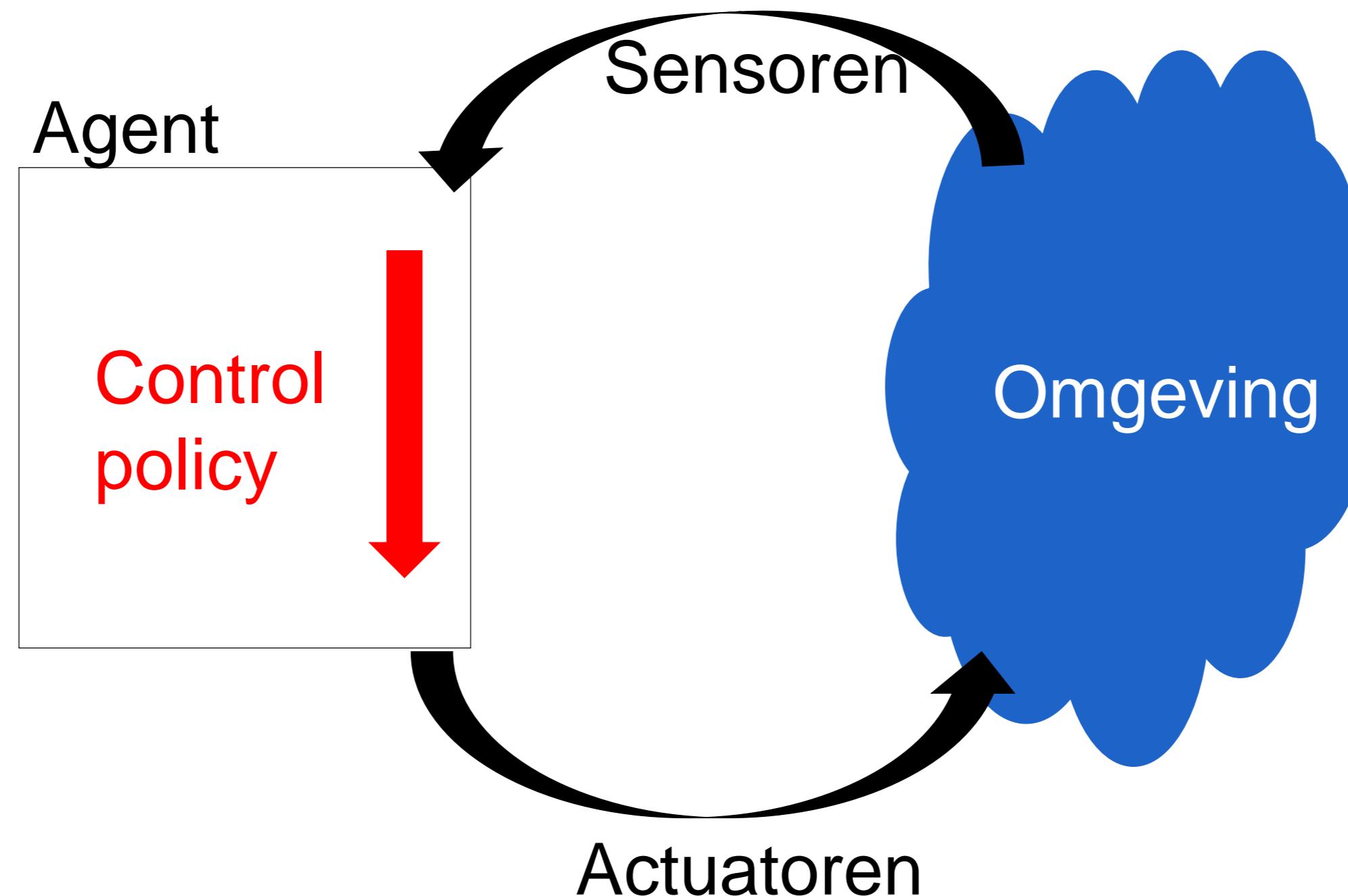


ZOEKHEURISTIEKEN – 1

prof. dr. Yvan Saeys (yvan.saeys@ugent.be)
Bureau: Sterre S9, 1e verdiep (naast leslokaal 1.1)

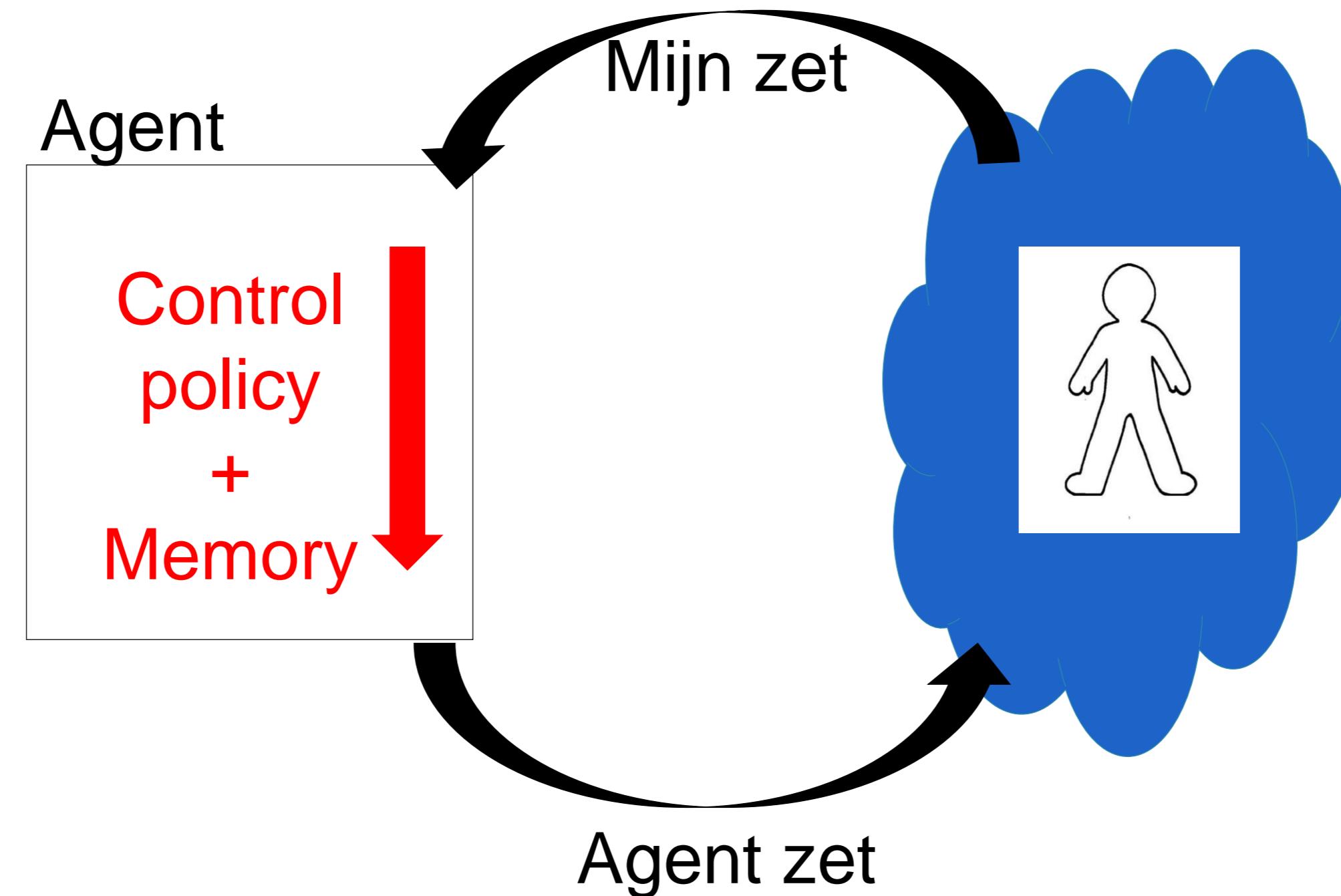
INTELLIGENT AGENTS



EIGENSCHAPPEN VAN DE OMGEVING

- Volledig observeerbaar
 - Sensoren kunnen de volledige toestandsinformatie zien
- Partieel observeerbaar
 - Sensoren kunnen enkel een deel van de toestandsinformatie zien, maar de agent kan vorige acties onthouden om bijkomende informatie over de toestand te verzamelen die momenteel niet gemeten kan worden

VOORBEELD: GAME-PLAYING AGENT



ANDERE EIGENSCHAPPEN VAN DE OMGEVING

- Volledig of partieel observeerbaar
- Deterministisch of stochastisch
- Discreet of continue
- Normaal of adversarial
 - Bvb tegenstander die wil winnen van jou

Omgeving	Partieel observeerbaar?	Stochastisch?	Continu?	Adversarial?
Dammen				
Poker				
Zelfrijdende auto				

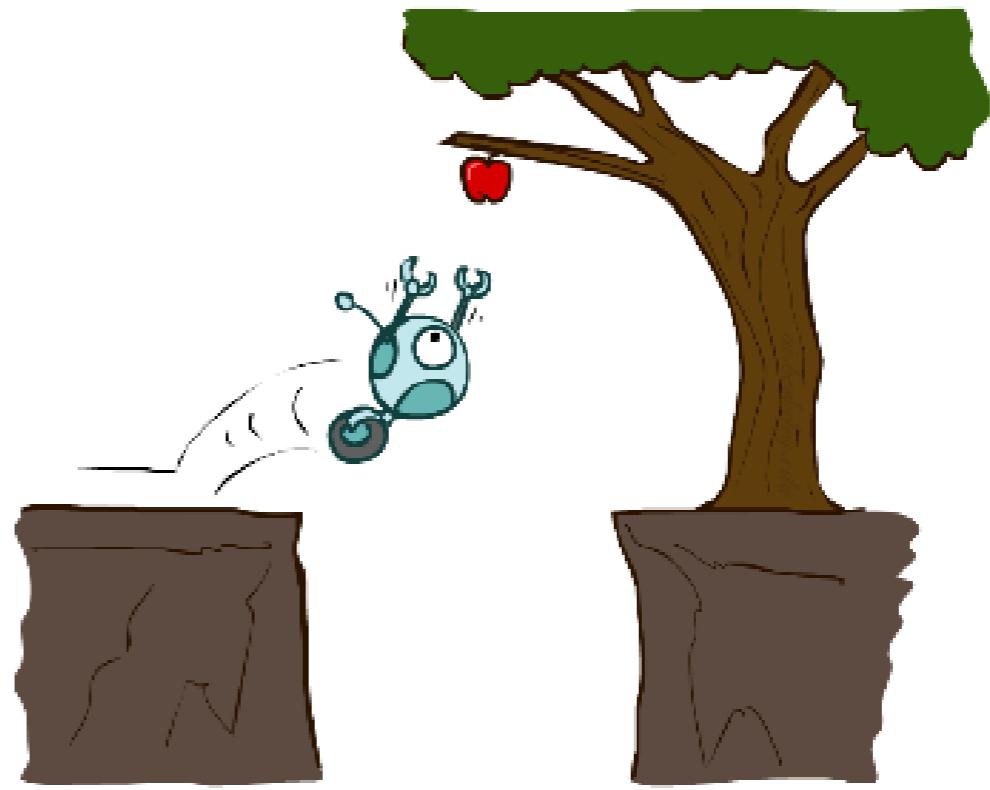


OVERZICHT

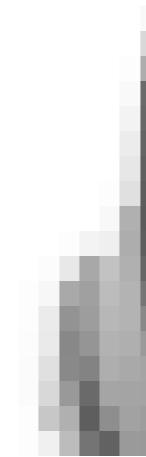
- Planning agents
- Zoekproblemen
- Zoekstrategieën:
 - Uninformed search
 - Diepte-eerst, breedte-eerst, uniforme cost
 - Informed search
 - Heuristieken, greedy search, A^{*}
 - Graph search

REFLEX AGENTS

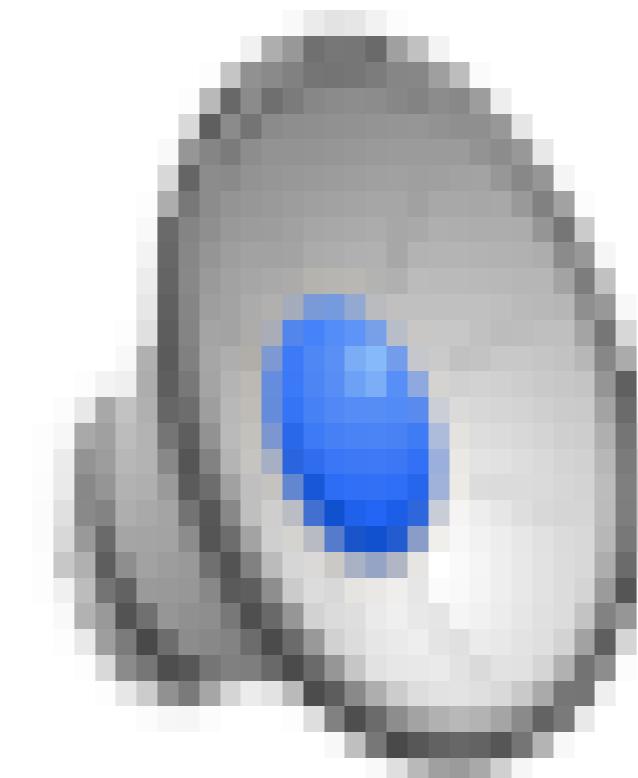
- Kiest een actie gebaseerd op huidige waarneming
- Kan beschikken over geheugen of een model van de huidige toestand van de omgeving
- ***Beschouwt geen consequenties van zijn acties***
- Bekijkt enkel hoe de wereld momenteel is



“GELUKKIG TOEVAL” REFLEX AGENT

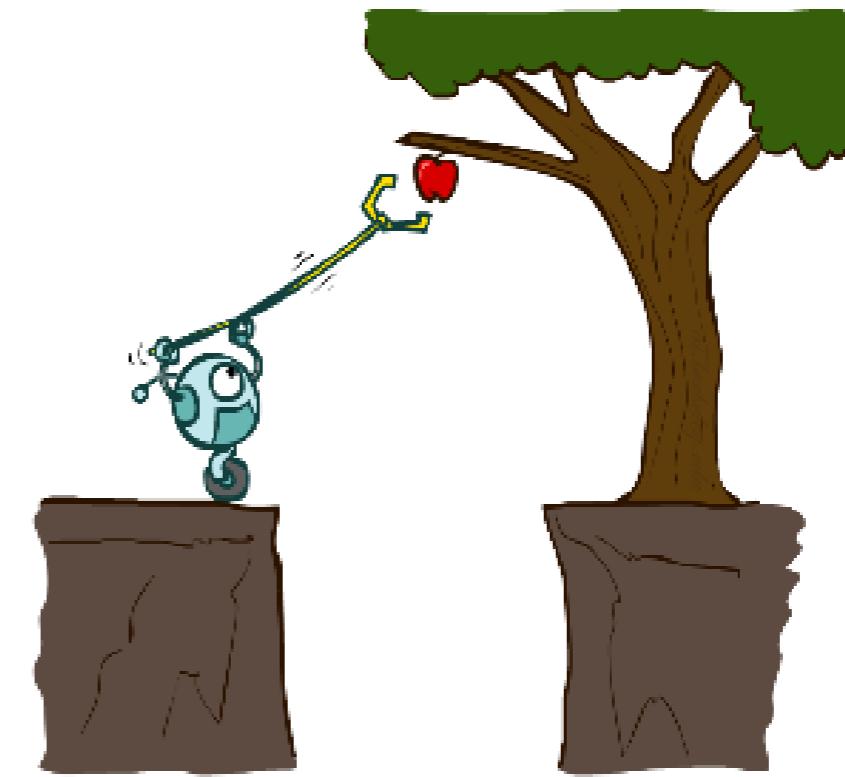


“PECHVOGEL” REFLEX AGENT



PLANNING AGENT

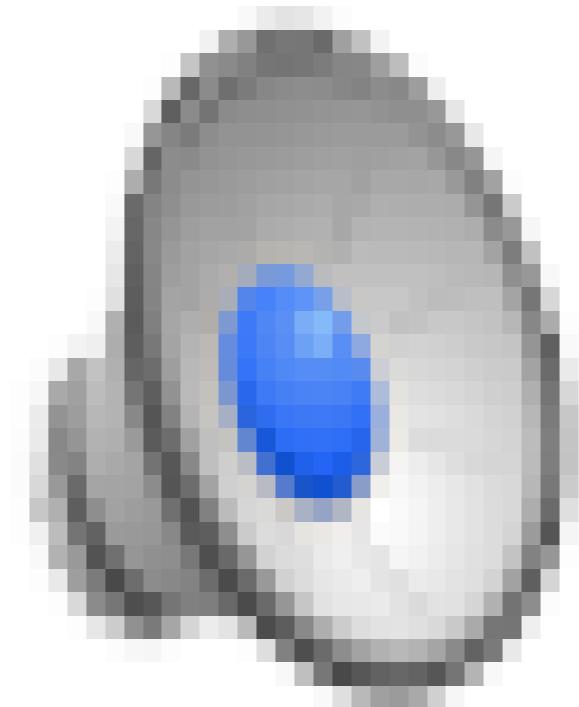
- Maakt beslissing gebaseerd op veronderstelde consequenties van zijn acties
- Heeft een model nodig van hoe de wereld verandert als hij acties uitvoert
- Moet een doeltest formuleren
- Beschouwt hoe de wereld zou zijn



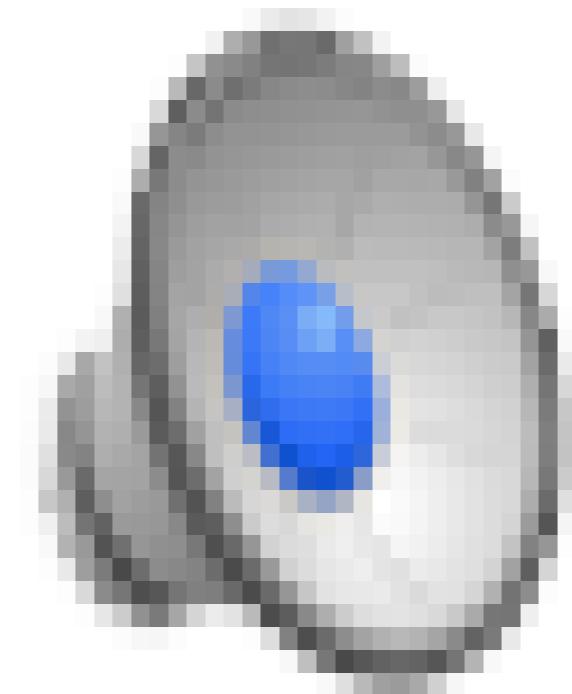
Compleet ←→
optimaal plannen

Plannen ←→ “herplannen”

OPTIMALE PLANNING AGENT



“REPLANNING” AGENT

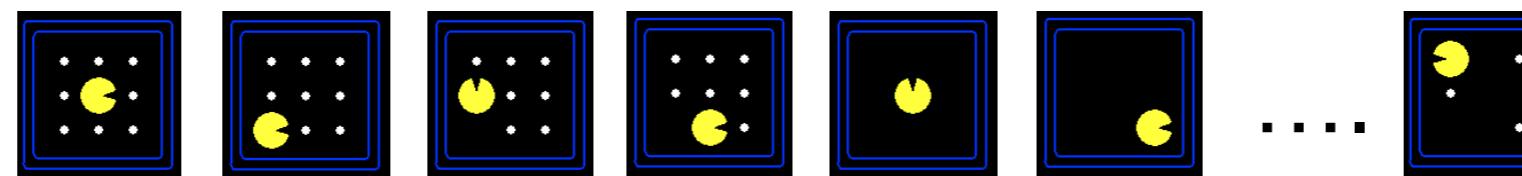


OVERZICHT

- Planning agents
- Zoekproblemen
- Zoekstrategieën:
 - Uninformed search
 - Diepte-eerst, breedte-eerst, uniforme cost
 - Informed search
 - Heuristieken, greedy search, A*
 - Graph search

FORMULEREN VAN EEN ZOEKPROBLEEM

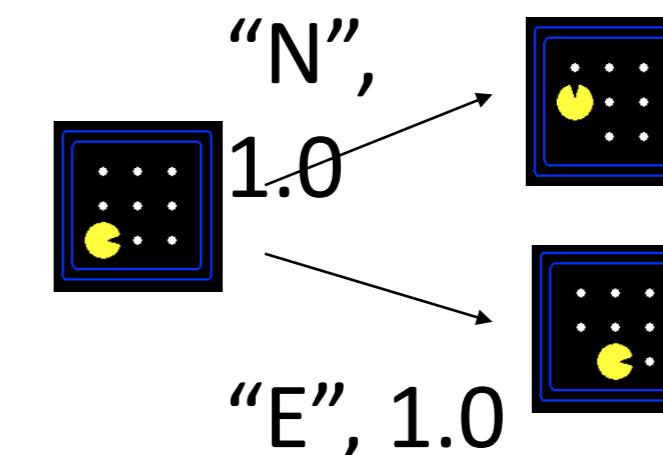
1. Toestandsruimte



2. Overgangsmodel

- Actie
- Kost

3. Starttoestand en doeltest



Oplossing = opeenvolging van acties (=plan) die ons van de starttoestand naar een doeltoestand brengt

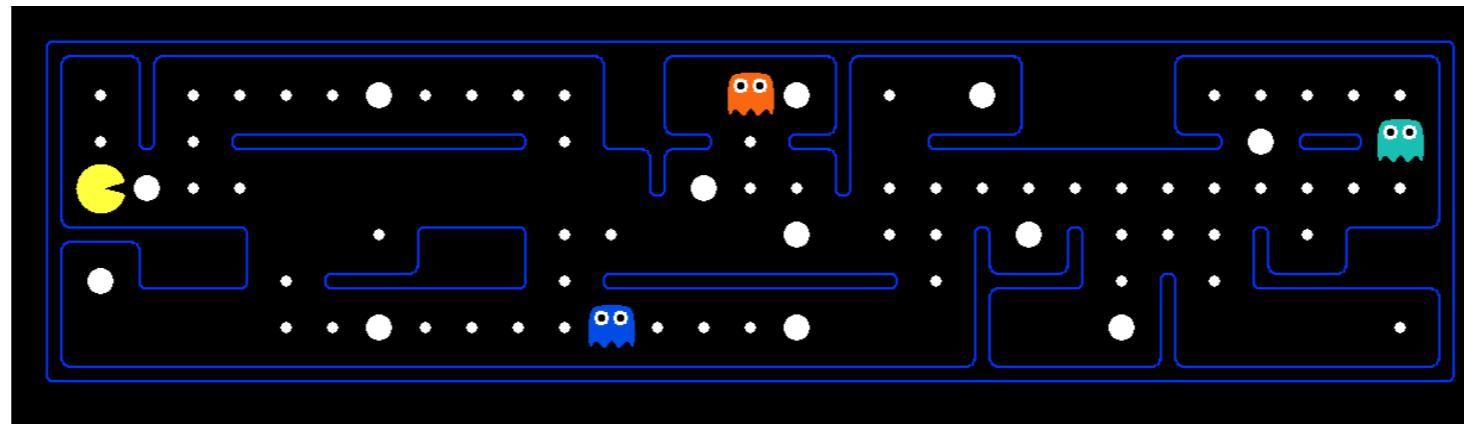
VOORBEELD: ROUTEPLANNER



- Toestandsruimte?
- Actie?
- Kost?
- Starttoestand?
- Doeltest?

WAT MOET ER IN DE TOESTANDSRUIMTE ?

