Inteligență Artificială

Bogdan Alexe
bogdan.alexe@fmi.unibuc.ro

Secția Tehnologia Informației, anul III, 2022-2023
Cursul 9

Note Proiect Kaggle - 361

Nr. crt.	Nume prenume student	Nota Proiect
1	ALBEI C. C. LIVIU-ANDREI	0,8
2	BALAWI I. USAMA	0
3	BORA T. I. DRAGOŞ-IONUŢ	0,85
4	CIOFLAN D. C. CEZAR-OCTAVIAN	0
5	CODREANU C. RADU-ŞTEFAN	0,9
6	COJOCARU L. ANDREI-LAURENŢIU	1,34
7	CREŢU V. LAURENŢIU-VASILE	1,42
8	CROITORU I. EDUARD-ADRIAN	0
9	DAVID D. V. VICTOR	1,16
10	FLORIAN D. LUCA-PAUL	1,04
11	IANCU G. FLORENTINA-MIHAELA	0,8
12	MARCU V. IOAN	0,65
13	MINCOV A. VIRGINICA-CRISTINA	0
14	NAE I. E. MIRCEA-ŞTEFAN	0,5
15	NĂSTASE D. MARIUS-NICOLAE	0,75
16	NIŢĂ B. AL. RAUL-ALEXANDRU	0
17	POPESCU C. ALINA-ELENA	1,03
18	SCORBUREANU V. SERGIU-NICHITA	0,5
19	SORETE G. ROBERT-ALEXANDRU	1,38
20	SPĂTARU D. CĂTĂLIN-GABRIEL	1,43
21	STANA M. D. MARIUS-VLAD	0,8
22	STOIAN C. O. VLAD	0,7
23	ŢUGUI I. IUSTIN-ION	0,5

Note Proiect Kaggle - 362

Nr. crt.	Nume prenume student	Nota Proiect
1	BILICI C. MIHAI-RAZVAN	0,65
2	BLOGU AL. FL. ADRIAN-TOMA	0,73
3	BORCAN M. CRISTIAN-BOGDAN	1,16
4	BRÎNCEANU E. ANDI-MĂDĂLIN	1,21
5	CIBOTARI GH. AUGUSTIN-ION	0,97
6	DRAGHIOTI I.-C. ANDREEA-MARIA	0,85
7	GHERGU M. NICOLAE-MARIUS	1,02
8	HOLMANU M. ANTONIO-MARIUS	0
9	LICU N. MIHAI-GEORGE	1,4
10	LINCĂ I. M. BIANCA-MARIA	0
11	MATACHE M. ALEXANDRU	0,9
12	MATEI N. B. TUDOR-CRISTIAN	1,43
13	MORARU M. RADU-ANDREI	1,41
14	MUŞAT E. R. SILVIU-GEORGE	0,7
15	STANCIU M. ALEXANDRA-ANDREEA	1,42
16	STĂTESCU I. RELU	1
17	TELEA C. C. MARIA-LAURA	1,31
18	TOMA M. FL. ALEXANDRU	0,85
19	TRĂNCĂNĂU M. O. CEZAR-ALEXANI	1,16
20	VRACIU V. ANDREEA	0,9

Note Proiect Kaggle - 363

Nr. crt.	Nume prenume student	Nota Proiect
1	ANTONESCU N. CRISTINA-ANDREEA	0,93
2	APOSTU A. V. MIHAI-ADRIAN	0,8
3	BANCEA C. DAN-ANDREI	0,75
4	BĂNESARU I. DENISA-GEORGIANA	1,24
5	BURLACU V. MIRCEA-FLORIAN	0,8
6	CROITORU M. VLADIMIR-ANDREI	0
7	CUCU AL. ŞTEFAN-CĂTĂLIN	1,44
8	DURA D. FL. ALEXANDRU-BOGDAN	1,01
9	ENCIU L. C. ELENA-CRISTINA	0,85
10	FĂRCĂŞANU E.C. TUDOR- ANDREI	1,5
11	FILIP L. RAZVAN ADRIAN	0,85
12	GAVRILĂ M.-S. VLAD-THEODOR	0,62
13	ISAC V. O. ŞTEFAN	0
14	MILITARU A.A ANDREI- ALEXANDRU	1,19
15	NIŢĂ M. ANDREEA-DIANA	1,13
16	PAVEL I. FL. ALEXANDRU	0,86
17	POPESCU G. FLORIN- DANIEL	0
18	SON V. ANDREEA-MARINA	1,24
19	STANA D. M. ANDREEA-THEODORA	0,85
20	TRĂŞCĂLIE L. RADU-NICOLAE	0,92
21	TUDOR M. NICU-CORNEL	0
22	UDREA C. IULIA-MARIA	1,27
23	ZOTIC V. D. MELANIA-ANEMONA	1,23

Note Proiect Kaggle - 364

Nr. crt.	Nume prenume student	Nota Proiect
1	ALEXANDRESCU D. AL. TUDOR-ALE	1,35
2	ALEXANDRESCU D. MARIAN-GABRI	0,8
3	BRÎNCEANU D. RALUCA-ALEXANDR	0,54
4	BUTURUGĂ C. E. GEORGE-ALEXAND	0,7
5	CRIŢĂ B. M. ANDREI-IONUŢ	0
6	DINOIU V. C. NICOLETA-ANASTASIA	1,35
7	DUMITRACHE V. V. RĂZVAN-CRISTI	0,5
8	DUMITRU M. L. RADU-ANDREI	0,95
9	GHEORGHIŢĂ E. ELENA-RALUCA-LO	0,75
10	GHIŢĂ G. D. DARIUS-FILIP	0,5
11	GUGIU E. ALEXANDRU	0
12	IONESCU C. LORENA-ELENA	1,07
13	IONIŢĂ M. R. ROXANA-DIANA	0,93
14	ISTRATE V. DORIAN-MIHAI	0,85
15	LASCU M. V. GEORGE-ALBERT	0,87
16	MANOLE A. GEORGE-ADRIAN	1,24
17	OANCEA E. E. VLAD	0
18	POPA V. D. BOGDAN-GABRIEL	1
19	POPESCU C. MIHNEA	1,28
20	ŞAMATA GH. ROBERT	0,65
21	URLĂŢEANU C. A. ALEXANDRU-IOA	0,8

Note Proiect Kaggle – anul 4

Anul 4

Nr. crt.	Nume prenume student	Nota Proiect
1	DINU ALEXANDRU	1,07
2	HONCIU BOGDAN	0,5
3	VARASCIUC ANDREI	0,52
4	BUCURIE BOGDAN	0,9
5	CRISTEA MIHNEA STEFAN	0,7
6	JIGAU ANDREI	0,96

Recapitulare – cursul trecut

1. Agenți inteligenți și mediile în care aceștia funcționează

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

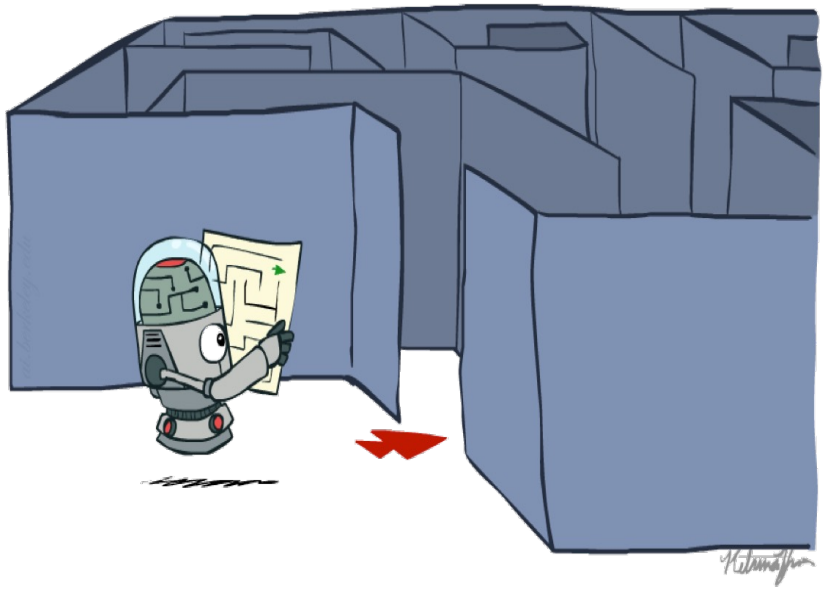
Cuprinsul cursului de azi

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

Căutare

- Problemă de căutare
  - stări (configurații ale lumii)
  - acțiuni și costuri
  - funcția succesor
  - stare inițială și stare scop

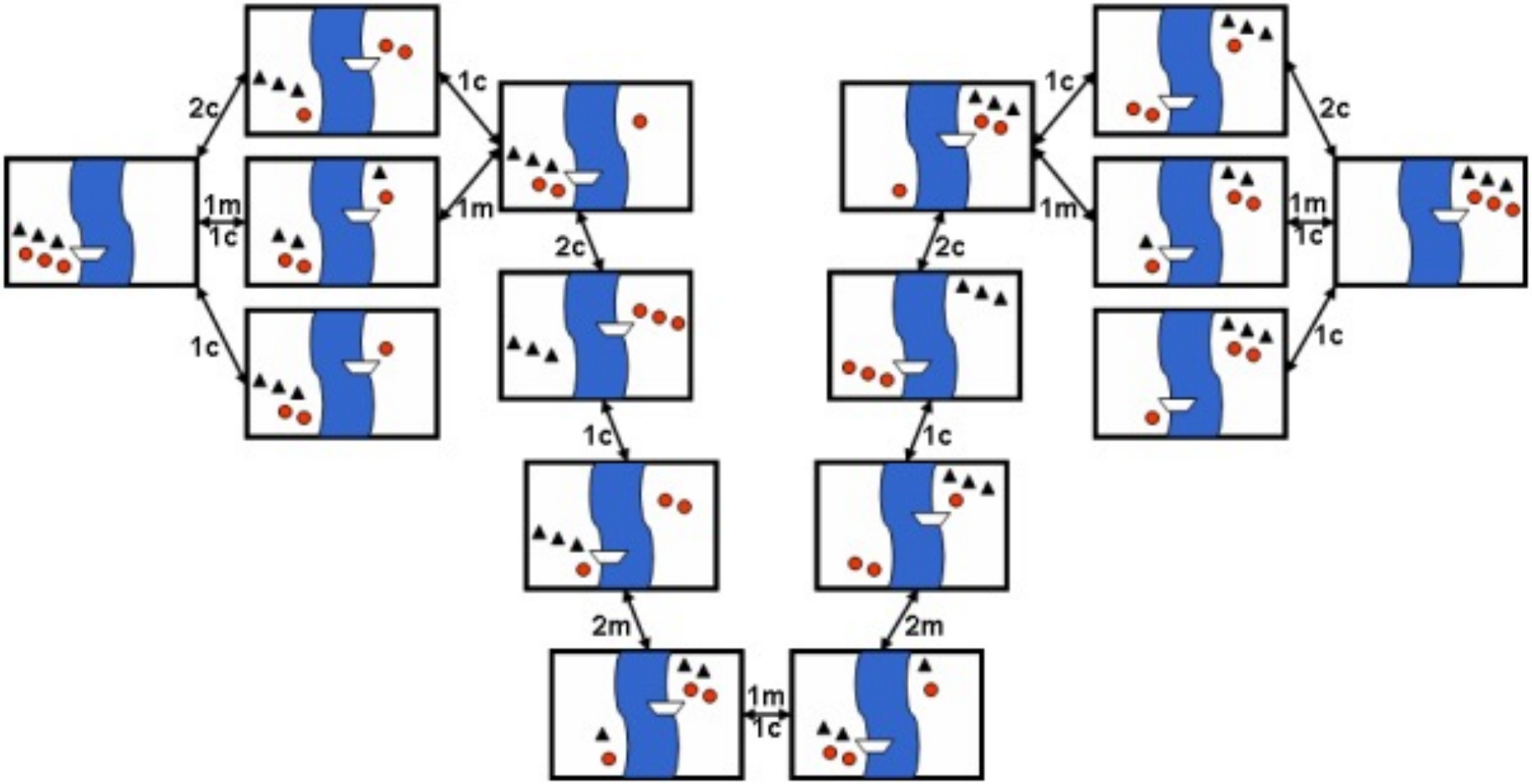


- Arbore de căutare
  - noduri: soluții parțiale
  - arce: acțiuni
  - o soluție parțială are asociat un cost (suma costurilor acțiunilor = arcele)

