Inteligență Artificială

Bogdan Alexe
bogdan.alexe@fmi.unibuc.ro

Secția Tehnologia Informației, anul III, 2022-2023
Cursul 10

Recapitulare – cursul trecut

1. Rezolvarea problemelor prin căutare
   - Graful stărilor
   - Arborele de căutare

2. Căutare neinformată
   - Căutare în lățime (Bread-First Search)
   - Căutare în adâncime (Depth-First Search)
   - Căutare în adâncime limitată (depth-limited search)

Strategii de căutare neinformată

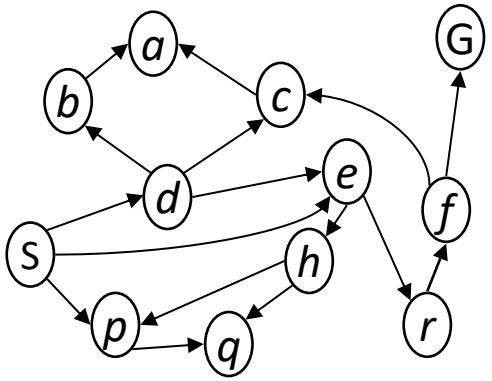
Strategii neinformate – folosesc numai informație disponibilă din definirea problemei. Pot genera succesori și distinge dacă o stare este scop sau. Din acest motiv se mai numesc și strategii de căutare ***oarbe***.

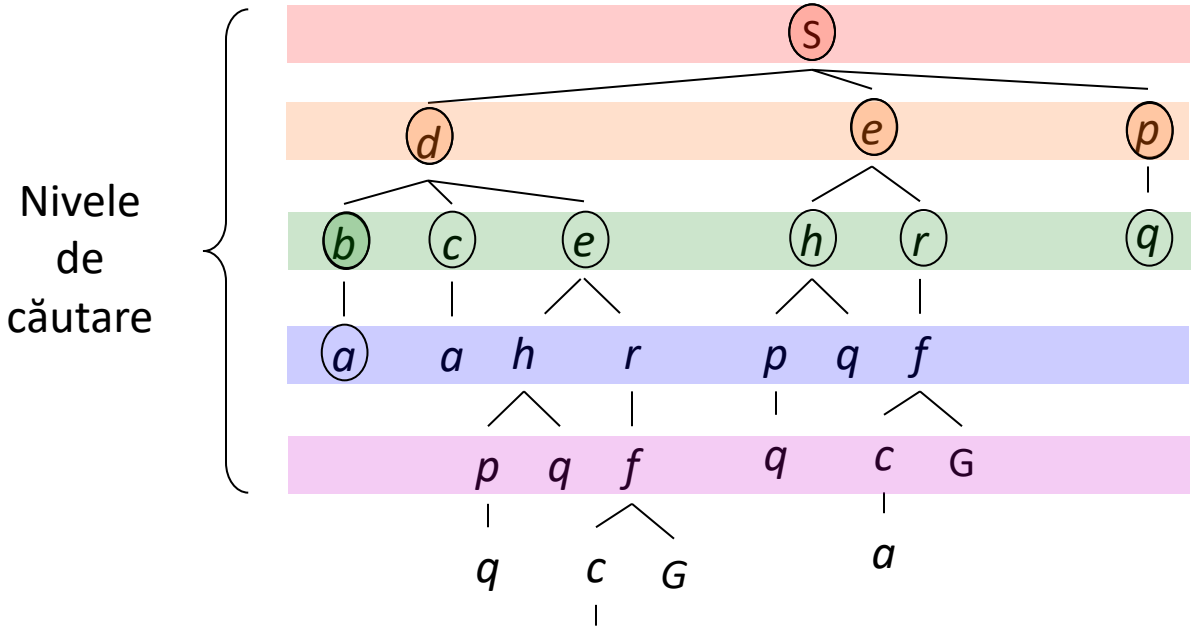
- căutare în lățime (bread-first search)
- căutare în adâncime (depth-first search)
- căutare cu cost uniform (uniform-cost search)
- căutare cu adâncime limitată (depth-limited search)
- căutare cu adâncime incrementală (iterative deepening search)

Căutare în lățime (bread-first = BF)

Strategie: expandează succesiv toate nodurile în ordinea adâncimii începând cu nodul rădăcină

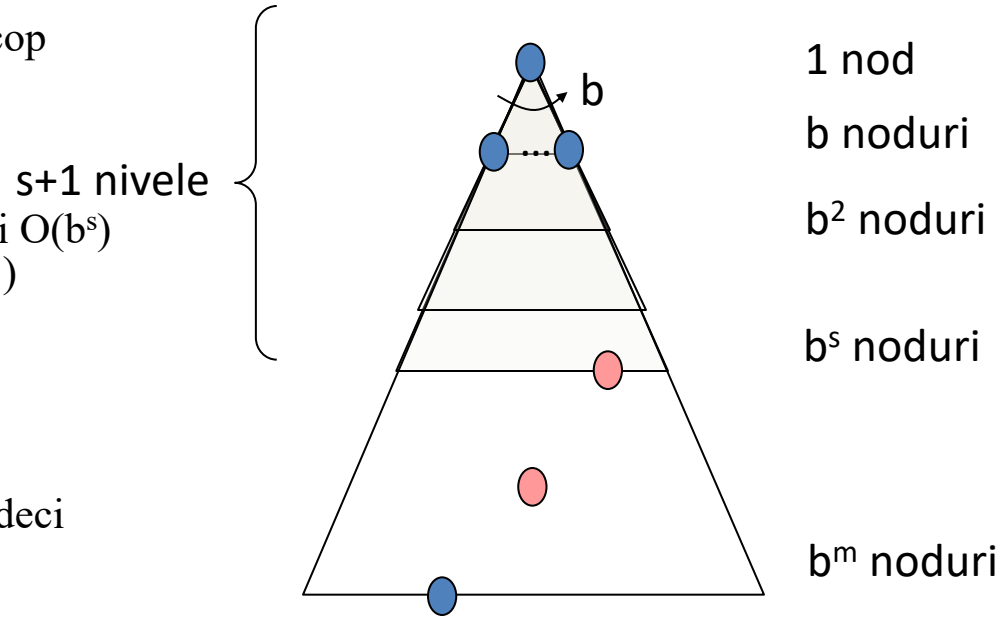
Frontiera este implementată cu o coadă (FIFO)





Proprietățile căutării în lățime

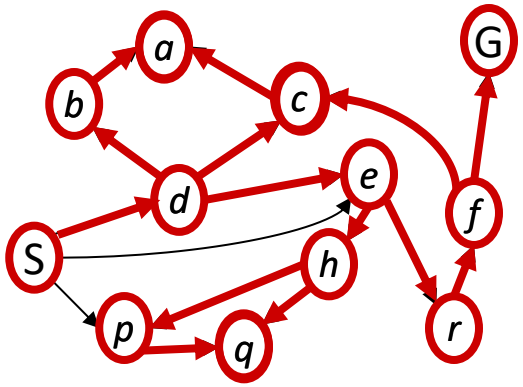
- Ce fel de noduri expandează BF?
  - procesează toate nodurile deasupra soluției de adâncime minimă
  - fie s = adâncimea minimă a unui nod scop
  - complexitate timp $O(b^s)$
- Cât de mult spațiu necesită BF?
  - este spațiul pentru memorarea frontierei $O(b^s)$ + memorarea nodurilor explorate $O(b^{s-1})$
  - limită superioară dată de ultimul nivel
  - complexitate $O(b^s)$
- Completitudine?
  - dacă o soluție există atunci s este finit, deci DA!
- Optimalitate?
  - dacă costurile fiecărei muchii sunt egale (= 1) atunci DA!

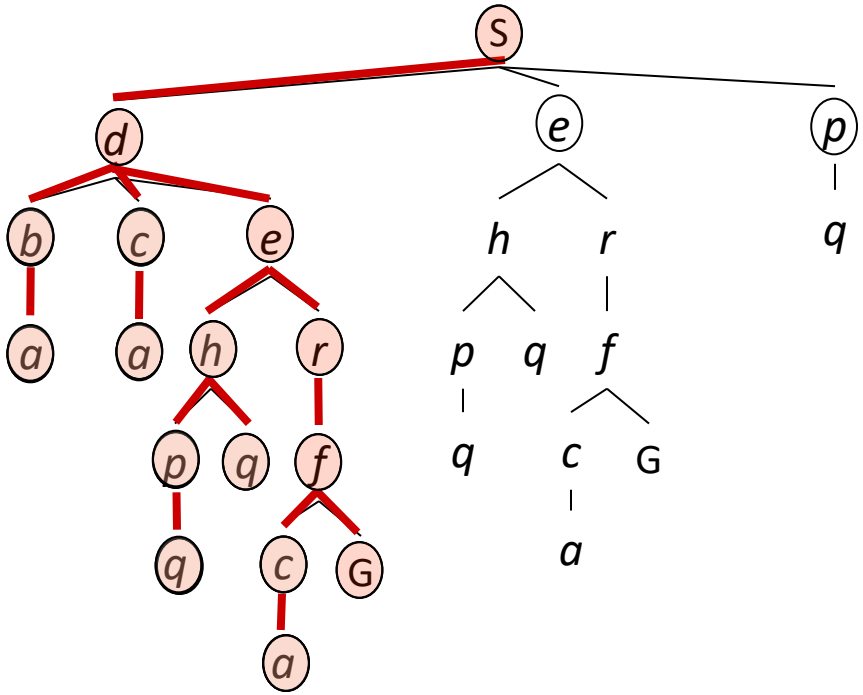


Căutarea în adâncime (depth-first = DF)

Strategie: expandează nodul curent de adâncime maximă

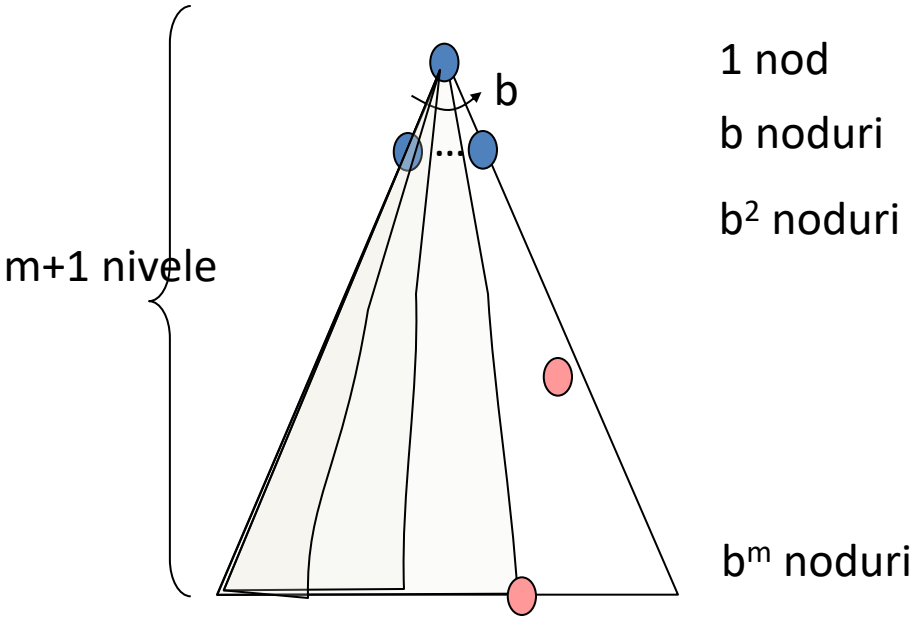
Frontiera este implementată cu o stivă (LIFO)





Proprietățile căutării în adâncime

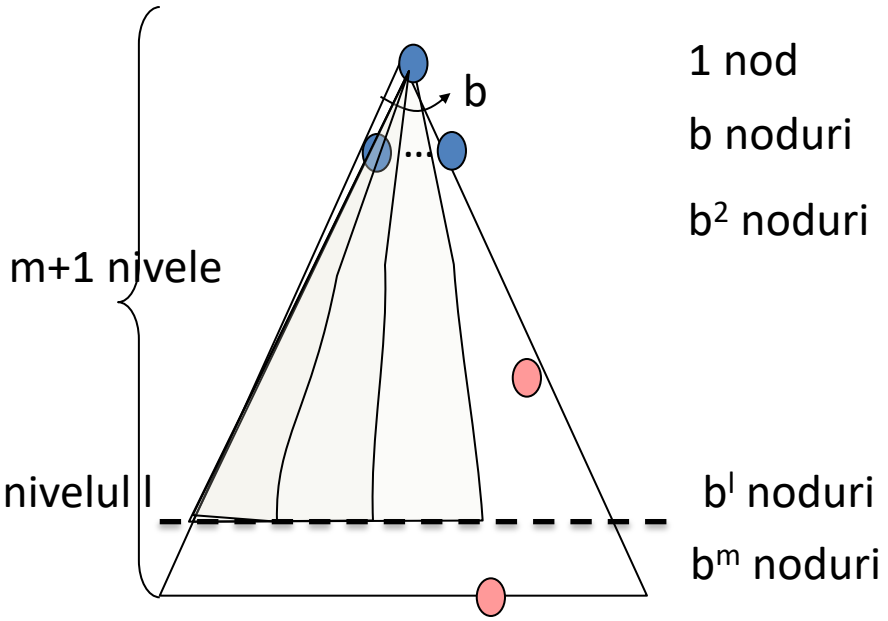
- Ce fel de noduri expandează DF?
  - partea stângă prefix din arbore
  - poate procesa întreg arborele!
  - complexitate timp $O(b^m)$
- Cât de mult spațiu necesită DF?
  - numai fiii de la rădăcină spre frunză neexpandați, deci O(bm)
- Completitudine?
  - m ar putea fi infinit, în acest caz NU!
  - m finit + dacă se țin minte nodurile vizitate DA!
- Optimalitate?
  - NU, găsește "cea mai din stânga" soluție, indiferent de adâncime sau cost



Căutare în adâncime limitată

Căutare în adâncime până la un anumit nivel l (depth-limited DL).

- Ce fel de noduri expandează DL?
  - partea stângă prefix din arbore, numai până la nivelul l
  - complexitate timp $O(b^l)$
- Cât de mult spațiu necesită DL?
  - fiii de la rădăcină spre frunză + nodurile neexplorate, deci O(bl)
- Completitudine?
  - NU, poate rata soluția dacă l este prea mic
- Optimalitate?
  - NU, găsește "cea mai din stânga" solutie de adâncime maximă l, indiferent de cost



Cuprinsul cursului de azi

1. Căutare neinformată
   - Căutare în adâncime incrementală (iterative deepening search)
   - Căutare uniformă după cost (uniform-cost search)

2. Căutare informată
   - căutare Greedy
   - algoritmul A*
   - euristici admisibile, dominante și banale

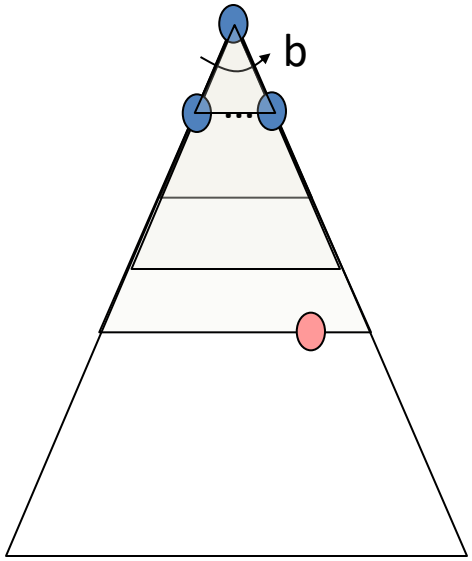
Căutare incrementală în adâncime

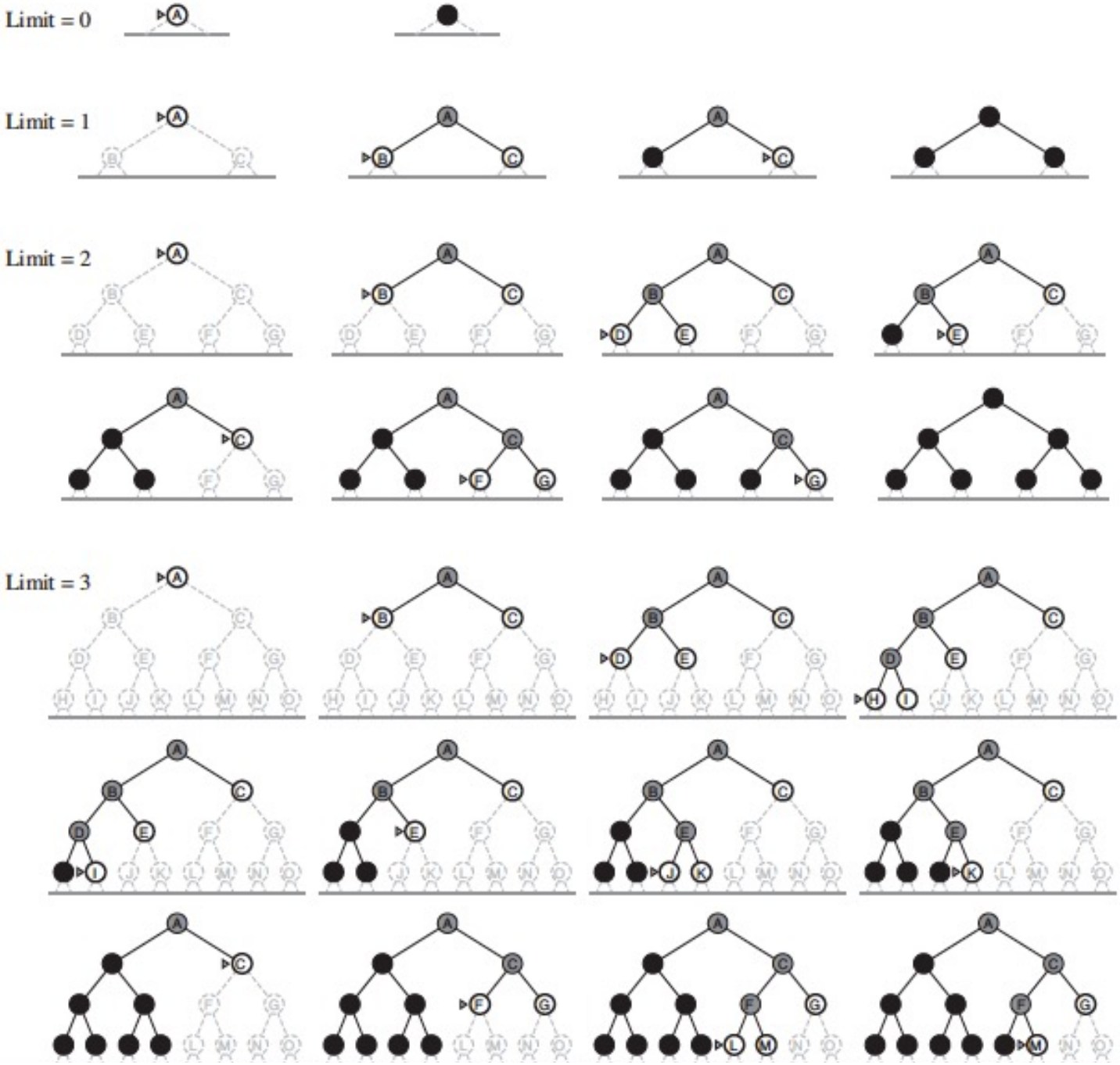
- Combină căutarea în adâncime (DF) cu căutarea în lățime (BF):
  - Rulează DF cu adâcimea limitată la 1.

  Dacă nu găsește o soluție …
  - Rulează DF cu adâcimea limitată la 2.

  Dacă nu găsește o soluție …
  - Rulează DF cu adâcimea limitată la 3.

  Dacă nu găsește o soluție …
  - Rulează DF cu adâcimea limitată la 4….




Limit = 0
Limit = 1
Limit = 2
Limit = 3

Căutare incrementală în adâncime

Ce fel de noduri expandează IDS?

BF + DF până la nivelul s al soluției de adâncime minimă

Complexitate timp $O(b^s)$

$N(IDS) = s*b + (s-1) * b^2 + \ldots + b^s$ (noduri vizitate de IDS)

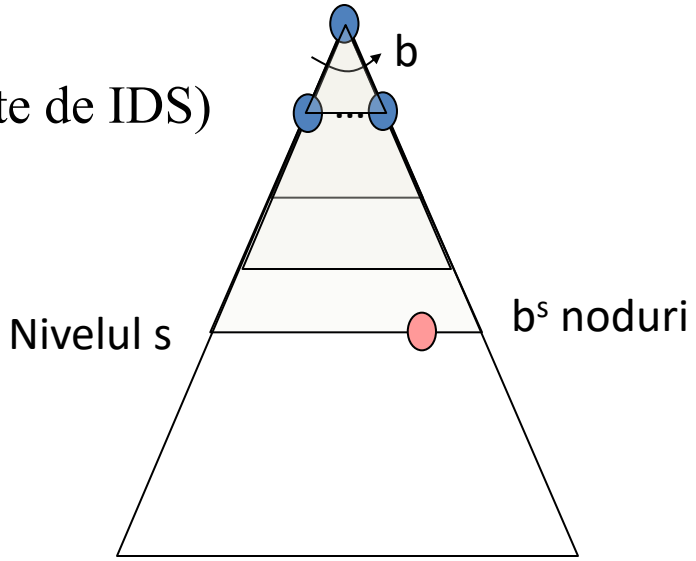
$N(BF) = b + b^2 + \ldots + b^s$ (noduri vizitate de BF)

Pentru b = 10, s = 5:

N(IDS) = 50 + 400 + … + 100000 = 123450

N(BF) = 10 + 100 + ... + 100000 = 111110

Nu este foarte redundant



Cât de mult spațiu necesită DL?

fii de la rădăcină spre frunză + cele neexplorate, deci O(bs)

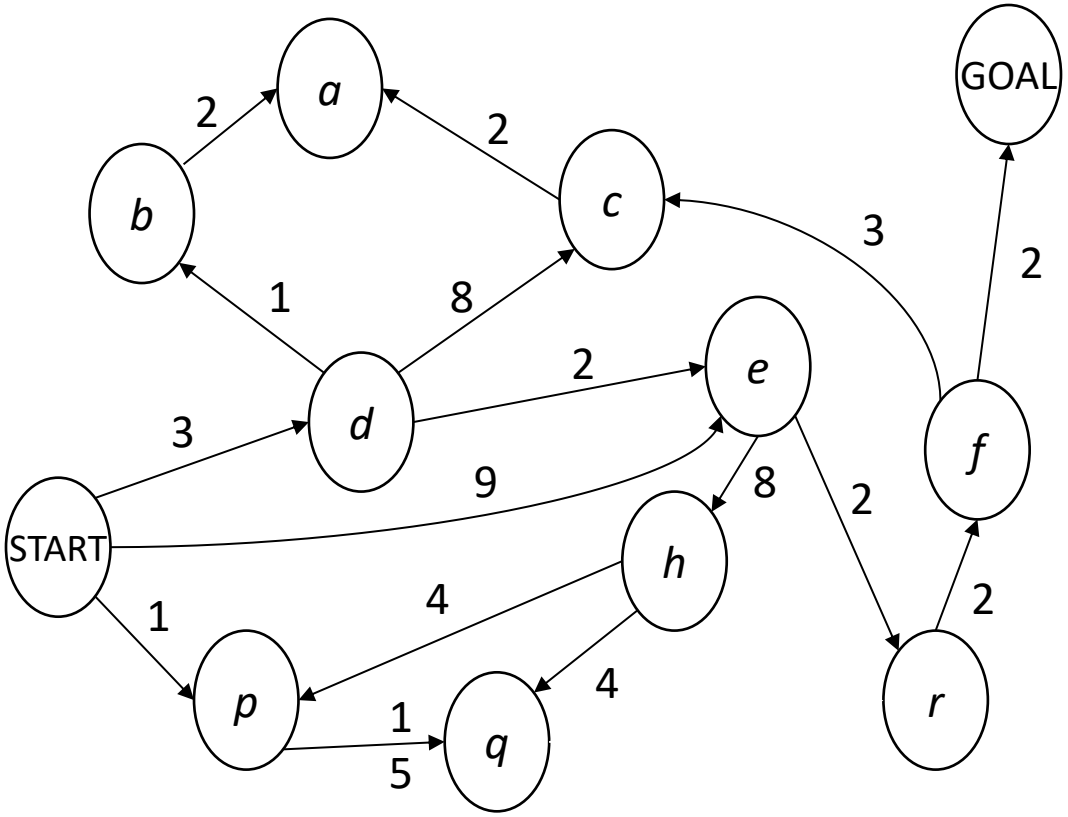
Completitudine? (de la BF)

dacă o soluție există atunci s este finit, deci DA!

Optimalitate? (de la BF)

dacă costurile fiecărei muchii sunt egale (= 1) atunci DA!

Căutare în funcție de cost

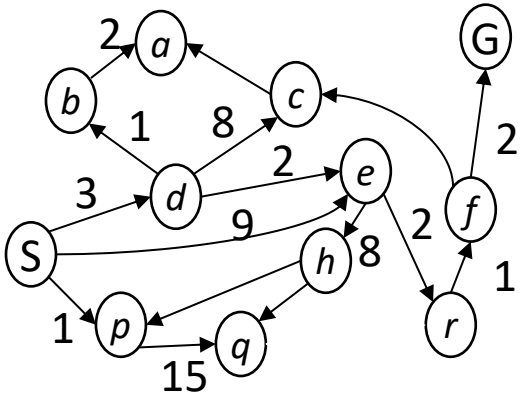


BF găsește cel mai scurt drum relativ la numărul de arce (acțiuni). Nu ia în considerare nici un cost (implicit fiecare arc are același cost). Studiem algoritmi de căutare pe bază de cost.

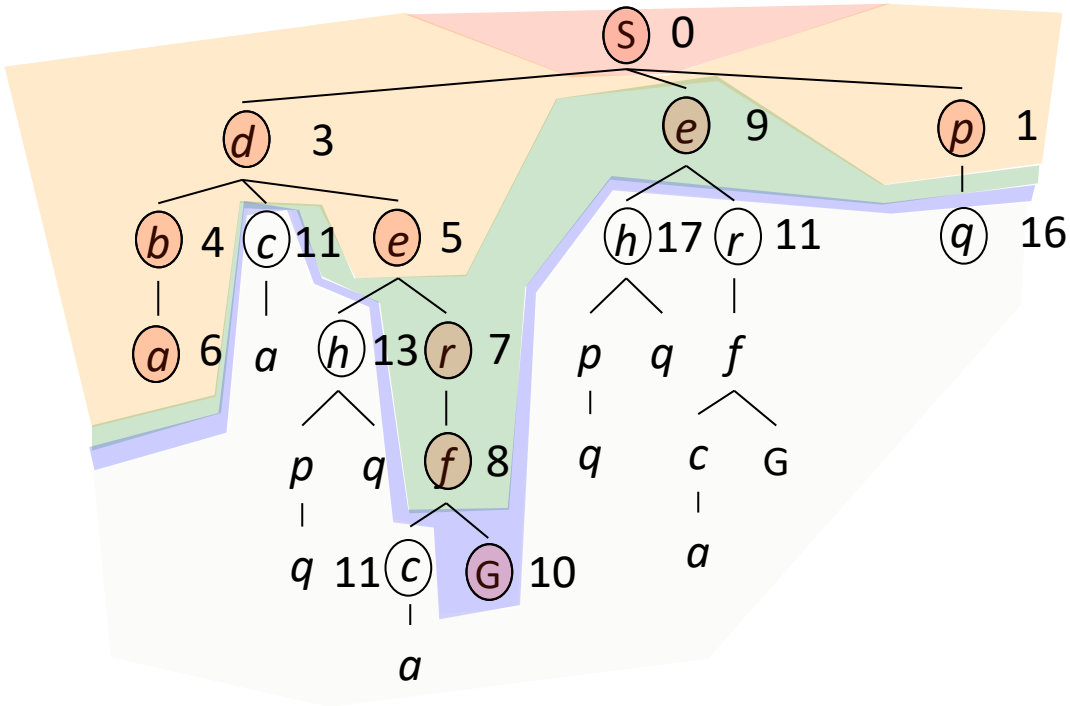
Căutare uniformă după cost (uniform cost search – UCS)

Strategie: expandează nodul care face parte dintr-un drum de cost minim la pasul curent.