De omgevingstoestand (world state) bevat elk detail van de omgeving



De **toestandsruimte** bevat enkel de informatie nodig om te kunnen plannen (abstractie)

Probleem: pad vinden

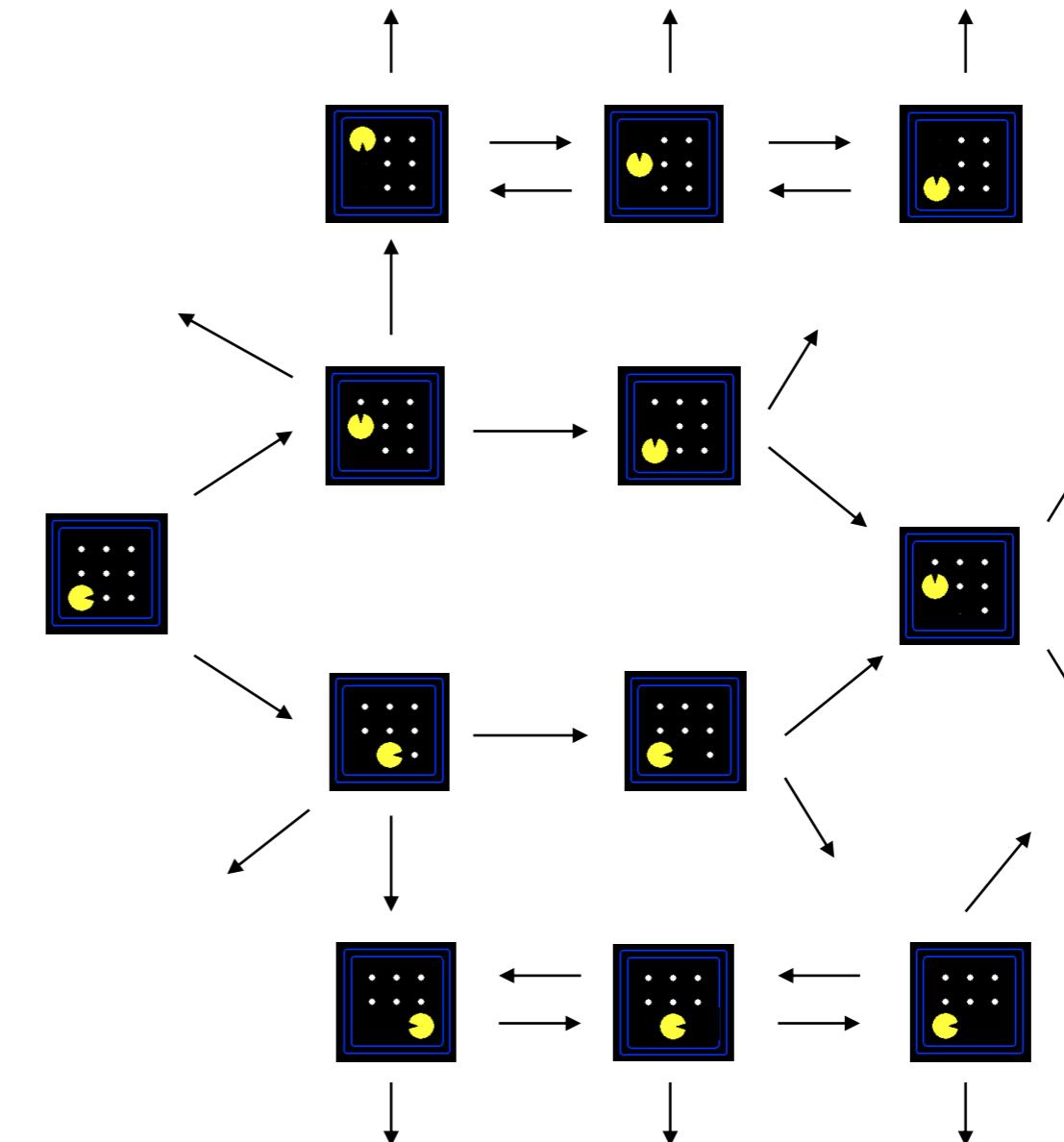
- Toestand: (x,y) locaties
- Transitiemodel: update locatie
 - Acties: NSEW
- Doeltest: is $(x,y)=\text{DOEL}$

Probleem: eet alle puntjes

- Toestand: $\{(x,y), \text{dot booleans}\}$
- Transitiemodel: update locatie
 - + eventueel 1 dot boolean
 - Acties: NSEW
- Doeltest: dots booleans allen false

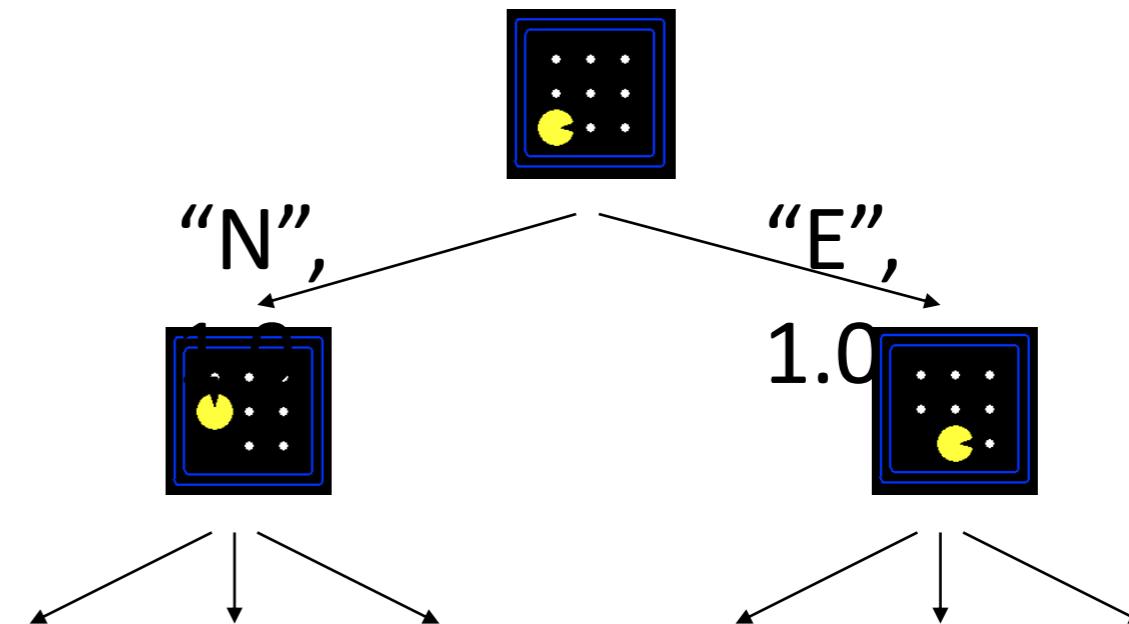
GRAAF VAN DE TOESTANDSRUIMTE

- Elke knoop stelt een toestand voor
- Bogen stellen opvolgerfunctie voor (actieresultaten)
- Doeltest is een deelverzameling van knopen
- Elke toestand komt slechts 1 keer voor
- Meestal te groot om expliciet op te bouwen



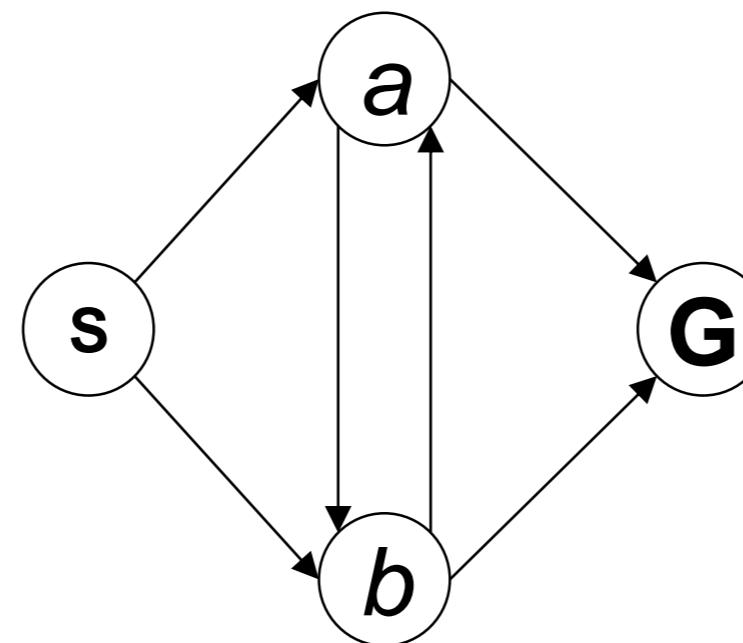
ZOEKBOOM

- Starttoestand is de wortelknoop
- Kinderen stellen transitiefunctie voor
- Knopen stellen toestanden voor, maar corresponderen met een plan om die toestand te bereiken (pad van wortel naar knoop)
- Toestanden kunnen meerdere keren voorkomen
- Meestal te groot om expliciet de hele boom op te bouwen



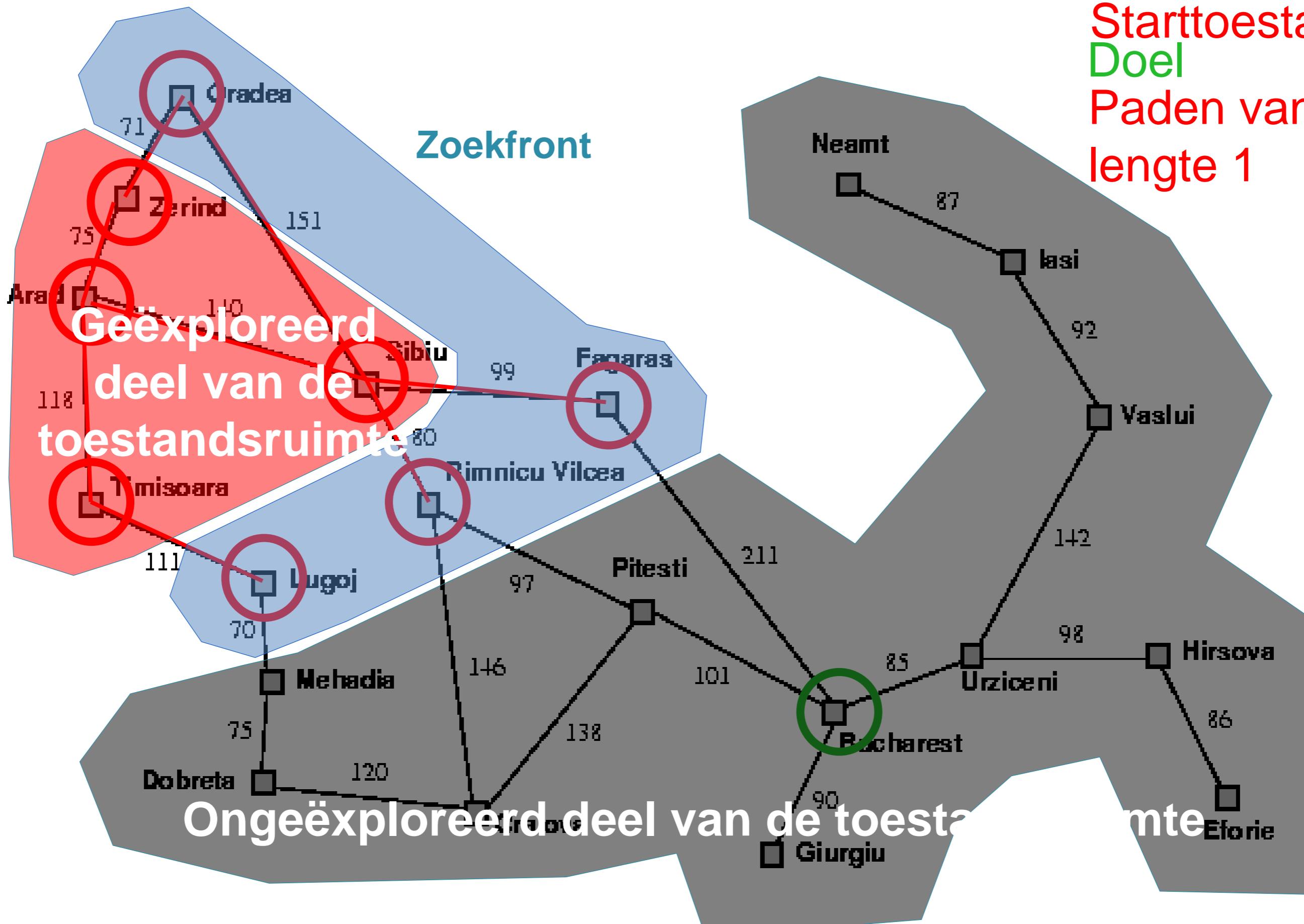
QUIZ

- Beschouw de volgende graaf van de toestandsruimte:



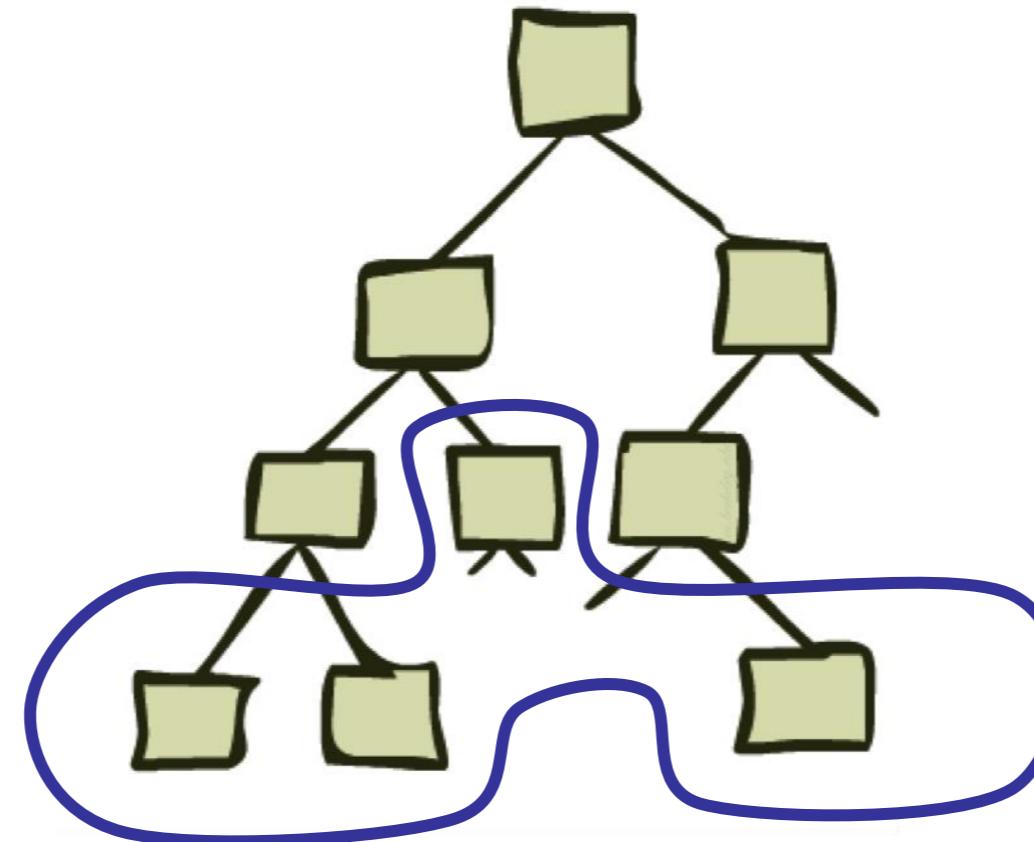
- Hoe groot is de zoekboom ?

Starttoestand
Doel
Paden van
lengte 1



ZOEKBOOM

- Expandeer potentiele plannen (knopen in de boom)
- Hou een zoekfront bij van partiële plannen
- Probeer zo weinig mogelijk knopen te expanderen als mogelijk



ALGEMENE ZOEKBOMEN

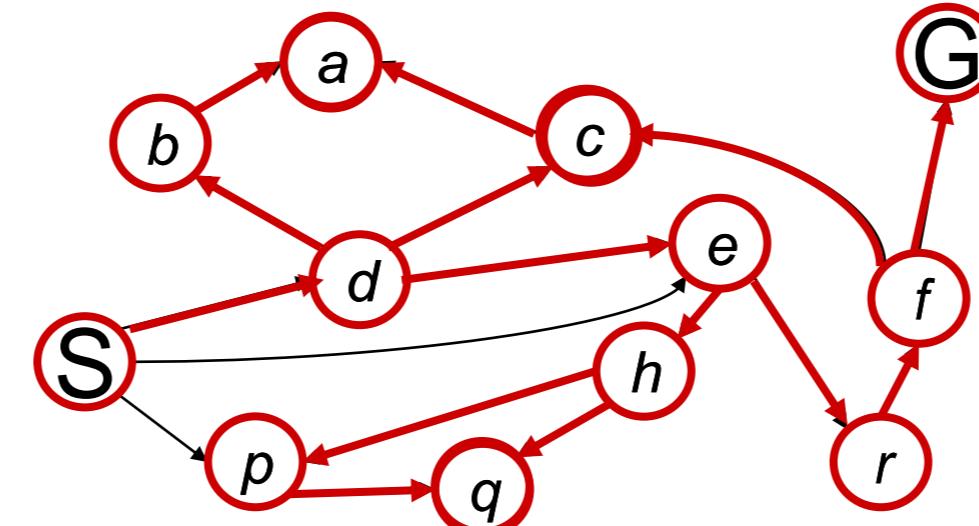
```
function TREE-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier
```

- Belangrijke componenten:
 - Zoekfront
 - Expansie
 - Exploratie-strategie
- Hoofdprobleem: welke knopen uit het zoekfront verder te exploreren ?

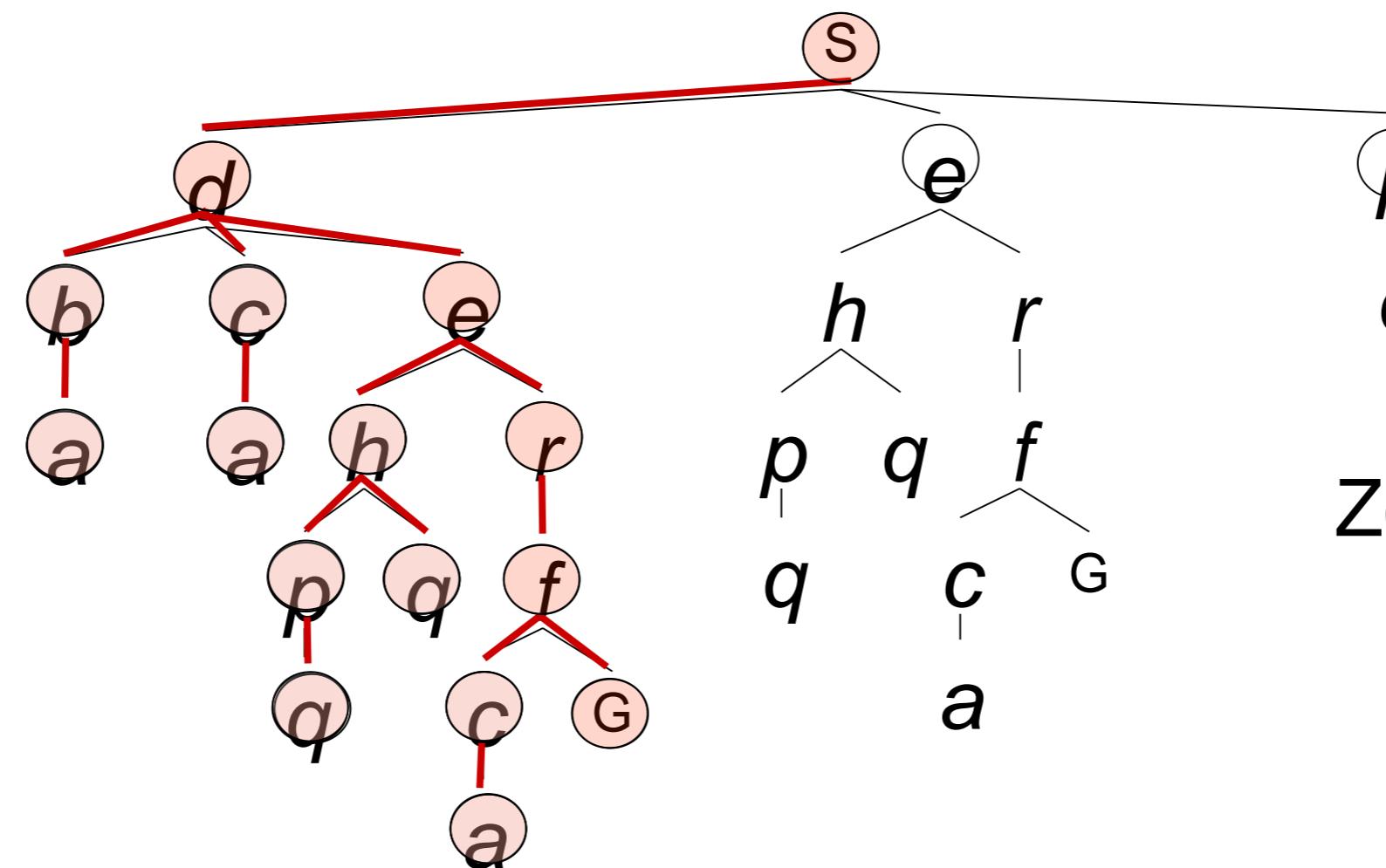
DIEpte-EERst ZOEKEN (DFS)



Expandeer diepste knoop eerst



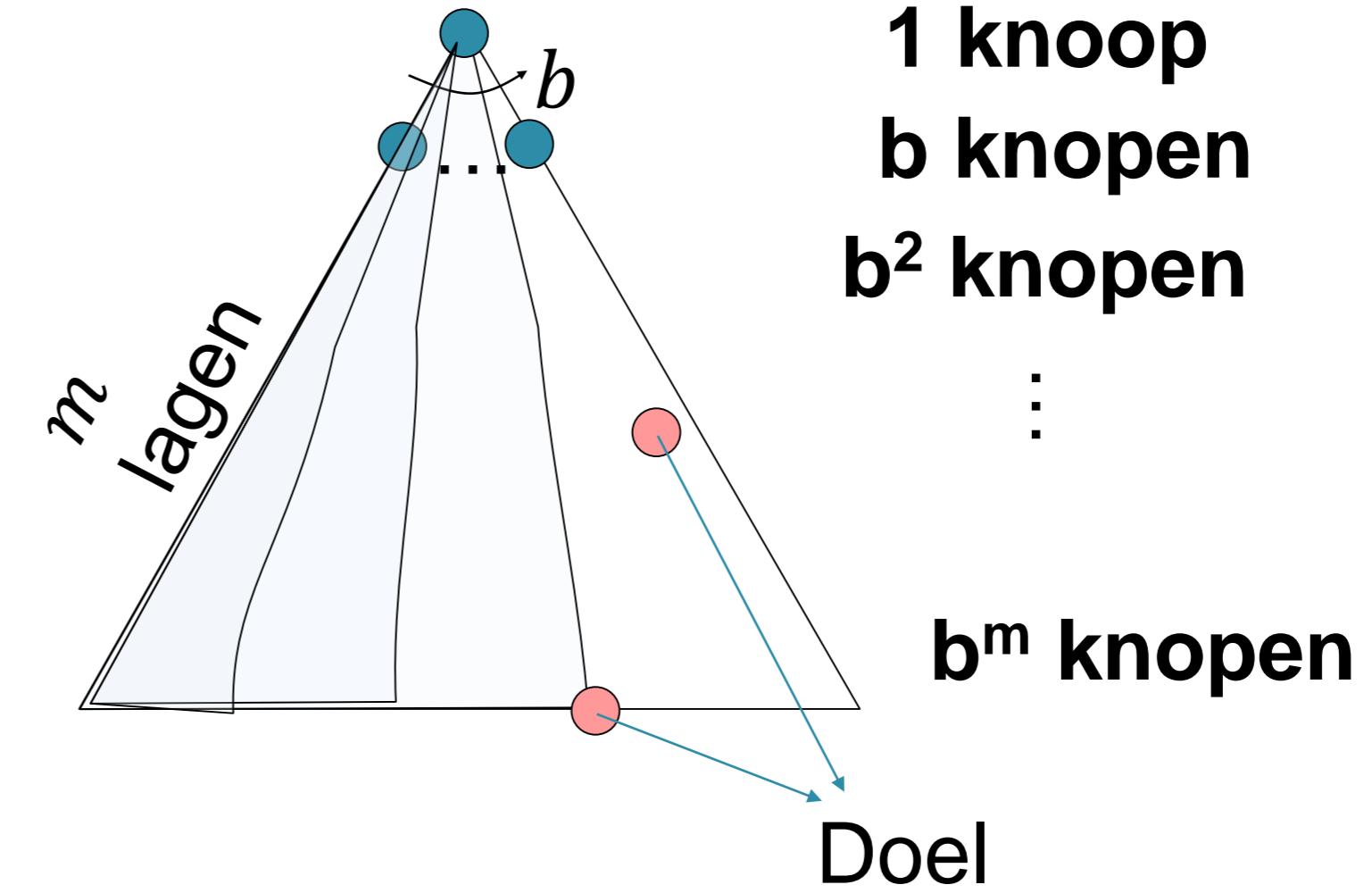
Graaf
toestandsruimte



Zoekboom

DFS: EIGENSCHAPPEN

- Compleet:
→ Vinden we een oplossing als er een is?
- Optimaal:
→ Vinden we het pad met de optimale kost?
- Tijdscomplexiteit?
- Geheugencomplexiteit?