Graful de stări al unei probleme

- Graf orientat care conține toate stările lumii (o stare = un nod)
- Fiecare stare apare o singură dată
- Arcele reprezintă acțiuni
  - modelează funcția succesor
  - în ce stări putem ajunge din starea curentă realizând o anumită acțiune
  - deseori avem arc de la starea A la starea B și invers (acțiuni simetrice), crează circuite în arborele de căutare

Graful de stări al problemei misionarilor și a canibalilor



Arborele de căutare al unei probleme

- Tot un graf orientat, dar organizat pe nivele:
  - nivelul 0 = rădăcina (starea inițială)
  - nivelul i = nod obținut prin traversarea a i arce de la rădăcină din graful de stări

- Un nod desemnează o soluție parțială
- Un arc reprezintă un pas în construirea unei soluții
- Nodurile pot conține stări care se repetă
  – pot ajunge într-o stare urmând drumuri diferite

- Vrem să găsim un drum (= soluție) în graful de stări de la starea inițială la o stare scop (stare finală).

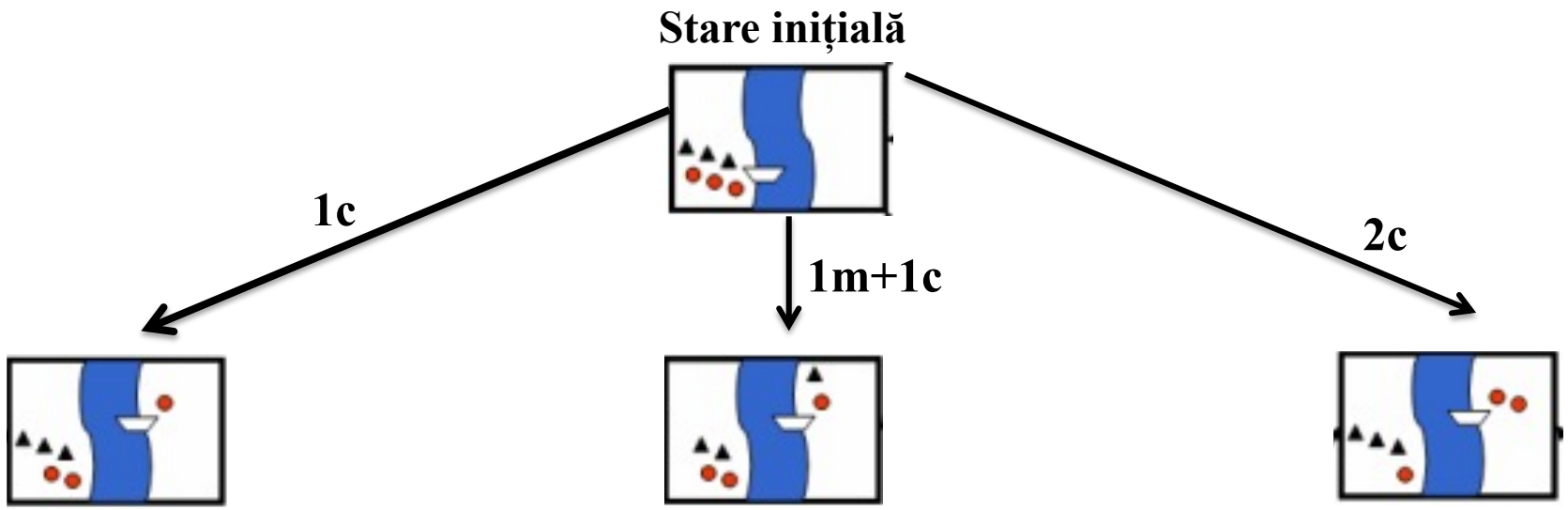
Arborele de căutare al problemei misionarilor și canibalilor

1c
1m
1m+1c
2c
2m

Stare inițială

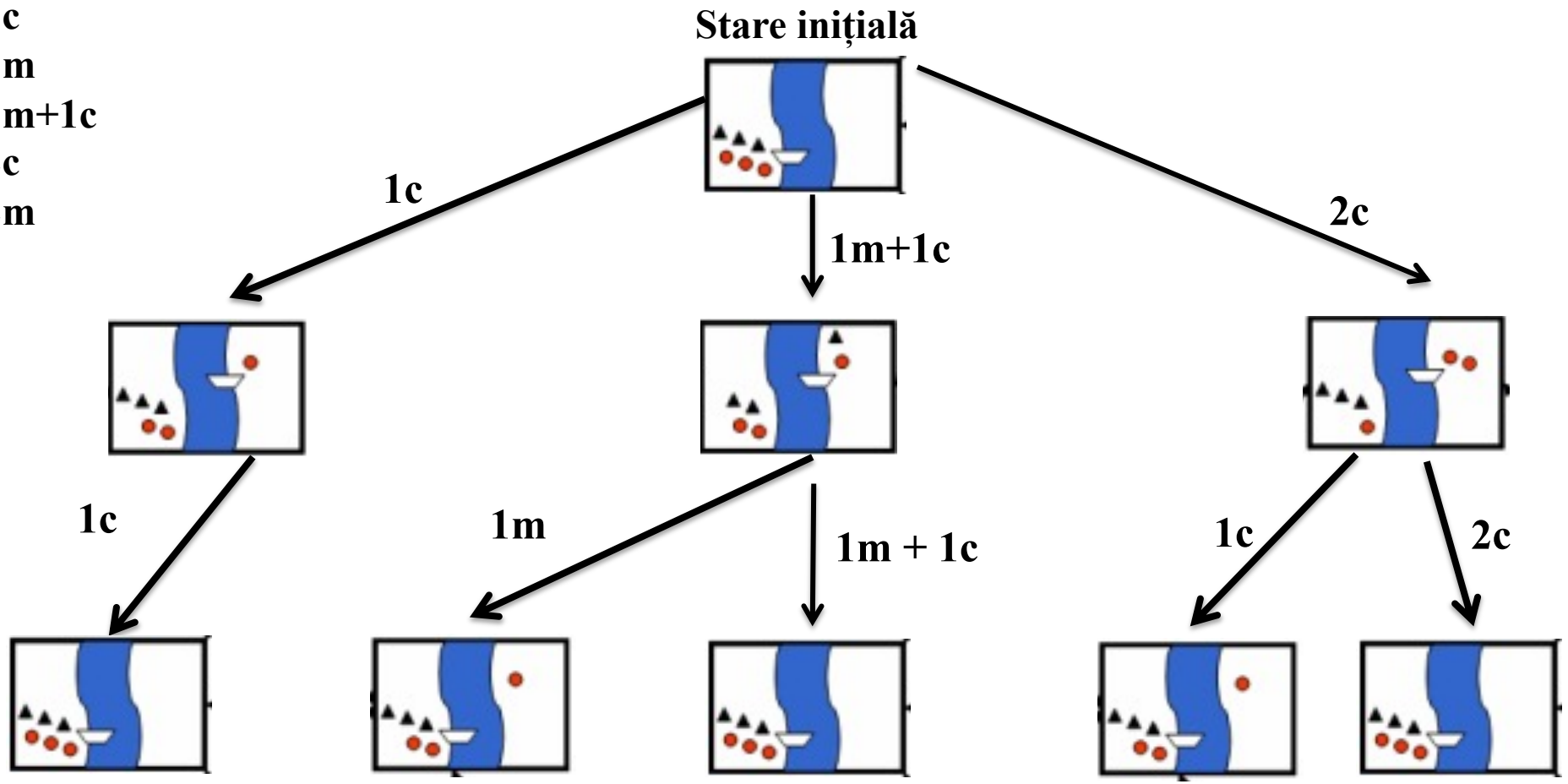
Arborele de căutare al problemei misionarilor și canibalilor

1c
1m
1m+1c
2c
2m



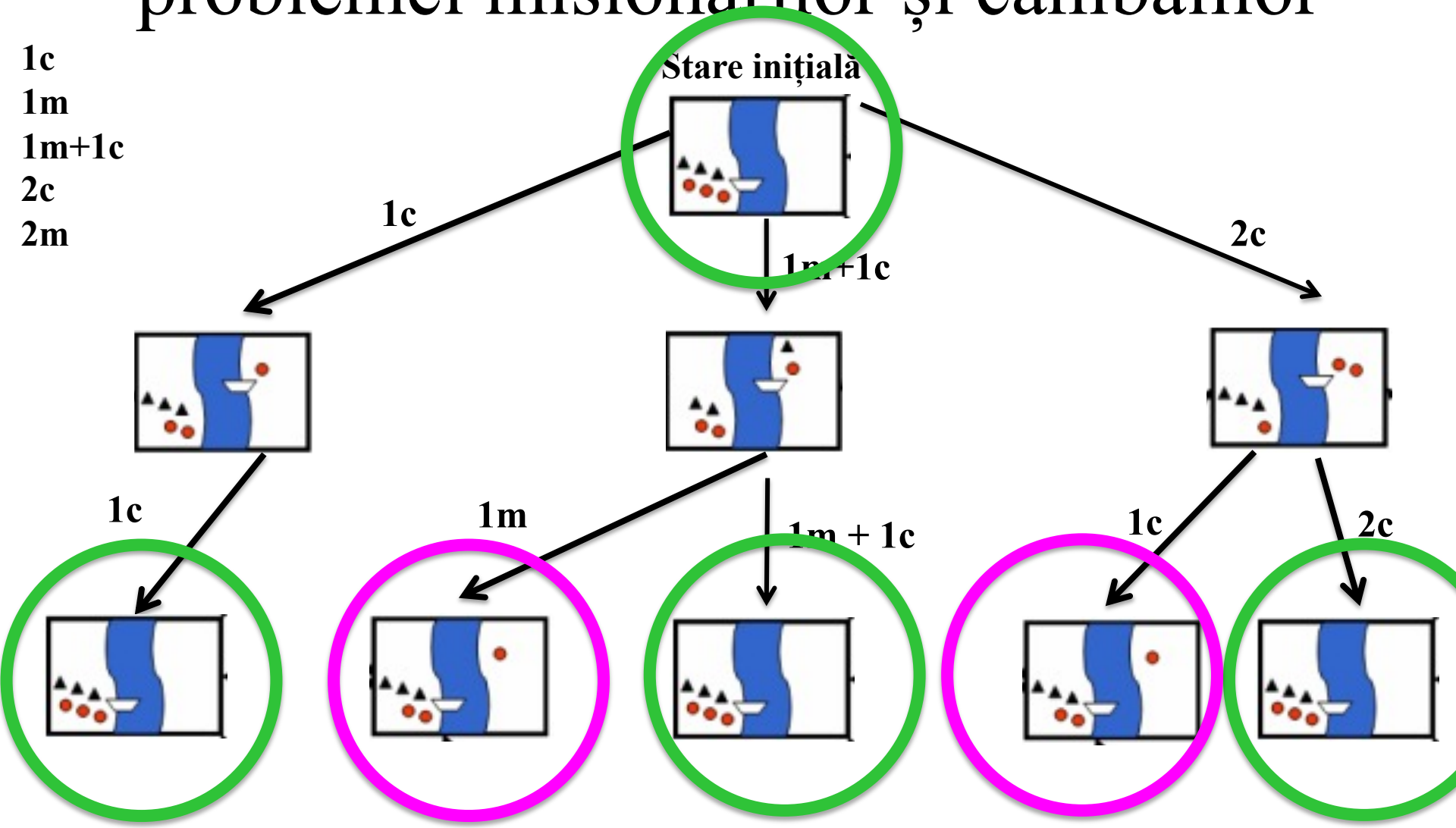
Arborele de căutare al problemei misionarilor și canibalilor

