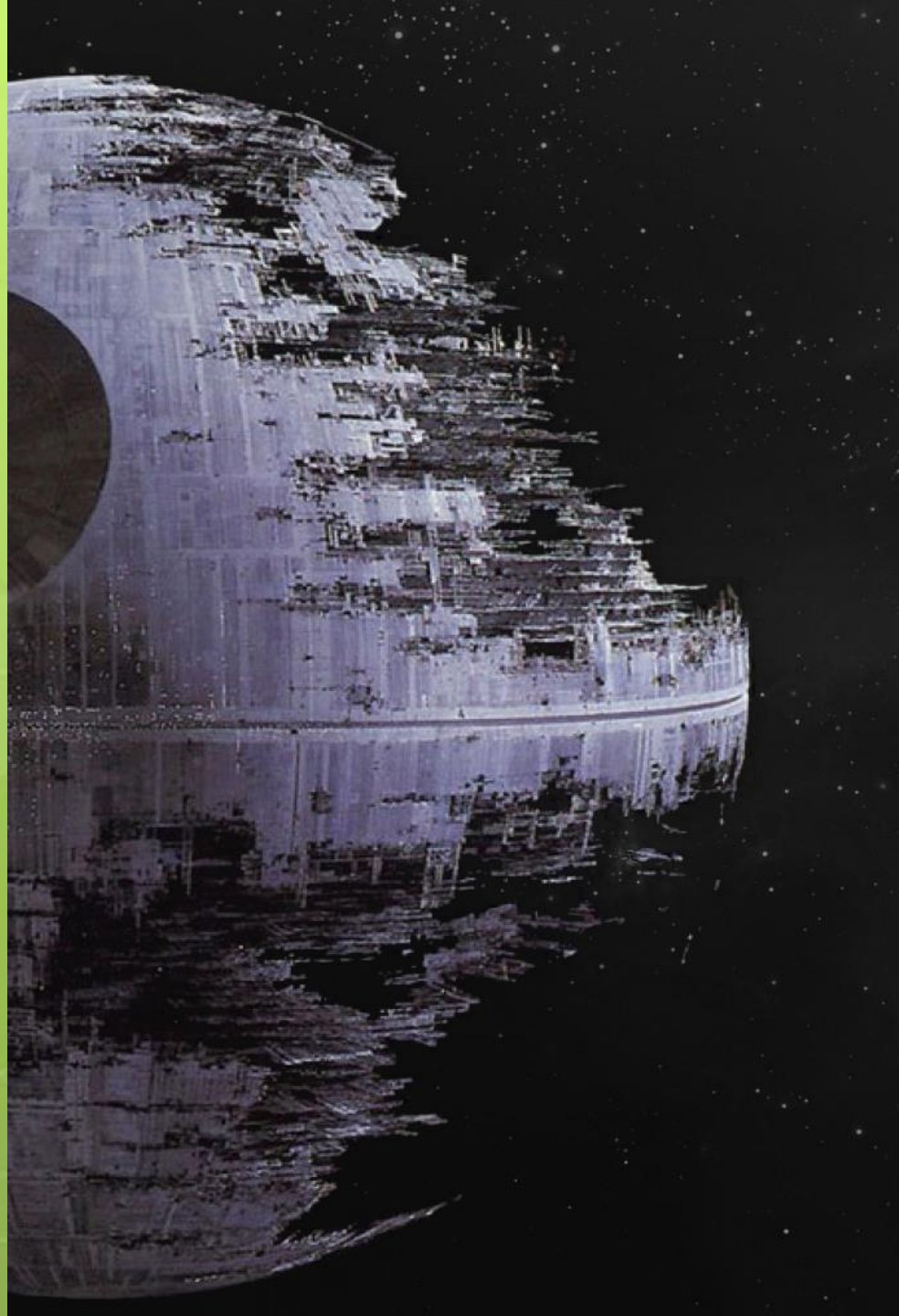
# Constructieve Algoritmes

## Buildings in state space

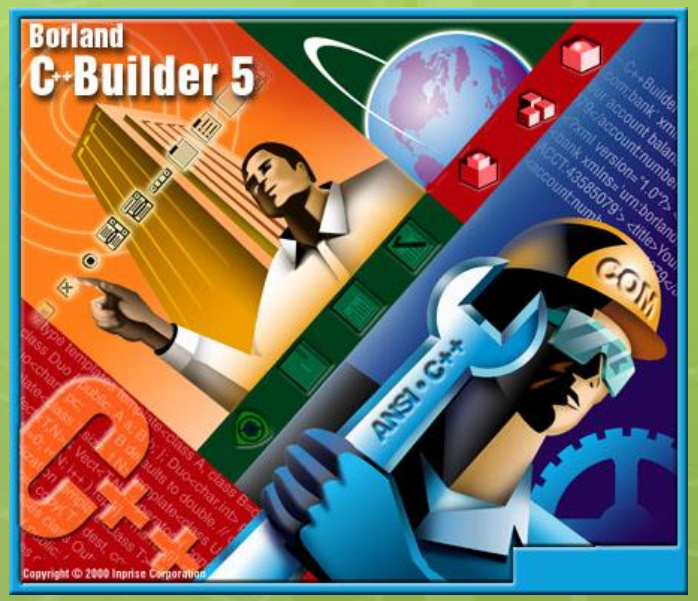


Daan van den Berg
Editie 0.99c, November 2015, MinProg, UvA

# Constructieve Algoritmes

## De basics

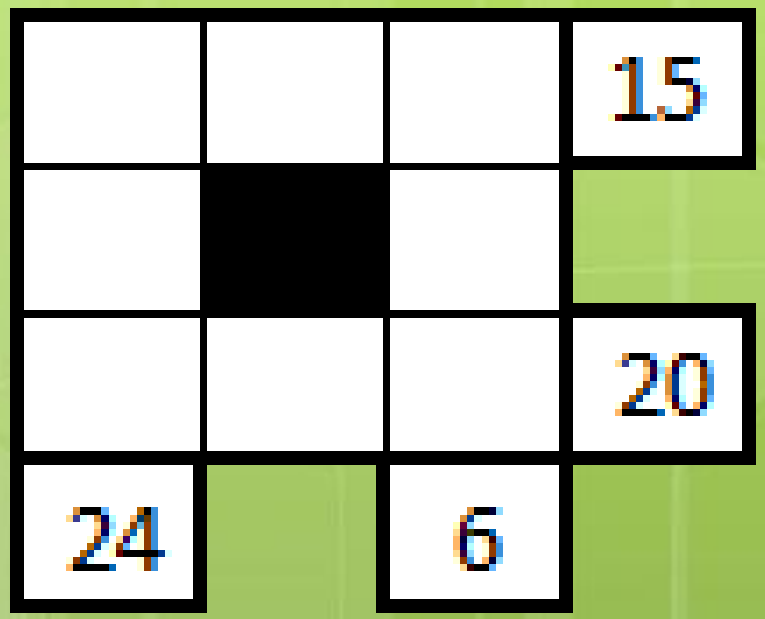
- Stap-voor-stap naar een oplossing Kan het ook anders?
- “In de breedte of in de diepte”
- Met een queue of met een stack
- Brute-force en *compleet* for now



# Constructieve Algoritmes

## De basics

- (mini)kakuro
- Wat is exact de vraag?

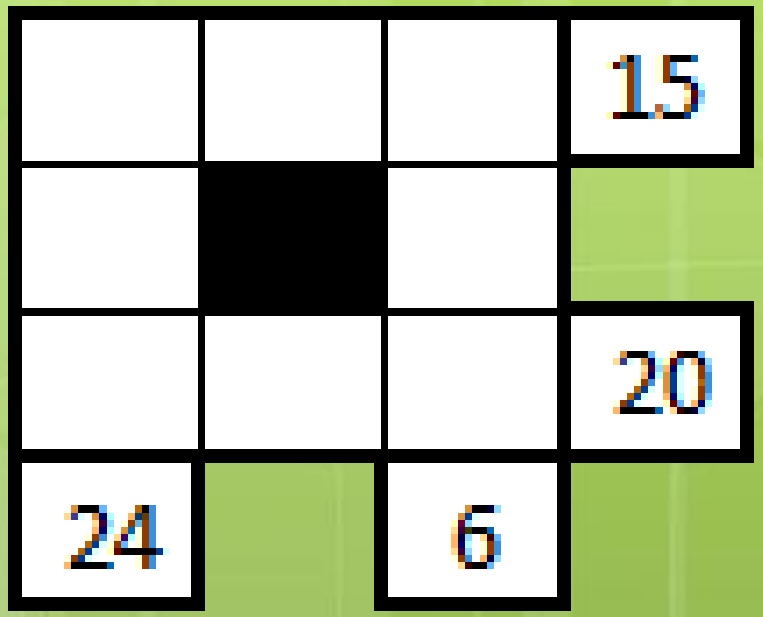



*小カックロ*

# Constructieve Algoritmes

## De basics

- (mini)kakuro
- Wat is **exact** de vraag?



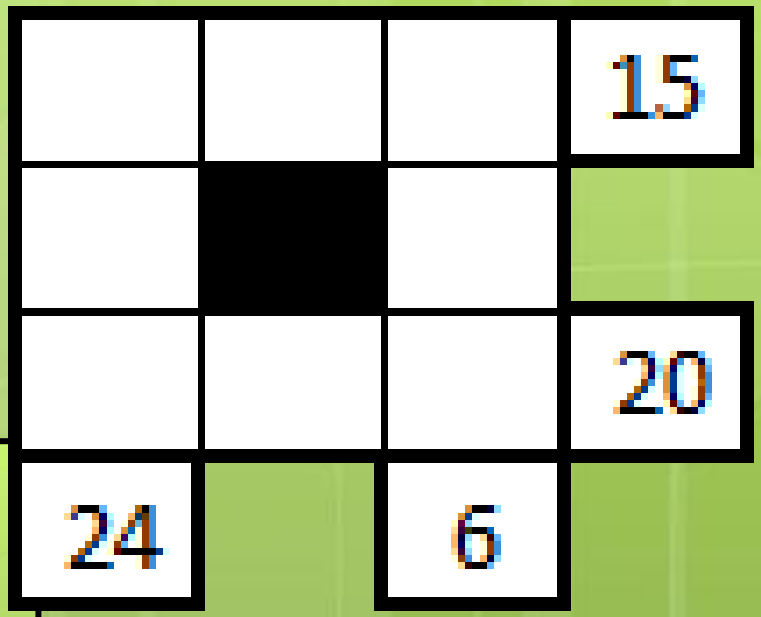

小カックロ

"Is er een oplossing?"
"Wat is de oplossing?"
"Wat is een oplossing?"
"Wat is de beste oplossing?"