b : branching
factor

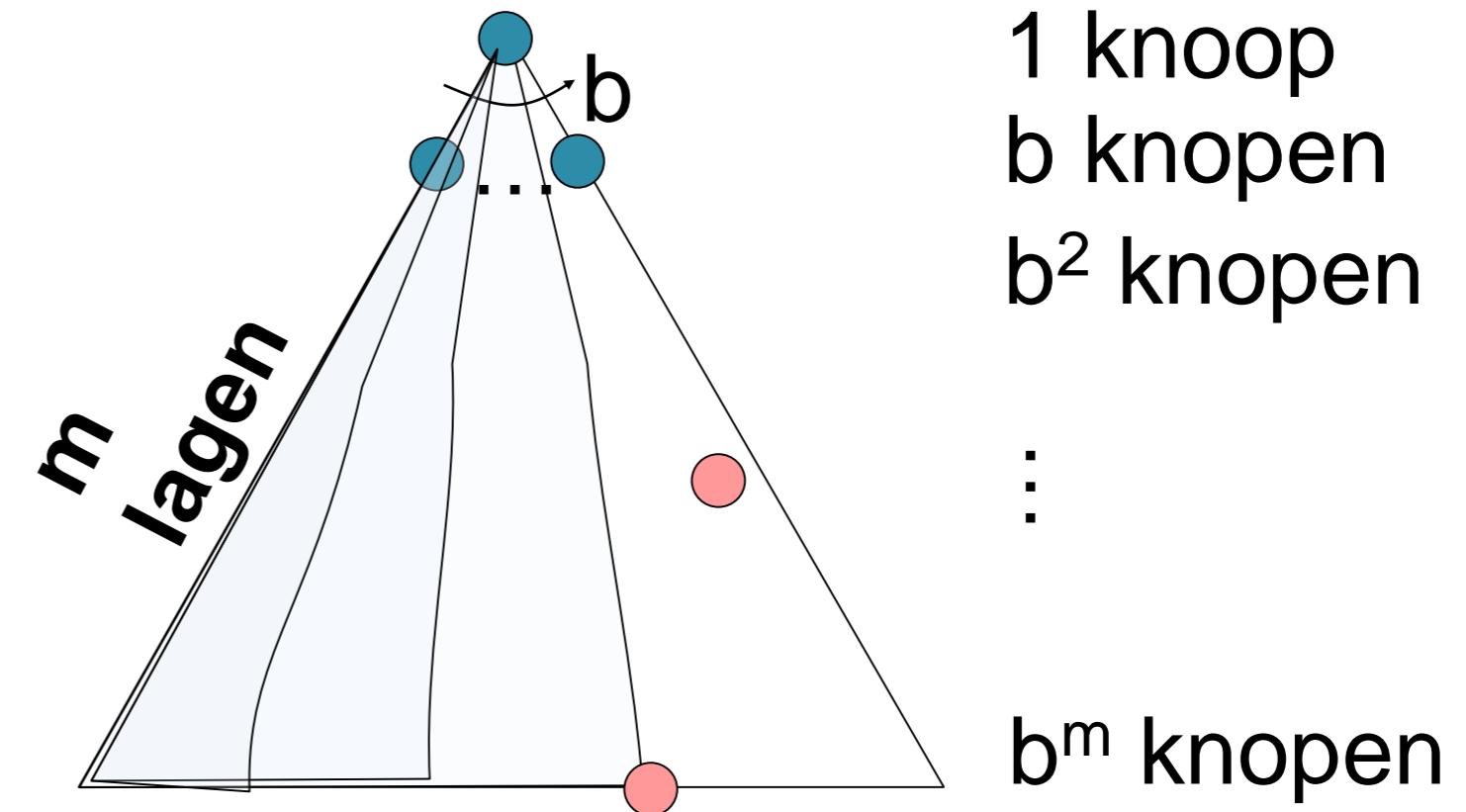
DFS: EIGENSCHAPPEN

Tijdscomplexiteit

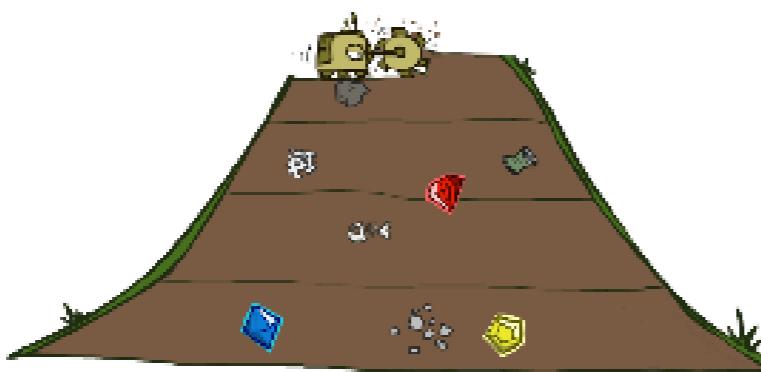
- Welke knopen expandeert DFS ?
 - Linker gedeelte van de boom
 - In het slechtste geval de hele boom!
 - Als m eindig is, orde $O(b^m)$

Geheugencomplexiteit

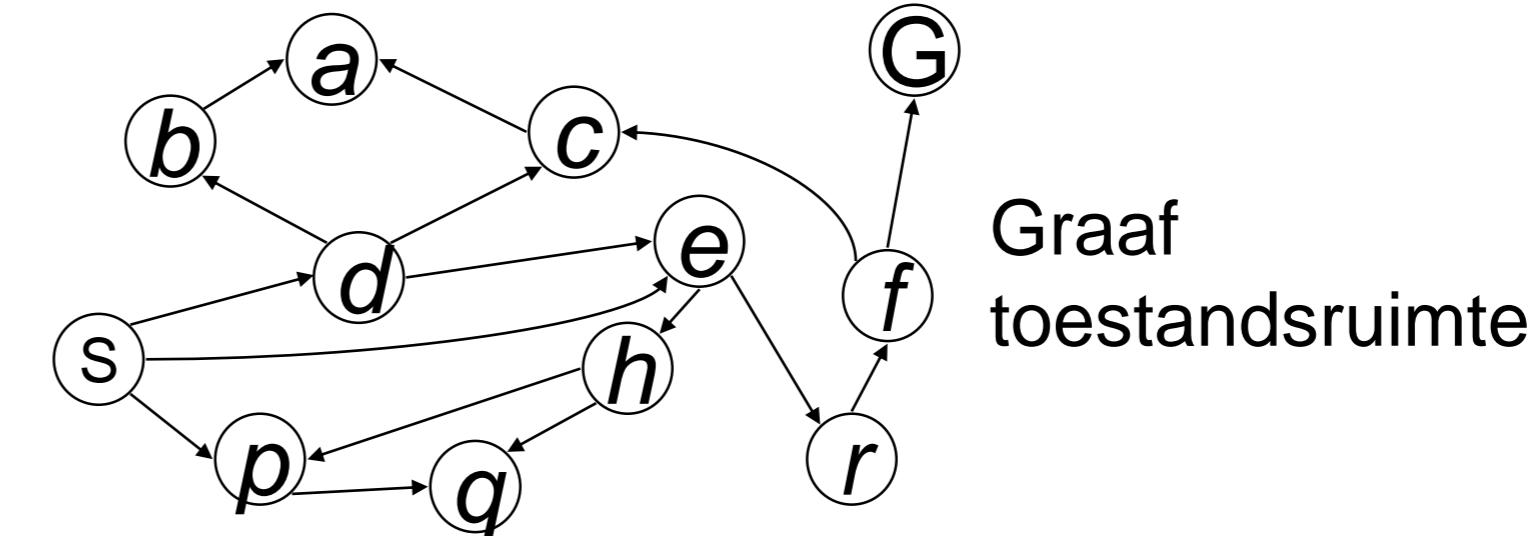
- Grootte van het zoekfront
 - Alle “siblings” van de wortel tot zoekdiepte $m = O(bm)$



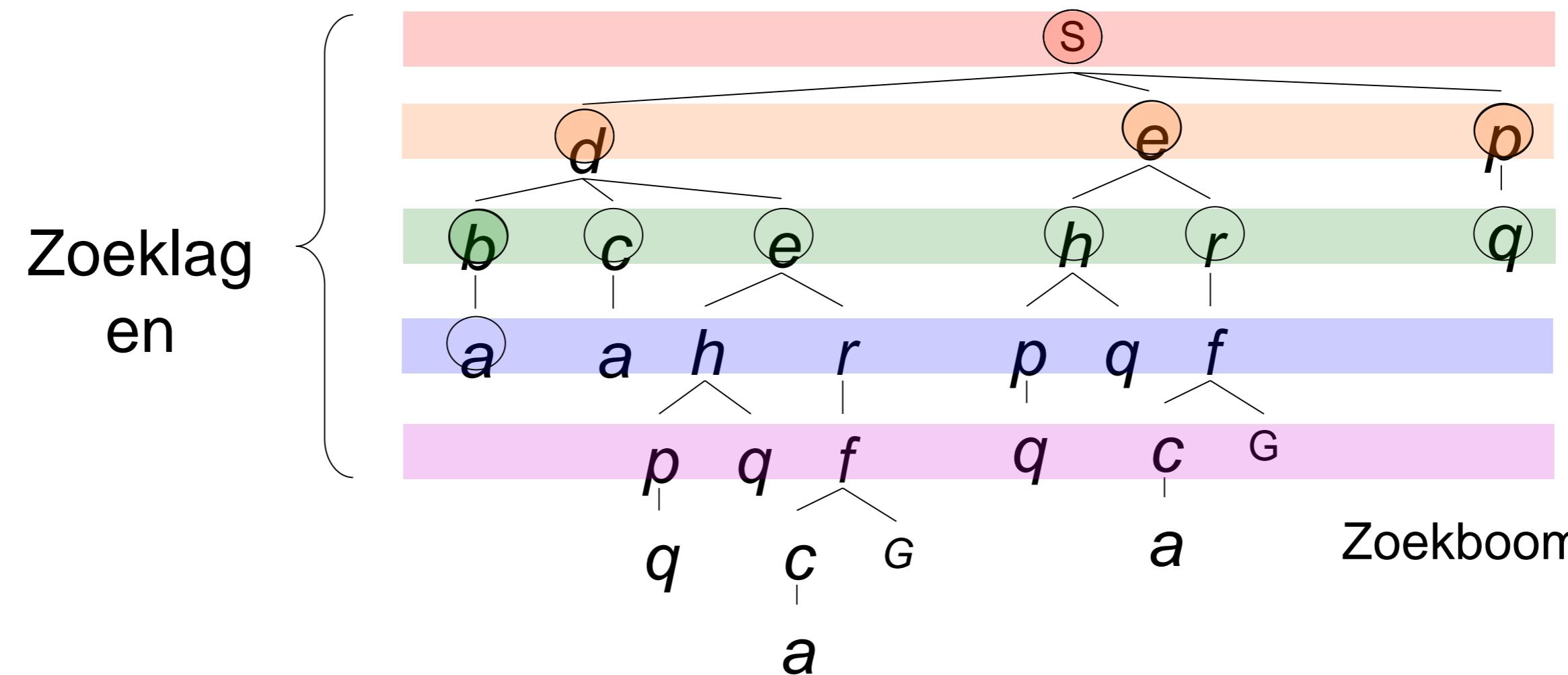
BREEDTE-EERST ZOEKEN (BFS)



Expandeer ondiepste
knoop eerst



Graaf
toestandsruimte



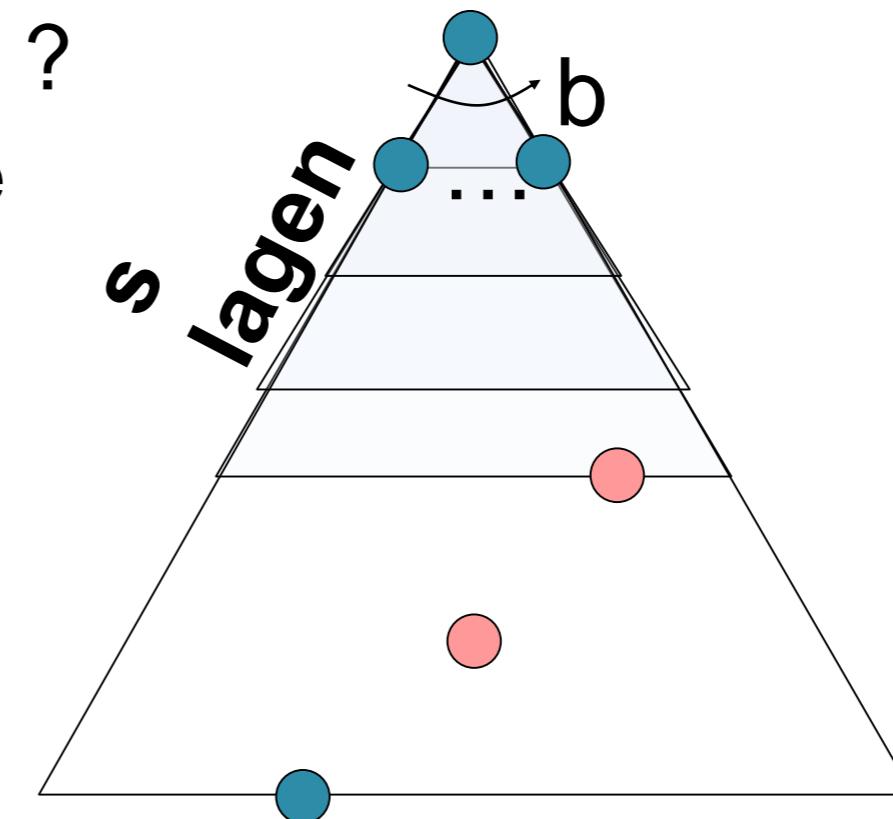
BFS: EIGENSCHAPPEN

Tijdscomplexiteit

- Welke knopen expandeert BFS ?
 - Alle knopen boven de minste diepe oplossing (s lagen)
 - Orde $O(b^s)$

Geheugencomplexiteit

- Grootte van het zoekfront
 - Gedomineerd door de diepste laag, dus $O(b^s)$

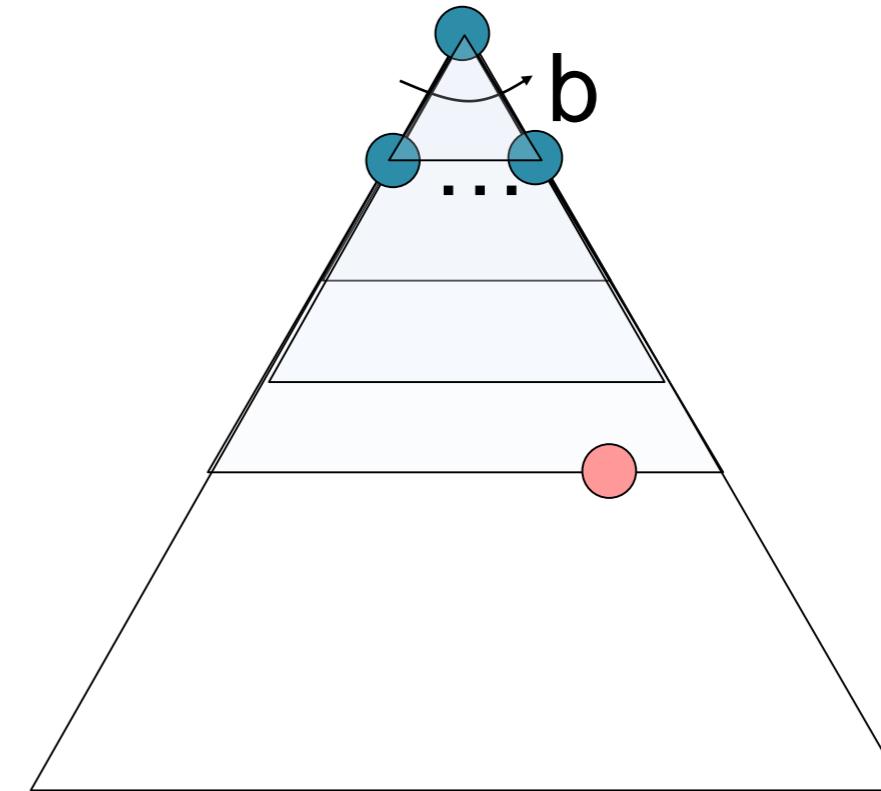


1 knoop
b knopen
 b^2 knopen
 b^s knopen
 b^m knopen

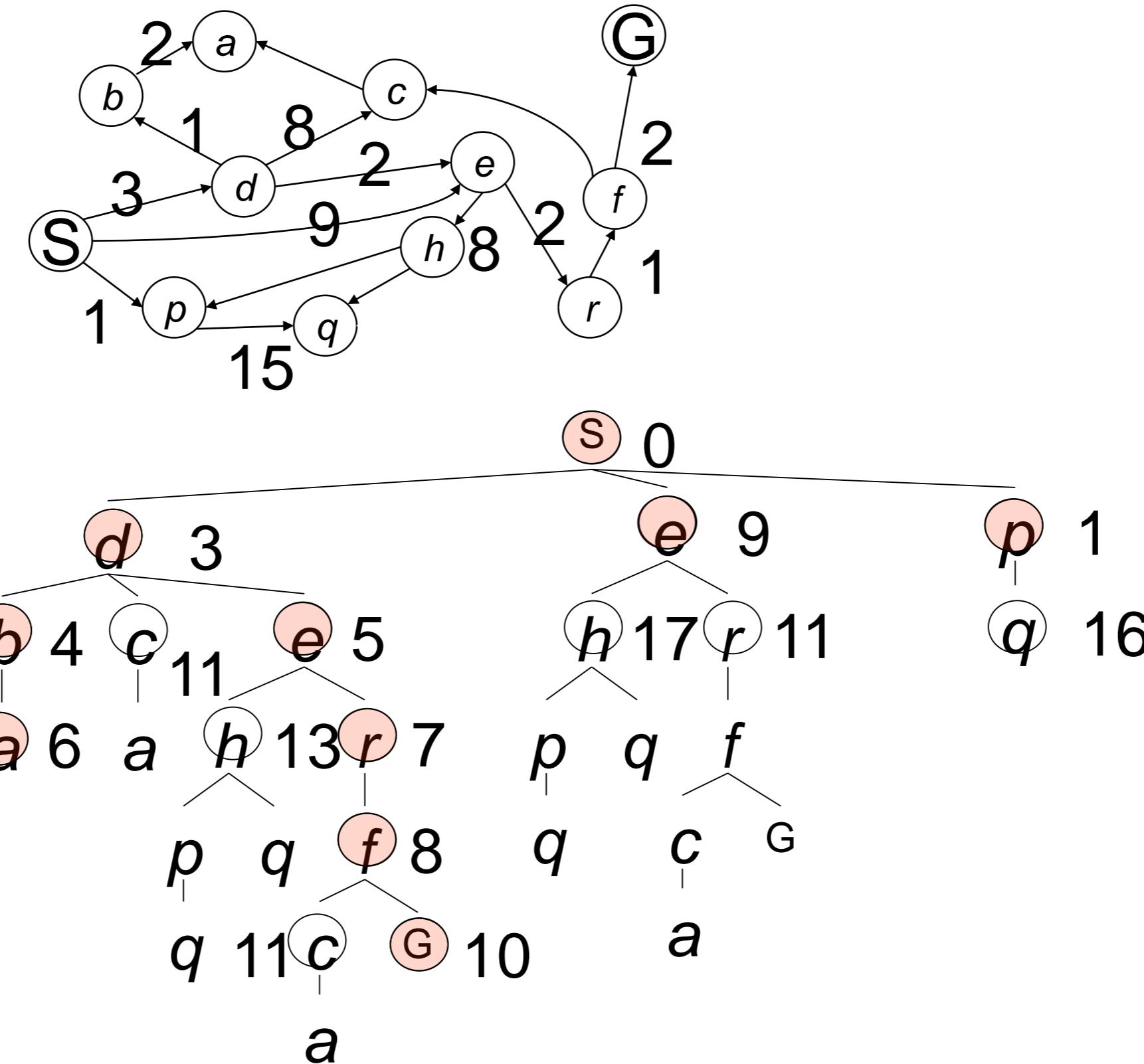
Compleet, optimaal ?

ITERATIEF VERDIEPEN (IDA)

- Motivatie:
 - Combineer geheugencomplexiteit van DFS met tijdscomplexiteit van BFS:
 - Loop DFS met diepte-limiet 1 als geen oplossing...
 - Loop DFS met diepte-limiet 2 als geen oplossing...
 - Loop DFS met diepte-limiet 3...
- Overhead
 - Zoektijd wordt gedomineerd door diepste laag



COST-SENSITIVE SEARCH UNIFORM COST SEARCH (CHEAPEST FIRST, UCS)



UCS: EIGENSCHAPPEN

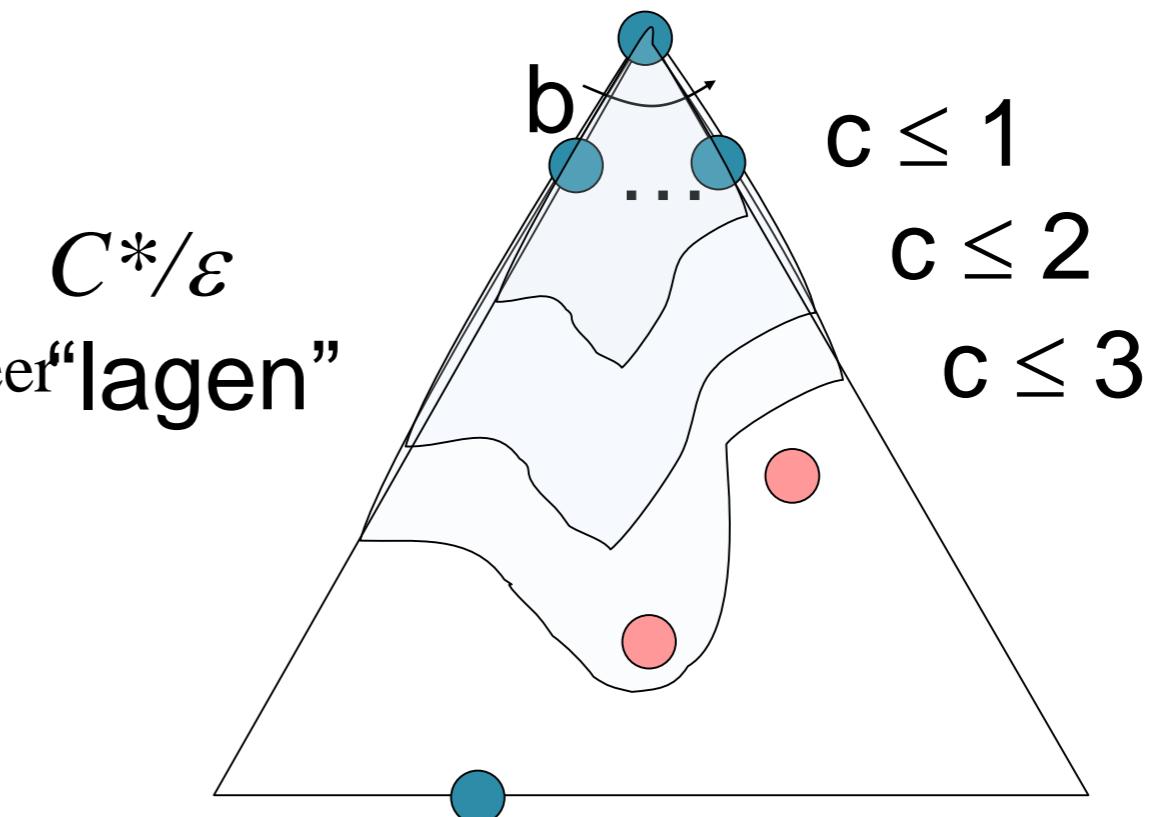
Tijdscomplexiteit

- Welke knopen expandeert UCS ?
 - Alle knopen met kost minder dan de goedkoopste oplossing
 - Als deze oplossing C^* kost en elke boog minstens ε , dan is de “effectieve” diepte ongeveer “lagen” C^*/ε
 - Totale tijd = orde $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$
(exponentieel in effectieve diepte)

Geheugencomplexiteit

- Grootte van het zoekfront
 - Ongeveer de grootte van de laatste laag = $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$

Compleet, optimaal ?