Frontiera este implementată cu o coadă de priorități (prioritate: costul minim al unui drum curent)

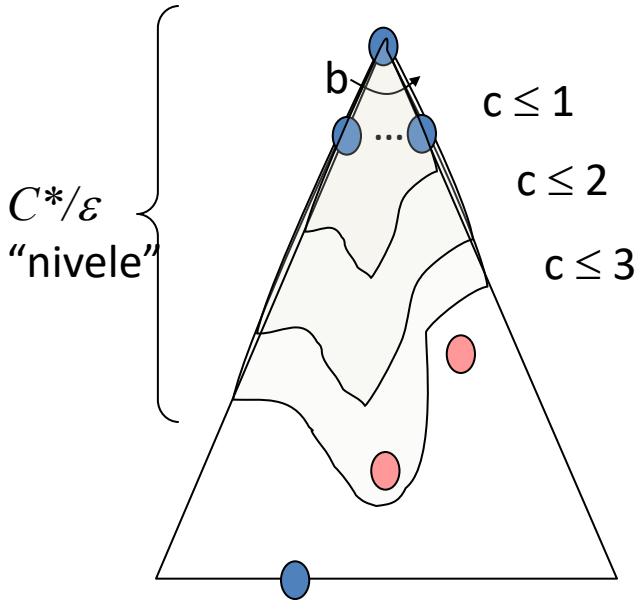


Contur al căutării în funcție de cost



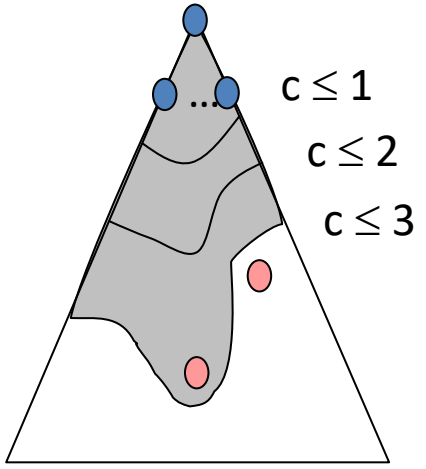
Proprietăți ale căutarii uniforme după cost

- Ce noduri expandează UCS?
  - procesează toate nodurile cu cost mai mic decât soluția de cost minim!
  - dacă soluția de cost minim are costul C^* iar fiecare arc costă cel puțin ε, atunci ajungem la o adâncime în jur de C^*/ε
  - complexitate timp $O(b^{C^*/\varepsilon})$ (exponențial în adâncime)
- Complexitate spațiu (a memoriei pentru frontieră)?
  - La fel ca la BF, aproximativ reține toate nodurile de pe ultimul nivel, deci $O(b^{C^*/\varepsilon})$
- Completitudine?
  - DA (dacă soluția cea mai bună are cost finit iar costurile arcelor sunt pozitive)
- Optimalitate?
  - DA! (demonstrația se leagă de algoritmul A*)



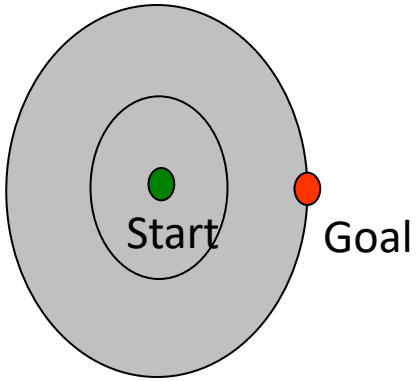
Avantaje/dezavantaje ale căutarii uniforme după cost

- UCS explorează drumuri în ordinea costurilor

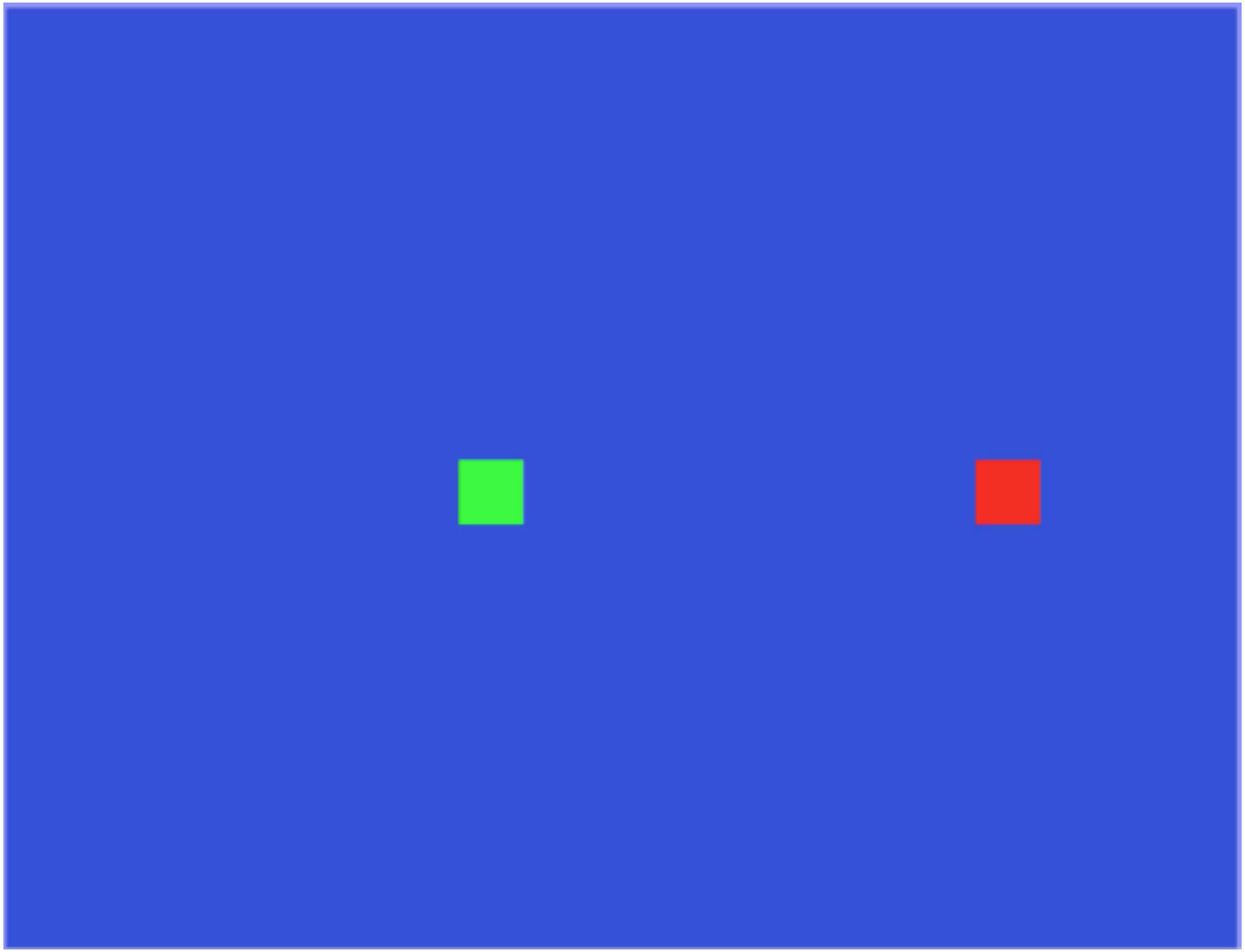


- Avantaje: UCS este complet și optimal!

- Dezavantaje:
  - explorează drumurile în toate direcțiile (căutare uniformă)
  - nu are nicio informație despre poziția nodului scop

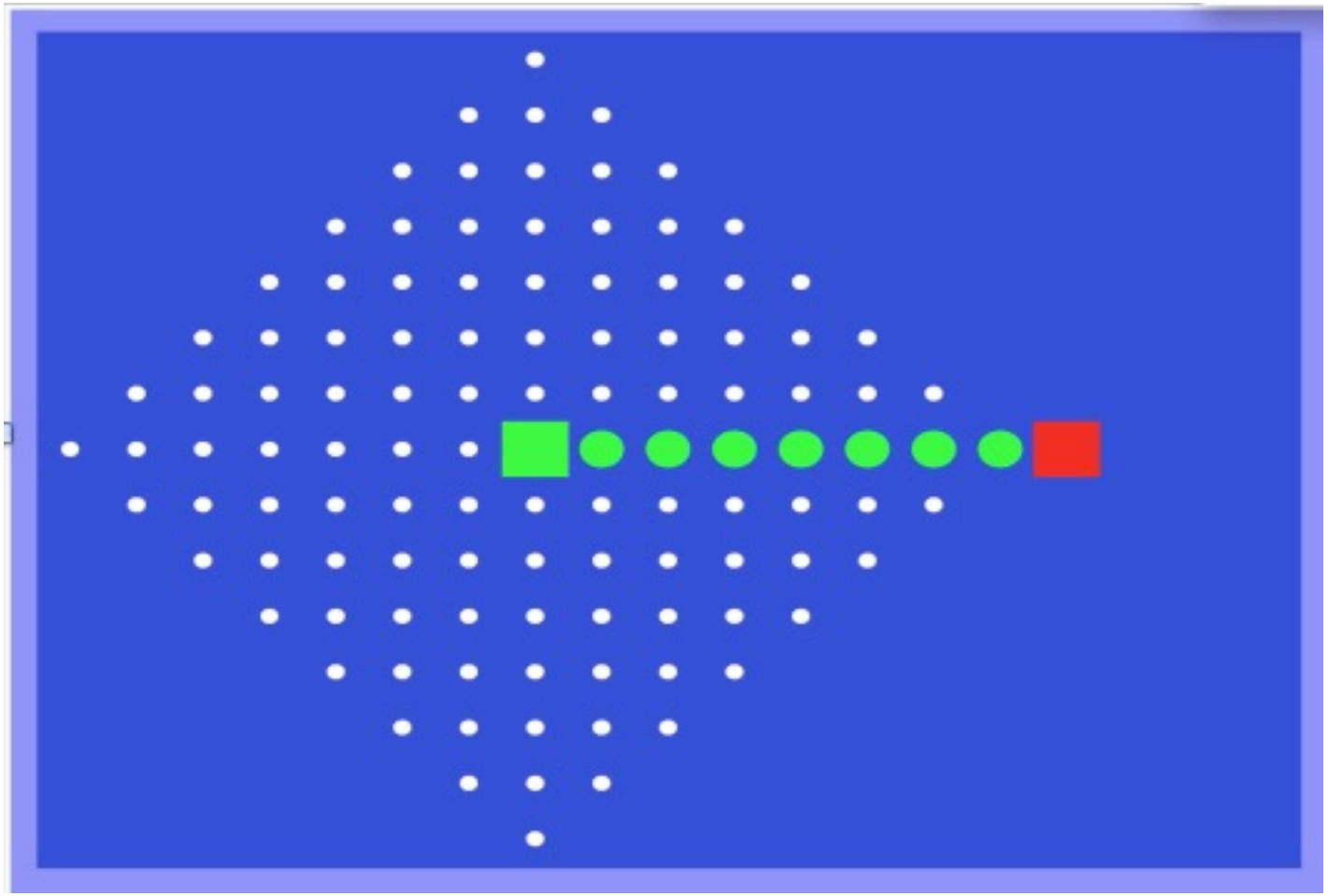


UCS vs BF vs DF



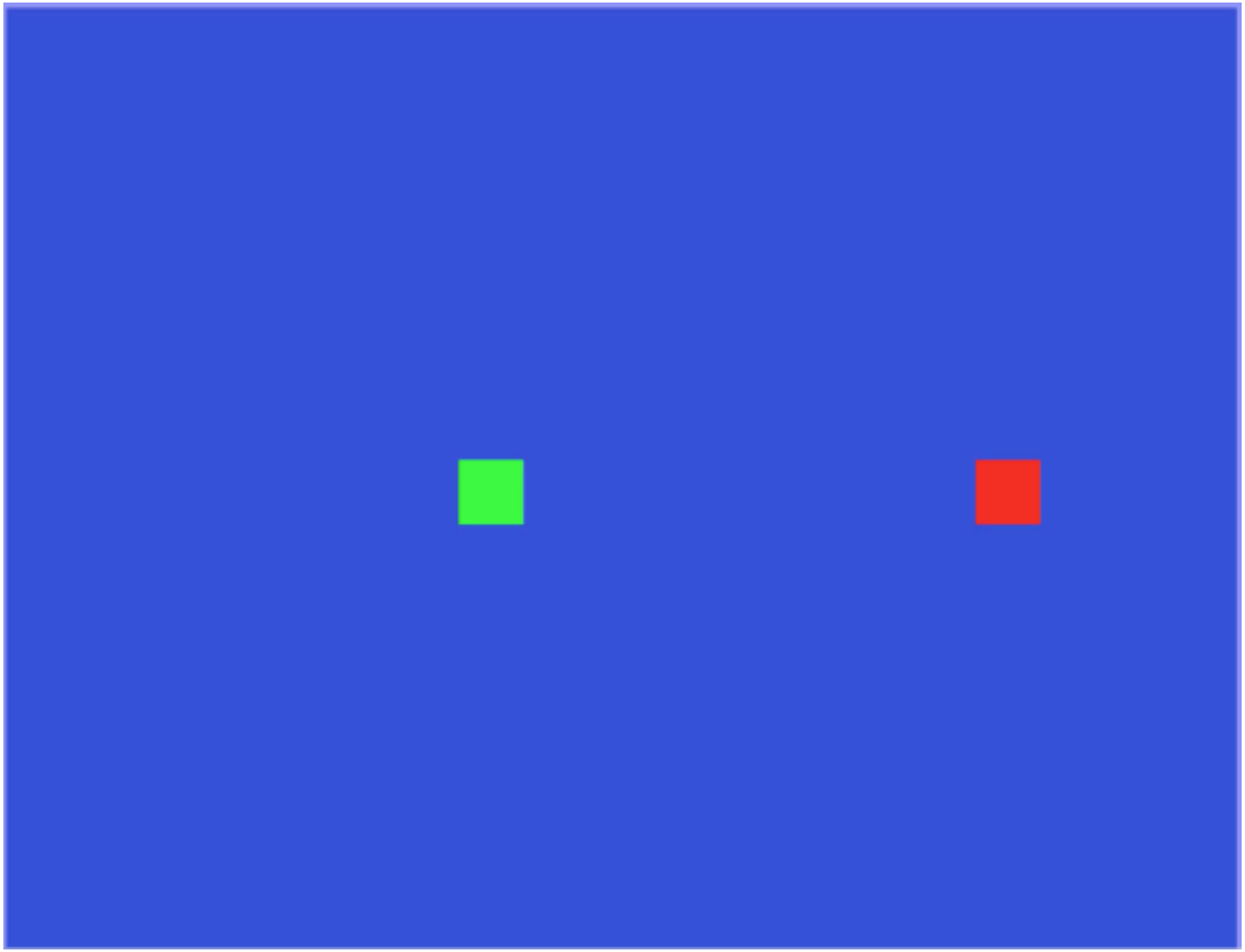
Căutare uniformă după cost - demo

Căutare uniformă după cost - demo



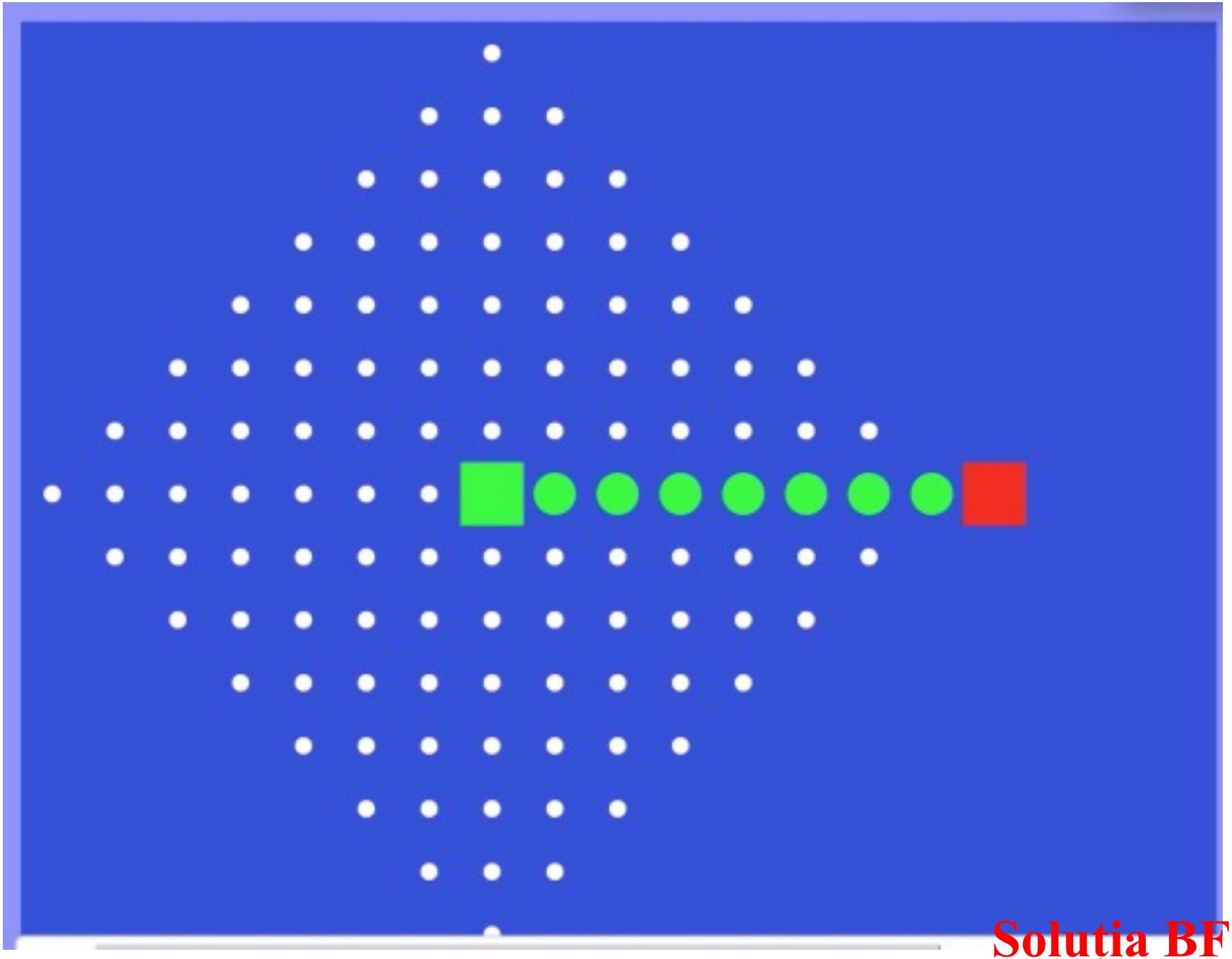
Soluția UCS

BF vs DF



BF vs DF

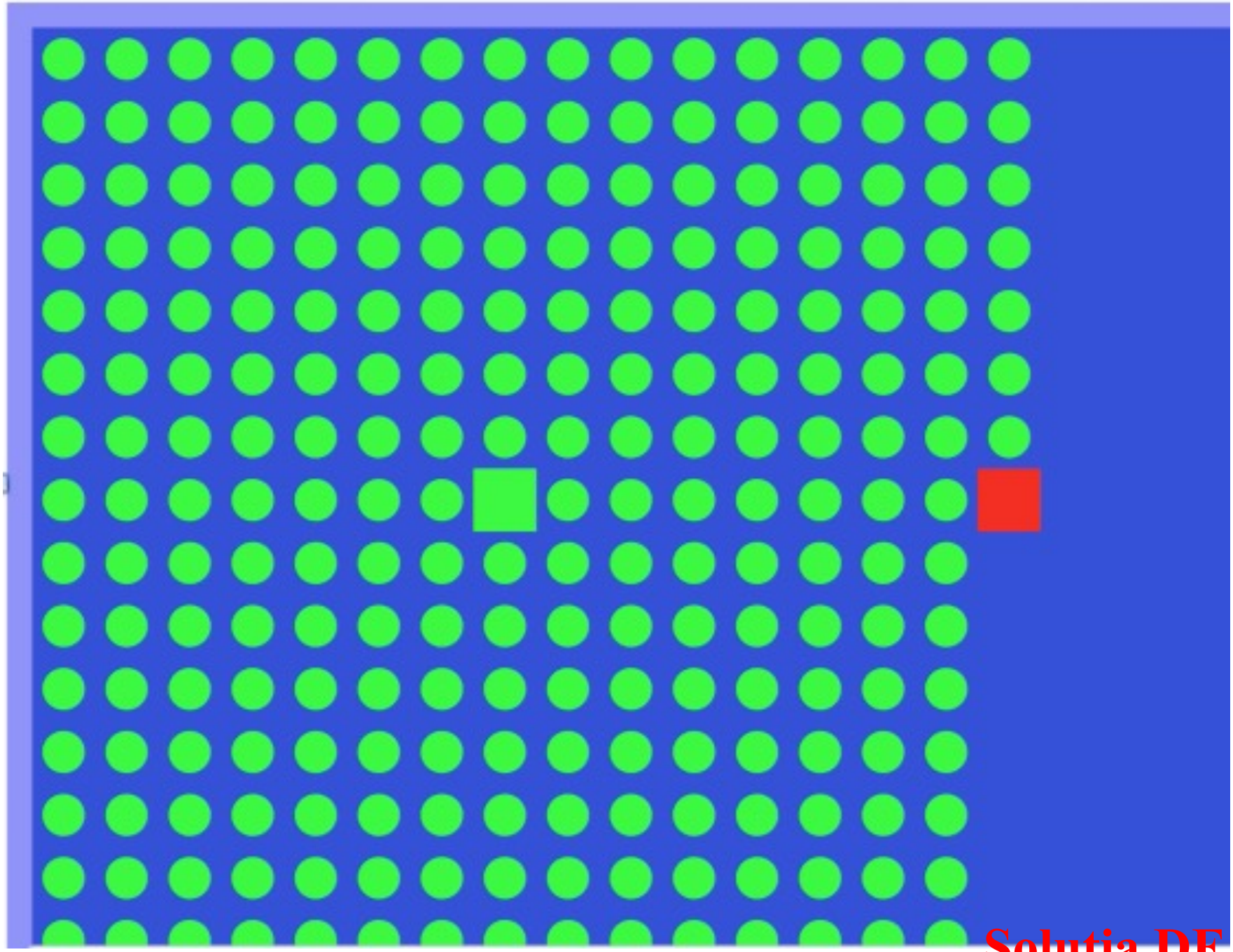
Soluția BF



Soluția BF

BF vs DF

Soluția DF

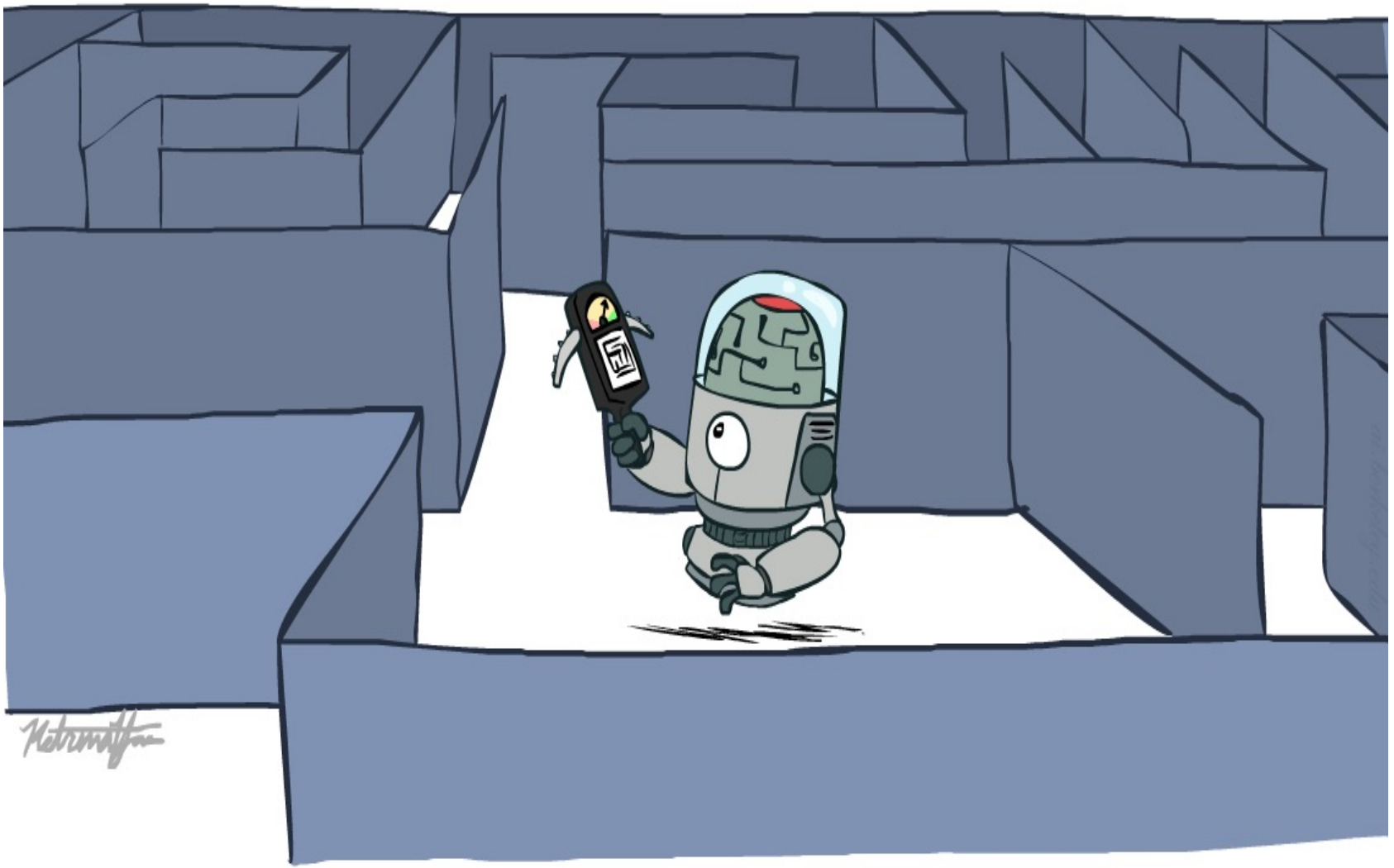

Soluția DF

Compararea strategiilor de căutare neinformată

Criteriu	Căutare în lățime (BF)	Căutare uniformă pe bază de cost (UCS)	Căutare în adâncime (DF)	Căutare în adâncime limitată (DL)	Căutare iterativă în adâncime (ID)
Completitudine	DA[a]	DA[a,b]	NU	NU	DA[a]
Timp	$O(b^d)$	$O(b^{C^*/\varepsilon})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Spațiu	$O(b^d)$	$O(b^{C^*/\varepsilon})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimalitate	DA[c]	DA	NU	NU	DA[c]

b – branching factor (factor de ramificare)
d – adâncimea minimă a unei soluții
m – adâncimea maximă în arborele de căutare
DA[a] – complet dacă b este finit
DA[b] – complet dacă fiecare arc are costul $\geq \varepsilon$
DA[c] – optimal dacă toate costurile sunt egale = 1

Căutare informată



Arbore de căutare general

funcția ARBORE-CAUTARE (**problema**, **strategie**) **returnează** o soluție sau eșec

inițializează **frontiera** folosind **starea inițială a problemei**

ciclează
dacă frontiera este vidă **atunci returnează** eșec
alege un **nod frunză** conform **strategiei** și elimină-l din **frontieră**
dacă nodul conține o **stare scop**
atunci returnează soluția corespunzătoare
expandează nodul ales, adăugând nodurile succesor generate în **frontieră**

Frontiera = structură de date ce păstrează toate nodurile care mai trebuie expandate
Strategia de căutare = precizează ordinea în care expandăm/explorăm nodurile

Arbore de căutare general

funcția ARBORE-CAUTARE (**problema**, **strategie**) **returnează** o soluție sau eșec

inițializează **frontiera** folosind **starea inițială a problemei**
inițializează mulțimea de noduri explorate cu mulțimea vidă
ciclează
dacă frontiera este vidă **atunci returnează** eșec
alege un **nod frunză** conform **strategiei** și elimină-l din **frontieră**
dacă nodul conține o **stare scop**
atunci returnează soluția corespunzătoare
adaugă nodul curent mulțimii de noduri explorate (vizitate)
expandează nodul ales, adăugând nodurile succesor generate în **frontieră**
dacă nodurile nu sunt în frontieră sau în mulțimea de noduri explorate

Frontiera = structură de date ce păstrează toate nodurile care mai trebuie expandate
Strategia de căutare = precizează ordinea în care expandăm/explorăm nodurile

Căutare informată vs căutare neinformată

Frontiera = structură de date ce păstrează toate nodurile care mai trebuie expandate

Strategia de căutare = precizează ordinea în care expandăm/explorăm nodurile

- funcție de evaluare f(n) – evaluează nodul n
- alege un nod n pe baza lui f(n)

Căutare neinformată

- f(n) nu codează informații specifice despre problemă
- BF: f(n) = primul nod dintr-o coadă, nod de adâncime minimă
- DF: f(n) = primul nod dintr-o stivă, nod de adâncime maximă
- UCS: f(n) = costul nodului, listă de priorități