# Constructieve Algoritmes

## De basics

- (mini)kakuro
- Wat is **exact** de vraag?




小カックロ

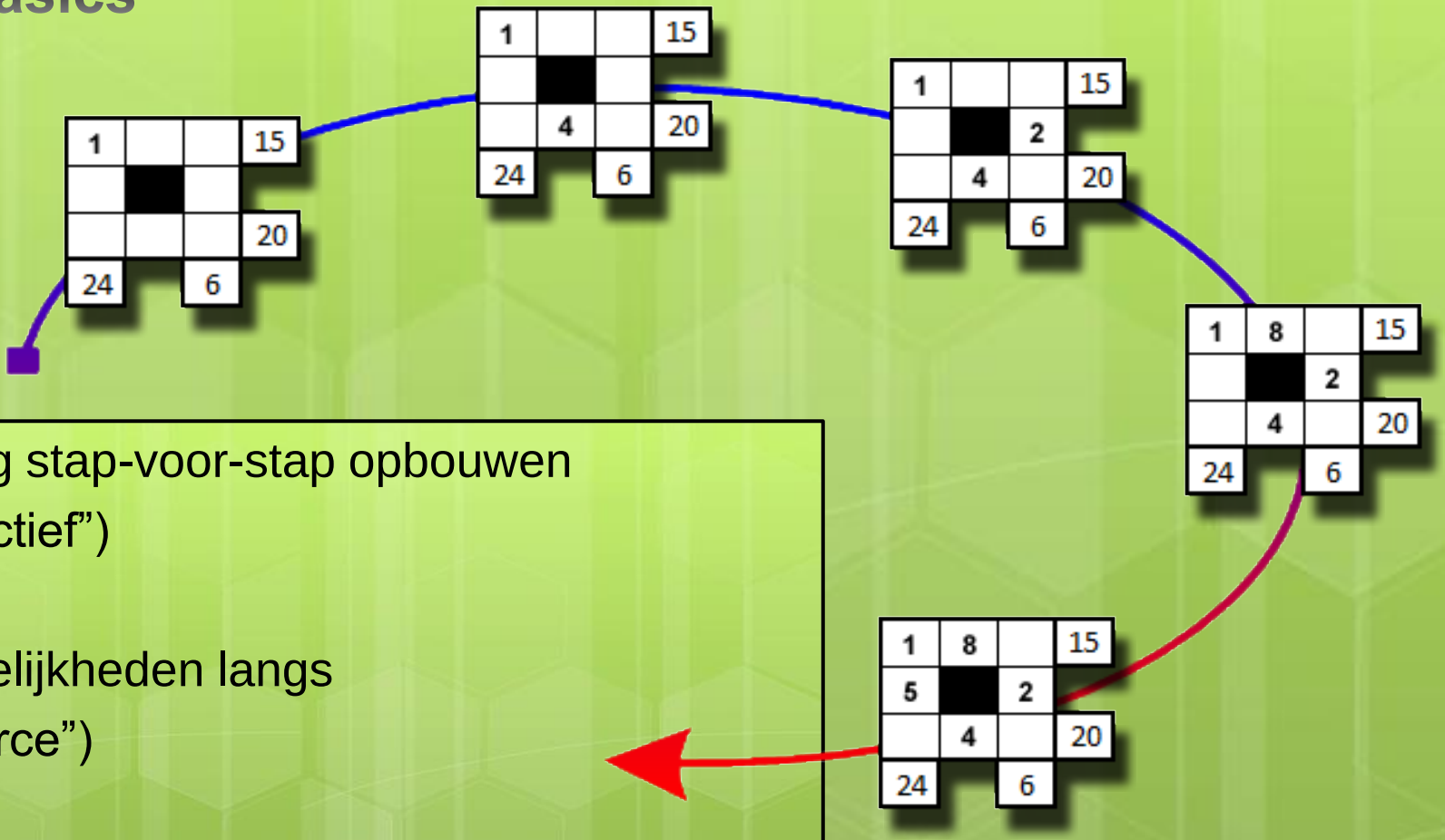
"Vind de oplossing."

(er is er dus maar één, (en ze zijn allemaal even goed ) )

(een oplossing is een invulling van de vakjes, elk met een integer van 1-9, zodat in iedere rij en iedere kolom drie verschillende integers staan en zodat de sommen kloppen)

# Constructieve Algoritmes

## De basics




Oplossing stap-voor-stap opbouwen
(“constructief”)

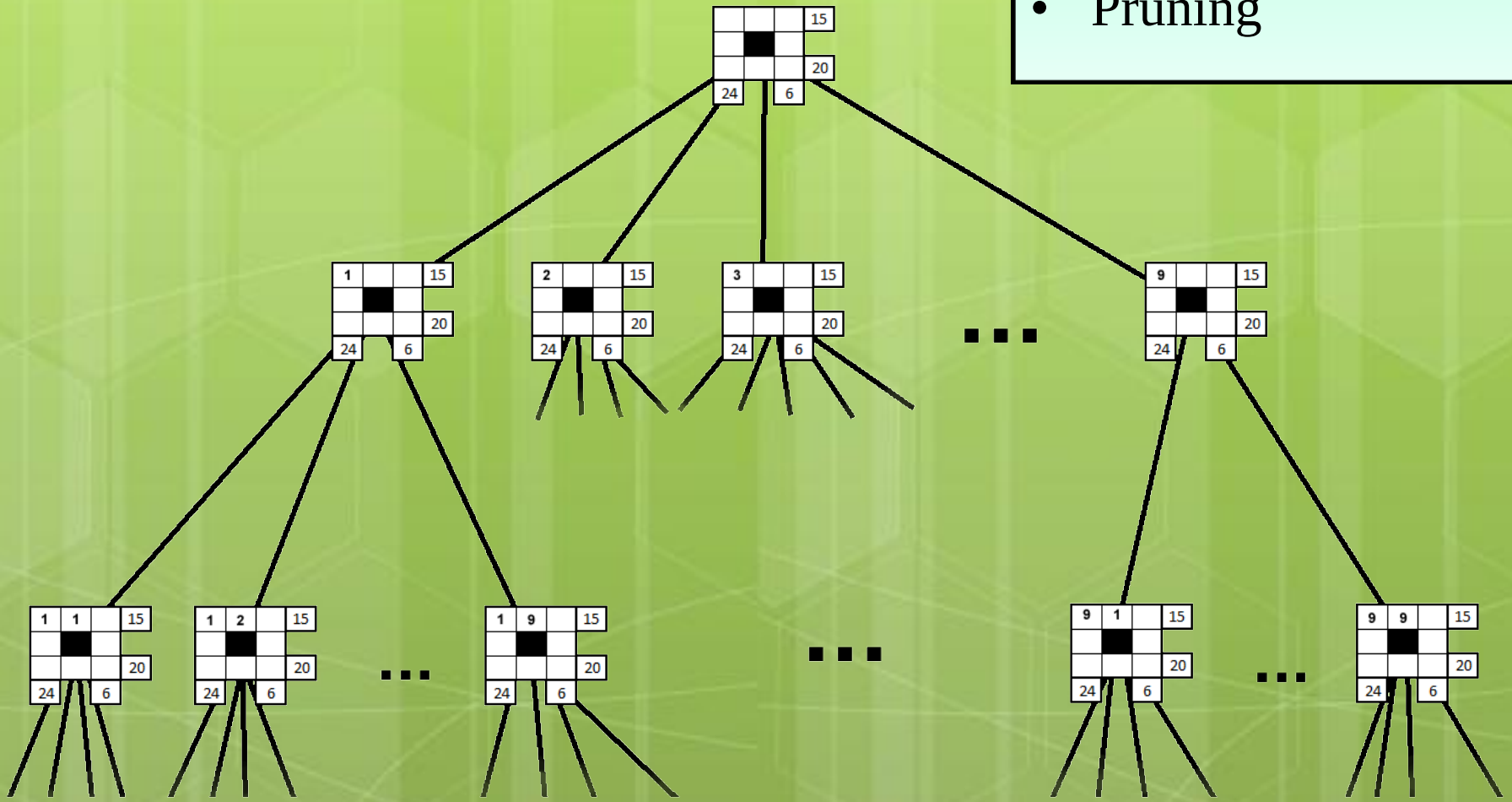
Alle mogelijkheden langs
(“brute force”)

Oplossing teruggeven, of een “bestaat niet”
(“compleet”)

