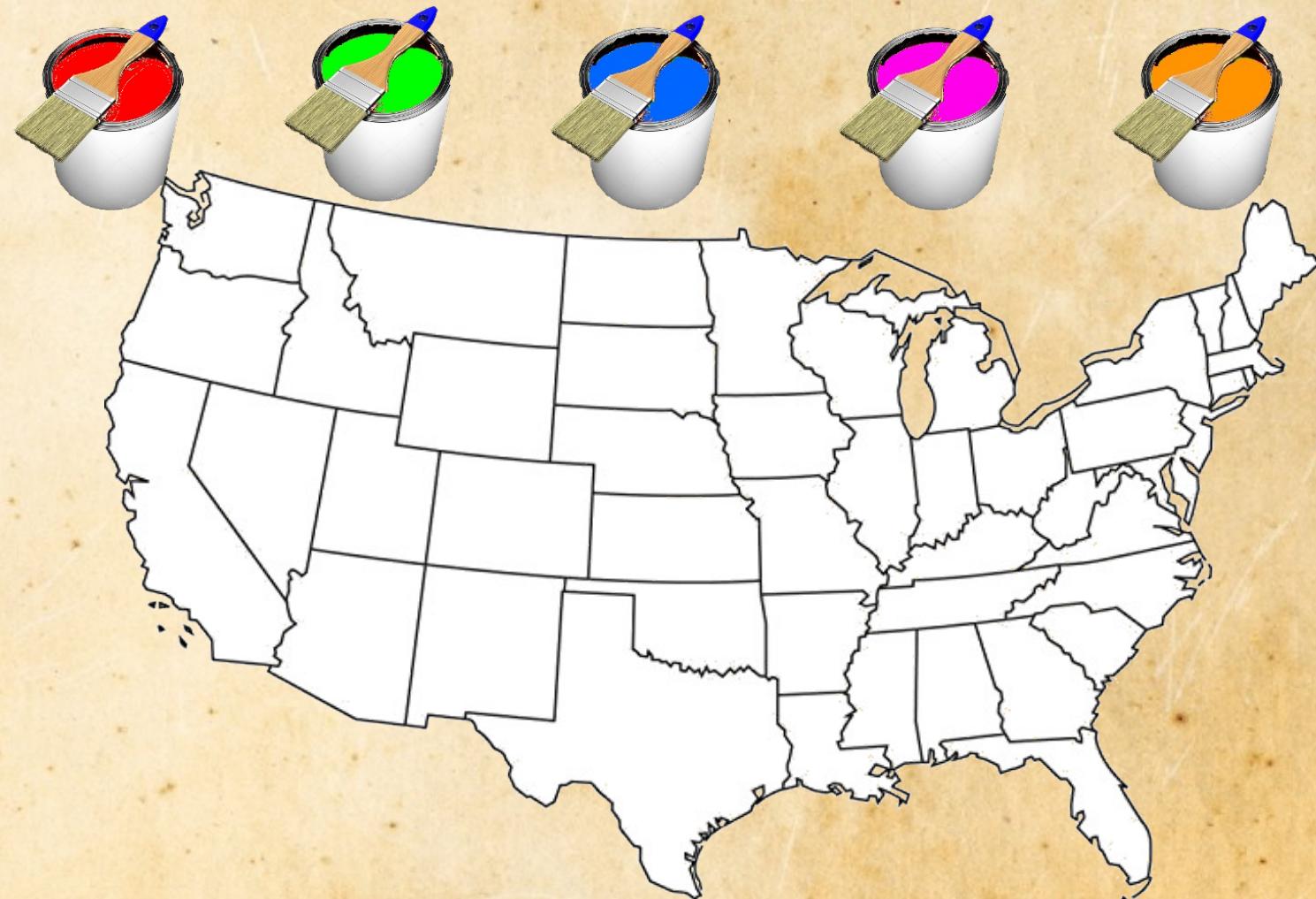
```
def depth_first(state,depth):
    print(state)
    if len(state) < depth:           # no deeper than 'depth'
        for i in ['L','R']:           # for each possible action
            depth_first(state+i,depth) # recursive call

depth_first("",3) # start the recursion
```

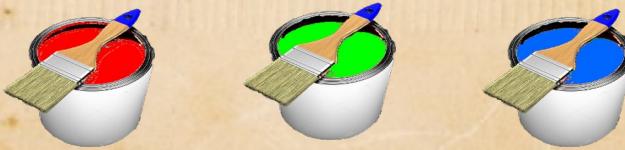
# Breadth of Depth first?



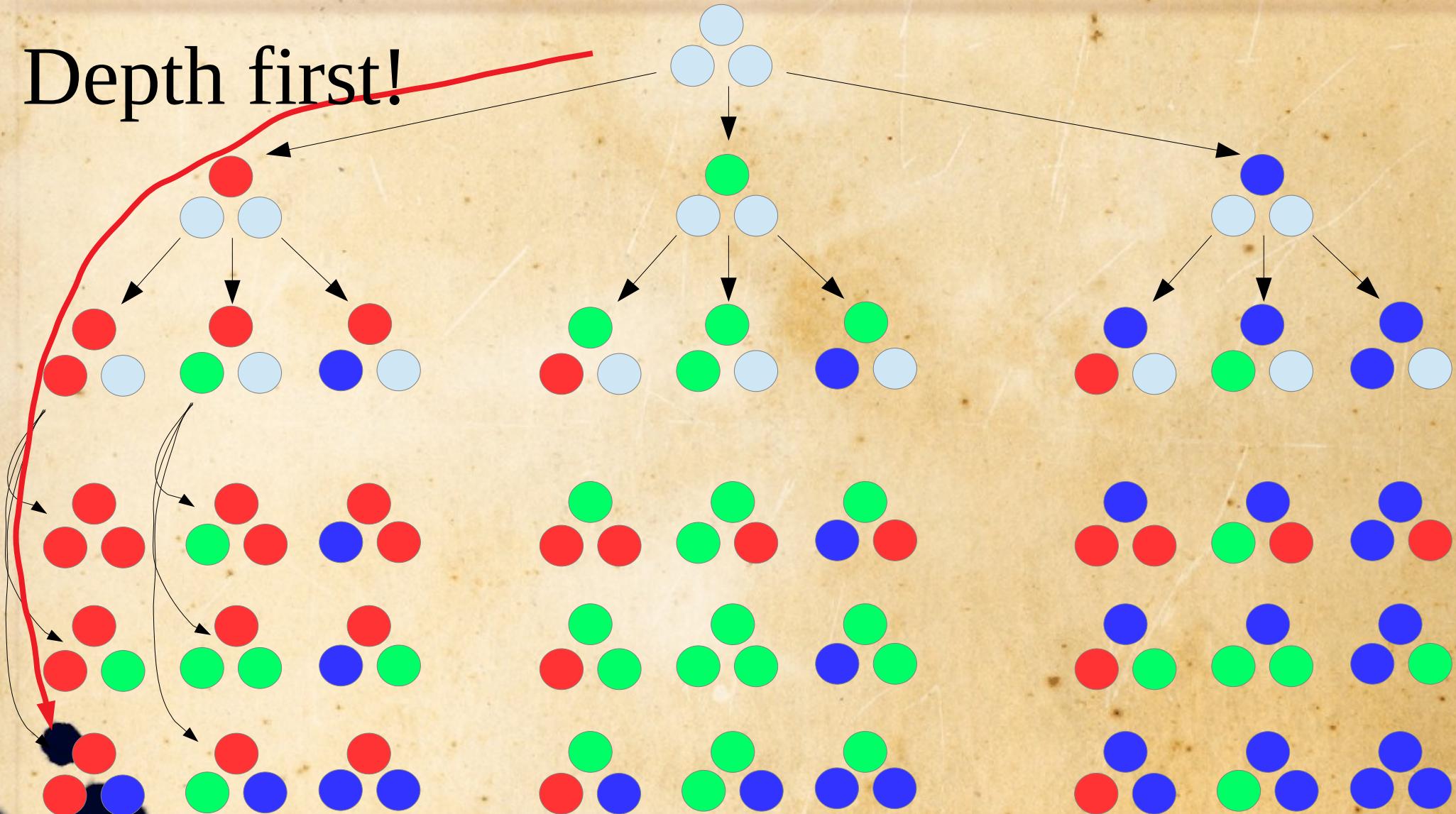
We willen snel oplossingen vinden, maakt niet uit wat het kost



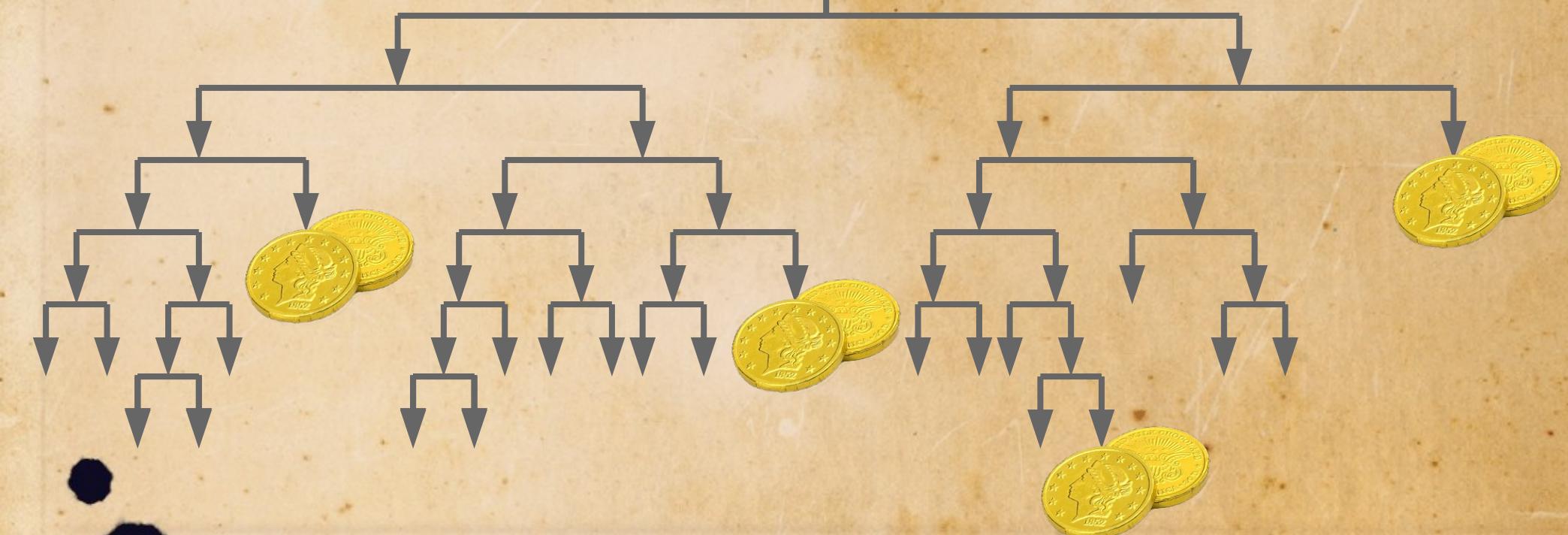
# Kaart kleuren



Depth first!

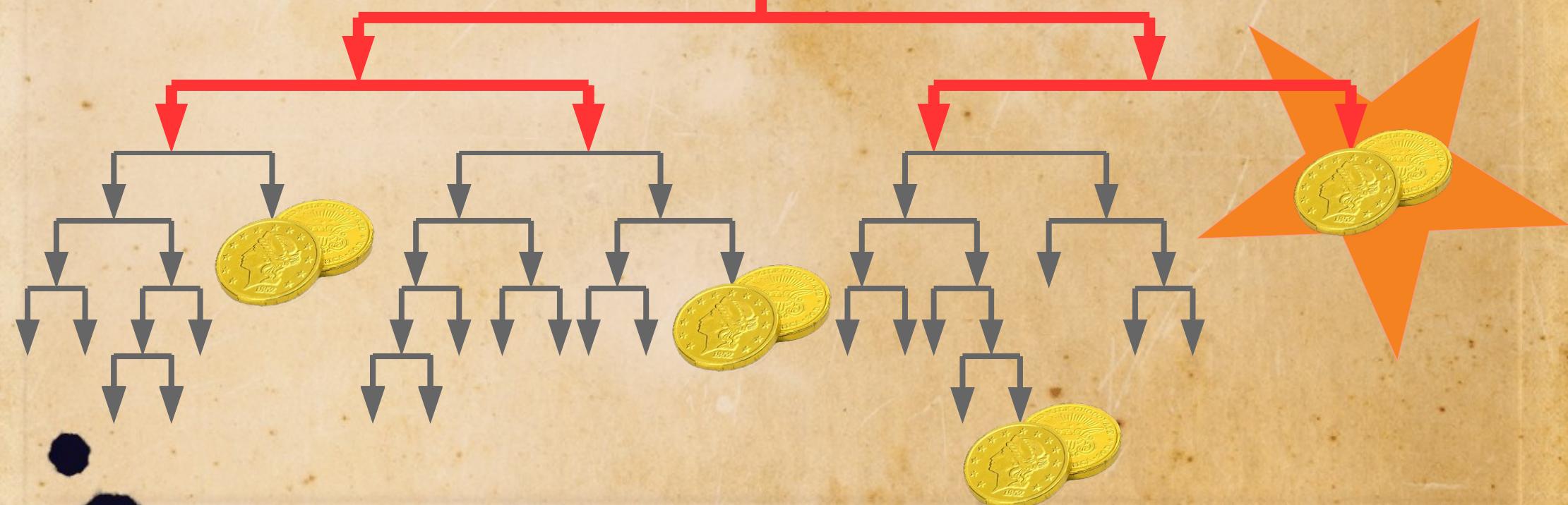


# Breadth of Depth first?



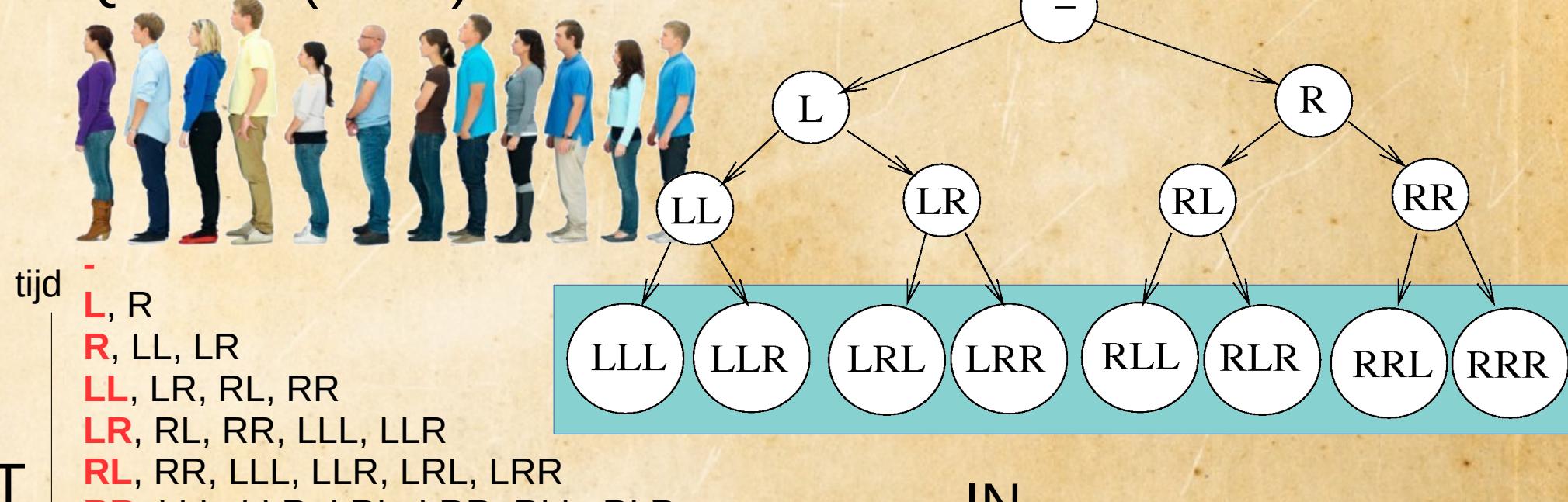
# Breadth of Depth first?

Breadth first!



# Breadth first, geheugen?

## QUEUE (FIFO)



OUT



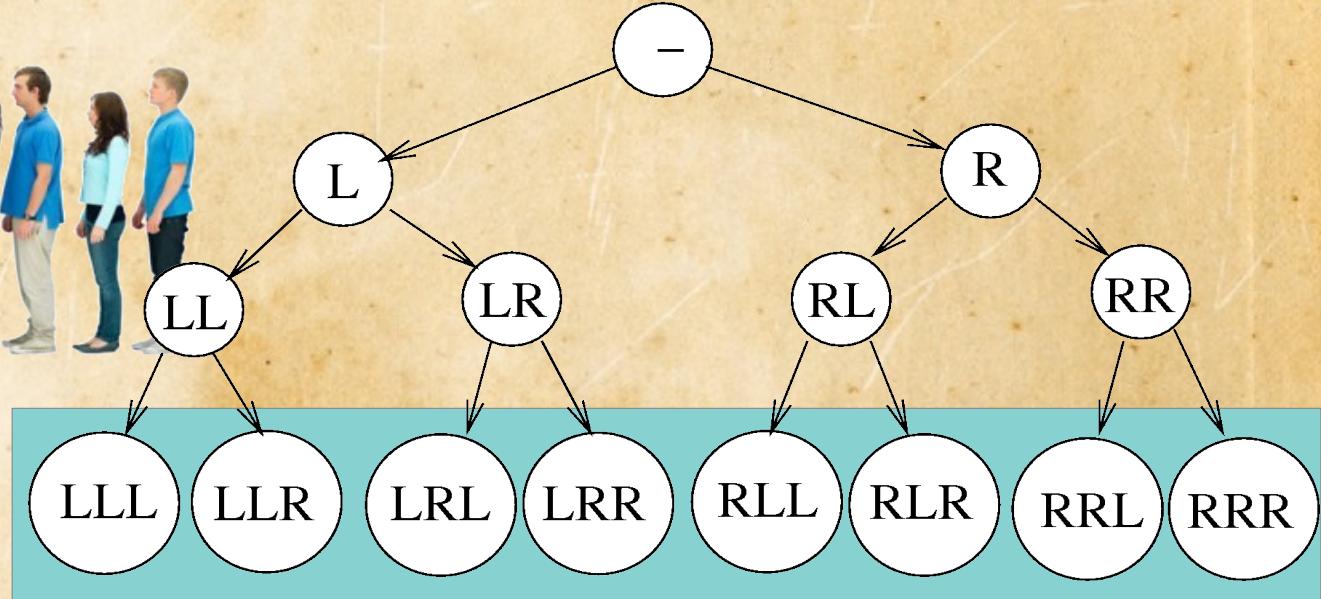
IN



Bij zoeken op diepte 3 max 8 states in de Queue  
Hoeveel states op diepte 4?  
Hoeveel states op diepte 5?  
Hoeveel states op diepte N?

# Breadth first, geheugen?

## QUEUE (FIFO)



tijd

L, R

R, LL, LR

LL, LR, RL, RR

LR, RL, RR, LLL, LLR

RL, RR, LLL, LLR, LRL, LRR

RR, LLL, LLR, LRL, LRR, RLL, RLR

LLL, LLR, LRL, LRR, RLL, RLR, RRL, RRR

LLR, LRL, LRR, RLL, RLR, RRL, RRR

LRL, LRR, RLL, RLR, RRL, RRR

LRR, RLL, RLR, RRL, RRR

RLL, RLR, RRL, RRR

RLR, RRL, RRR

RRL, RRR

RRR

OUT



IN



Bij zoeken op diepte 3 max 8 states in de Queue

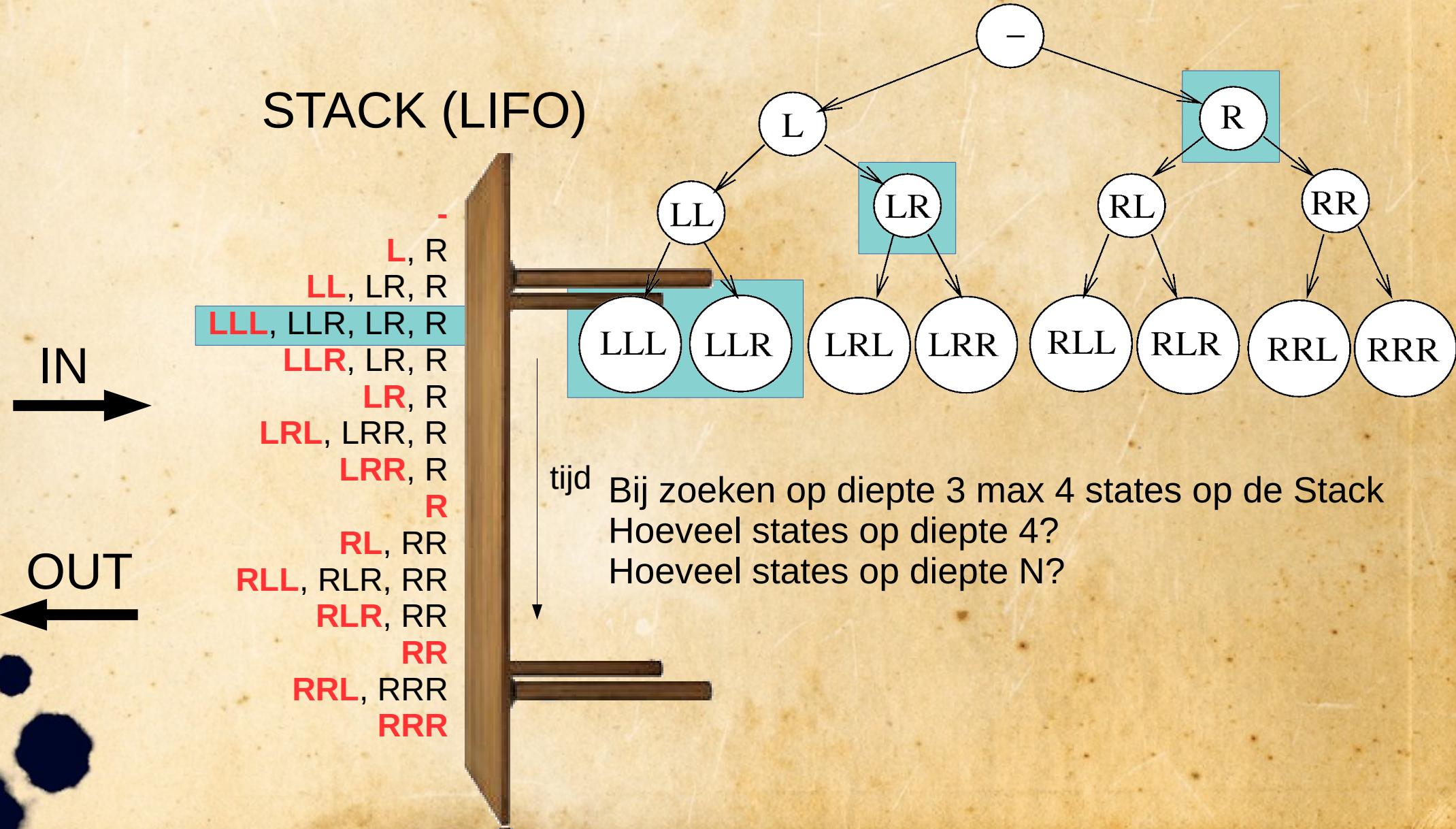
Hoeveel states op diepte 4?

Hoeveel states op diepte 5?

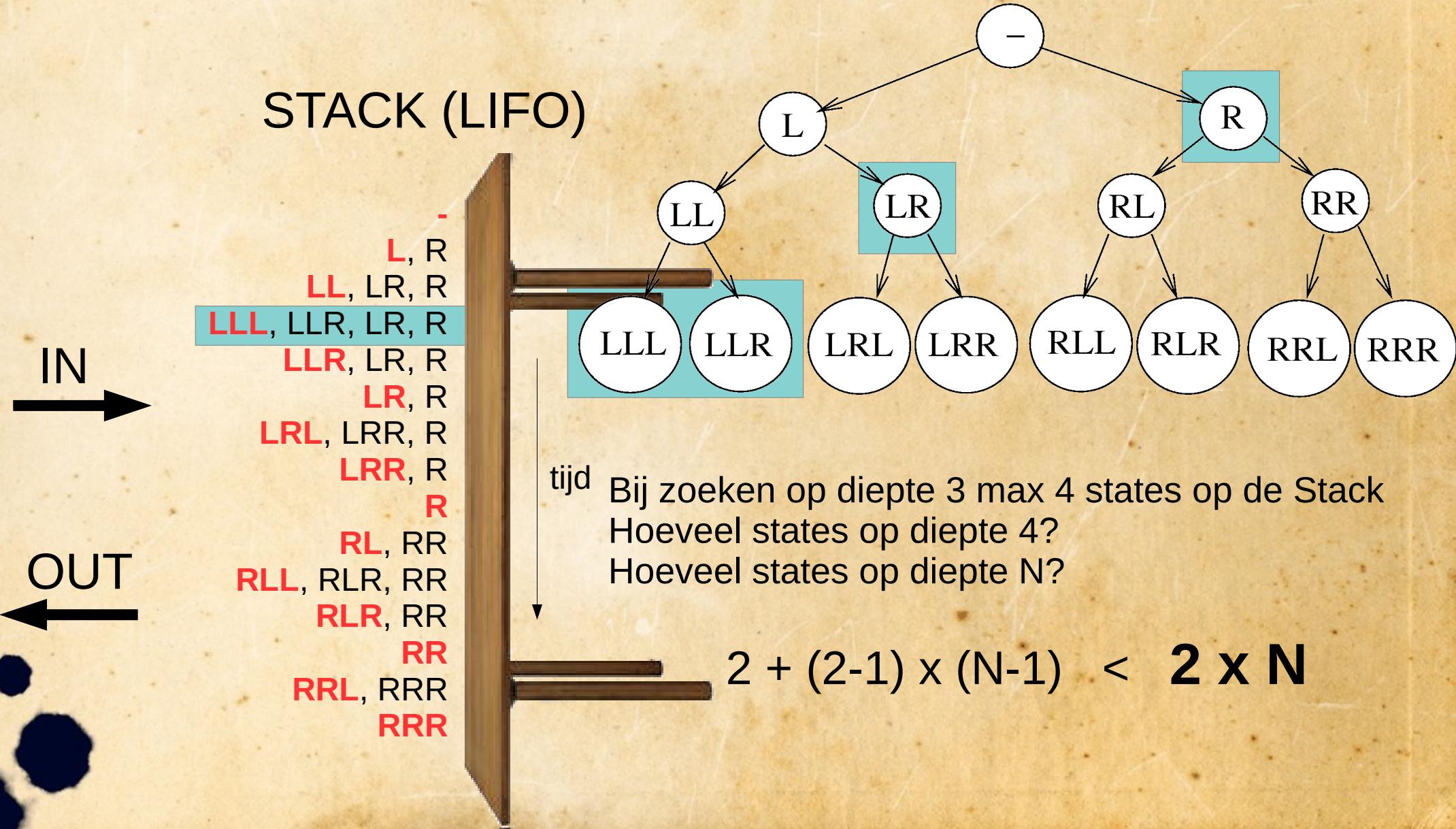
Hoeveel states op diepte N?

$2^N$

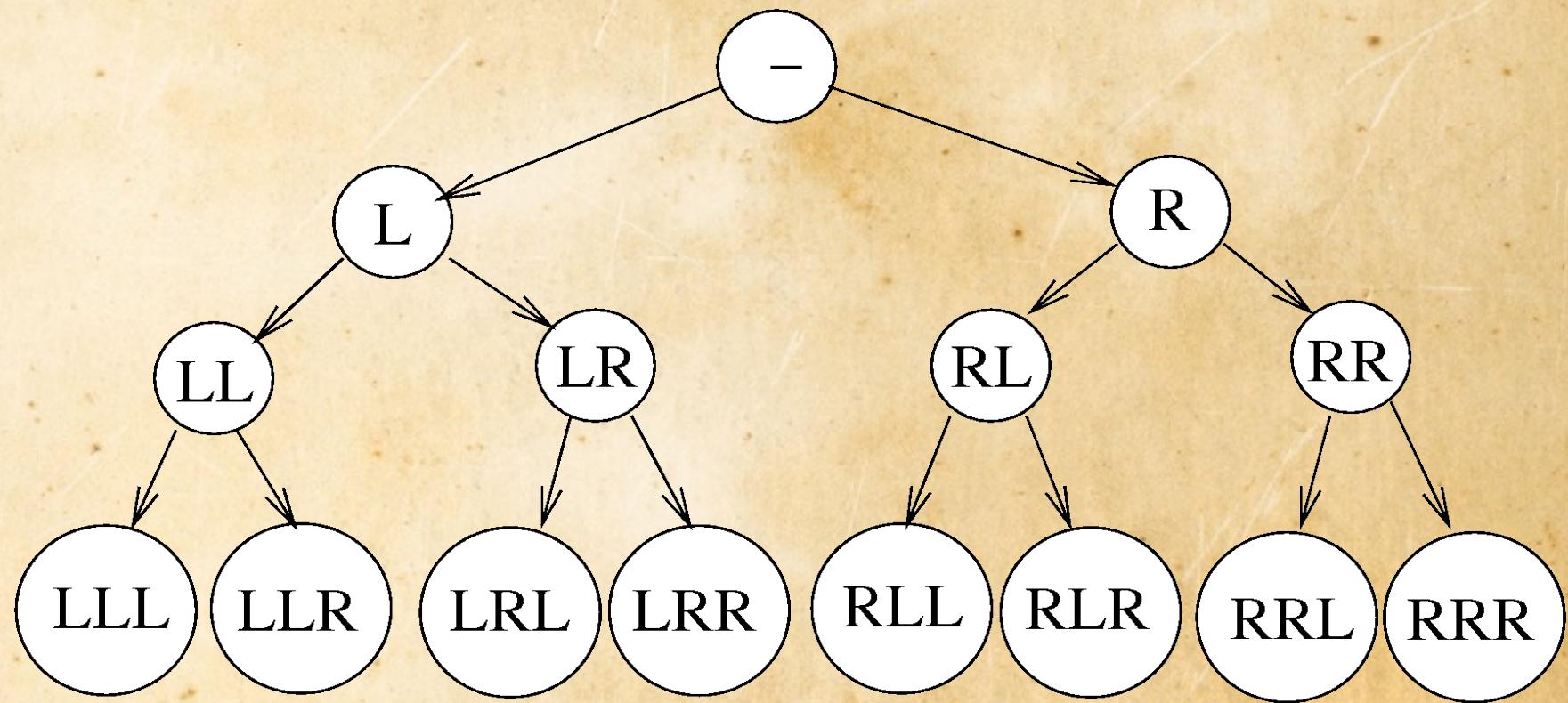
# Depth first, geheugen?