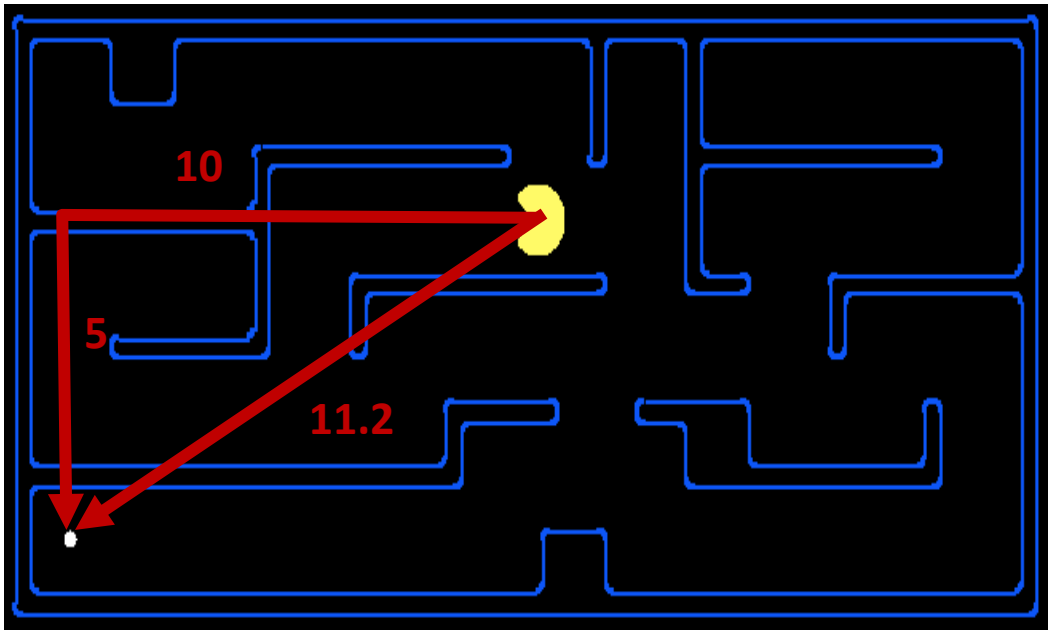
Căutare informată

- f(n) codează informații specifice despre problemă
- includem în f(n) și euristici (nu le avem la căutare neinformată)

Euristici de căutare

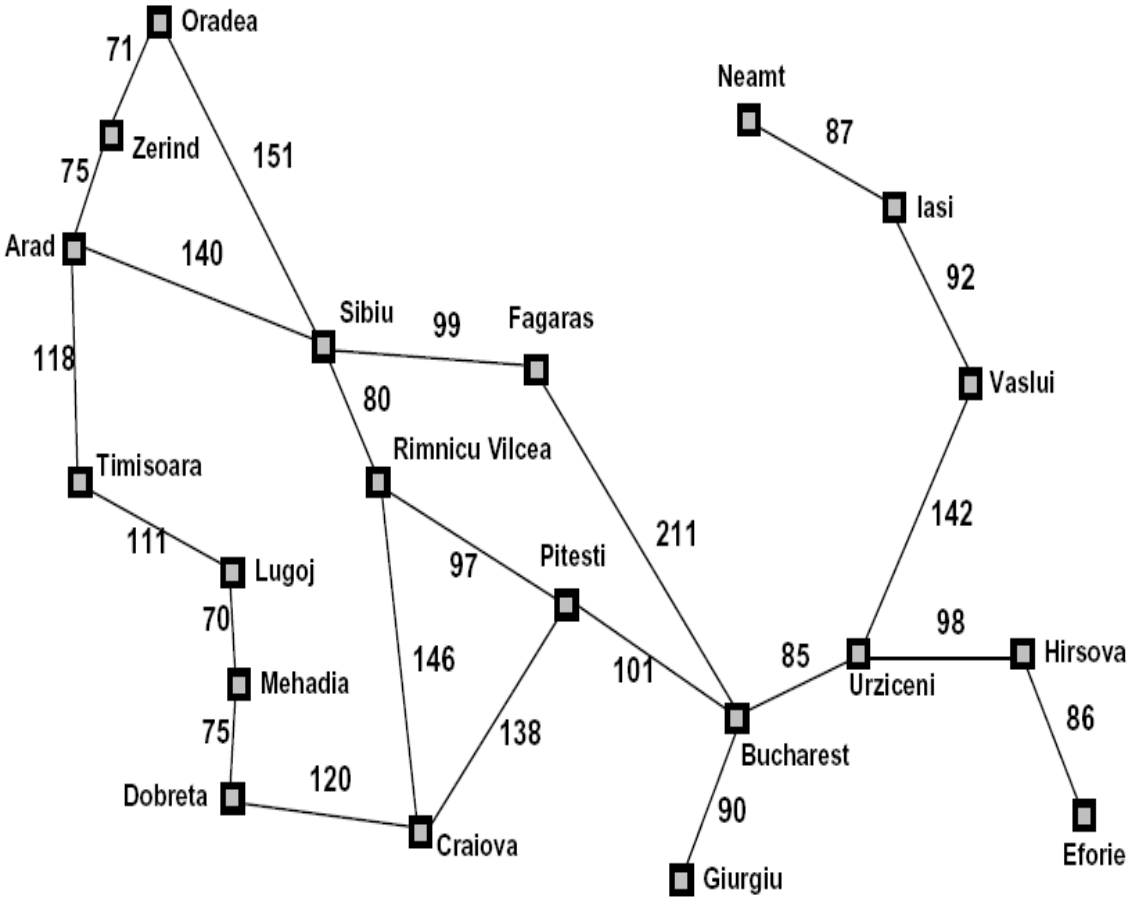
O euristică este:

- o funcție h care *estimează* cât de aproape suntem de o stare-scop;
- h(n) ≥ 0, pentru orice stare n;
- h(stare-scop) = 0;
- specifică pentru fiecare problemă;
- exemple: distanța Manhattan, distanța Euclideană pentru găsirea celui mai scurt drum;



Exemplu de funcție euristică

h(x) = distanța în linie dreaptă între orașul x și București

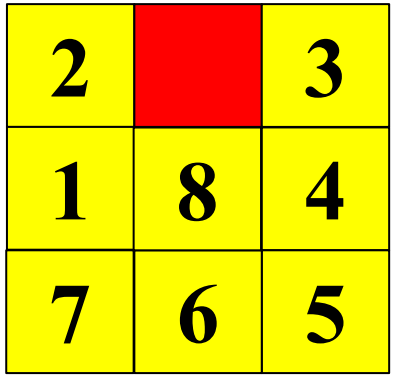


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

Exemplu de funcție euristică

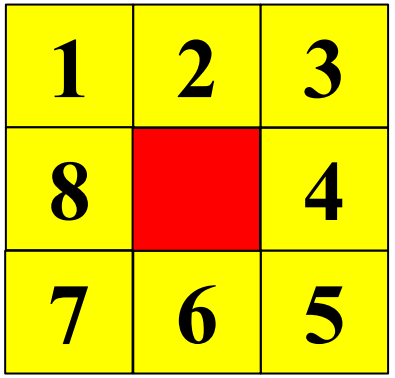
- Pentru problema 8-puzzle putem folosi câteva euristici:
  - h_1: numărul de piese așezate greșit față de starea-scop
  - h_2: suma distanțelor pieselor față de poziția lor în starea scop (suma distanțelor Manhattan)



Stare inițială

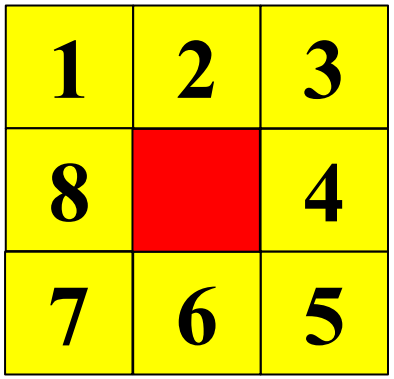
$h_1 = 4$

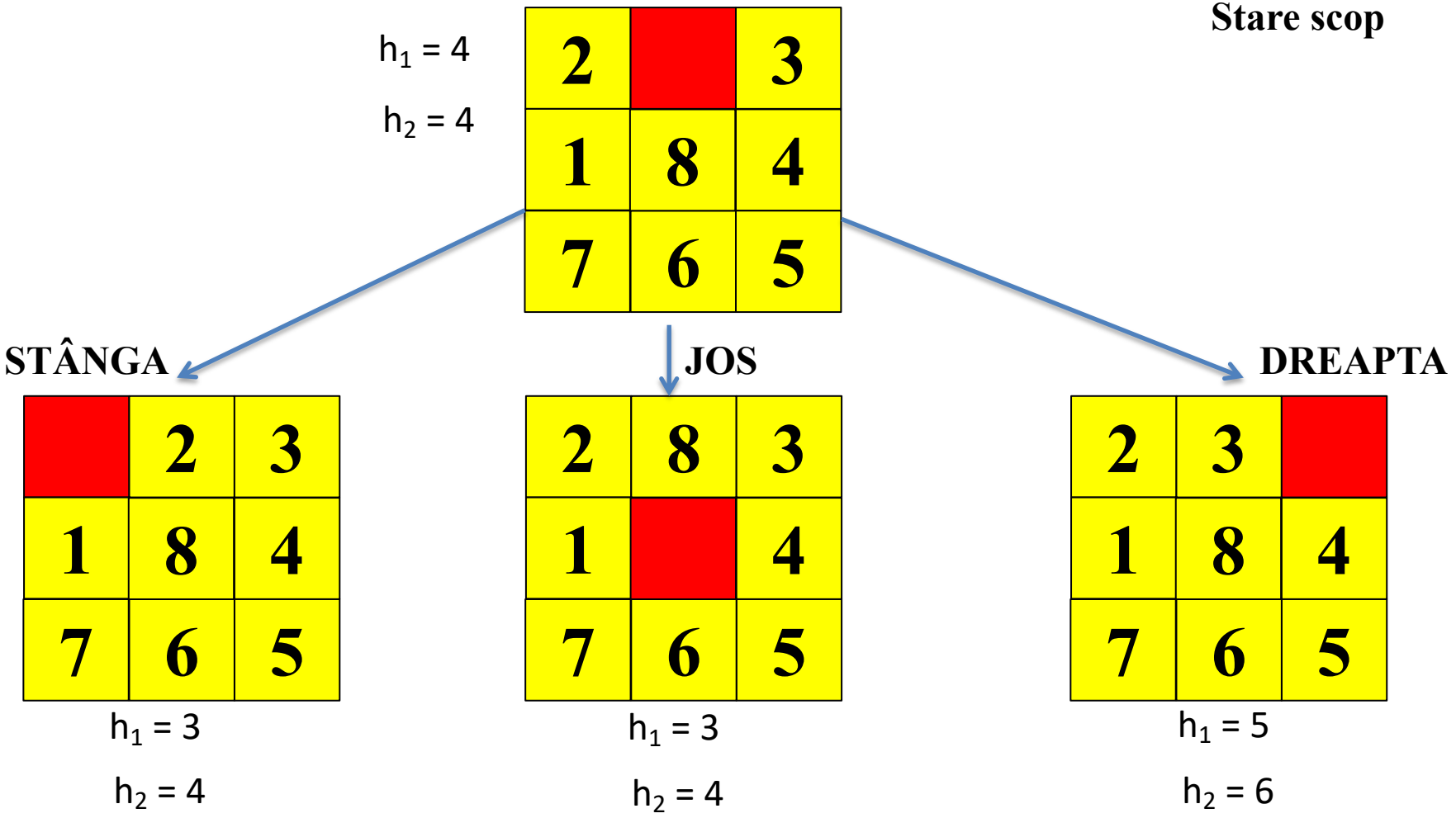
$h_2 = 1 + 1 + 0 + 0 + 0 + 0 + 0 + 1 + 1 = 4$



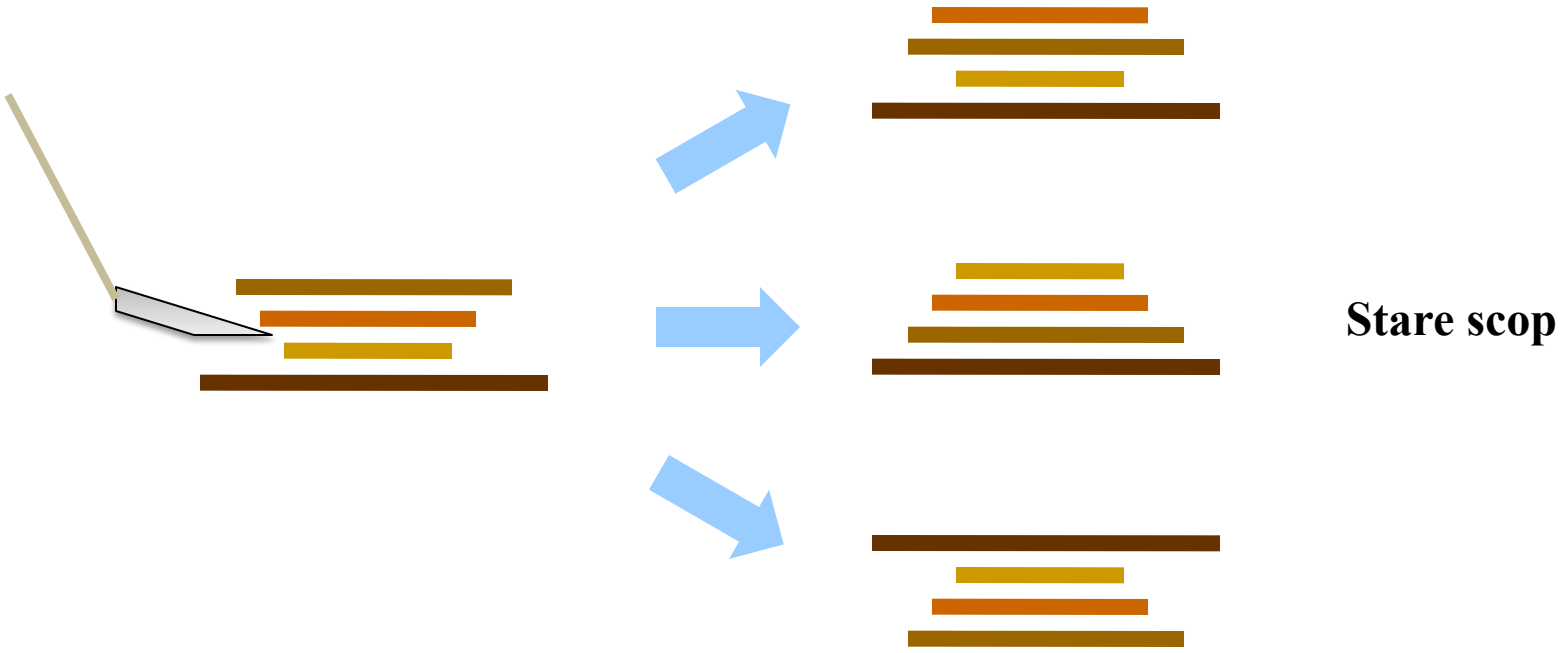
Stare scop

Exemplu de funcție euristică





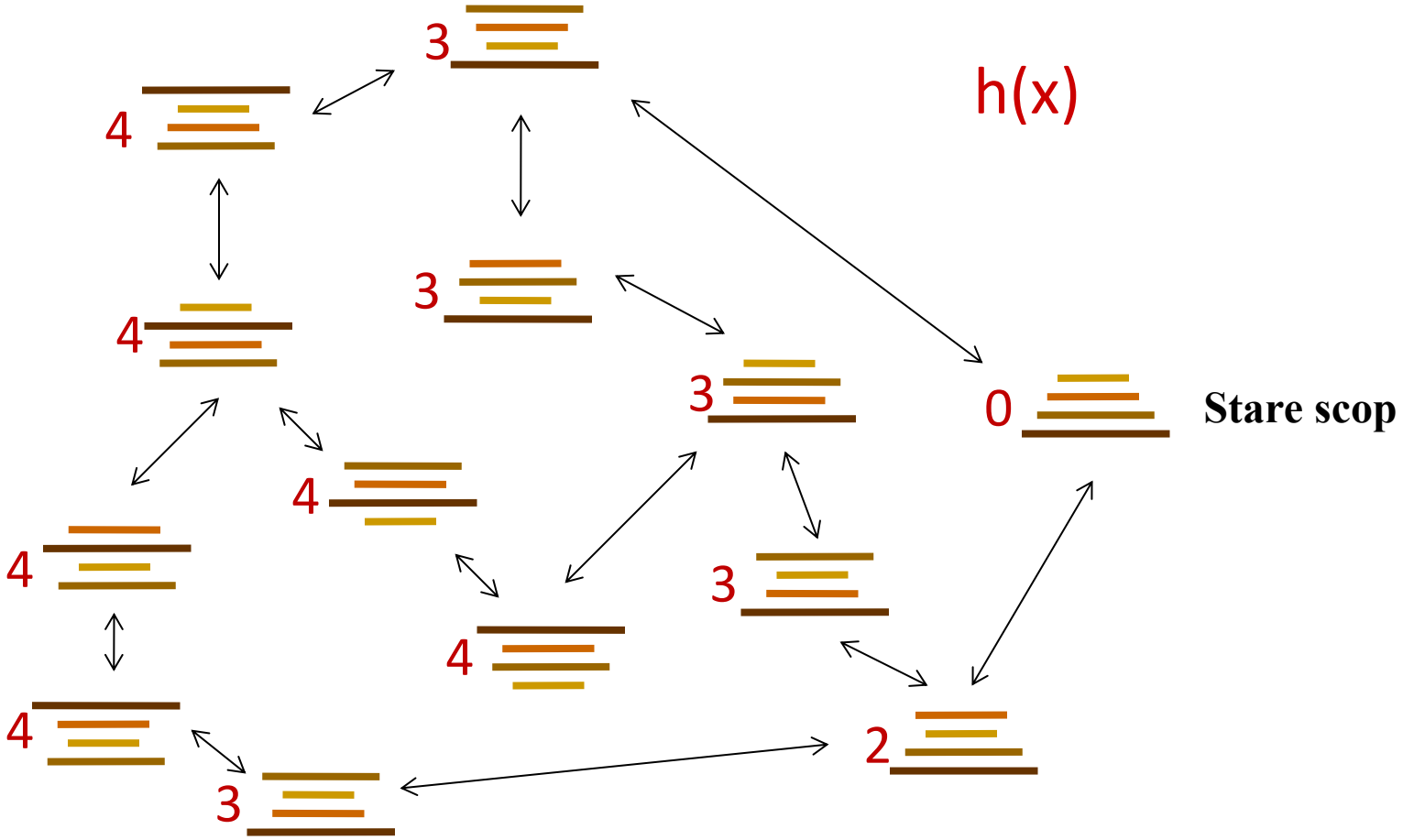
Problema sortării clătitelor



Cost: numărul total de clătite întoarse

Exemplu de funcție euristică

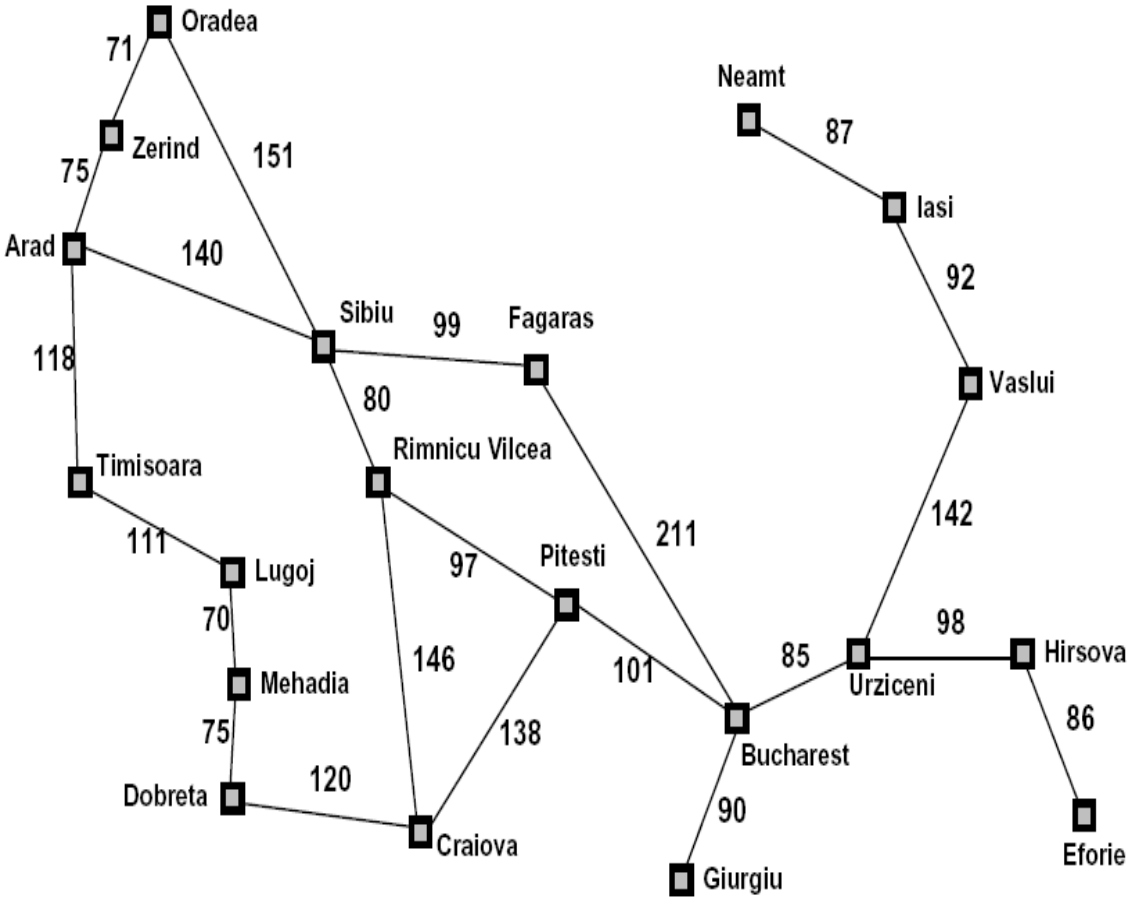
h(x): care este clătita de diametru cel mai mare din configurația x care nu este la locul ei?



Căutare Greedy (best-first)

Exemplu de funcție euristică

h(x) = distanța în linie dreaptă între orașul x și București

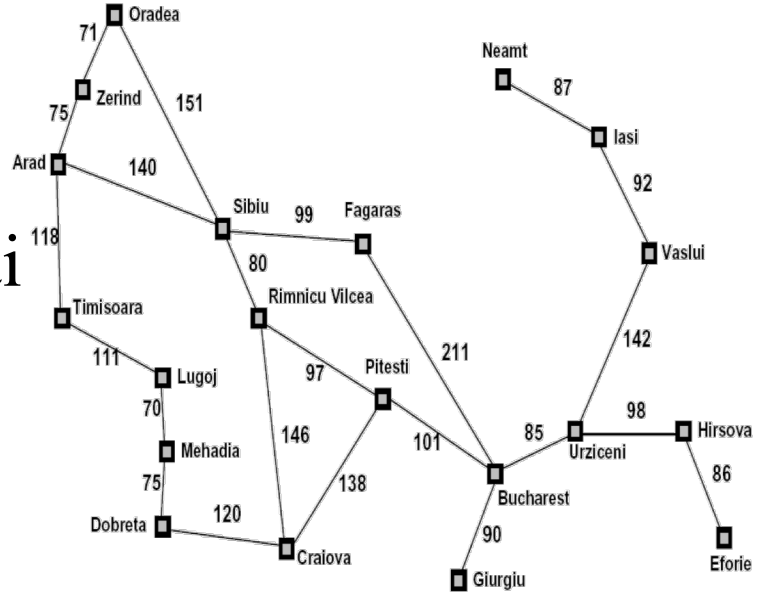


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

Căutare Greedy

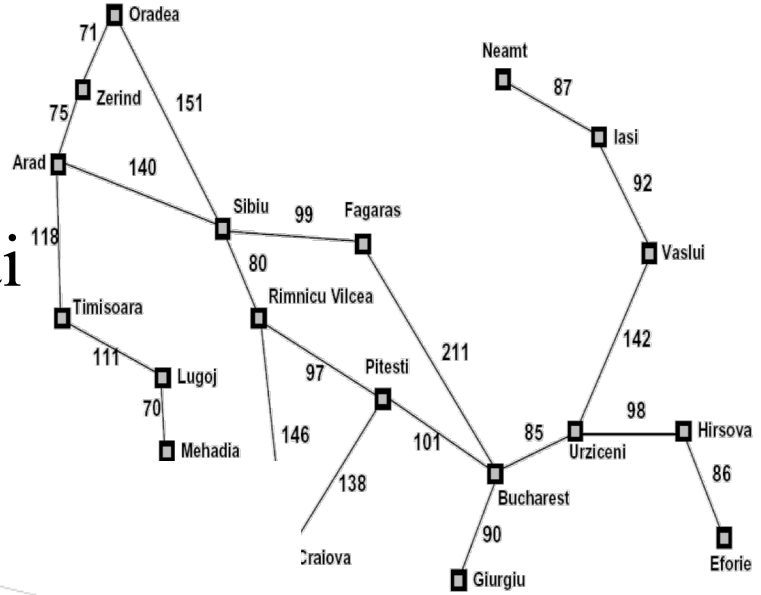
- Explorează nodul care pare cel mai aproape de un nod-scop (dpdv al euristicii, ea ghidează căutarea)

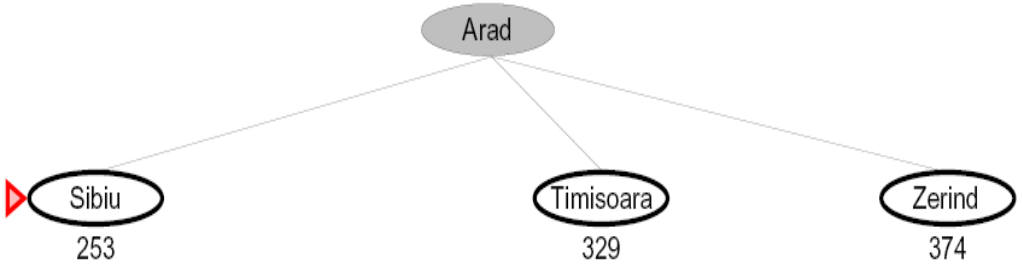


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Căutare Greedy

- Explorează nodul care pare cel mai aproape de un nod-scop (dpdv al euristicii)

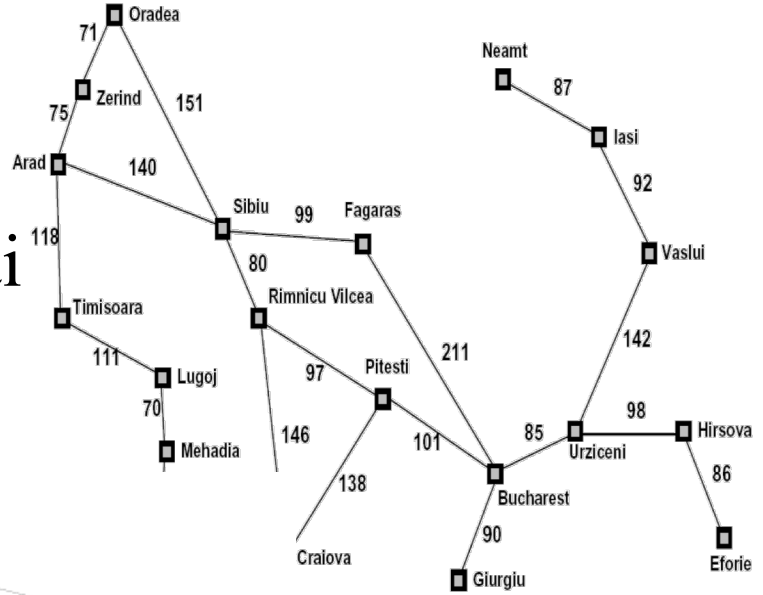


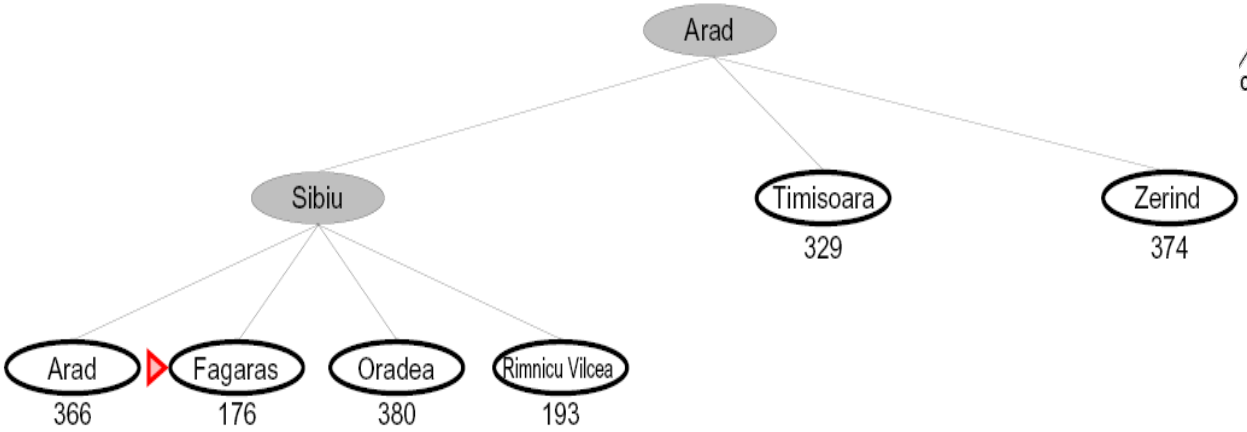


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Căutare Greedy

- Explorează nodul care pare cel mai aproape de un nod-scop (dpdv al euristicii)