# Constructieve Algoritmes

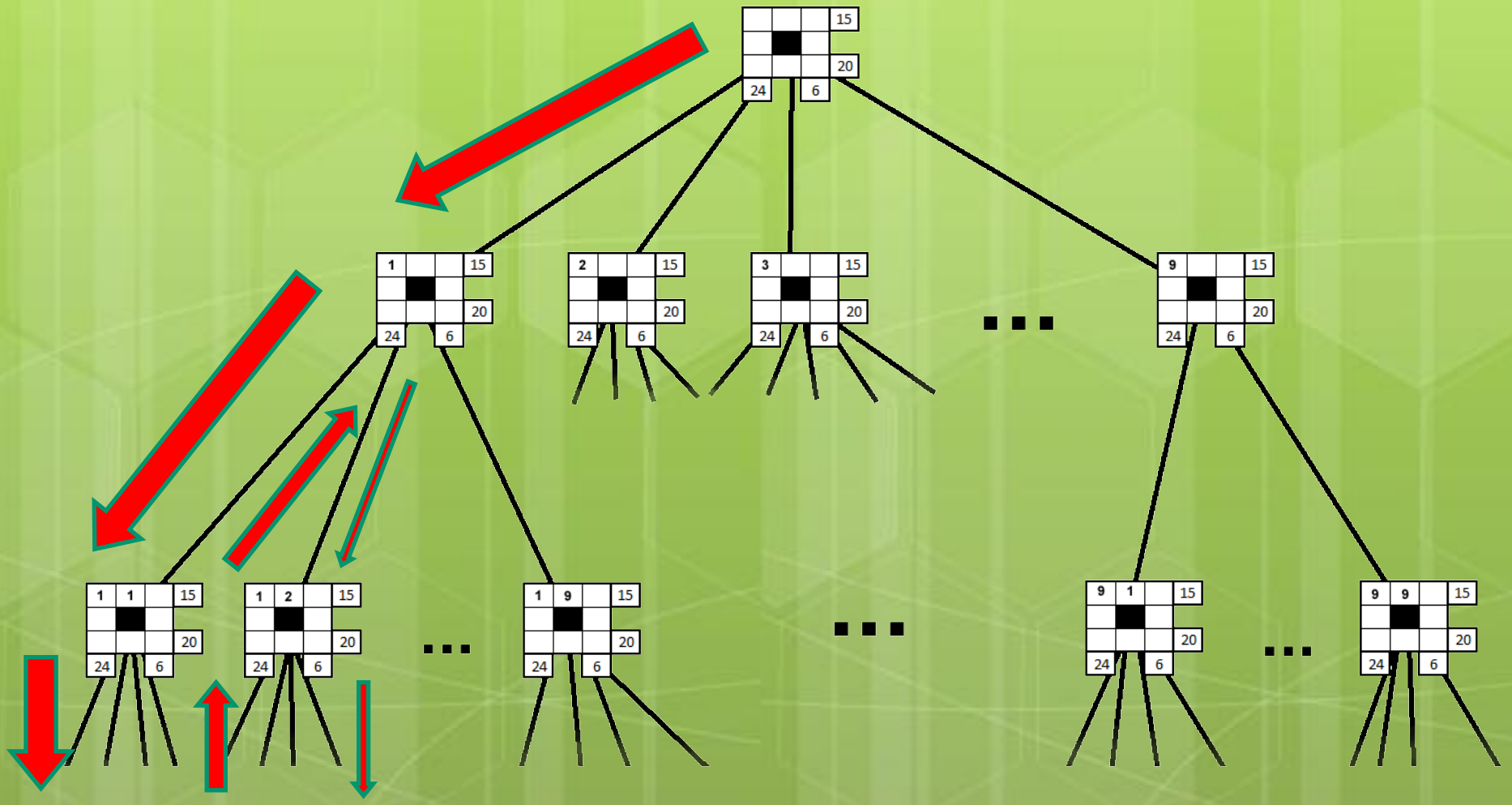
## De basics

- Search-tree van states
- Branchfactor B
- Diepte d
- $B^d$ = ||State-Space||
- Pruning



# Depth-first search

## Recept

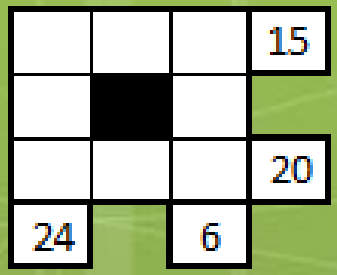


# Depth-first search

## Recept

Maak ‘kakuro-klasse’ en een stack (waar kakuro’s in kunnen)
Definieer beginsituatie; en push die op de stack

```
while(stack not empty && solution not found) {
        parent = stack.pop();
        C = GenerateAllChildren(parent);
        for (all x in C) {
                if (x == solution) {print(x); stop; } else {
                        stack.push(x);
        }
}
```

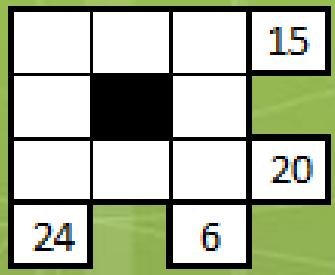


# Depth-first search

## Alternatief Recept

- Recursief

```
vulVolgendeVakje (parent) {
        C = GenerateAllChildren(parent);
        for (all x in C) {
                if(laatstevakje) { checkVoorOplossing(x);}
                else { vulVolgendeVakje (x); }
}
```



# Depth-first search

## De basics

- Schuifpuzzel (a.k.a. 8-puzzle, block-sliding puzzle)
- Wat is exact de vraag?

# Depth-first search

## De basics

- Schuifpuzzel (a.k.a. 8-puzzle, block-sliding puzzle)
- Wat is **exact** de vraag?

"Is er een oplossing?"
"Wat is de oplossing?"
"Wat is een oplossing?"
"Wat is de beste oplossing?"

# Depth-first search

## De basics

- Schuifpuzzel (a.k.a. 8-puzzle, block-sliding puzzle)
- Wat is **exact** de vraag?

"Vind de beste oplossing" if existent at all.

(Ingewikkelder dan minikakuro!)
and that's not the only thing that's harder

(een oplossing is een sequentie van states, gedefiniëerd door de schuifoperator)

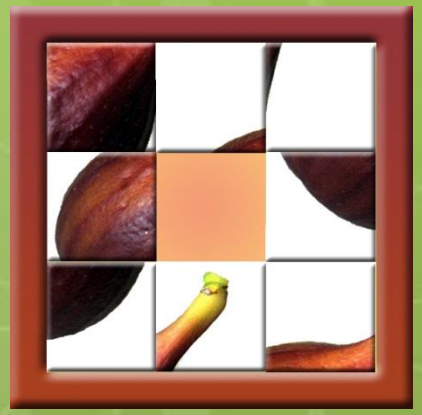
# Depth-first search

## Recept

Maak ‘schuifpuzzel-klasse’ en e
en stack (waar schuifpuzzels in kunnen)

Definieer beginsituatie en push die op de stack

```
while(stack not empty && solution not found) {
        parent = stack.pop();
        C = GenerateAllChildren(parent);
        for (all x in C) {
                if (x == solution) {print(x); stop; } else {
                        stack.push(x);
        }
}
```



# Depth-first search

## Recept

Maak ‘schuifpuzzel-klasse’ en
een stack (waar schuifpuzzels in kunnen)

Definieer beginsituatie en push die op de stack

```
while(stack not e          solution not found) {
        parent             ();
        C = C              ren(parent);
        for
                           {print(x); stop; } else {
                           h(x);
        }
}
```

# Depth-first search

## Recept

- Issue: loops (“hangen!”)
- Issue: kwaliteit oplossing
- Issue: *vorm* oplossing

# Depth-first search

## Recept

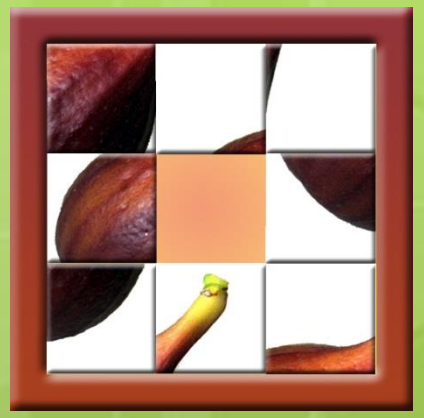
- Administratie
- *Puzzelnodes* en *Archief*

# Depth-first search

## Recept

Maak ‘schuifpuzzel-klasse’, een stack en een ‘archief’
(waar schuifpuzzels in kunnen)

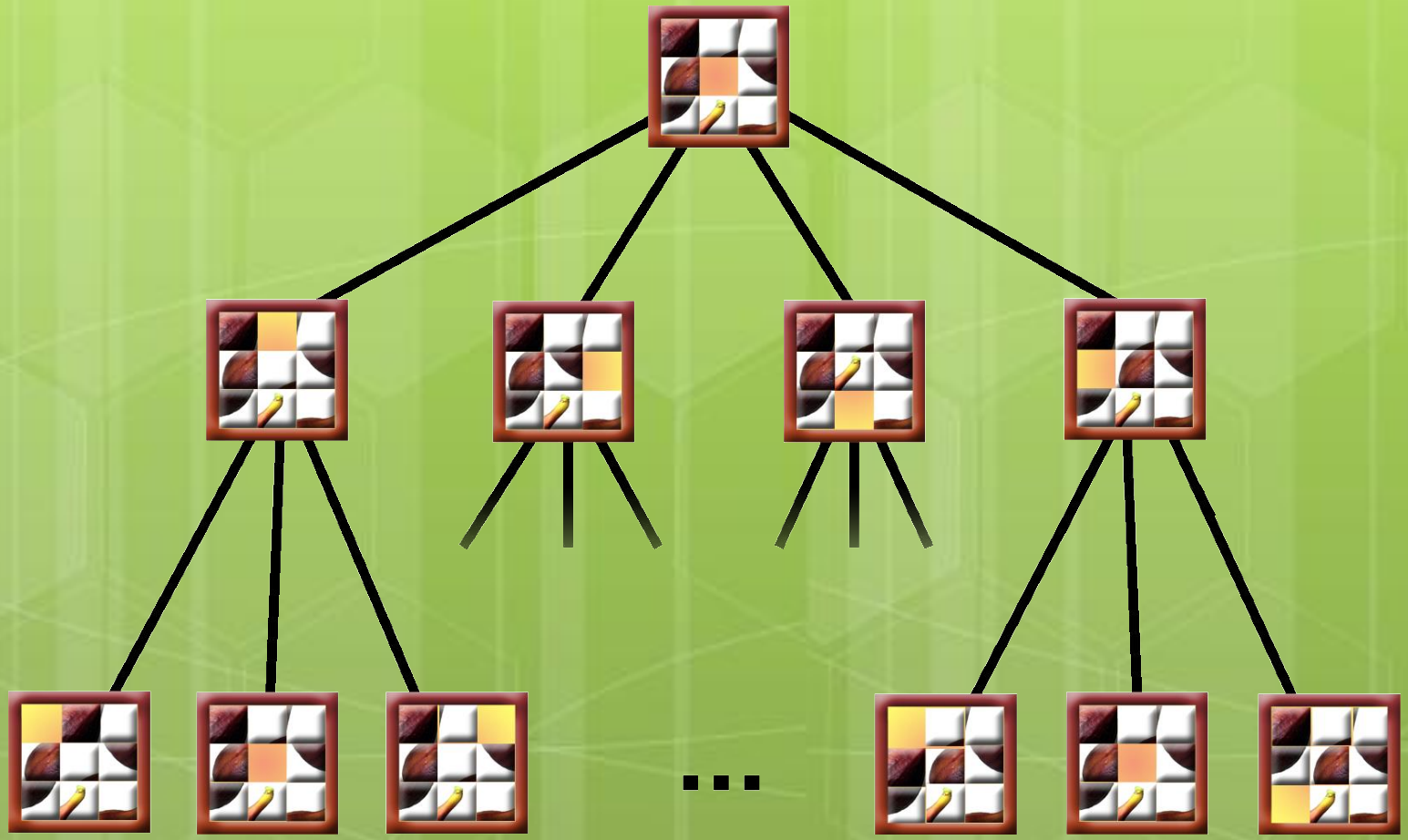
Definieer beginsituatie en push die op de stack



```
while(stack not empty && solution not found) {
        parent = stack.pop();
        C = GenerateAllChildren(parent);
        for (all x in C) {
                if( ! archief.bevat(x))
                        { archief.add(x) } else {
                        if (x == solution) {print(x); stop; } else {
                stack.push(x);
        }}}
```