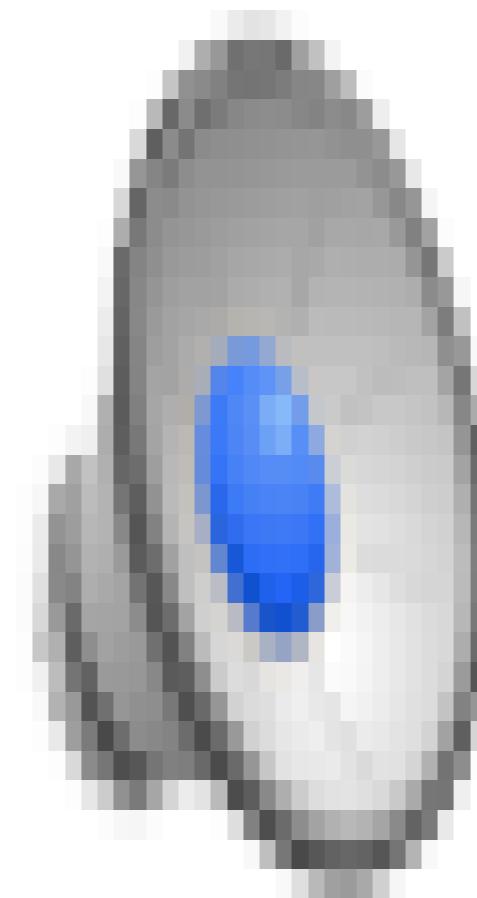


VERGELIJKING ZOEKBOMEN

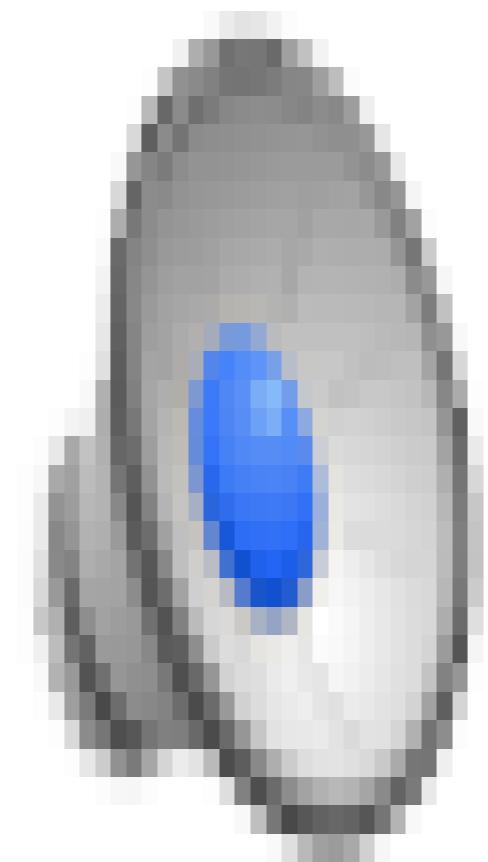
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes ^a	Yes ^{a,b}	No	No	Yes ^a	Yes ^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes ^c	Yes	No	No	Yes ^c	Yes ^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; ℓ is the depth limit. Superscript caveats are as follows: ^a complete if b is finite; ^b complete if step costs $\geq \epsilon$ for positive ϵ ; ^c optimal if step costs are all identical; ^d if both directions use breadth-first search.

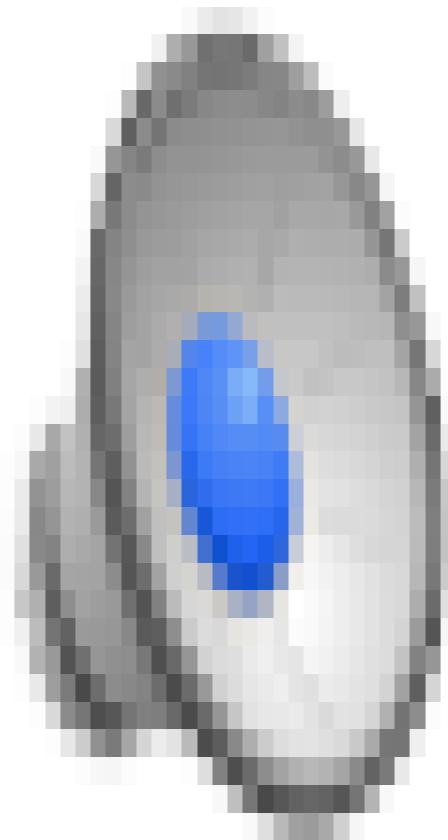
QUIZ: DFS, BFS, UCS



QUIZ: DFS, BFS, UCS



QUIZ: DFS, BFS, UCS



SAMENVATTING

- Al deze zoekalgoritmen zijn hetzelfde behalve de manier waarop ze het zoekfront expanderen:
 - Conceptueel zijn alle zoekfronten “priority queues” (verzameling van knopen met prioriteiten)
 - Voor DFS en BFS kan overhead vermeden worden door gewone stacks (LIFO) of queues (FIFO) te gebruiken
- Zoekfunctie modelleert de omgeving
 - De agent probeert niet alle plannen in de echte wereld uit, maar doet alles in simulatie, en voert dan het gekozen plan uit

OVERZICHT

- Planning agents
- Zoekproblemen
- Zoekstrategieën:
 - Uninformed search
 - Diepte-eerst, breedte-eerst, uniforme cost
 - **Informed search**
 - Heuristieken, greedy search, A*
 - Graph search

VOORBEELD: PANNENKOEKEN SORTEREN

BOUNDS FOR SORTING BY PREFIX REVERSAL

William H. GATES

Microsoft, Albuquerque, New Mexico

Christos H. PAPADIMITRIOU*†

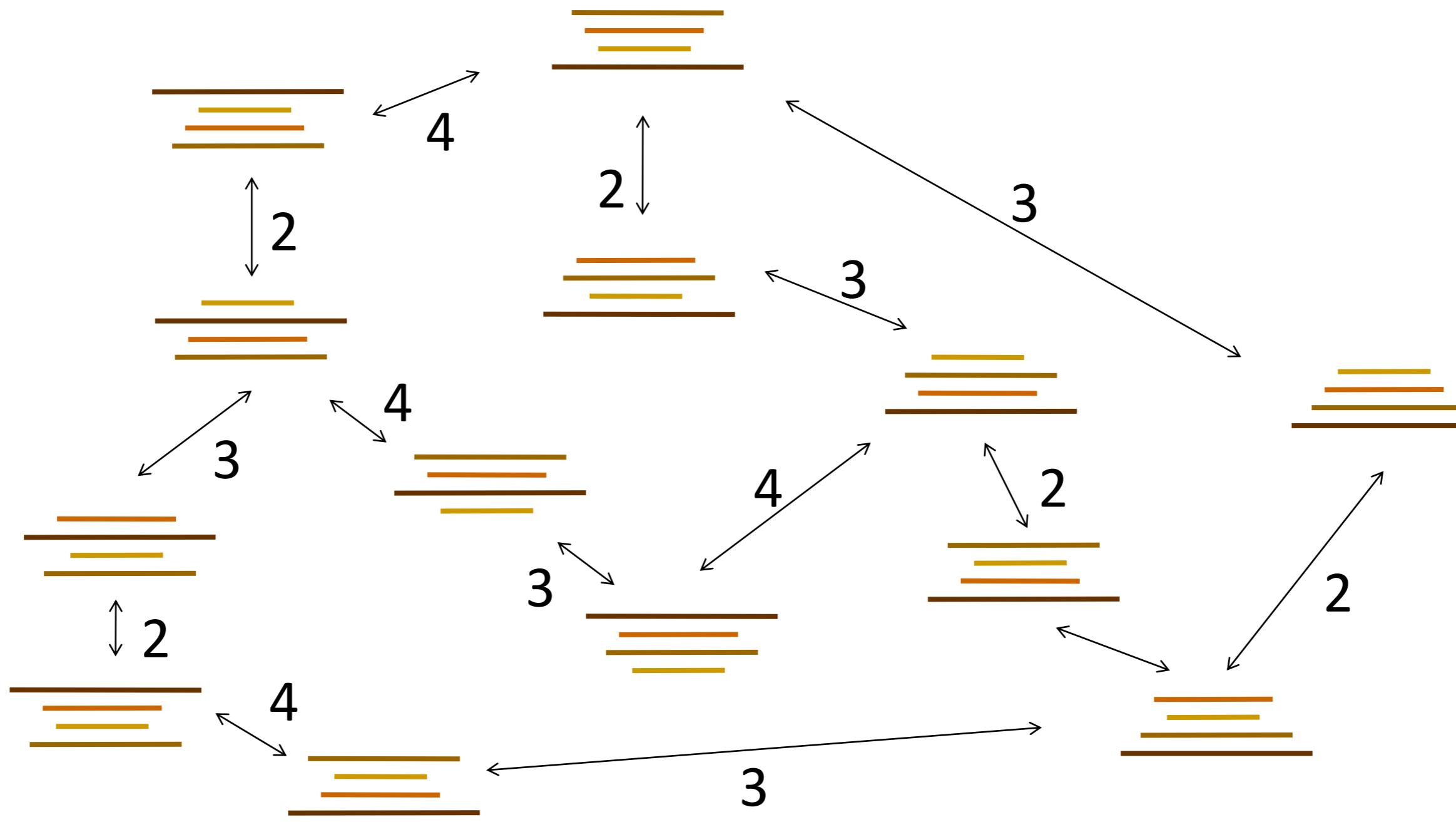
Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.

Received 18 January 1978

Revised 28 August 1978

For a permutation σ of the integers from 1 to n , let $f(\sigma)$ be the smallest number of prefix reversals that will transform σ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all σ in (the symmetric group) S_n . We show that $f(n) \leq (5n+5)/3$, and that $f(n) \geq 17n/16$ for n a multiple of 16. If, furthermore, each integer is required to participate in an even number of reversed prefixes, the corresponding function $g(n)$ is shown to obey $3n/2 - 1 \leq g(n) \leq 2n + 3$.

TOESTANDSRUIMTE



TREE SEARCH

function TREE-SEARCH(*problem*) **returns** a solution, or failure

 initialize the frontier using the initial state of *problem*

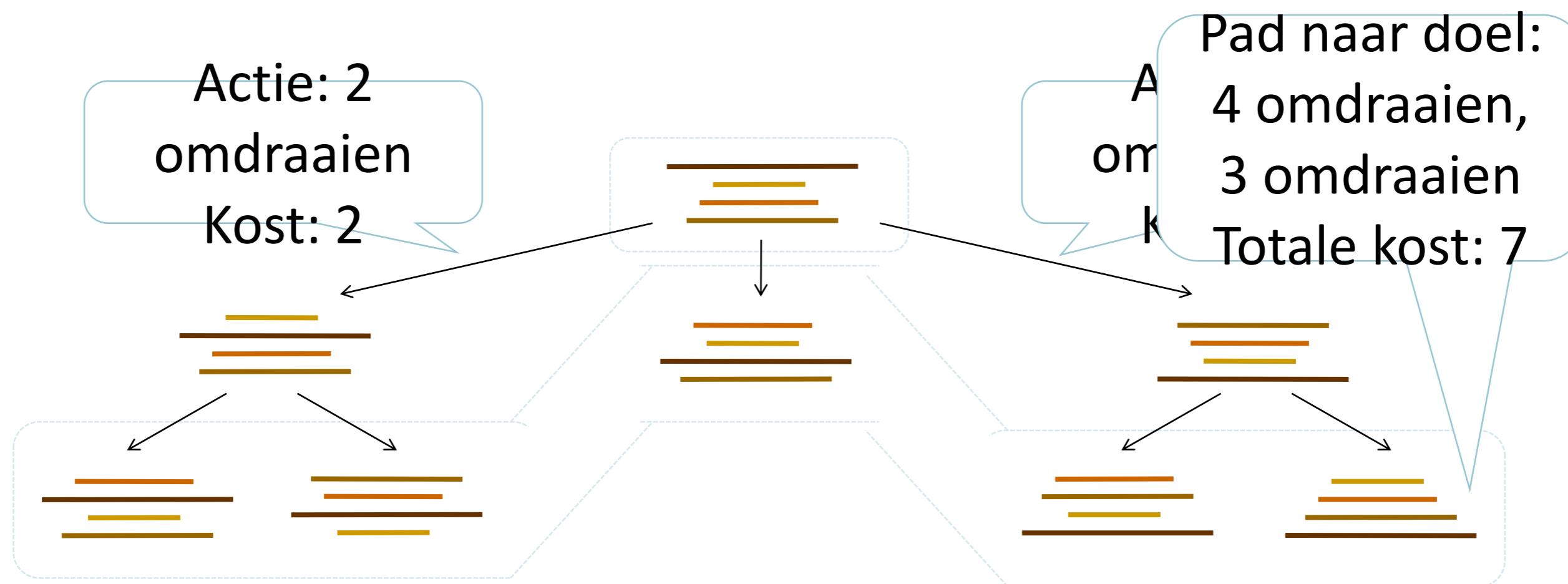
loop do

if the frontier is empty **then return** failure

 choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the corresponding solution

 expand the chosen node, adding the resulting nodes to the frontier



UNIFORM COST SEARCH

Strategie

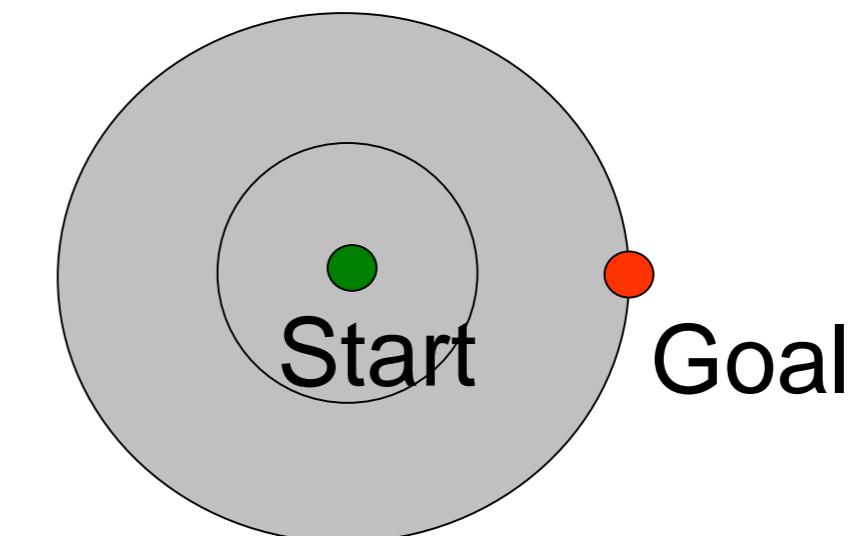
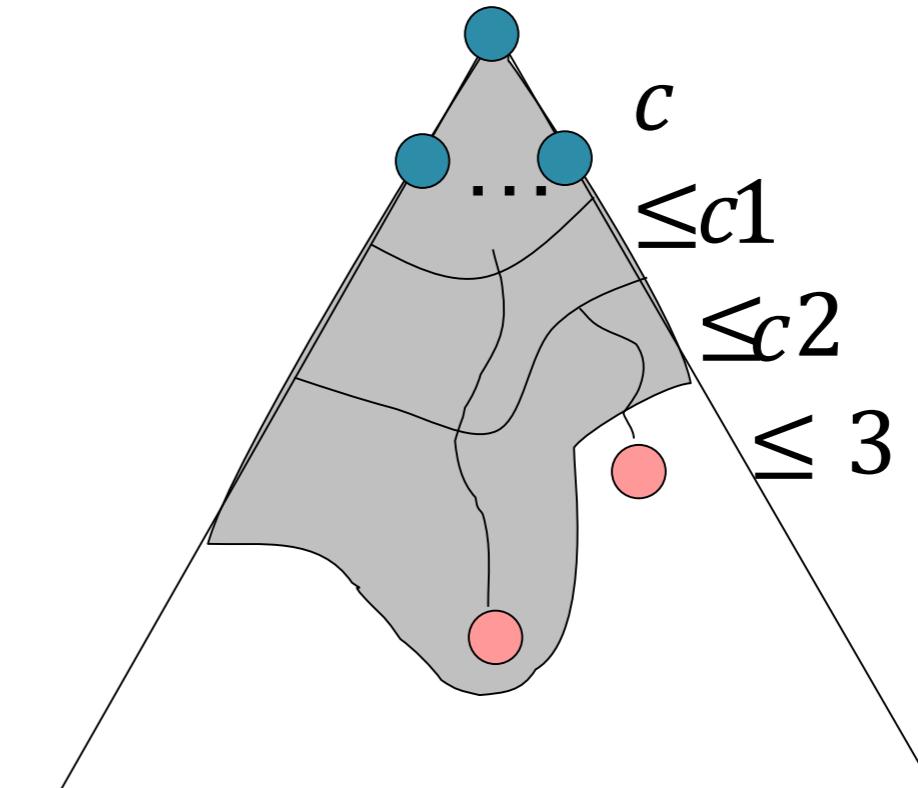
- Expandeer het goedkoopste pad

Voordeel

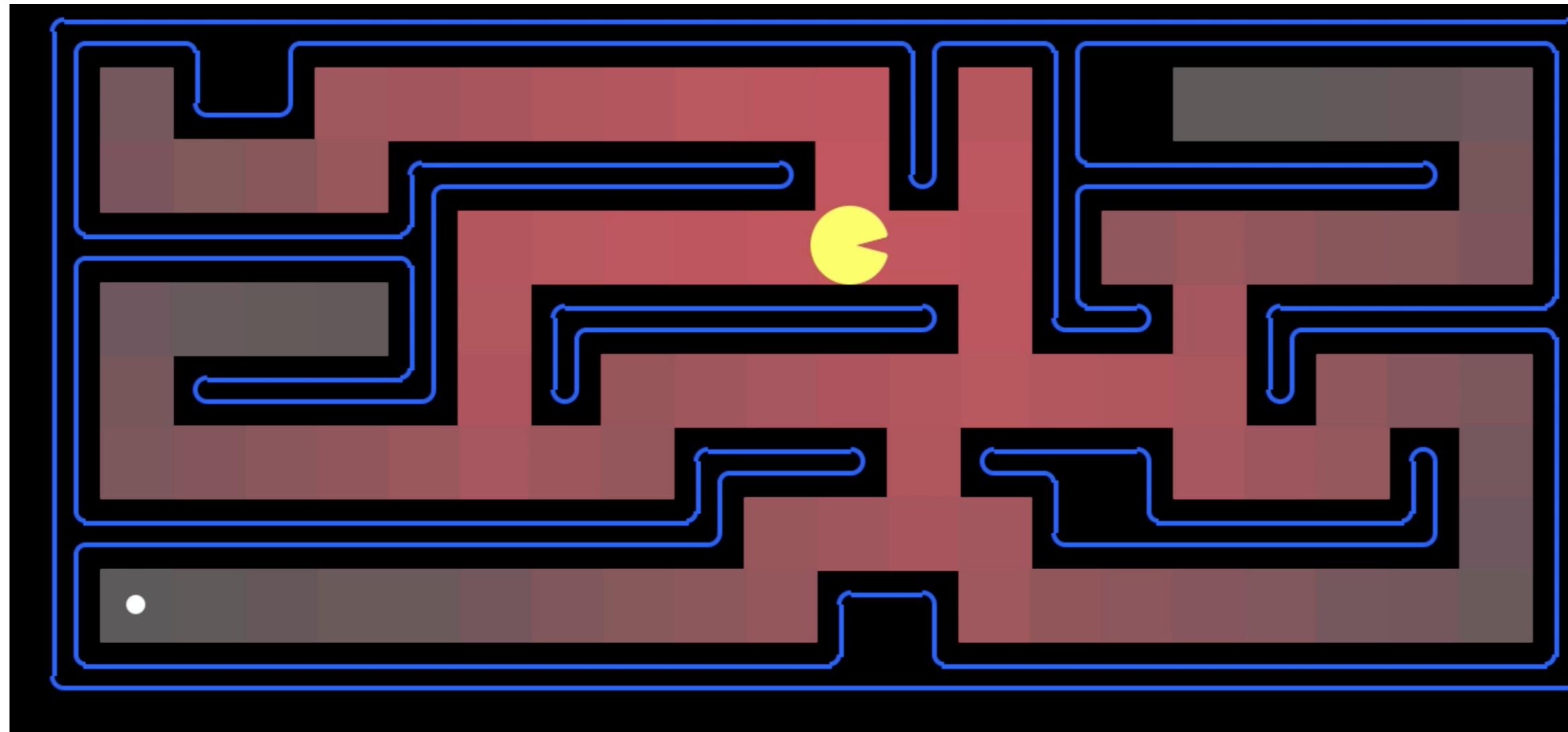
- Compleet en optimaal

Nadeel

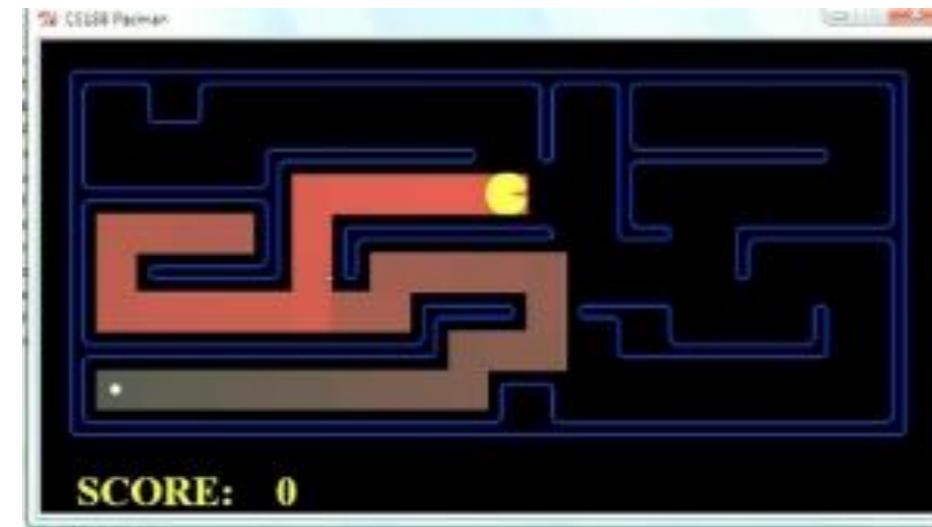
- Exploreert in alle richtingen, geen informatie over welke richting “goed” is om naar het doel te geraken



VOORBEELD: PACMAN WERELD (UCS)



VERGELIJKING TUSSEN GREEDY, UCS EN A*



Greedy (snel, niet optimaal)



Uniform Cost (traag, wel optimaal)



A* (snel, optimaal)