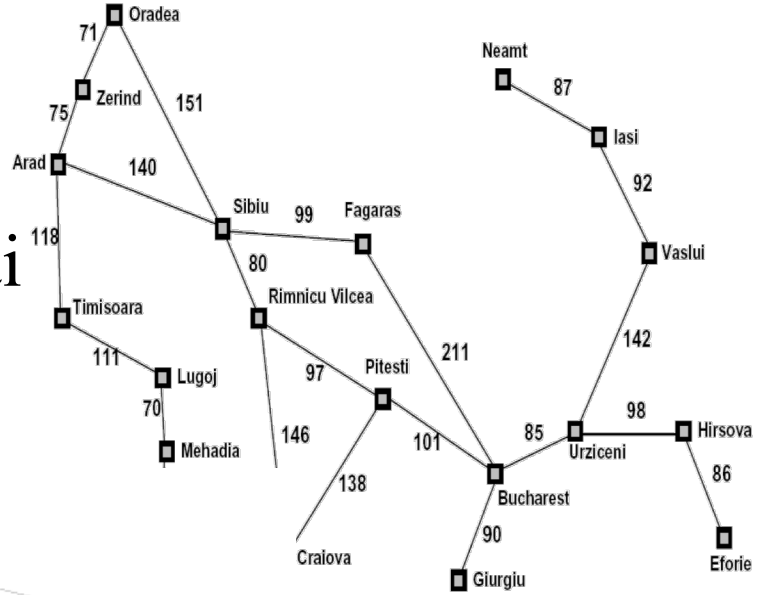


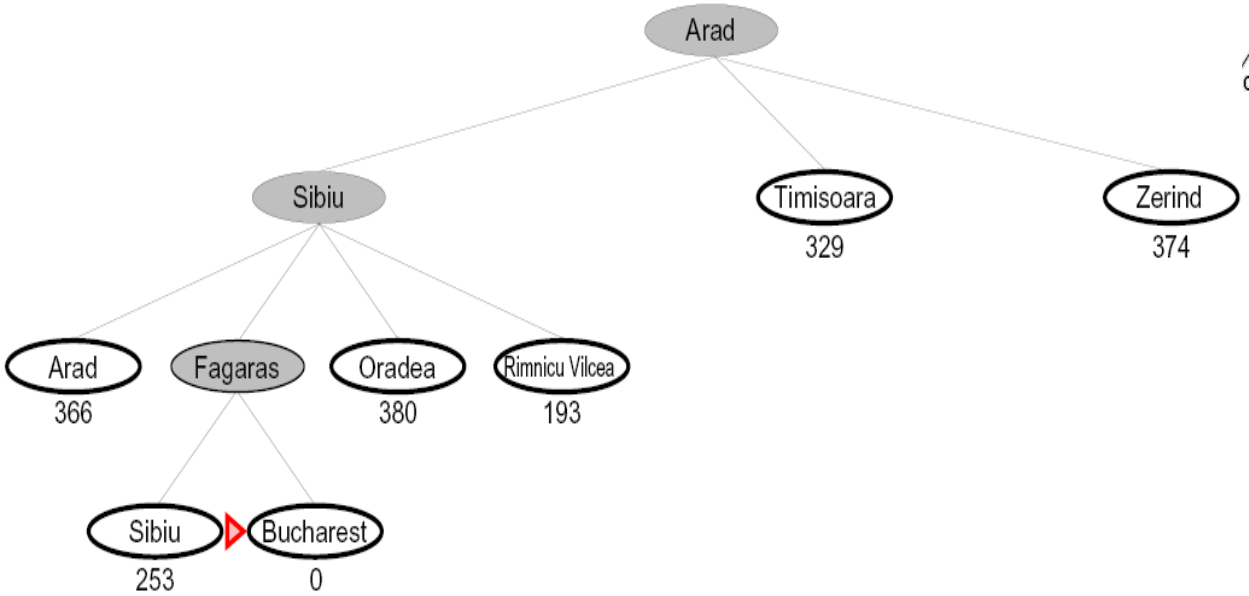


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Căutare Greedy

- Explorează nodul care pare cel mai aproape de un nod-scop (dpdv al euristicii)



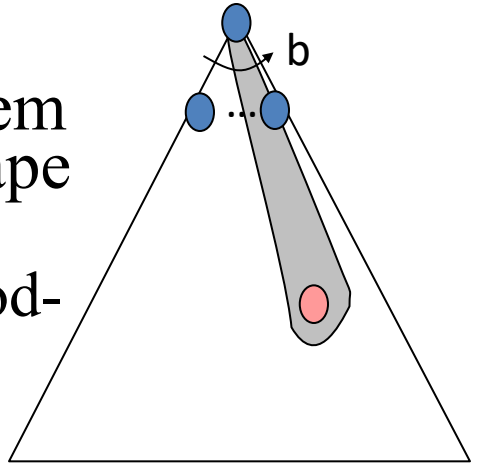


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

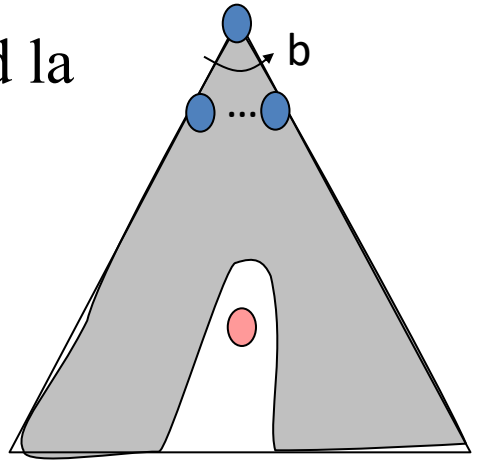
- Ce se poate întâmpla?
  - să obținem o soluție suboptimă: Arad – Sibiu – Făgăraș - București, cu un cost mai mare decât soluția optimă: Arad – Sibiu – Râmnicu-Vâlcea-Pitești-București

Căutare Greedy

- Strategia: expandează un nod care credem (dpdv al euristicii) că este cel mai aproape de un nod-scop
  - euristică: distanță estimată față de un nod-scop pentru fiecare nod curent



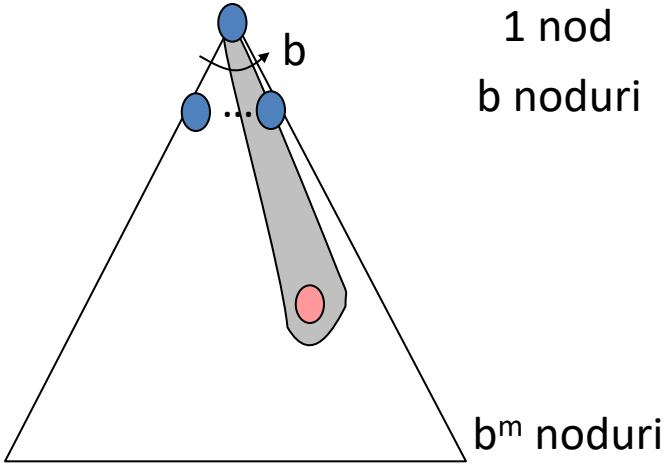
- Caz uzual:
  - strategia Greedy (best-first) ajunge rapid la o soluție, dar este suboptimă

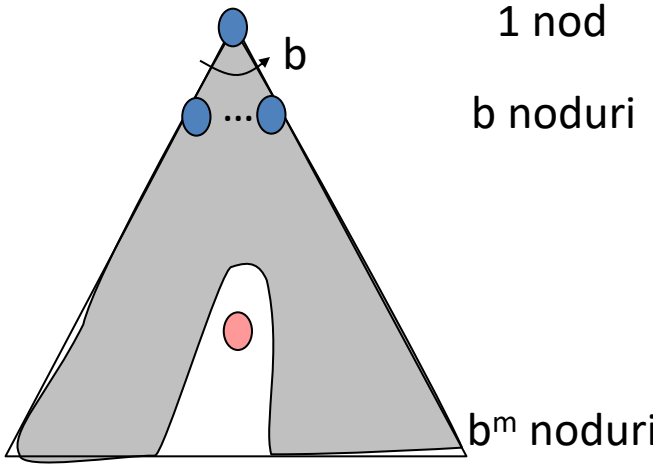


- Caz defavorabil: căutare în adâncime prost ghidată

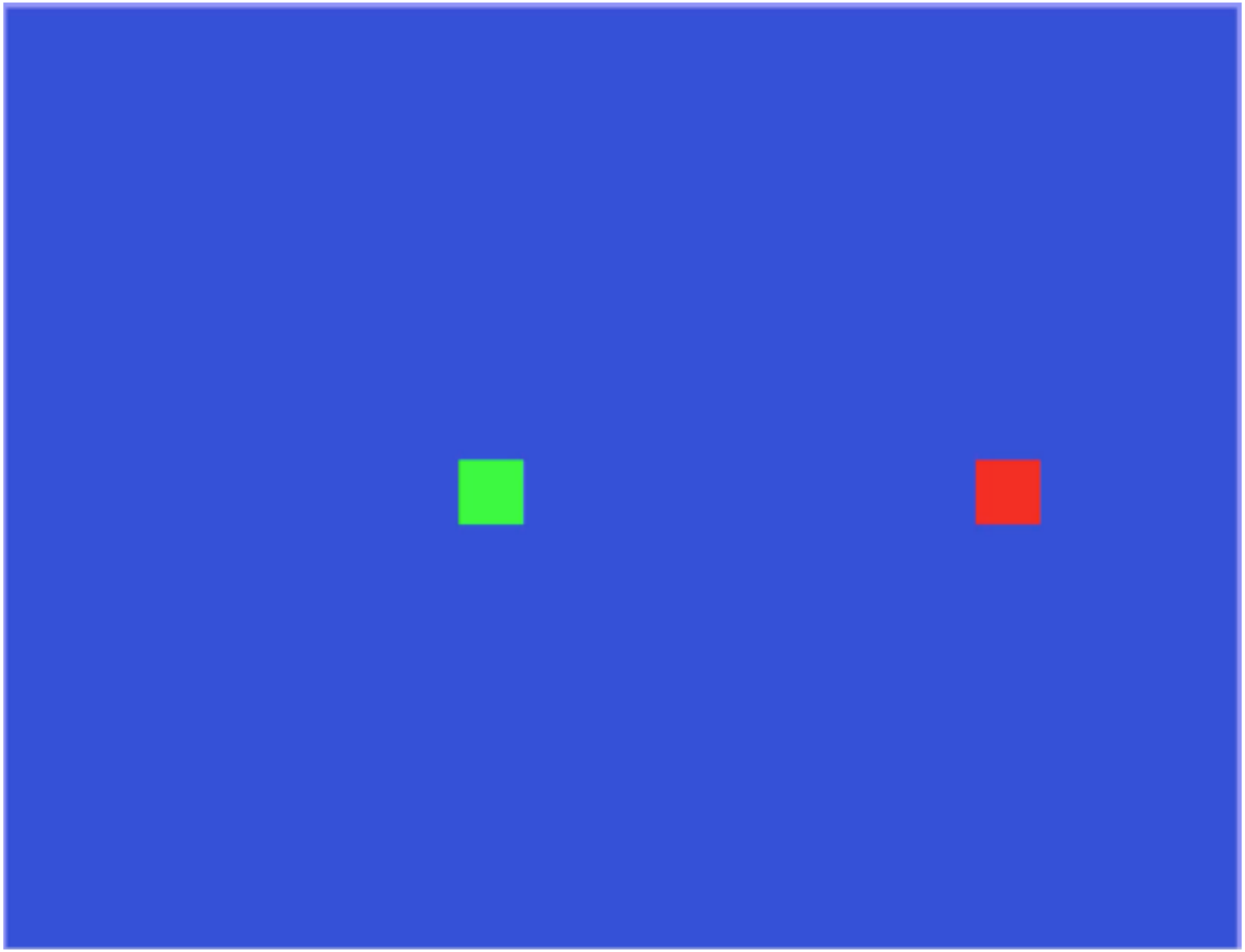
Proprietățile căutării Greedy

- Ce fel de noduri expandează Greedy?
  - poate procesa toate nodurile
  - complexitate timp $O(b^m)$
- Cât de mult spațiu necesită Greedy?
  - este spațiul pentru memorarea frontierei
  - complexitate $O(b^m)$
- Completitudine?
  - dacă o soluție există atunci DA! (dacă ține minte pe unde a fost și nu intră în circuite)
- Optimalitate?
  - NU



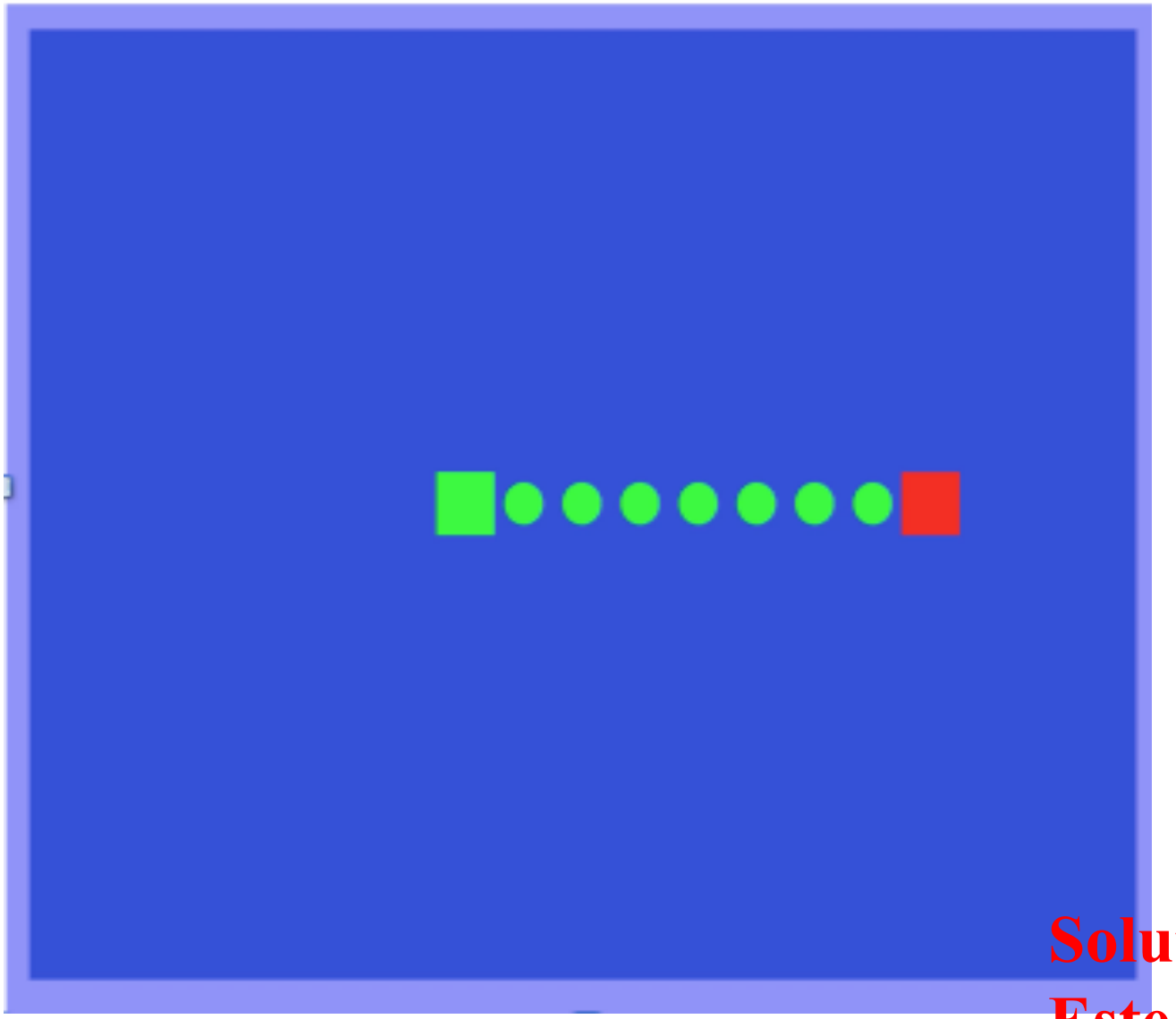


Căutare Greedy - demo



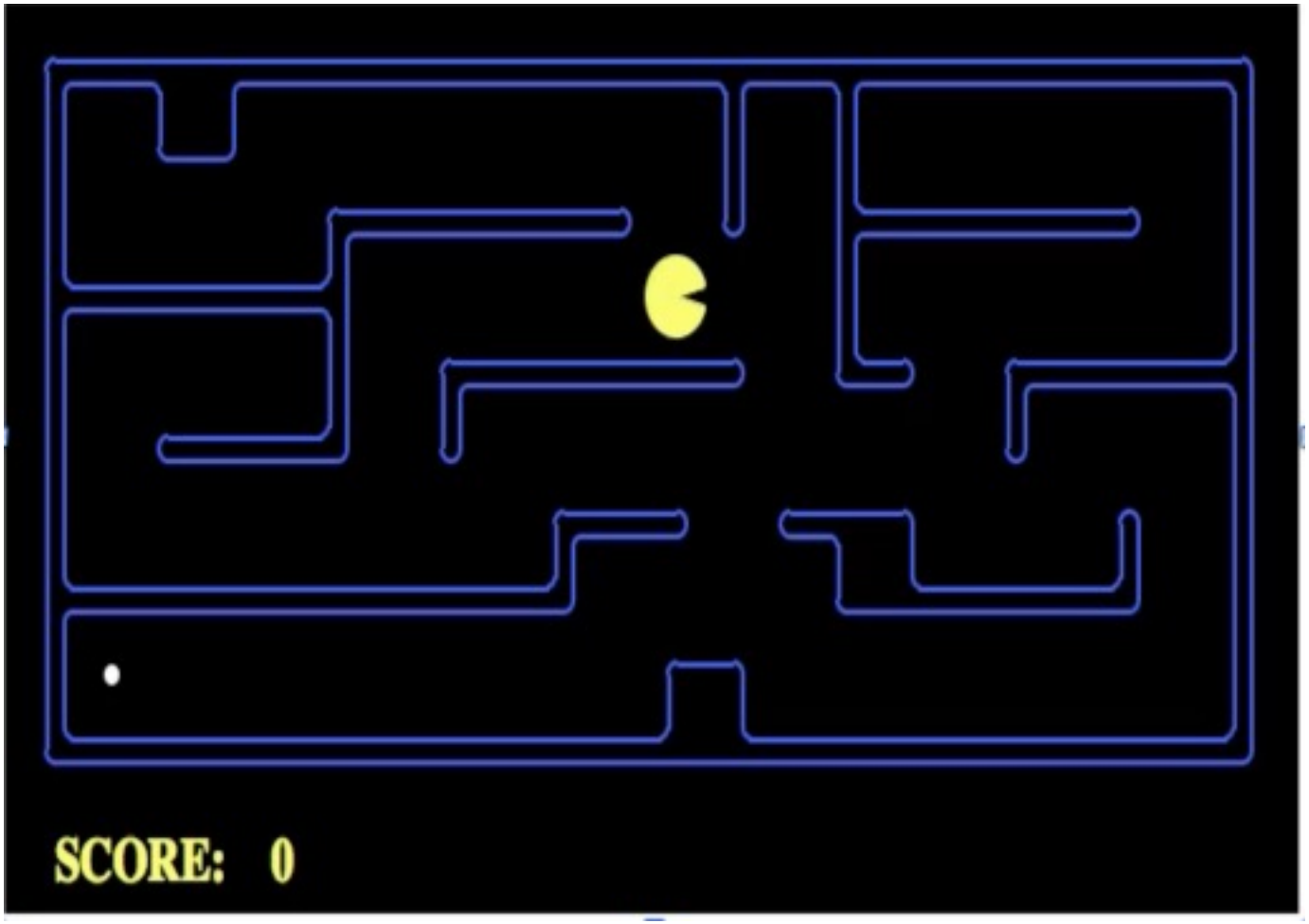
Căutare Greedy - demo

Căutare Greedy - demo



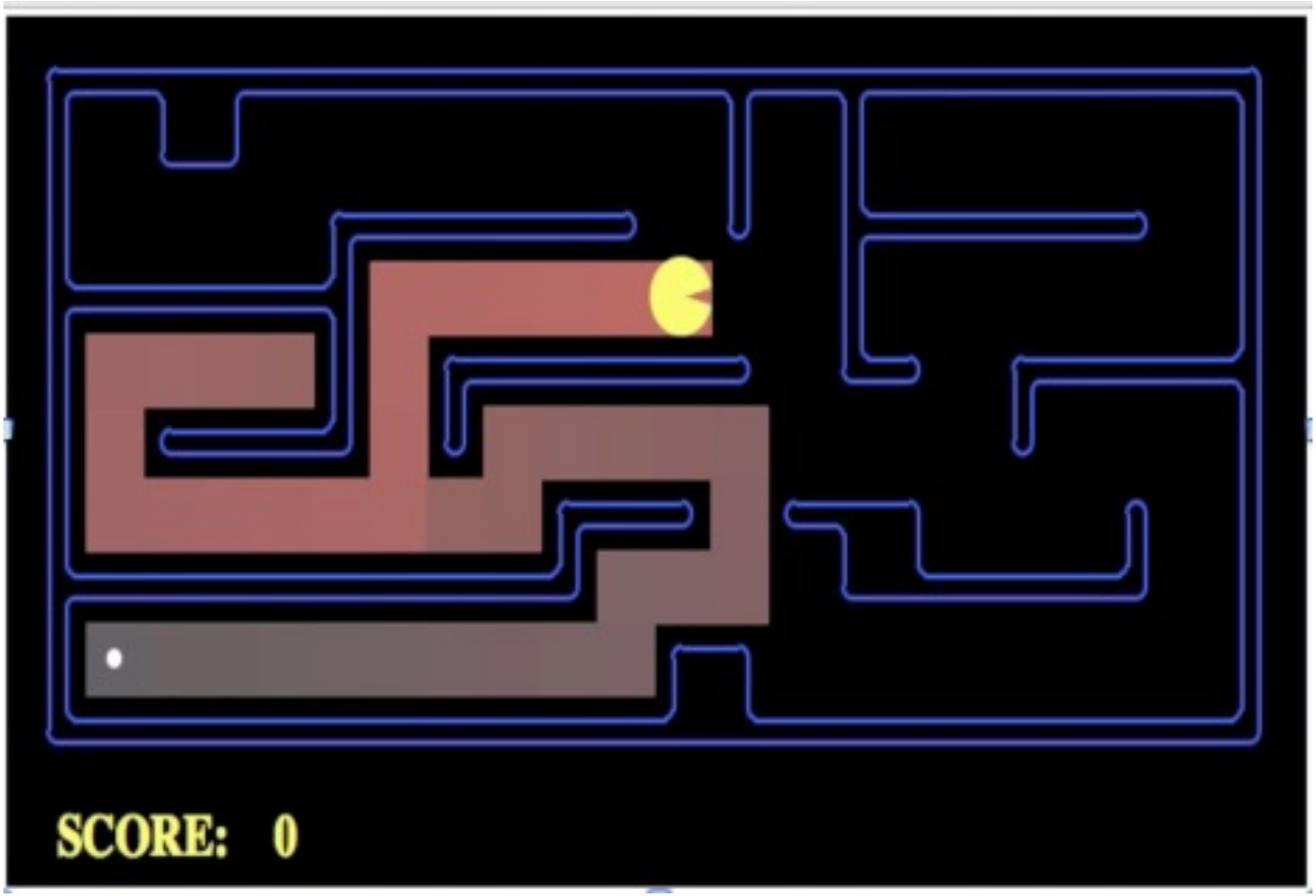
Soluția Greedy
Este optimă

Căutare Greedy - demo



Căutare Greedy - demo

Căutare Greedy - demo



Soluția Greedy
Nu este optimă

Algoritmul de căutare A*

Algoritmul de căutare A*

Algoritmul de căutare A*

UCS

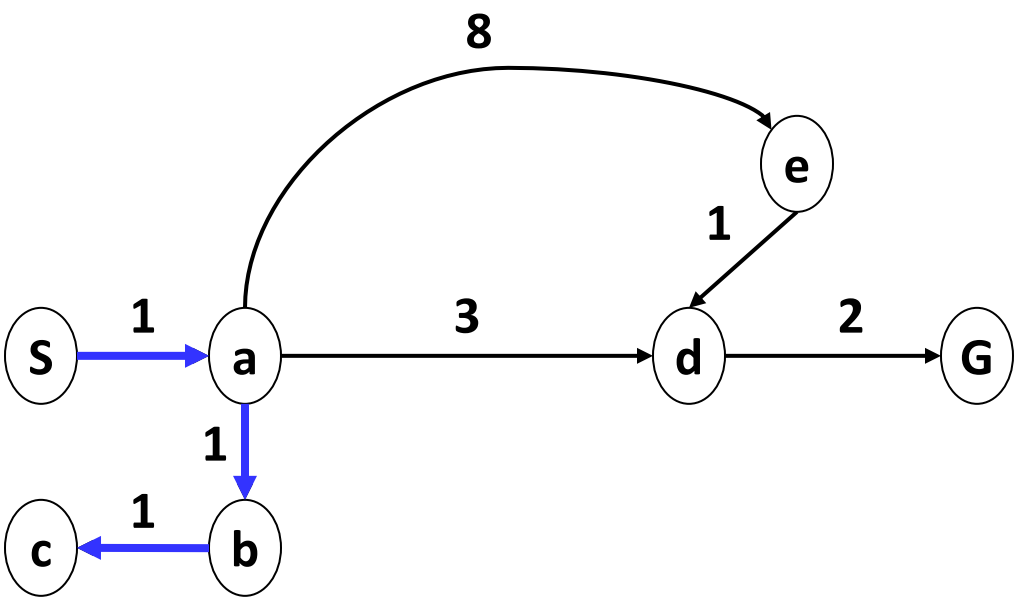
Greedy

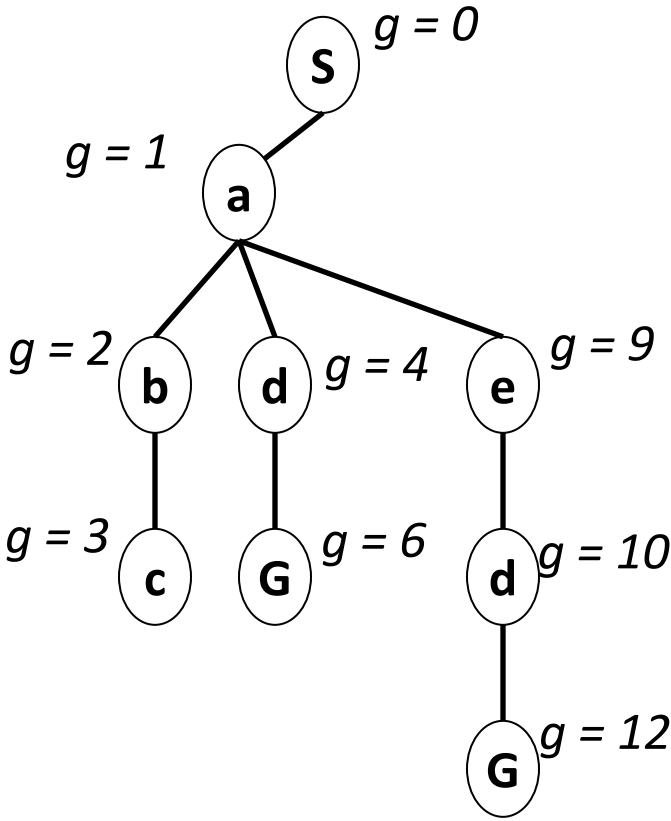
A*

Combinăm UCS + Greedy

- **Căutarea uniformă bazată pe cost** preferă să exploreze soluții parțiale (drumuri) de cost curent minim (se bazează pe un calcul real, nu există estimări)

  g(n) = backward cost = cât costă drumul de la nodul inițial la nodul curent
- Găsește în final soluția de cost minim 6: s➔a➔ d➔G