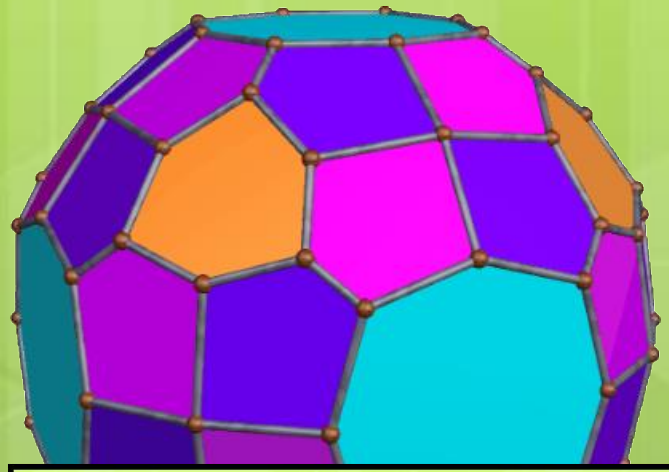
# Depth-first search

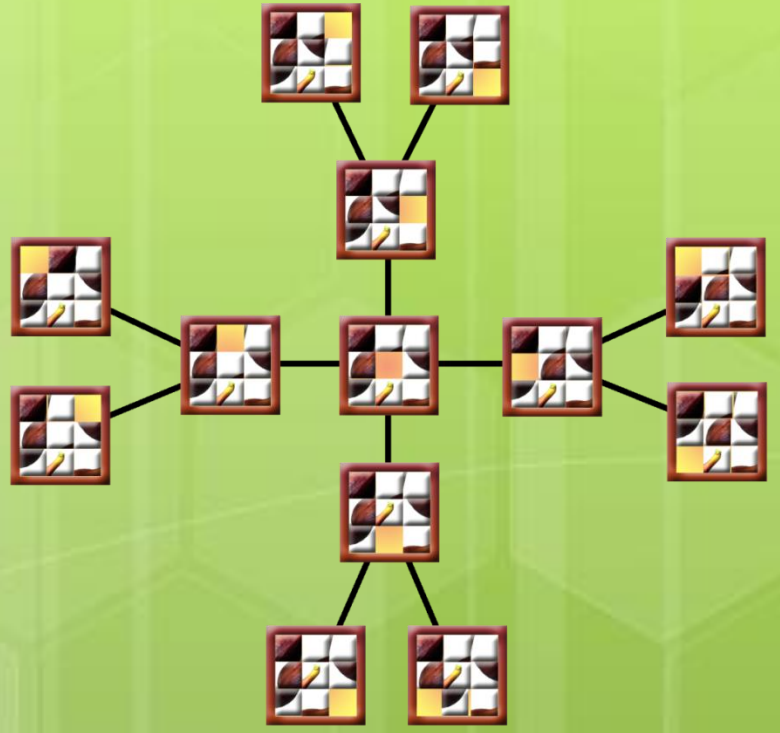
## De toestandsruimte herzien



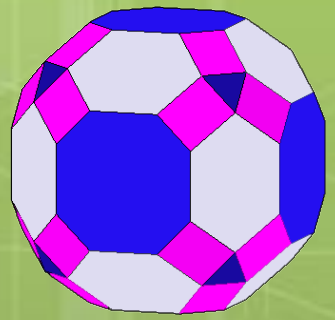
# Depth-first search

## De toestandsruimte herzien





- *Alle node*s (362 880) zijn states
- degree 4 (11%), 3 (44%), 2 (44%)
- Alle operaties zijn *inverteerbaar*
- (grote) cykels
- *maximale* afstand (“diameter”)



# Depth-first search

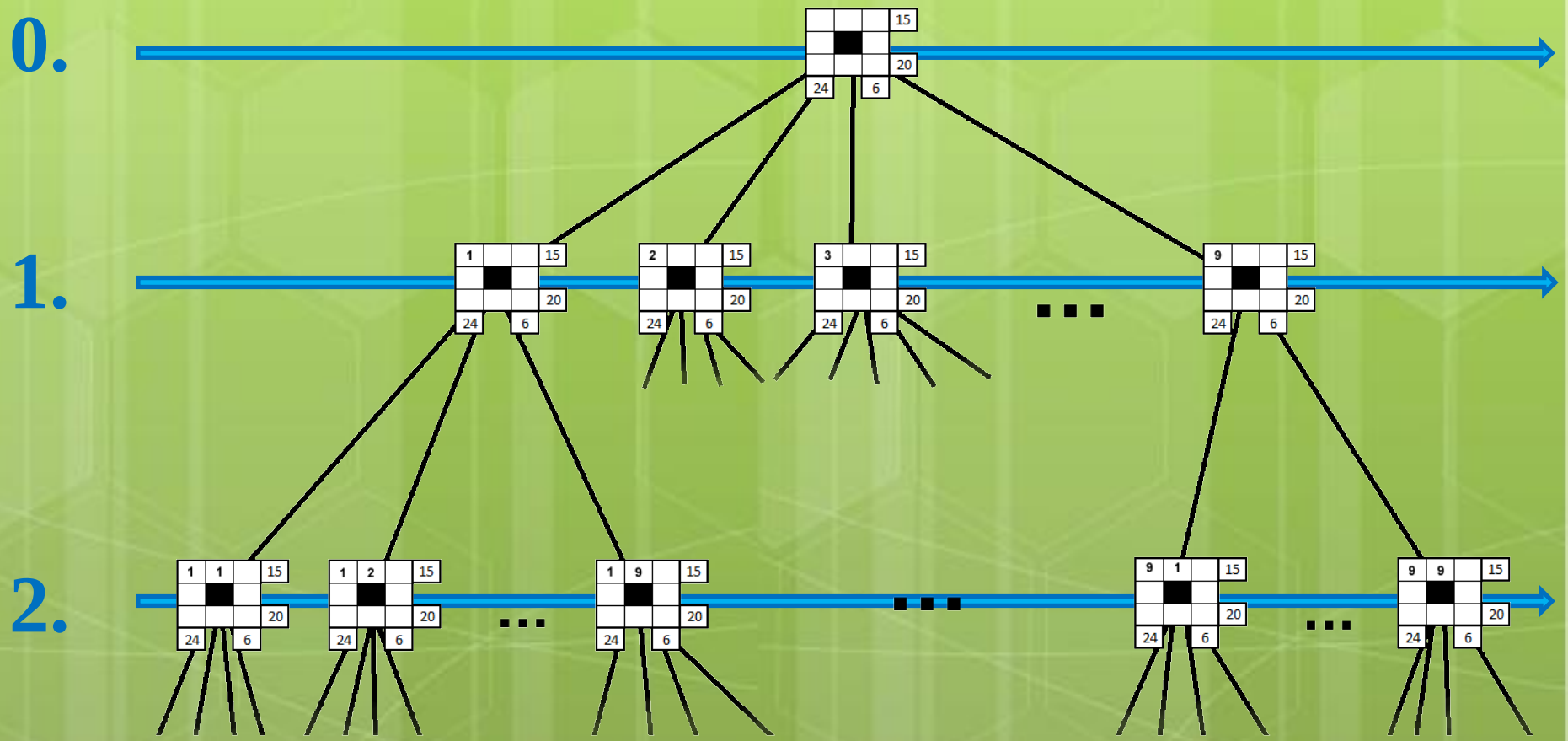
## De toestandsruimte herzien

- Zuivere depth-first search is *brute-force*

- Eindige state space?
- Loops?
- Pruning
- Depth-limited, Iterative deepening
- Backtracking

# Breadth-first search

## Recept

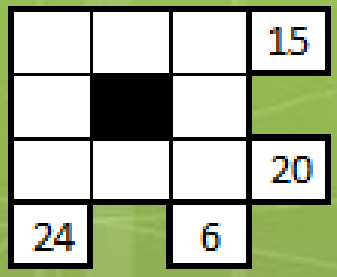


# Breadth-first search

## Recept

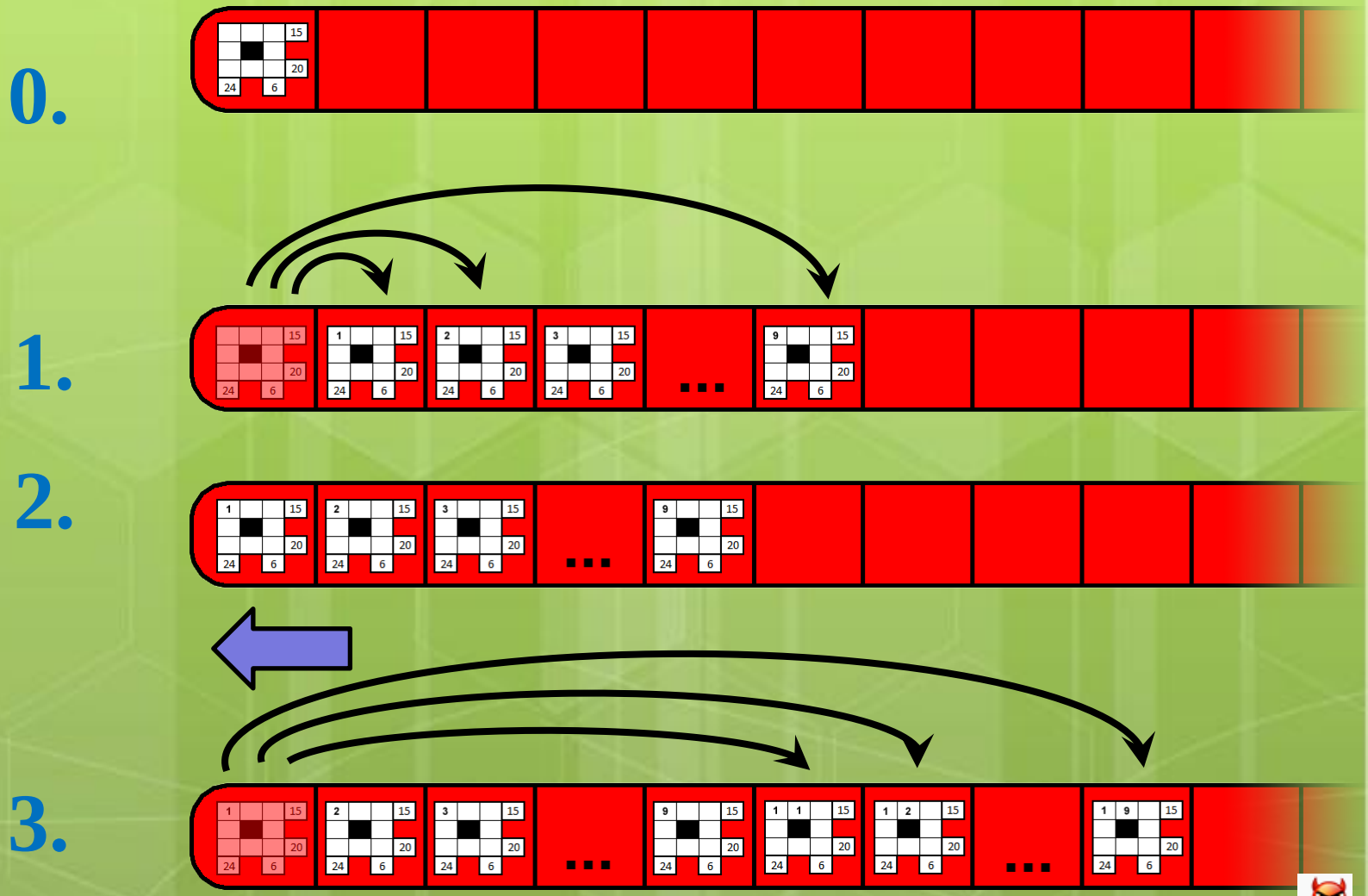
Maak ‘kakuro-klasse’ en een **queue** (waar kakuro’s in kunnen)
Definieer beginsituatie en *add* die aan de queue

```
while(queue not empty && solution not found) {
        parent = queue.remove();
        C = GenerateAllChildren(parent);
        for (all x in C) {
                if (x == solution) {print(x); stop; } else {
                        queue.add(x);
        }
}
```