INFORMED SEARCH

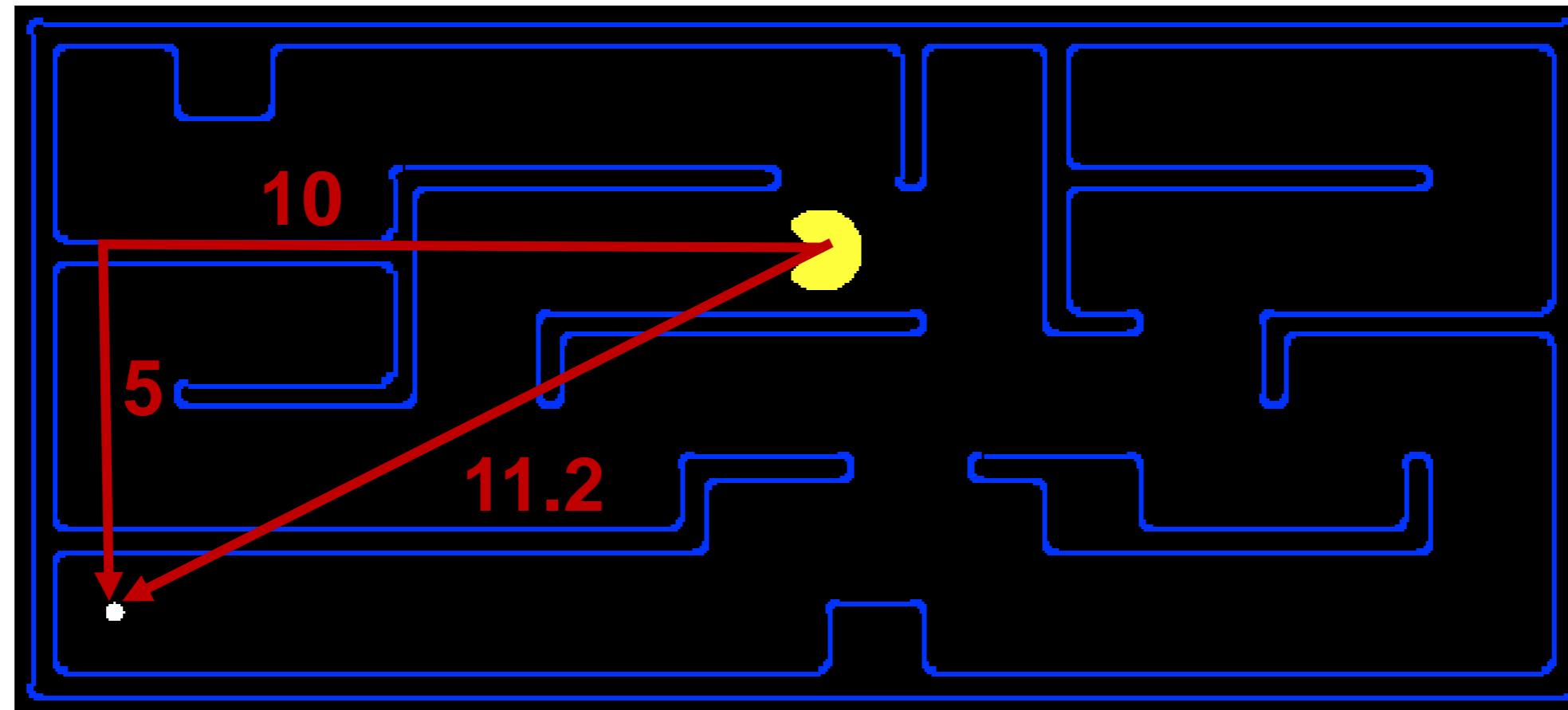
- Kunnen we extra informatie gebruiken die ons sneller naar een doeltoestand breng



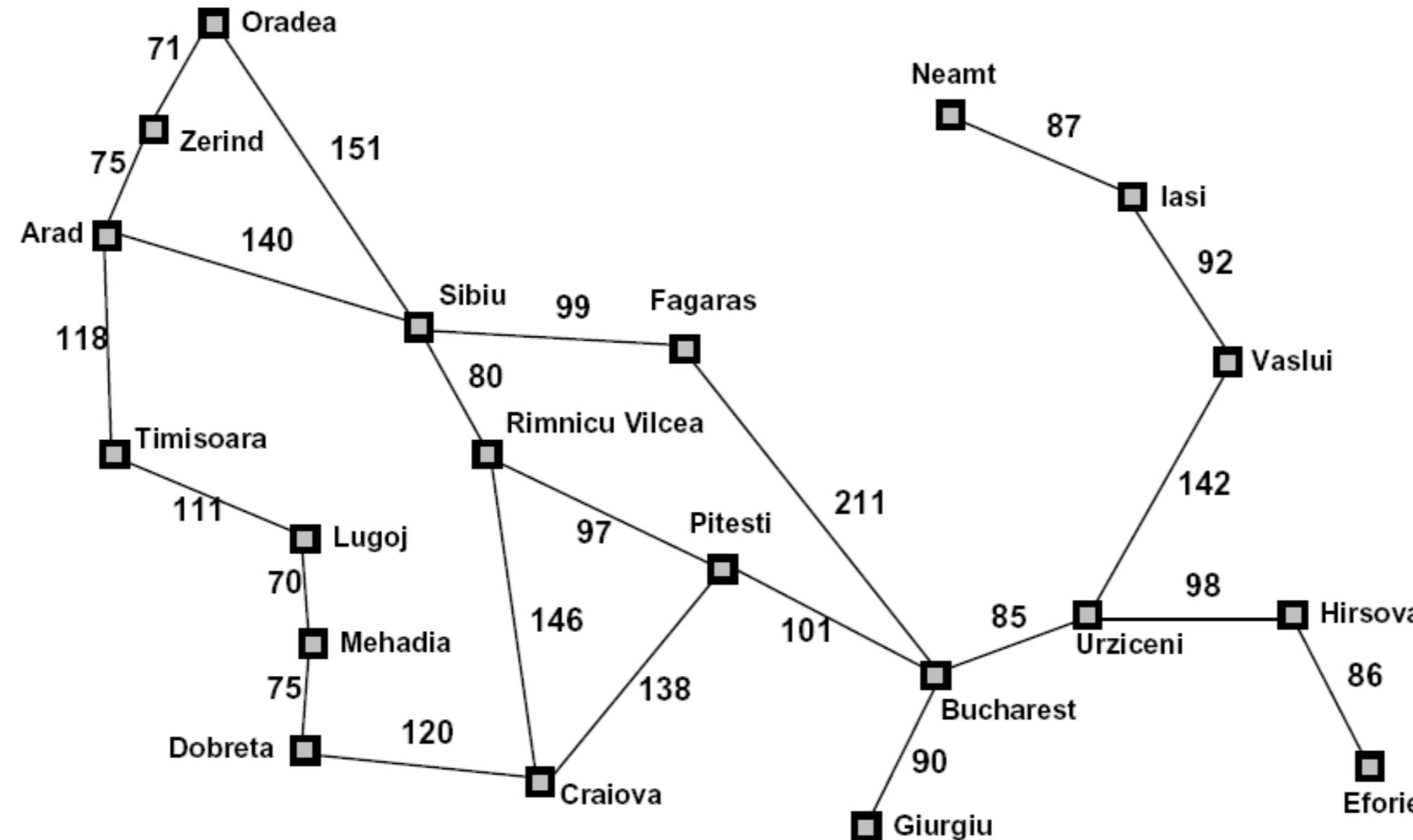
ZOEKHEURISTIEKEN

Zoekheuristiek = een functie die schat hoe dicht een toestand in de buurt van een doeltoestand is

- Specifiek voor elk zoekprobleem
- Voorbeeld: Manhattan afstand, Euclidische afstand (kortste afstand in vogelvlucht) tot het doel



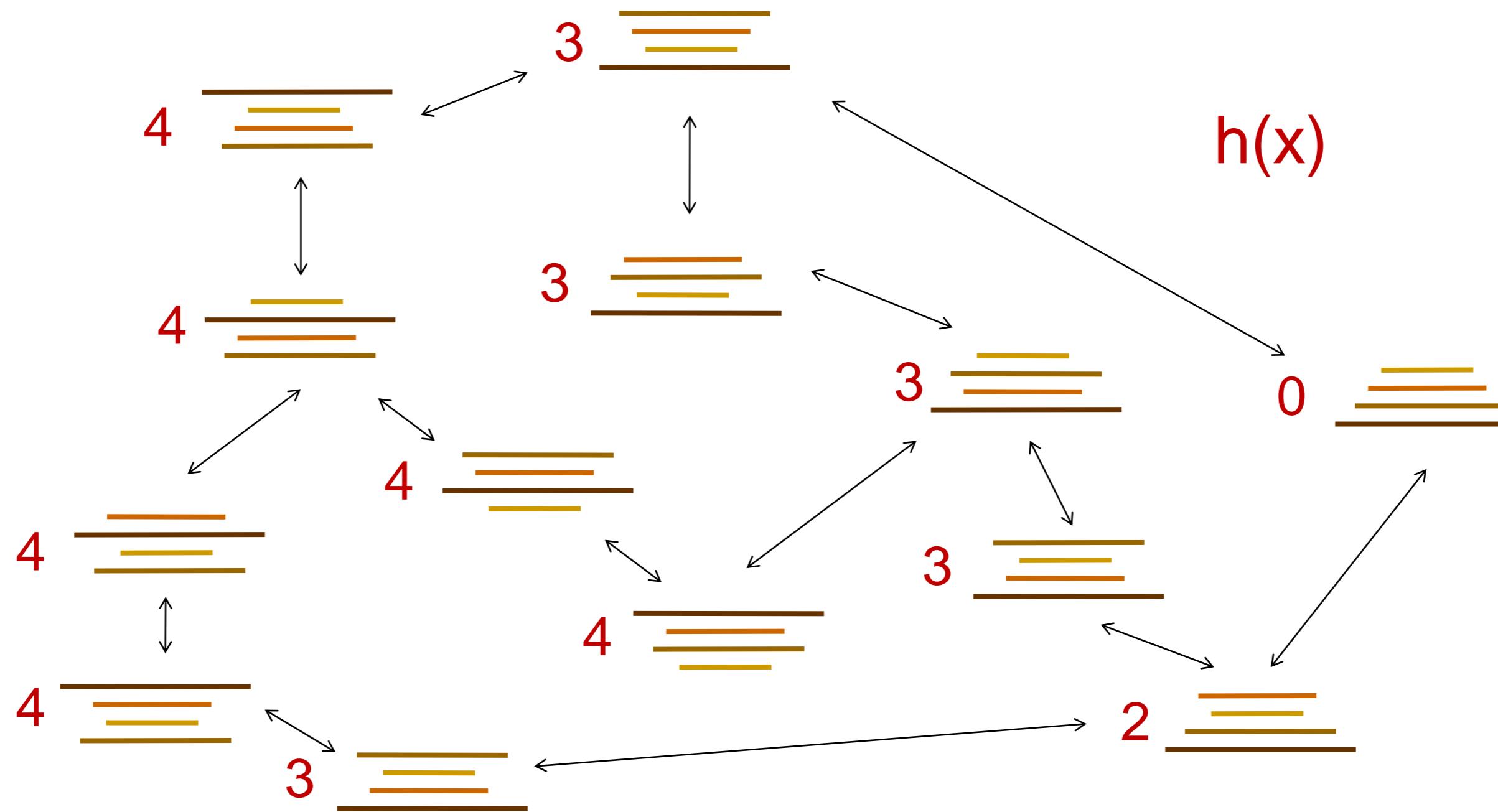
VOORBEELD: ROUTEPLANNING



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

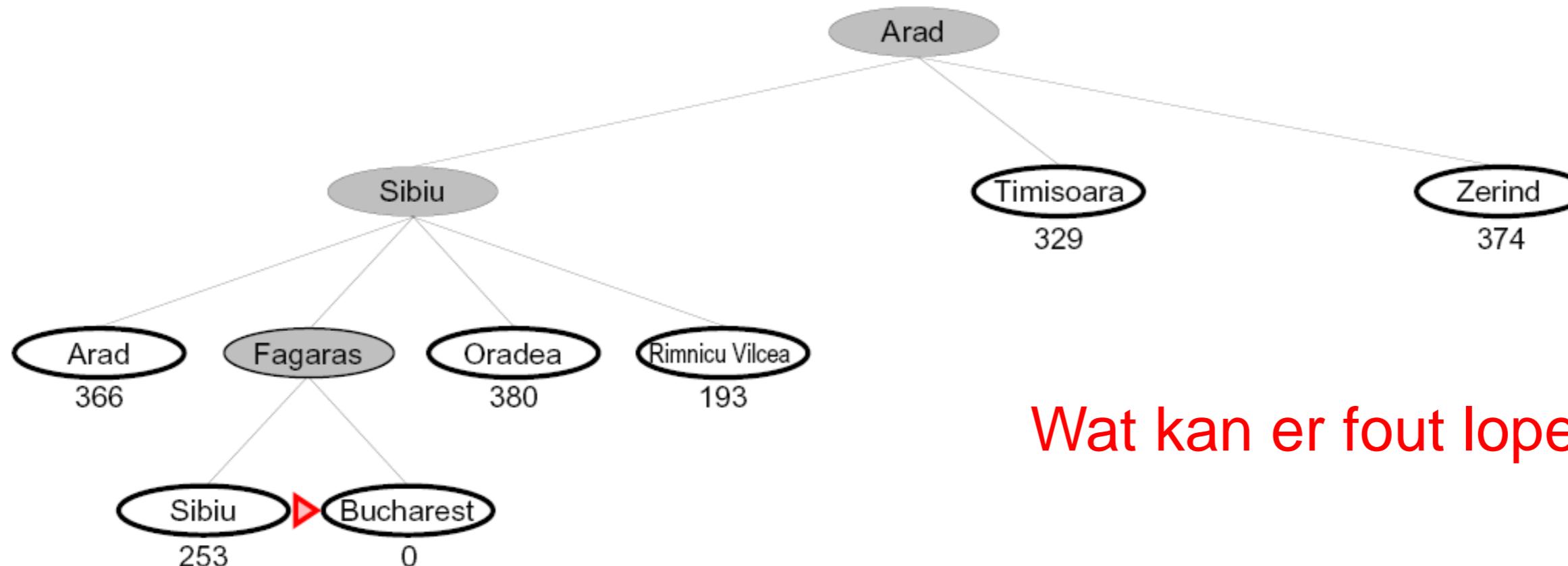
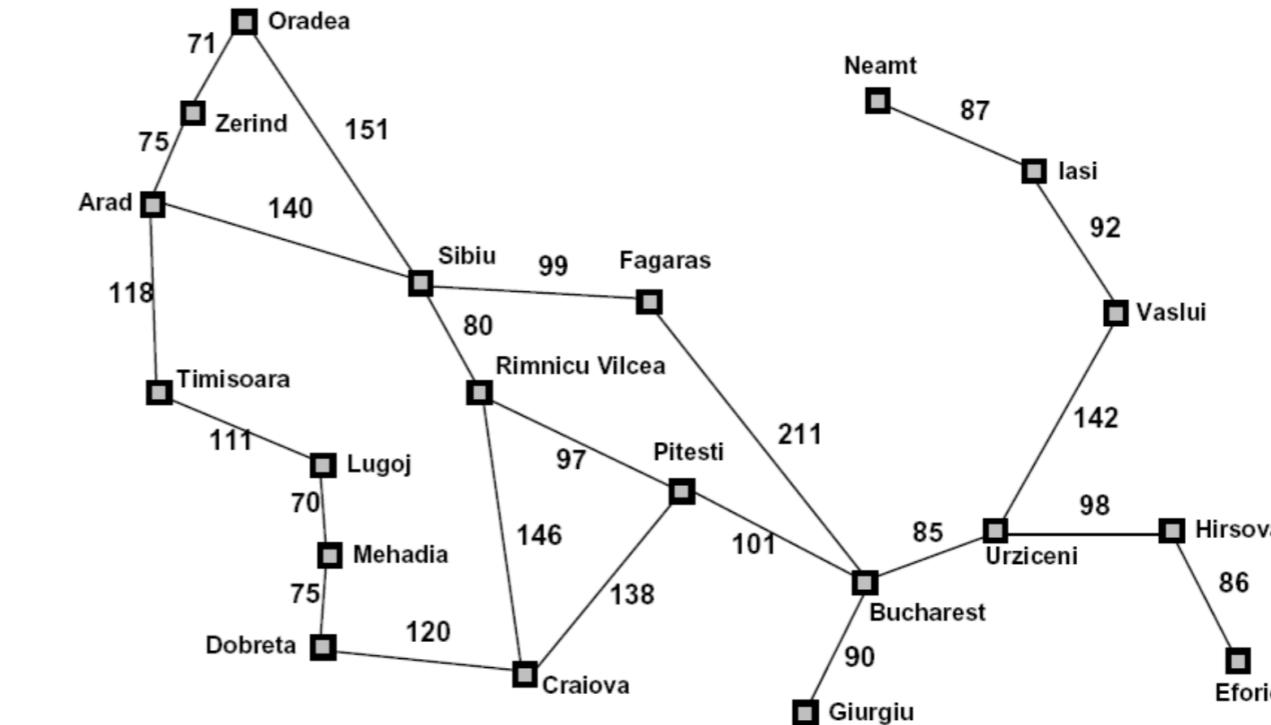
$h(x)$

VOORBEELD: DISCRETE OPTIMALISATIE



GREEDY SEARCH

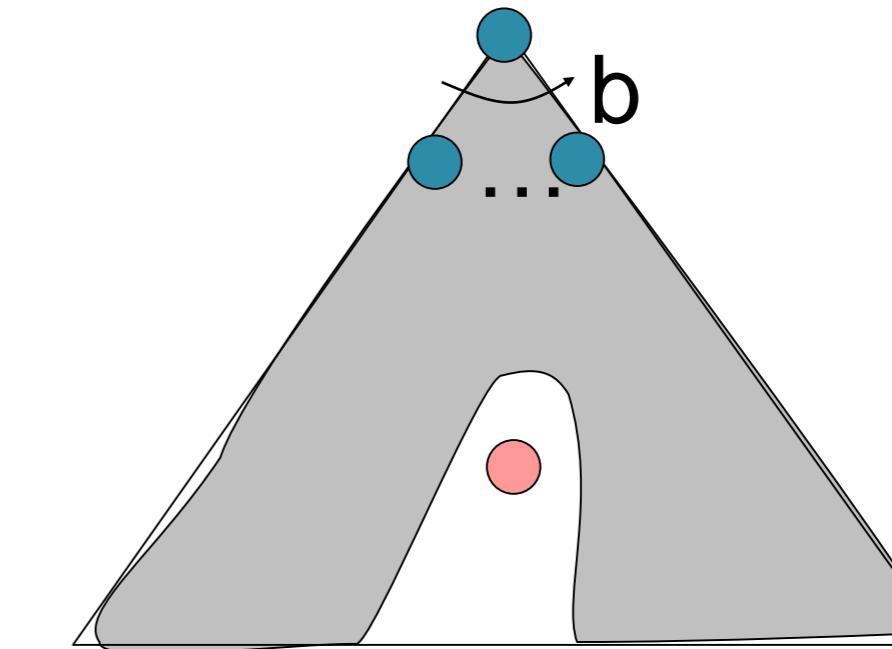
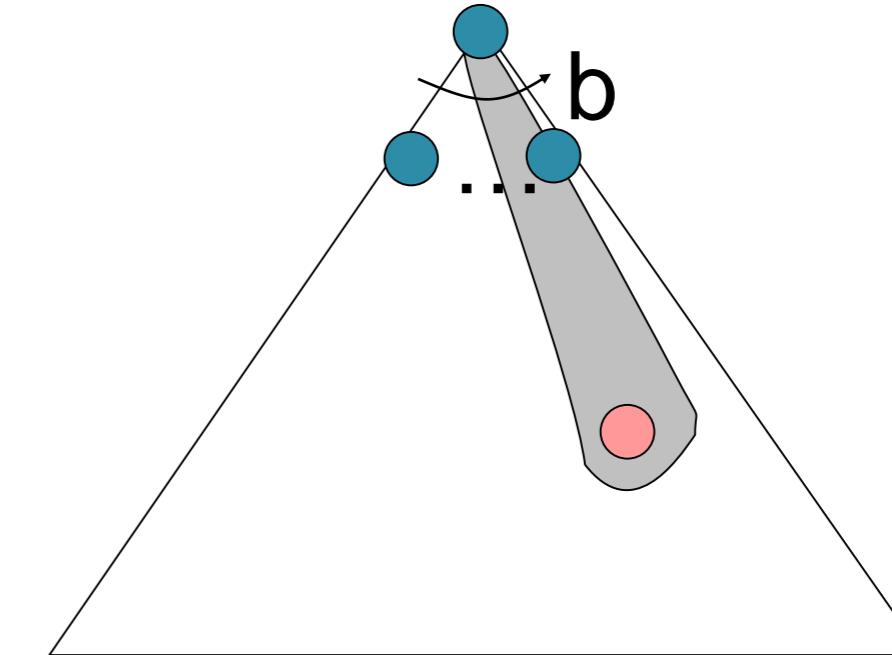
Expandeer de knoop die het dichtst bij de doeltoestand lijkt



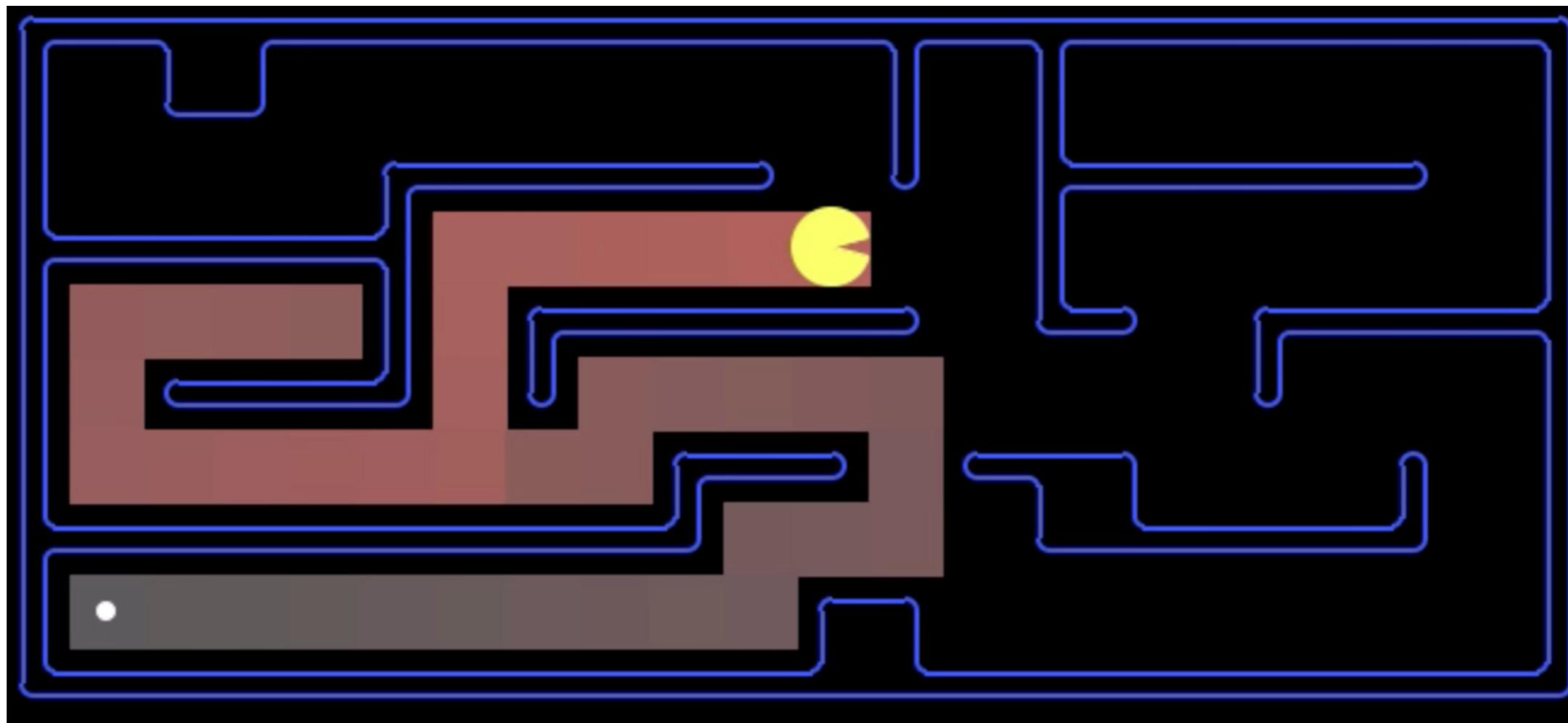
Wat kan er fout lopen ?