# Depth first, geheugen?

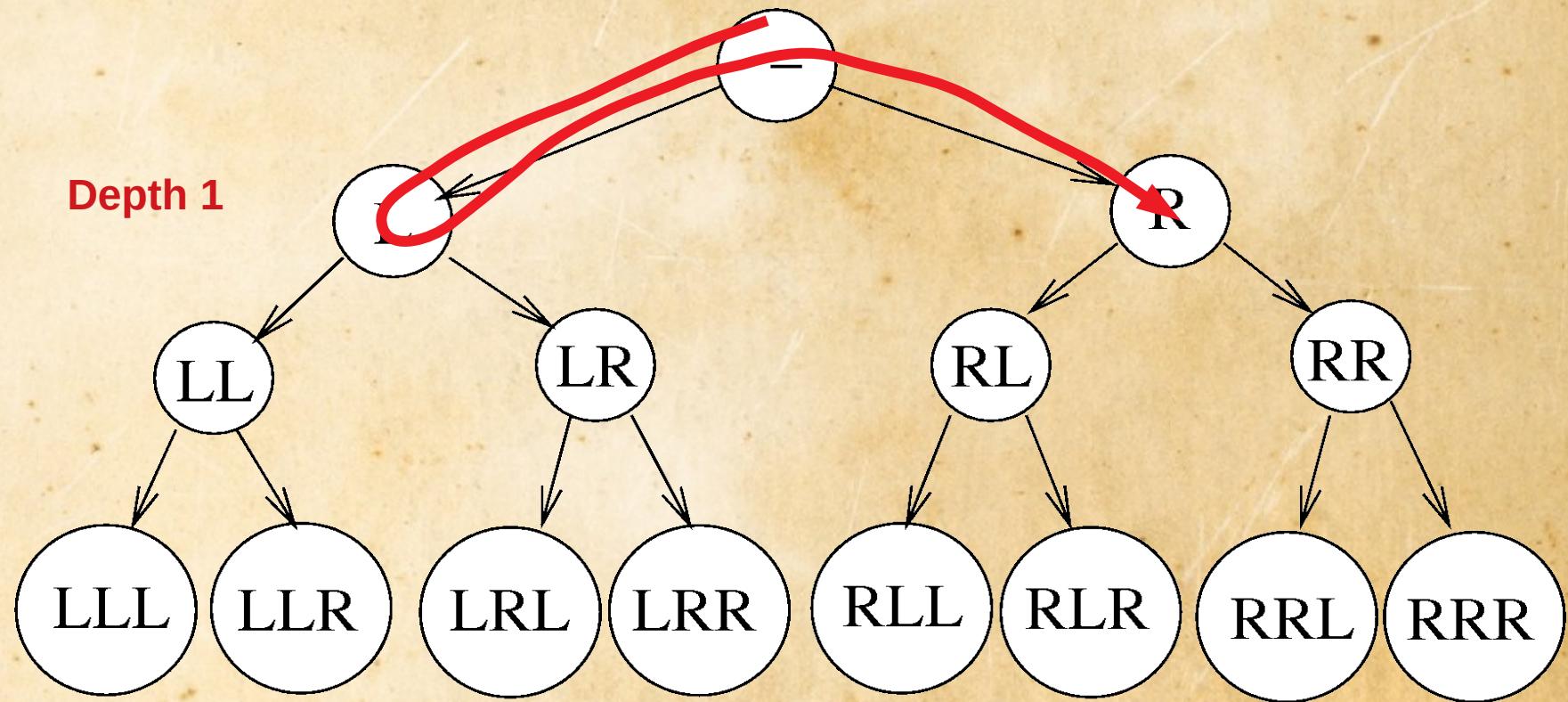


# Iterative deepening depth first



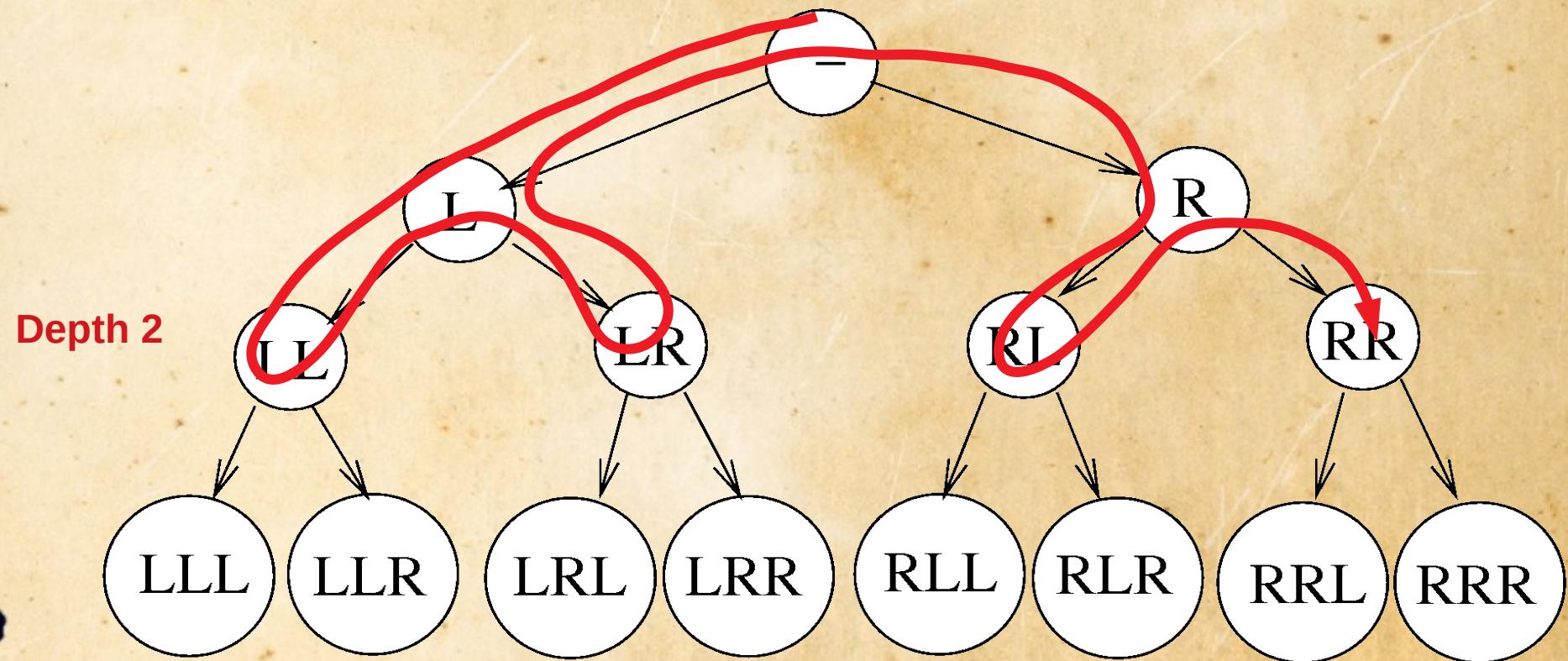
# Iterative deepening depth first

- Eerst depth first zoeken tot diepte 1



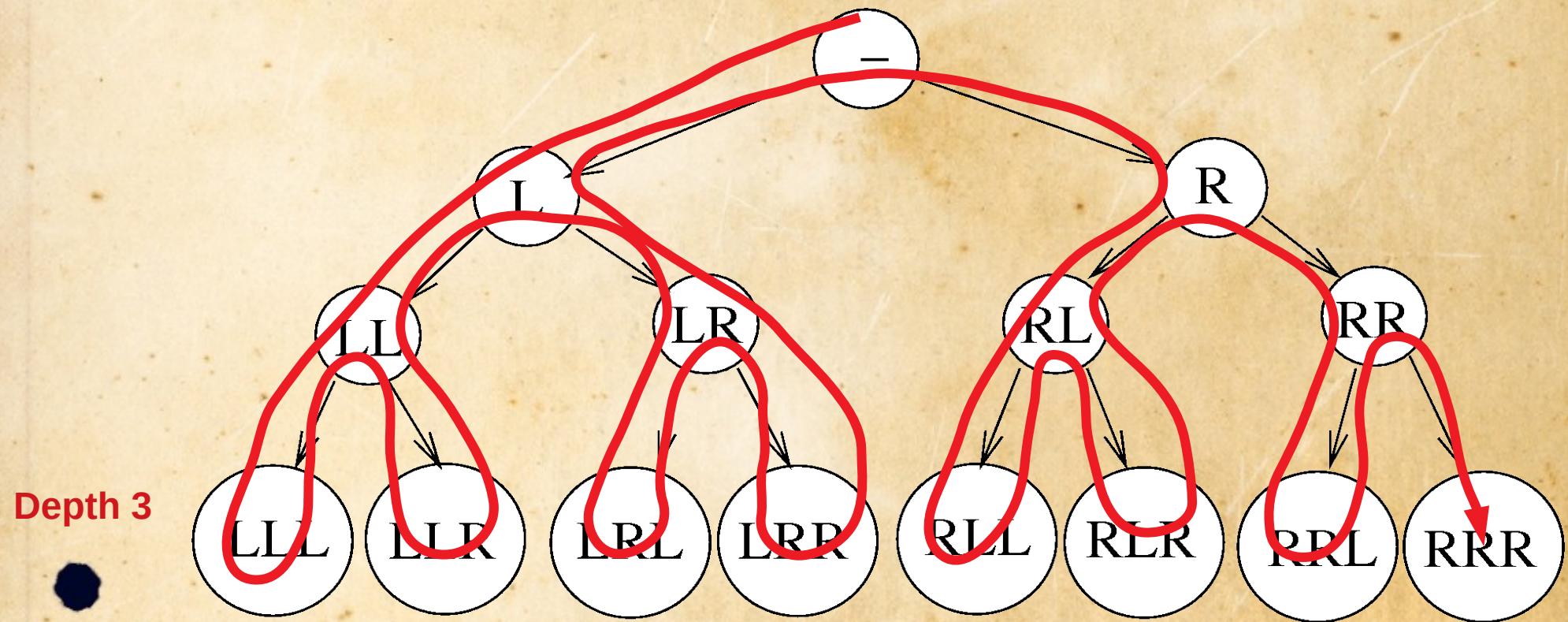
# Iterative deepening depth first

- Dan opnieuw beginnen met zoeken tot diepte 2



# Iterative deepening depth first

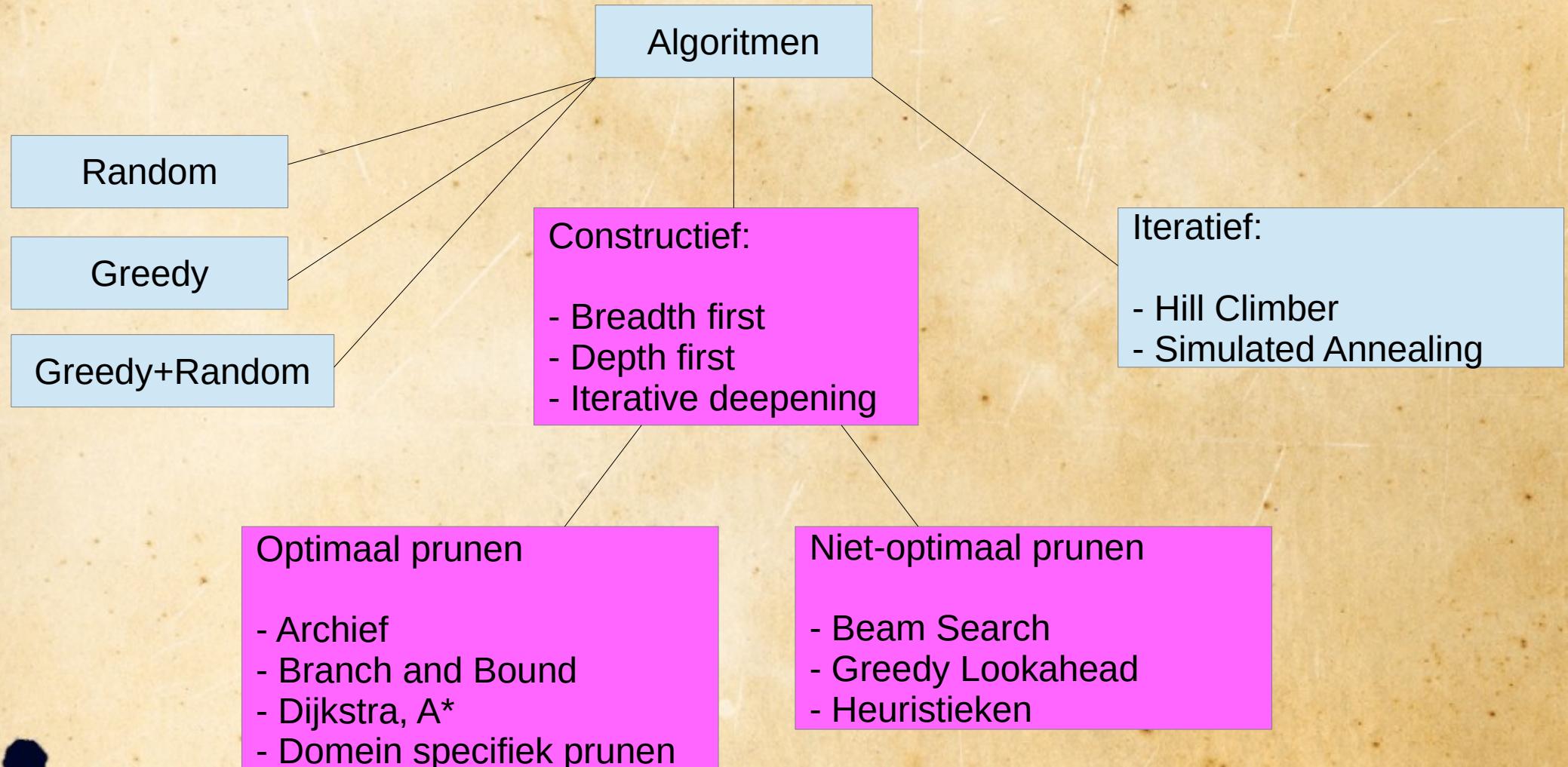
- Dan opnieuw beginnen met zoeken tot diepte 3



# Breadth/Depth first demo

- Path finding in grid world met depth=4
  - `python breadthFirstViz.py grid.gr 55 0 4`
  - `python depthFirstViz.py grid.gr 55 0 4`

# Algoritmen



# Optimaal prunen

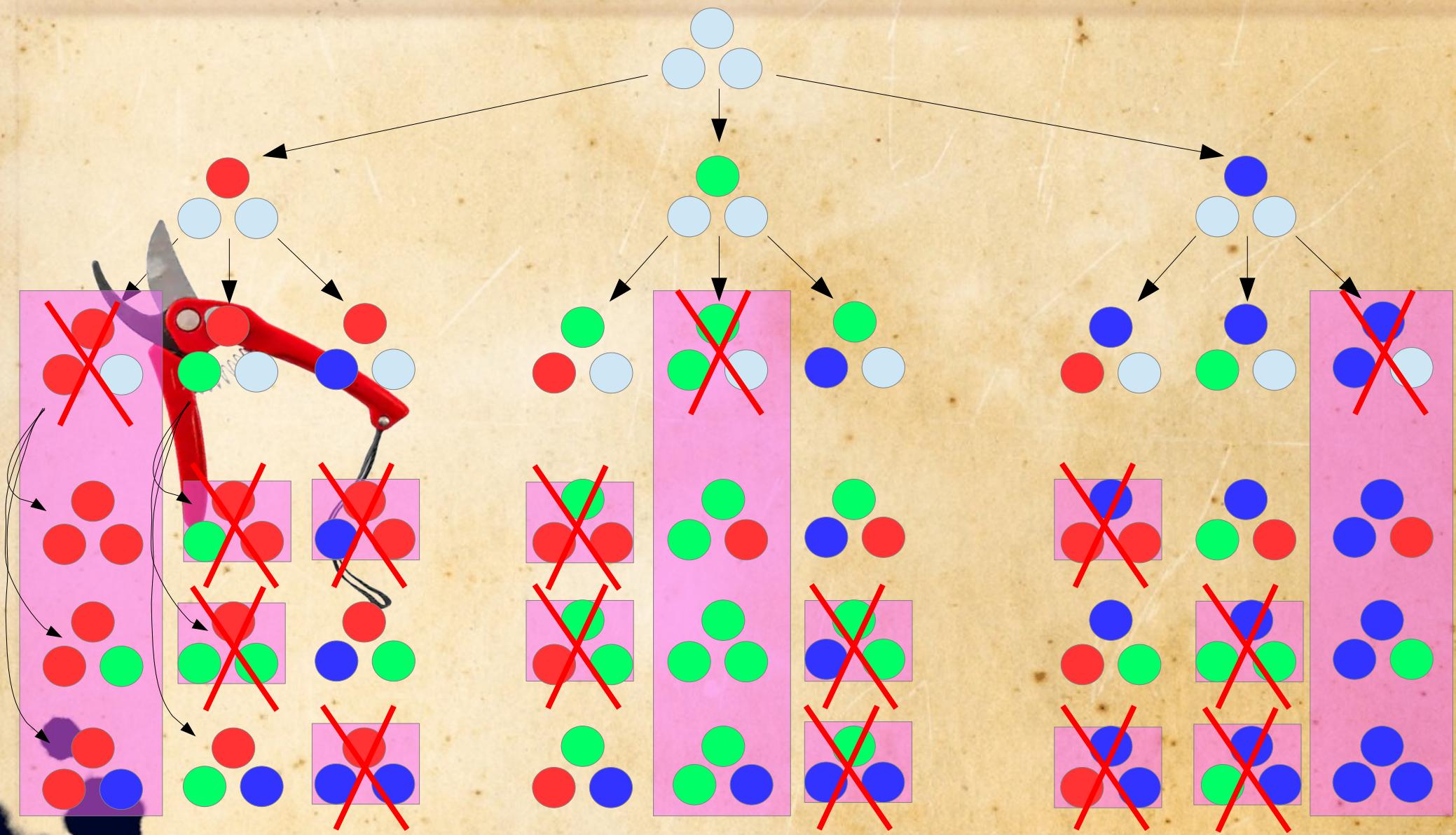


# Early Constraint Checking



- Breadth of Depth first zoeken:
  - Constraint zo **vroeg mogelijk** controleren

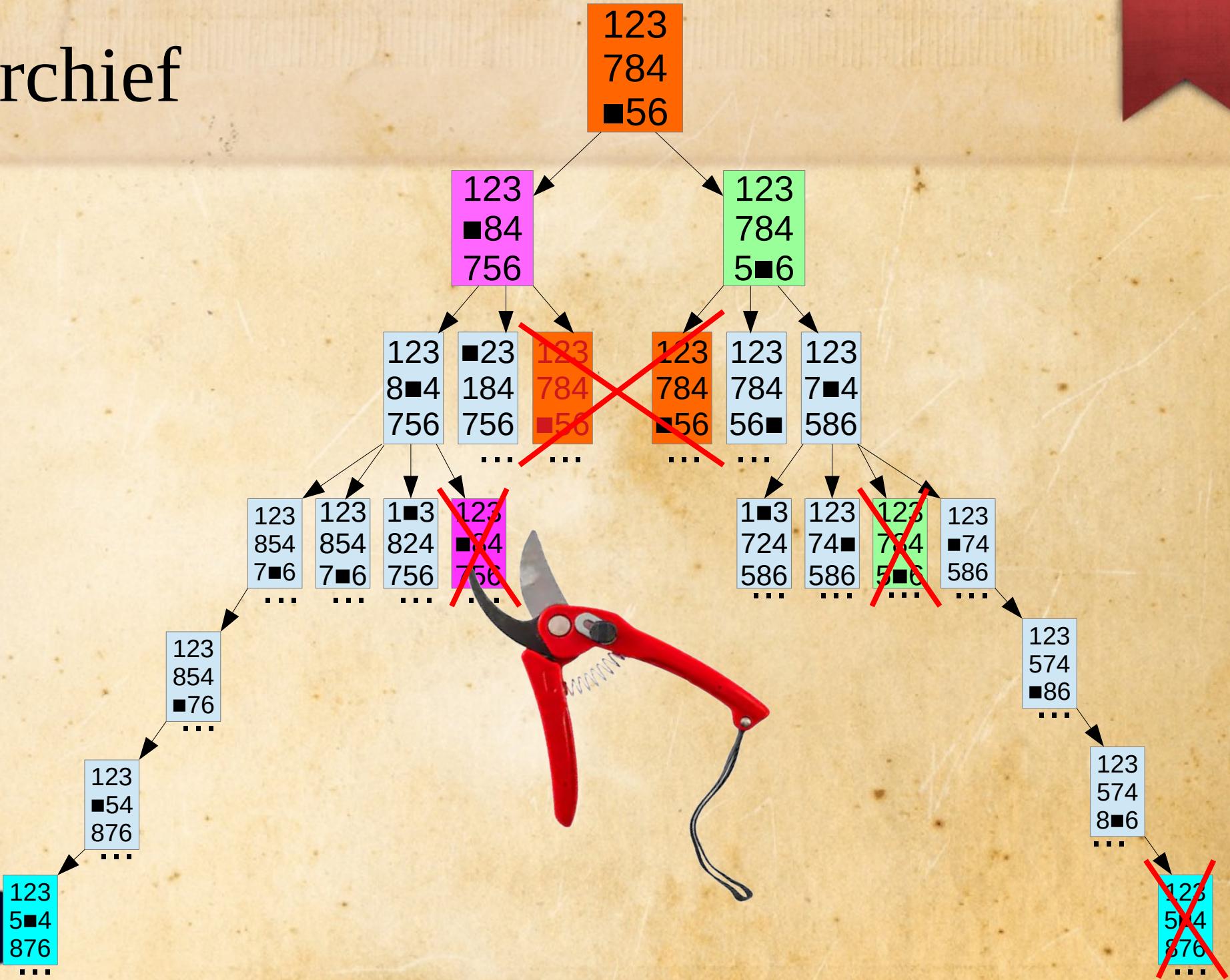
# Early Constraint Checking



# Archief

- Breadth or Depth first zoeken
  - **Onthouden van states** die al bezocht zijn
    - bv door deze op te slaan in een Python ”set” of ”dictionary” zodat je **snel** states kunt terugvinden