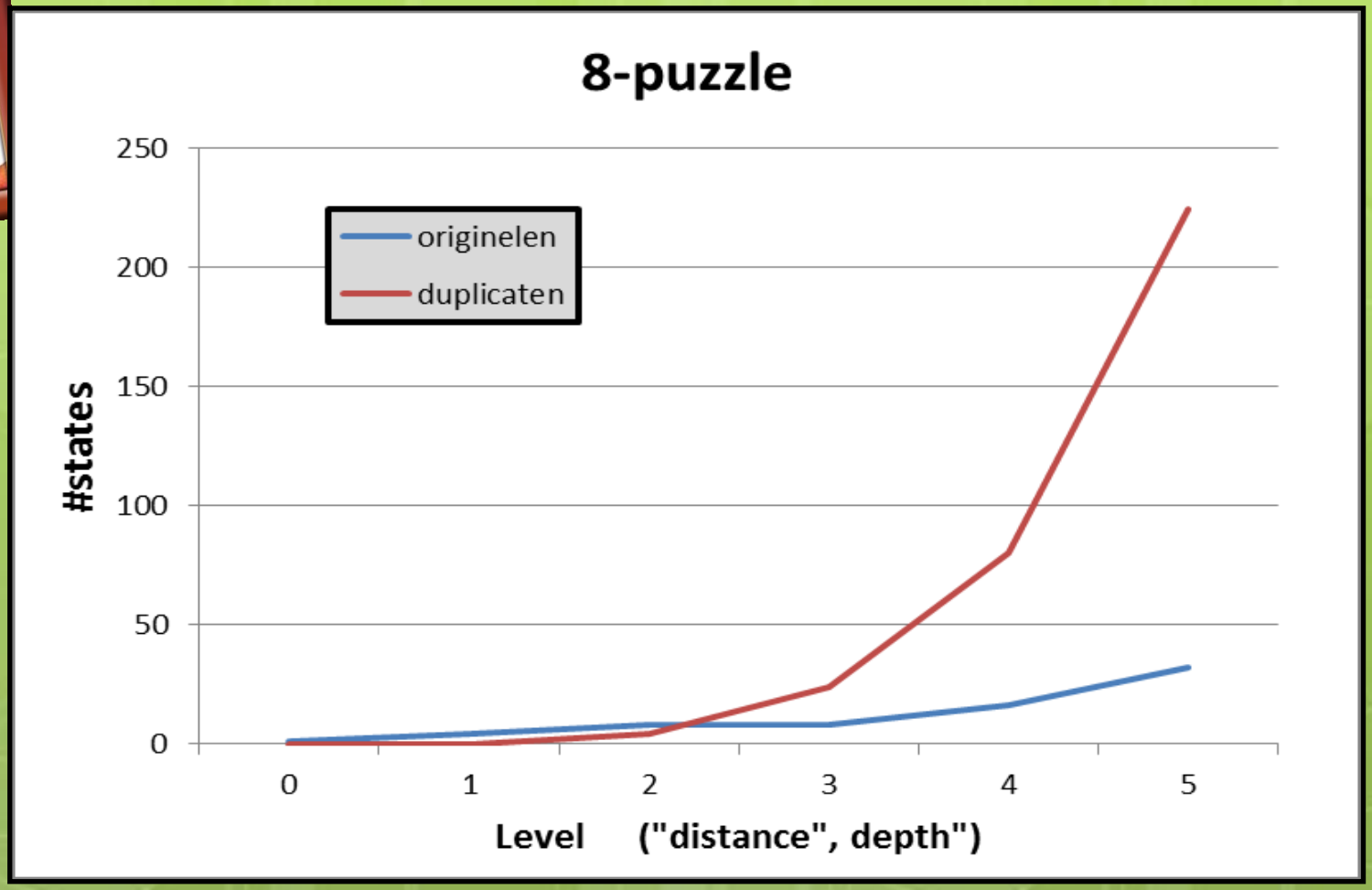


# Breadth-first search

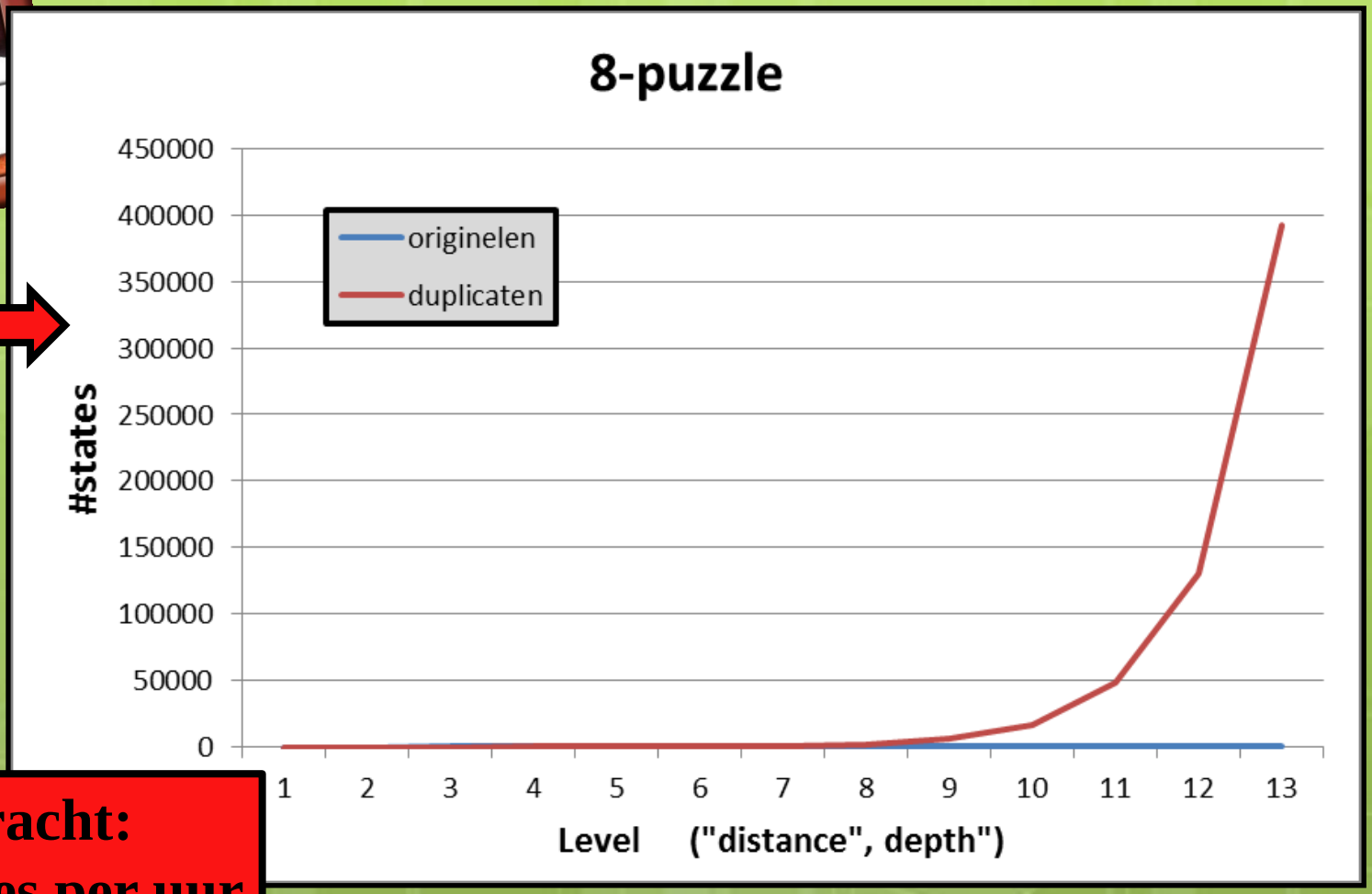
## Recept



# Invloed van loops

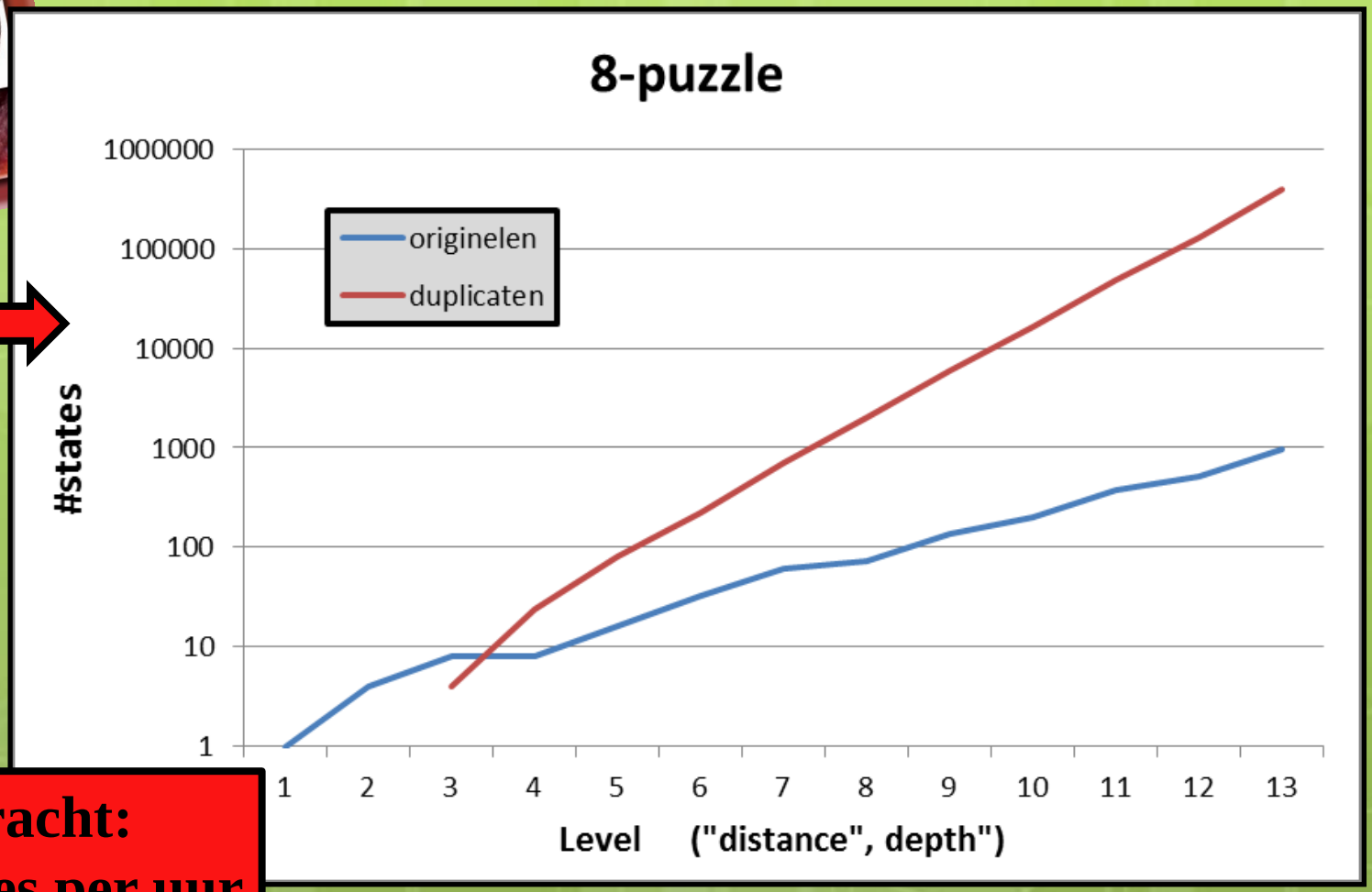


# Invloed van loops




**rekenkracht:**
**100 000 states per uur**

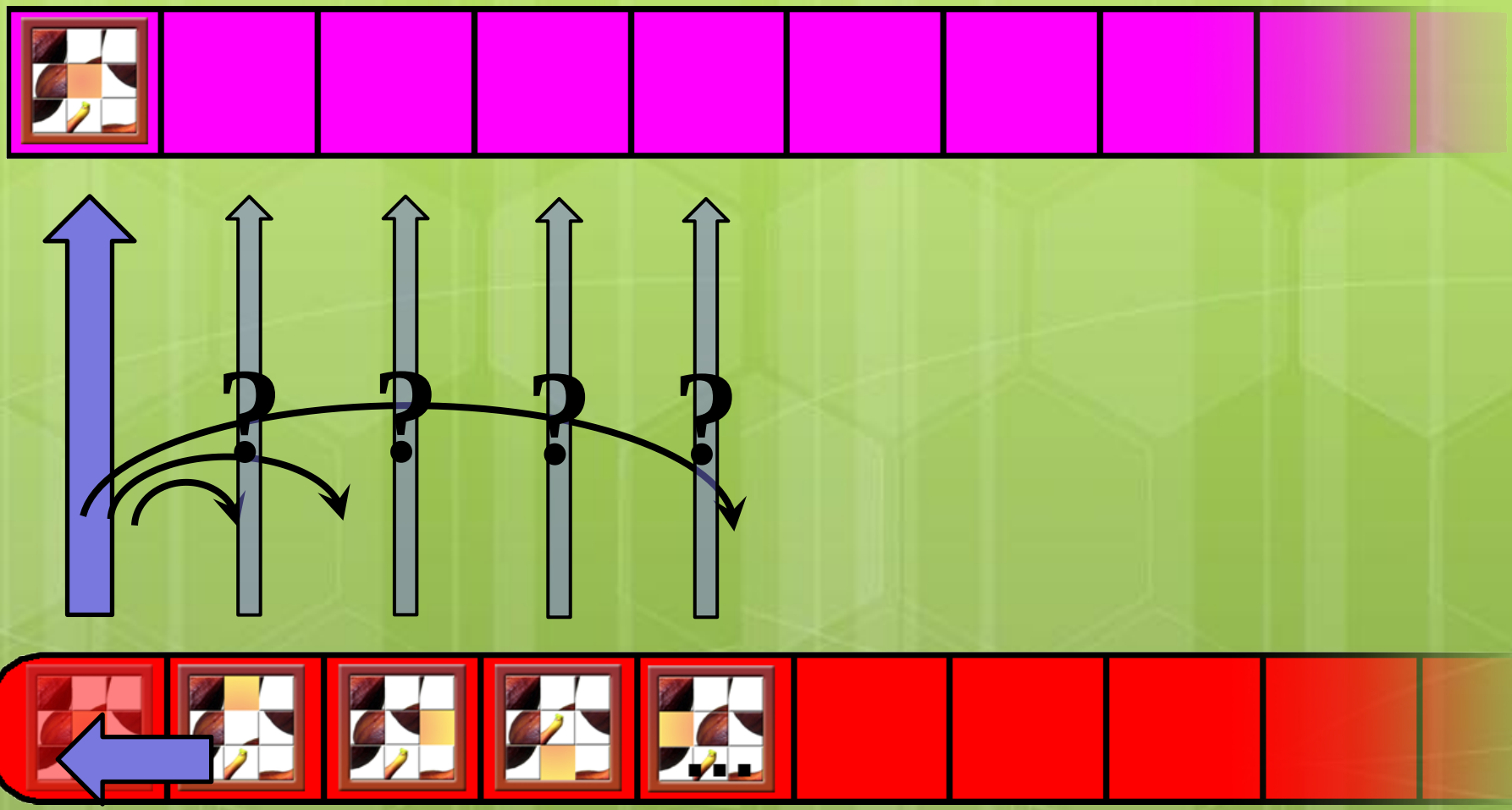
# Invloed van loops




**rekenkracht:**
**100 000 states per uur**

# Breadth-first search

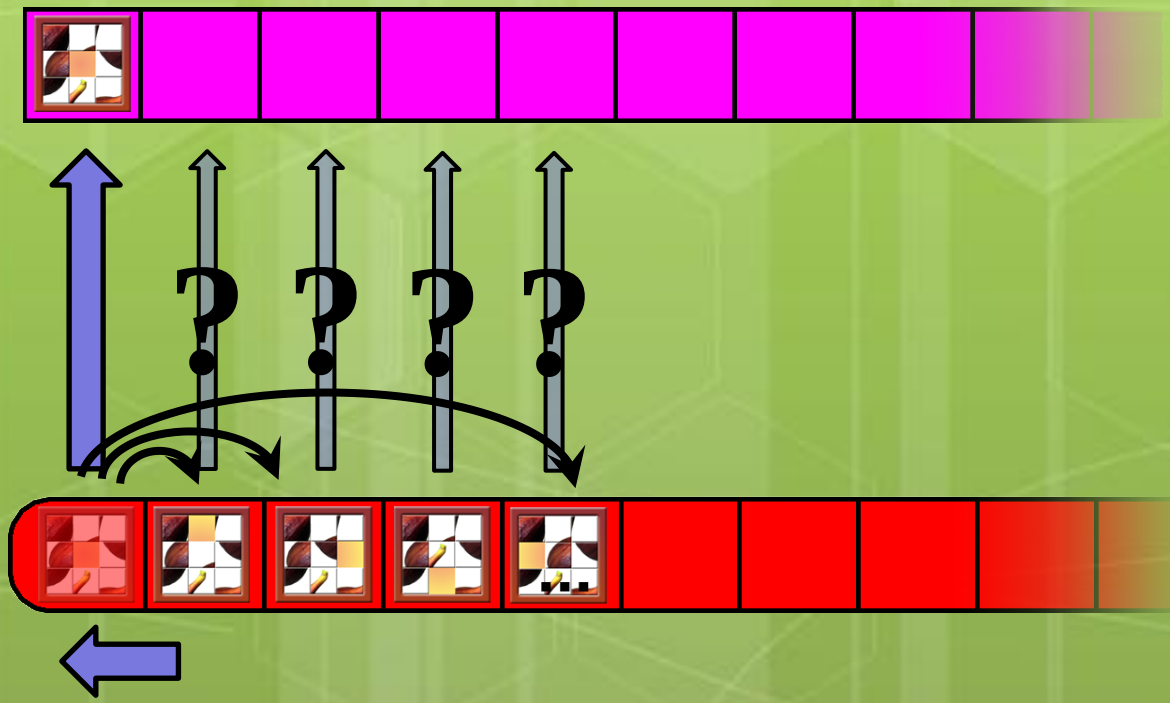
**Recept *met archief***



# Breadth-first search

**Recept *met archief***

- Geheugengebruik?