GREEDY SEARCH: EIGENSCHAPPEN

- **Strategie:**
 - expandeer de node waarvan je denkt dat hij het dichtst bij de doeltoestand is
- **Heuristiek:**
 - schatting van de afstand tot de meest nabije doeltoestand voor elke toestand
- Beste geval: recht naar het doel
- Meest voorkomend geval: recht naar het (verkeerde) doel
- Worst-case: slecht geïnformeerd DFS



VOORBEELD: PACMAN WERELD

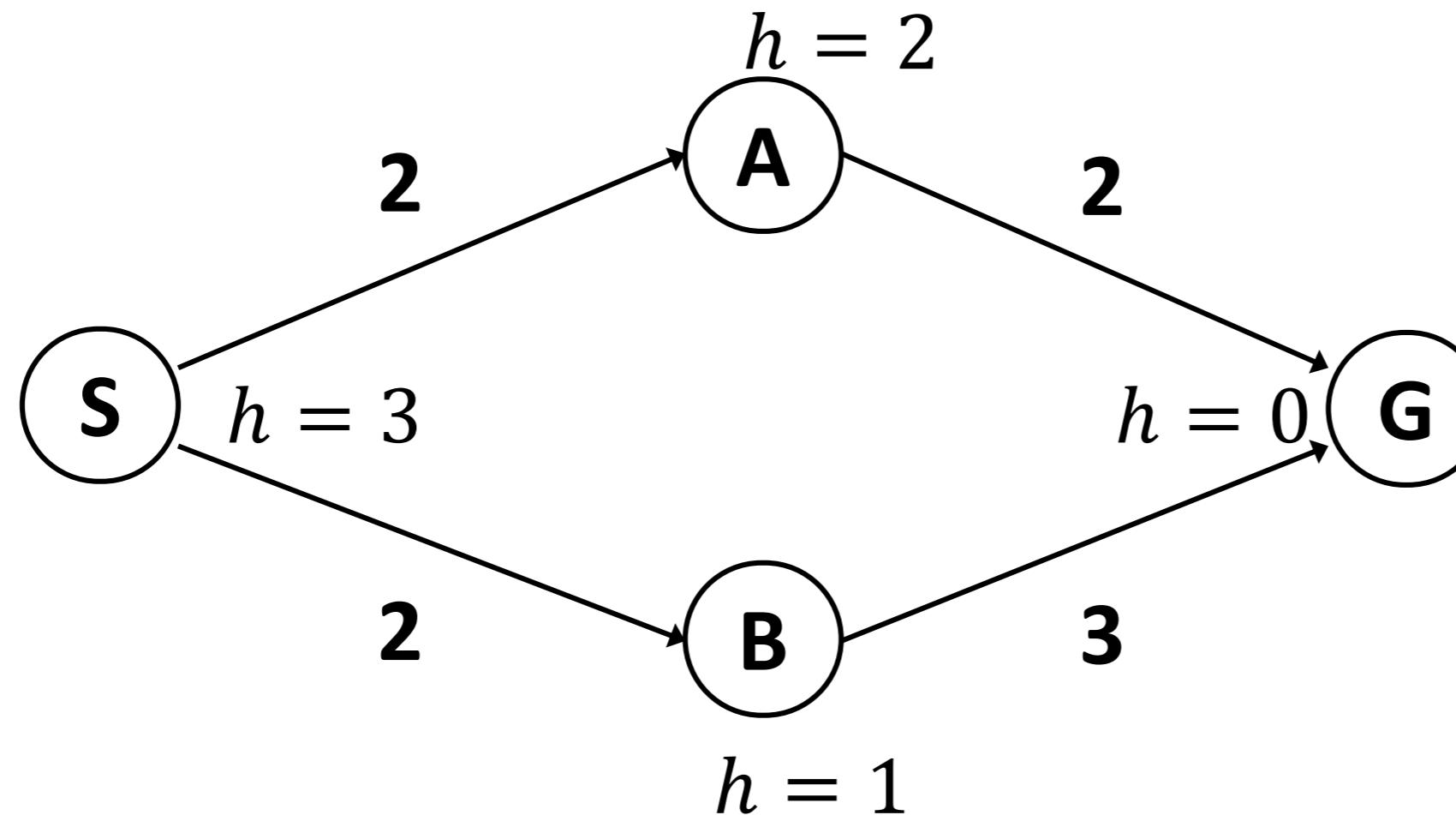


HET BESTE VAN 2 WERELDEN: A*

- Uniform-cost expandeert volgens pad-kost (backward cost): $g(n)$
- Greedy search expandeert volgens heuristiek, i.e. nabijheid tot doeltoestand (forward cost): $h(n)$
- Kunnen we de beide combineren ?
 - A* expandeert volgens de som: $f(n) = g(n) + h(n)$

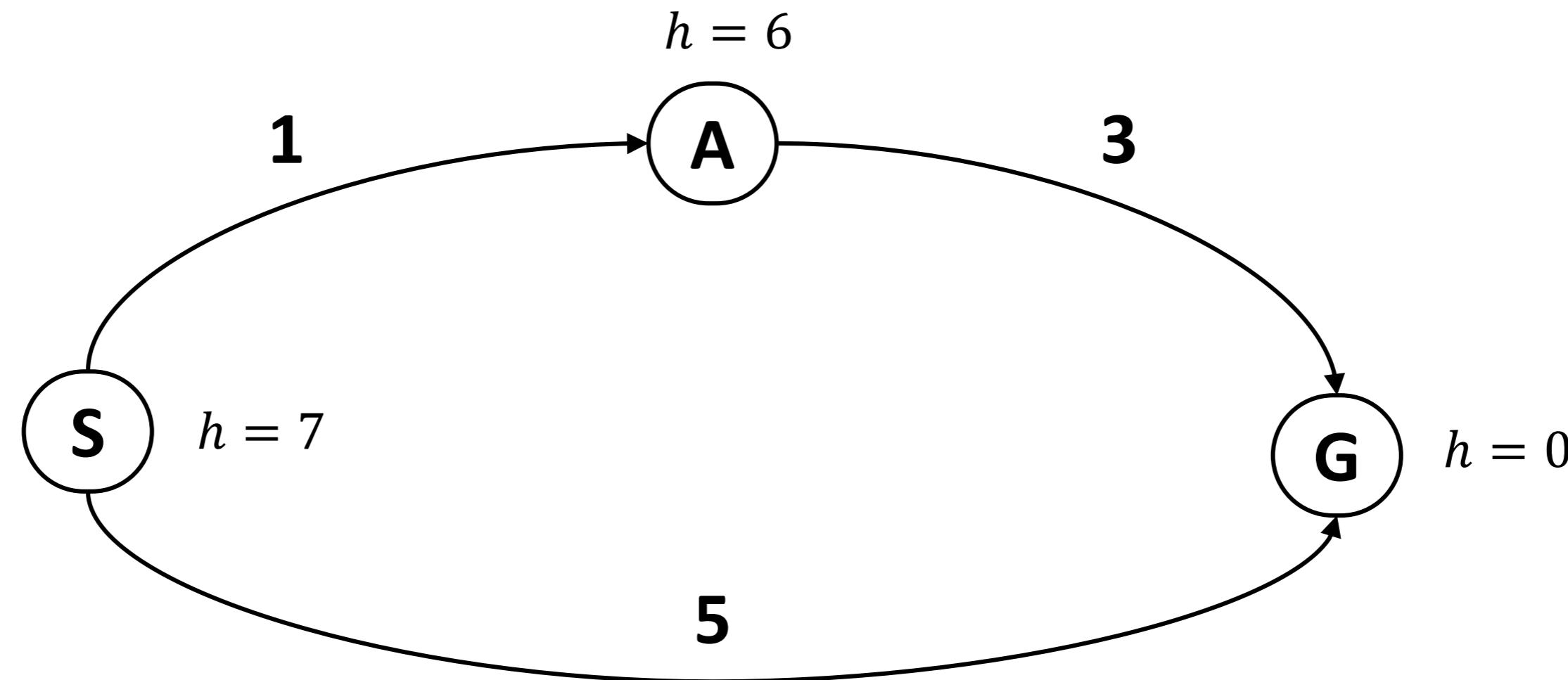
WANNEER STOPT A*?

Stoppen als we een doeltoestand tegenkomen ?



Neen, enkel stoppen wanneer de doeltoestand van de priority queue verwijderd wordt.

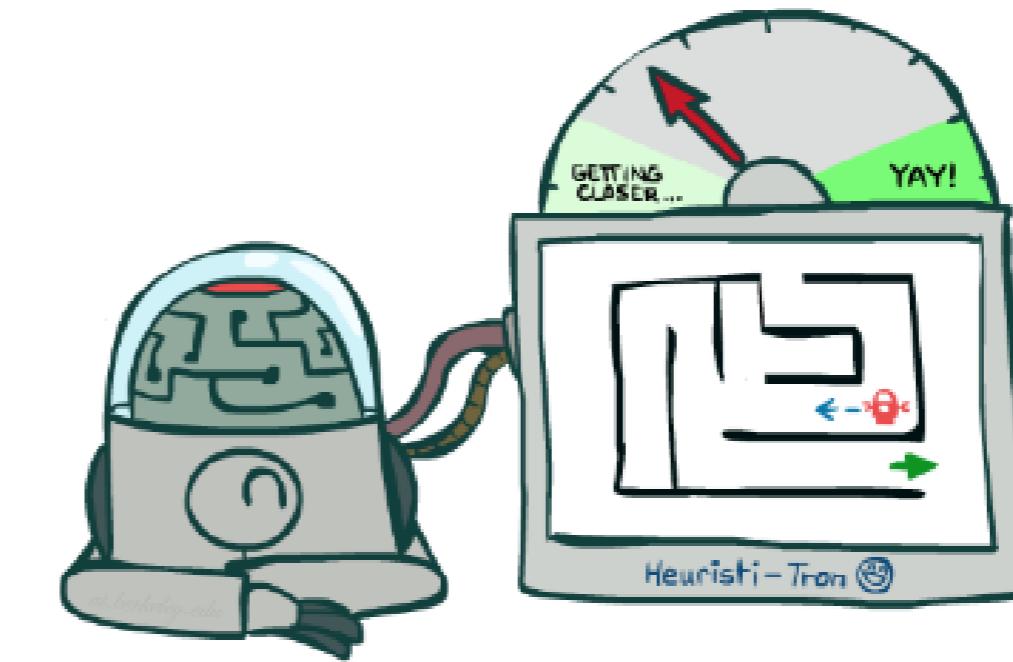
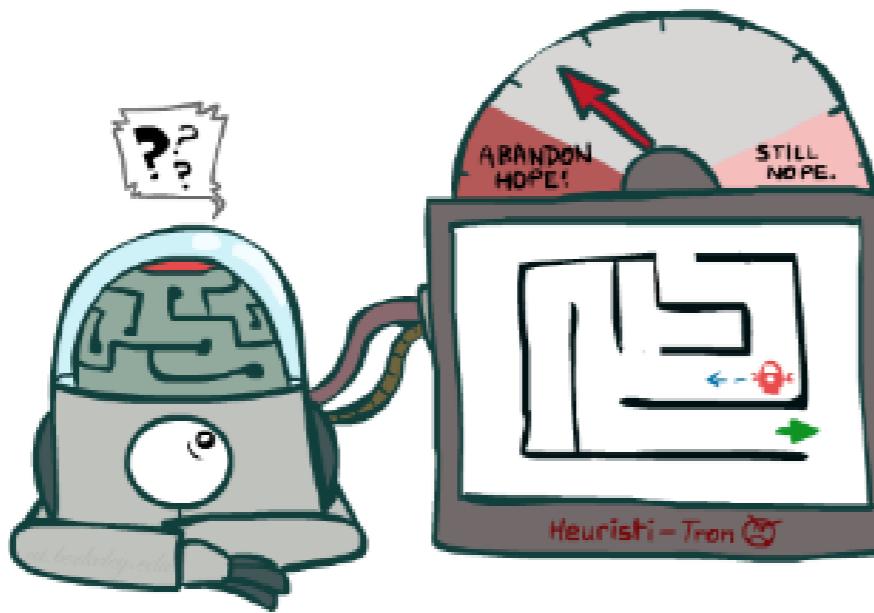
IS A* OPTIMAAL?



Wat gaat er verkeerd ?

- Slechte heuristiek: overschat de echte kost
- Schattingen moeten altijd een onderschatting zijn

TOELAATBARE HEURISTIEKEN



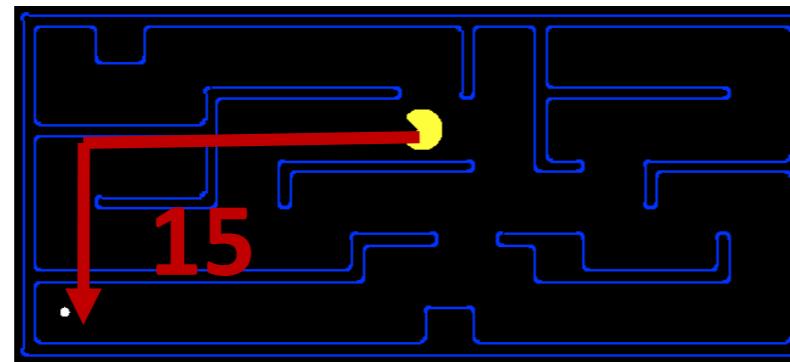
Ontoelaatbare (pessimistische) heuristiek doorbreekt de optimaliteit door goede plannen niet verder te laten expanderen

Toelaatbare (optimistische) heuristiek vertraagt slechte plannen, maar overschat nooit de echt kost naar het doel

TOELAATBARE HEURISTIEKEN

Een heuristiek h is toelaatbaar als
 $0 \leq h(n) \leq h^*(n)$
waarbij $h^*(n)$ de echte kost is naar het doel

- Voorbeelden



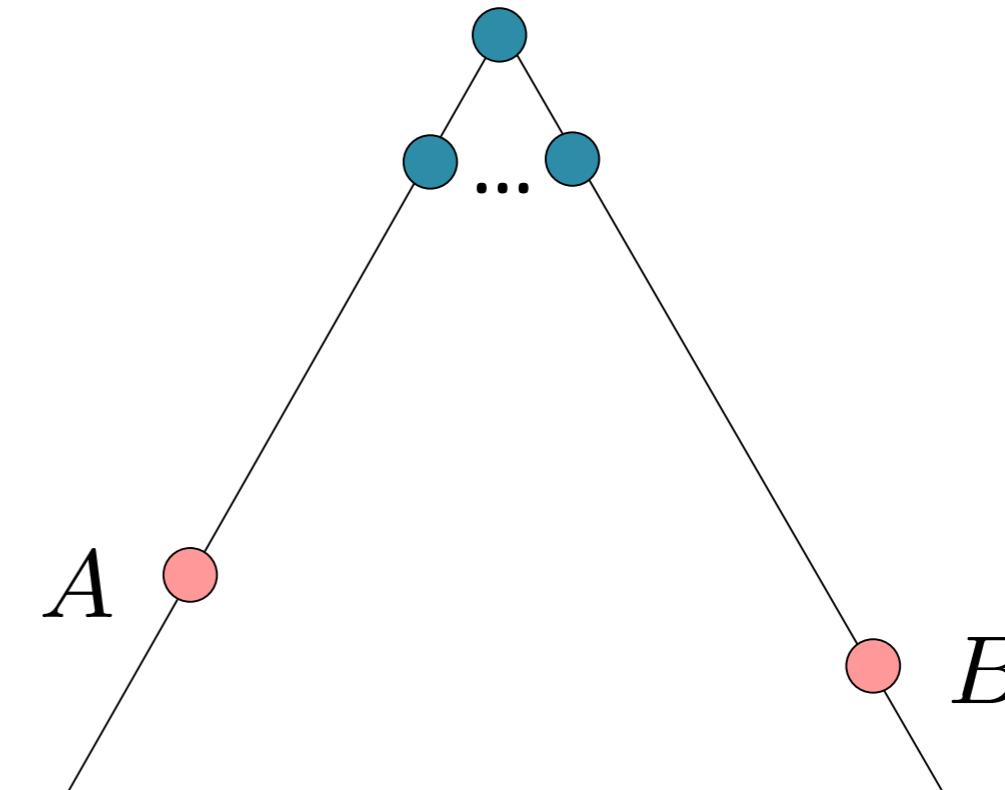
- Het ontwerpen van goede, toelaatbare heuristieken is de grootste uitdaging bij het oplossen van specifieke problemen met A*

A* IS OPTIMAAL

- We veronderstellen:
 - A is een optimaal doel
 - B is een suboptimal doel
 - h is toelaatbaar

Stelling:

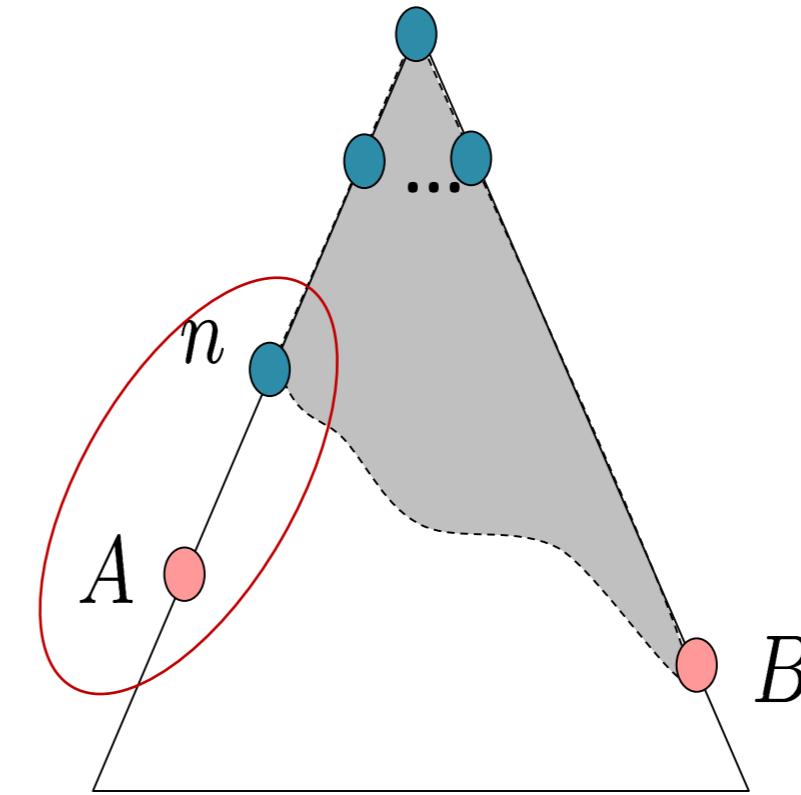
A zal van het zoekfront gehaald worden voor B



OPTIMALITEIT VAN A*: BEWIJS

Bewijs:

- Stel dat B op het zoekfront ligt
- Een voorouder n van A ligt ook op het zoekfront
- Stelling: n zal geëxpandeerd worden voor B
 1. $f(n) \leq f(A)$



$$f(n) = g(n) + h(n) \quad \text{Definitie totale kost}$$

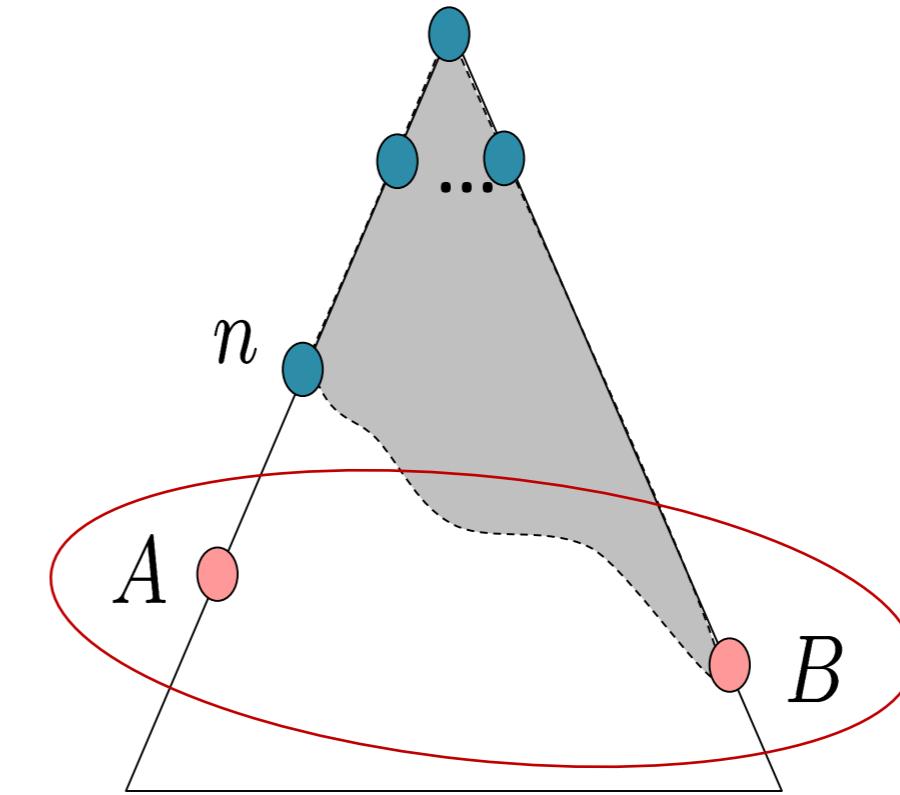
$$f(n) \leq g(A) \quad h \text{ is toelaatbaar}$$

$$g(A) = f(A) \quad h = 0 \text{ bij doel}$$

OPTIMALITEIT VAN A*: BEWIJS

Bewijs:

- Stel dat B op het zoekfront ligt
- Een voorouder n van A ligt ook op het zoekfront
- Stelling: n zal geëxpandeerd worden voor B
 1. $f(n) \leq f(A)$
 2. $f(A) \leq f(B)$



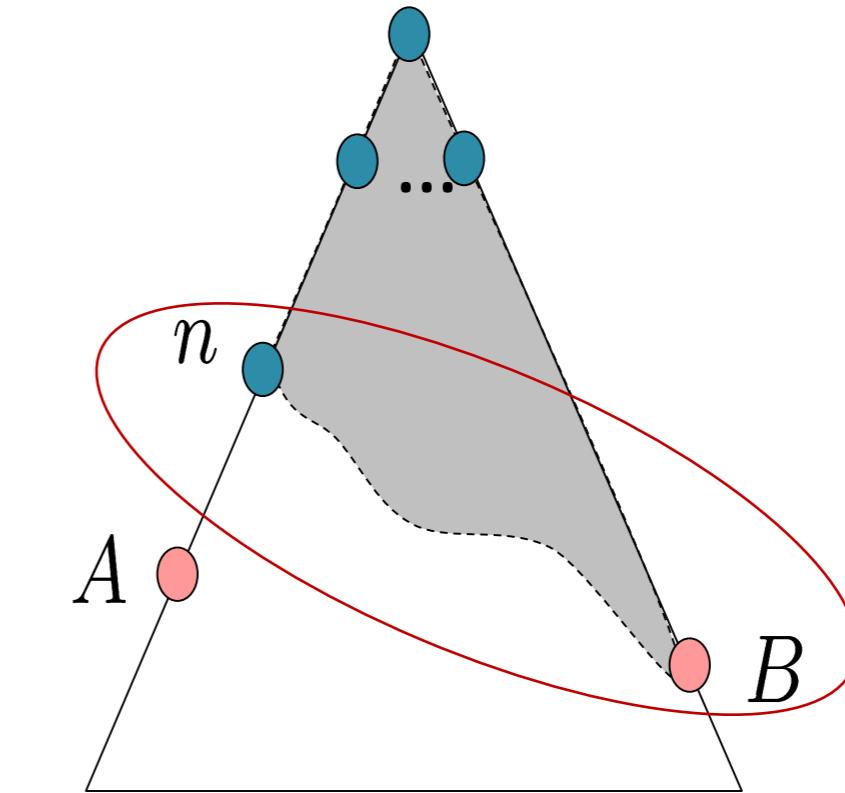
$g(A) < g(B)$ B is suboptimaal

$f(A) < f(B)$ $h = 0$ bij doel

OPTIMALITEIT VAN A*: BEWIJS

Bewijs:

- Stel dat B op het zoekfront ligt
- Een voorouder n van A ligt ook op het zoekfront
- Stelling: n zal geëxpandeerd worden voor B
 1. $f(n) \leq f(A)$
 2. $f(A) \leq f(B)$
 3. n wordt voor B geëxpandeerd
- Alle voorouders van A worden voor B geëxpandeerd
- A wordt voor B geëxpandeerd