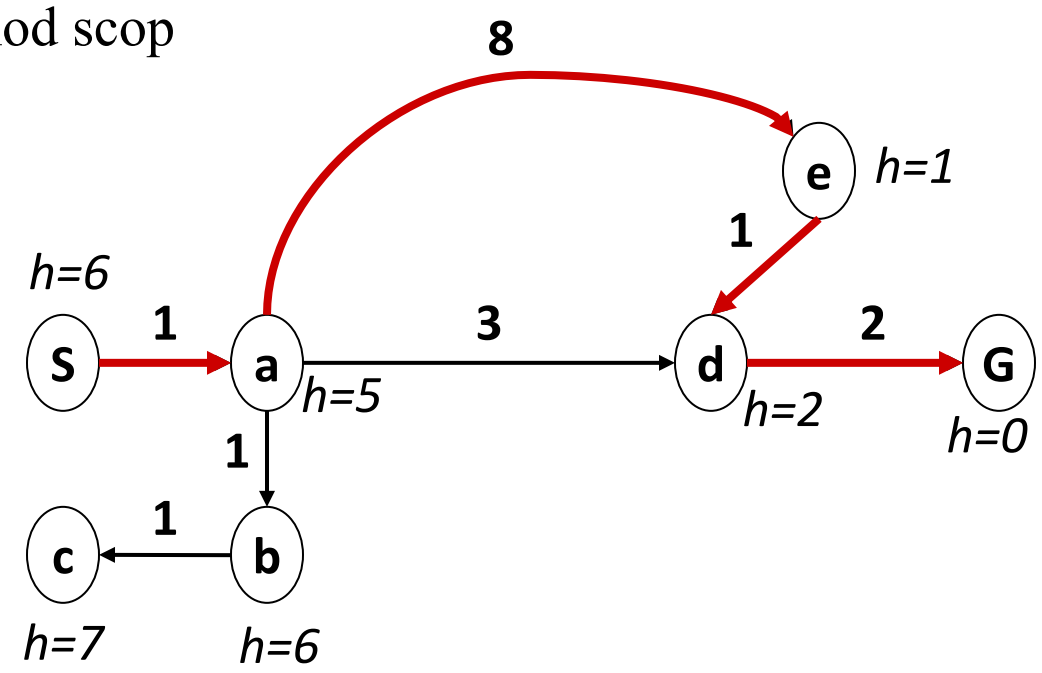
Combinăm UCS + Greedy

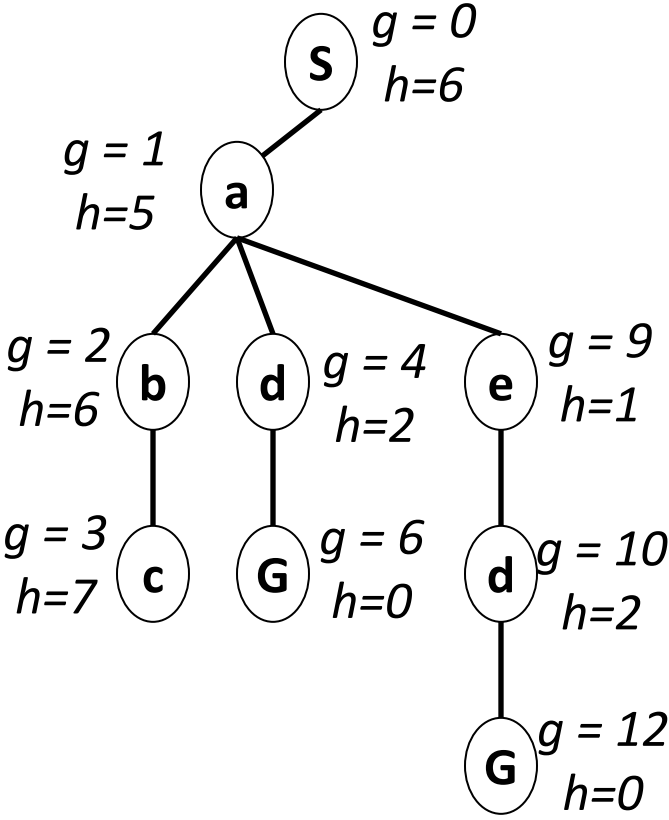
- **Căutarea uniformă bazată pe cost** preferă să exploreze soluții parțiale (drumuri) de cost curent minim (se bazează pe un calcul real, nu există estimări)

  g(n) = backward cost = cât costă drumul de la nodul inițial la nodul curent

- **Căutarea Greedy** preferă să exploreze soluții parțiale (drumuri) pentru care nodul curent este cel mai apropiat de un nod scop (se bazează pe o estimare)

  h(n) = forward cost = cât estimez că va costa drumul de la nodul curent la un nod scop





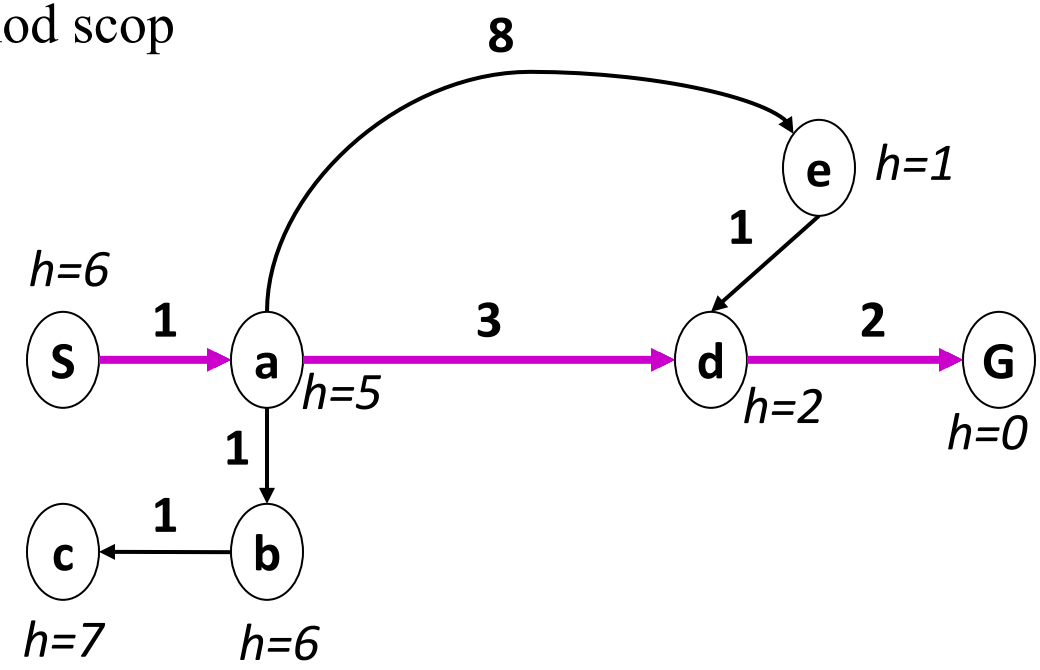
Combinăm UCS + Greedy

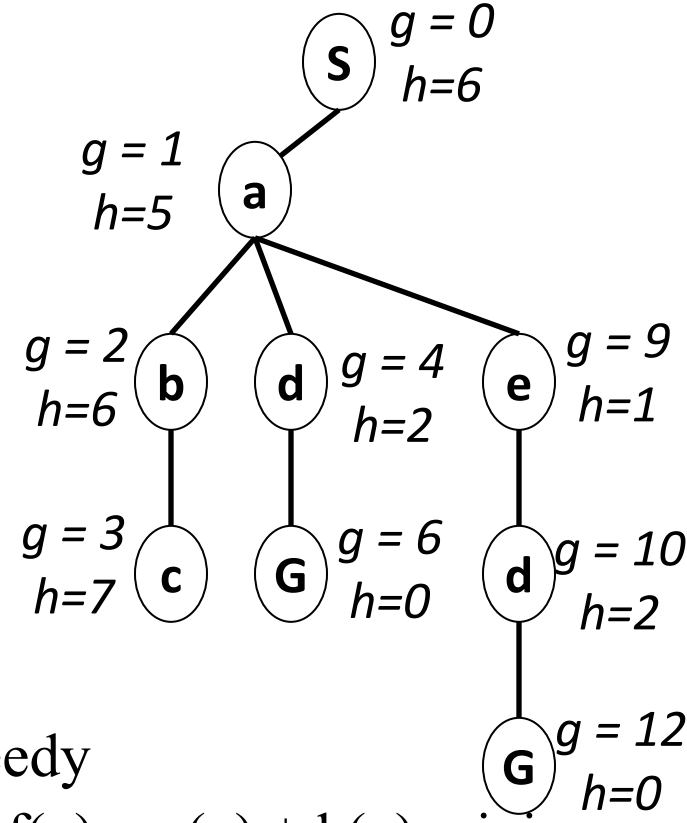
- **Căutarea uniformă bazată pe cost** preferă să exploreze soluții parțiale (drumuri) de cost curent minim (se bazează pe un calcul real, nu există estimări)

  g(n) = backward cost = cât costă drumul de la nodul inițial la nodul curent

- **Căutarea Greedy** preferă să exploreze soluții parțiale (drumuri) pentru care nodul curent este cel mai apropiat de un nod scop (se bazează pe o estimare)

  h(n) = forward cost = cât estimez că va costa drumul de la nodul curent la un nod scop

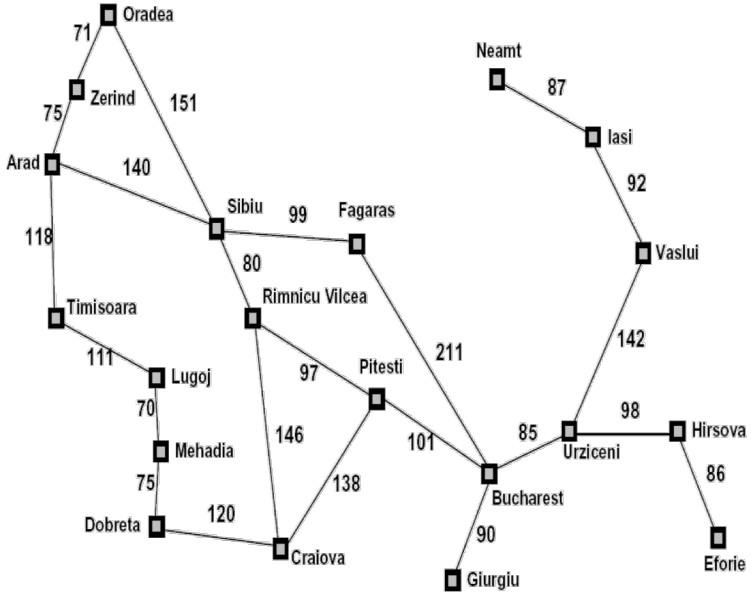




- Algoritmul de căutare A* combină UCS + Greedy
  - Preferă să exploreze soluțiile parțiale de cost f(n) = g(n) + h(n) minim

Exemplu de căutare A*

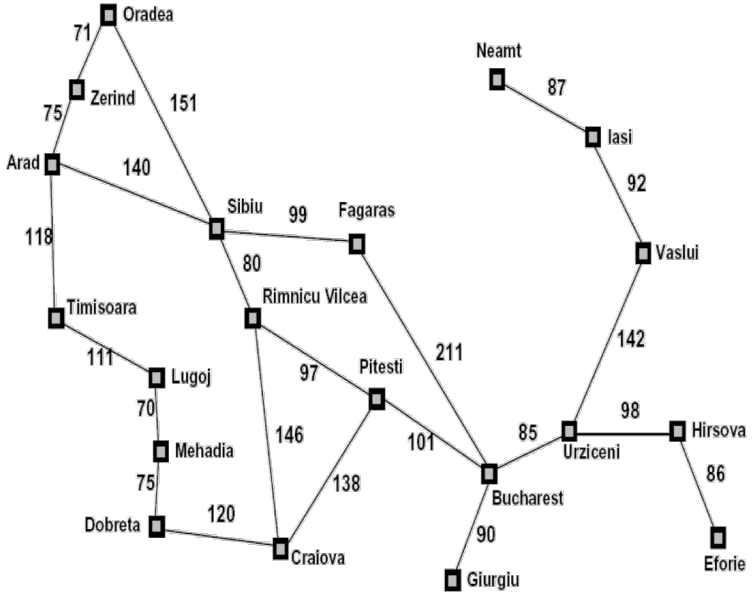
- Explorează nodul care pare cel mai promițător din perspectiva
  f(n) = g(n) + h(n)
  trecut (exact) **viitor (estimat)**

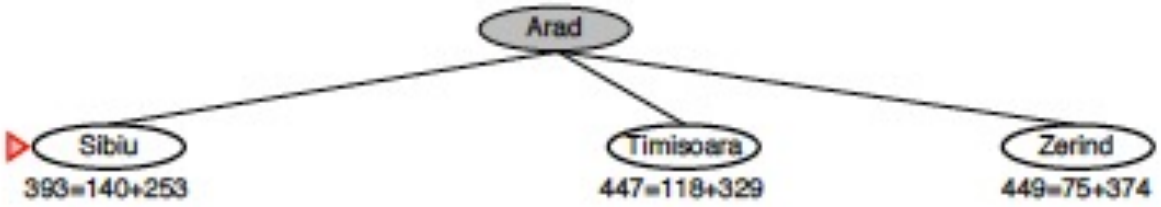


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Exemplu de căutare A*

- Explorează nodul care pare cel mai promițător din perspectiva f(n) = g(n) + h(n)

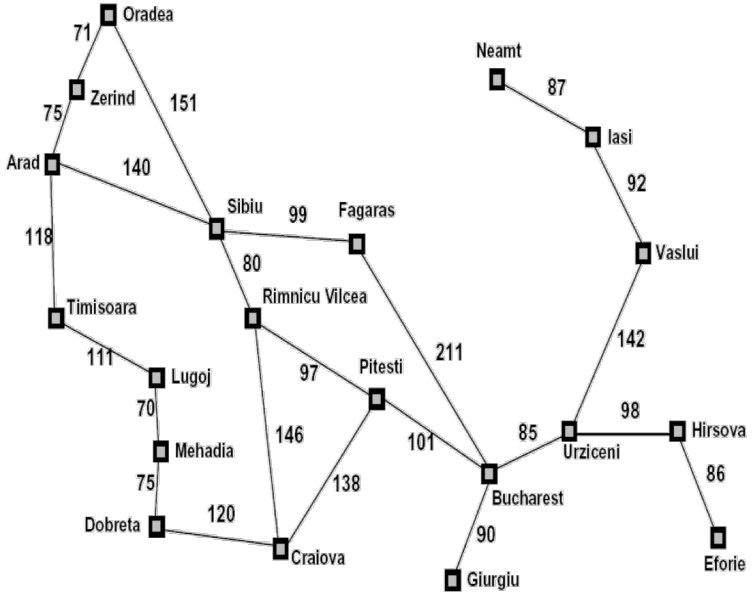


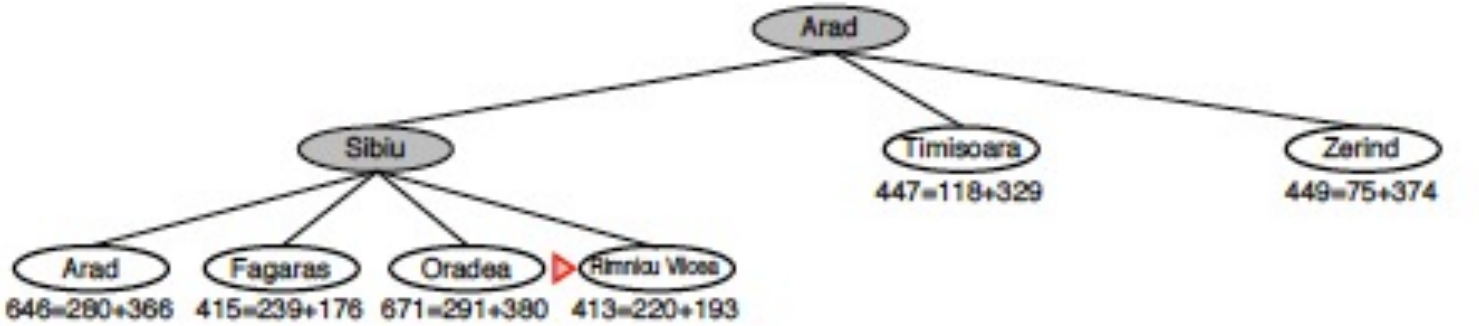


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Exemplu de căutare A*

- Explorează nodul care pare cel mai promițător din perspectiva f(n) = g(n) + h(n)

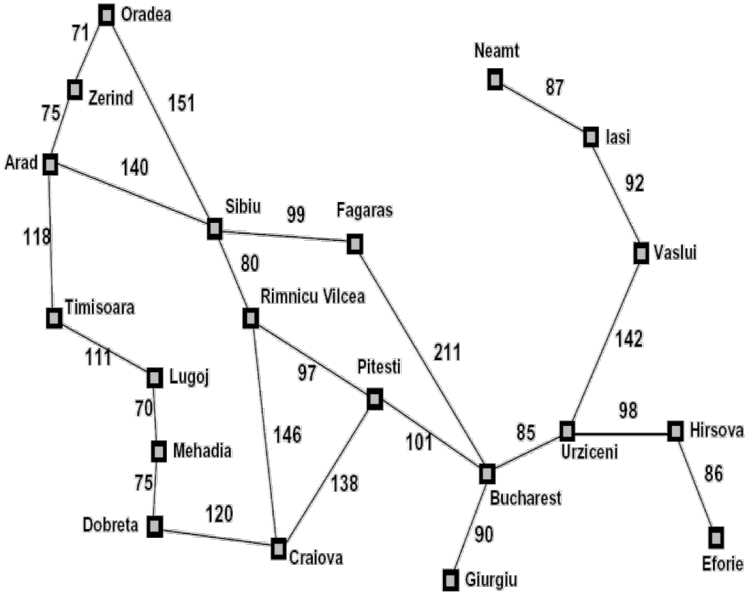


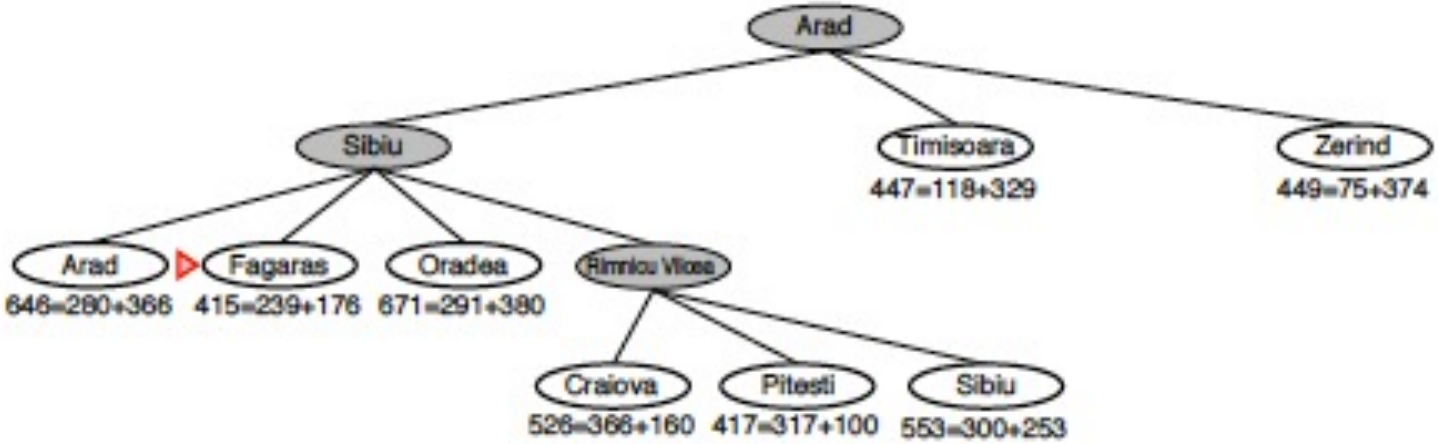


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Exemplu de căutare A*

- Explorează nodul care pare cel mai promițător din perspectiva f(n) = g(n) + h(n)

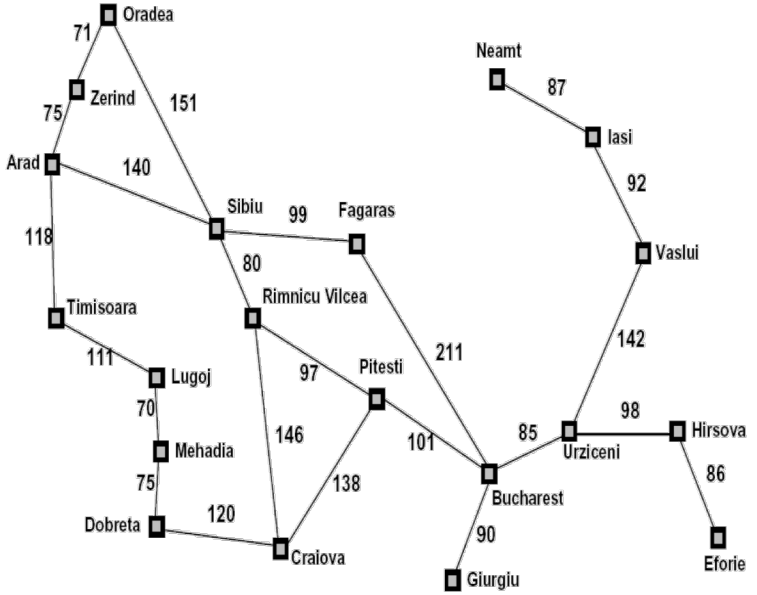


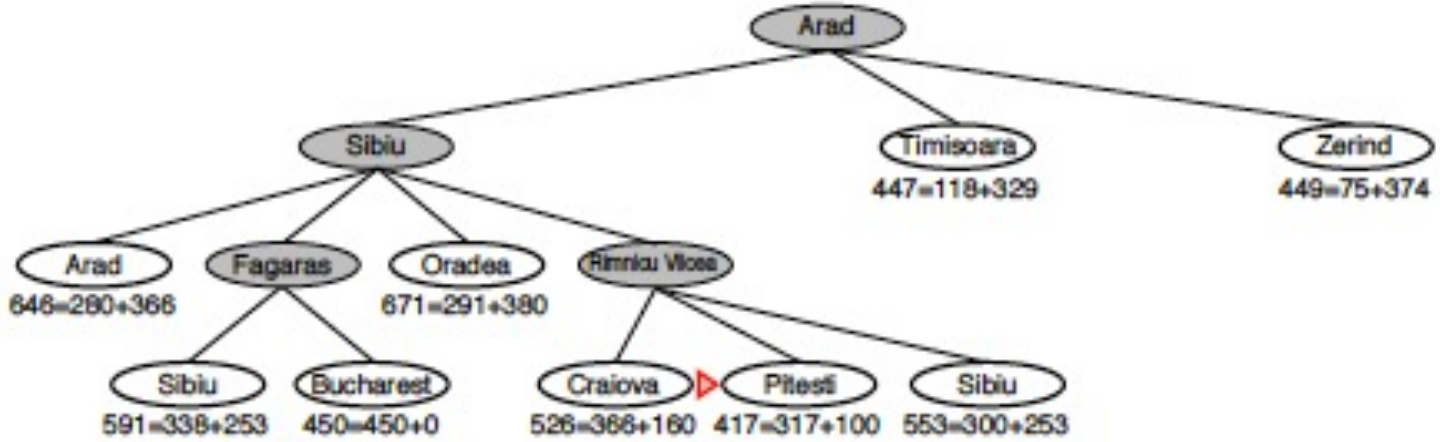


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Exemplu de căutare A*

- Explorează nodul care pare cel mai promițător din perspectiva f(n) = g(n) + h(n)

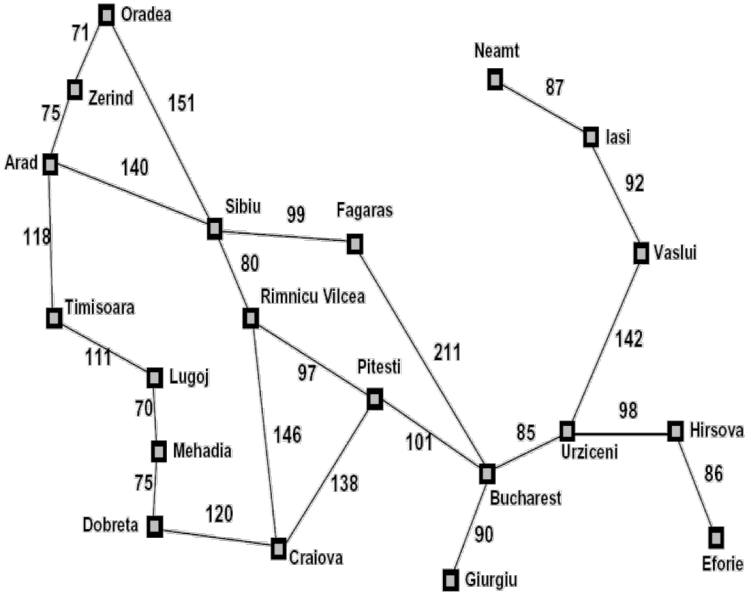


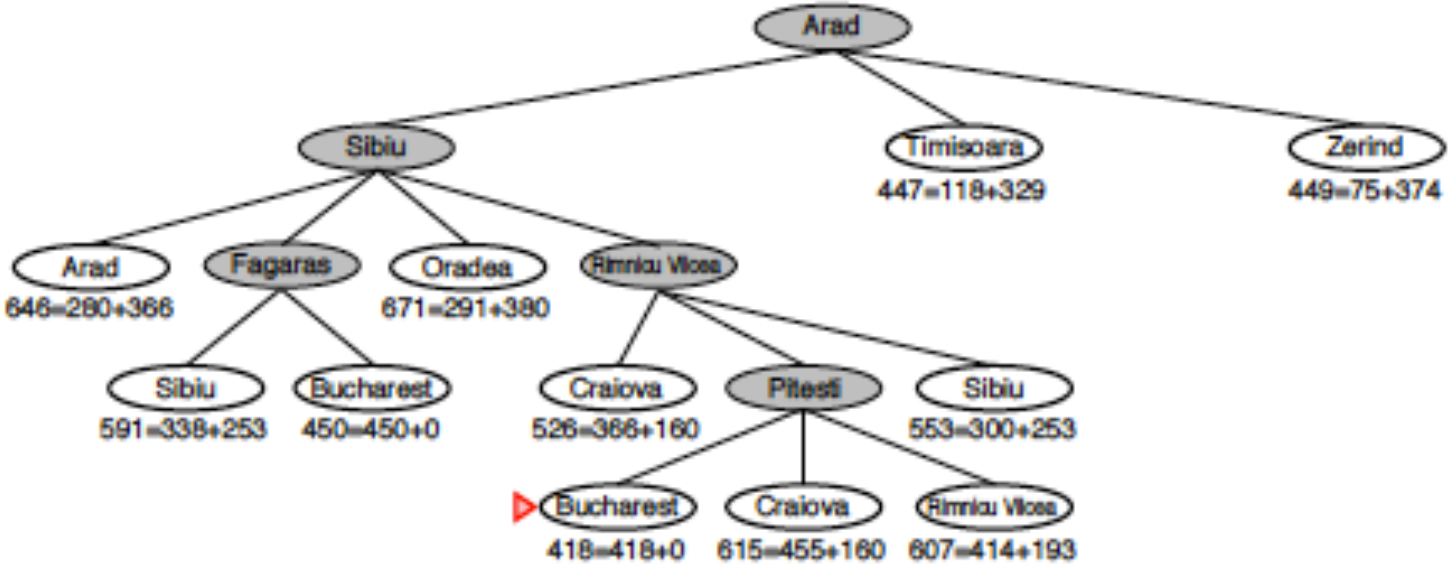


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Exemplu de căutare A*

- Explorează nodul care pare cel mai promițător din perspectiva f(n) = g(n) + h(n)