- Datastructuur?
- Enumeratie?
- Checkproces?
- HHK



# Uniform-cost search

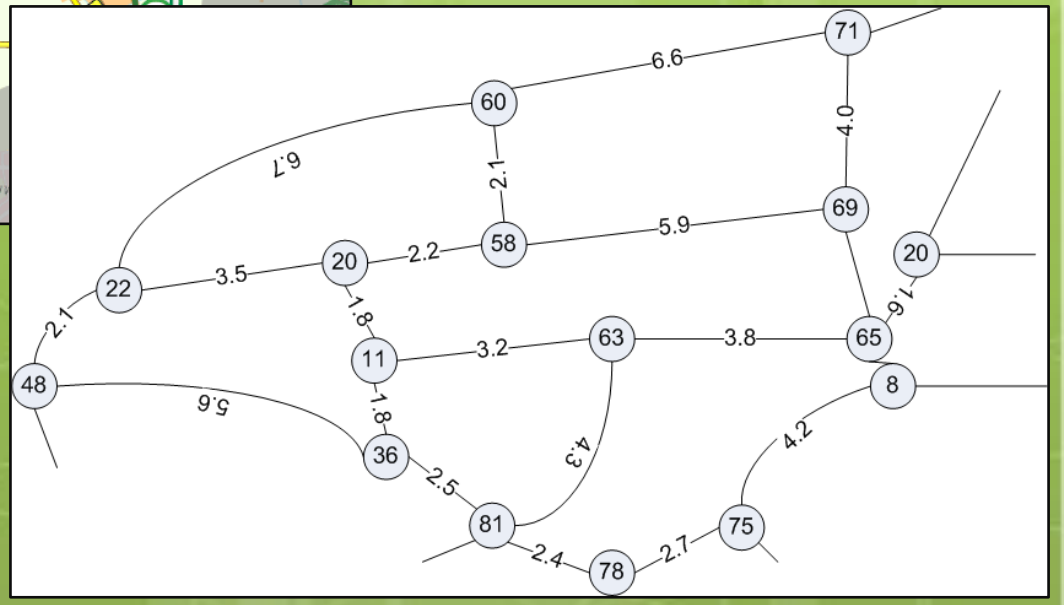
## Zoeken in een gewogen graaf

# Uniform-cost search

## Zoeken in een gewogen graaf

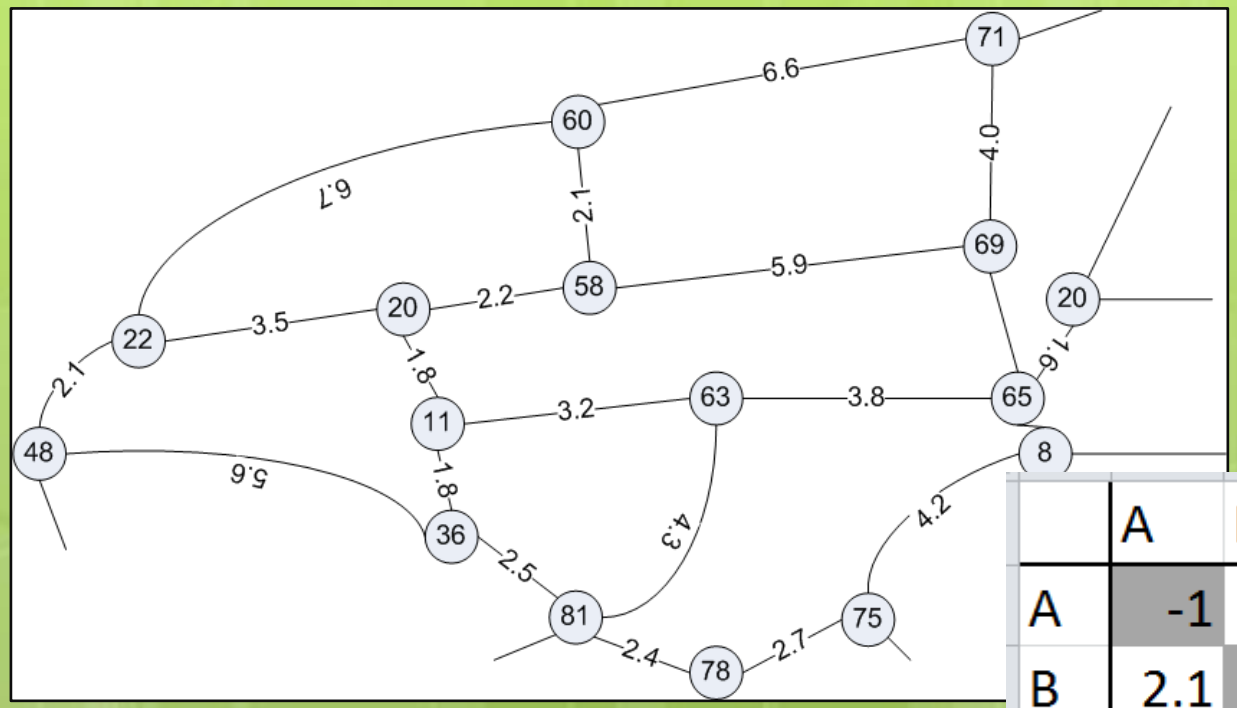
# Uniform-cost search

## Zoeken in een gewogen graaf

# Uniform-cost search

## Zoeken in een gewogen graaf




| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| A | -1 | 2.1 | -1 | -1 | -1 | -1 | 5.6 |