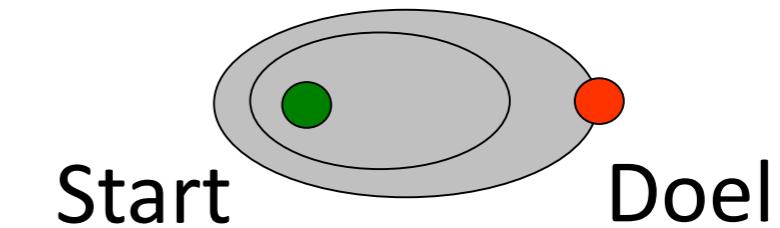
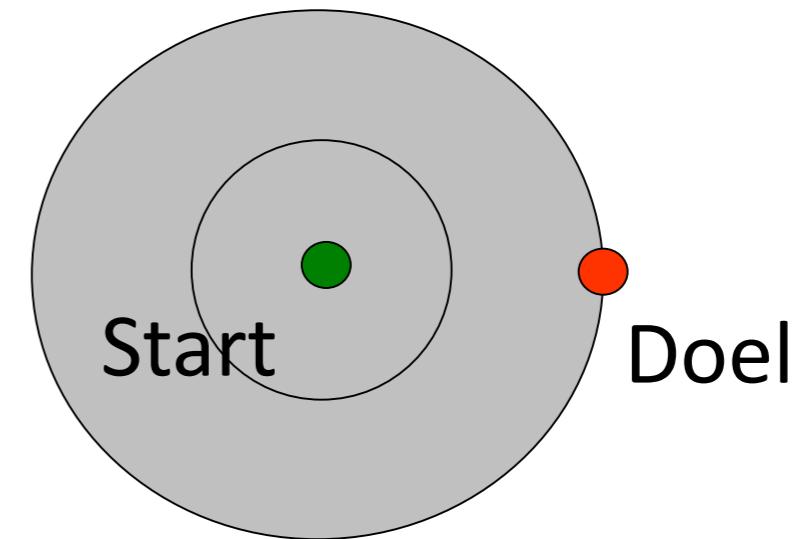


$$f(n) \leq f(A) < f(B)$$

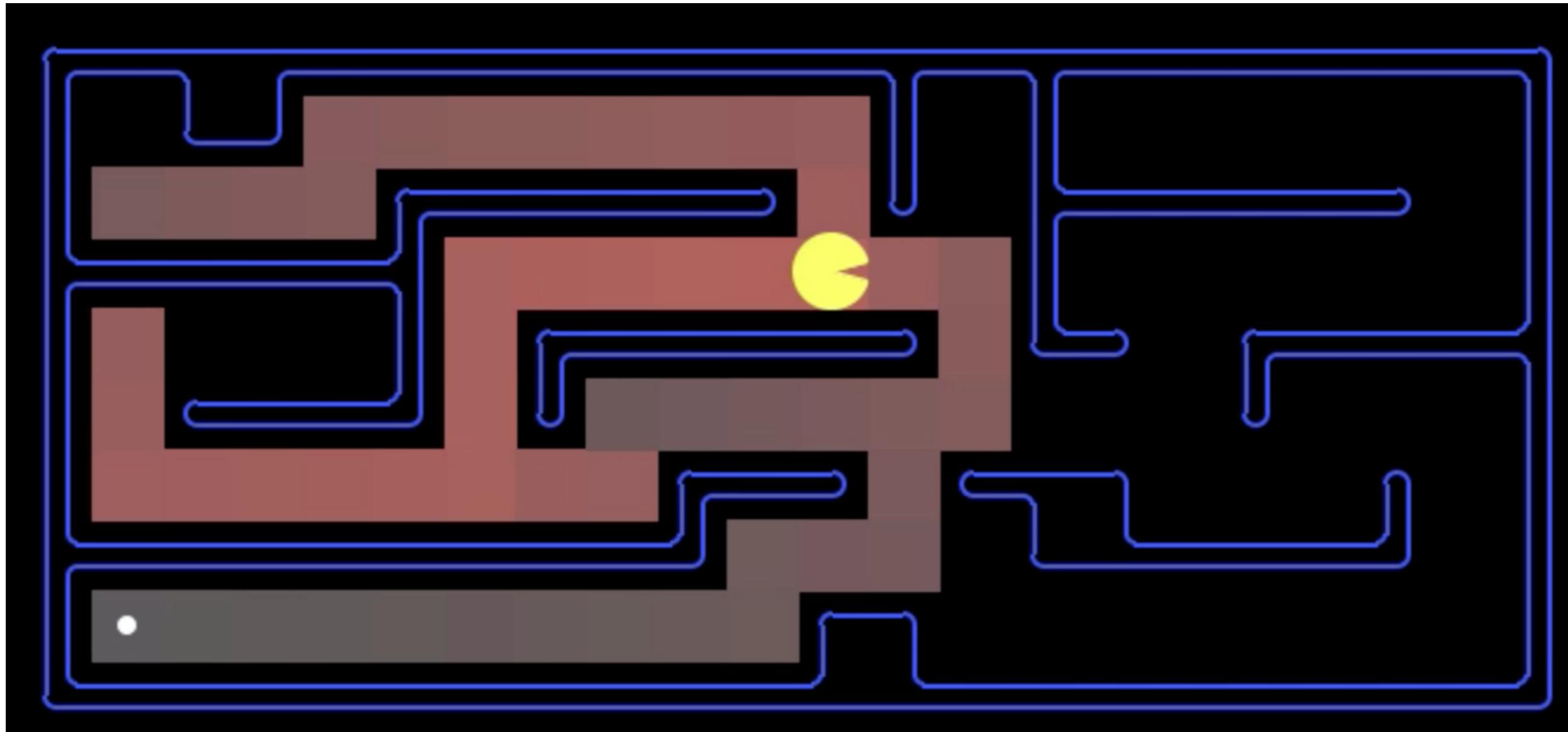
A* is optimaal

EIGENSCHAPPEN VAN A*

- Uniform-cost zoeken
expandeert gelijkmatig
in alle richtingen
- A* expandeert
voornamelijk in de
richting van het doel
maar houdt alle opties
open om optimaliteit te
garanderen

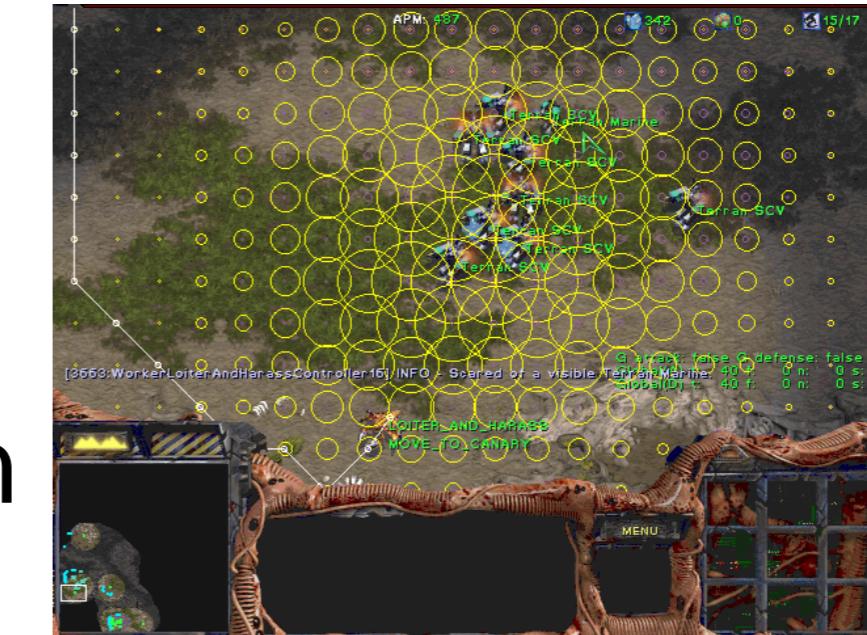


VOORBEELD: PACMAN WERELD



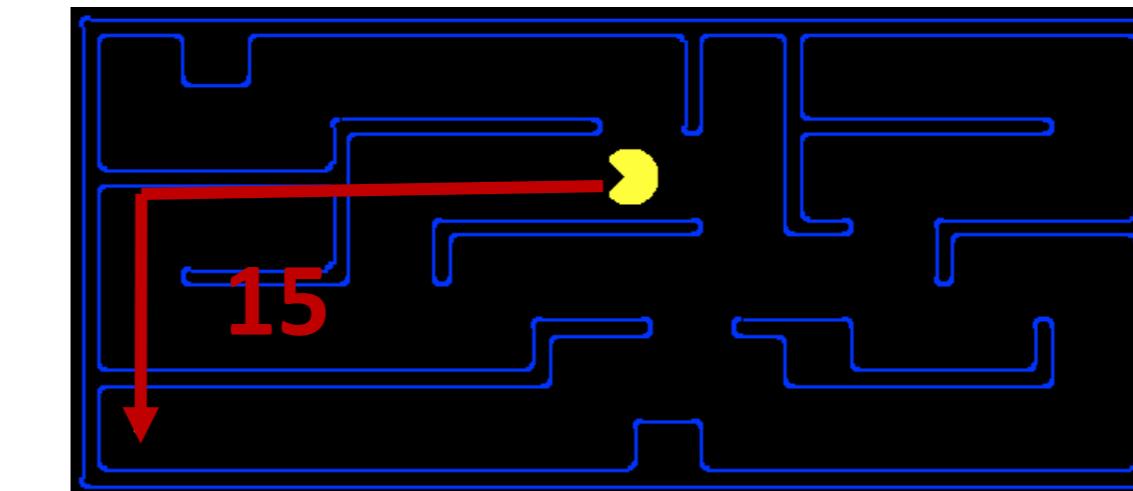
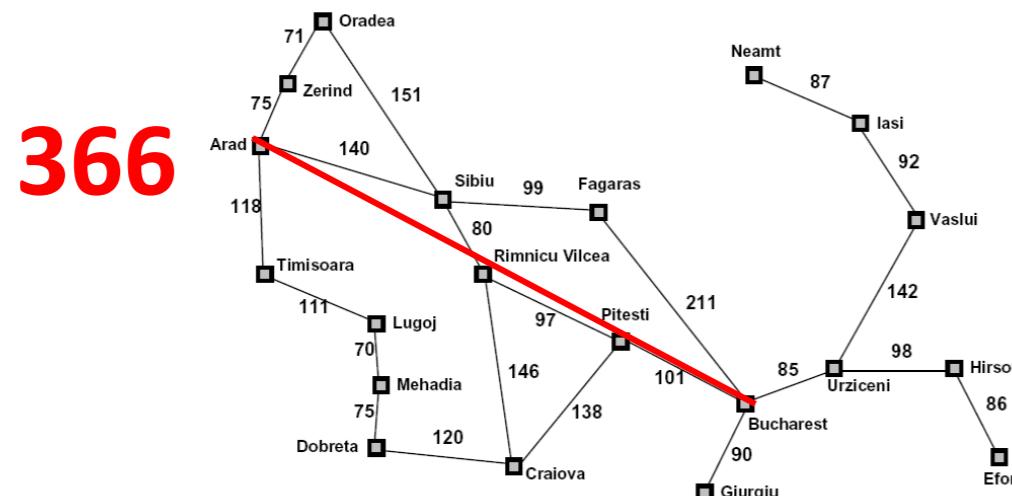
TOEPASSINGEN VAN A*

- Videospelletjes
- Padvinden / routeringsproblemen
- Resource planning problemen
- Robot motion planning
- NLP
- Machine-vertaling
- Spraakherkenning
- ...

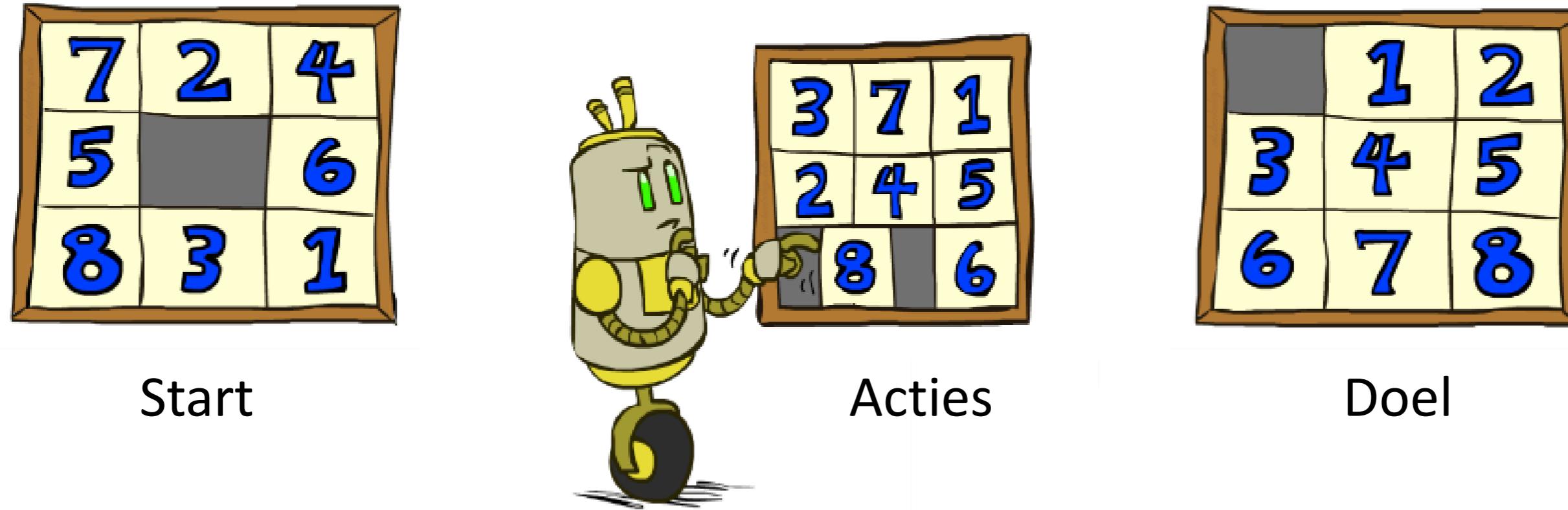


TOELAATBARE HEURISTIEKEN OPSTELLEN

- Het opstellen van goede heuristieken voor moeilijke zoekproblemen is de grootste uitdaging
- Toelaatbare heuristieken zijn vaak oplossingen voor een simpelere versie van het echte zoekprobleem, waar nieuwe acties mogelijk zijn (bvb door een muur gaan in PacMan wereld)



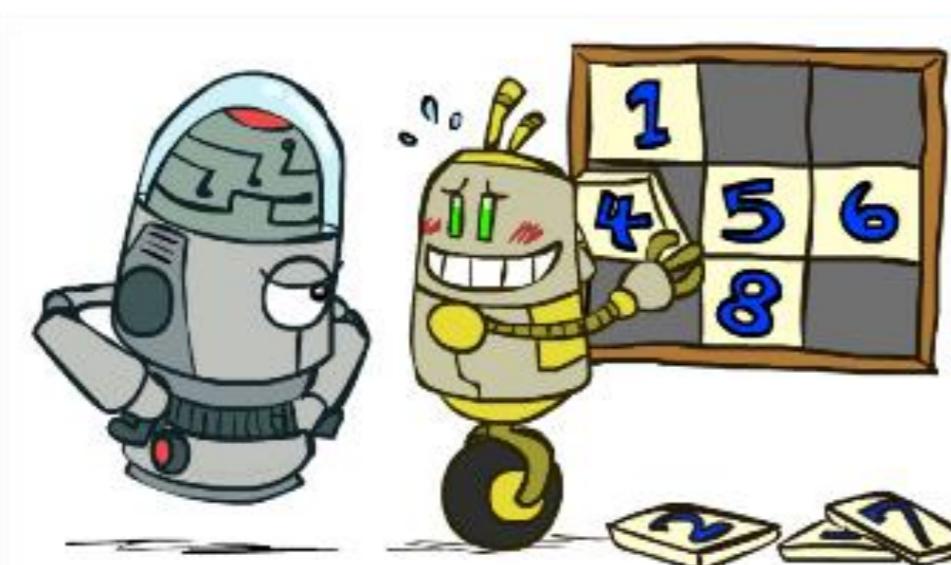
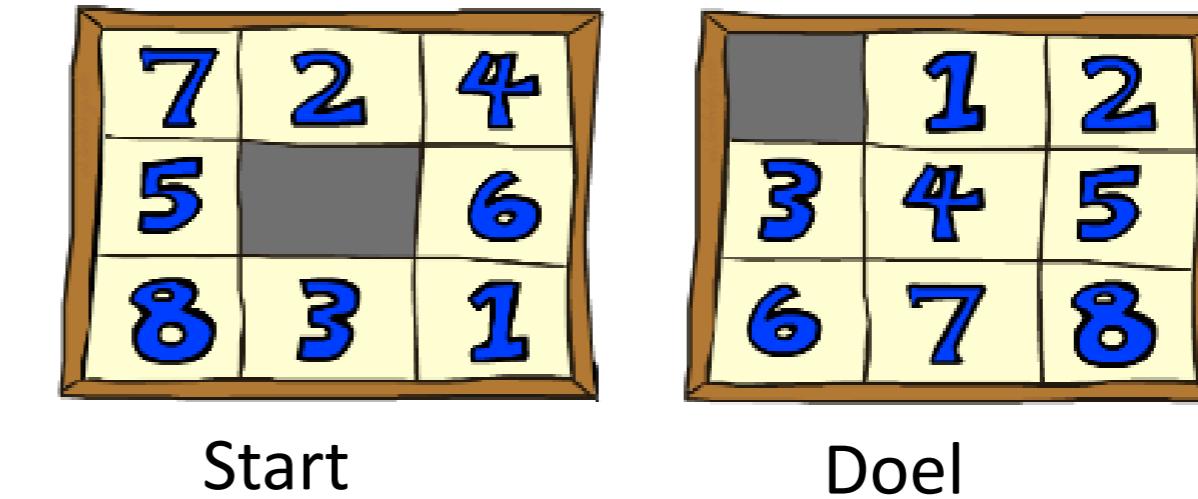
VOORBEELD: SCHUIFPUZZEL



- Wat zijn de toestanden? Hoeveel zijn er?
- Wat zijn de acties?
- Hoeveel opvolgers zijn er vanuit de start-toestand?
- Wat is de kost?

HEURISTIEK 1

- Heuristiek:
#tegels verkeerd geplaatst
- Toelaatbaar ?
- $h(\text{start}) = 8$
- Formulering (“relaxatie”)

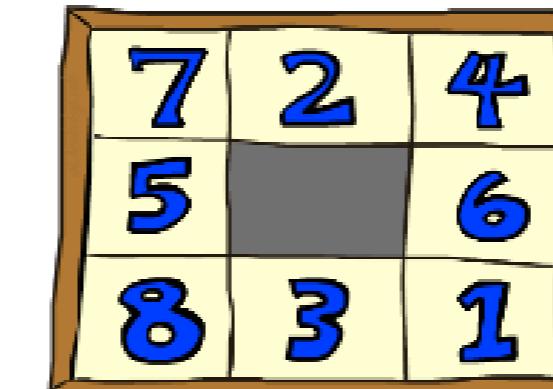


Gemiddeld aantal nodes
geëxpandeerd wanneer lengte
optimale pad = ...

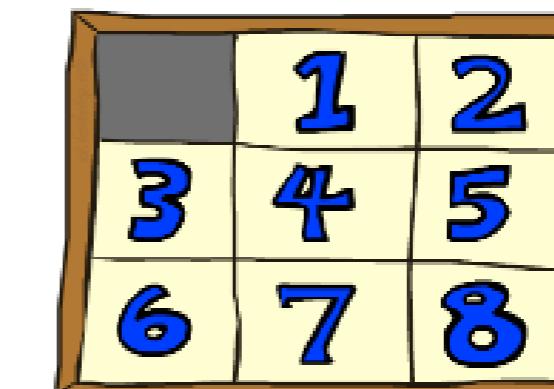
	...4 stappen	...8 stappen	...12 stappen
UCS	112	6,300	3.6×10^6
TILES	13	39	227

HEURISTIEK 2

- Wat als we een simpelere versie nemen waar elke tegel op elk moment elke richting kan uitgeschoven worden ?



Star



Doe

- Totale Manhattan afstand

- Toelaatbaar?

- $h(\text{start}) = 3 + 1 + 2 + \dots = 18$

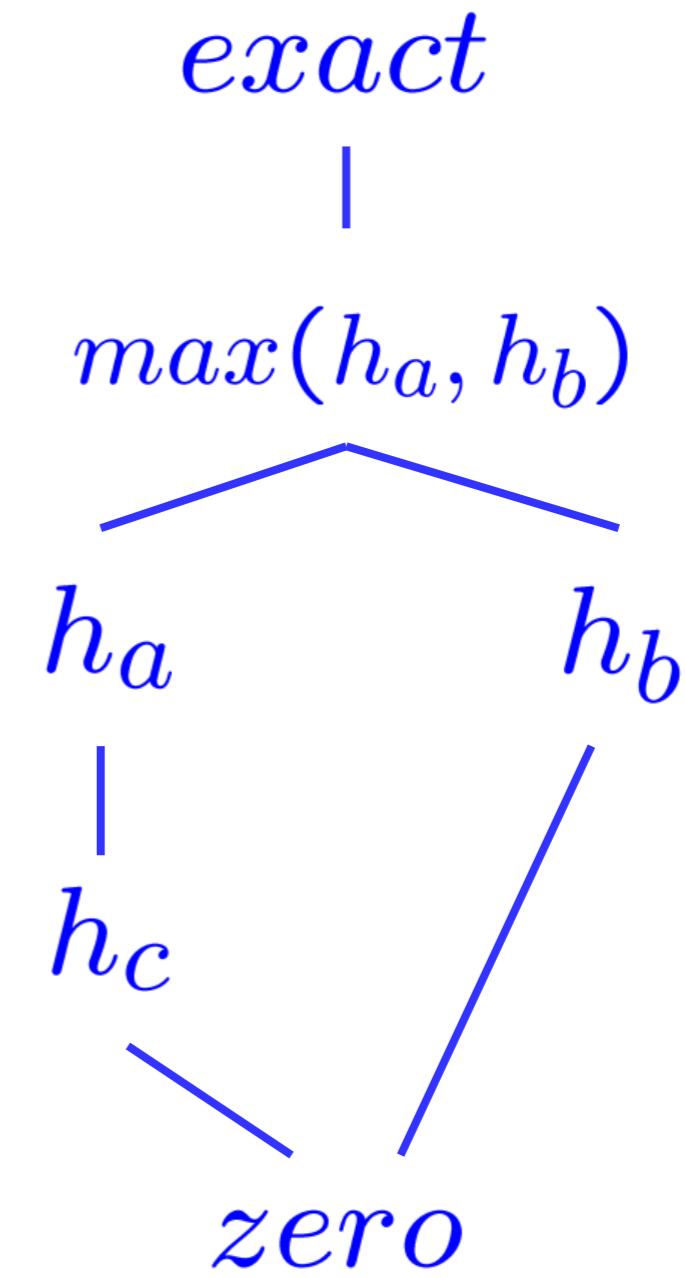
	Gemiddeld aantal nodes geëxpandeerd wanneer lengte optimale pad = ...		
18	...4 stappen	...8 stappen	...12 stappen
TILES	13	39	227
MANHATTAN	12	25	73