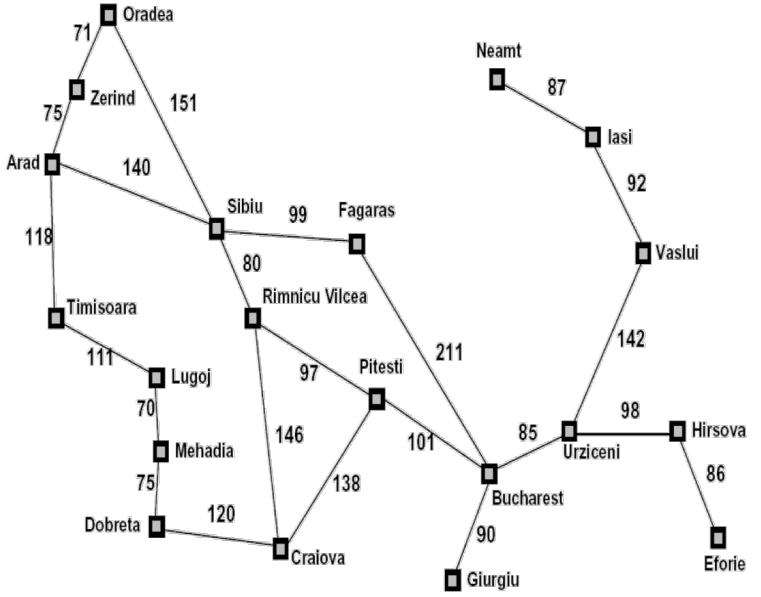


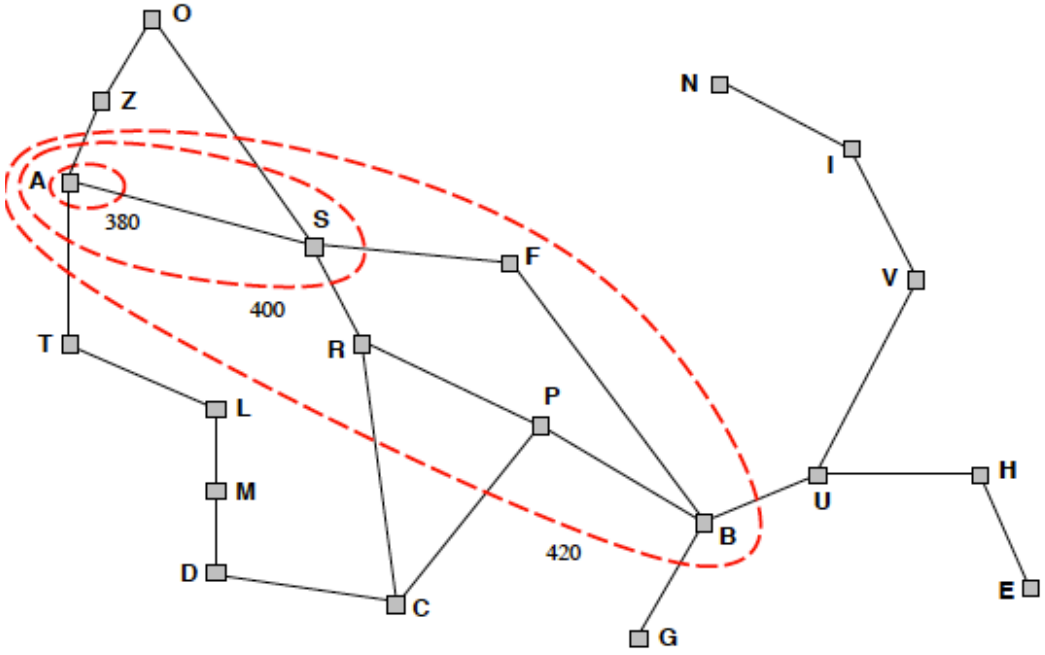


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Exemplu de căutare A*

- Explorează nodul care pare cel mai promițător din perspectiva f(n) = g(n) + h(n)
- Rezultat teoretic: A* explorează pe rând noduri în ordinea crescătoare a funcției f asociate

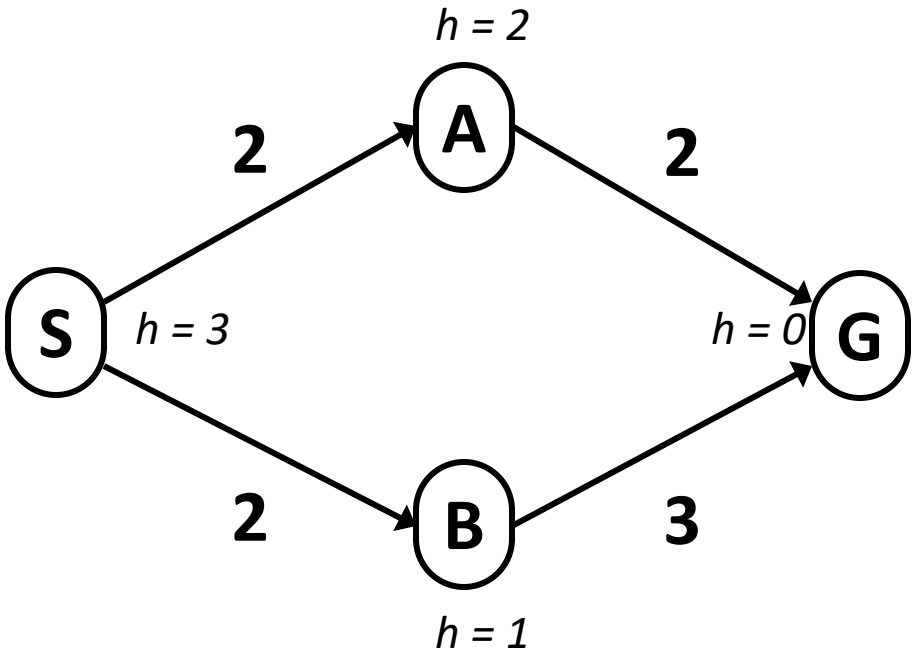




Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

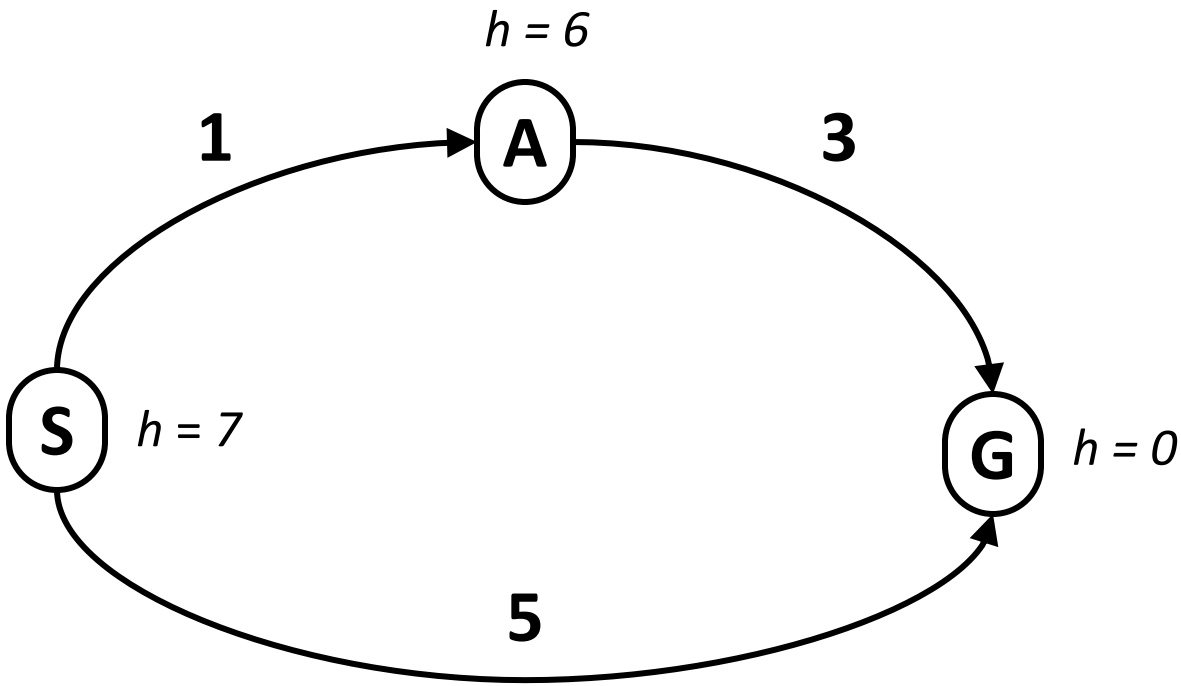
Când trebuie să ne oprim cu algoritmul A*?

- Ne oprim când ajungem la o stare –scop?



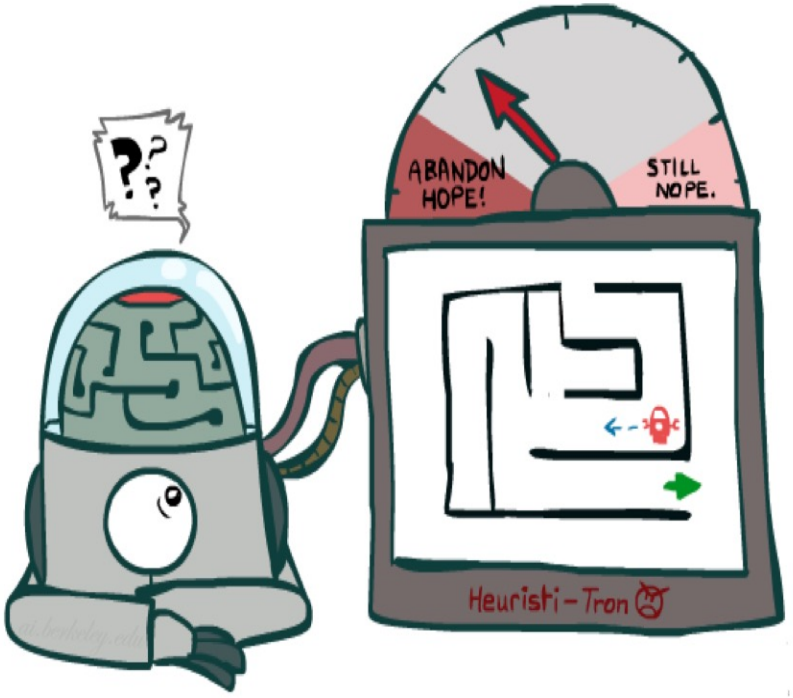
- NU: ne oprim numai după ce explorăm toate nodurile care au șansa să conducă la soluție optimă

Este algoritmul A* optimal?

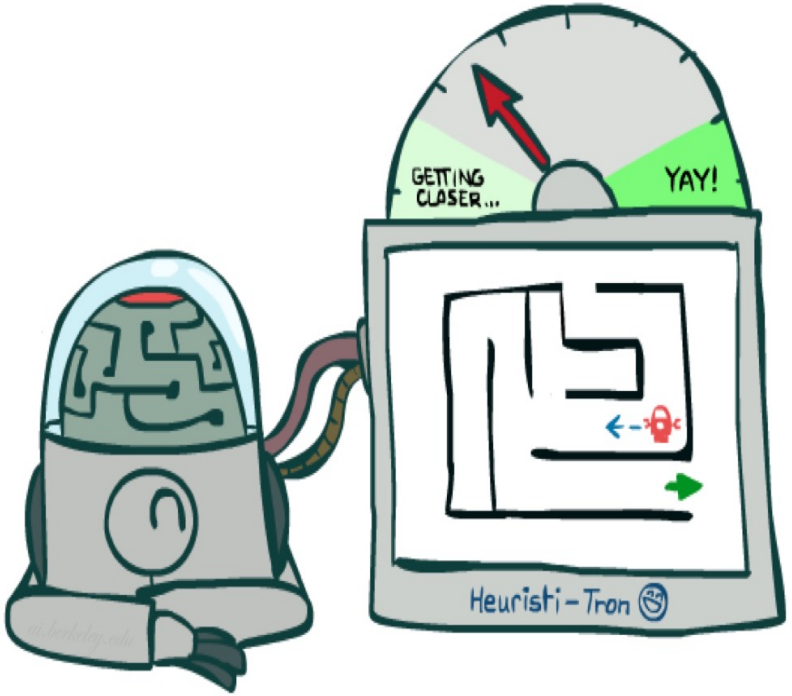


- Avem nevoie de estimări (date de euristici) care să fie mai mici decât costurile reale (euristici optimiste)
- Optimalitate condiționată de euristici cu o asemenea caracteristică: A* este optim dacă avem euristici optimiste

Admisibilitatea euristicilor



Euristici inadmisibile (pesimiste): supraestimează costul unui drum, nu conduc la soluții optime, stopează explorarea unor drumuri de cost minim



Euristici admisibile (optimiste): subestimează costul unui drum, încetinesc explorarea unor drumuri de cost mare, nu supraestimează costurile drumurilor

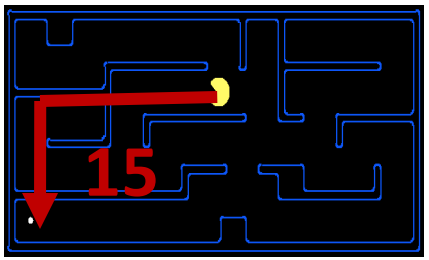
Euristici admisibile

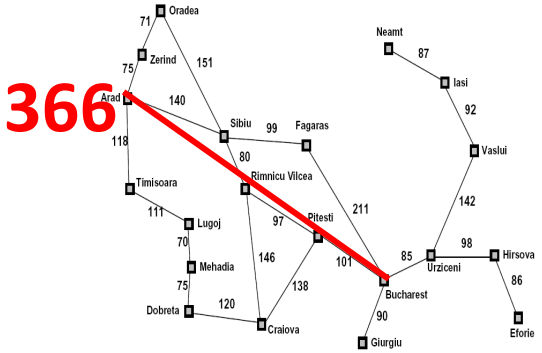
- O euristică *h* este *admisibilă* (optimistă) dacă:

$$0 \leq h(n) \leq h^*(n)$$

unde $h^*(n)$ este costul real către cel mai apropiat nod scop

- Exemple:

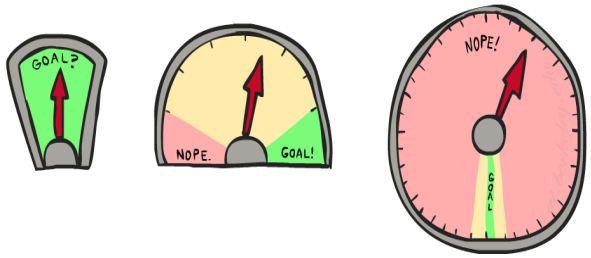




- Găsirea de euristici admisibile este partea cea mai grea în aplicarea algoritmului A*.

Euristici pentru A*

- de ce să nu folosim ca euristică *costul real* până la un nod scop?
  - este o asemenea euristică admisibilă?
  - explorăm mai multe sau mai puține noduri?
  - există vreun dezavantaj?



- compromis între calitatea estimării dată de euristică și numărul de noduri explorate:
  - pe măsură ce o euristică se apropie mai bine de costul real, A* va explora mai puține noduri, însă ia mai mult timp să calculeze valoarea euristicii la nodul curent

Euristici dominante și banale

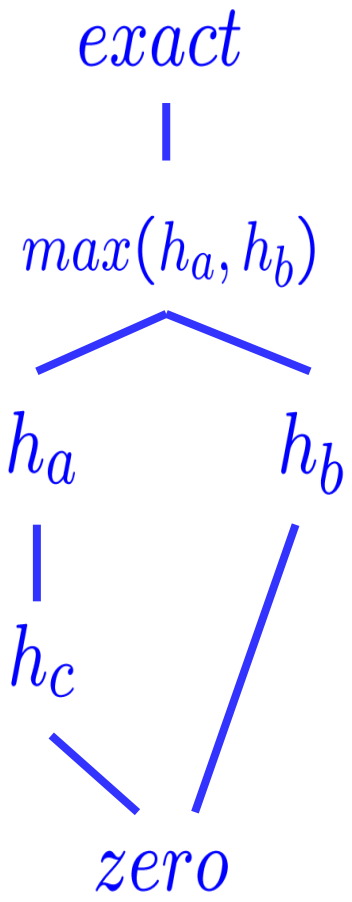
- dominanță: $h_a \geq h_c$ dacă

  $\forall n : h_a(n) \geq h_c(n)$

- euristicile formează o semi-latice:
  - maximum a două euristici admisibile este o euristică admisibilă

  $h(n) = max(h_a(n), h_b(n))$

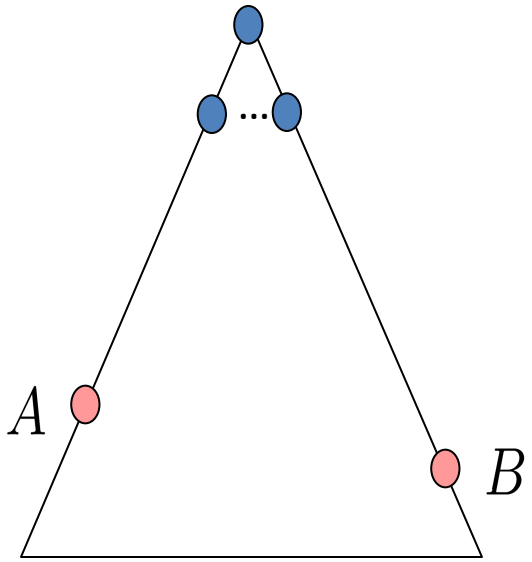
- euristici banale:
  - euristica zero (la ce conduce în A*?)
  - euristica exactă



Optimalitatea algoritmului A*

Presupunem:

- A este un nod-scop optim global (nu există alt nod-scop cu cost mai mic)
- B este un nod-scop suboptimal (are cost mai mare decât soluția optimă)
- h este euristică admisibilă



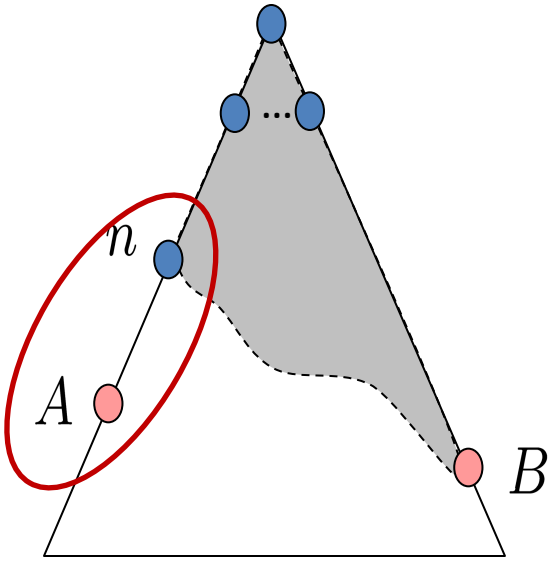
Afirmație:

- A va fi explorat de algoritmul A* înaintea lui B

Optimalitatea algoritmului A*

Demonstrație:

- presupunem că B este pe frontieră
- un nod ascendent *n* al lui A (strămoș) este de asemenea pe frontieră (poate chiar nodul A!)
- afirmație: *n* va fi explorat înaintea nodului B
  1. f(n) ≤ f(A)



$f(n) = g(n) + h(n)$ **definiția costului f**

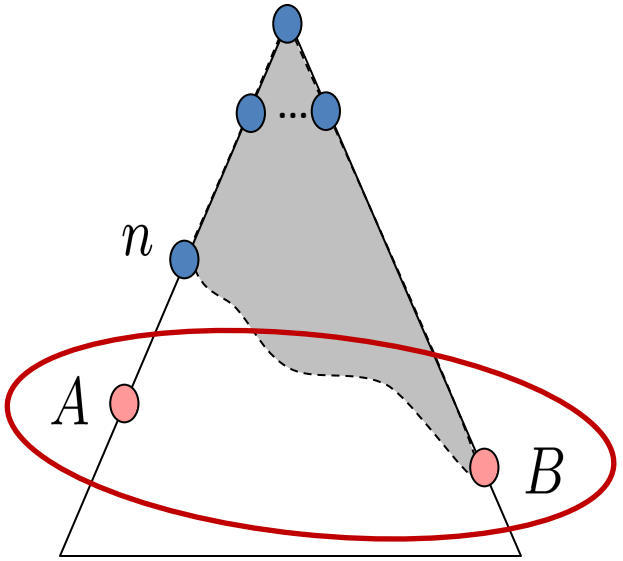
$f(n) \leq g(A)$ **h subestimează costul până la A**

$g(A) = f(A)$ **h(A) = 0, A e nod-scop**

Optimalitatea algoritmului A*

Demonstrație:

- presupunem că B este pe frontieră
- un nod ascendent *n* al lui A (strămoș) este de asemenea pe frontieră (poate chiar nodul A!)
- afirmație: *n* va fi explorat înaintea nodului B
  1. f(n) ≤ f(A)
  2. f(A) < f(B)



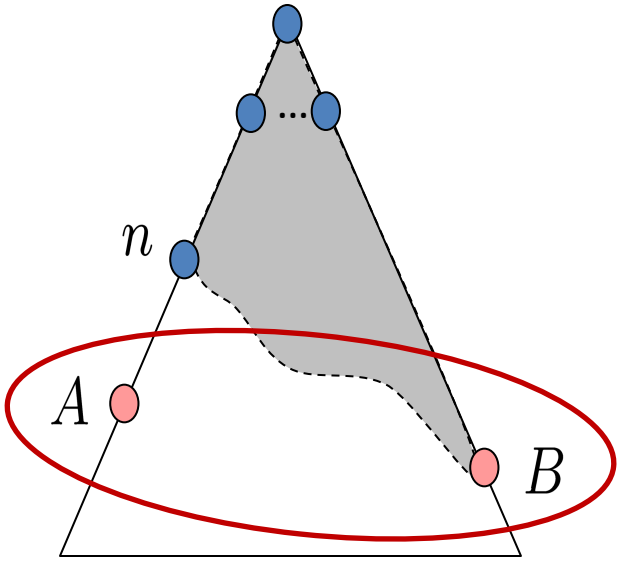
$g(A) < g(B)$ **B este nod suboptimal**

$f(A) < f(B)$ **h(A) = 0, h(B) = 0** admisibilitatea lui h

Optimalitatea algoritmului A*

Demonstrație:

- presupunem că B este pe frontieră
- un nod ascendent *n* al lui A (strămoș) este de asemenea pe frontieră (poate chiar nodul A!)
- afirmație: *n* va fi explorat înaintea nodului B
  1. f(n) ≤ f(A)
  2. f(A) < f(B)
  3. n este explorat înaintea lui B

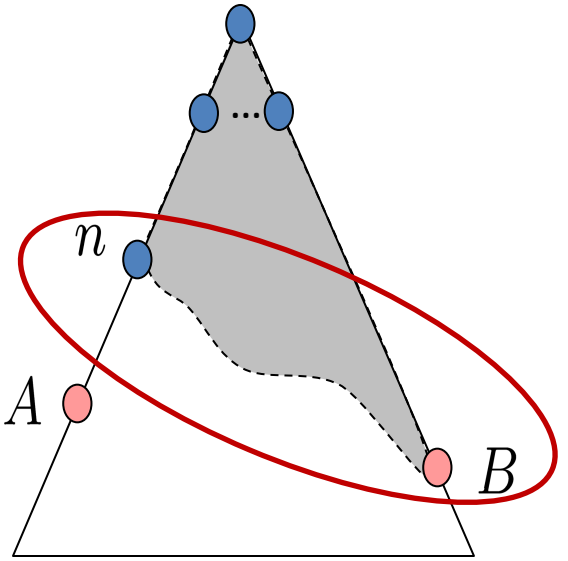


$$f(n) \leq f(A) < f(B)$$

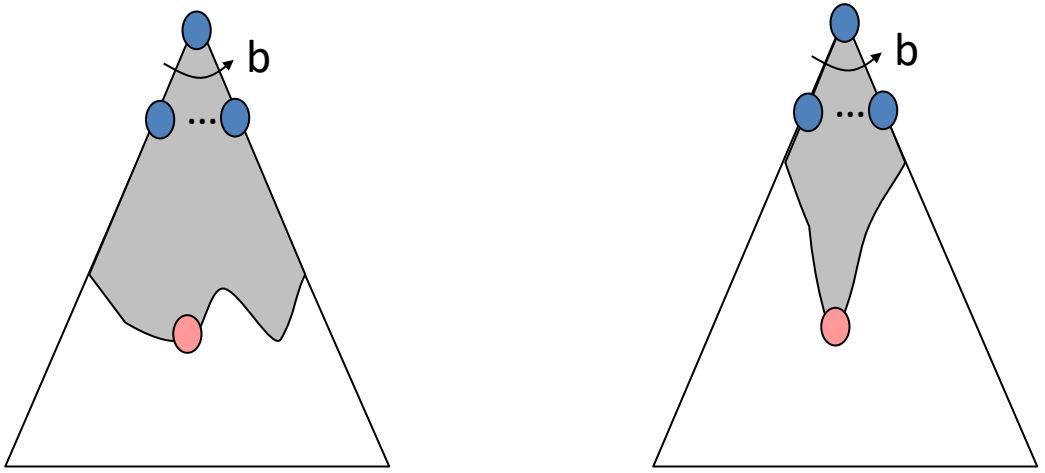
Optimalitatea algoritmului A*

Demonstrație:

- presupunem că B este pe frontieră
- un nod ascendent *n* al lui A (strămoș) este de asemenea pe frontieră (poate chiar nodul A!)
- afirmație: *n* va fi explorat înaintea nodului B
  1. f(n) ≤ f(A)
  2. f(A) < f(B)
  3. *n* este explorat înaintea lui B
- toți ascendenții lui A sunt explorați înaintea lui B
- A este explorat înaintea lui B
- algoritmul A* este optimal



Proprietățile algoritmului A*

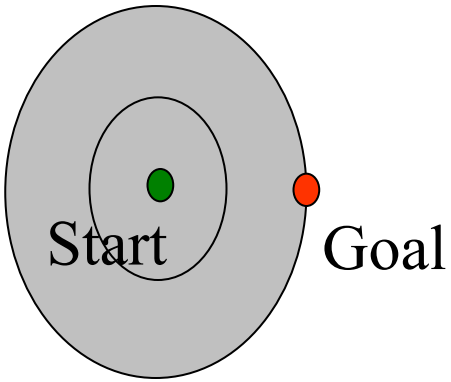


UCS A*

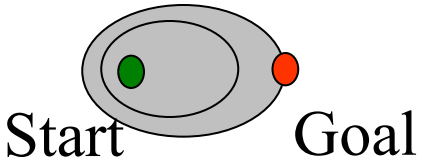
Căutare uniformă pe bază de cost explorează drumuri de cost minim, indiferent de cât de îndepărtate sunt de soluția-scop. Algoritmul A* beneficiază de euristica h care restricționează căutarea.

UCS vs A* - explorarea arborelui de căutare

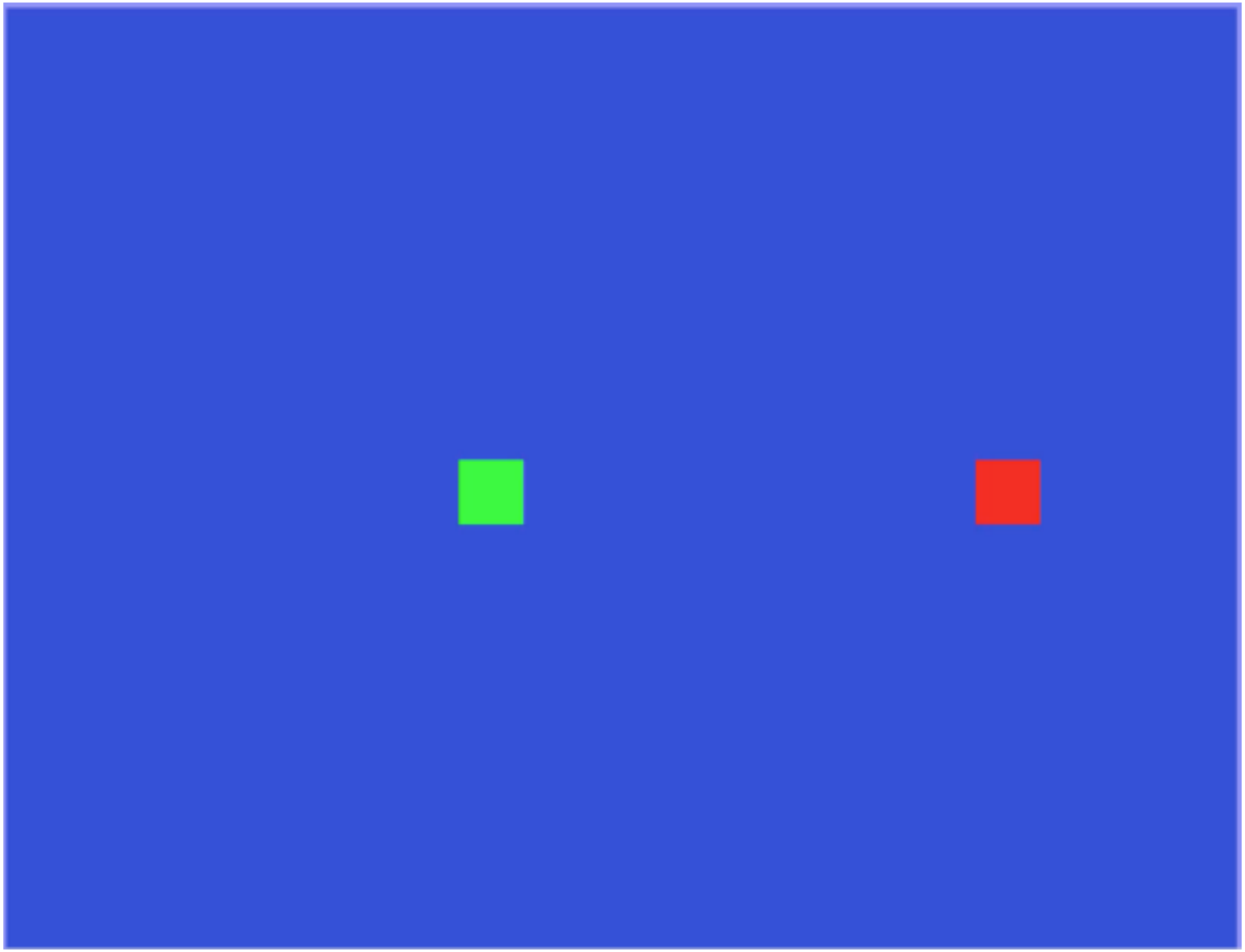
- Căutarea uniformă pe bază de cost (UCS) explorează soluții uniform în toate direcțiile.