1c
1m
1m+1c
2c
2m



Arborele de căutare al problemei misionarilor și canibalilor

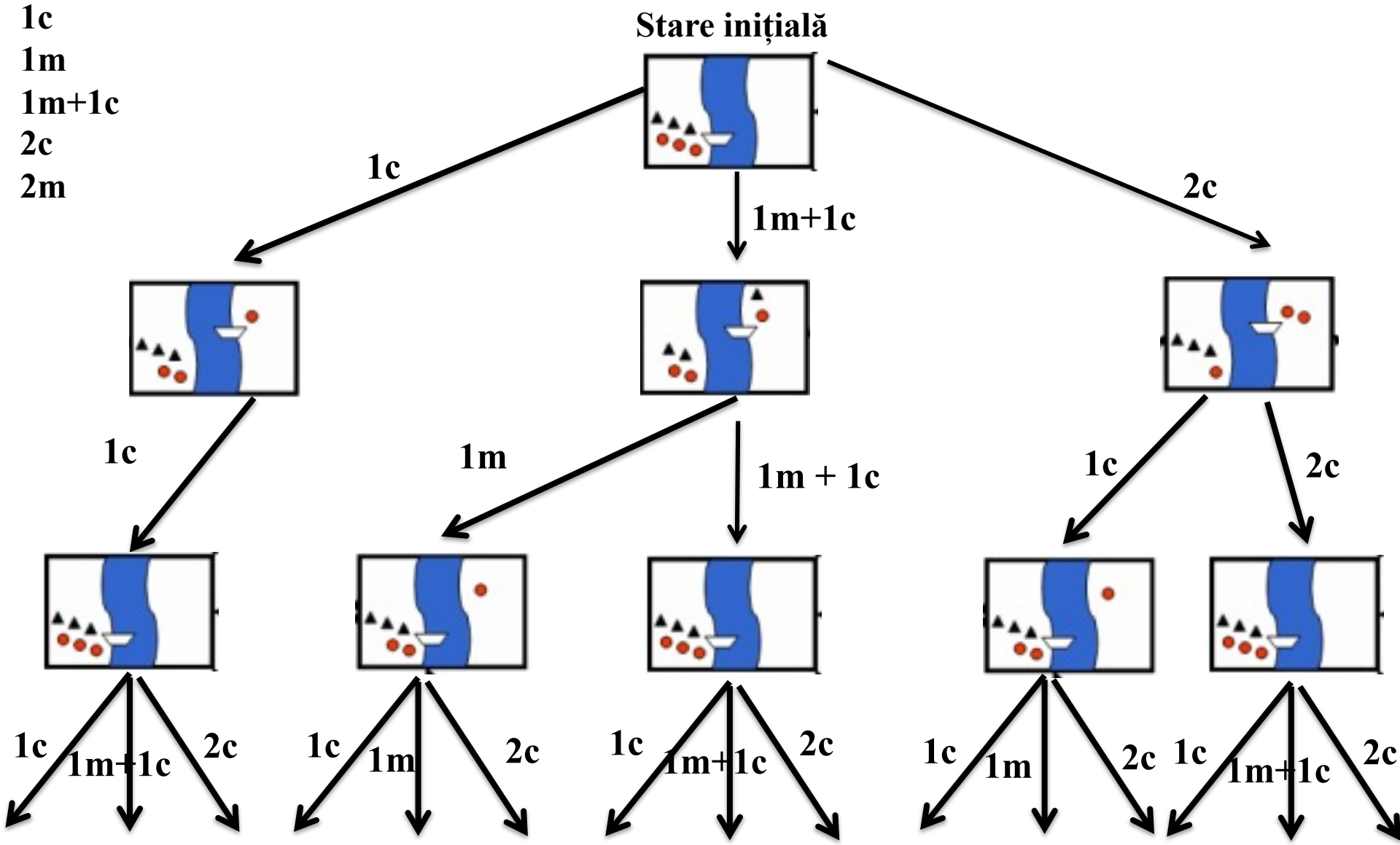
1c
1m
1m+1c
2c
2m



Stare care se repetă

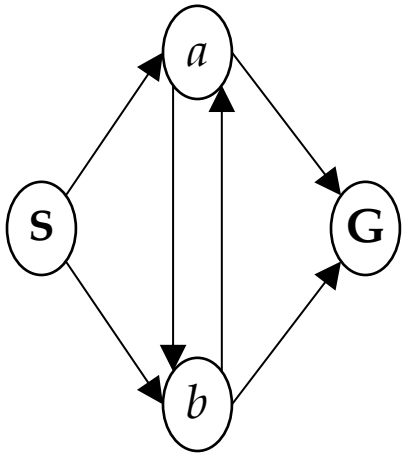
Stare care se repetă

Arborele de căutare al problemei misionarilor și canibalilor



Graful de stări vs. Arborele de căutare

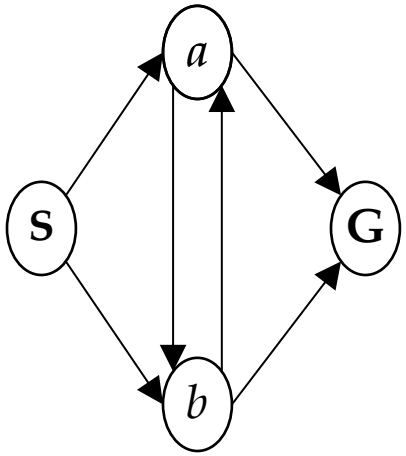
Considerăm graful cu 4 stări de mai jos:



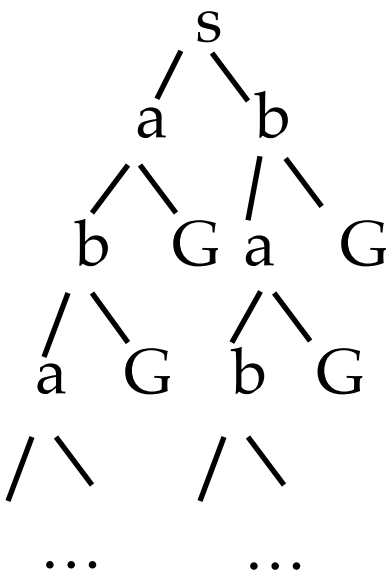
Cât de mare este arborele de căutare (pornim din S)

Graful de stări vs. Arborele de căutare

Considerăm graful cu 4 stări de mai jos:

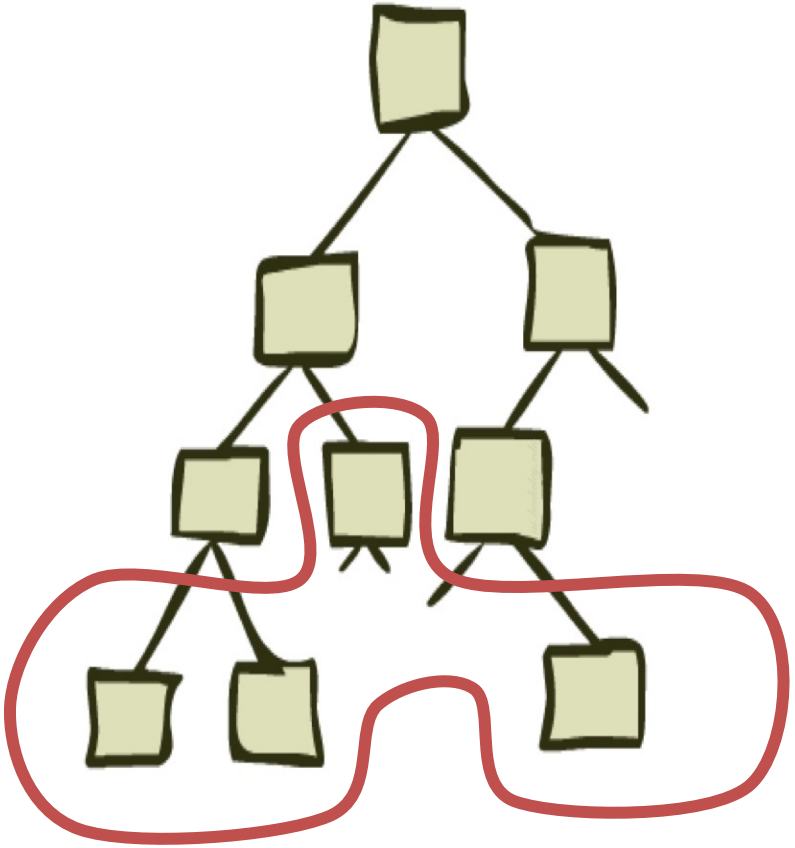


Cât de mare este arborele de căutare (pornim din S)