# Archief

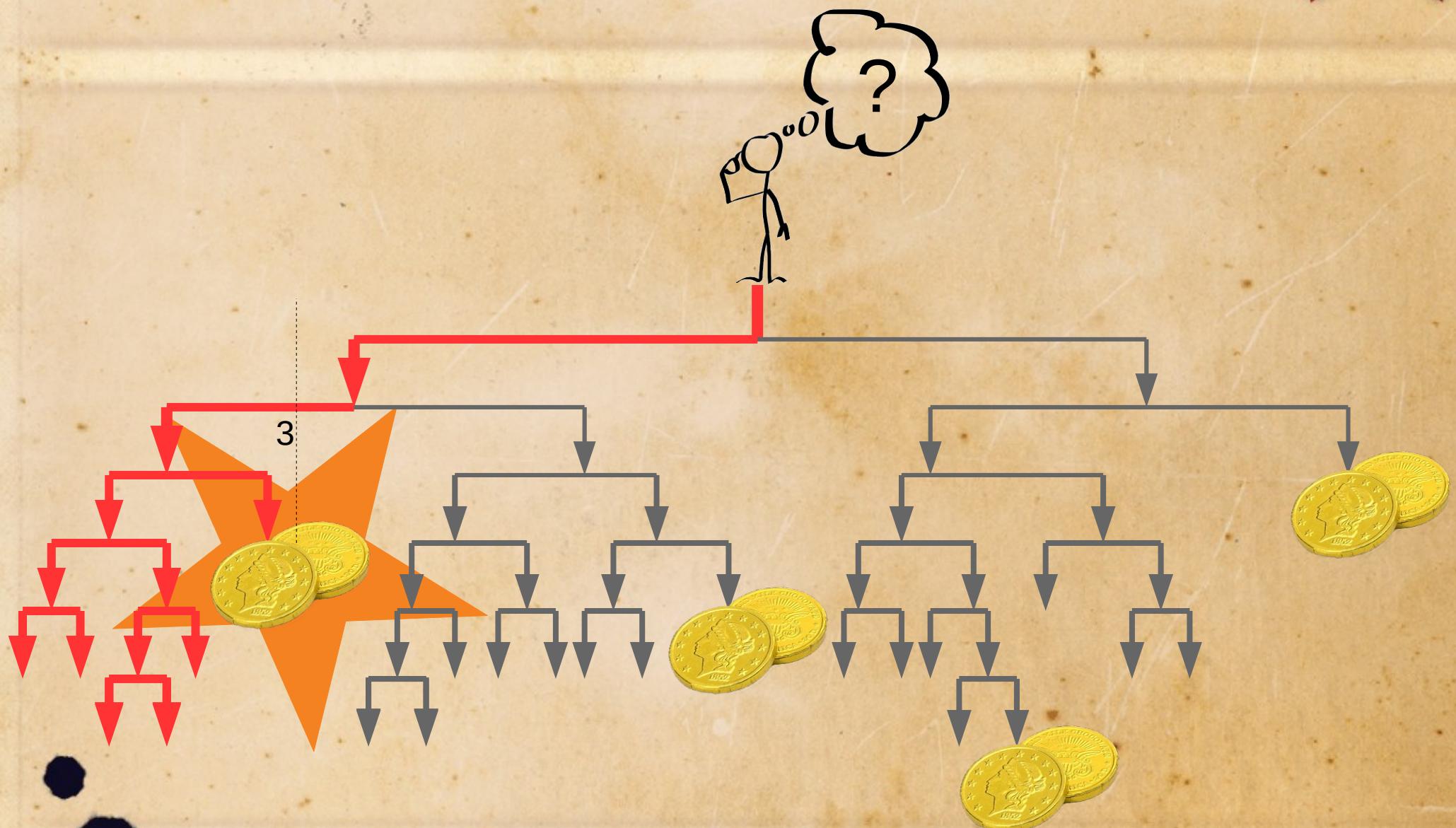


# Branch and Bound

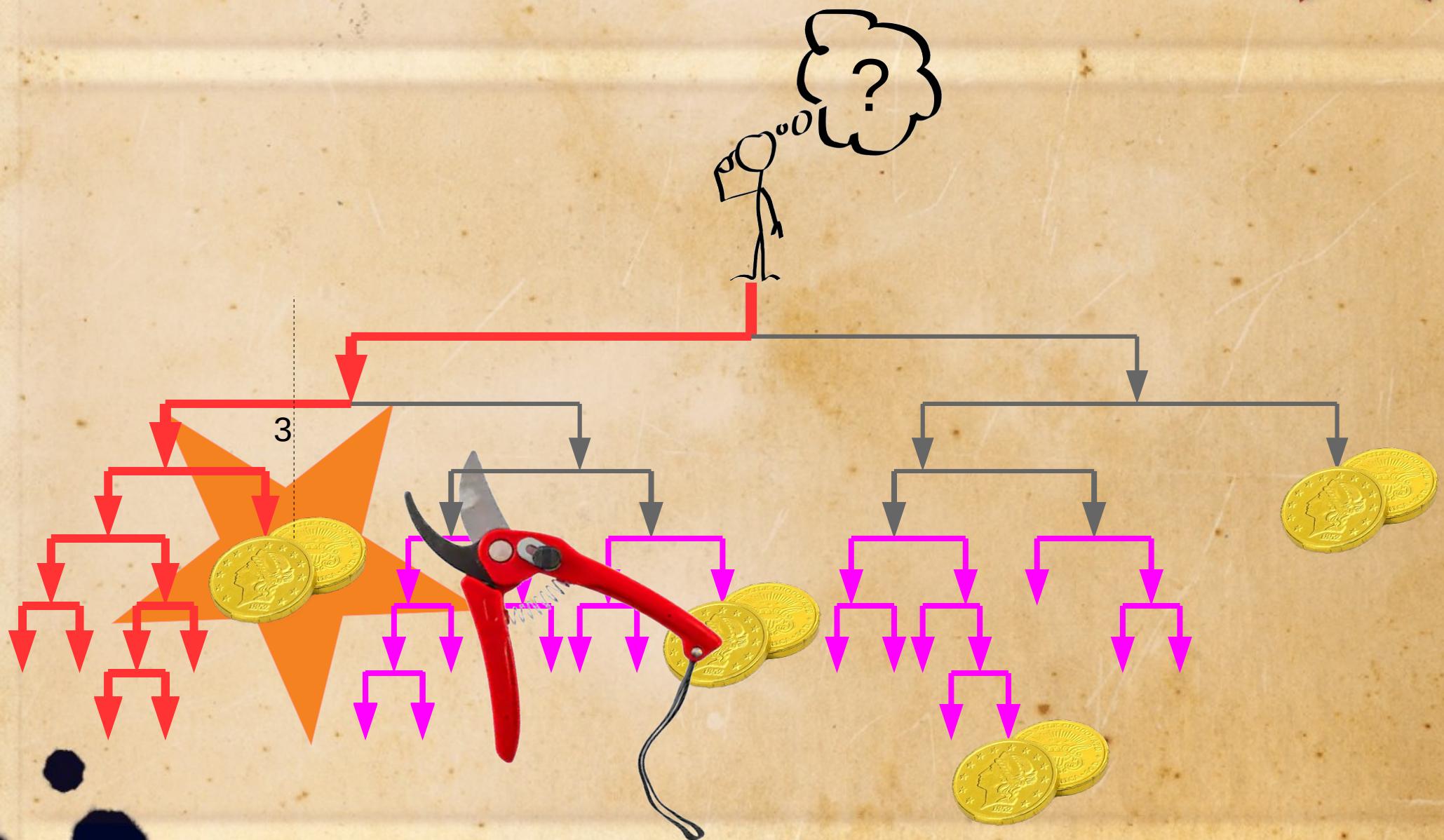


- Depth first zoeken
  - Maar niet **dieper** dan **kortst** gevonden pad

# Branch and Bound

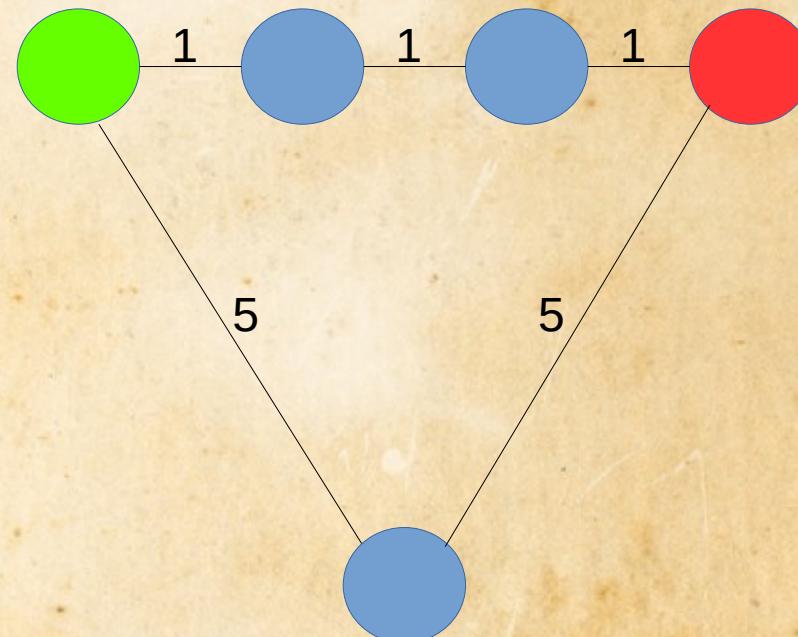


# Branch and Bound



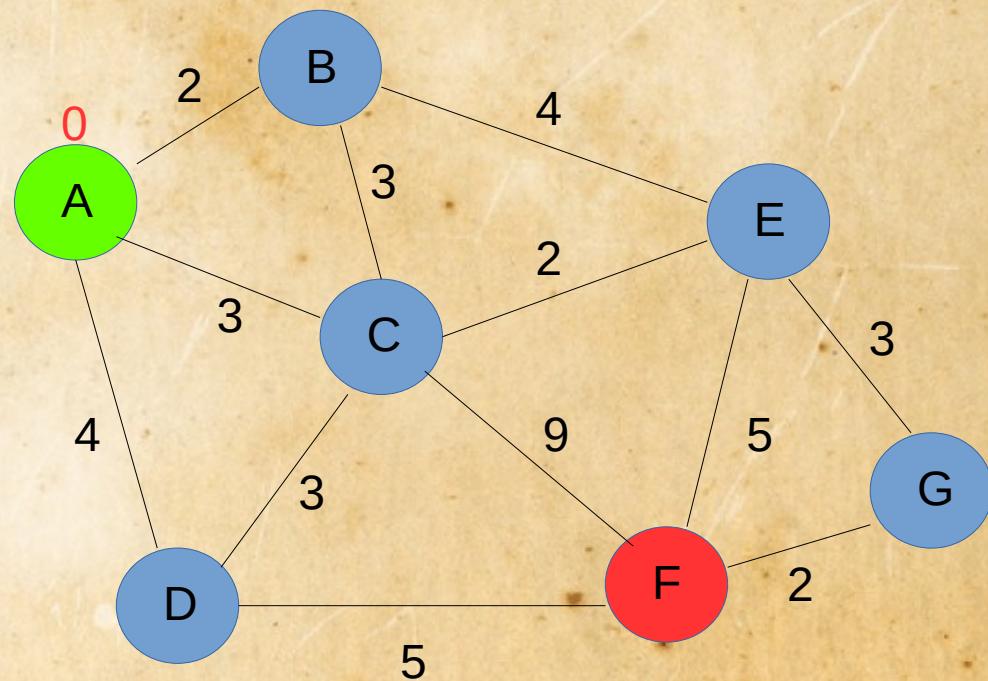
# Dijkstra Kortste Pad Algoritme

- Voor kortste pad in een graaf met verschillende kosten
  - Kortste pad garantie bij **geen** negatieve kosten



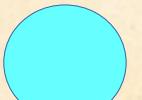
# Dijkstra Kortste Pad

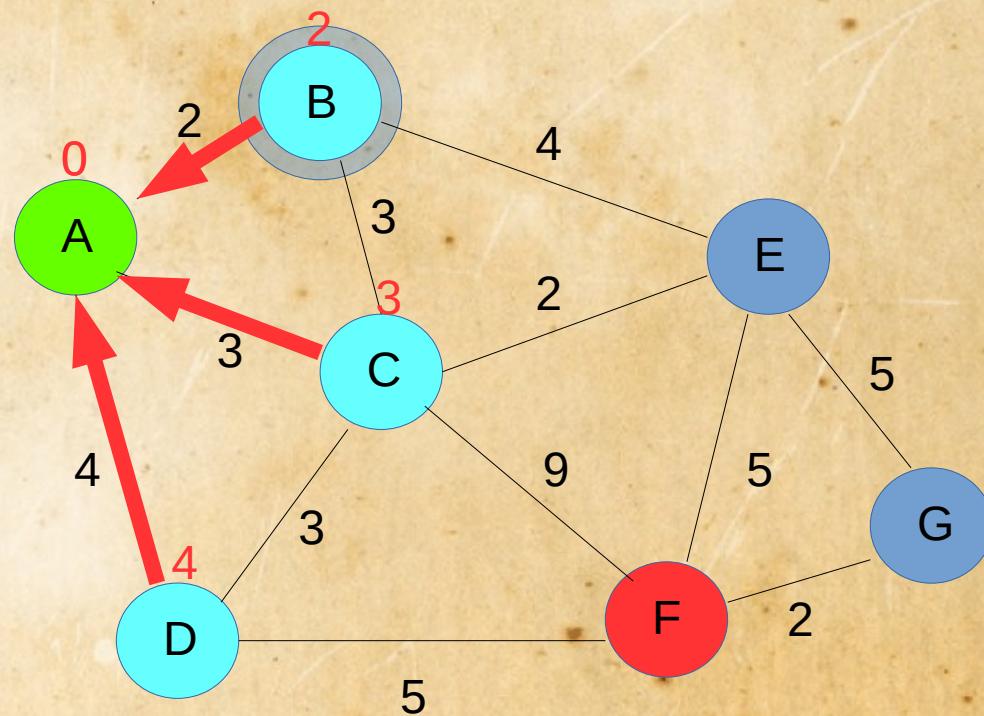
- Selecteer steeds de state met laagste totale kosten



# Dijkstra Kortste Pad

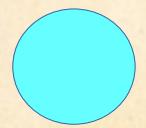
- Selecteer steeds de state met laagste totale kosten

-  Selectie states  
(priority queue)
-  Archief  
(dictionary)



# Dijkstra Kortste Pad

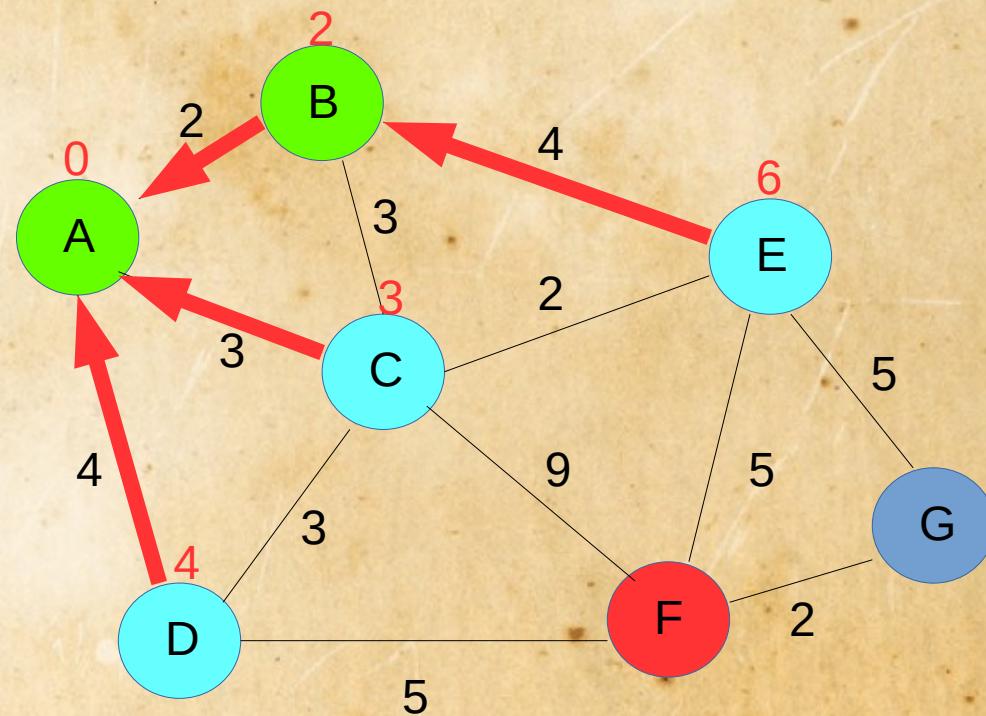
- Selecteer steeds de state met laagste totale kosten



Selectie states  
(priority queue)

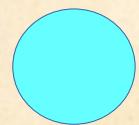


Archief  
(dictionary)

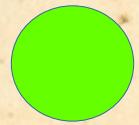


# Dijkstra Kortste Pad

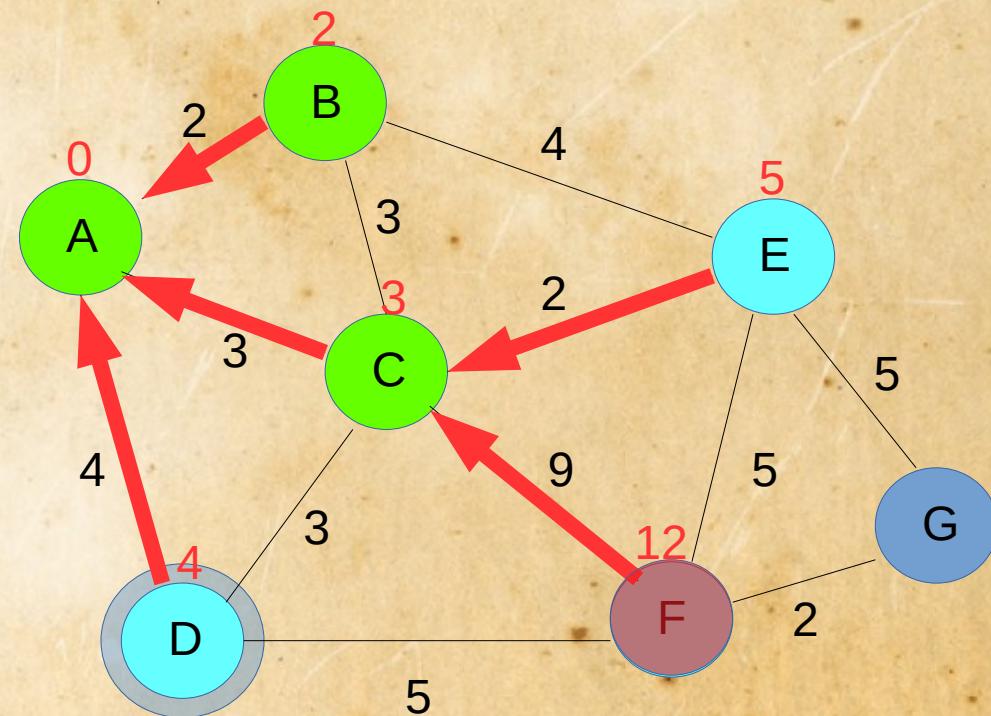
- Selecteer steeds de state met laagste totale kosten



Selectie states  
(priority queue)

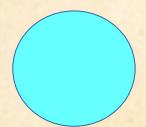


Archief  
(dictionary)



# Dijkstra Kortste Pad

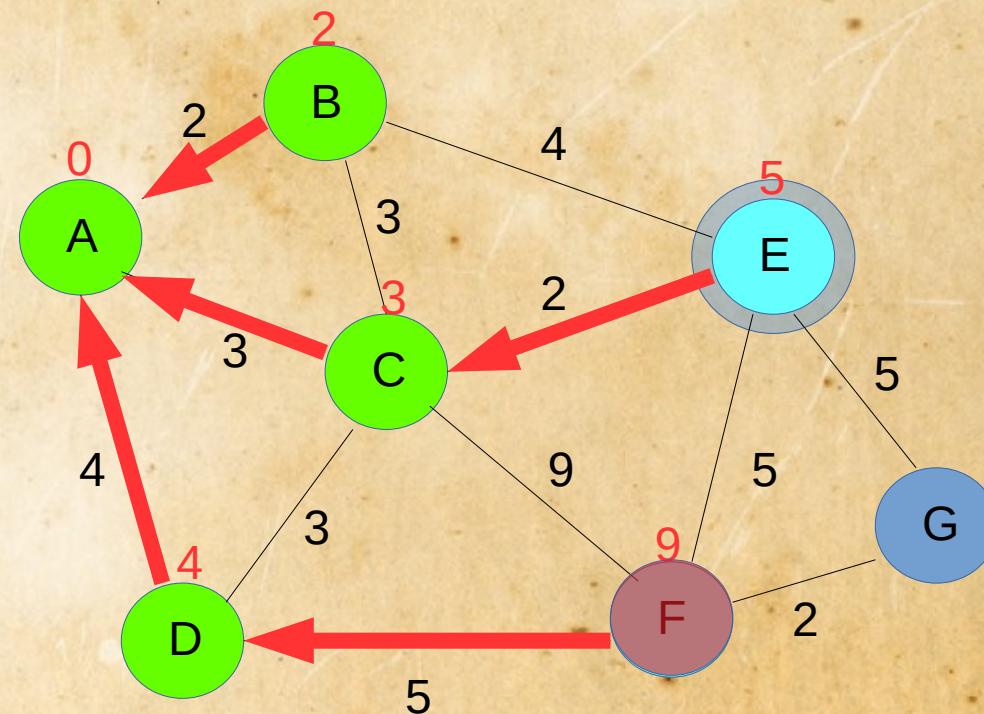
- Selecteer steeds de state met laagste totale kosten



Selectie states  
(priority queue)

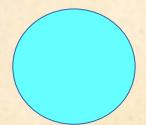


Archief  
(dictionary)



# Dijkstra Kortste Pad

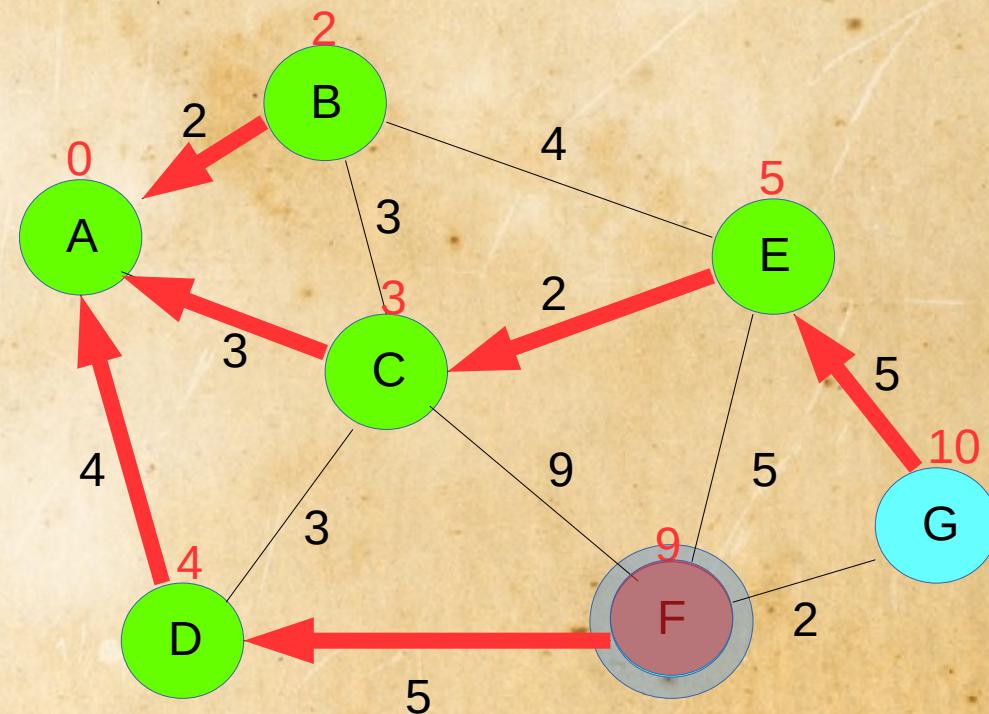
- Selecteer steeds de state met laagste totale kosten



Selectie states  
(priority queue)

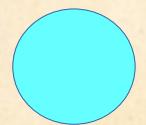


Archief  
(dictionary)



# Dijkstra Kortste Pad

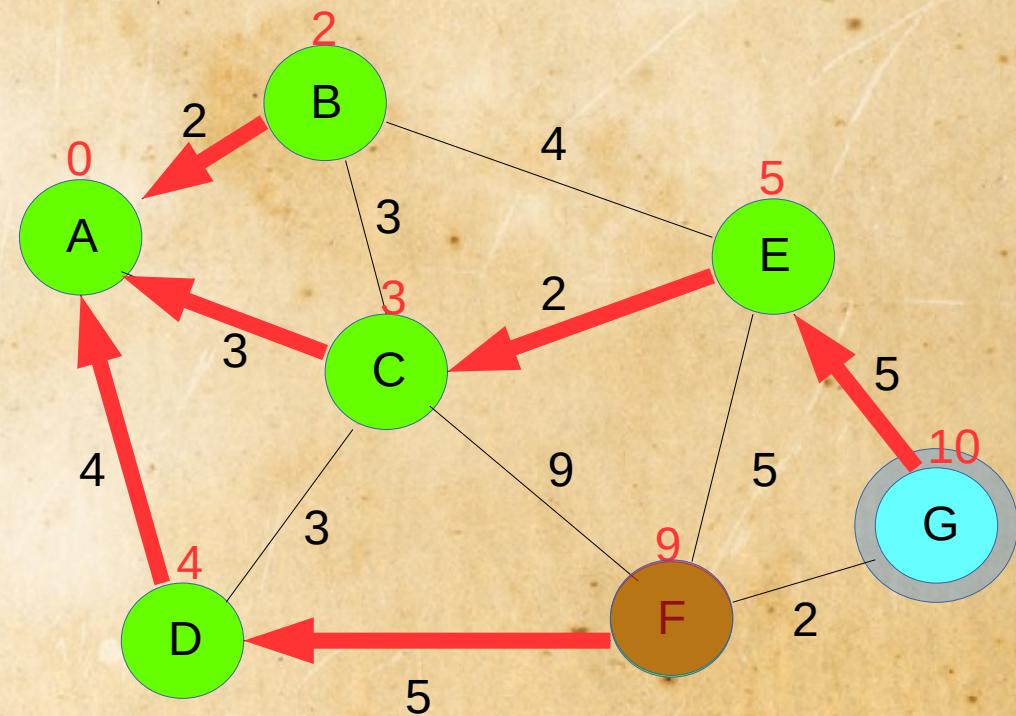
- Selecteer steeds de state met laagste totale kosten



Selectie states  
(priority queue)



Archief  
(dictionary)



# A\*

- Optimalisatie op Dijkstra met gebruik van **heuristiek**
  - De **heuristiek** geeft een schatting van de kosten van het pad wat we nog moeten afleggen
  - Korste pad garantie wanneer:
    - er **geen** negatieve kosten voorkomen
    - wanneer de heuristiek nooit een overschatting van de kosten van een pad geeft, **admissible**

# A\*

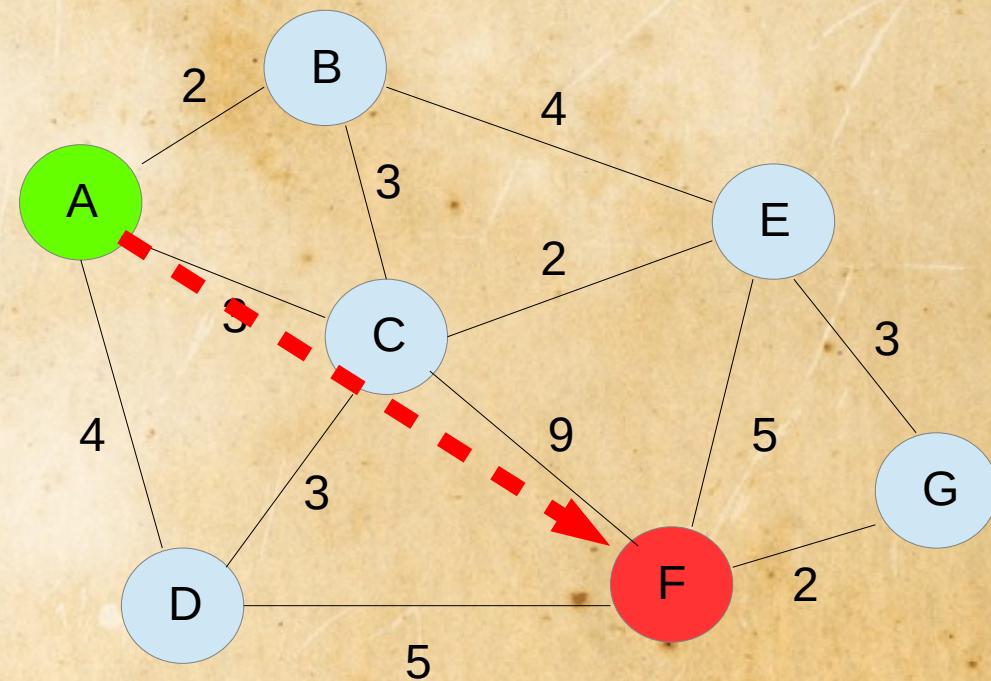
- **Admissible**, geen overschatting van kosten



# A\*

- Selecteer steeds de state met laagste som van kosten en heuristiek

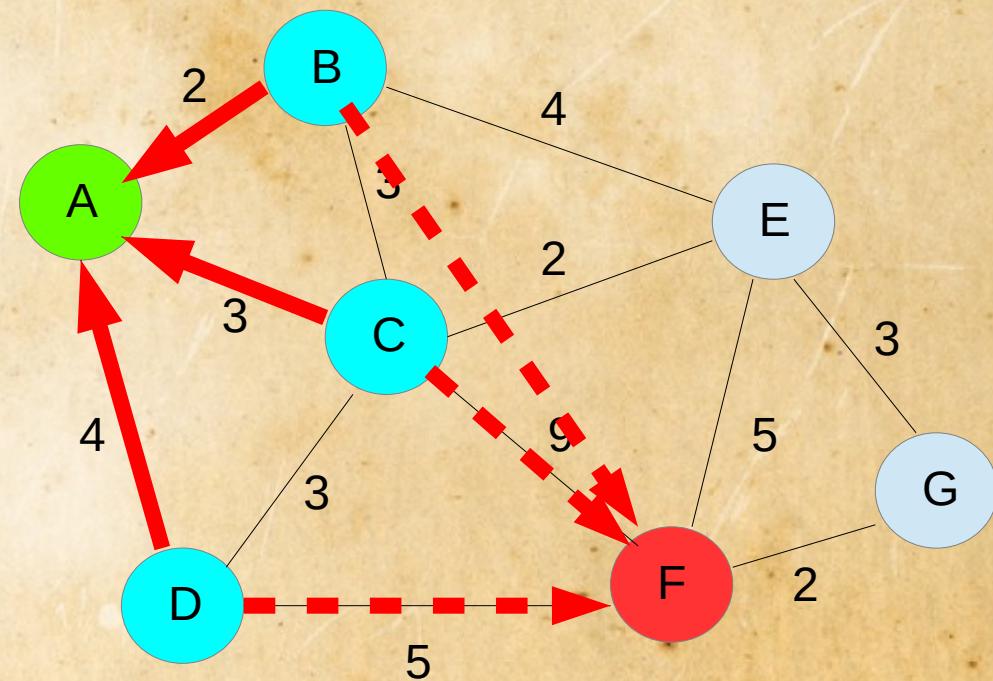
| node | prev | cost | heur | sum |
|------|------|------|------|-----|
| A    | -    | 0    | 8    | 8   |



# A\*

- Selecteer steeds de state met laagste som van kosten en heuristiek

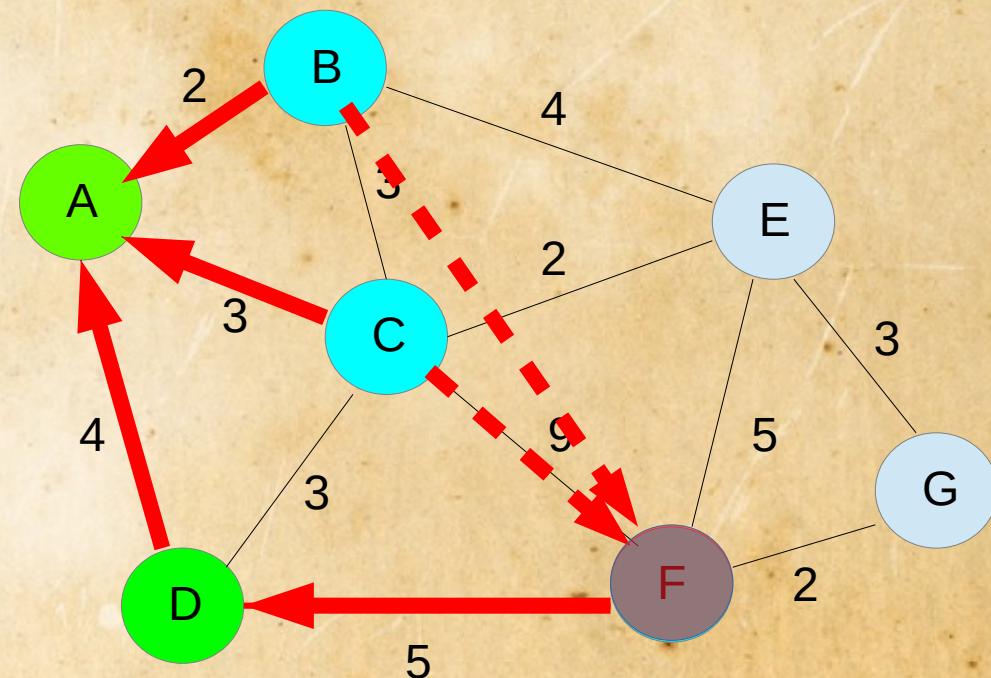
| node | prev | cost | heur | sum |
|------|------|------|------|-----|
| A    | -    | 0    | 8    | 8   |
| B    | A    | 2    | 9    | 11  |
| C    | A    | 3    | 7    | 10  |
| D    | A    | 4    | 5    | 9   |