| B | 2.1 | -1 | 6.7 | 3.5 | -1 | -1 | -1 |
| C | -1 | 6.7 | -1 | -1 | 2.1 | -1 | -1 |
| D | -1 | 3.5 | -1 | -1 | 2.2 | 1.8 | -1 |
| E | -1 | -1 | 2.1 | 2.2 | -1 | -1 | -1 |
| F | -1 | -1 | -1 | 1.8 | -1 | -1 | -1 |
| G | 5.6 | -1 | -1 | -1 | -1 | 1.8 | -1 |

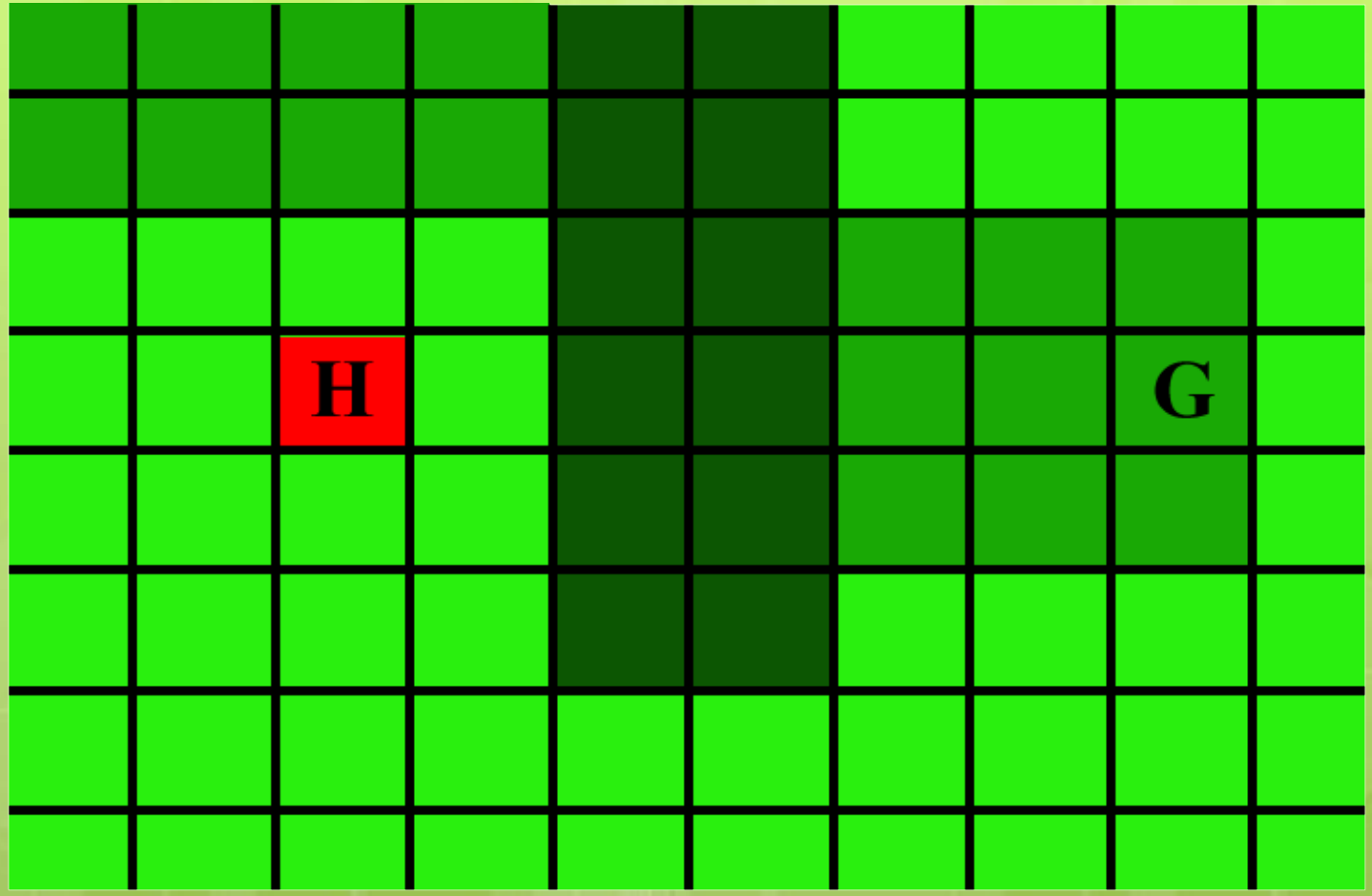
# Uniform-cost search

## Recept

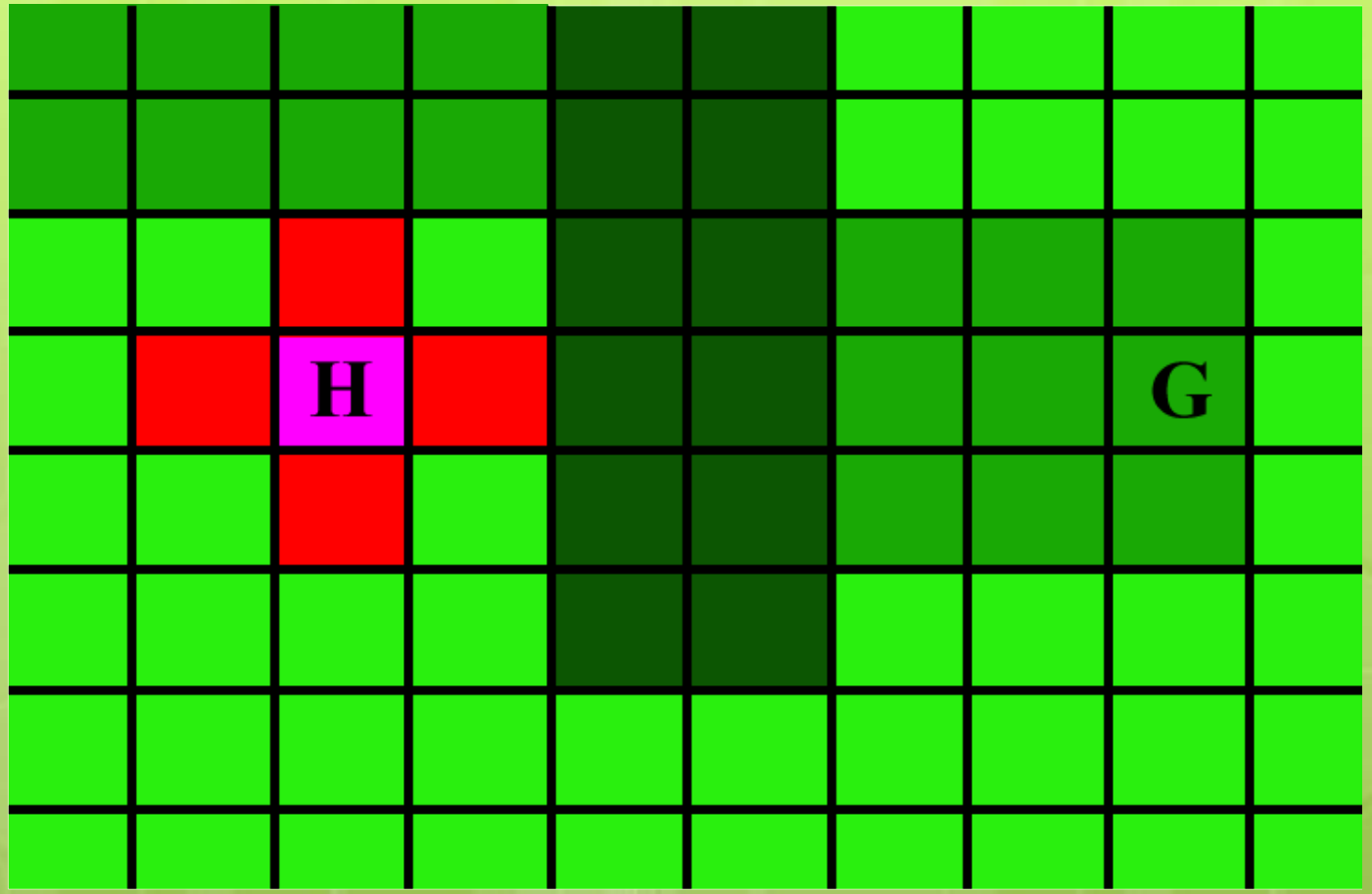
- Als breadth-first met archief, maar hou padlengte bij
- Queue = *Priority Queue*
- Prioriteit = padlengte of beter: *padkortte*