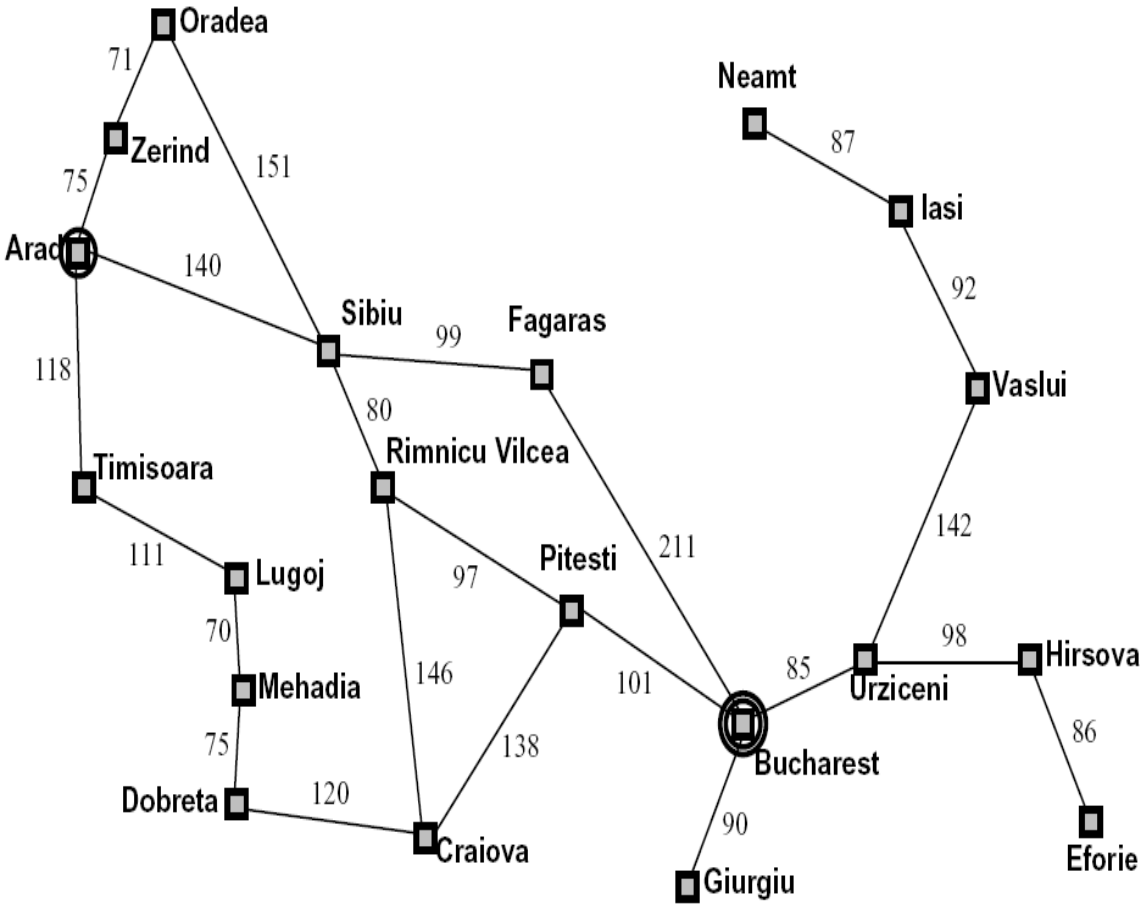


Important: foarte multe structuri repetitive în arborele de căutare

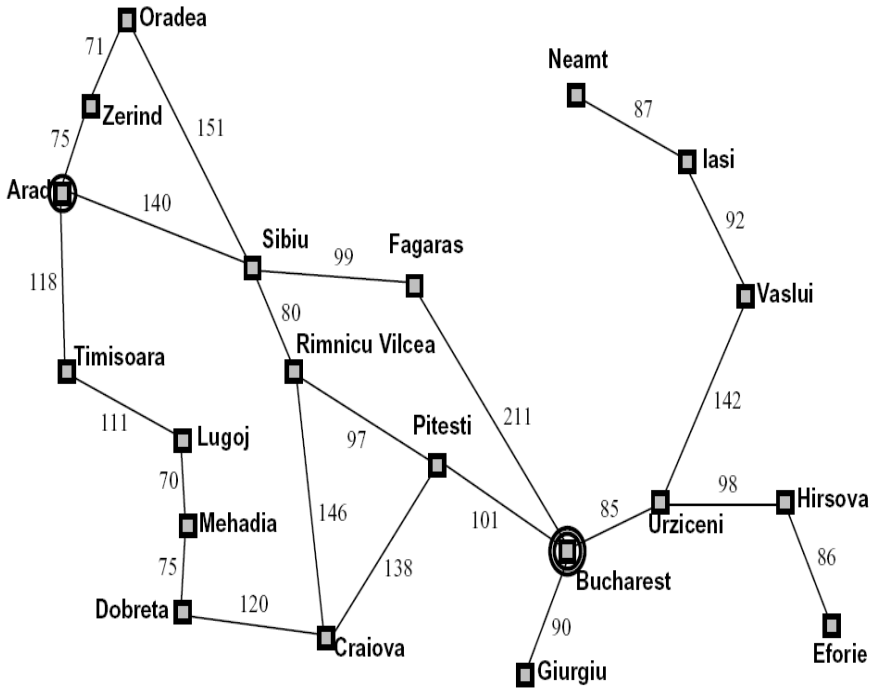
Căutare cu un arbore de căutare



Exemplu de căutare: Arad-București

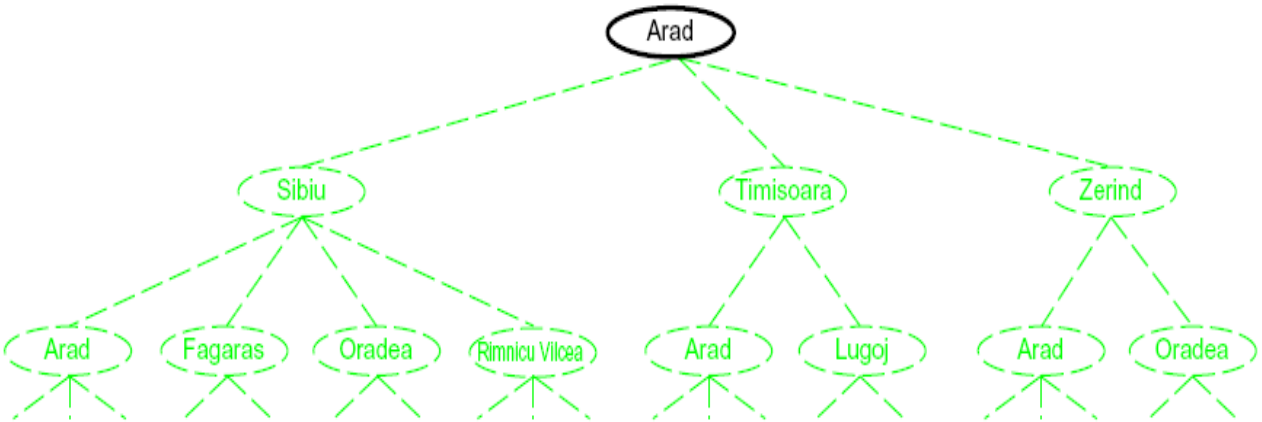


Exemplu de căutare: Arad-București



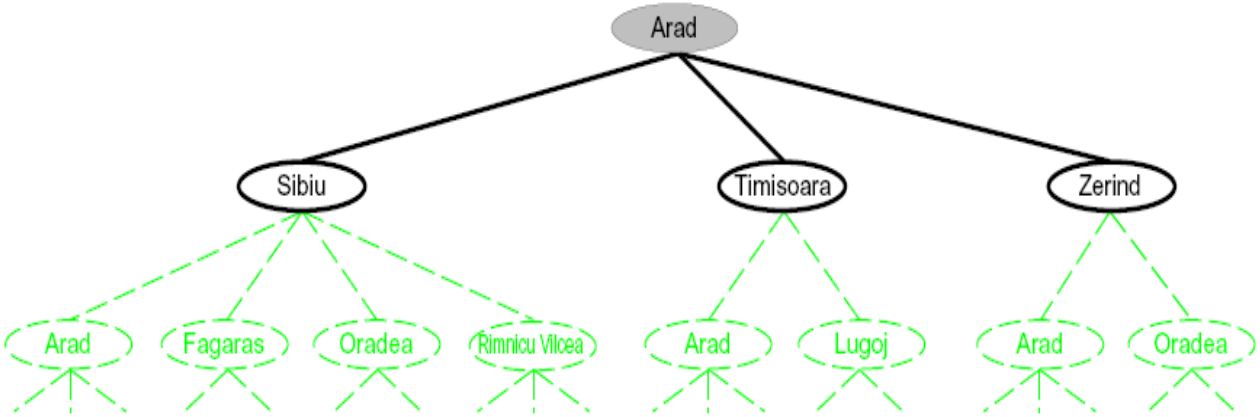
- Spațiul stărilor:
  - Orașe
- Funcție succesor:
  - drumuri: mergi la orașul adiacent cu costul = distanța
- Stare inițială:
  - Arad
- Scop care poate fi verificat:
  - stare == Bucharest?
- Soluție?

Căutare cu un arbore de căutare



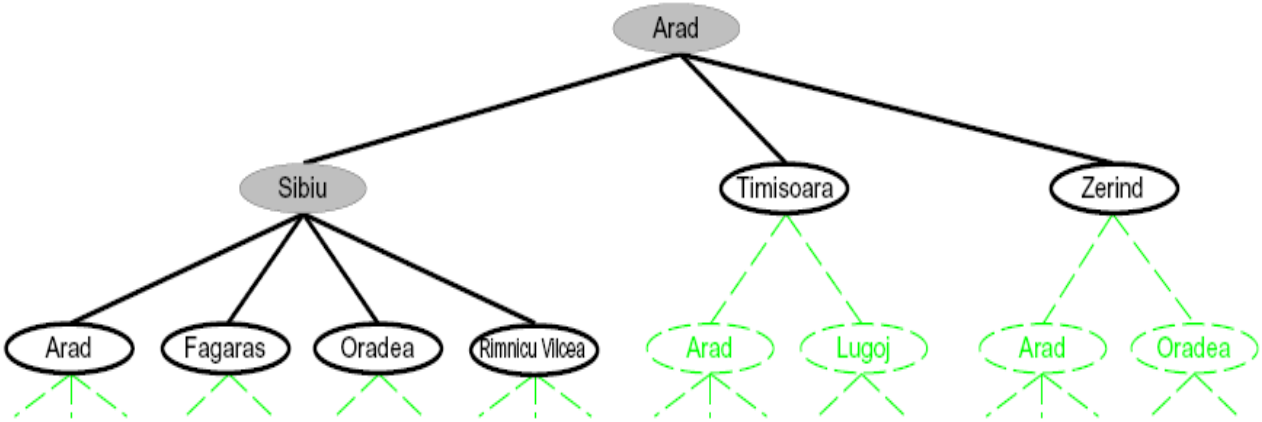
- Căutare:
  - expandarea planurile posibile (nodurile din arbore)
  - menținerea unei liste de noduri de frontieră, ce separă graful spațiului de căutare în regiunea explorată și regiunea neexplorată
  - se încearcă expandarea unui număr cât mai mic de noduri

Căutare cu un arbore de căutare



- Căutare:
  - expandarea planurile posibile (nodurile din arbore)
  - menținerea unei liste de noduri de frontieră, ce separă graful spațiului de căutare în regiunea explorată și regiunea neexplorată
  - se încearcă expandarea unui număr cât mai mic de noduri

Căutare cu un arbore de căutare



- Căutare:
  - expandarea planurile posibile (nodurile din arbore)
  - menținerea unei liste de noduri de frontieră, ce separă graful spațiului de căutare în regiunea explorată și regiunea neexplorată
  - se încearcă expandarea unui număr cât mai mic de noduri