HEURISTIEK 3

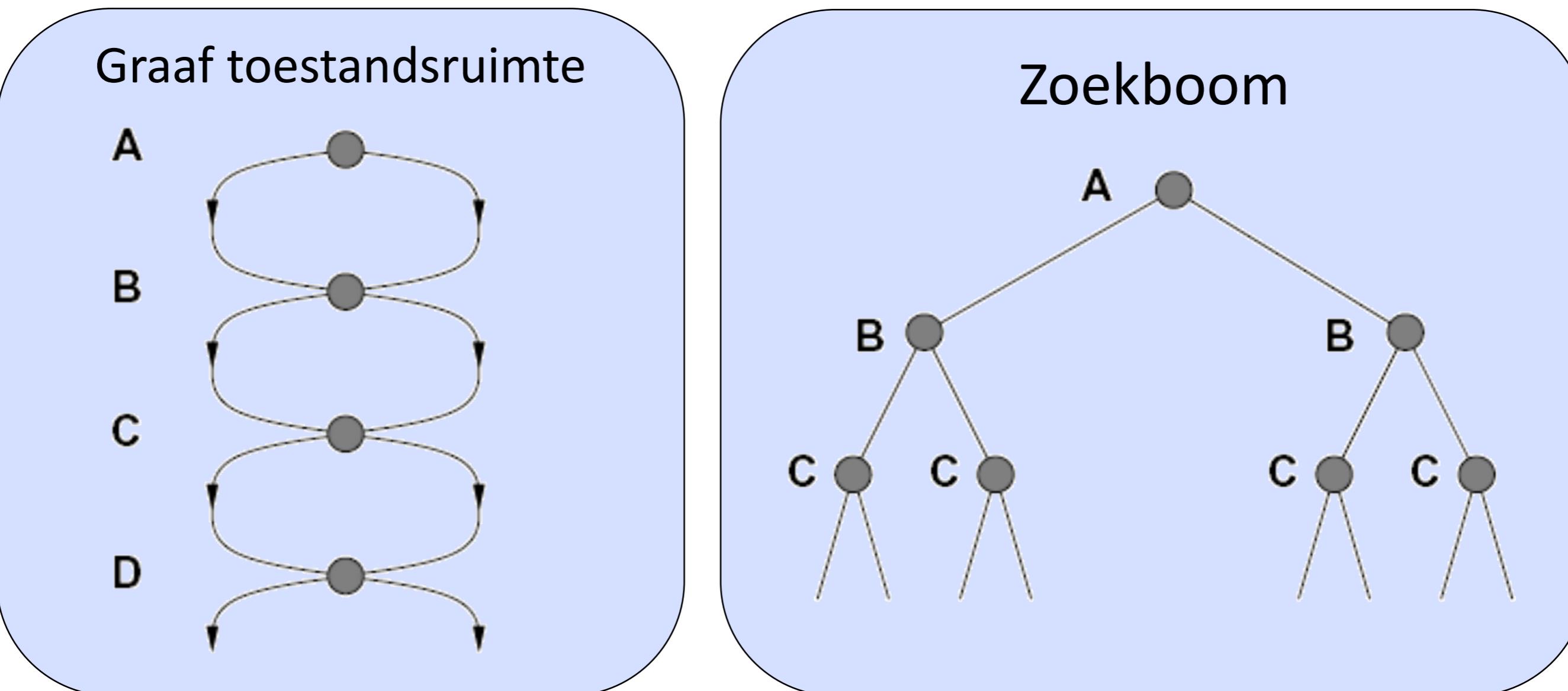
- Wat als we gewoon de echte kost gebruiken al heuristiek ?
 - Toelaatbaar ?
 - Zou het expanderen van knopen ons iets opleveren ?
 - Wat is het probleem ?
- Bij A*: trade-off tussen de kwaliteit van onze heuristiek en de hoeveelheid berekeningswerk per node
 - Heuristieken die de echte kost beter benaderen gaan typisch veel minder knopen expanderen, maar meer werk hebben per knoop om de heuristiek zelf te berekenen

TRALIESTRUCTUUR VAN HEURISTIEKEN

- Dominantie: $h_a \geq h_c$ als
$$\forall n : h_a(n) \geq h_c(n)$$
- Heuristieken vormen een semi-tralie:
 - Maximum van 2 toelaatbare heuristieken is een nieuwe, toelaatbare heuristiek
$$h(n) = \max(h_a(n), h_b(n))$$
- Triviale heuristieken:
 - Nul-heuristiek (met wat correspondeert dit ?)
 - De exacte heuristiek berekent de exacte kost tot het doel

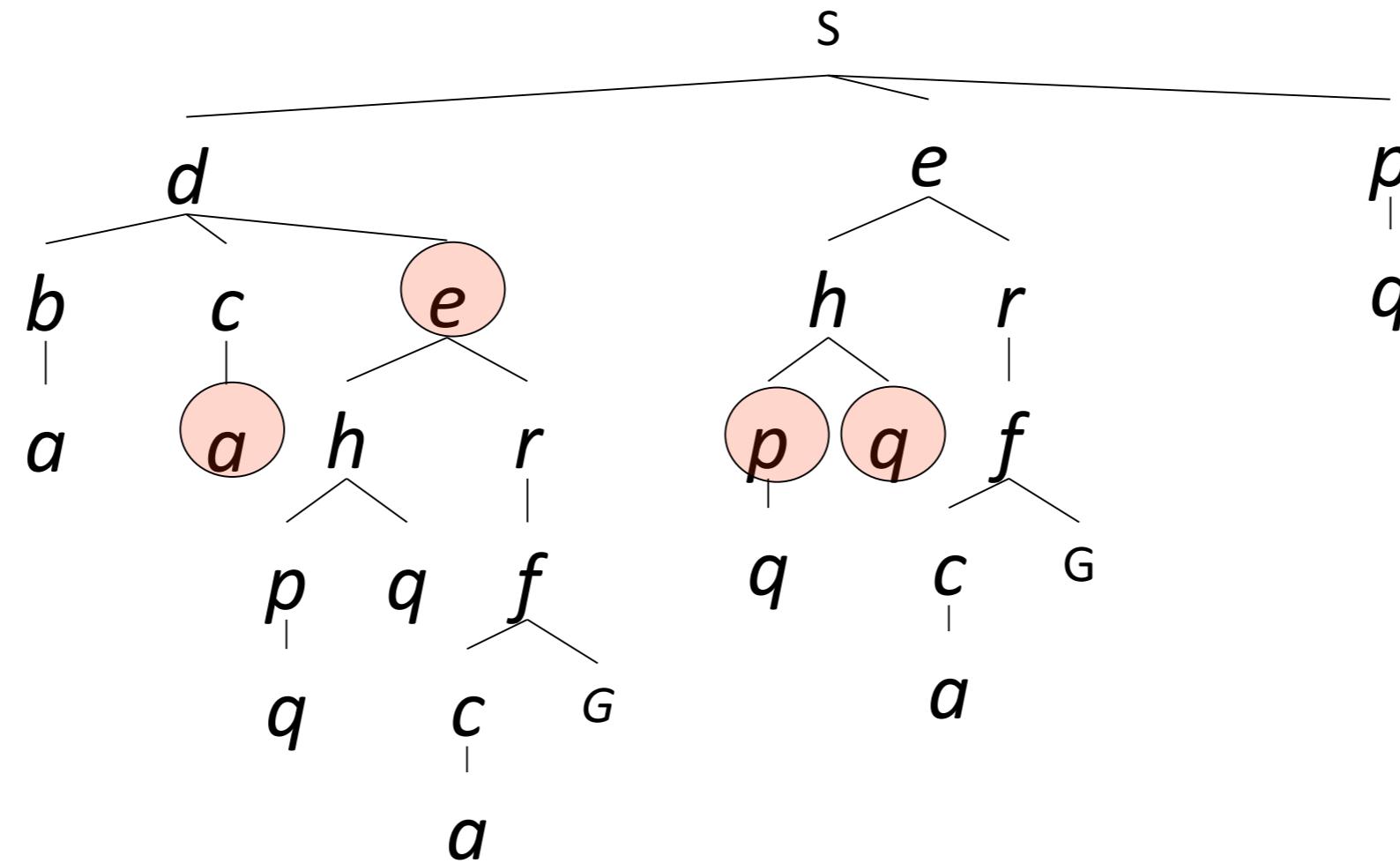


OMGAAN MET REDUNDANTIE



Het niet detecteren van herhaling van states kan exponentieel veel extra werk opleveren

GRAPH SEARCH



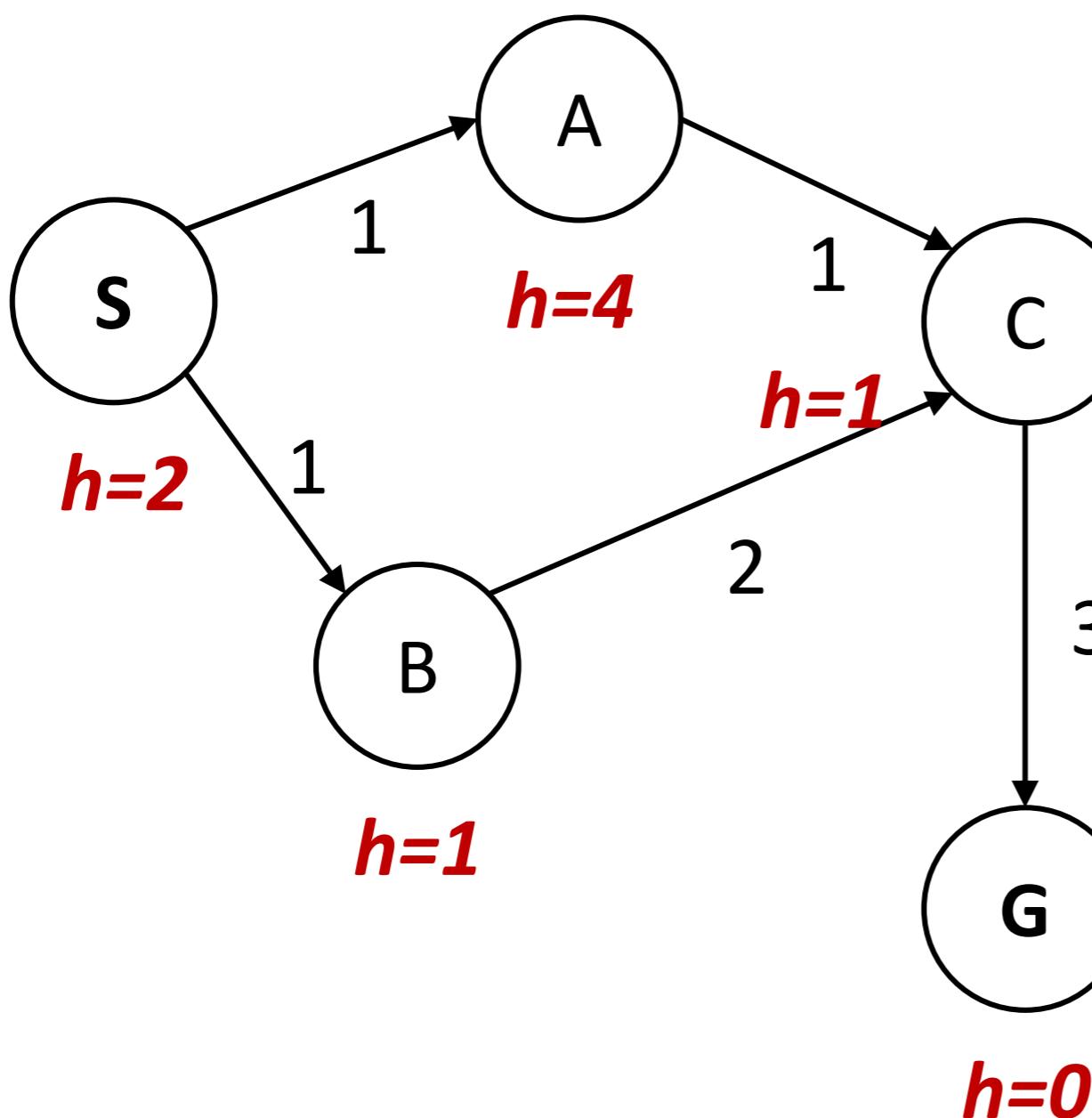
Bij BFS moeten we in principe de omcirkelde knopen niet verder expanderen. Waarom ?

GRAPH SEARCH

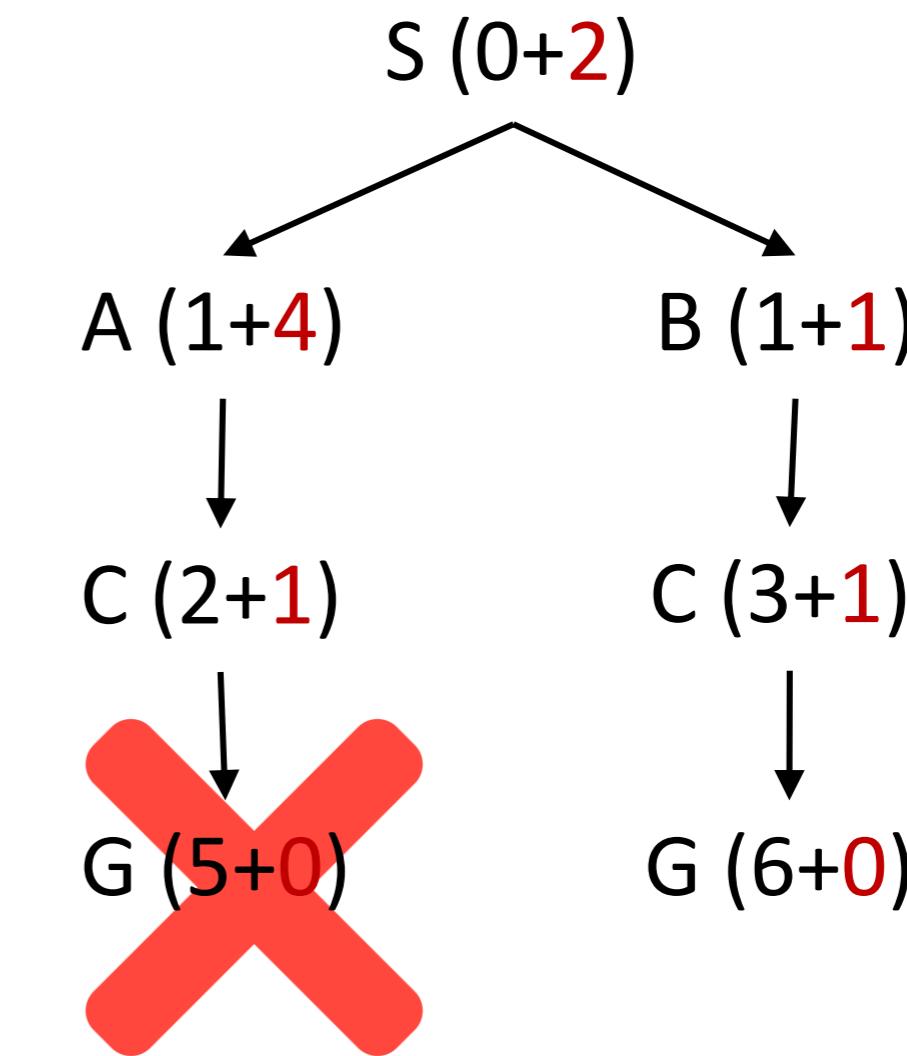
- Idee:
 - Nooit dezelfde toestand twee keer expanderen
- Algoritme:
 - Tree search + verzameling van geëxpandeerde toestanden (“closed set”)
 - Expandeer de zoekboom knoop per knoop, maar ...
 - ... alvorens een knoop te expanderen, check in de closed set of ze tevoren nog niet geëxpandeerd is
 - Voeg indien nodig de knoop toe aan de closed set
- Is A* Graph search compleet? Optimaal?

A* GRAPH SEARCH OPTIMALITEIT

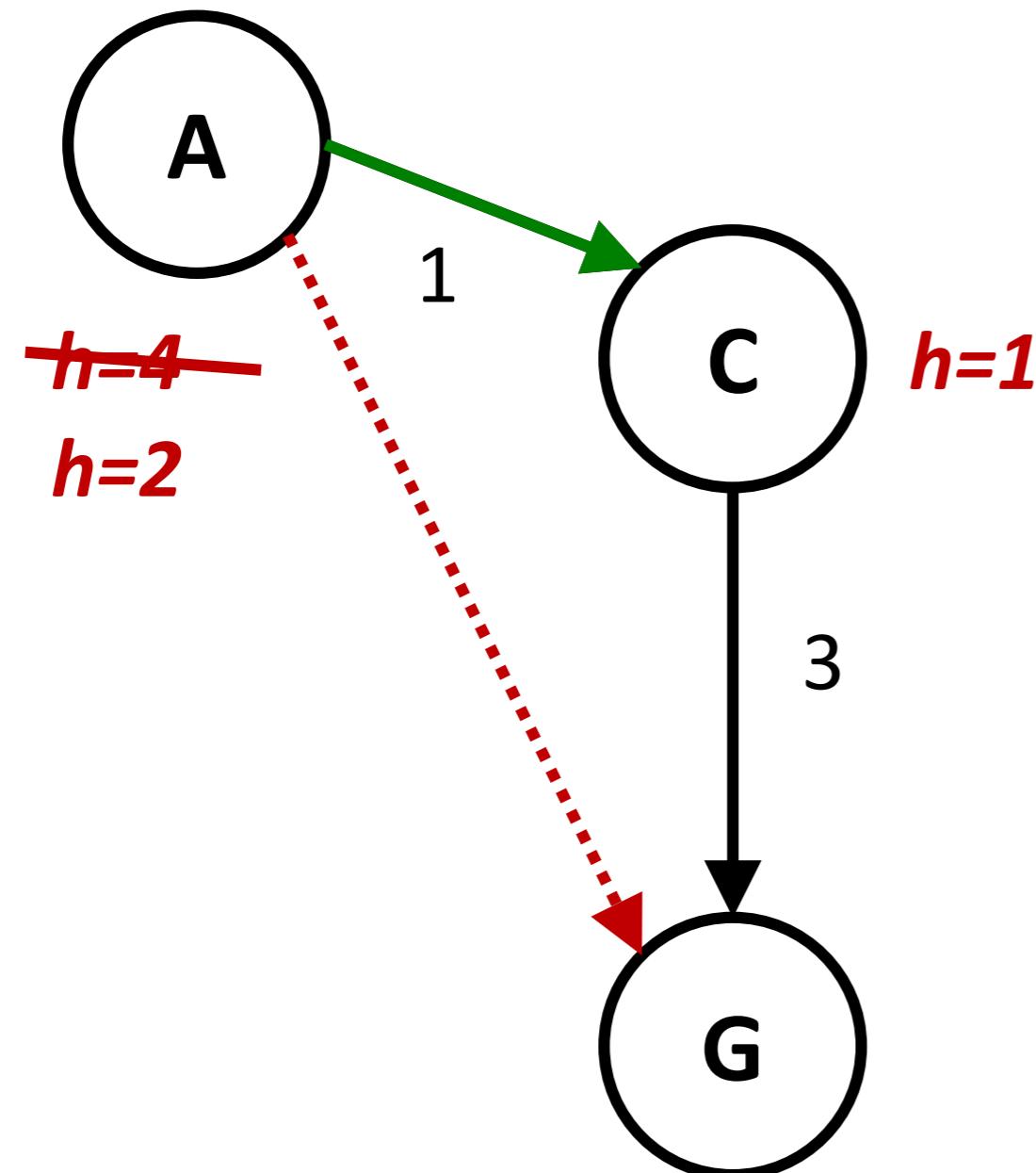
Graaf toestandsruimte



Zoekboom



WAT GAAT ER VERKEERD ?



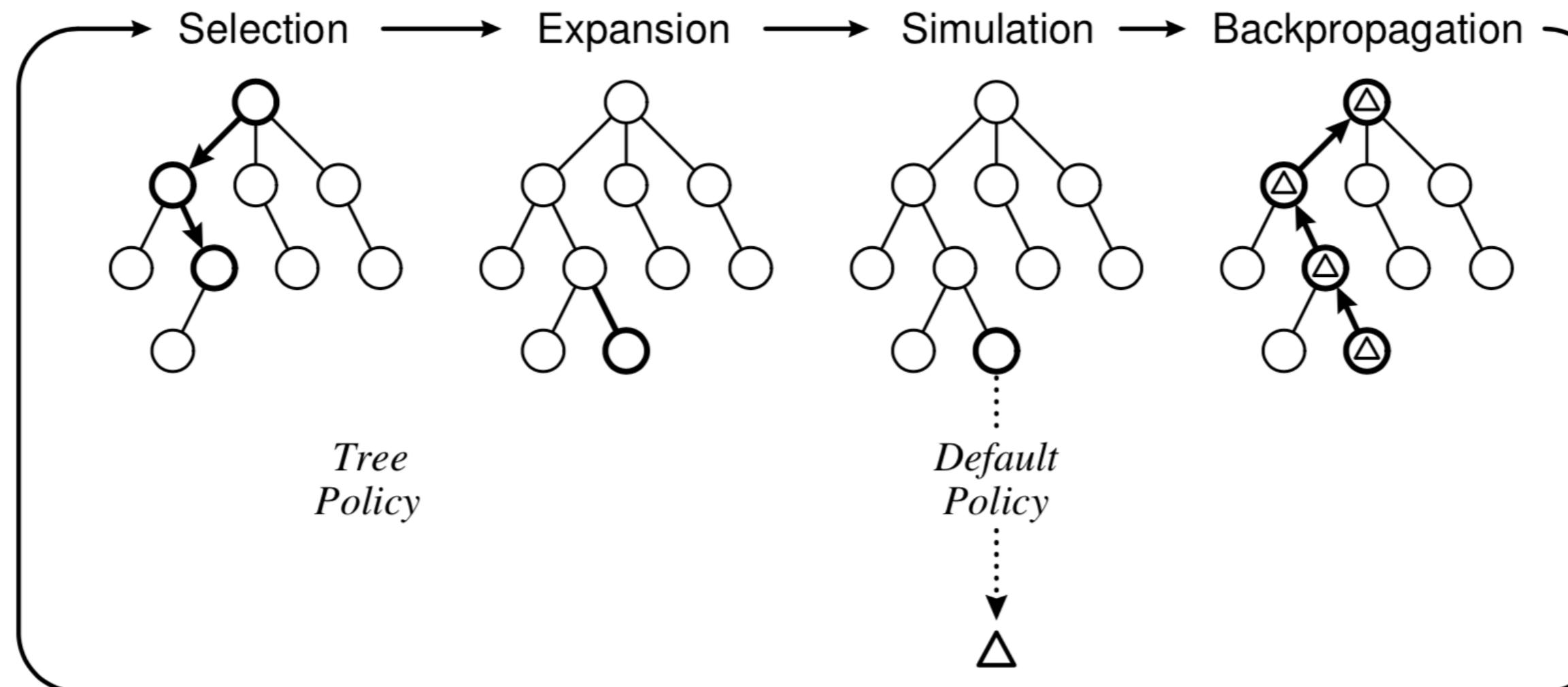
- Heuristiek h is toelaatbaar:
$$h(A) \leq \text{echte kost van } A \text{ naar } G$$
- Is h “consistent” ?
$$h(A) - h(C) \leq \text{echte kost van } A \text{ naar } C$$
- Gevolg van consistentie:
 - De f-waarde langs een pad daalt nooit
 - A* graph search is optimaal

CONCLUSIE

- Tree search:
 - A* is optimaal als de heuristiek toelaatbaar is
 - UCS is een special geval van A* ($h = 0$)
- Graph search:
 - A* is optimaal als de heuristiek consistent is
 - UCS is optimal ($h = 0$ is consistent)
- Consistentie impliceert toelaatbaarheid
- Over het algemeen zijn de meeste toelaatbare heuristieken consistent, zeker in het geval van “relaxaties”

MONTE CARLE TREE SEARCH

- Sampling-gebaseerde techniek om een spelboom op te stellen
- Behaalde zeer goede resultaten voor Go (AlphaGo)
- Anytime algoritme, gemakkelijk te paralleliseren



MONTE CARLO TREE SEARCH

- Selectie
 - Kies een top in de spelboom die nog niet bezocht is. Dit gebeurt volgens een zogenaamde *tree policy*, die een afweging hoort te maken tussen enerzijds *exploratie* (gebieden verkennen die nog niet eerder gezien zijn) en *exploitatie* (acties verkennen die er veelbelovend uitzien).
- Expansie
 - Genereer een of meerdere opvolgers van de geselecteerde top

MONTE CARLO TREE SEARCH

- Simulatie
 - Voer een simulatie uit van het spel vanaf een van de opvolgers van de geselecteerde top volgens een *default policy*. Hieruit verkrijgen we een score (bvb win/loss)
- Backpropagation
 - Werk de statistieken bij van de opvolger tot de wortelknoop. Elke top in de MCTS-spelboom houdt minstens twee waarden bij: de *visit count* (het aantal keer dat de top al bezocht is) en de *reward* (de som van de scores die we gekregen hebben over alle simulaties die langs deze top passeren).

TREE POLICY: UPPER CONFIDENCE BOUNDS FOR TREES (UCT)

- Selecteer de top v' die volgende hoeveelheid maximaliseert:
$$\frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$$
- Hierbij stelt $N(v')$ het aantal node visits van v' voor, $Q(v')$ de totale reward van v' (dit zijn de twee waarden die in elke node worden bijgehouden) en v is de parent van v'

BACKPROPAGATION

- Update van de toegevoegde knoop langs alle parents naar de root node:
 - Update node count
 - Update total reward