- Compleet

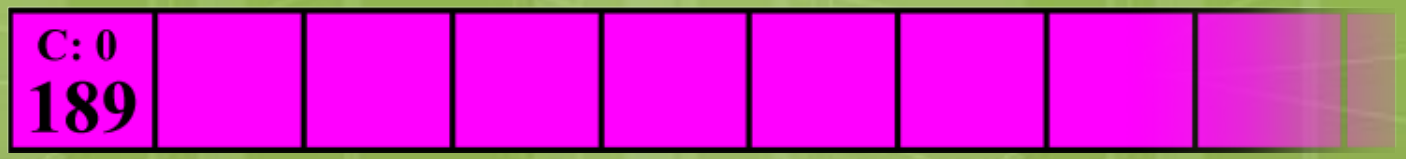
# Uniform Cost Search

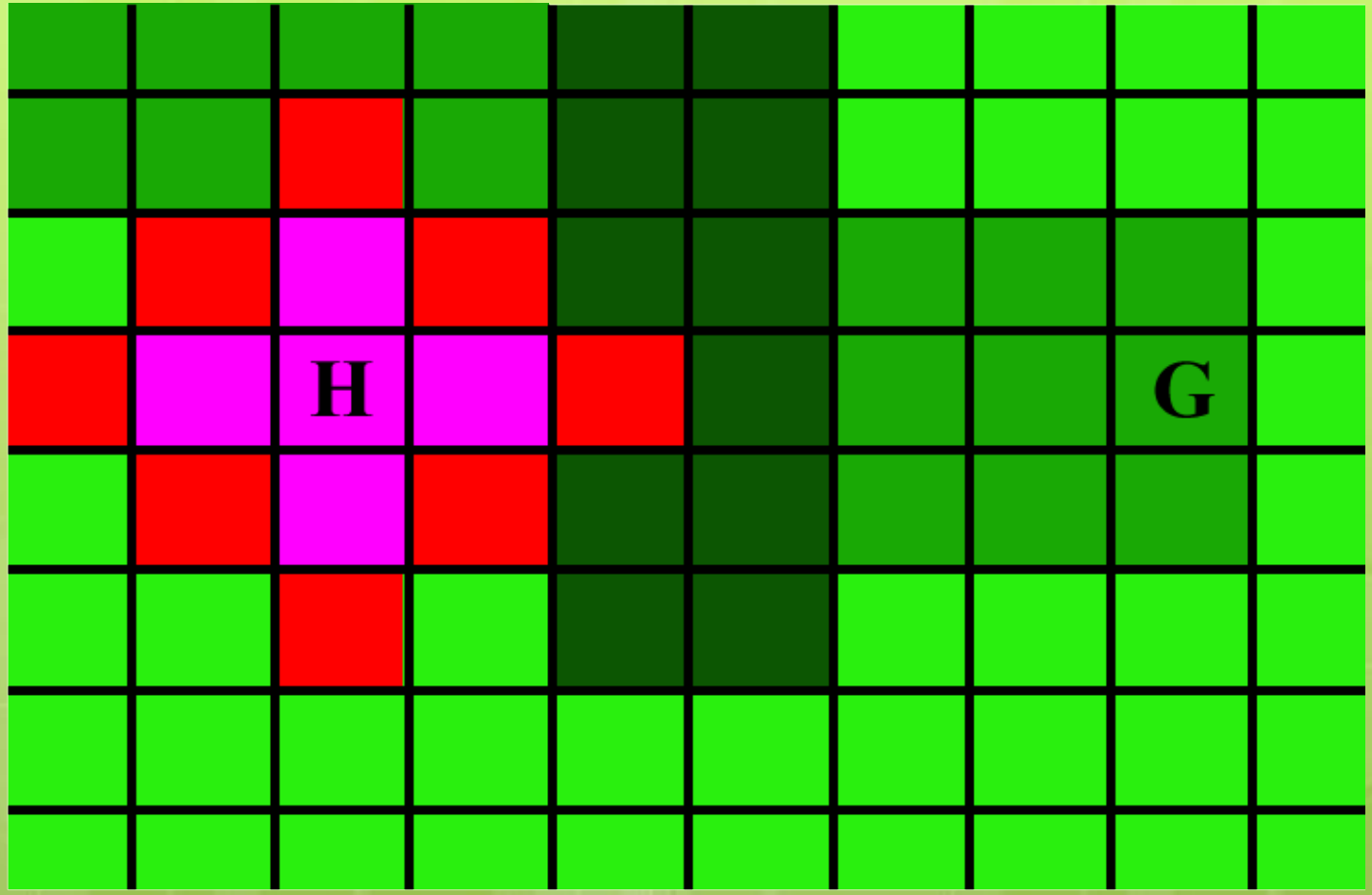



C: 0
189


H
G

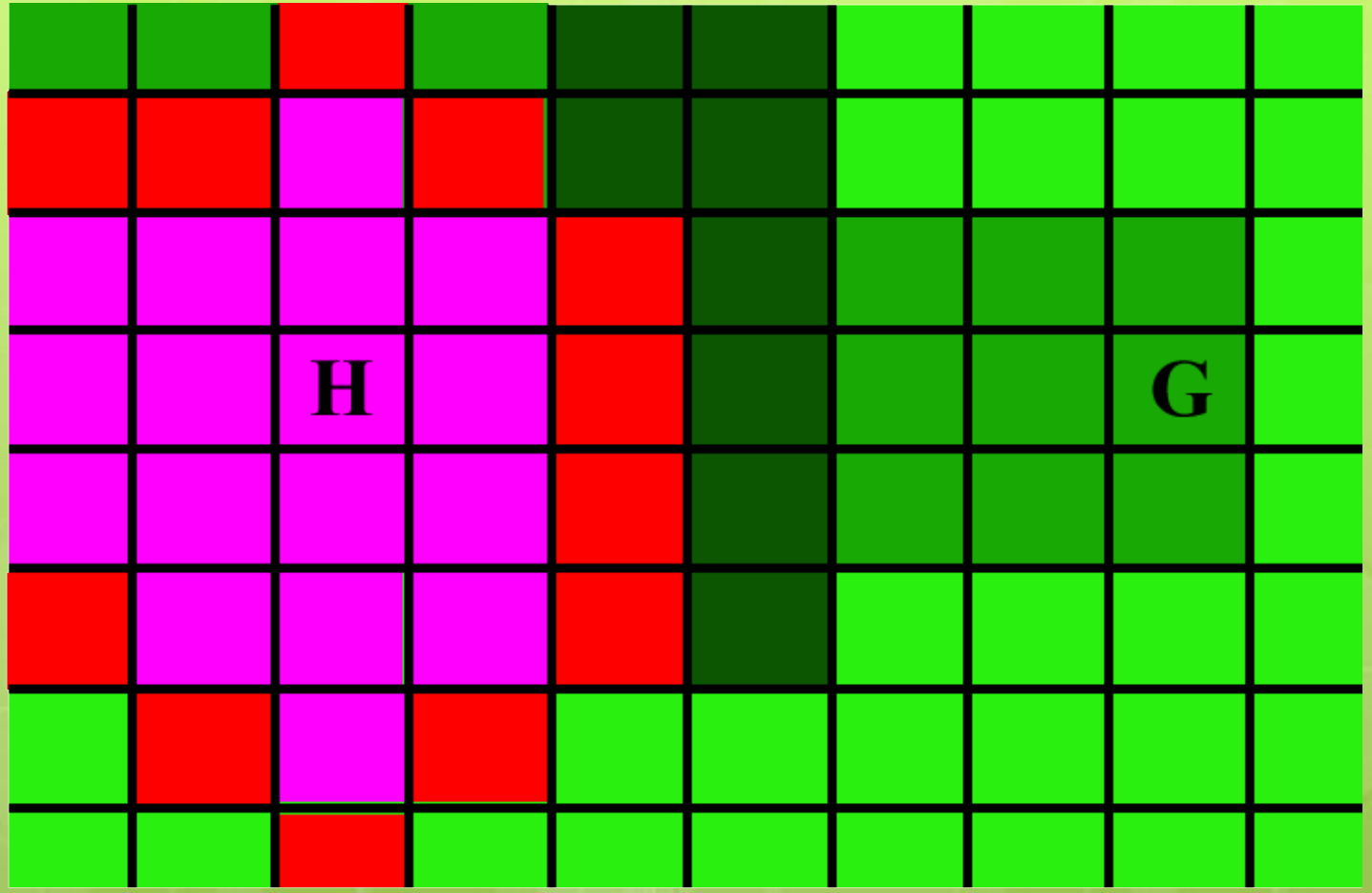

C: 1
139
C: 1
188
C: 1
190
C: 1
238


C: 0
189




| C: 2 | C: 2 | C: 2 | | C: 3 | C: 4 | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 138 | 140 | 187 | ... | 89 | 191 | | | | |

| C: 1 | C: 1 | C: 0 | C: 1 | C: 1 | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 139 | 188 | 189 | 190 | 238 | | | | | |




| C: 4 | C: 4 | C: 4 | | C: 5 | C: 6 | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 88 | 90 | 287 | ... | 87 | 291 | | | | |

| C: 3 | C: 3 | C: 2 | C: 1 | | C: 1 | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 89 | 137 | 138 | 139 | ... | 238 | | | | |