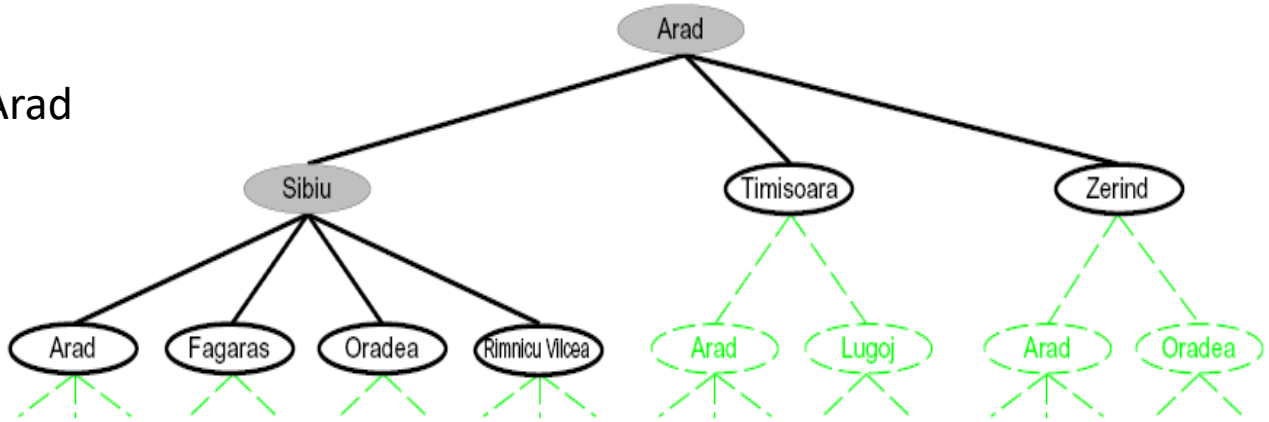
Arbore de căutare general

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

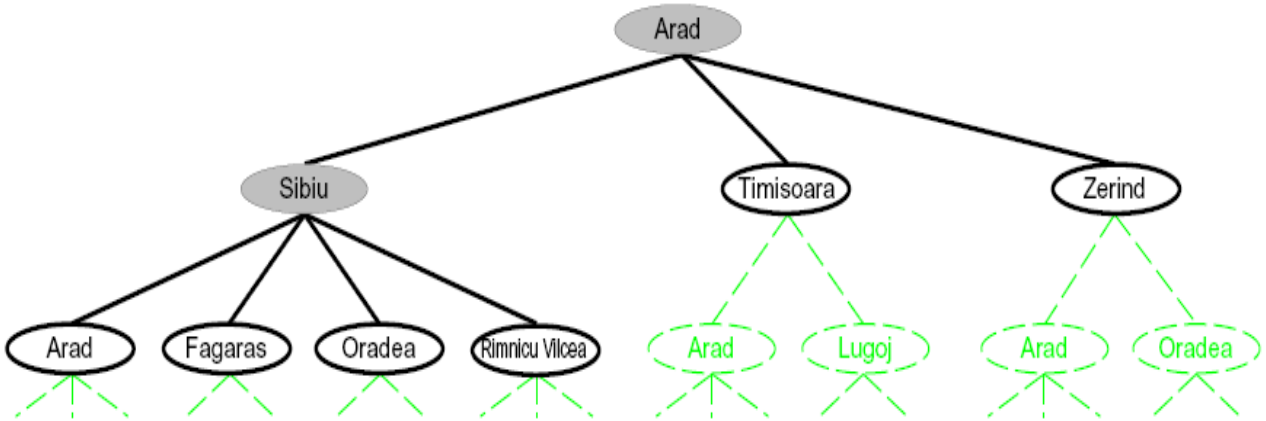
inițializează **frontiera** folosind **starea inițială a problemei**

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**

Drum redundant: Arad-Sibiu-Arad



Arbore de căutare general



Frontiera conține nodurile frunză din arborele de căutare (în desenul de mai sus frontiera e dată de nodurile albe: Arad, Făgăraș, Oradea, Râmnicu Vâlcea, Timișoara, Zerind).
Fiecare nod frunză reprezintă o soluție parțială construită = o secvență de acțiuni ce pornește din nodul rădăcină.

La fiecare pas:

- **alege pe baza unei strategii un anumit nod frunză din frontieră;**
- **testează dacă nodul curent este nod-scop: dacă este nod-scop returnează soluția;**
- **expandează nodul curent: generează toți succesorii nodului curent și îi adaugă în frontieră**

Arbore de căutare general

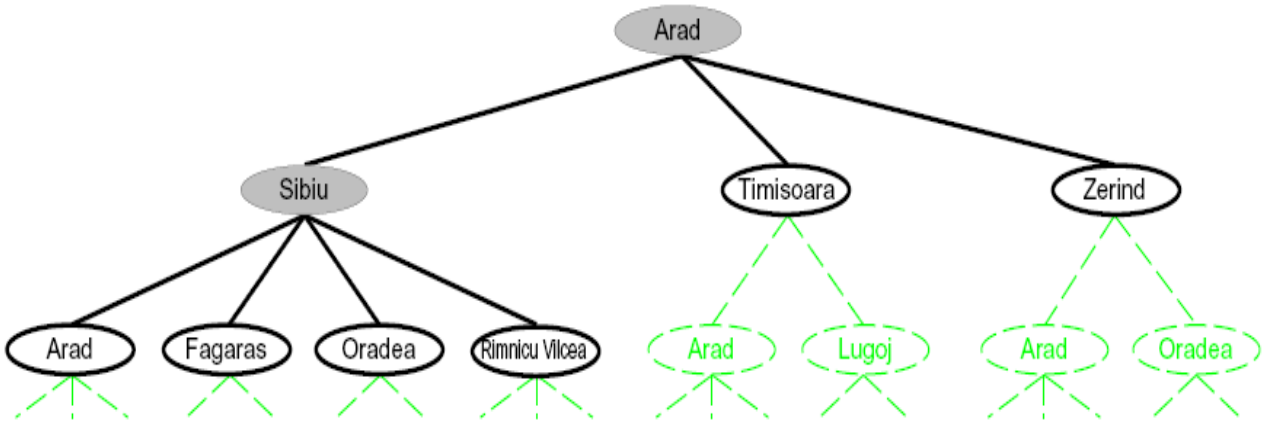
funcția ARBORE-CAUTARE (**problema**, **strategie**) **returnează** o soluție sau eșec

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

- coadă (FIFO) – scoate cel mai vechi element din listă
- stivă (LIFO) – scoate cel mai nou element din listă
- listă de priorități – scoate un element cu cea mai mare prioritate conform unei funcții de ordonare

Arbore de căutare general

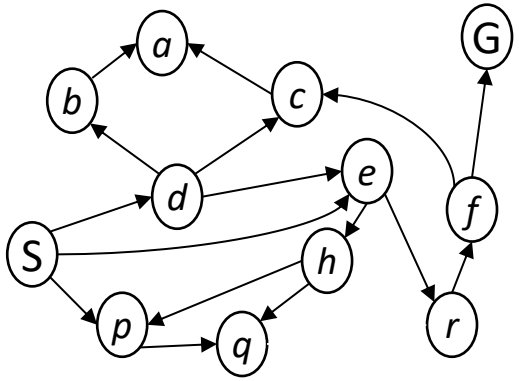


Frontiera conține nodurile frunză din arborele de căutare (în desenul de mai sus frontiera e dată de nodurile albe: Arad, Făgăraș, Oradea, Râmnicu Vâlcea, Timișoara, Zerind).
Fiecare nod frunză reprezintă o soluție parțială construită = o secvență de acțiuni ce pornește din nodul rădăcină.

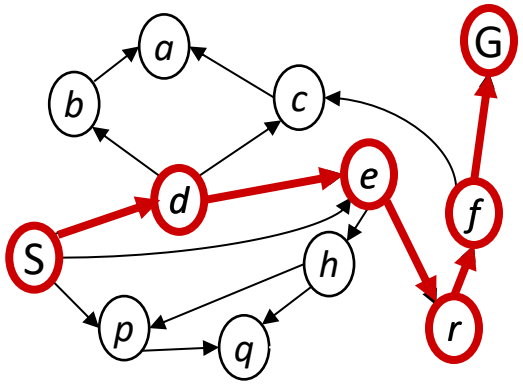
La fiecare pas:

- **alege pe baza unei strategii un anumit nod frunză din frontieră;**
- **testează dacă nodul curent este nod-scop: dacă este nod-scop returnează soluția;**
- **expandează nodul curent: generează toți succesorii nodului curent și îi adaugă în frontieră dacă aceștia nu au fost explorați (vizitați) sau nu sunt deja în frontieră (evită crearea de circuite)**