# A\*

- Selecteer steeds de state met laagste som van kosten en heuristiek

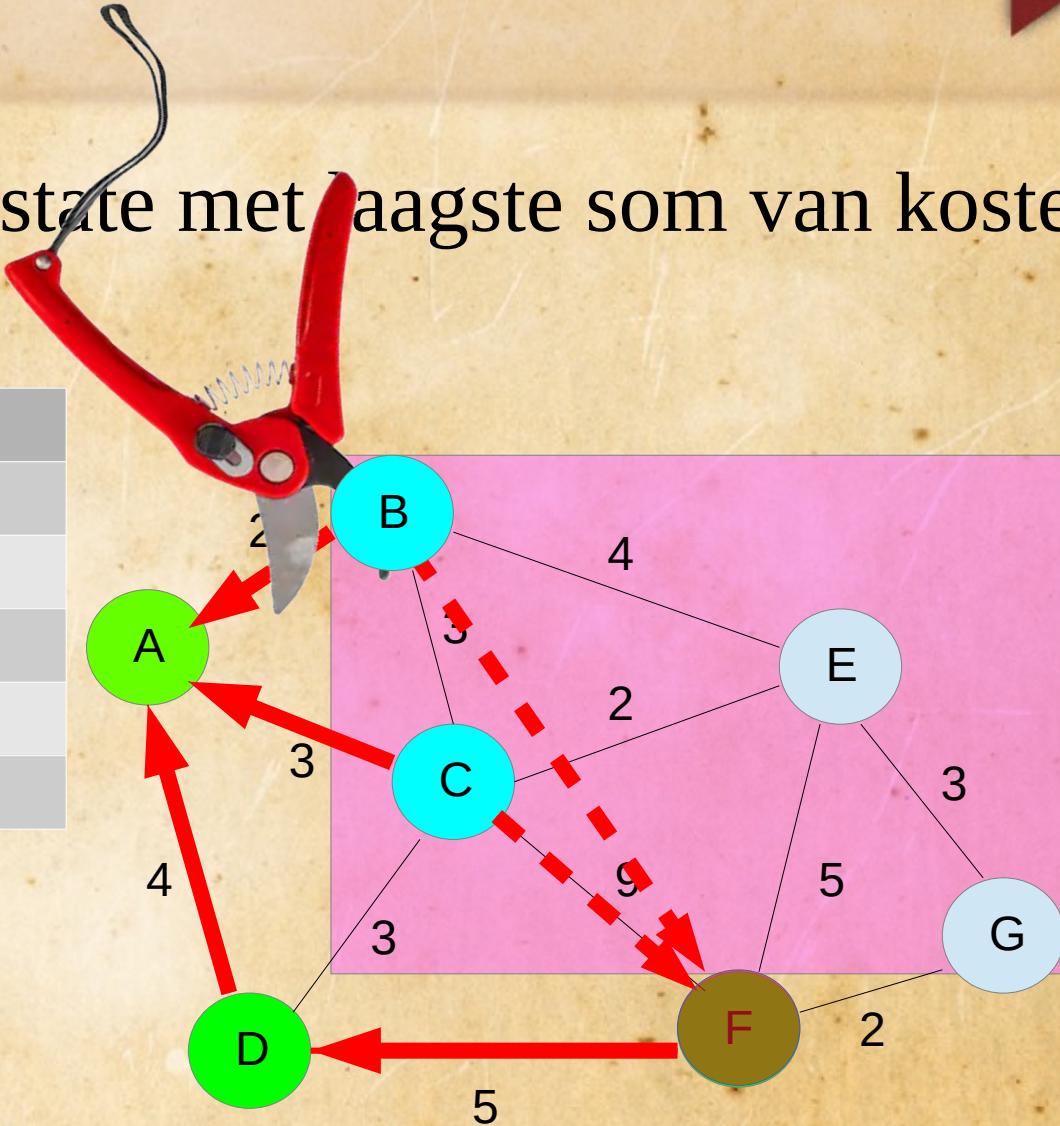
| node | prev | cost | heur | sum |
|------|------|------|------|-----|
| A    | -    | 0    | 8    | 8   |
| B    | A    | 2    | 9    | 11  |
| C    | A    | 3    | 7    | 10  |
| D    | A    | 4    | 5    | 9   |
| F    | D    | 9    | 0    | 9   |



# A\*

- Selecteer steeds de state met laagste som van kosten en heuristiek

| node | prev | cost | heur | sum |
|------|------|------|------|-----|
| A    | -    | 0    | 8    | 8   |
| B    | A    | 2    | 9    | 11  |
| C    | A    | 3    | 7    | 10  |
| D    | A    | 4    | 5    | 9   |
| F    | D    | 9    | 0    | 9   |

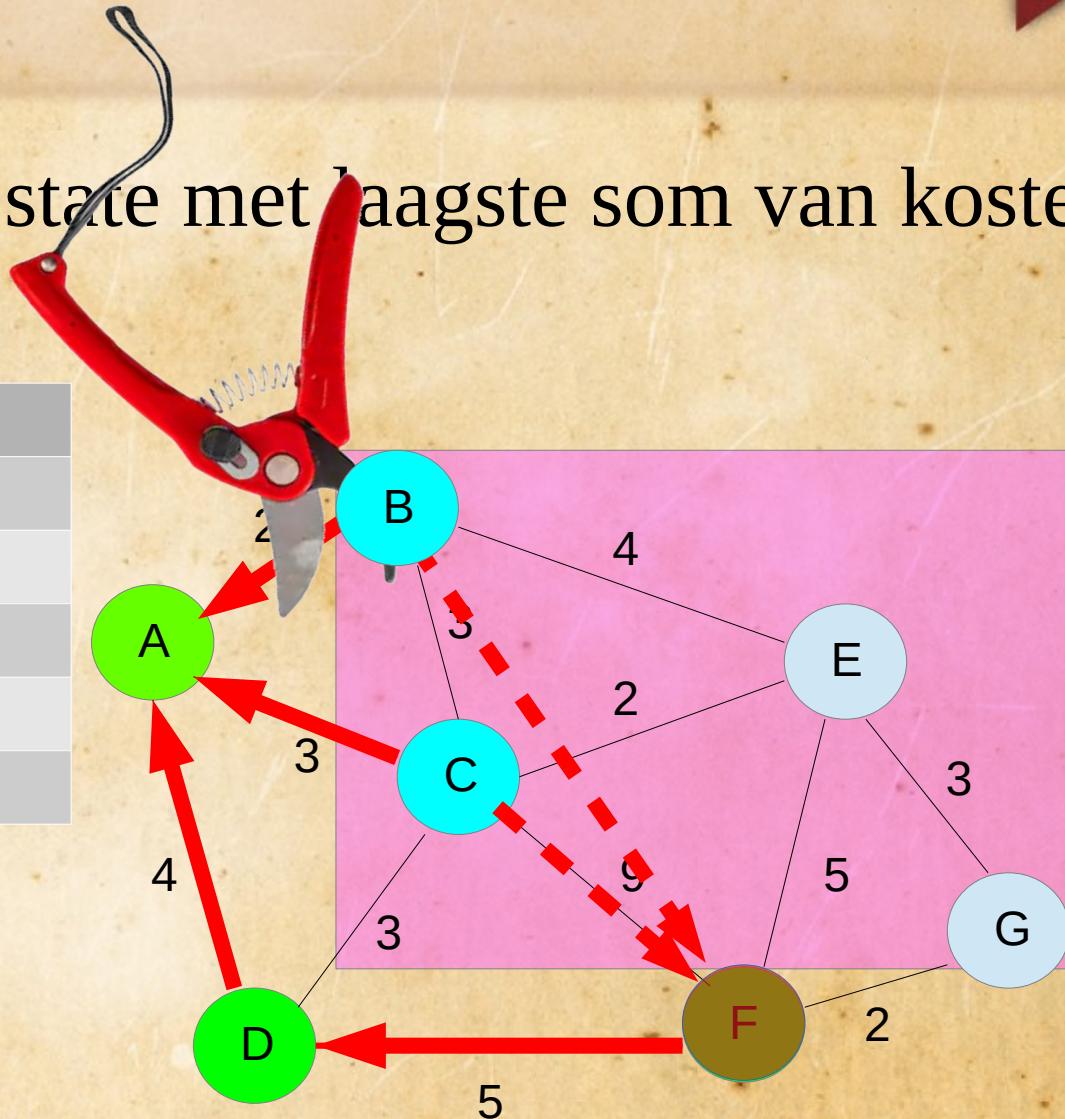


# A\*

- Selecteer steeds de state met laagste som van kosten en heuristiek

| node | prev | cost | heur | sum |
|------|------|------|------|-----|
| A    | -    | 0    | 0    | 8   |
| B    | A    | 2    | 0    | 2   |
| C    | A    | 3    | 0    | 3   |
| D    | A    | 4    | 0    | 4   |
| F    | D    | 9    | 0    | 9   |

Als heur=0 dan is  
A\* gelijk aan Dijkstra



# A\*

- Demos:
  - <http://qiao.github.io/PathFinding.js/visual/>
  - Path Planning in games, Daan van der Berg
    - Gras: 1
    - Kleine heuvel: 2
    - Heuvel: 3
    - Bos: 12
    - Berg: 14
    - Water: 1000000

# A\*

- Slide puzzle

|   |   |
|---|---|
| 3 | 1 |
| 2 |   |

Heuristic?



|   |   |
|---|---|
| 1 | 2 |
| 3 |   |



# A\*

- Slide puzzle

|   |   |
|---|---|
| 3 | 1 |
| 2 |   |

Heuristic: count wrong tiles

$$0+3=3$$

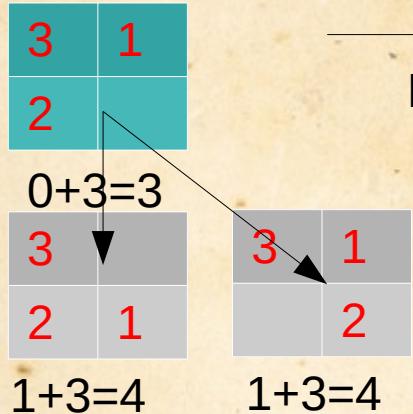


|   |   |
|---|---|
| 1 | 2 |
| 3 |   |



# A\*

- Slide puzzle

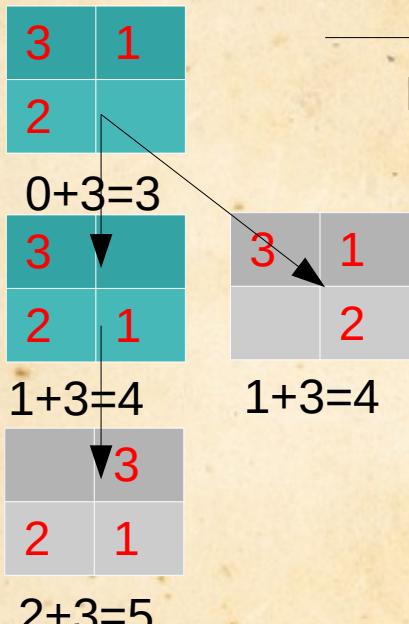


Heuristic: count wrong tiles



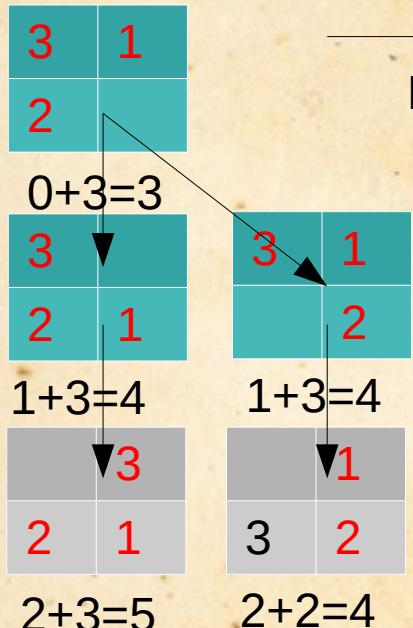
# A\*

- Slide puzzle



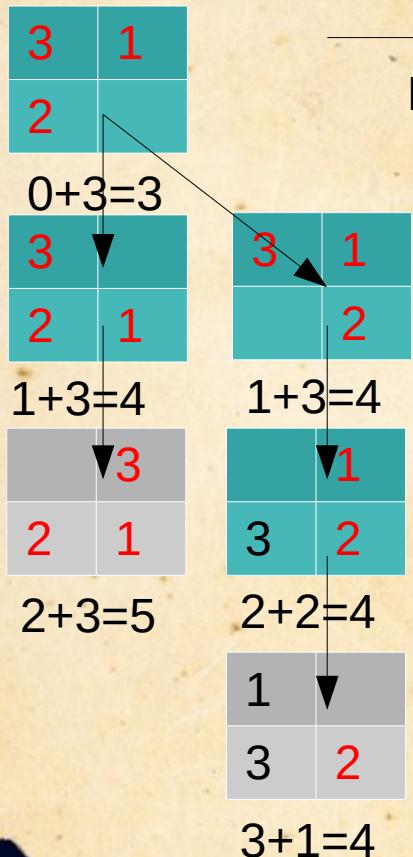
# A\*

- Slide puzzle



# A\*

- Slide puzzle



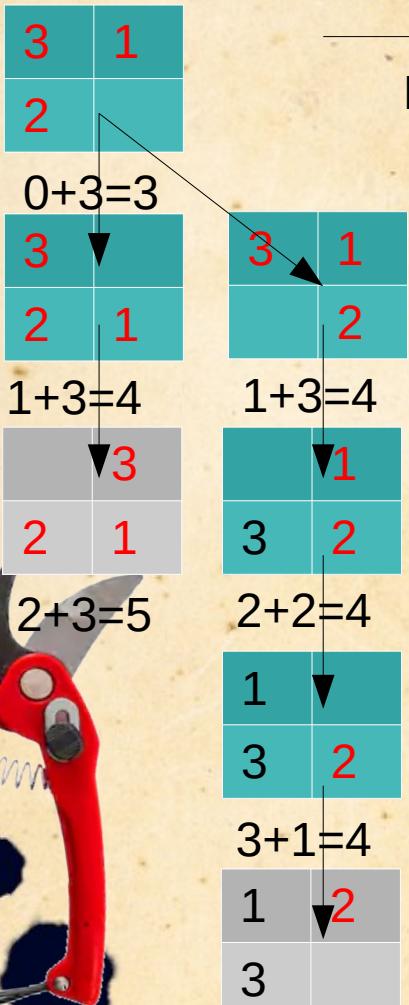
Heuristic: count wrong tiles

|   |   |
|---|---|
| 1 | 2 |
| 3 |   |



# A\*

- Slide puzzle



Heuristic: count wrong tiles

|   |   |
|---|---|
| 1 | 2 |
| 3 |   |



# A\*

- Slide puzzle

|   |   |
|---|---|
| 3 | 1 |
| 2 |   |

Heuristic: sum of tile distances



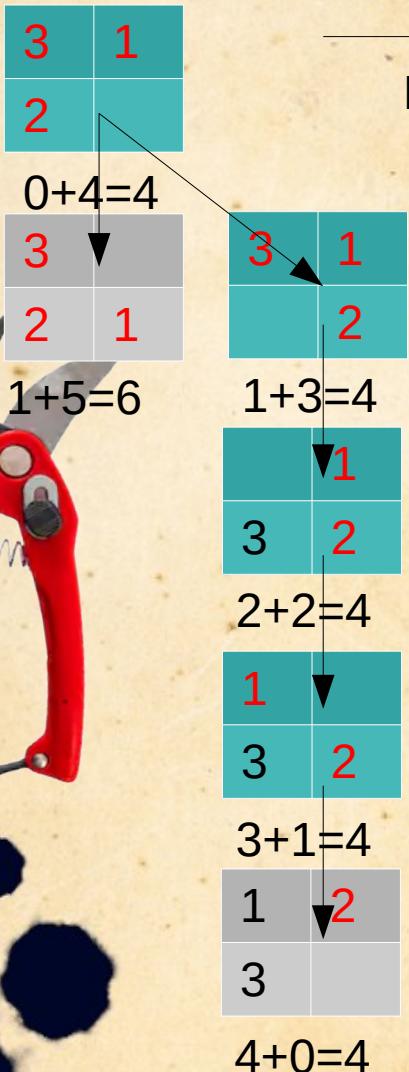
|   |   |
|---|---|
| 1 | 2 |
| 3 |   |

$$0+4=4$$



# A\*

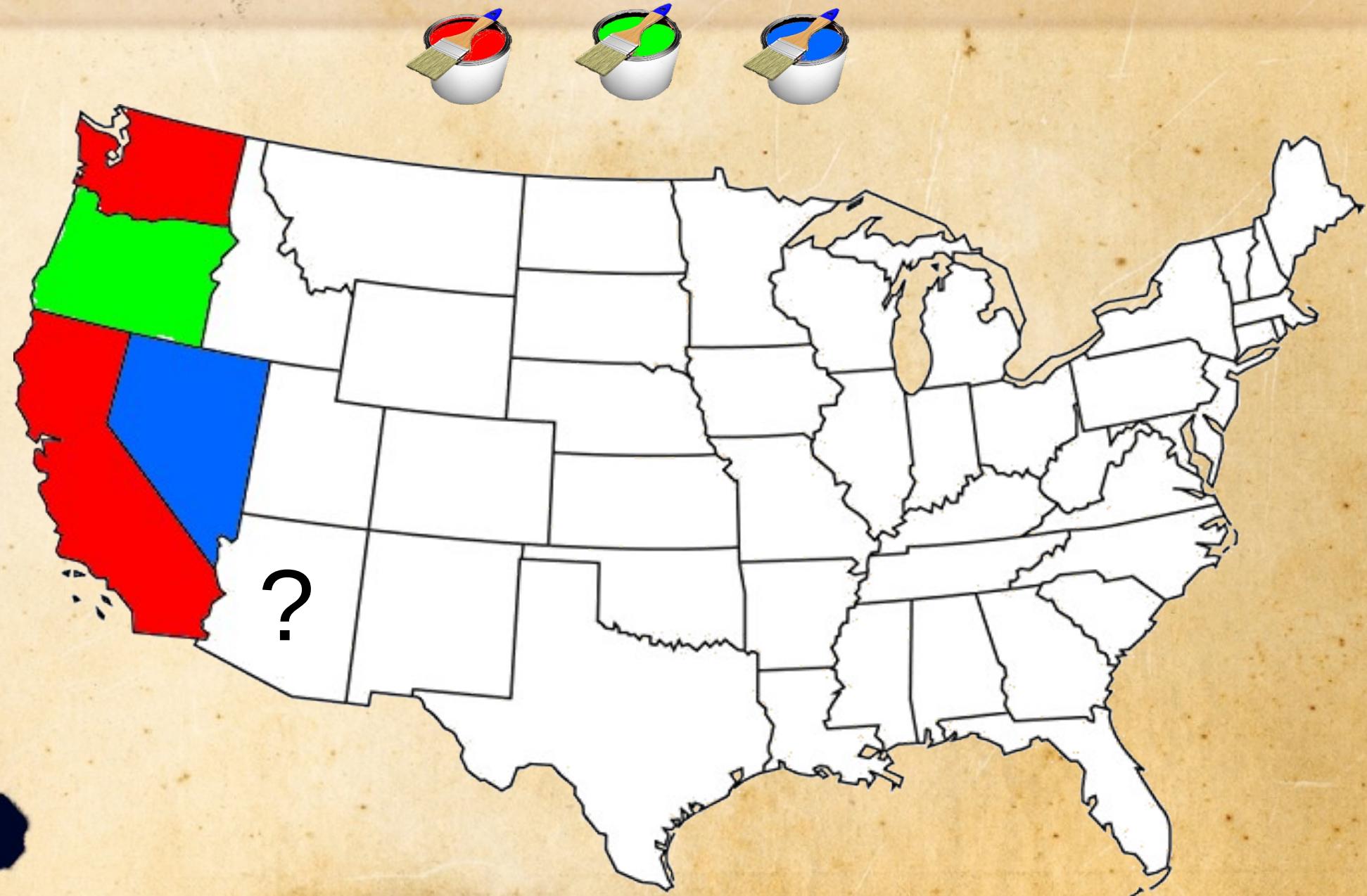
- Slide puzzle



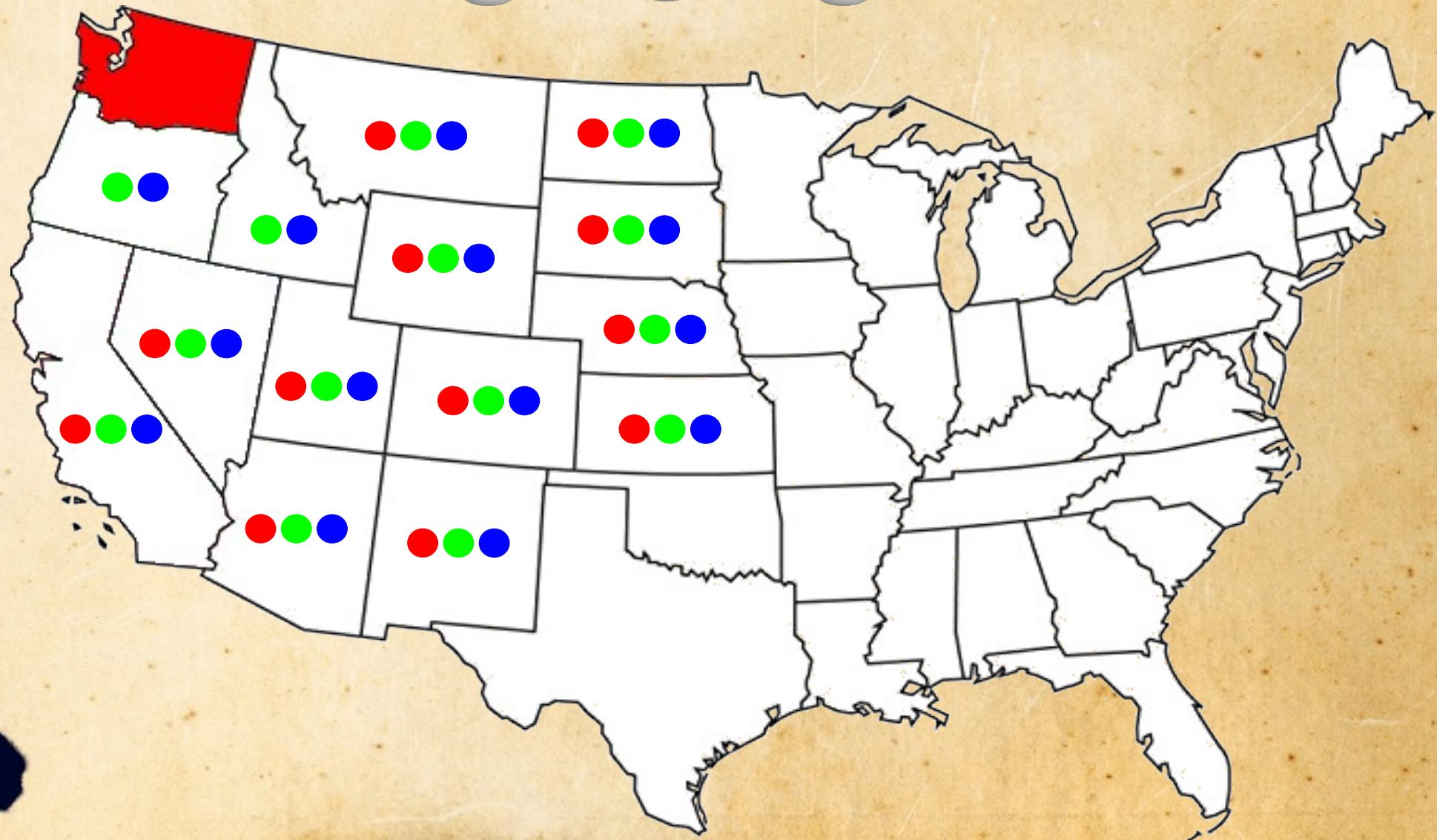
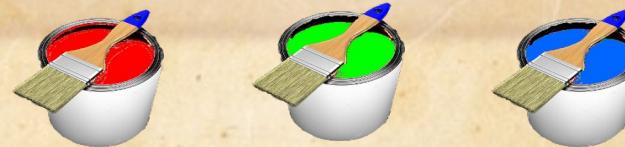
Heuristic: sum of tile distances



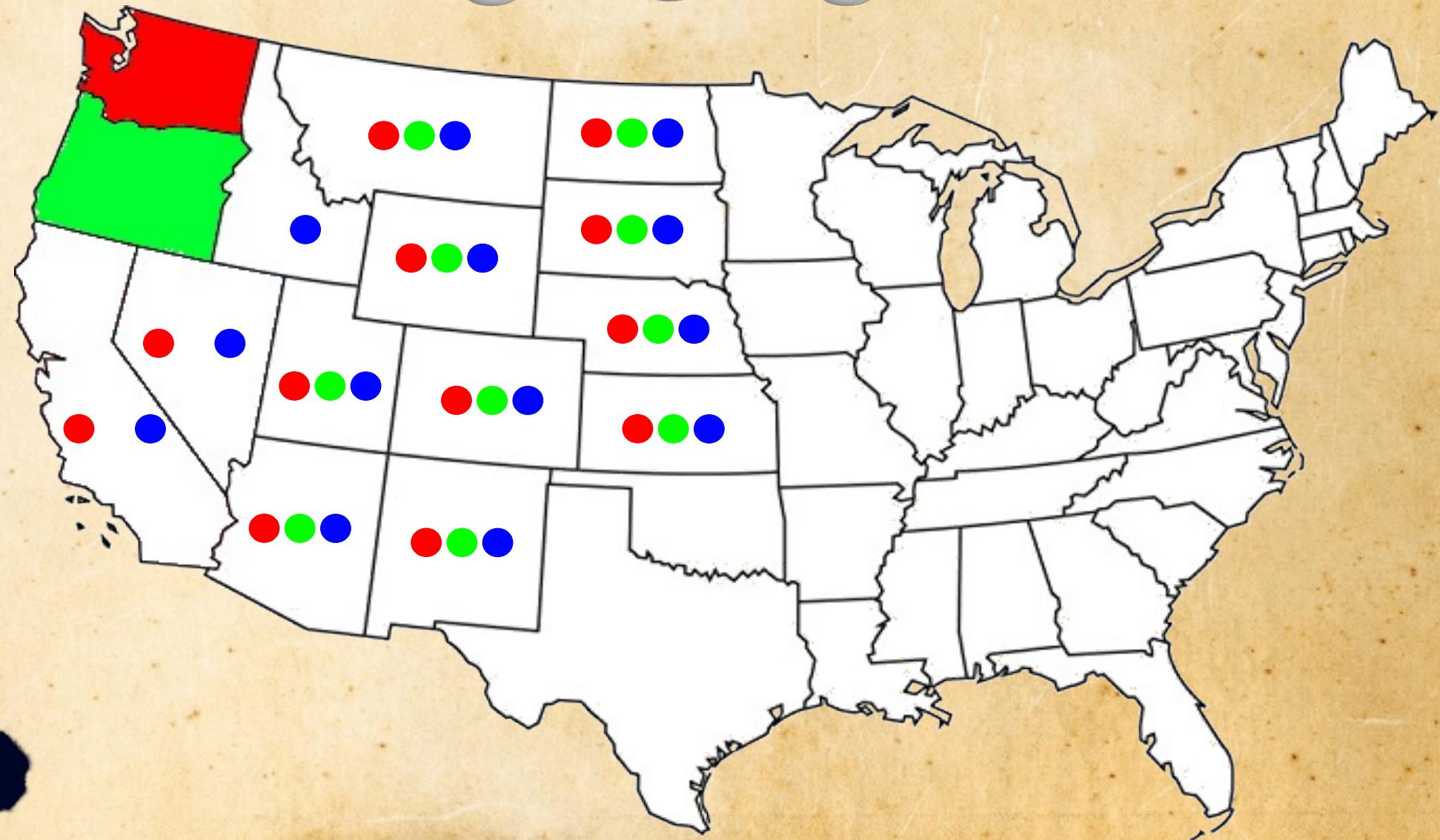
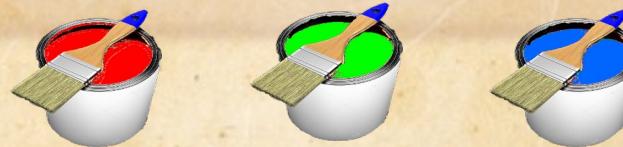
# Domein specifiek prunen