- Algoritmul A* explorează soluții în mare în direcția soluției, totuși păstrează și soluții în alte direcții (pentru păstrarea optimalității).

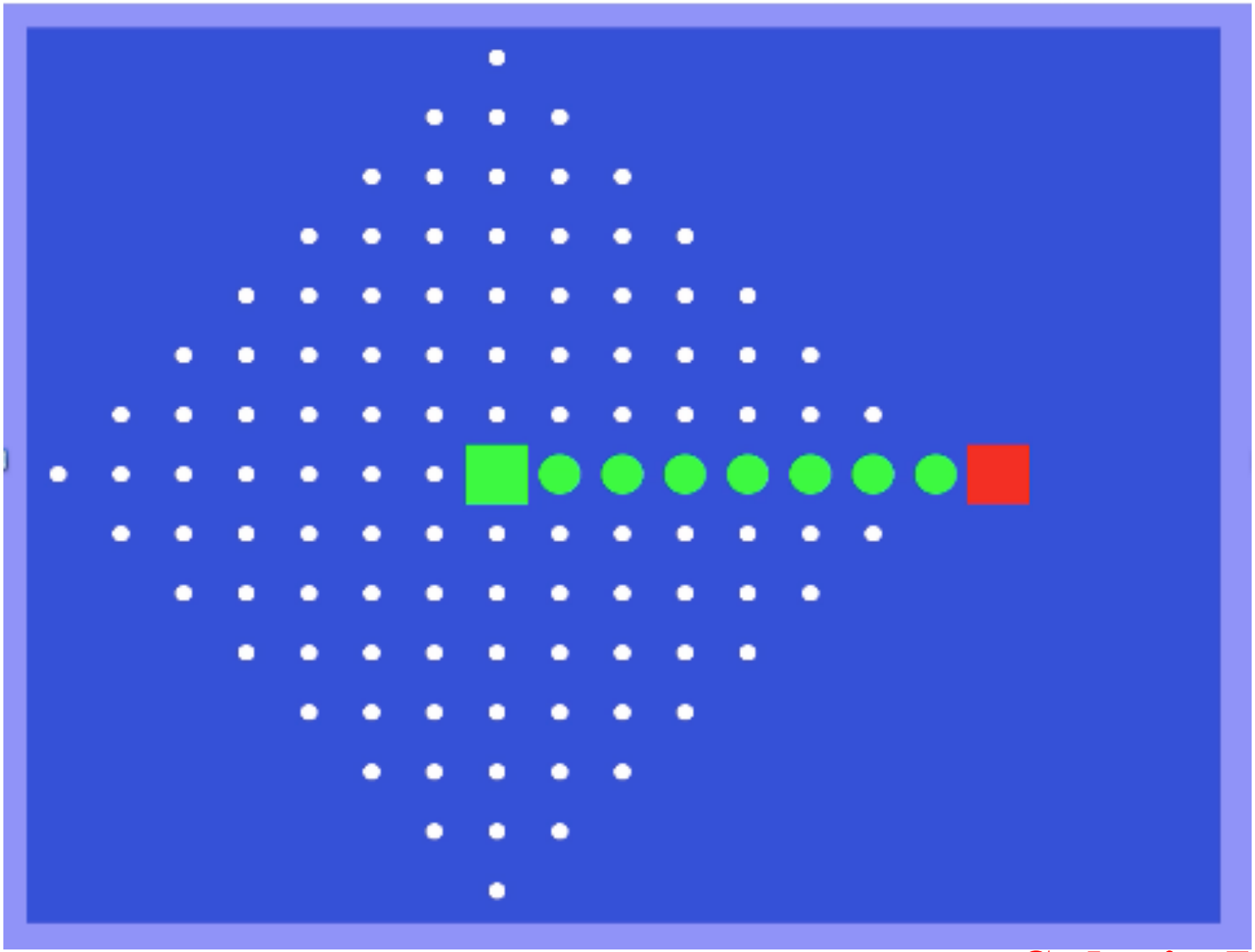


Explorarea arborelui de căutare - demo



Explorarea arborelui de căutare – demo UCS

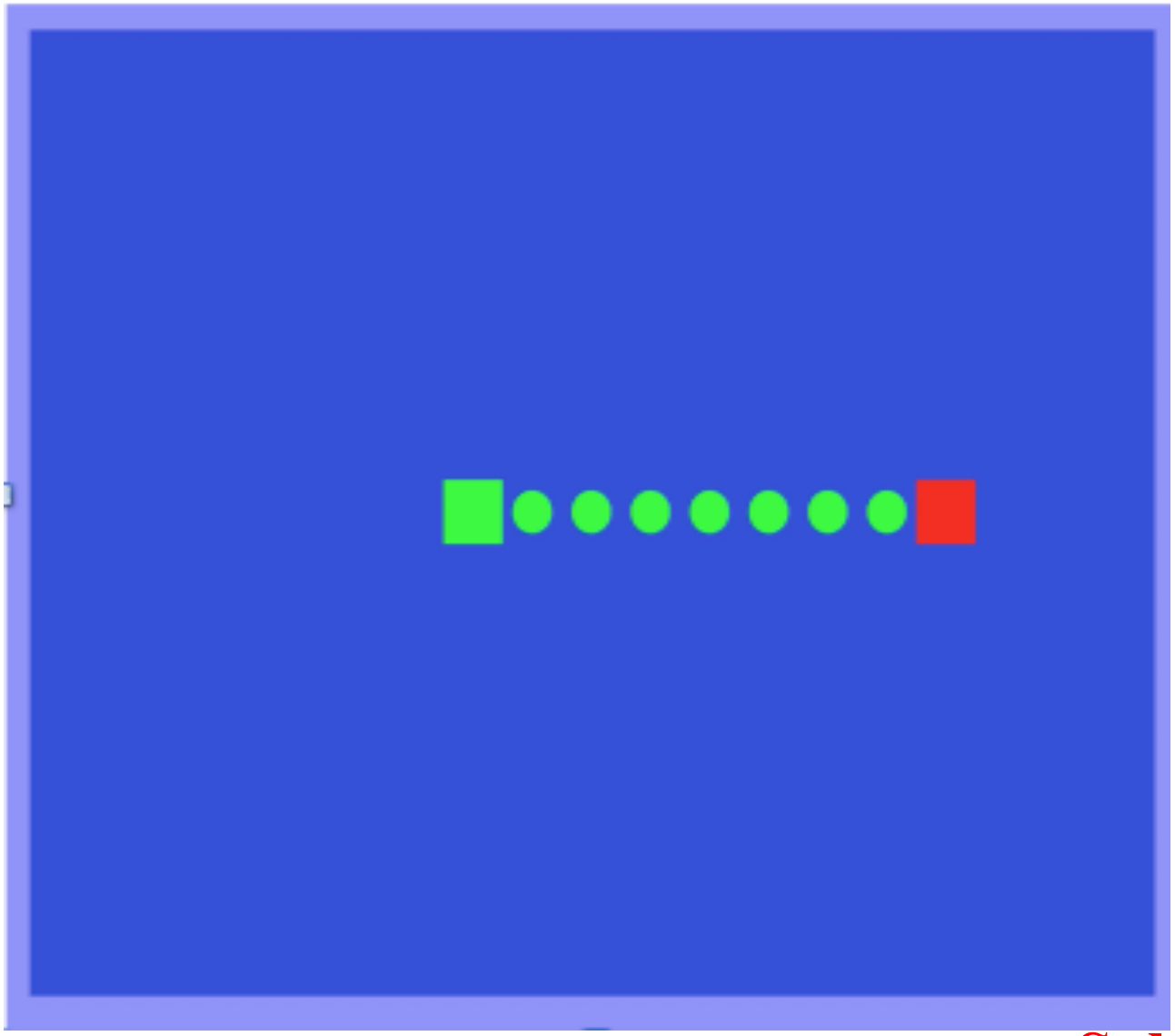
Explorarea arborelui de căutare – demo UCS



Soluția UCS

Explorarea arborelui de căutare – demo Greedy

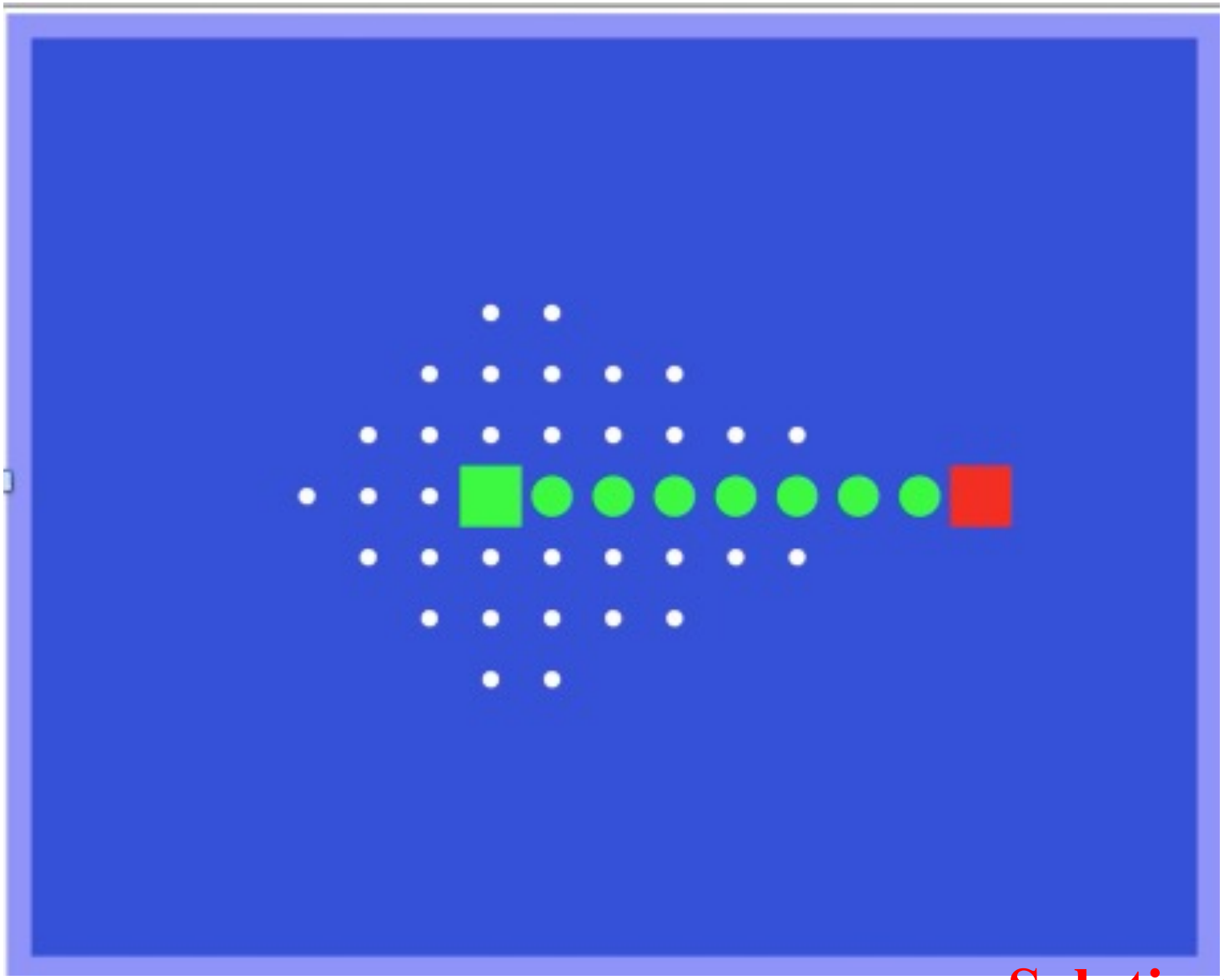
Explorarea arborelui de căutare – demo Greedy



Soluția Greedy

Explorarea arborelui de căutare – demo A*

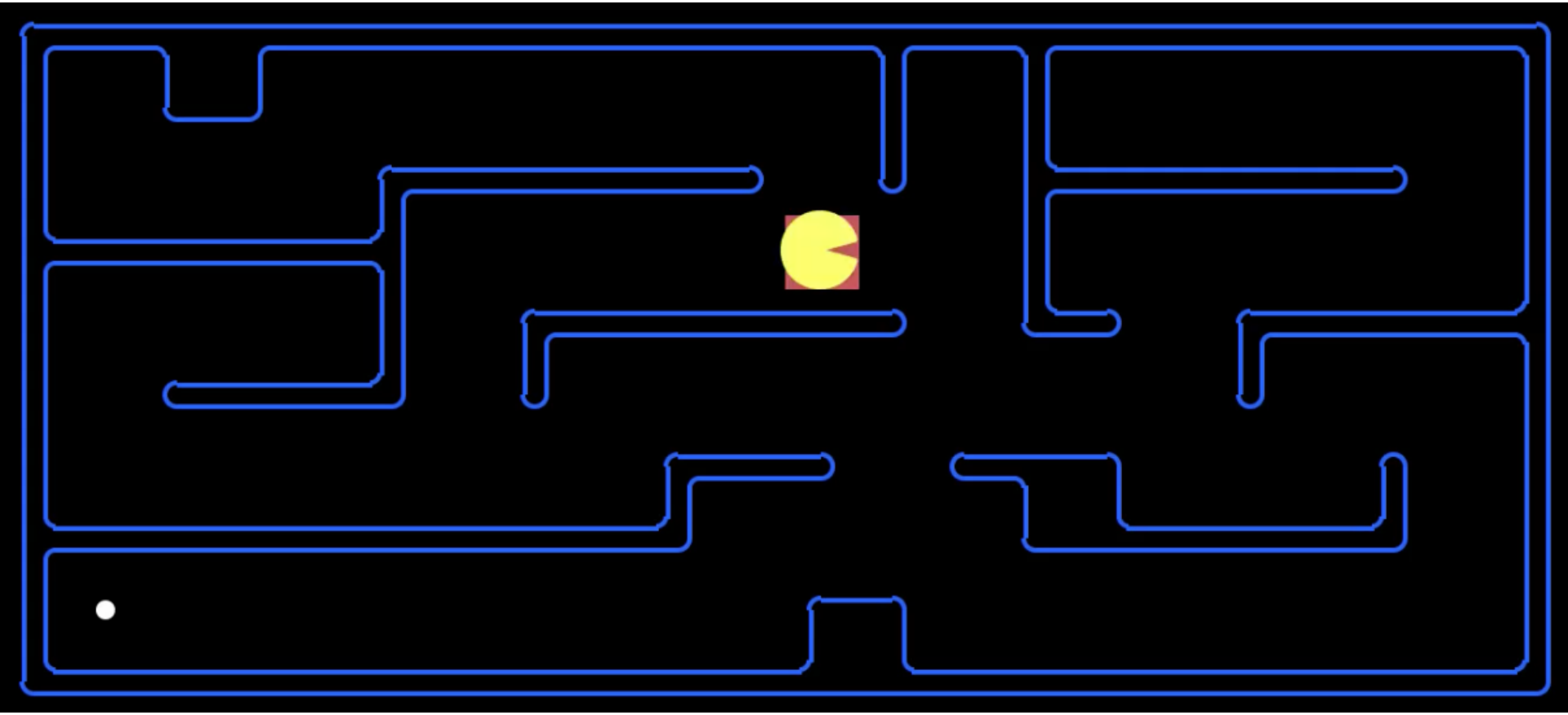
Explorarea arborelui de căutare – demo A*



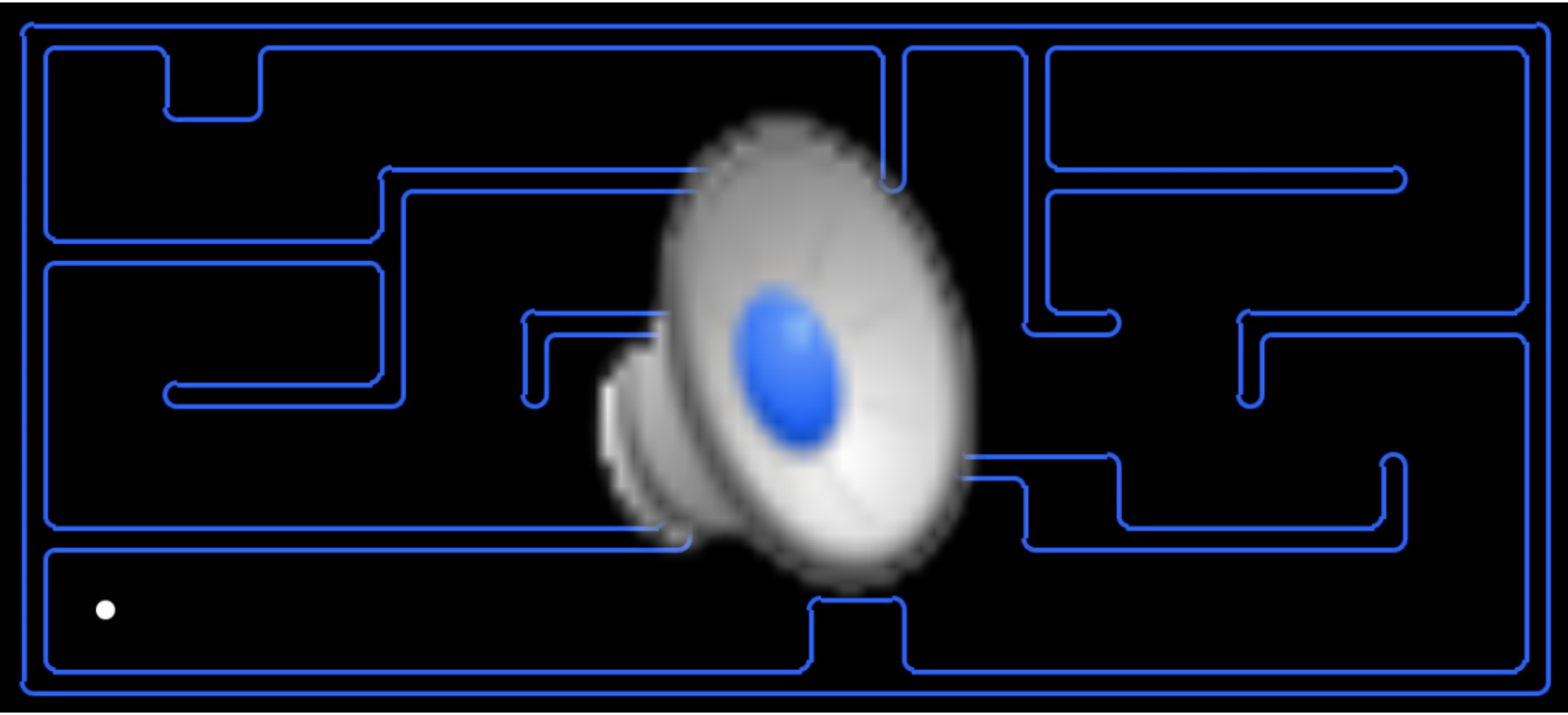
Soluția A*

Demo A* - explorarea arborelui de căutare

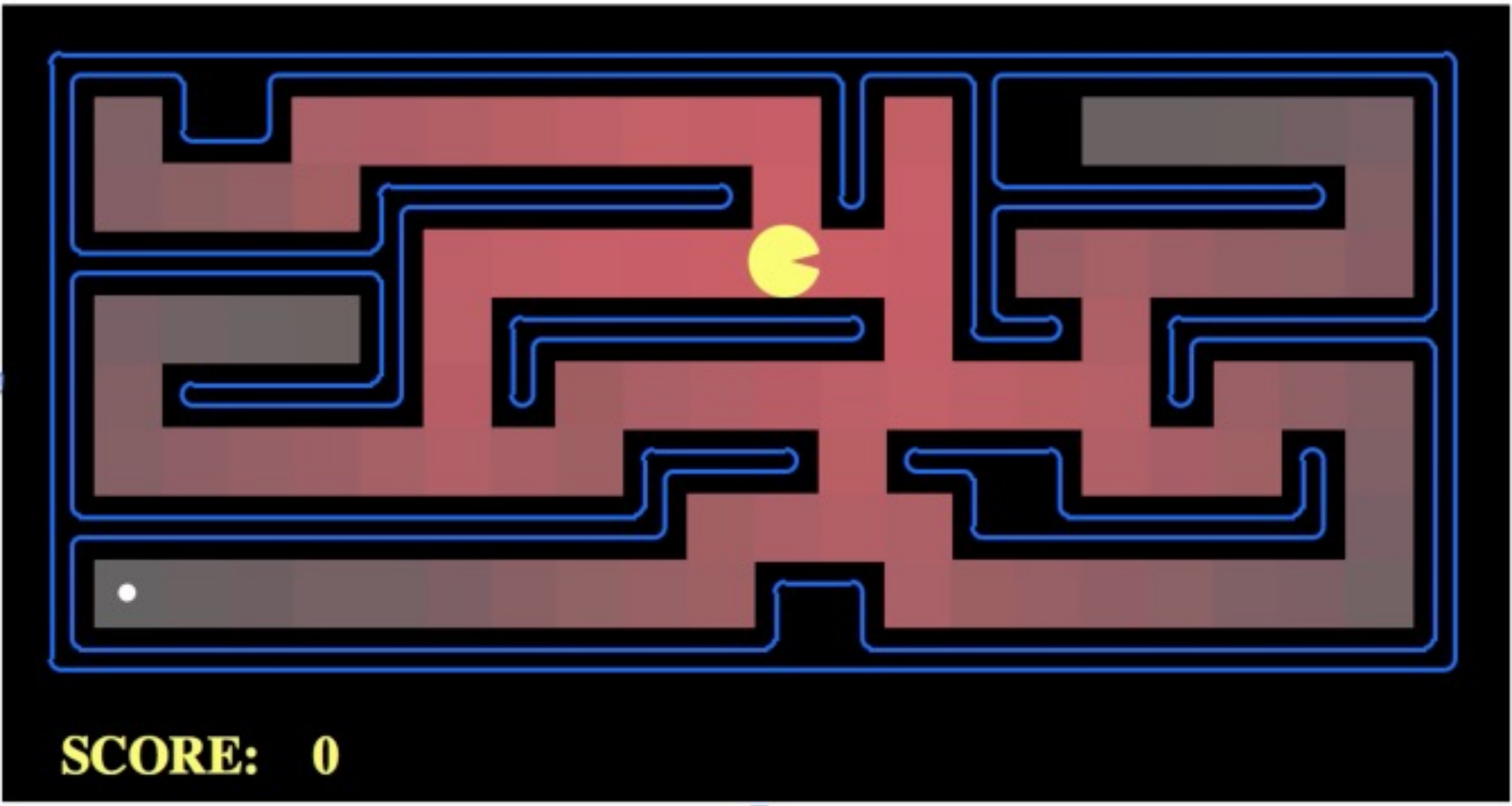
Explorarea arborelui de căutare - demo



Explorarea arborelui de căutare – demo UCS



Explorarea arborelui de căutare – demo UCS



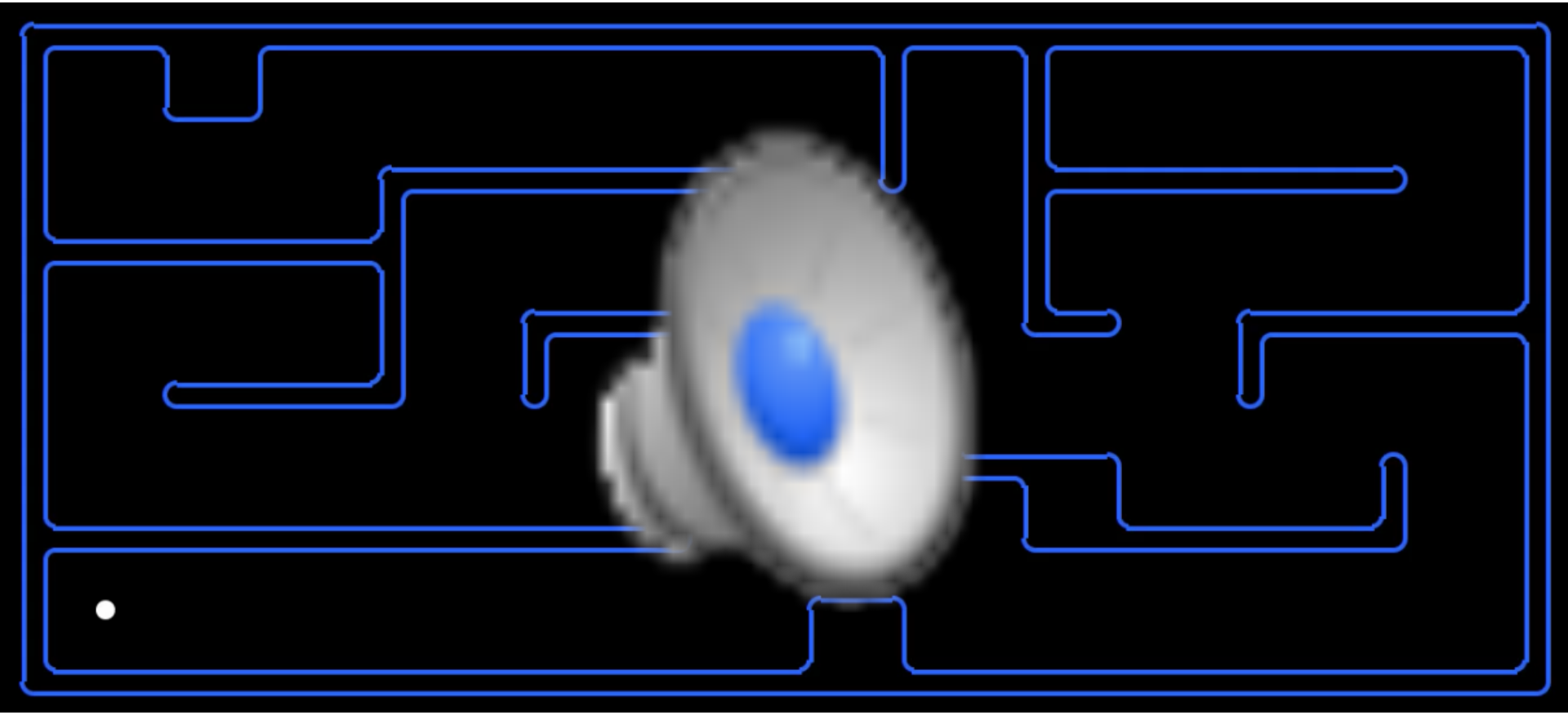
Ordinea de explorare a nodurile este dată de culoare:

Roșu intens – noduri explorate la început

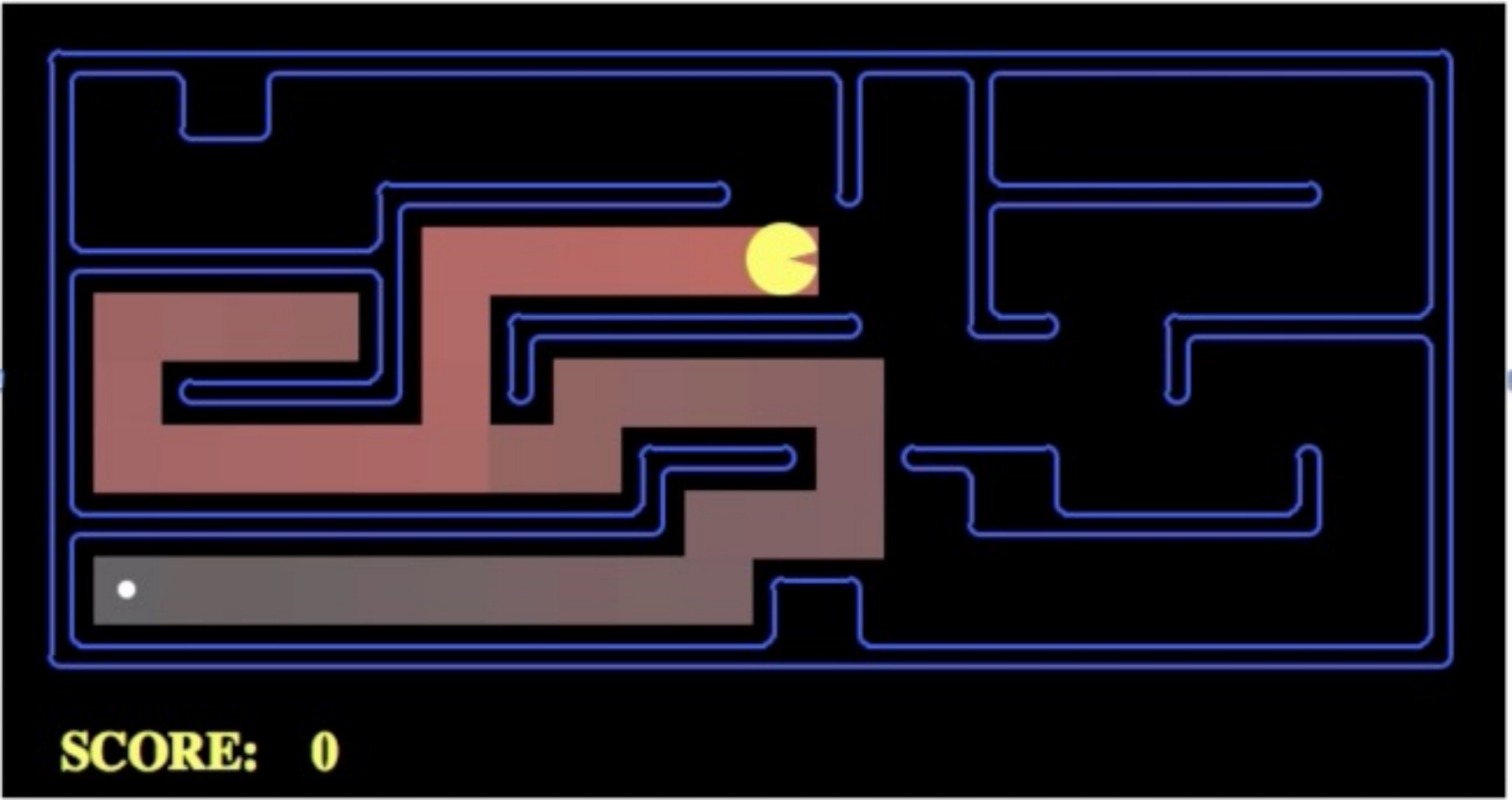
Gri – noduri explorate la sfârșit

Soluția UCS

Explorarea arborelui de căutare – demo Greedy

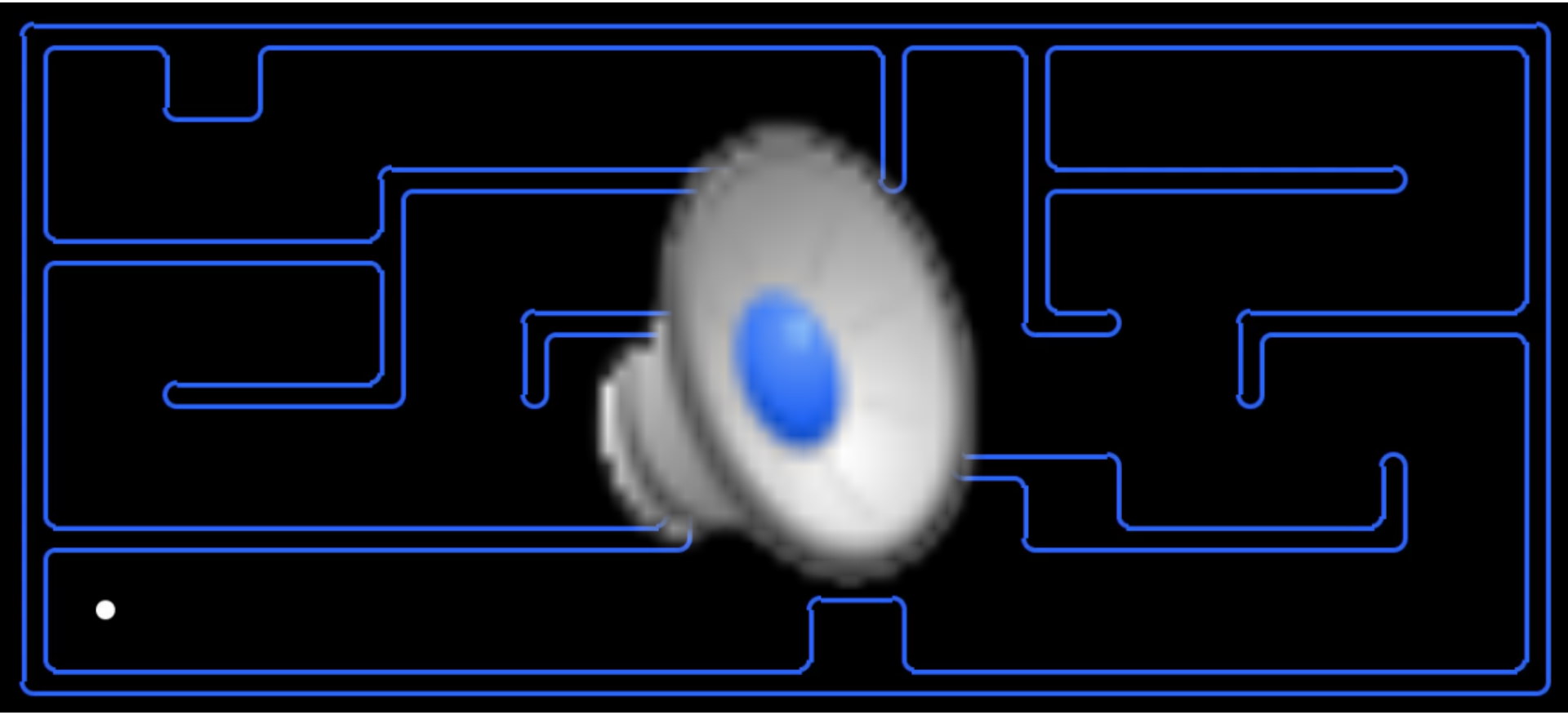


Explorarea arborelui de căutare – demo Greedy

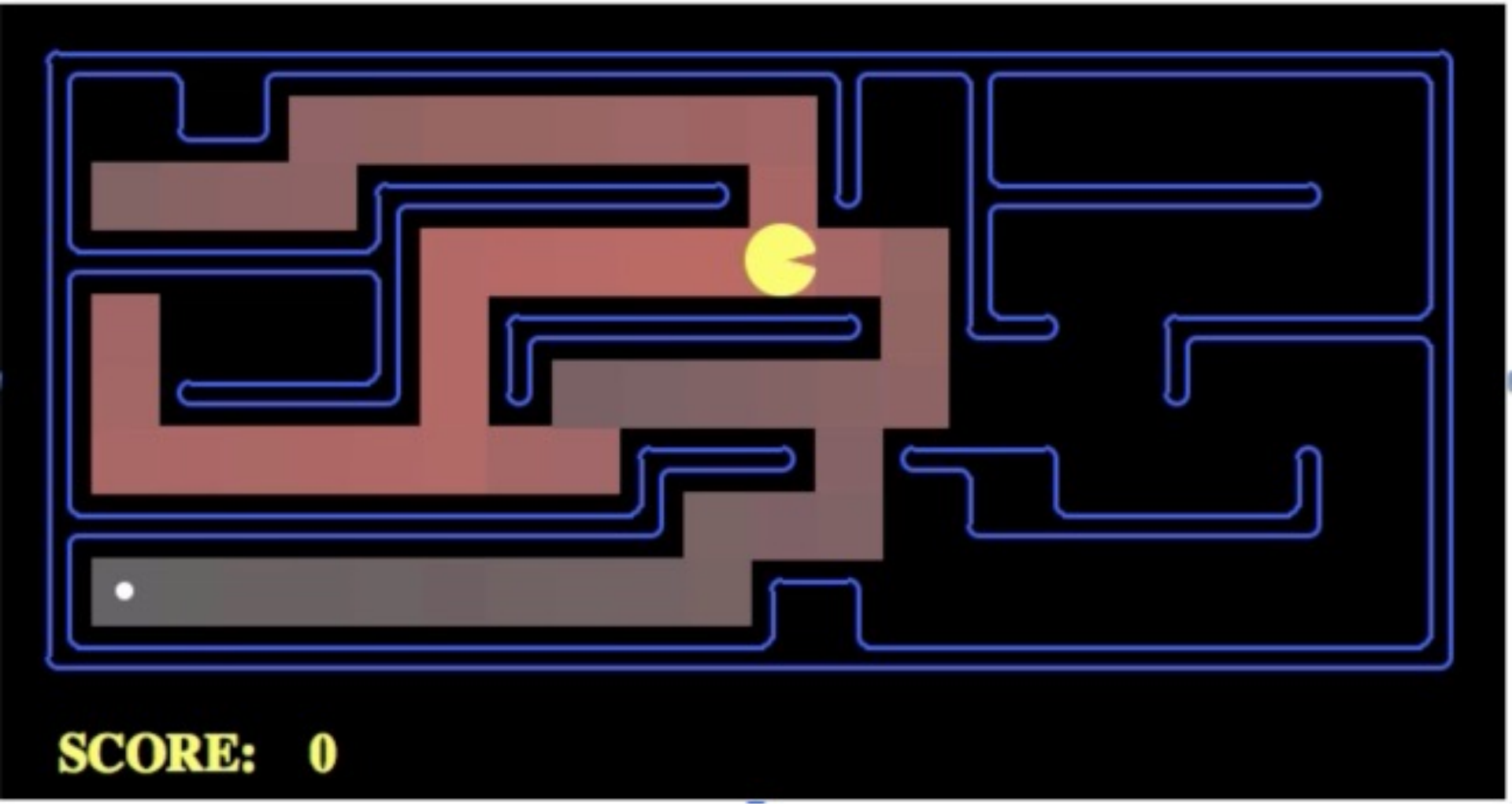


Soluția Greedy

Explorarea arborelui de căutare – demo A*



Explorarea arborelui de căutare – demo A*



Soluția A*

Comparare Pac-Man

Greedy

suboptimal

rapid

UCS

optimal

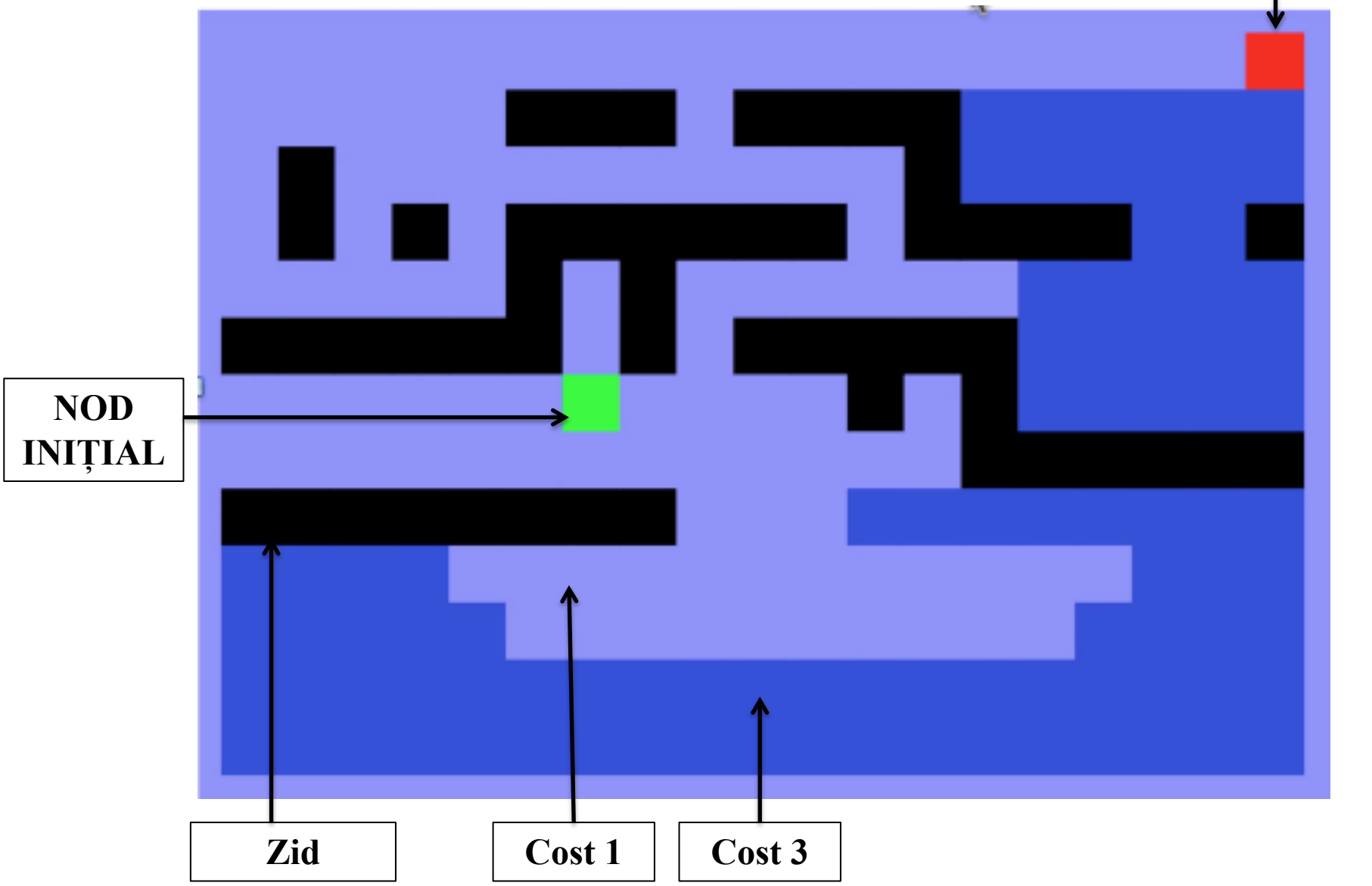
încet

A*

optimal

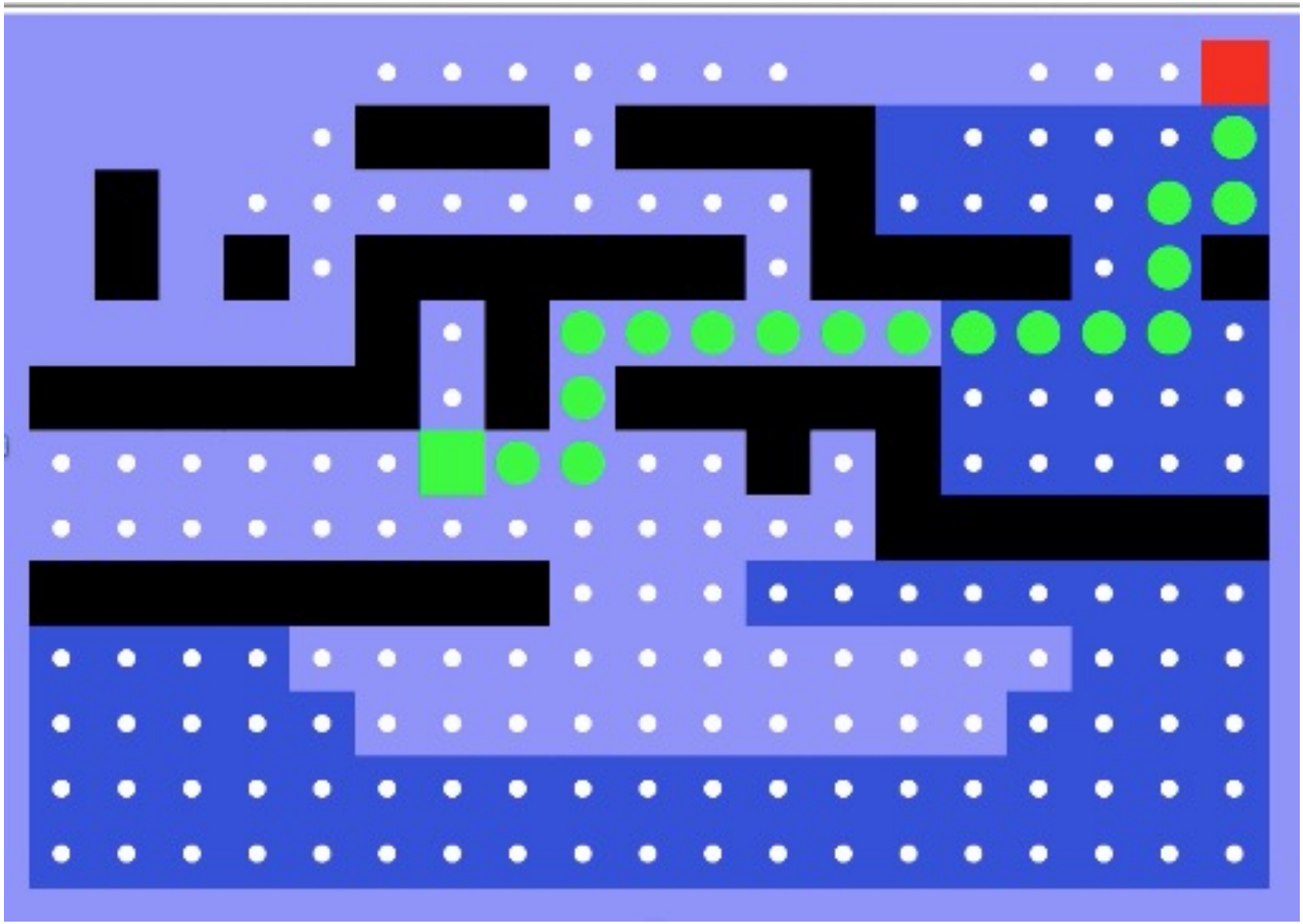
rapid

Labirint - demo



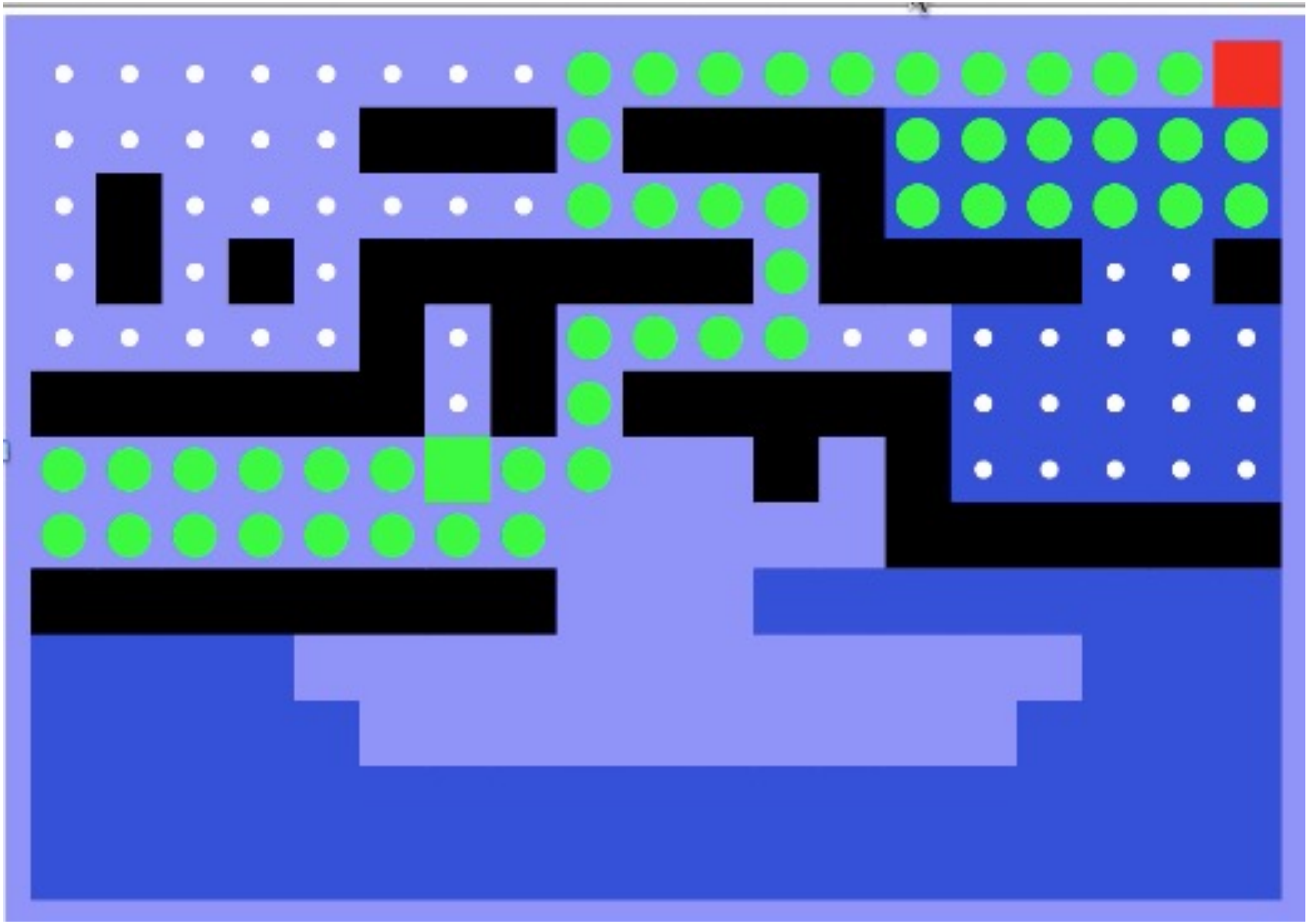
Labirint – demo Bread First search

Labirint – demo Bread First search



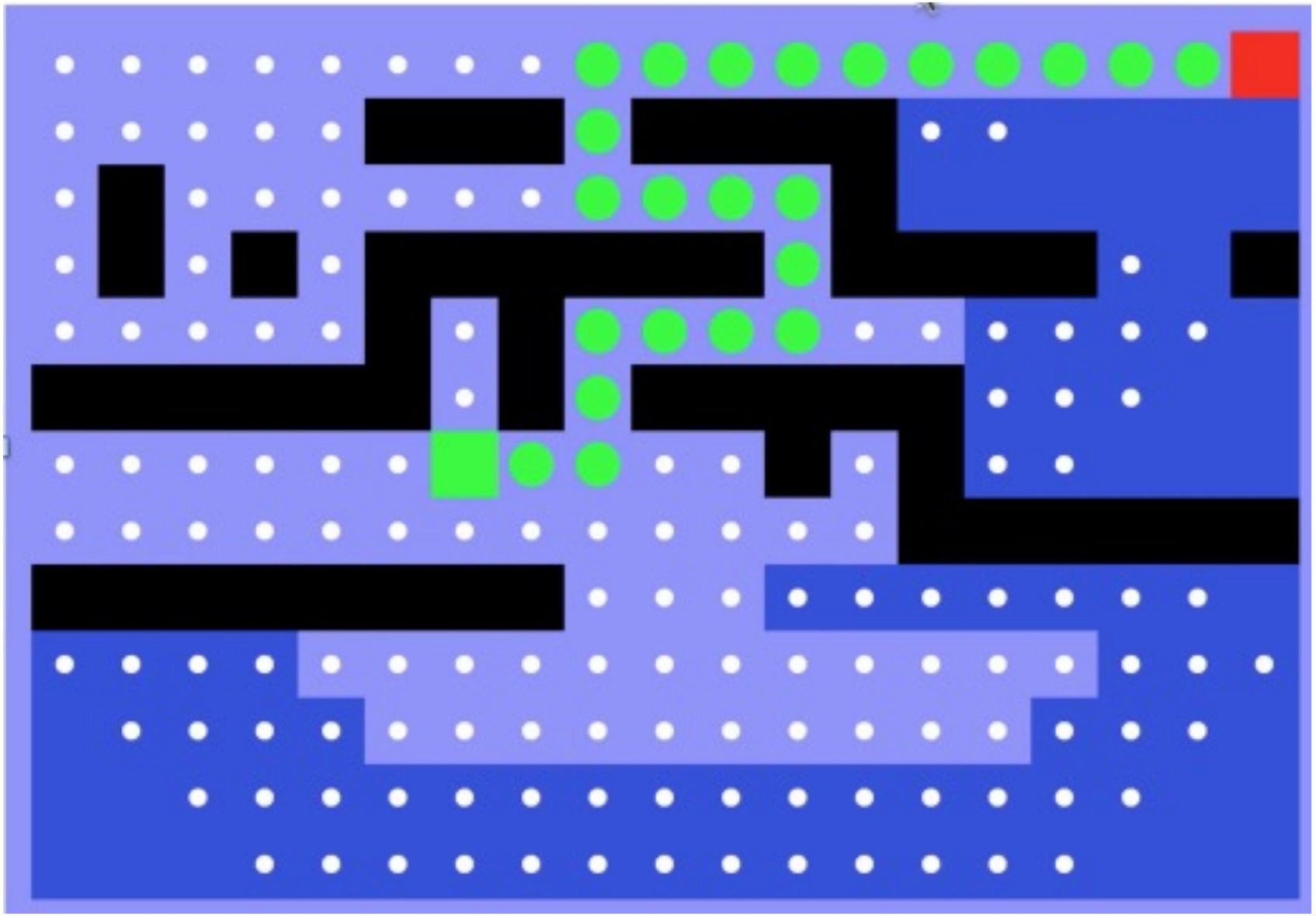
Labirint – demo Depth First search

Labirint – demo Depth First search



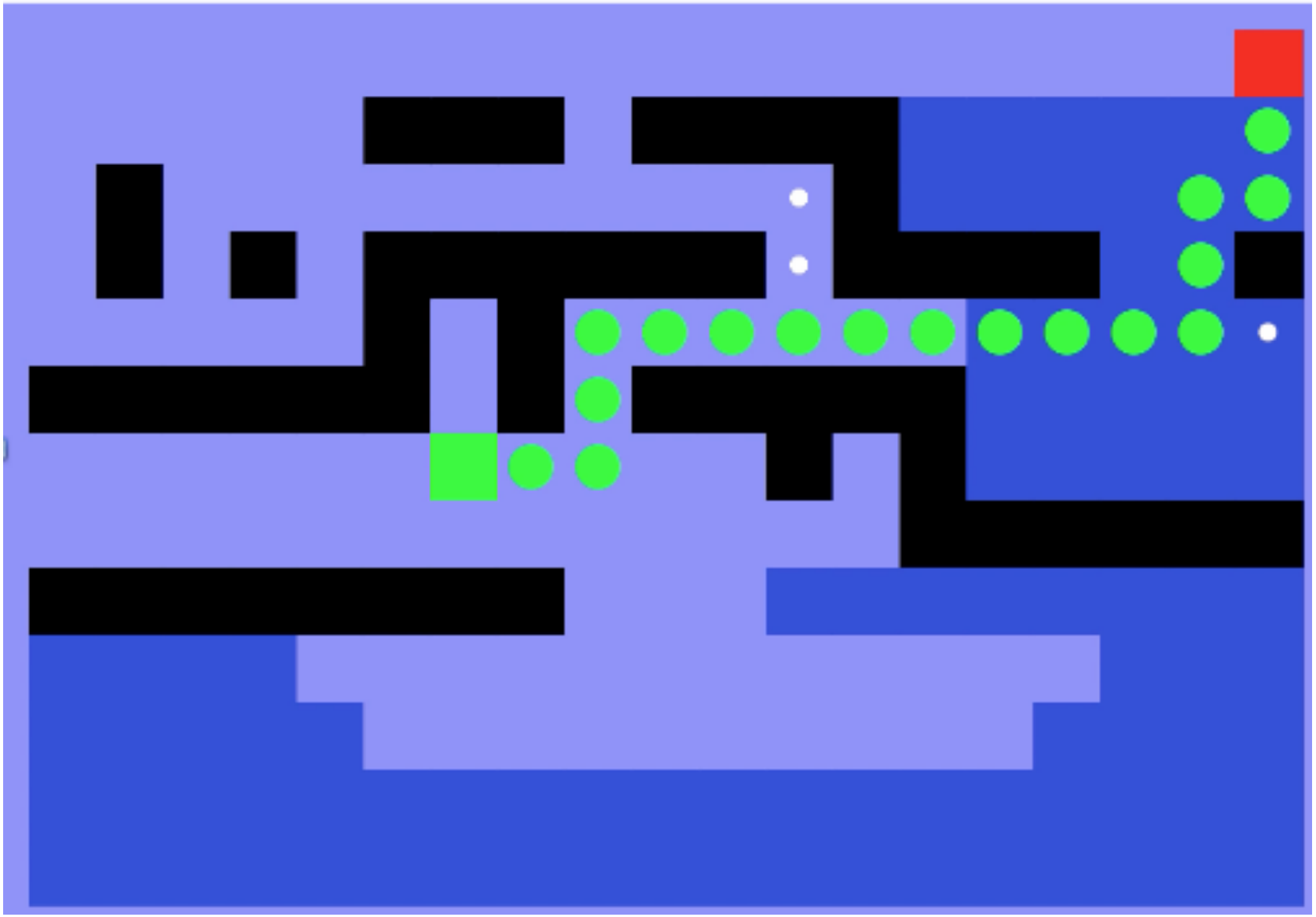
Labirint – demo Uniform Cost Search

Labirint – demo Uniform Cost Search



Labirint – demo Greedy search

Labirint – demo Greedy search



Labirint – demo A*

Labirint – demo A*