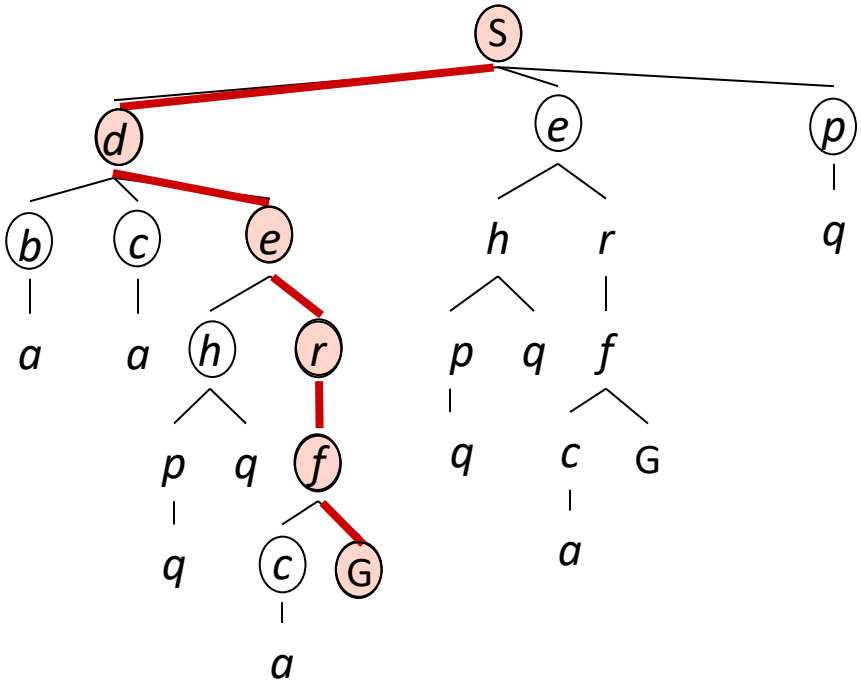
Arbori de căutare - exemple



Arbori de căutare - exemple



Arborele de căutare



Frontiera

~~s~~

~~s → d~~

s → e

s → p

s → d → b

s → d → c

~~s → d → e~~

s → d → e → h

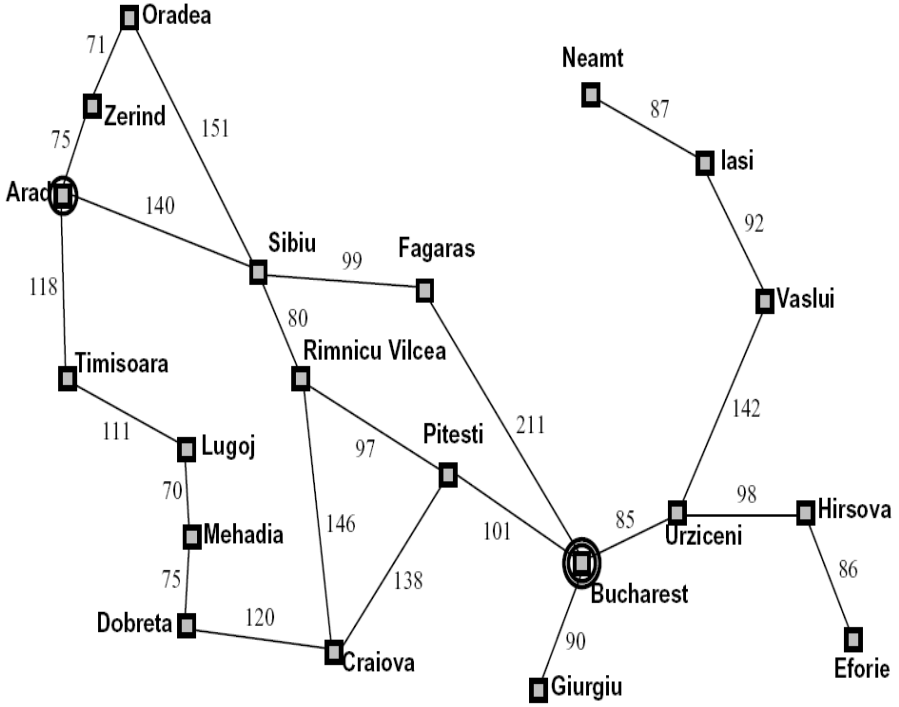
~~s → d → e → r~~

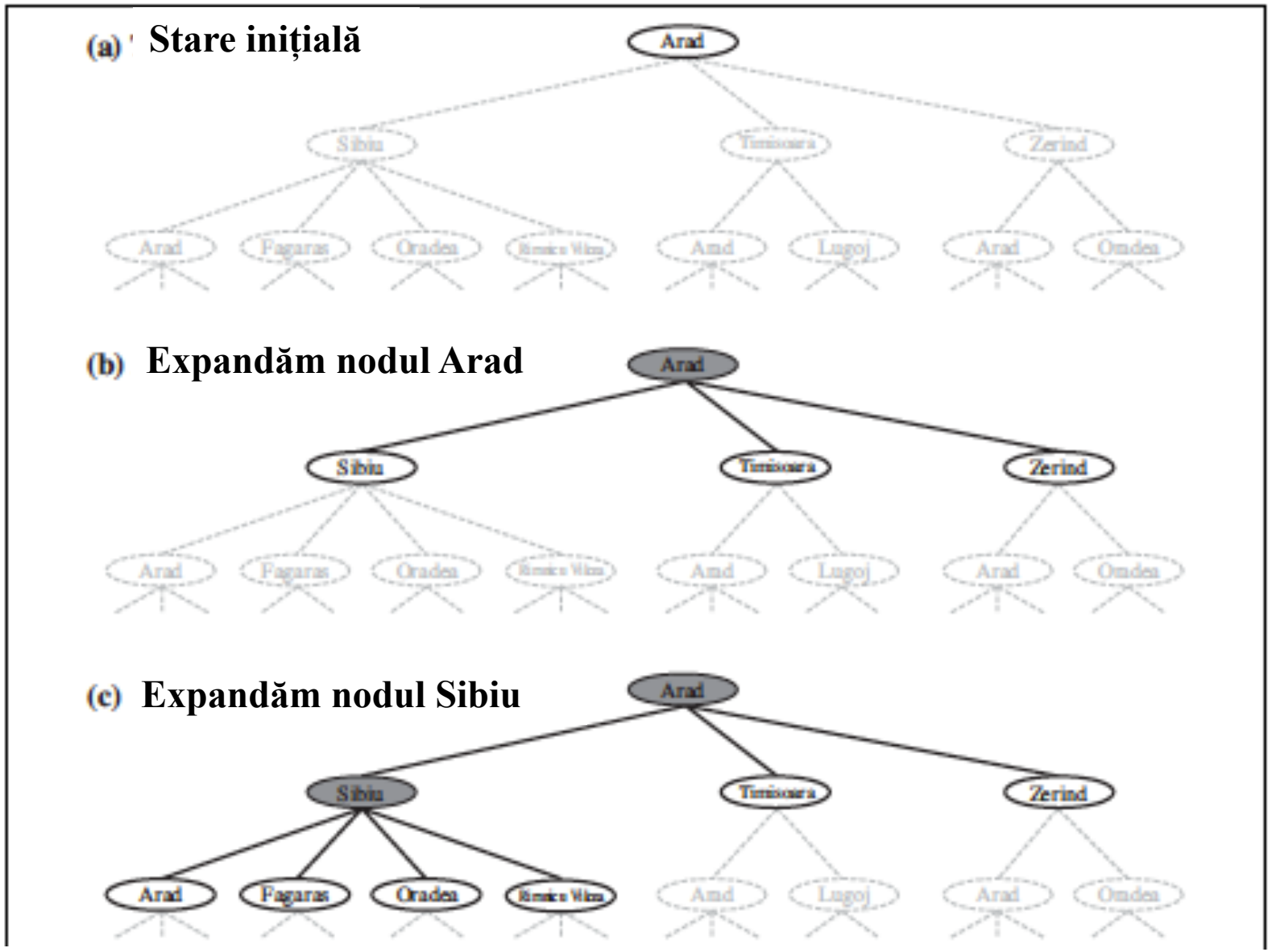
~~s → d → e → r → f~~

s → d → e → r → f → c

~~s → d → e → r → f → G~~

Arbori de căutare - exemple

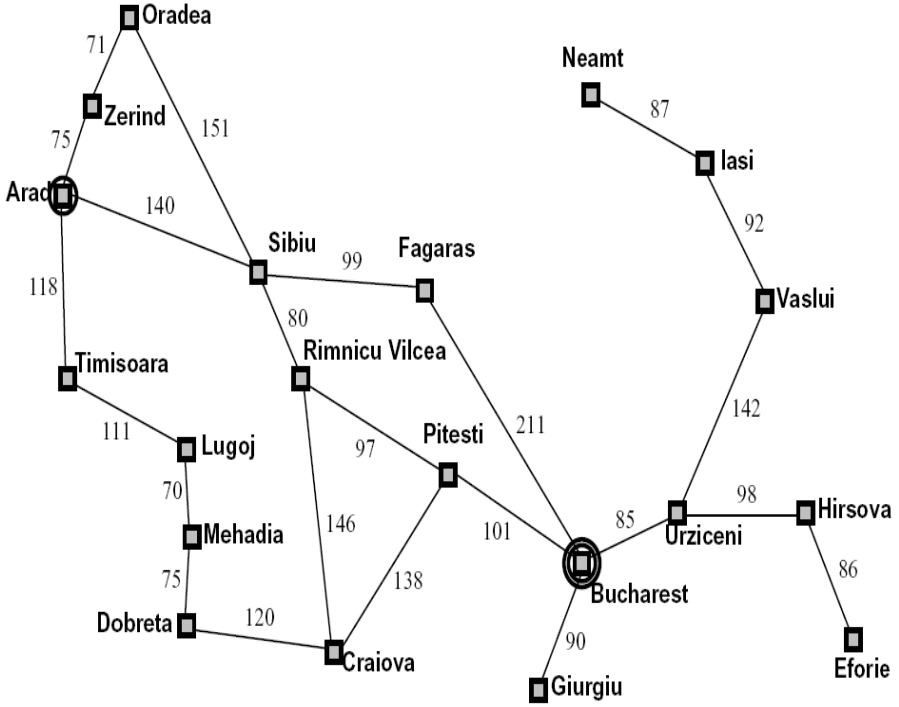




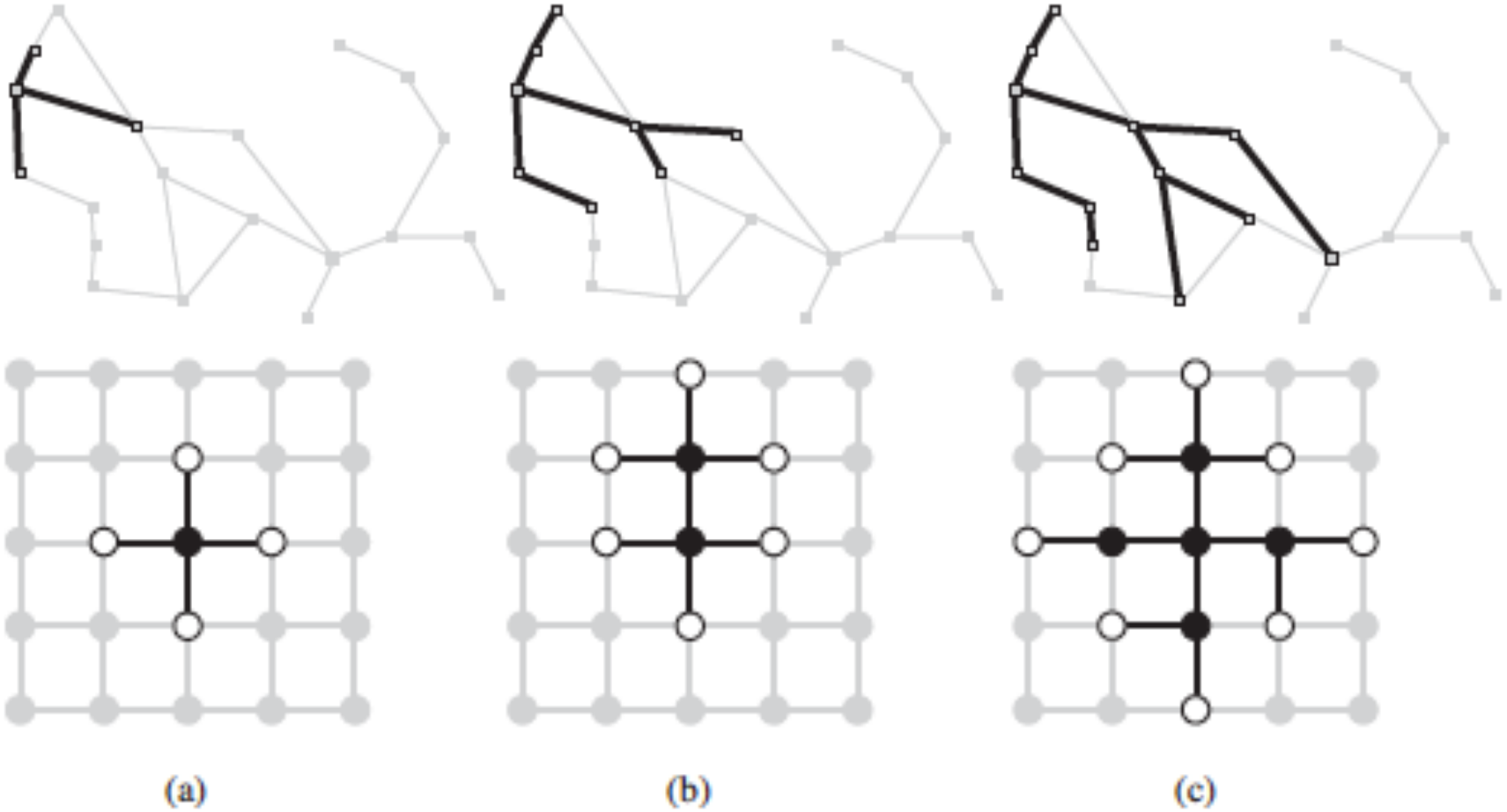
Trei tipuri de noduri:

- **noduri vizitate**
- **noduri din frontieră**
- **noduri negenerate**

Arbori de căutare - exemple



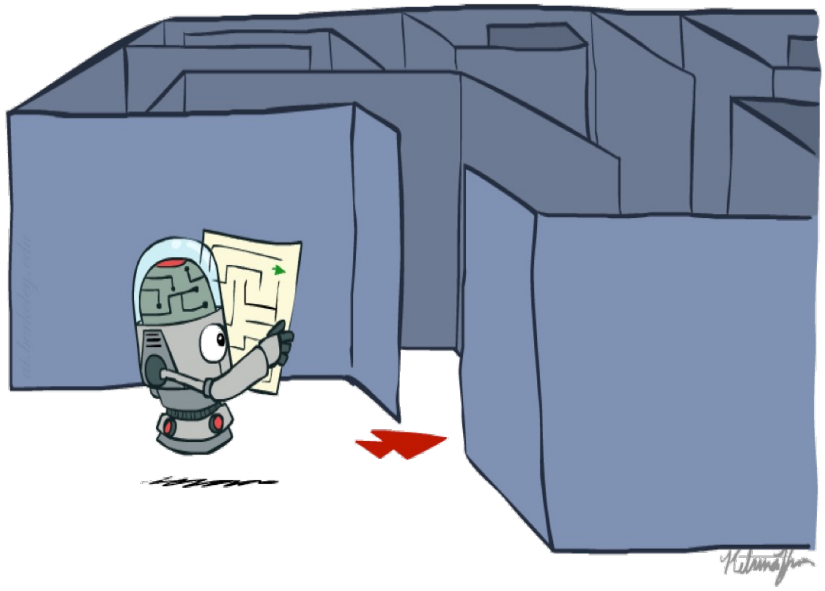
Arbori de căutare - exemple



Frontiera (nodurile albe) separă regiunea explorată a stărilor (nodurile negre) de regiunea neexplorată (nodurile gri). În (a), numai nodul rădăcină a fost expandat. În (b), un nod frunză din cei patru succesori ai nodului rădăcină a fost expandat. În (c), toți succesorii nodului rădăcină au fost expandați în ordinea acelor de ceasornic.

Căutare

- Problemă de căutare
  - stări (configurații ale lumii)
  - acțiuni și costuri
  - funcția succesor
  - stare inițială și stare scop



- Arbore de căutare
  - noduri: soluții parțiale
  - arce: acțiuni
  - o soluție parțială are asociat un cost (suma costurilor acțiunilor = arcele)

- Strategie de căutare
  - construiește un arbore de căutare
  - alege ordinea în care explorează/expandează nodurile din frontieră
  - optimale dacă găsesc soluțiile de cost minim

Strategii de căutare

Strategie = alegerea unei ordini de explorare a nodurilor

Strategiile de căutare se evaluează conform următoarelor patru criterii:

1. **Completitudine**: dacă, atunci când o soluție există, strategia dată garantează găsirea acesteia

2. **Complexitate a timpului**: durata în timp pentru găsirea unei soluții

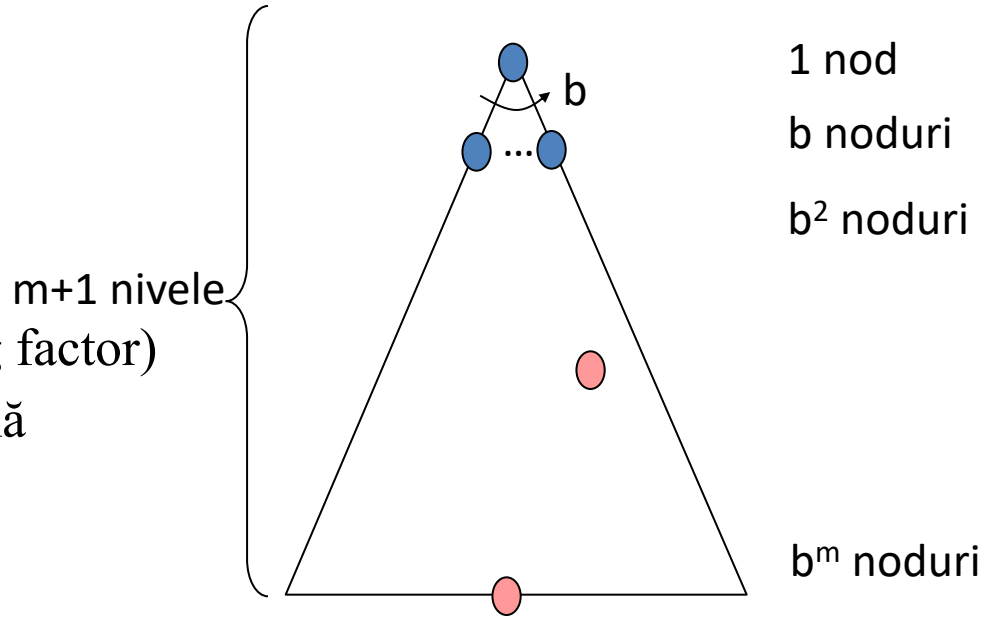
3. **Complexitate a spațiului**: necesitățile de memorie pentru efectuarea căutării

4. **Optimalitate**: atunci când există mai multe soluții, strategia dată o găsește pe cea mai bună dintre ele (pe baza unei funcții – spre exemplu găsește funcția cu costul cel mai mic)

Strategii de căutare

- Completitudine: garantarea unei soluții dacă ea există?
- Optimalitate: garantarea uni soluții de cost minim?
- Complexitate timp?
- Complexitatea spațiu?

- Ilustrare arbore de căutare:
  - b = factor ramificare (branching factor)
  - m = adâncimea (nivelul) maximă
  - soluții la adâncimi diferite



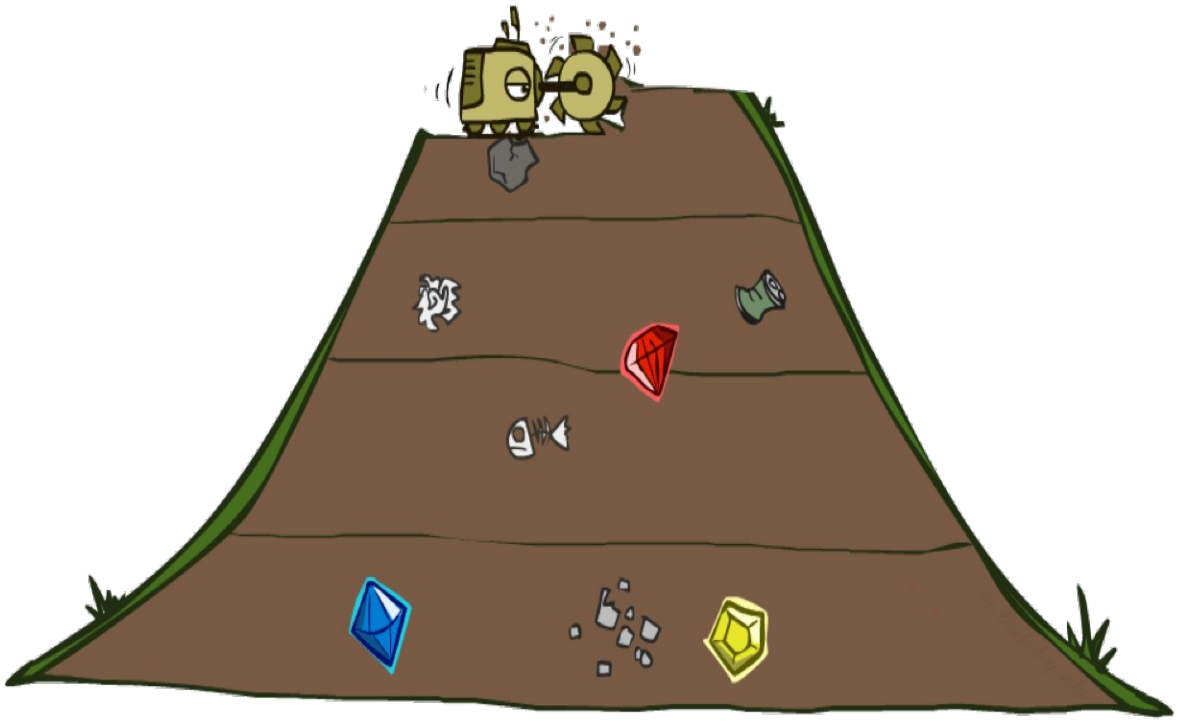
- Numărul de noduri în arbore?
  - $1 + b + b^2 + \ldots + b^m = O(b^m)$

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)

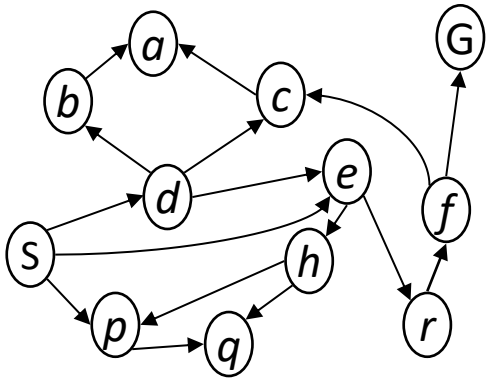
Căutare în lățime (bread-first = BF)

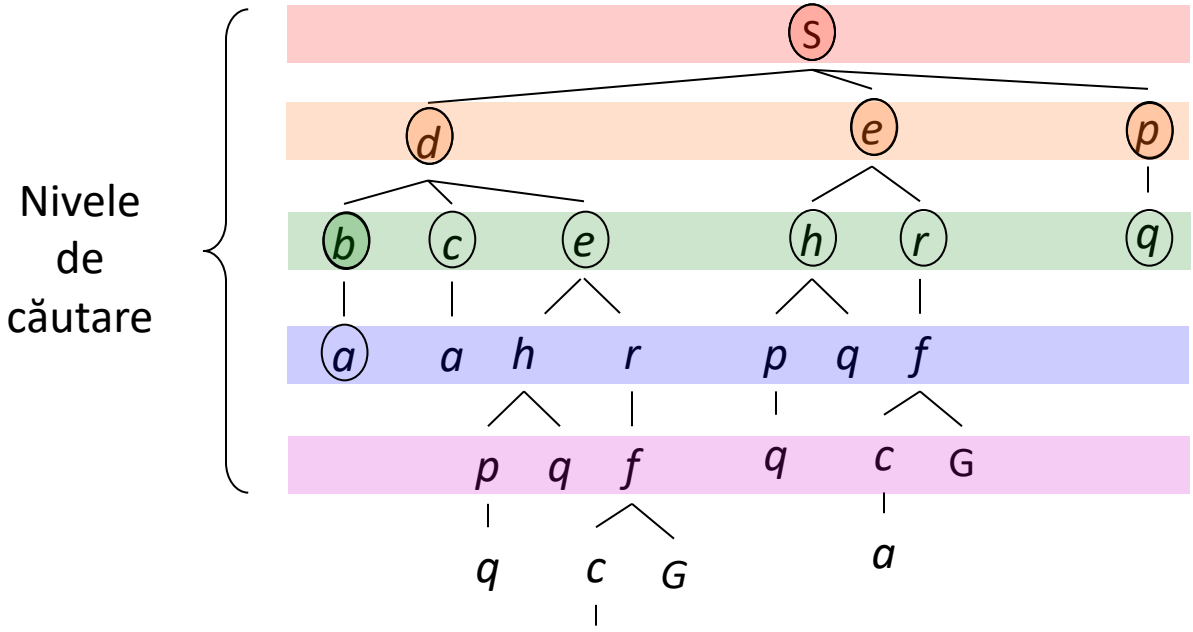


Căutare în lățime (bread-first = BF)