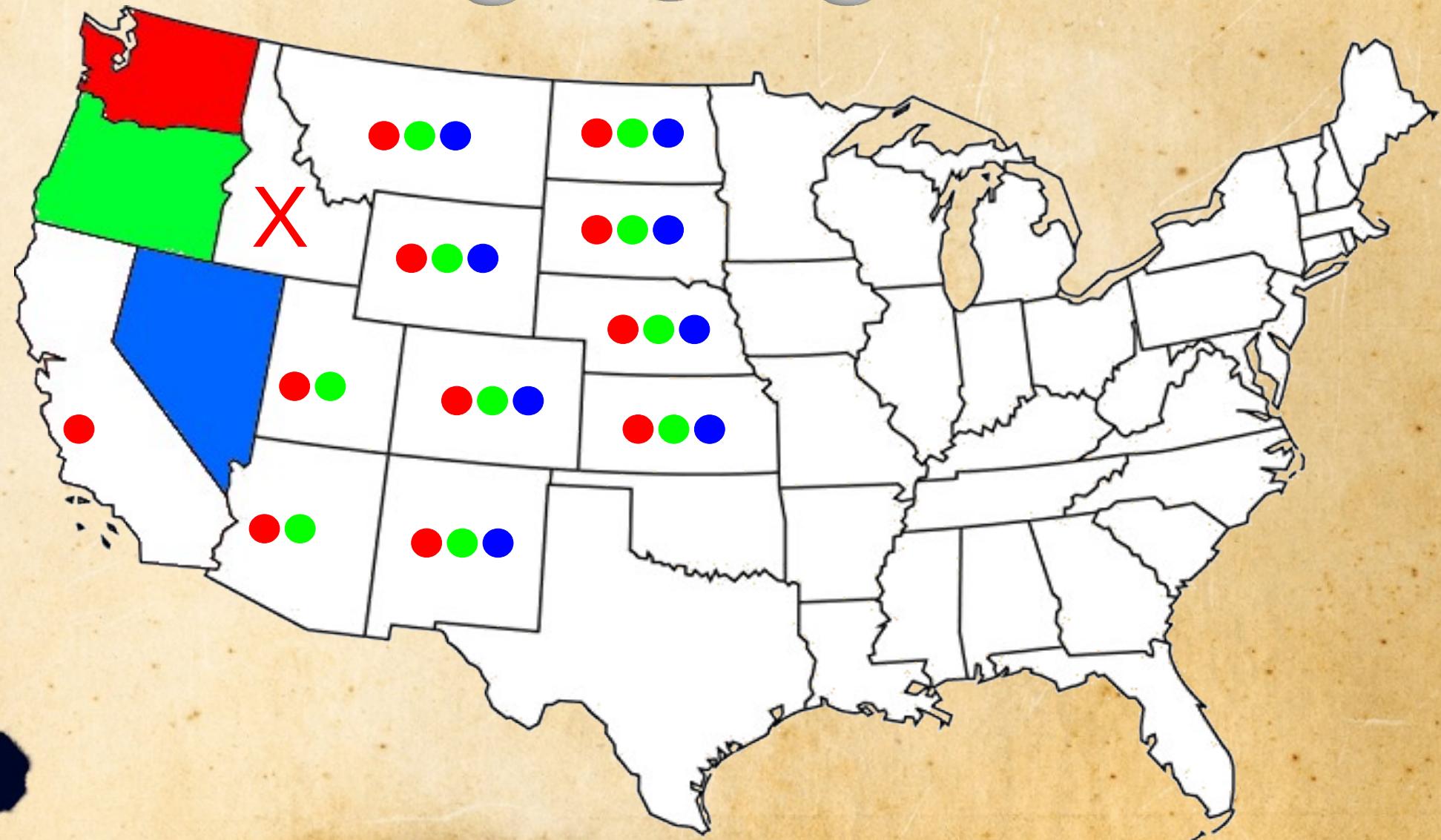
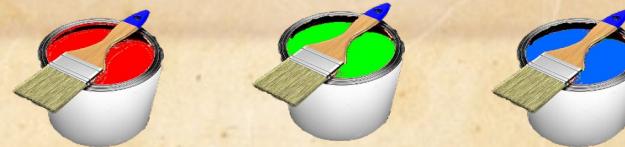
# Domein specifiek prunen



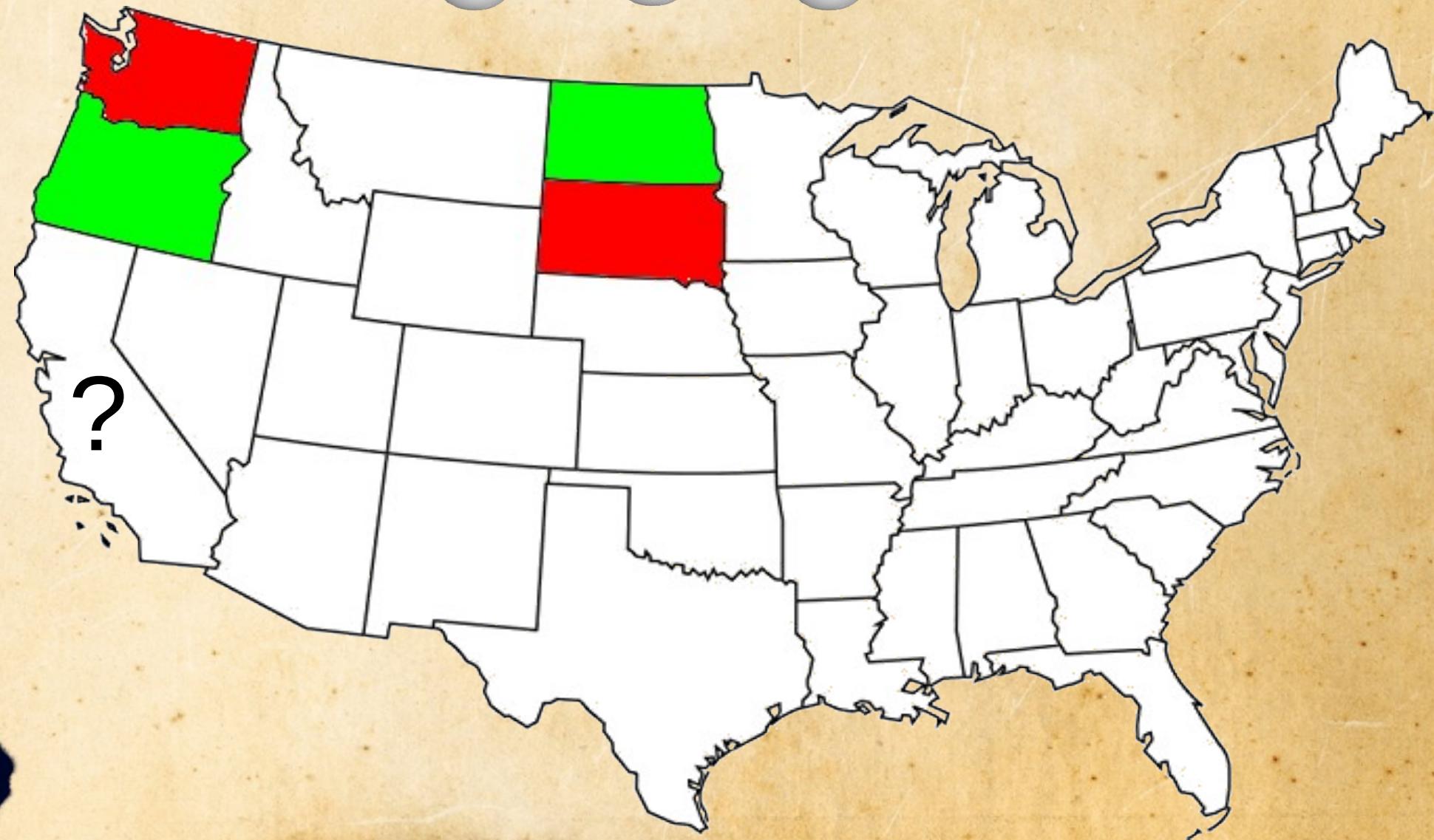
# Domein specifiek prunen



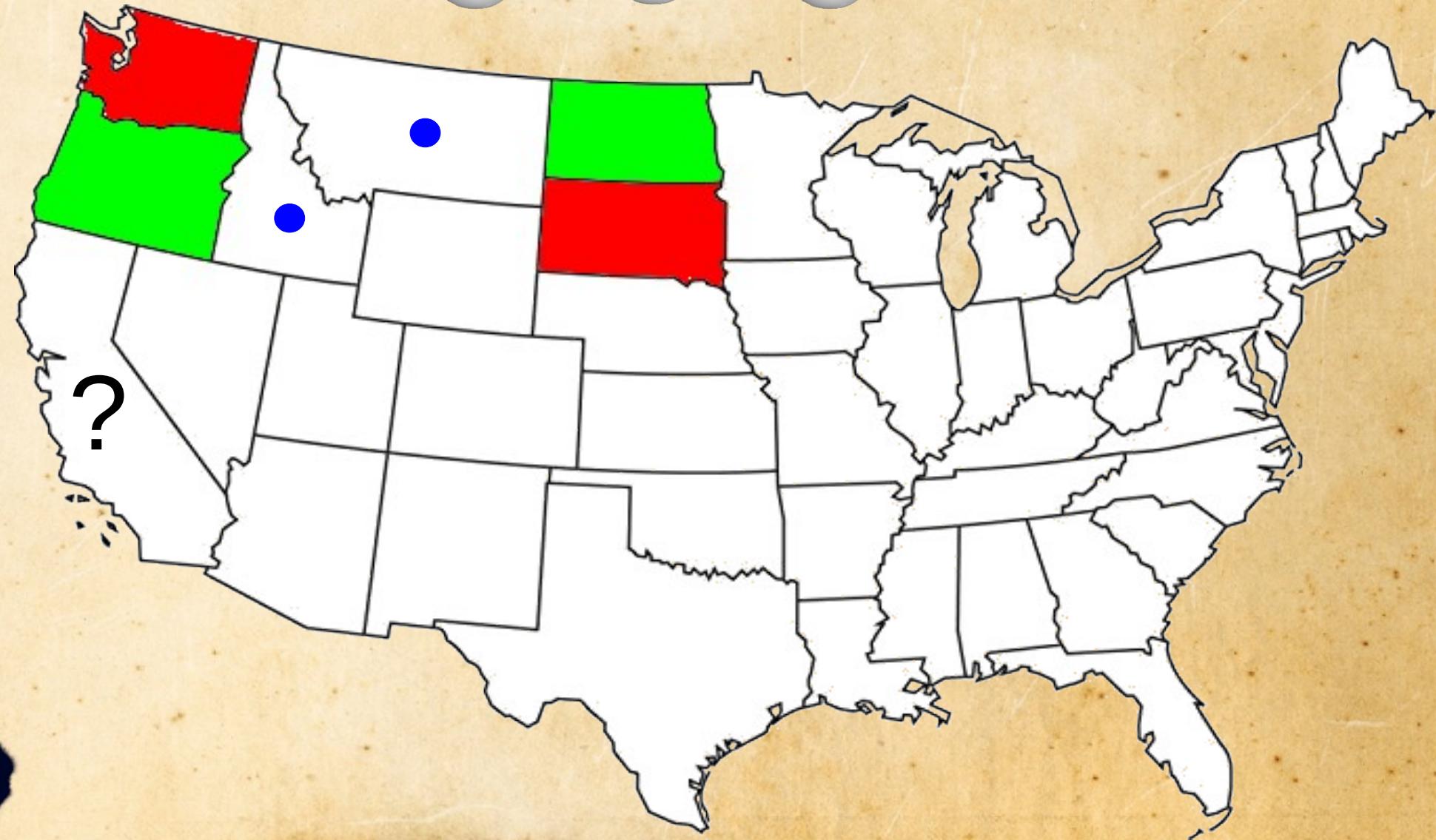
# Domein specifiek prunen



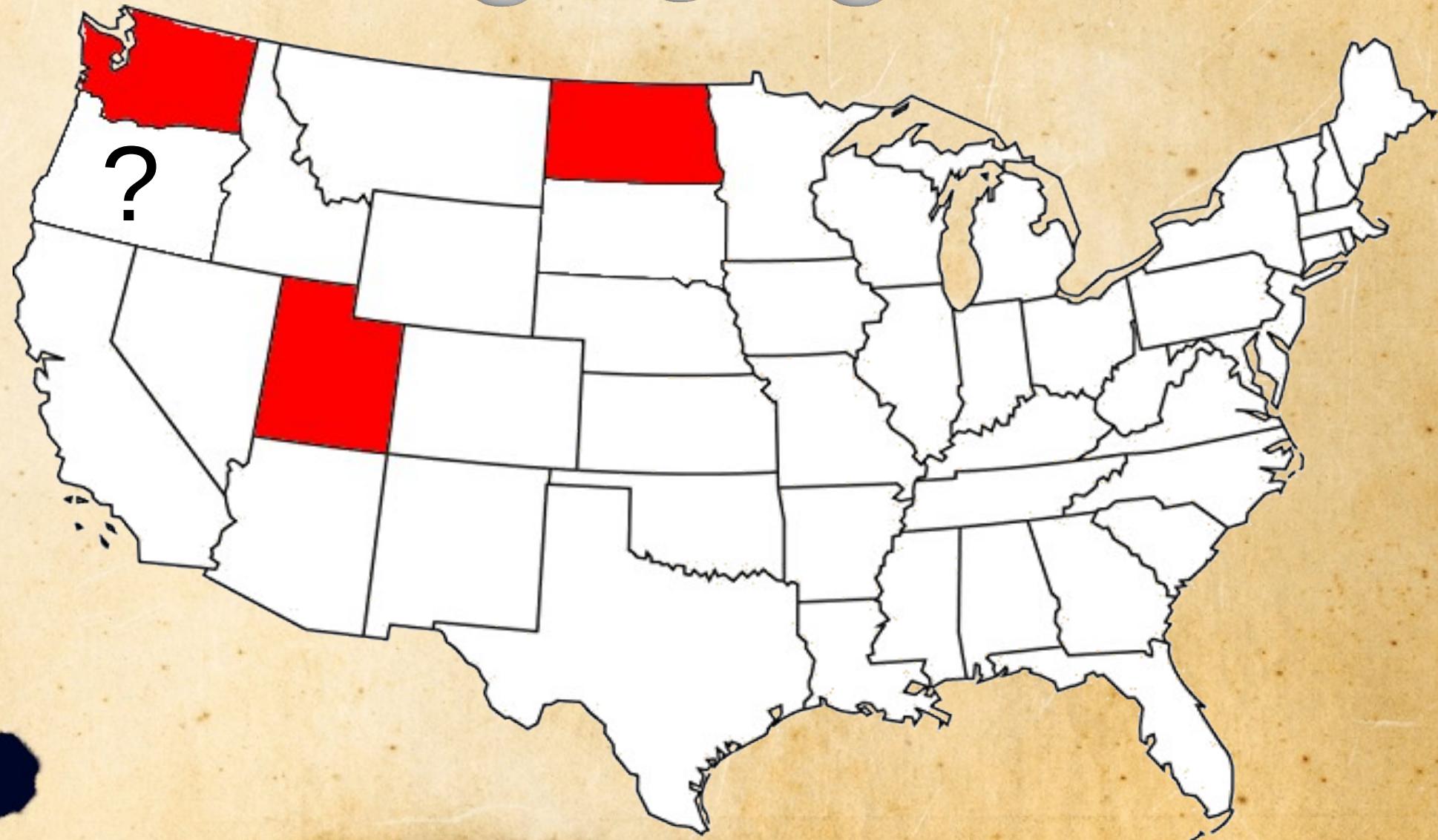
# Domein specifiek prunen



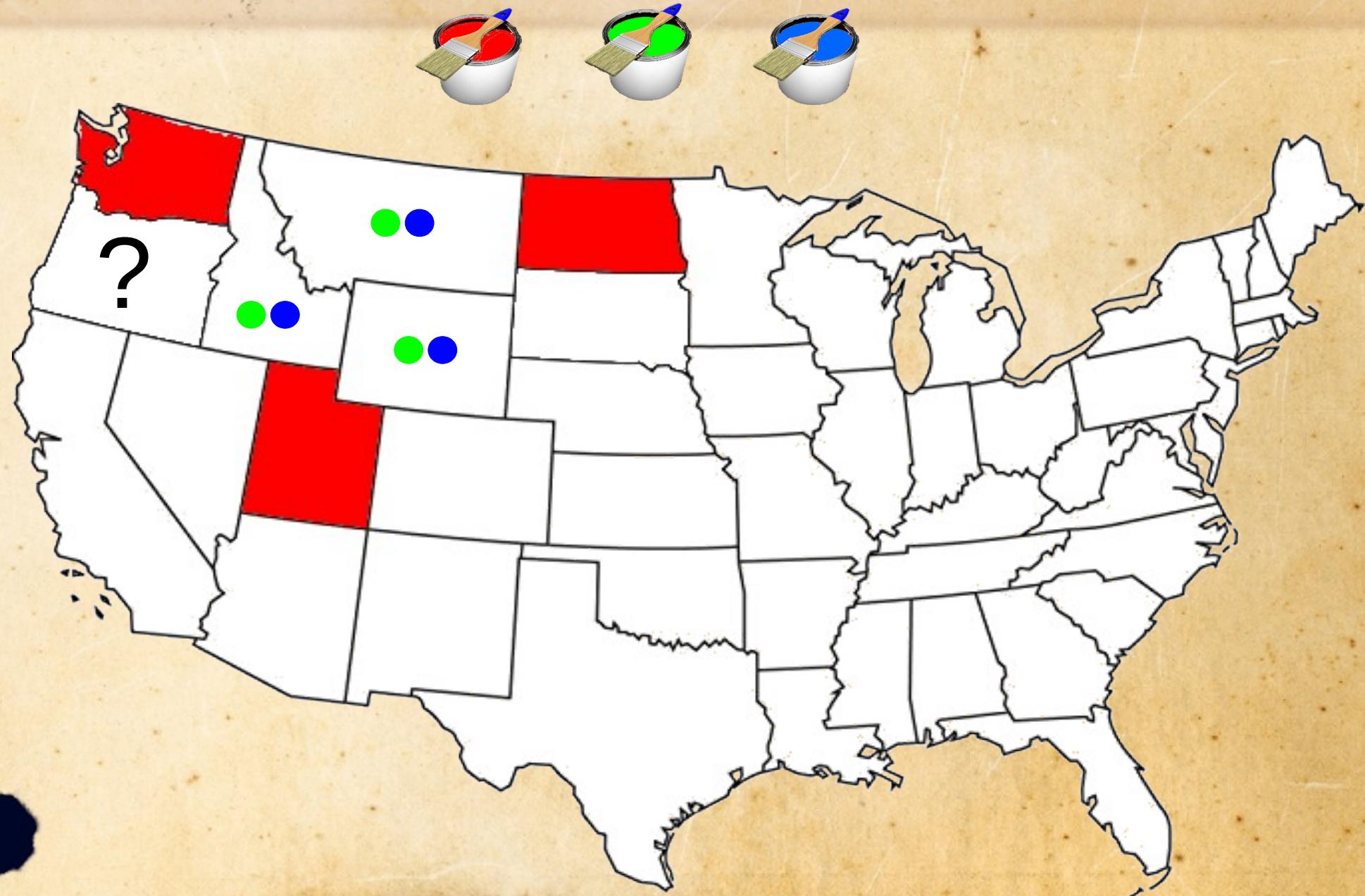
# Domein specifiek prunen



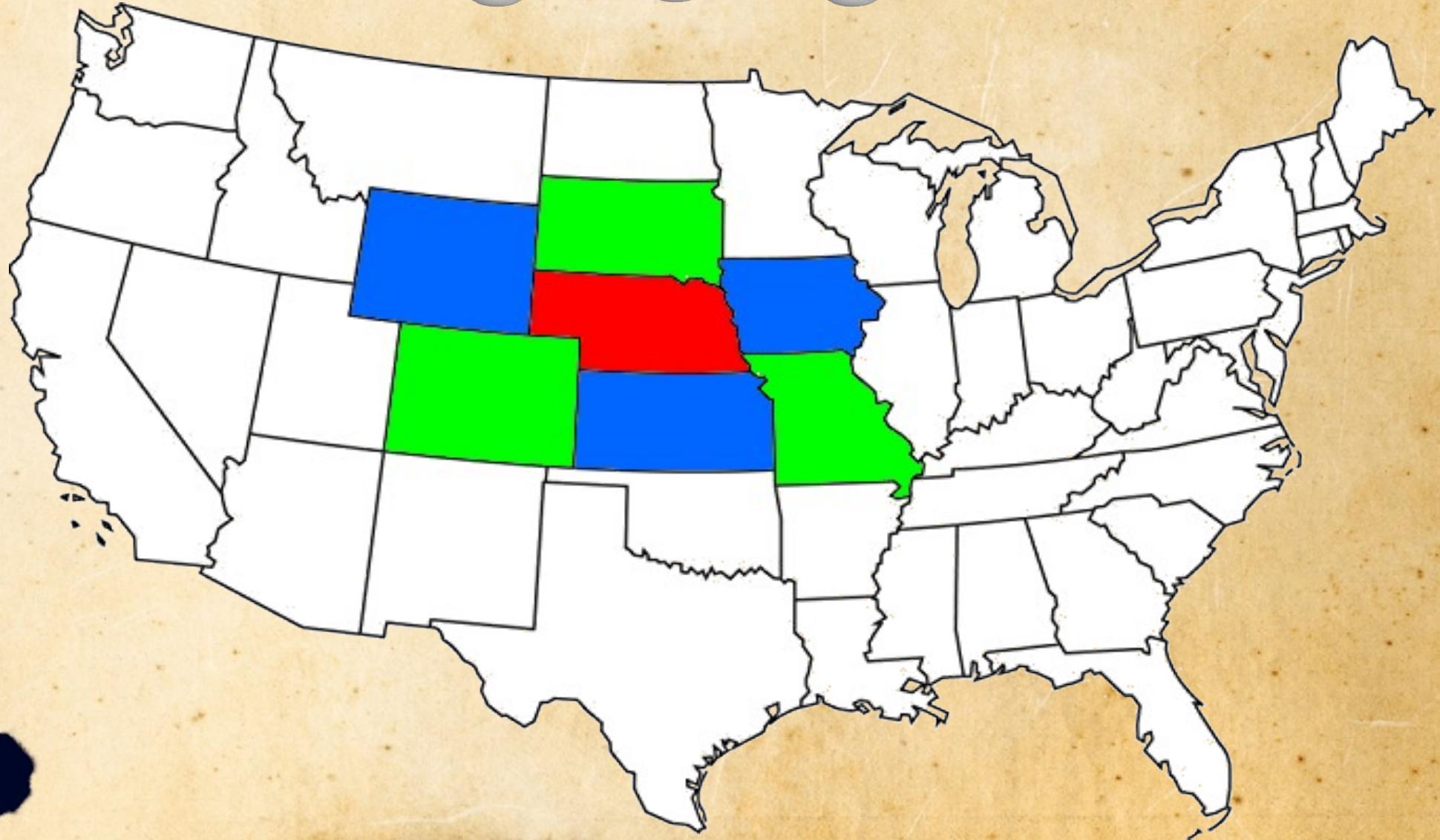
# Domein specifiek prunen



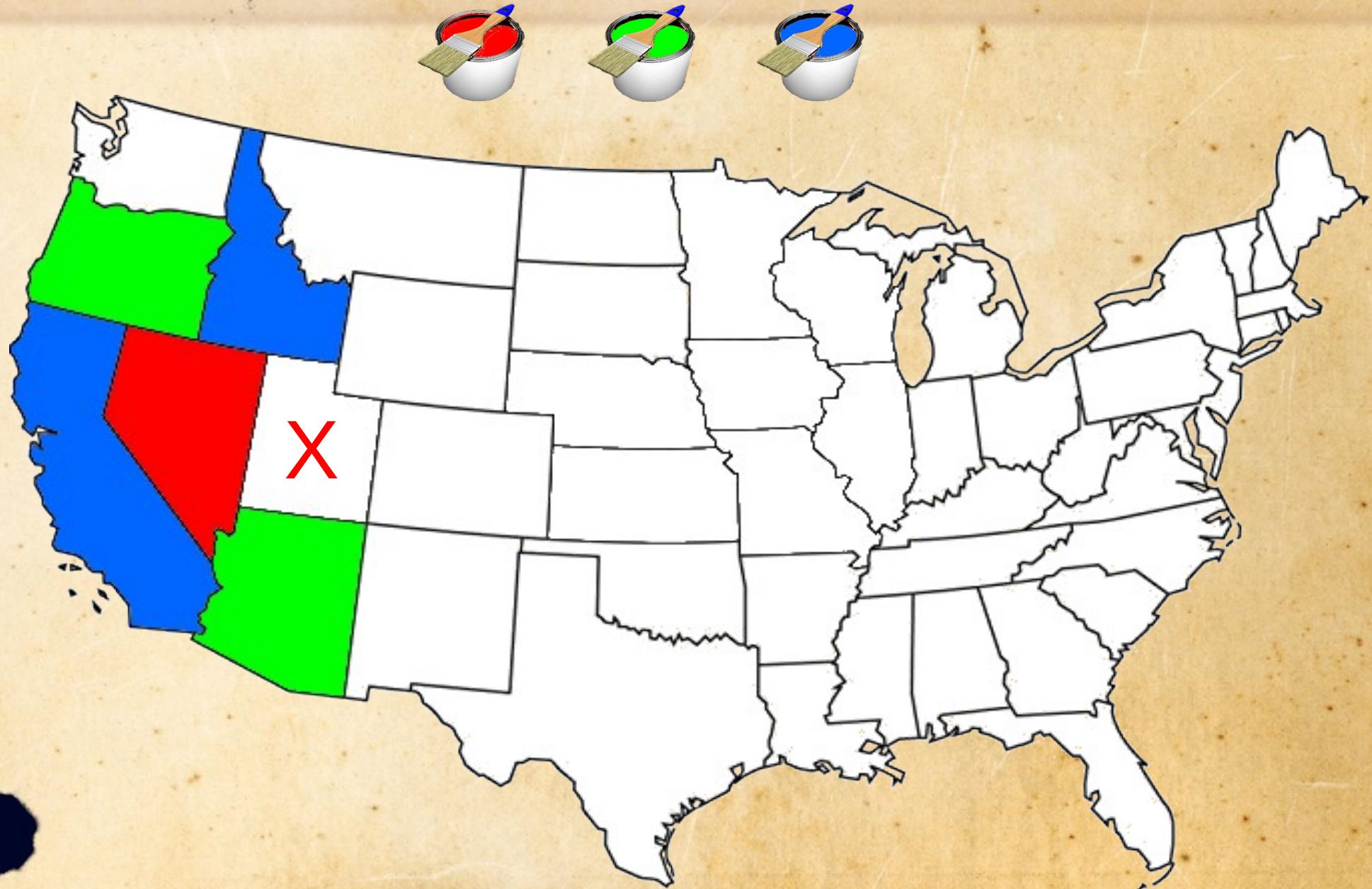
# Domein specifiek prunen



# Domein specifiek prunen

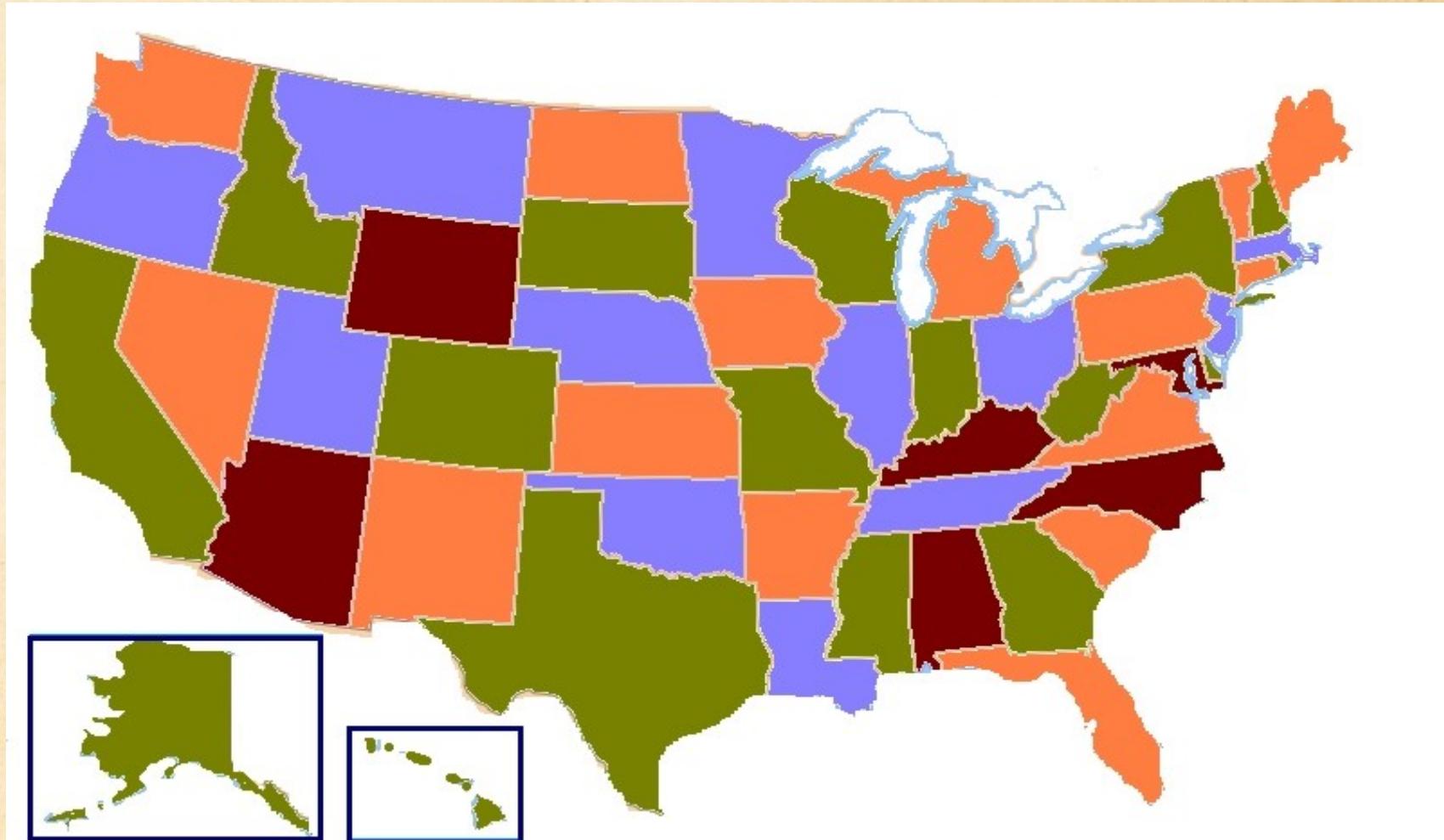


# Domein specifiek prunen



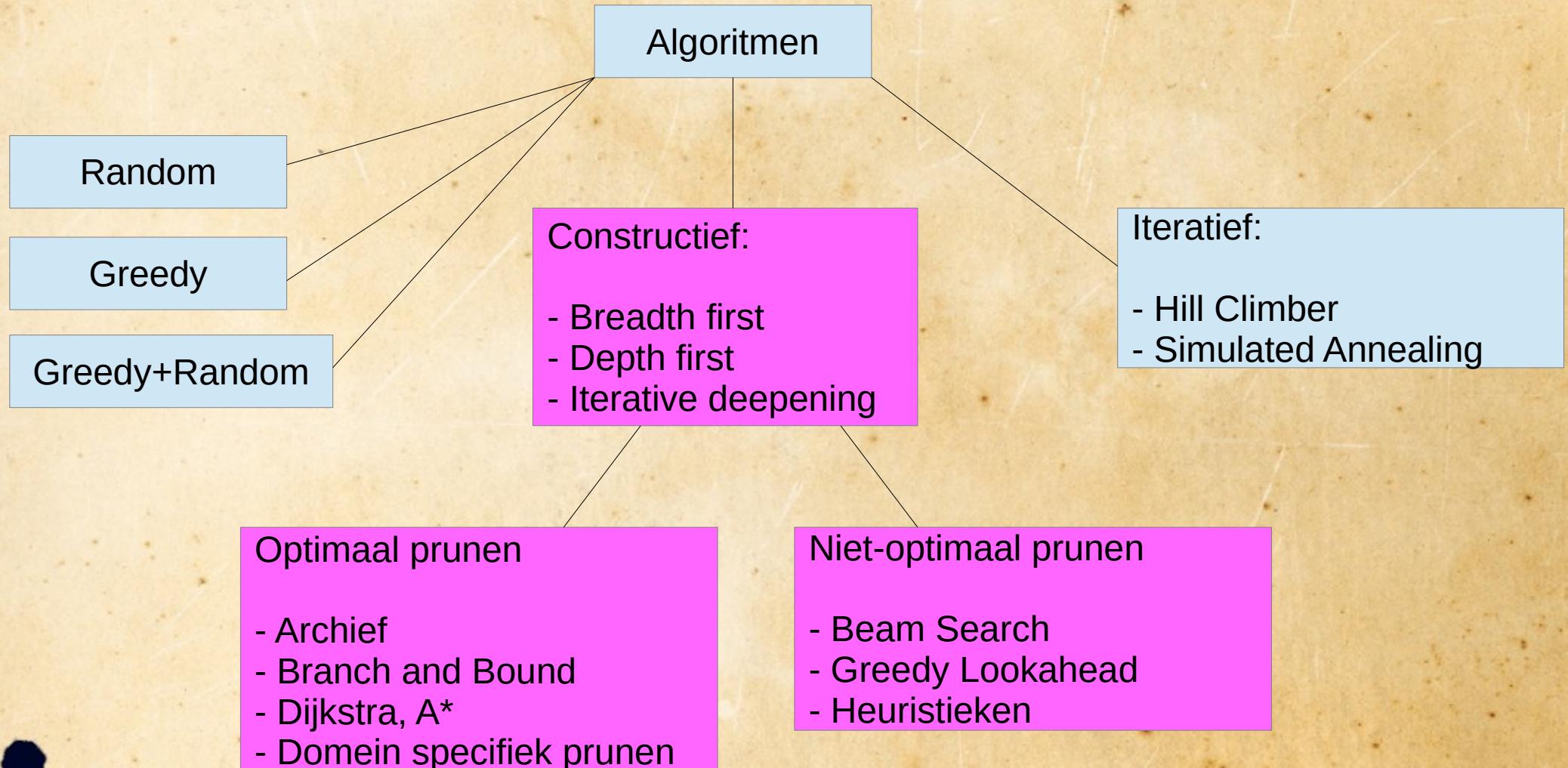
# Domein specifiek prunen

Met 4 kleuren kan het wel



[https://en.wikipedia.org/wiki/Four\\_color\\_theorem](https://en.wikipedia.org/wiki/Four_color_theorem)

# Algoritmen

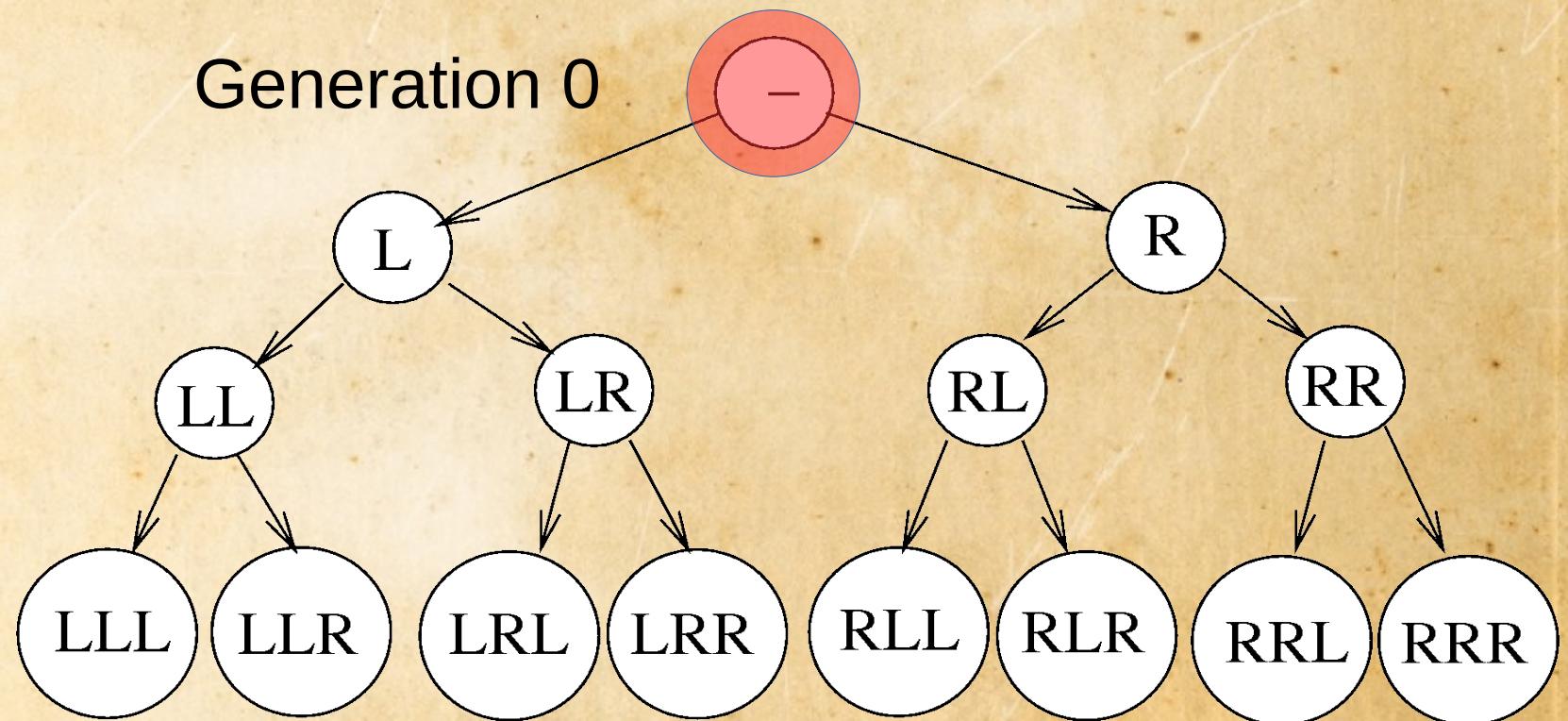


# Niet-optimaal prunen



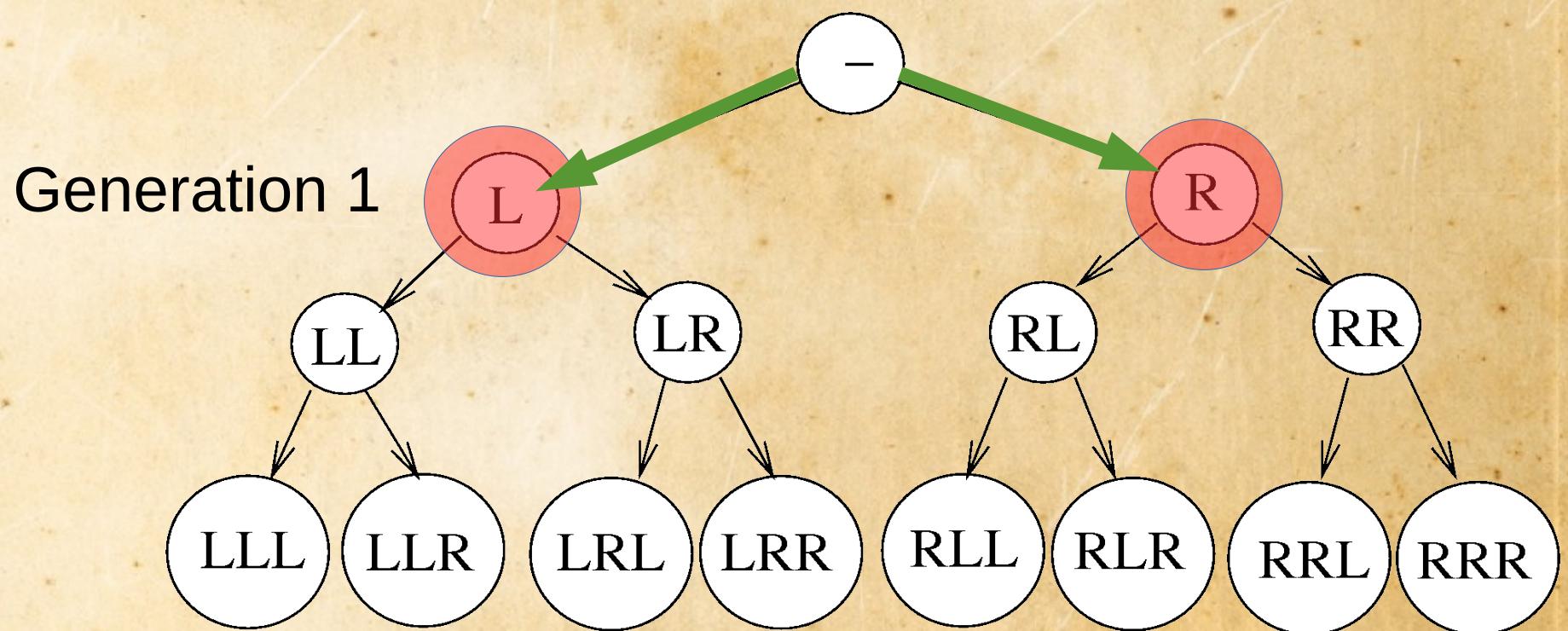
# Beam Search

- Breadth First met Beam Search
  - Zoek verder met alleen de **beste** Beam=3 states



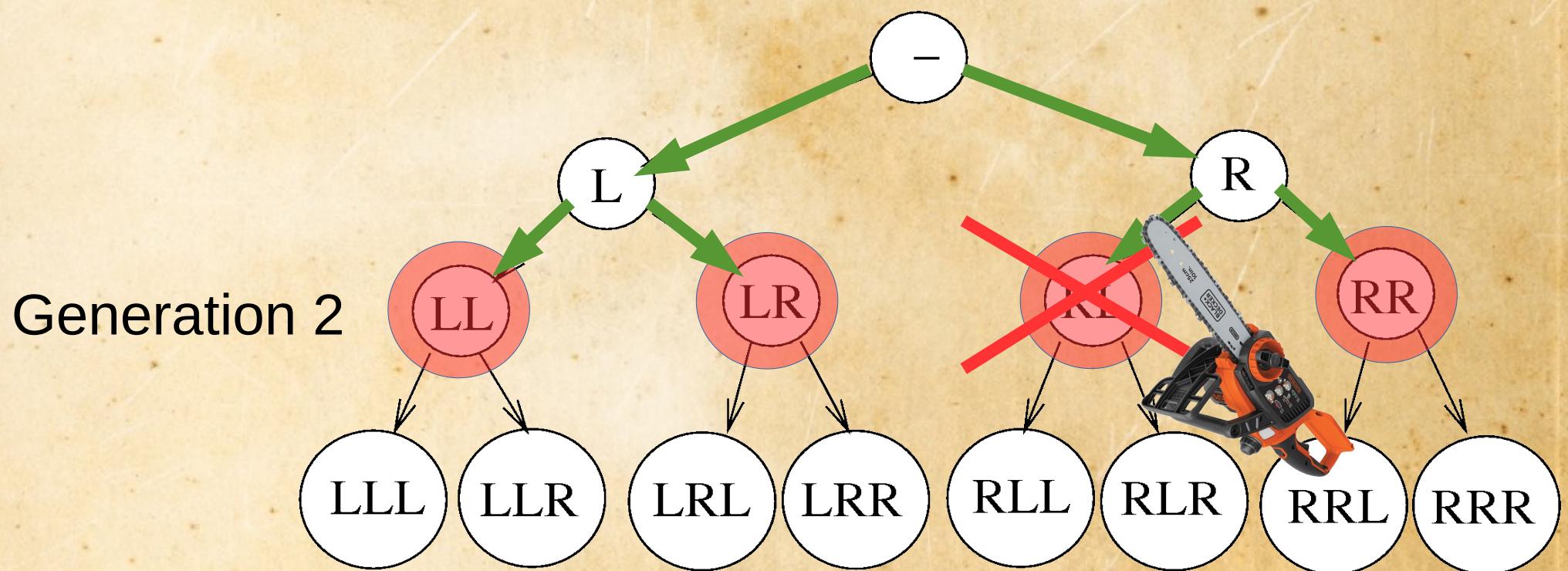
# Beam Search

- Breadth First met Beam Search
  - Zoek verder met alleen de **beste** Beam=3 states



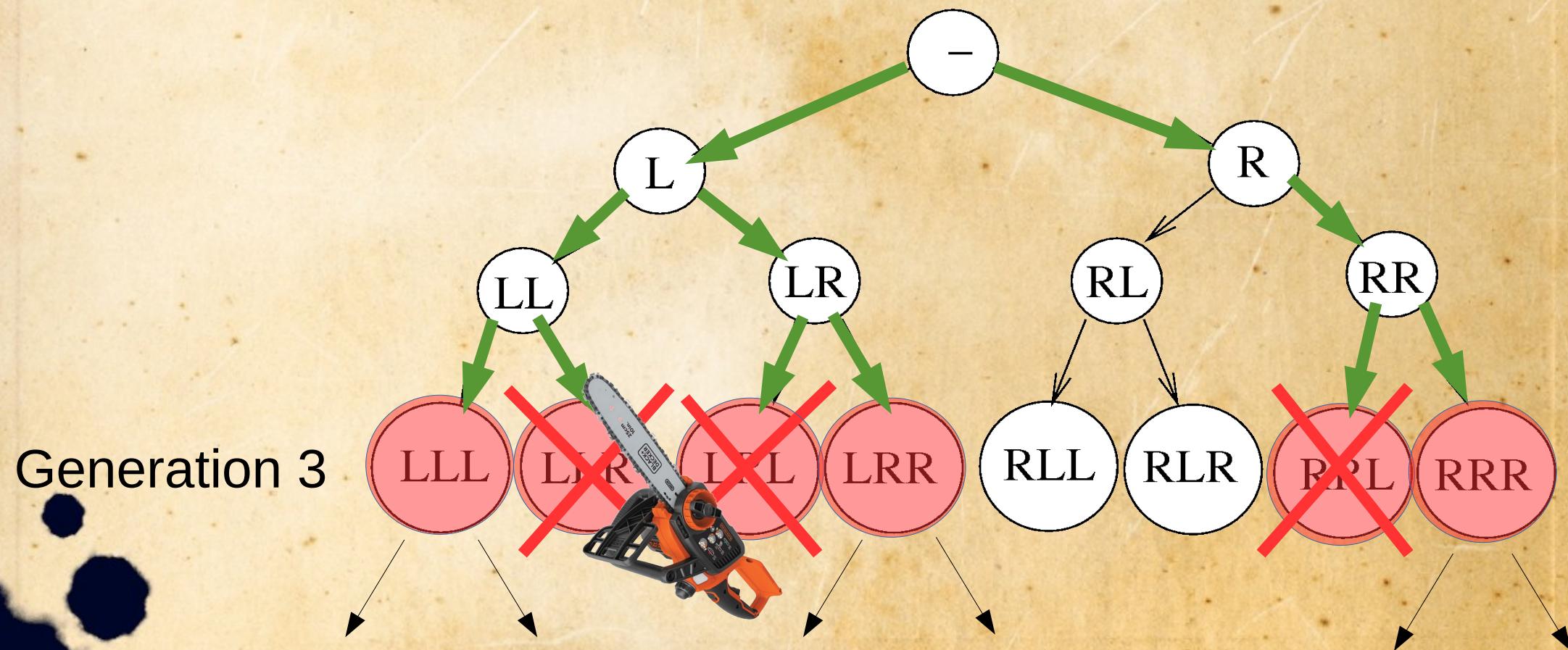
# Beam Search

- Breadth First met Beam Search
  - Zoek verder met alleen de **beste** Beam=3 states



# Beam Search

- Breadth First met Beam Search
  - Zoek verder met alleen de **beste** Beam=3 states

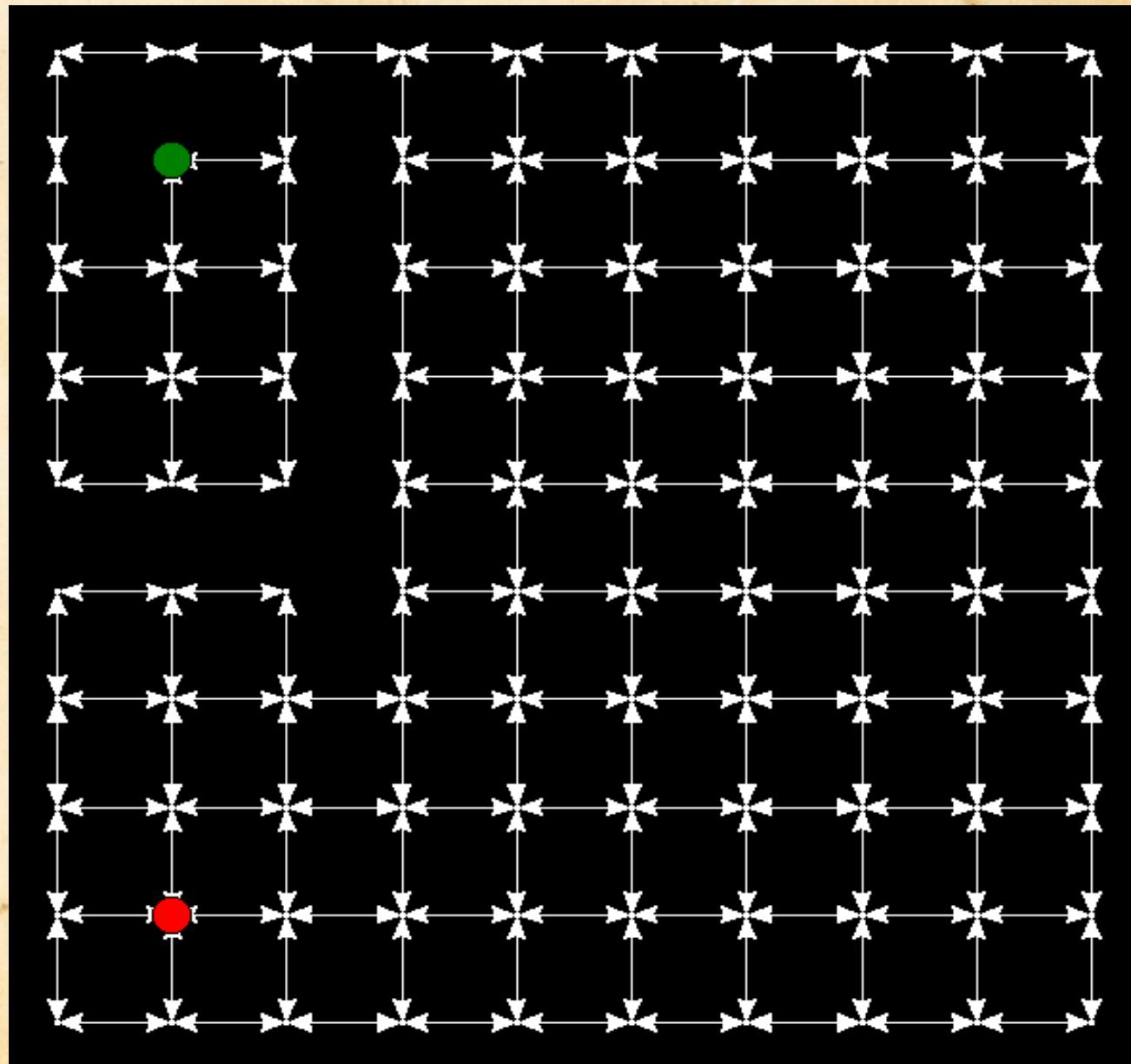


# Beam Search

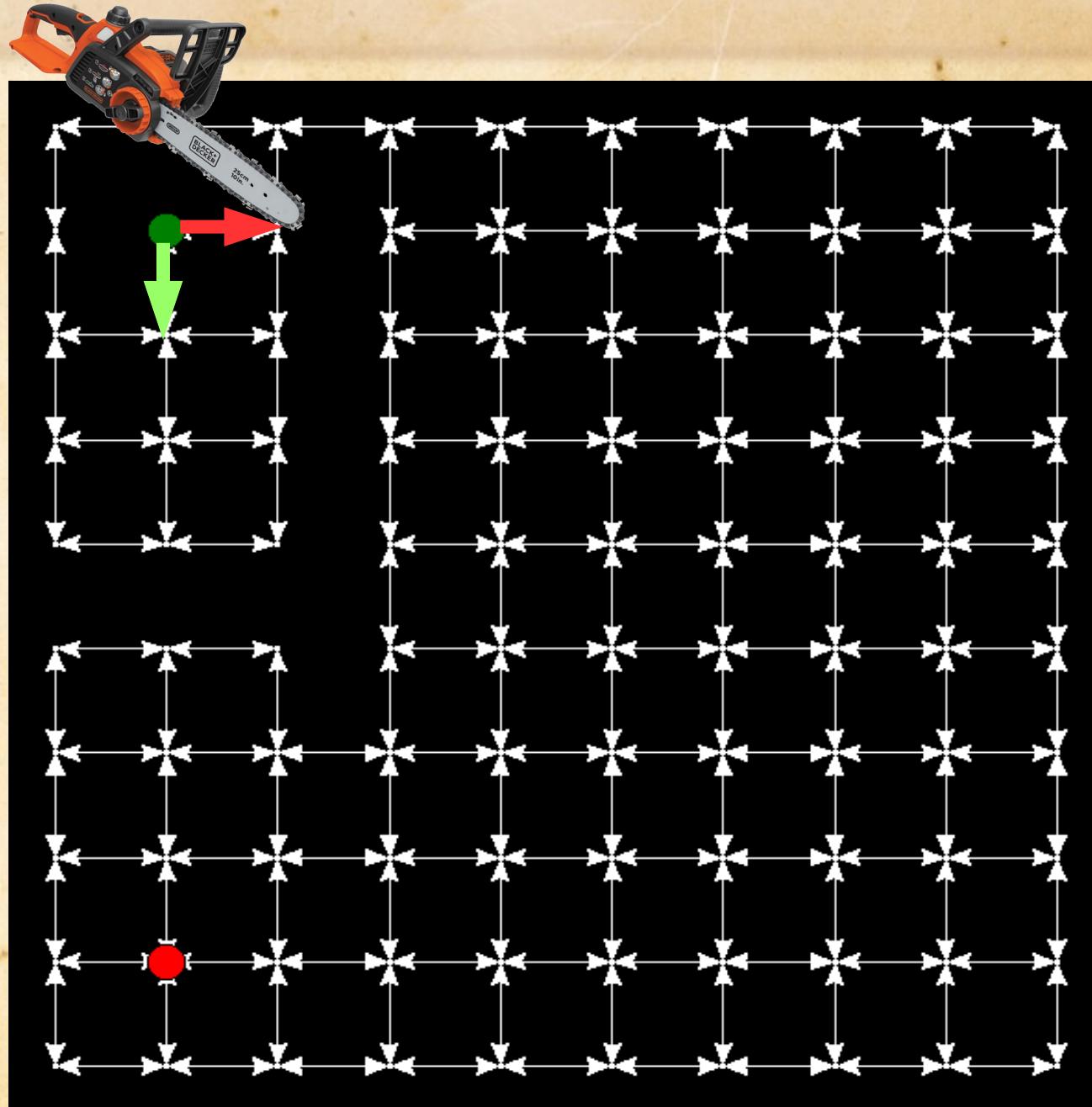
- Hoe groot kiezen we de Beam?
  - Beam=1 state (Greedy)
    - Super snel maar waarschijnlijk slechte oplossing
  - Beam= $\infty$  states (Breadth first)
    - Super traag en geheugen problemen maar optimale oplossing



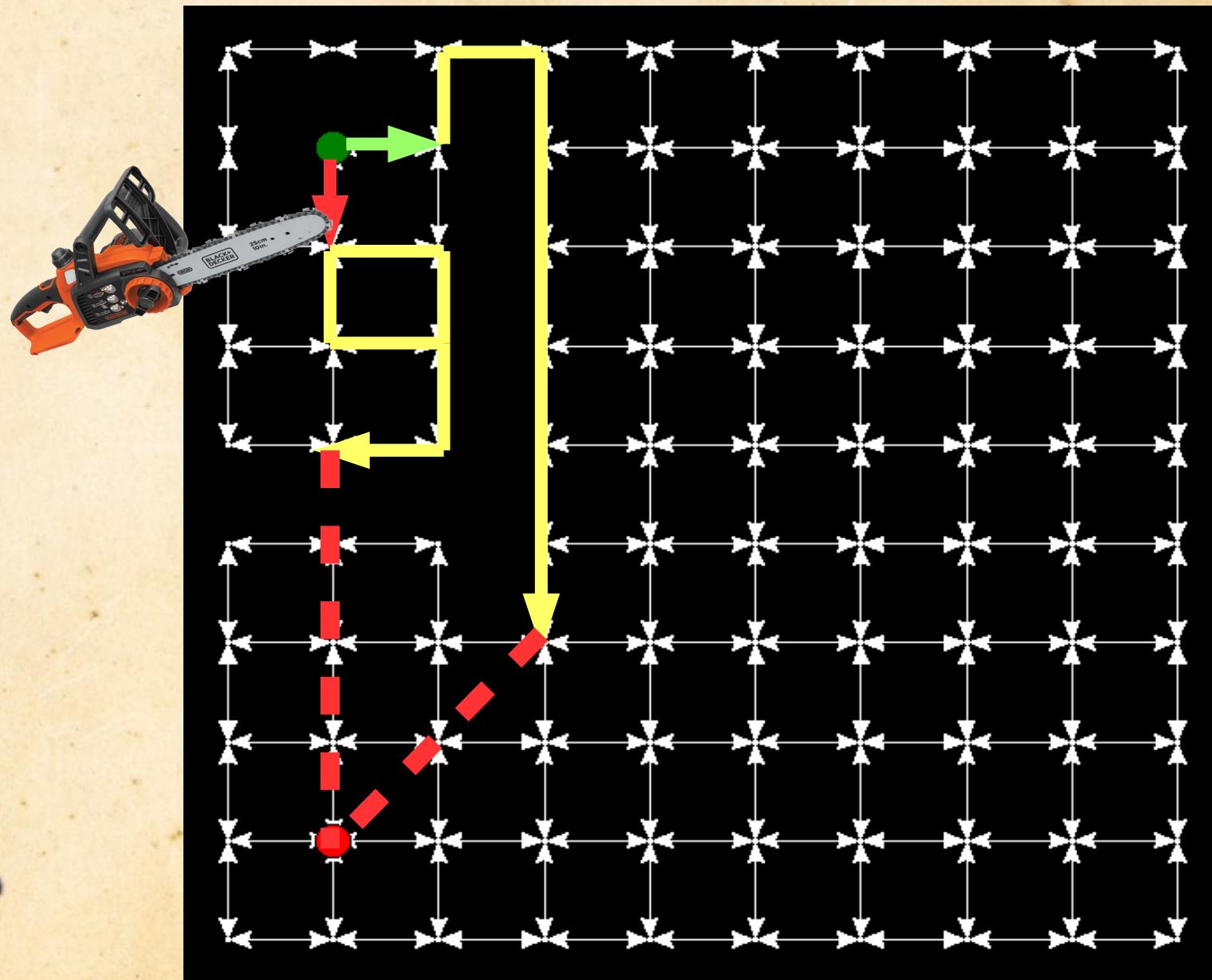
# Greedy without look-ahead



# Greedy without look-ahead



# Greedy with 8 step look-ahead



# Greedy with look-ahead

- Hoe ver look-en we ahead?
  - Steps=0 stappen (Greedy)
    - Super snel maar waarschijnlijk slechte oplossing
  - Steps= $\infty$  stappen (Depth first)
    - Super traag maar optimale oplossing



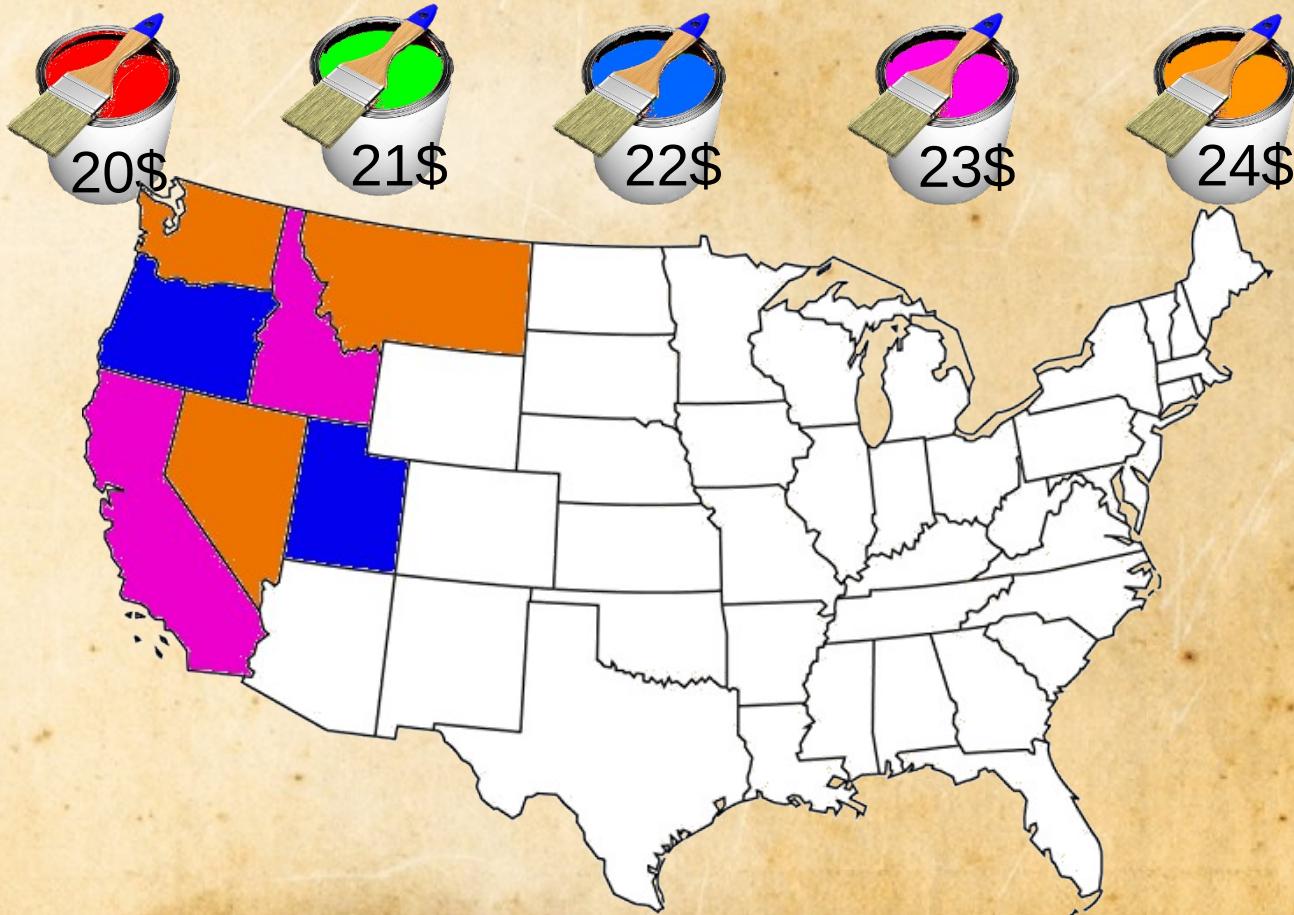
# Combinaties?



- Breadth first + Beam search + look-ahead?
- Depth first + pruning?

# Combinaties?

- Breadth first + Beam search + look-ahead?
- Depth first + pruning?



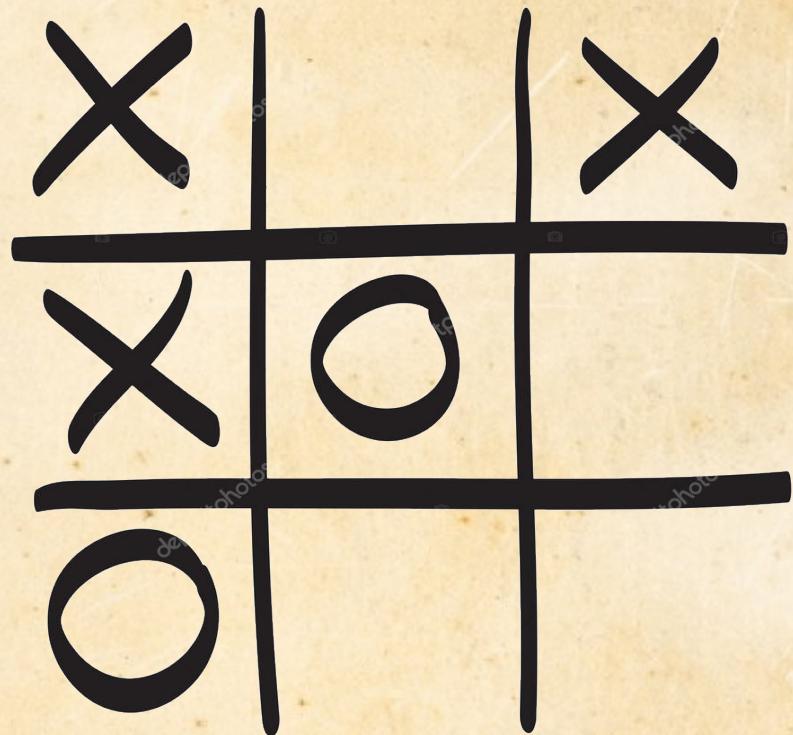
# Heuristieken



- Oud Grieks:
  - *heurískō*: vinden, vergelijken
  - *heurèka*: ik heb het gevonden
- Wikipedia:
  - Is any approach to problem solving or self-discovery that employs a practical method that is not guaranteed to be optimal, perfect or rational, but which is nevertheless sufficient for reaching an immediate, short-term goal.
- Bas:
  - Praktische vuistregel waarmee je (sneller) tot een (betere maar) (misschien) niet optimale oplossingen komt.

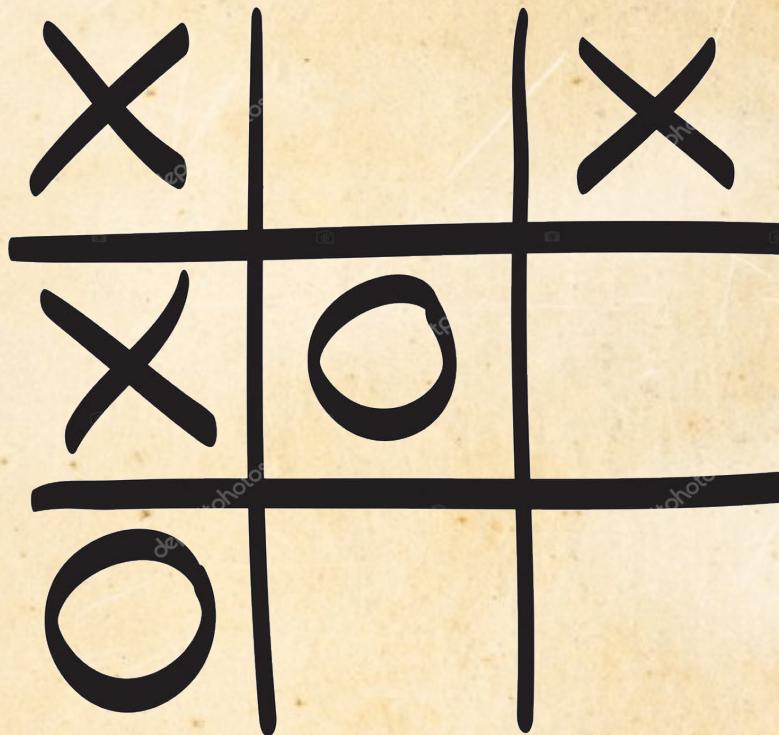
# Heuristieken

- Spelletjes: eerste zet doe ik in het midden



# Heuristieken

- Spelletjes: eerste zet doe ik in het midden



A Knowledge-based Approach of Connect-Four  
The Game is Solved: White Wins  
*Victor Allis, Vrije Universiteit, 1988*

Numberphile:  
<https://www.youtube.com/watch?v=yDWPi1pZOPo>



# Heuristieken

- Spelletjes: hoe goed is een positie?

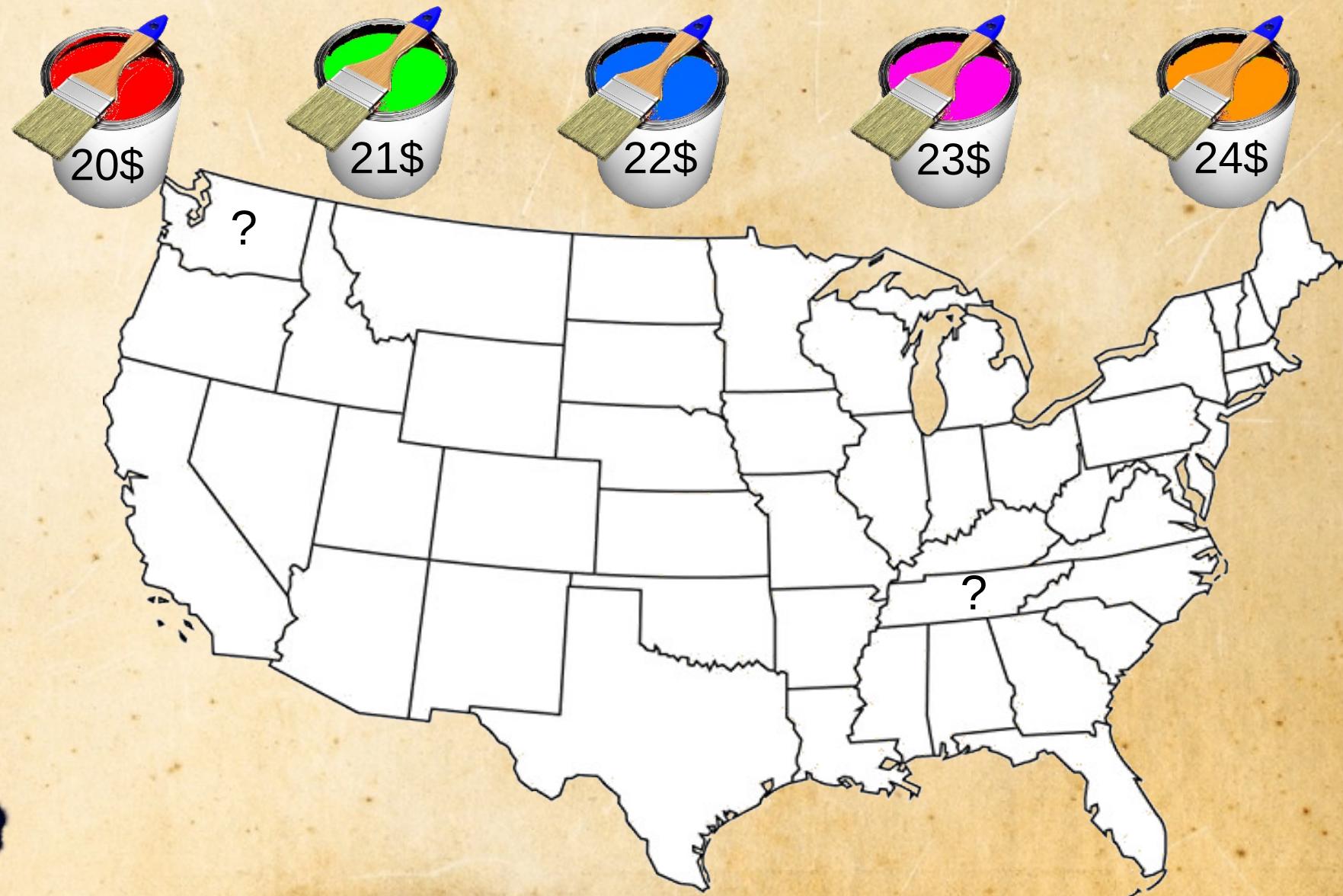


# Heuristieken

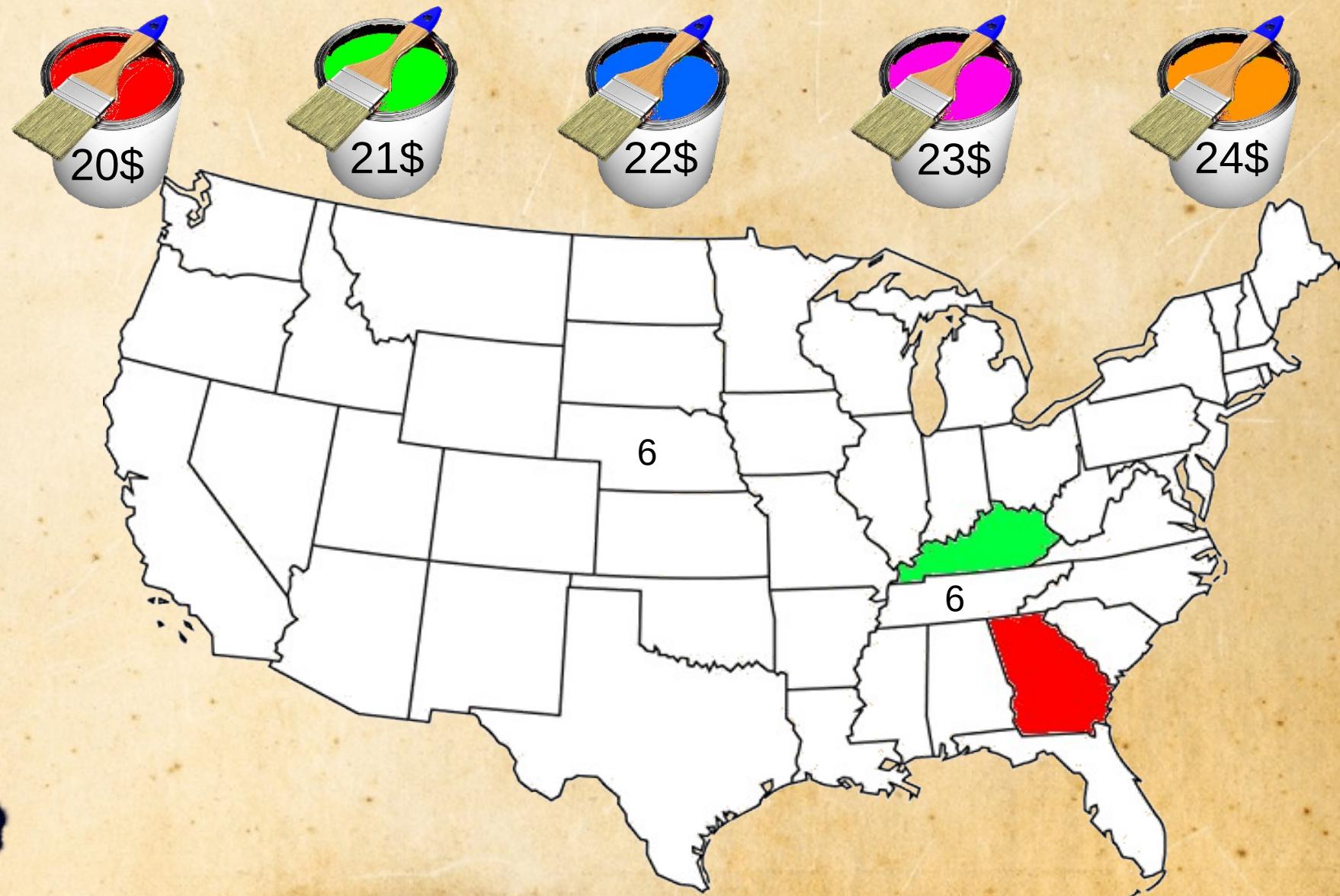
- Inpakken: grootste spullen eerst



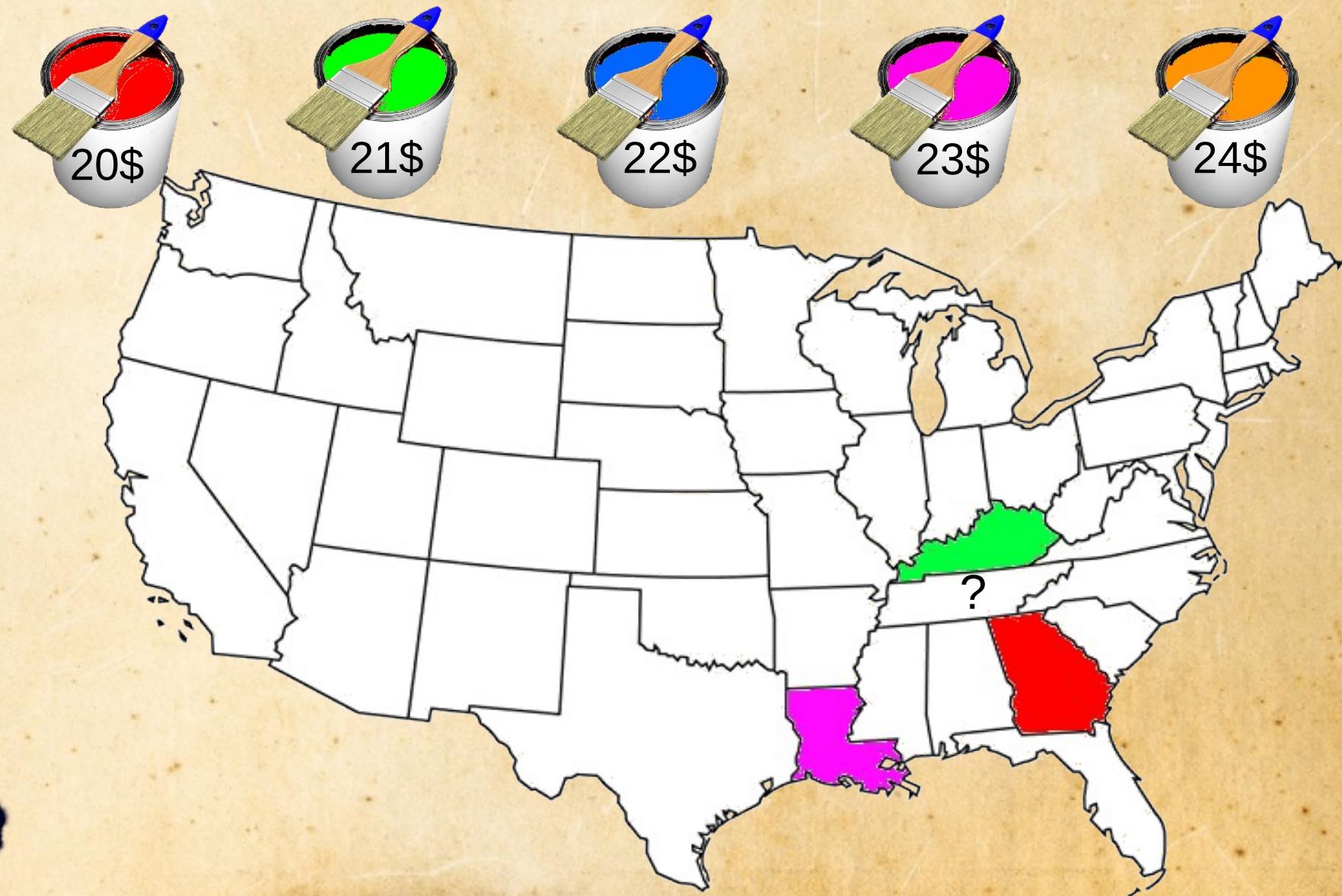
# Kaart kleuren



# Kaart kleuren



# Kaart kleuren

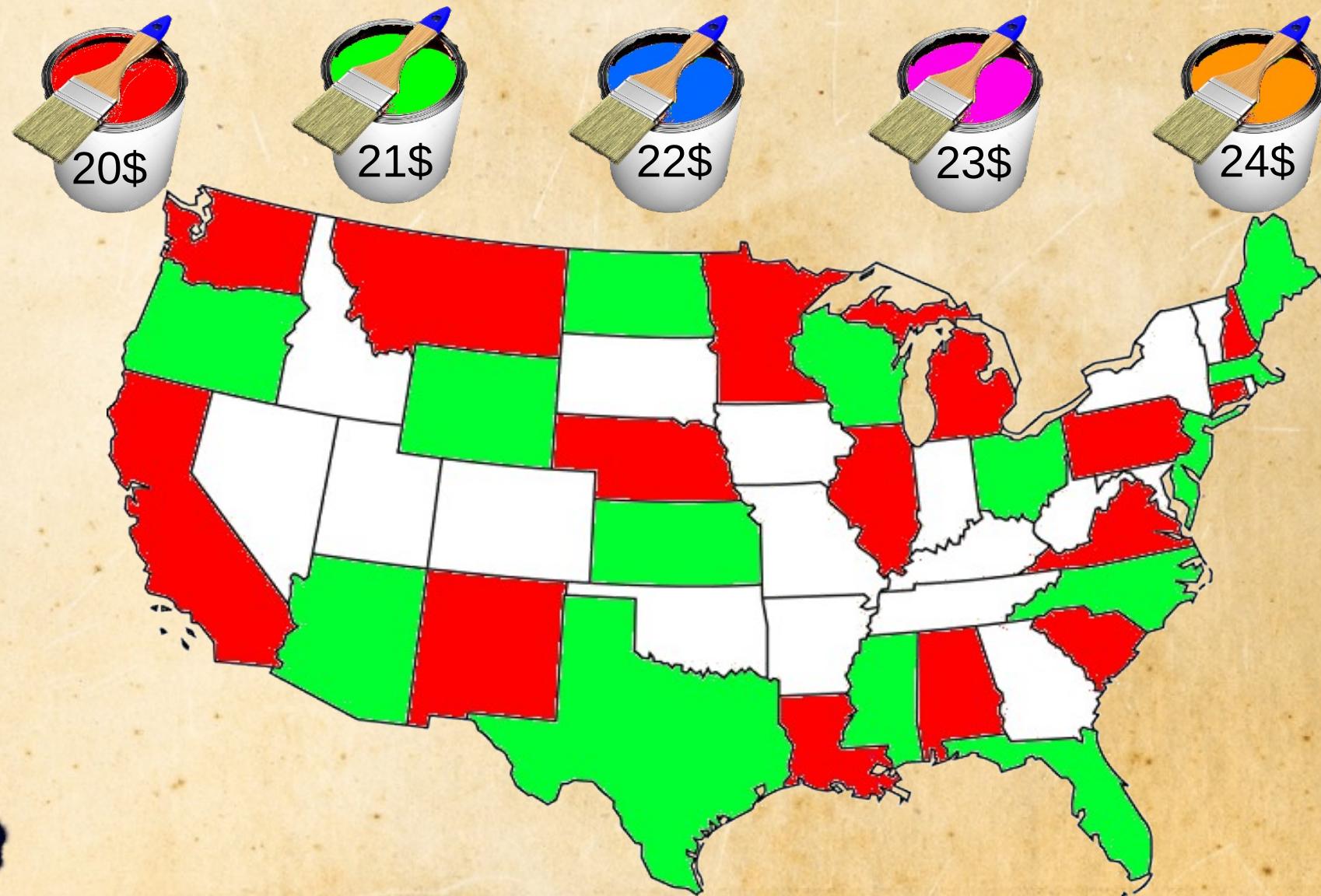


# Heuristiek

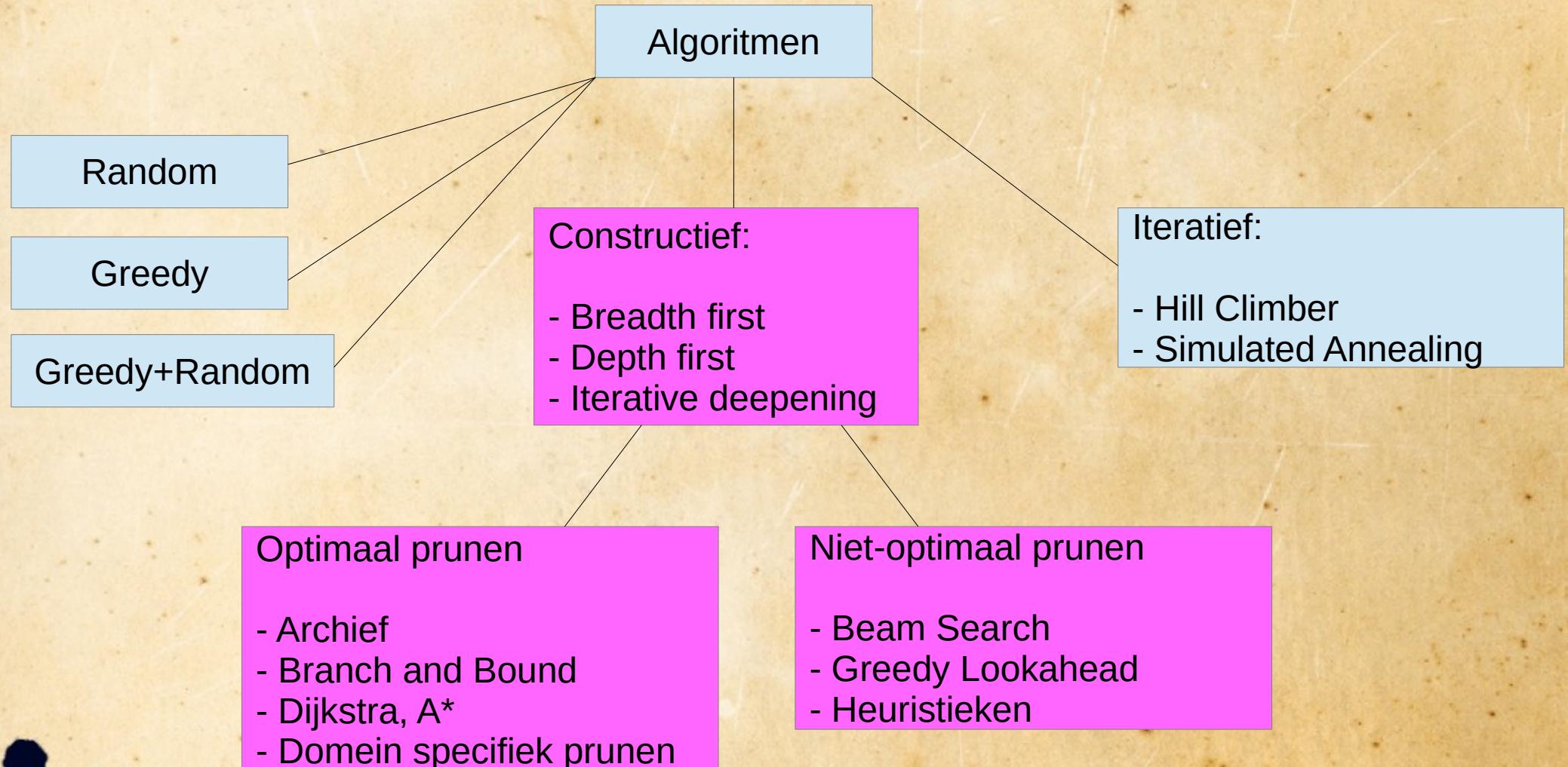


- Most Constraint Variable (MCV)
  - Meeste constraints met andere variabele
  - Minst aantal mogelijke waarden over
- Least Constraining Value (LCV)

# Heuristiek: goedkoopste eerst?



# Algoritmen



# Literatuur

## Artificial Intelligence: A Modern Approach

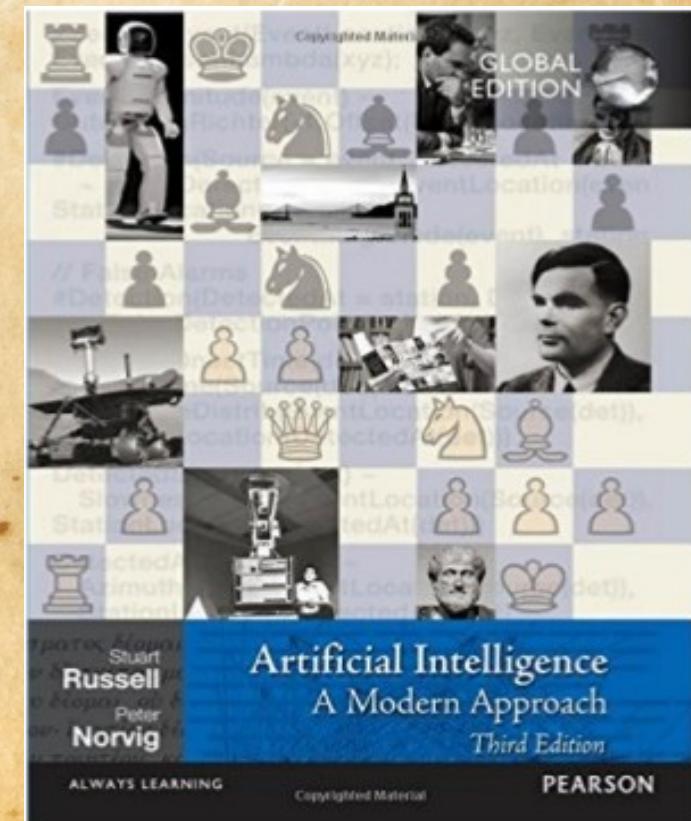
Third Edition

Stuart J. Russell and Peter Norvig

prgrf 3.4 Breadth first, Depth first

prgrf 3.5 A\*

prgrf 6.3 Most Constraint Variable heuristic

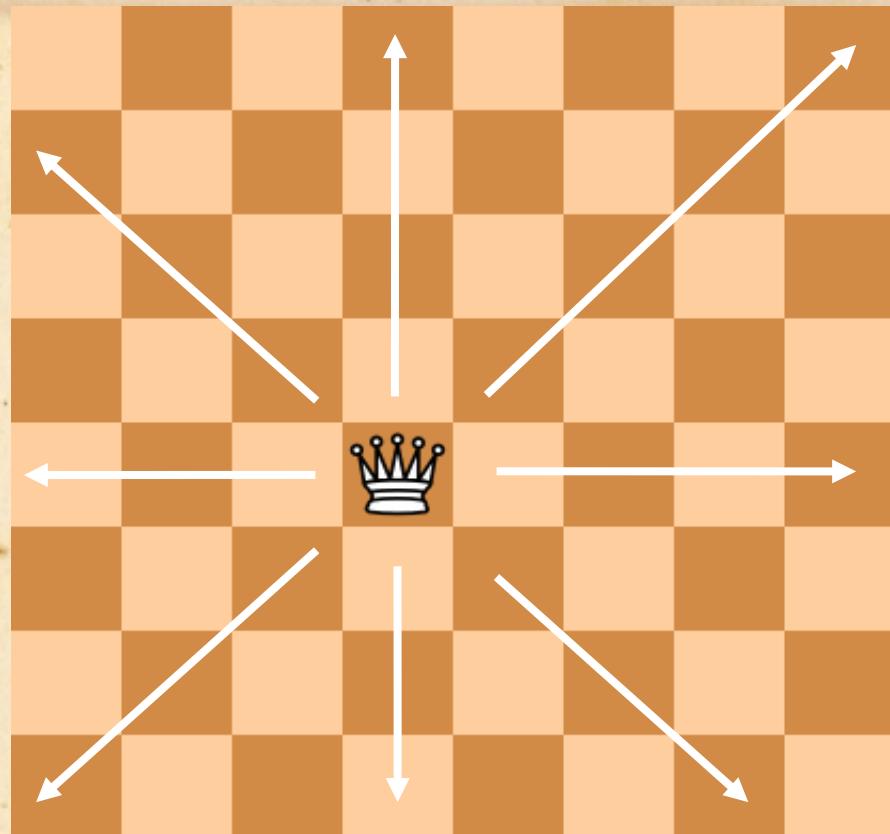


# Lecture video 2020, terugkijken



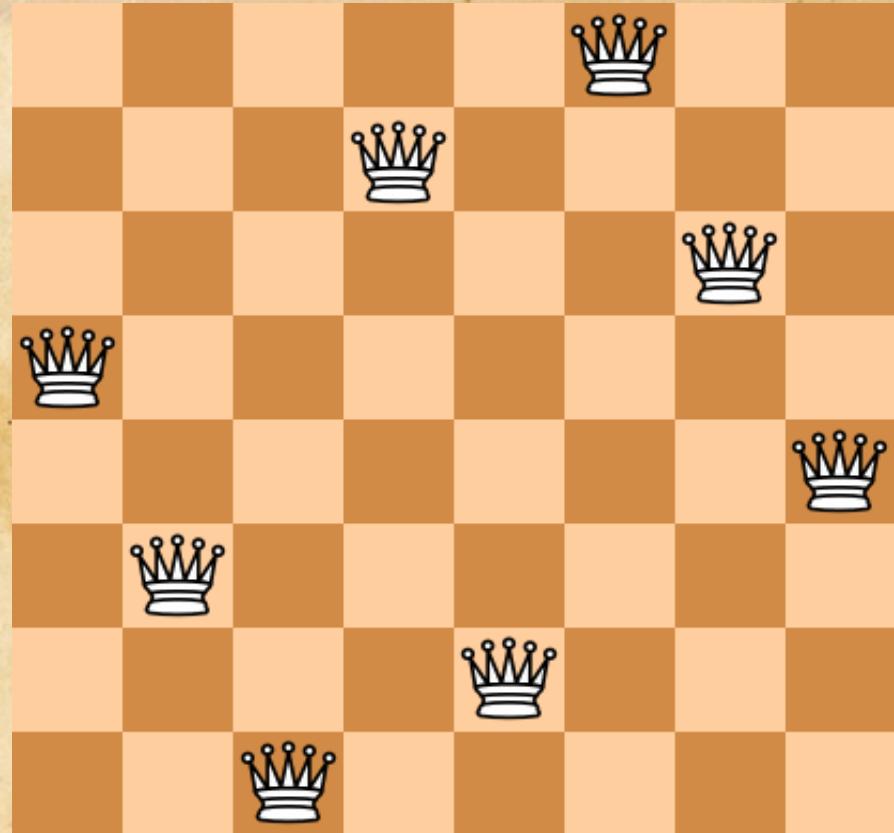
- Youtube playlist, Constructieve algoritme
  - <https://www.youtube.com/playlist?list=PLJBtJTYGPSzIfEzXpszM8Ewsllwfa0d6T>

# Toetje: N-Queens



State space?

Order: no Repetition: no



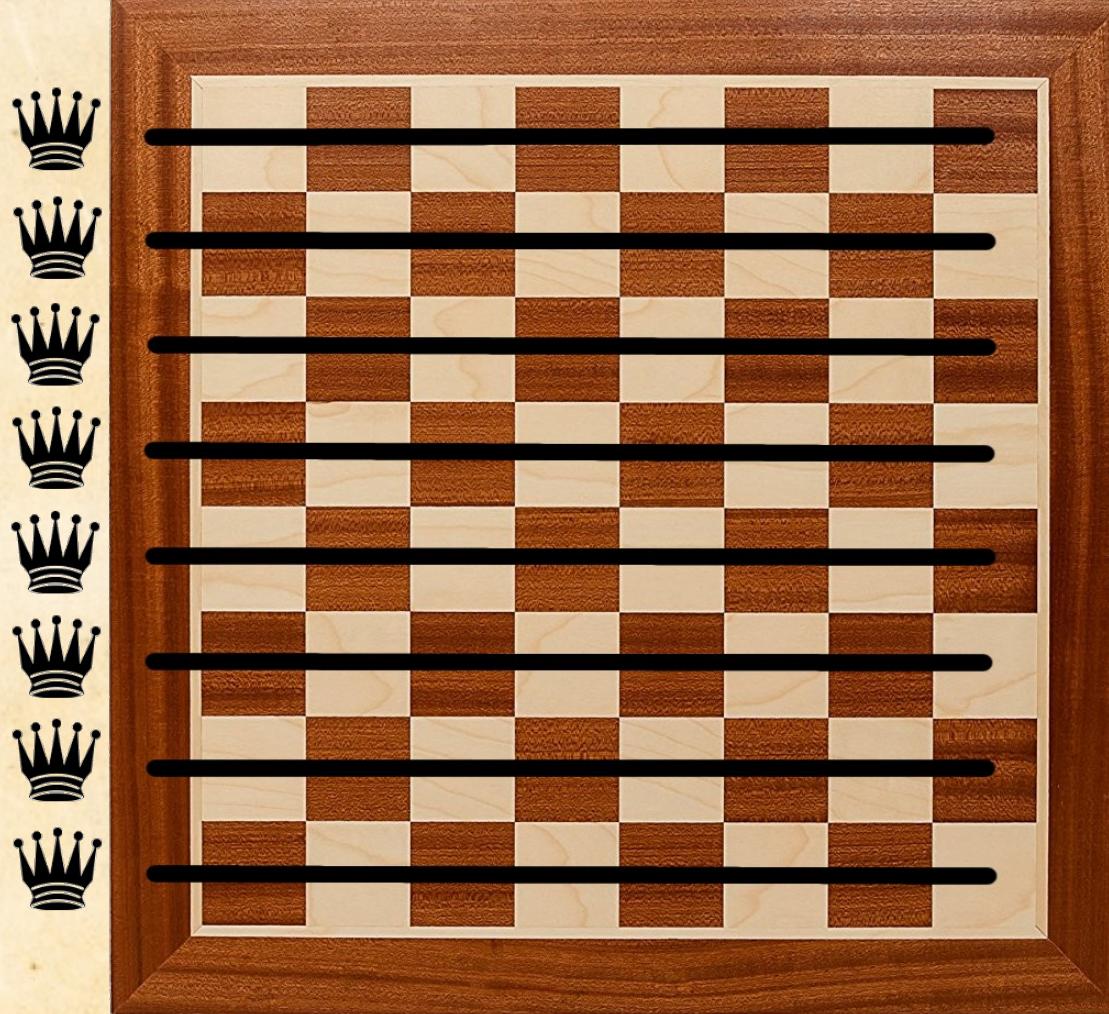
$$\frac{(N \times N)!}{(N! \times (N \times N - N)!)}$$

$$N=8: \frac{(8 \times 8)!}{(8! \times (8 \times 8 - 8)!) } = \frac{64 \times 63 \times 62 \times 61 \times 60 \times 59 \times 58 \times 57}{8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1} = 4426165368$$

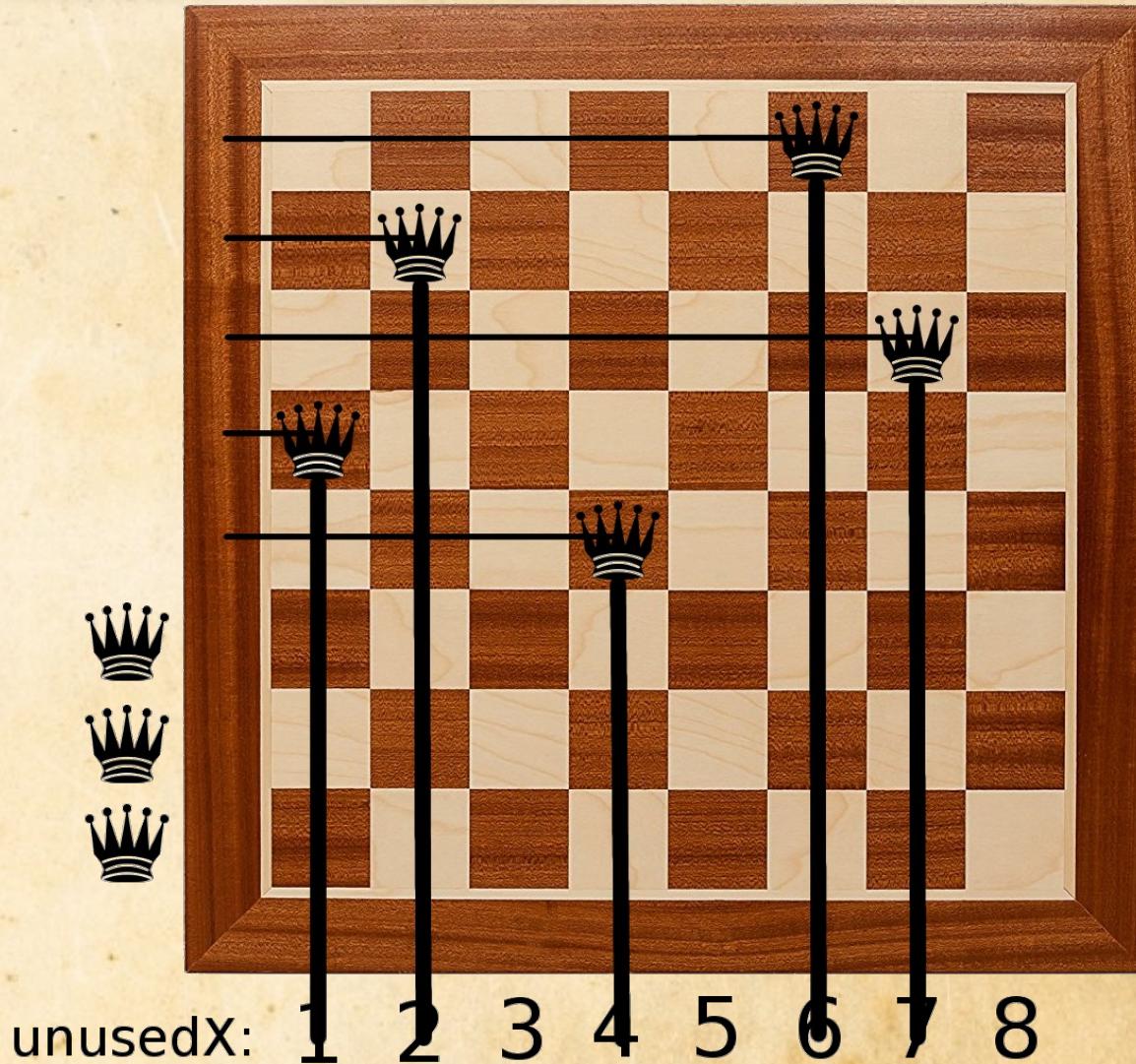
# N-Queens

| Queens | Squares | State Space                         | Approximate Time to Solve         |
|--------|---------|-------------------------------------|-----------------------------------|
| 8      | 64      | 4426165368                          | 4.42 Seconds                      |
| 9      | 81      | 260887834350                        | 4.34 Minutes                      |
| 10     | 100     | 17310309456440                      | 4.8 Hours                         |
| 11     | 121     | 1276749965026540                    | 14.77 Days                        |
| 12     | 144     | 103619293824707000                  | 39.40 Months                      |
| 13     | 169     | 9176358300744340000                 | 290.78 Years                      |
| 14     | 196     | 880530516383349000000               | 27 902.92 Years                   |
| 15     | 225     | 9100556781177500000000              | 2 883 854.02 Years                |
| 16     | 256     | 100787516020223000000000000         | 319 383 187 Years                 |
| 17     | 289     | 11907390443444900000000000000       | $3.77330493 \times 10^{10}$ Years |
| 18     | 324     | 1494824923341950000000000000000     | $4.73691552 \times 10^{12}$ Years |
| 19     | 361     | 198708670535438000000000000000000   | $6.29683229 \times 10^{14}$ Years |
| 20     | 400     | 27883609836709000000000000000000000 | $8.83597149 \times 10^{16}$ Years |

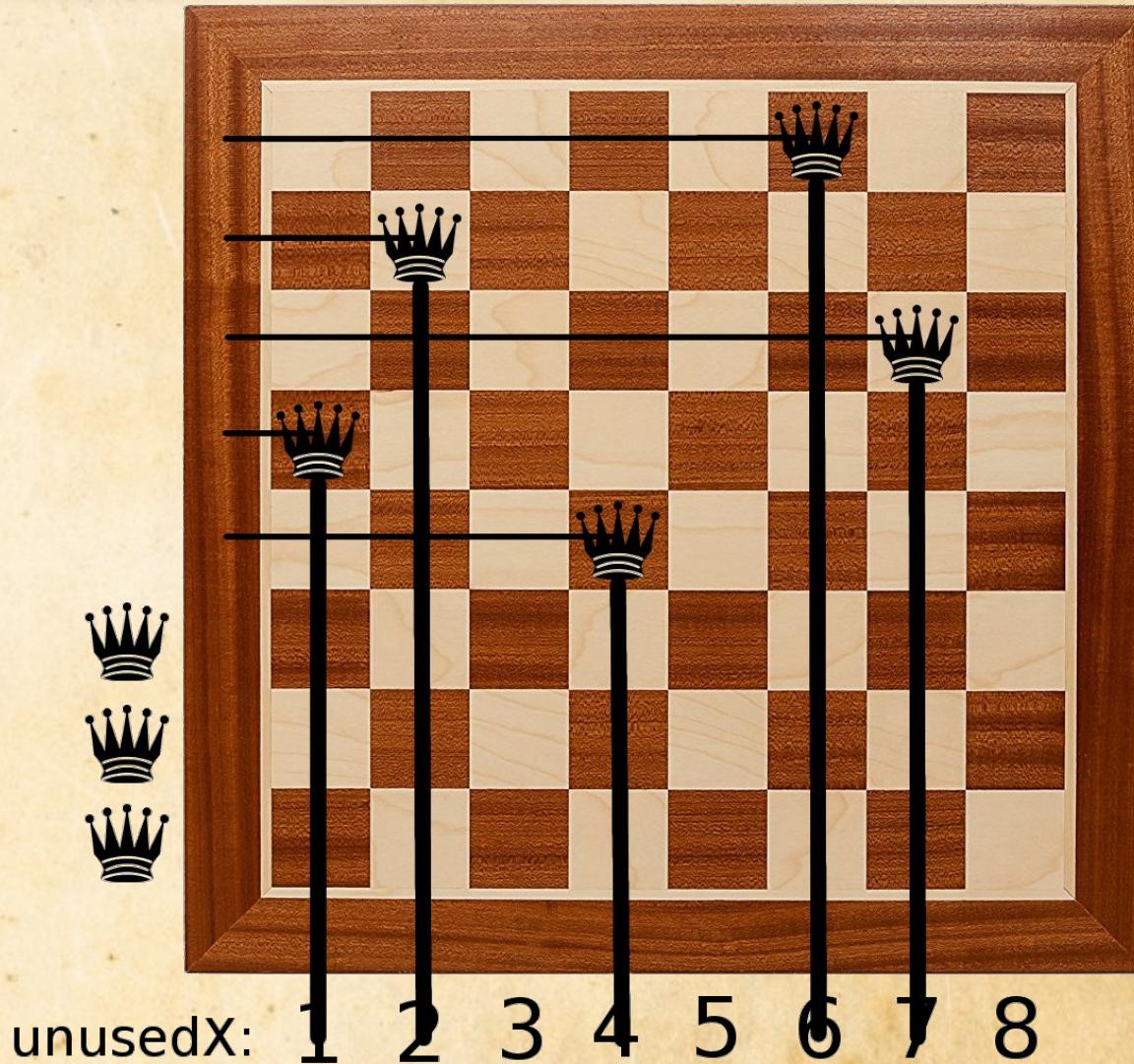
# N-Queens, representatie



# N-Queens, representatie



# N-Queens, representatie

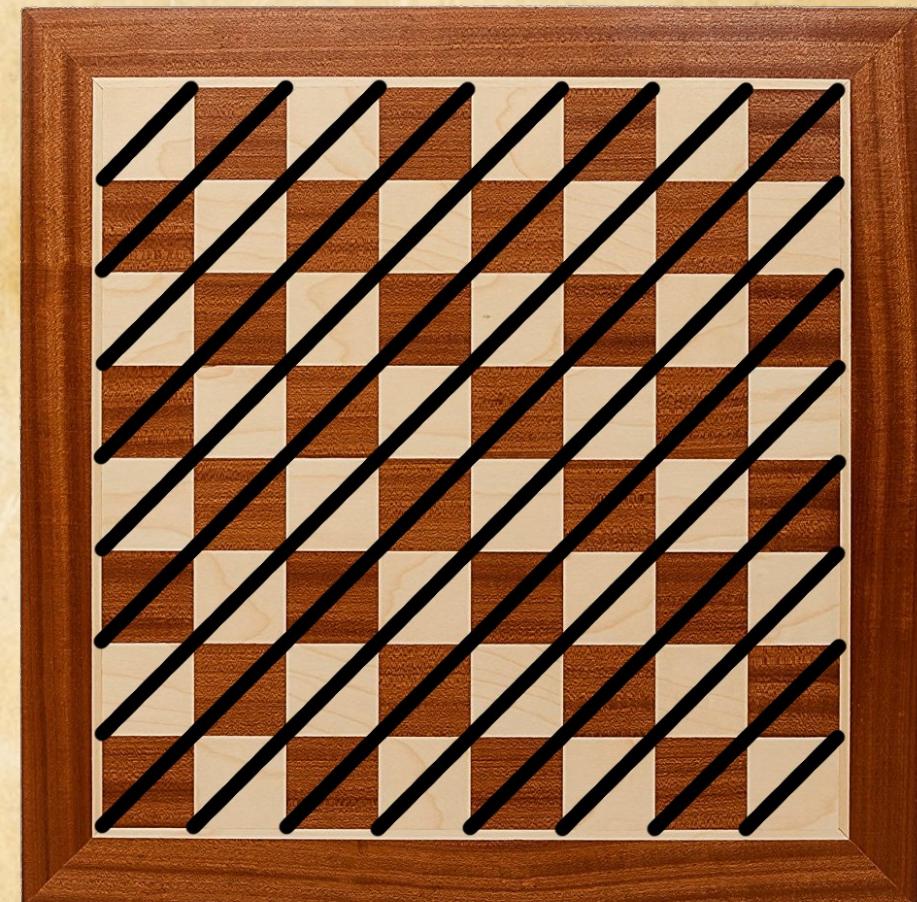
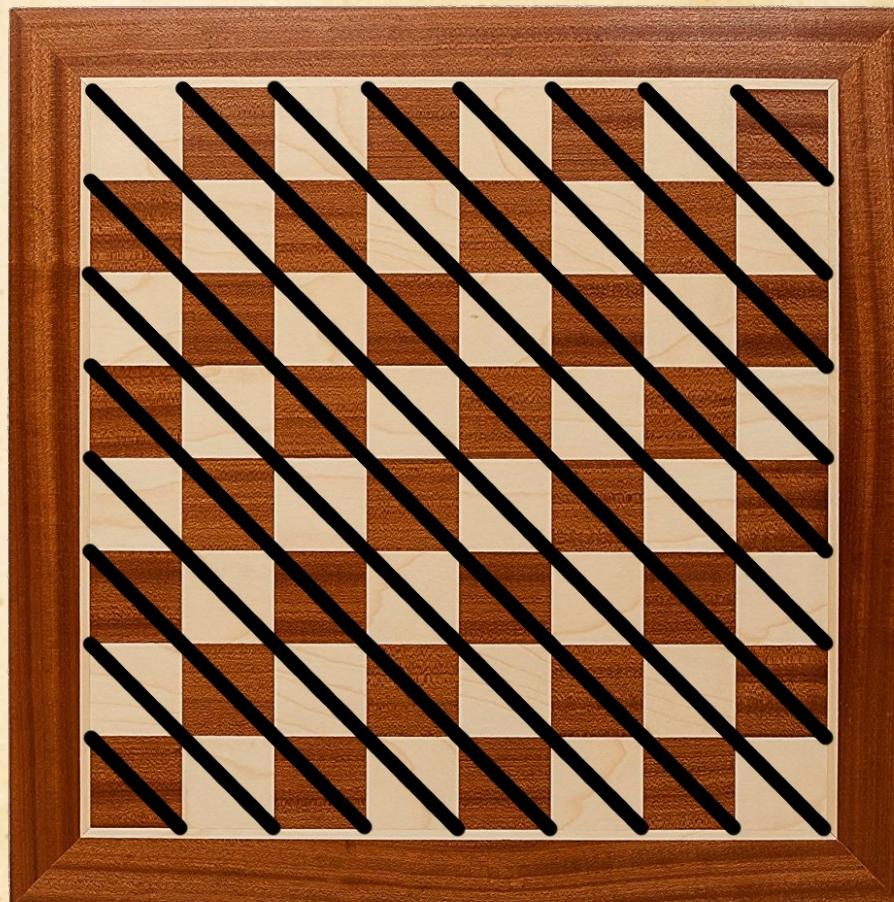


State space:  $N!$  (symmetrie?)

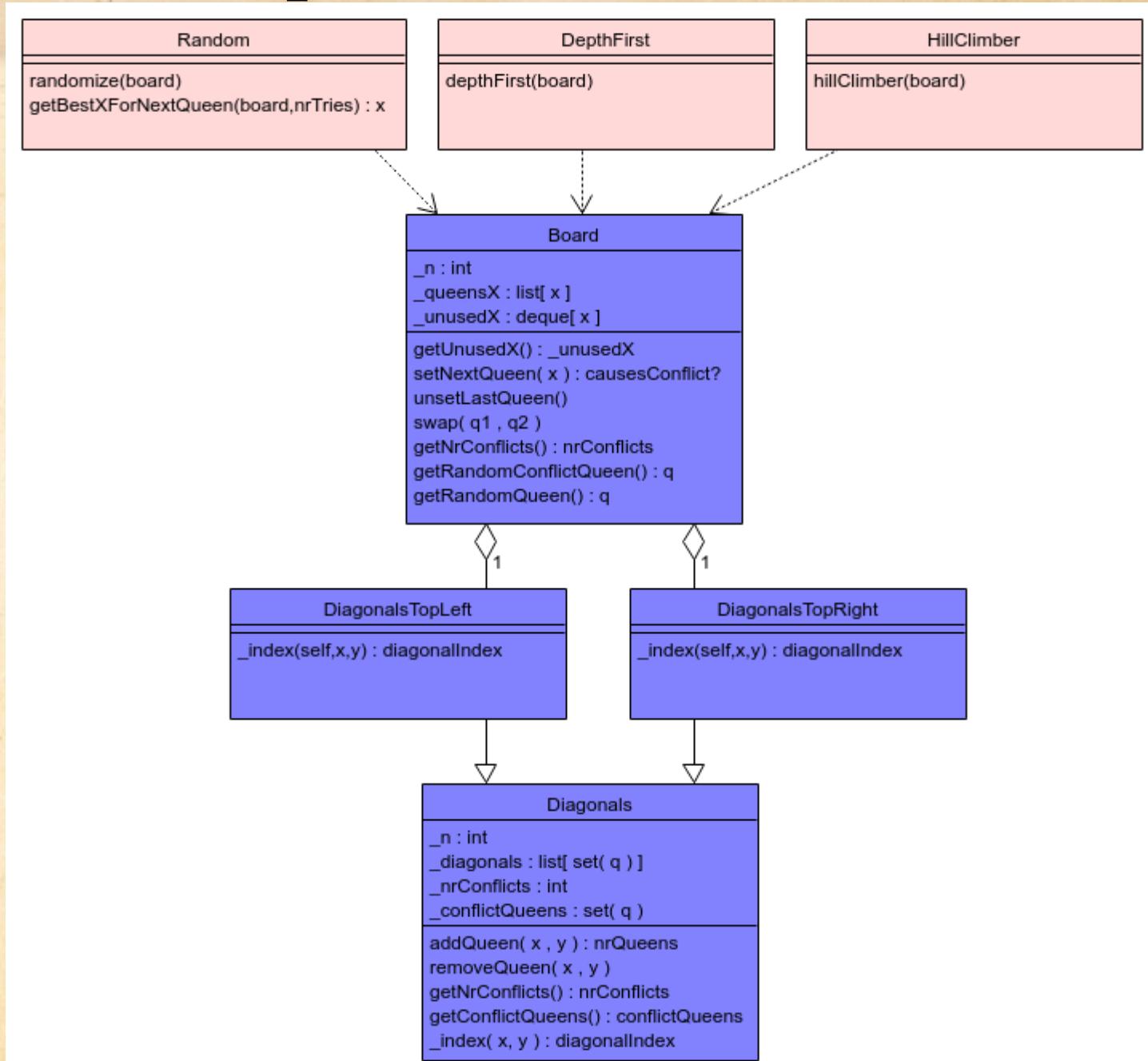
# N-Queens, representatie



Alleen nog conflicten op de  $N+N-1$  'Top-Left' en  $N+N-1$  'Top-Right' diagonalen



# N-Queens, representatie



# N-Queens, Demo



GitHub: <https://github.com/bterwijn/NQueens>