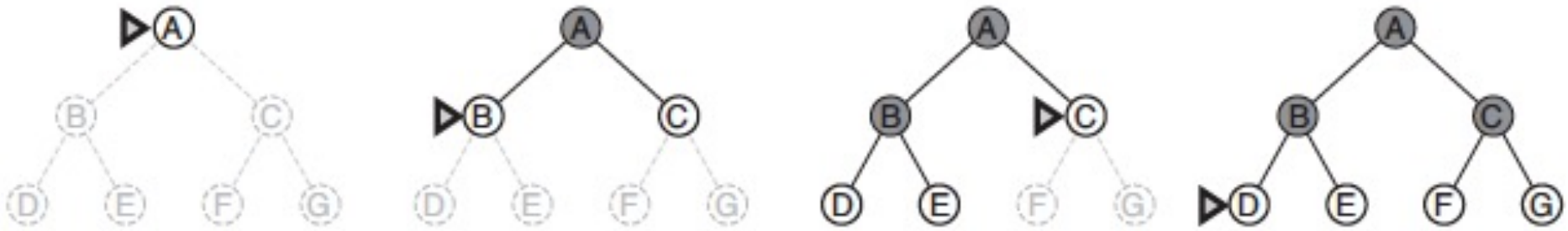
Strategie: expandează succesiv toate nodurile în ordinea adâncimii începând cu nodul rădăcină

Frontiera este implementată cu o coadă (FIFO)





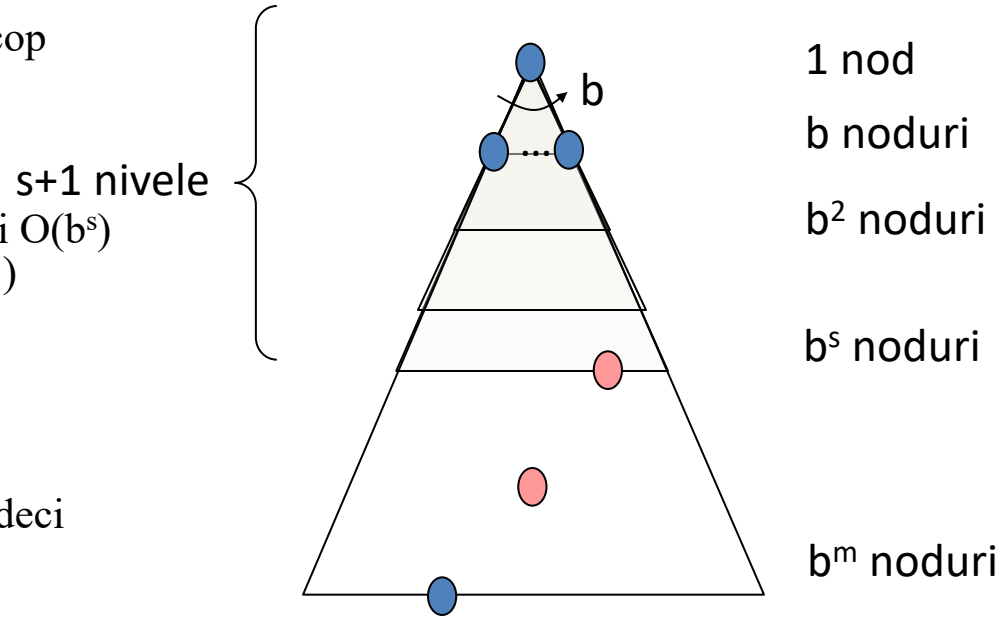
Căutare în lățime pe un arbore binar



Căutare în lățime pe un arbore binar: la fiecare pas, săgeata indică nodul care va fi expandat.

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!



Reducerea complexității pentru căutare în lățime

```
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
```

- pentru reducerea complexității timp pentru căutarea în lățime se verifică dacă un nod succesor generat este nod scop.
- dacă generăm noduri soluție la nivelul s, nu putem avea un nod de pe nivelul s+1 soluție
- reduce complexitatea timp de la $O(b^{s+1})$ la $O(b^s)$

Reducerea complexității pentru căutare în lățime

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

    inițializează frontiera folosind starea inițială a problemei
    dacă starea inițială este o stare scop
                atunci returnează soluția corespunzătoare
    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
        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
            dacă nodul succesor generat conține o stare scop
                atunci returnează soluția corespunzătoare
```

- pentru reducerea complexității timp pentru căutarea în lățime se verifică dacă un nod succesor generat este nod scop.
- dacă generăm noduri soluție la nivelul s, nu putem avea un nod de pe nivelul s+1 soluție
- reduce complexitatea timp de la $O(b^{s+1})$ la $O(b^s)$

Timpul și memoria necesară pentru căutarea în lățime - BF.

Considerăm:

- factor de ramificare (branching factor) b = 10;
- timp de procesare = 1 milion de noduri/secundă;
- necesat de memorie pentru stocarea unui nod = 1000 bytes

Adâncime	#Noduri	Timp	Memorie
2	110	.11 ms	107 kB
4	11110	11 ms	10.6 MB
6	10^6	1.1 sec	1 GB
8	10^8	2 min	103 GB
10	10^{10}	3 ore	10 TB
12	10^{12}	13 zile	1 PB
14	10^{14}	3.5 ani	99 PB
16	101^6	350 ani	10 EB

Timpul și memoria necesară pentru căutarea în lățime - BF.

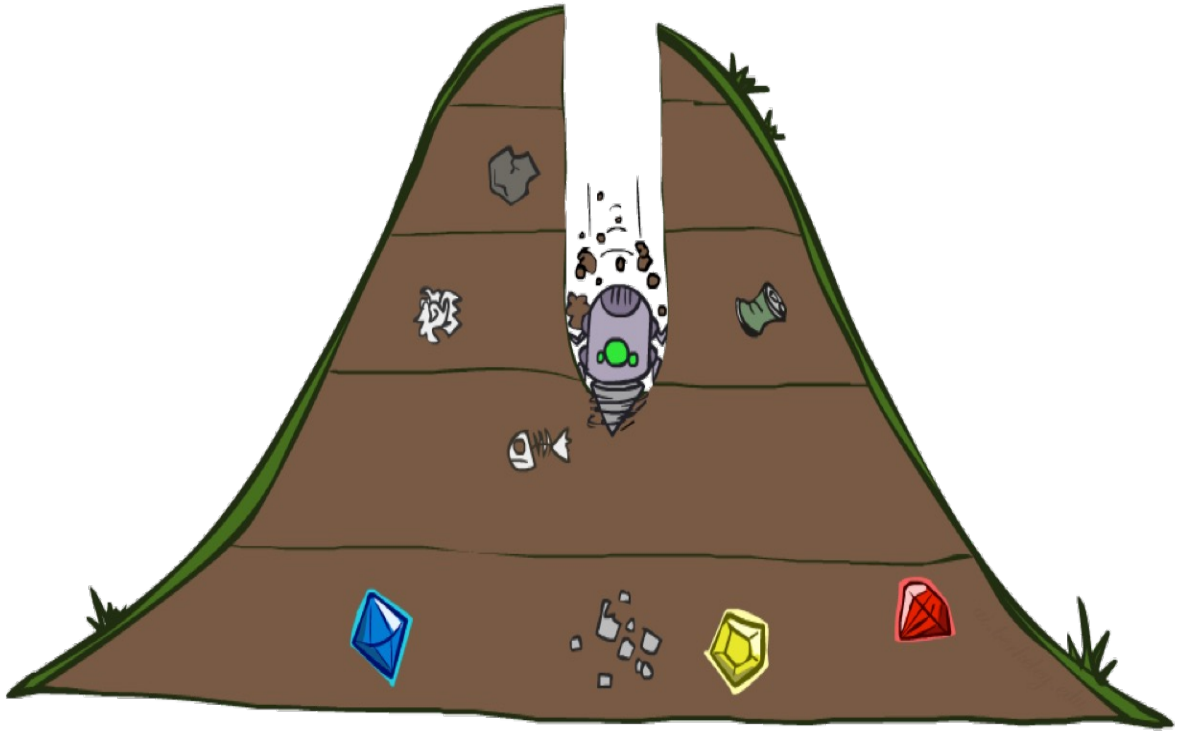
Adâncime	#Noduri	Timp	Memorie
2	110	.11 ms	107 kB
4	11110	11 ms	10.6 MB
6	10^6	1.1 sec	1 GB
8	10^8	2 min	103 GB
10	10^{10}	3 ore	10 TB
12	10^{12}	13 zile	1 PB
14	10^{14}	3.5 ani	99 PB
16	101^6	350 ani	10 EB

Remarci:

- complexitate exponențială timp și spațiu

- timp de execuție aproape infinit pentru adâncimi relativ moderate

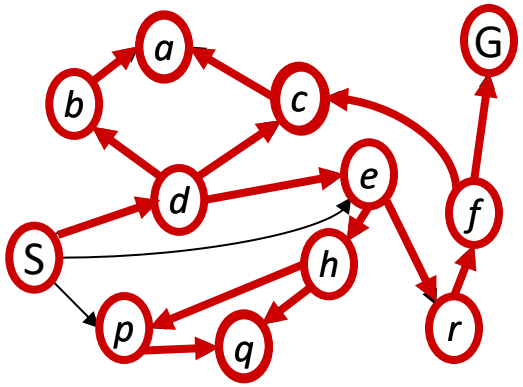
Căutarea în adâncime (depth-first = DF)

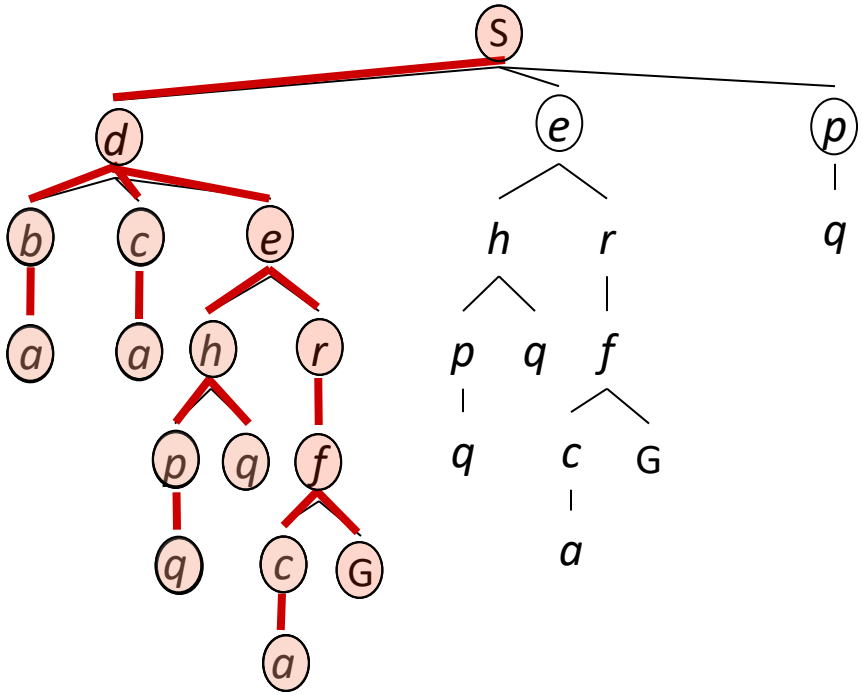


Căutarea în adâncime (depth-first = DF)

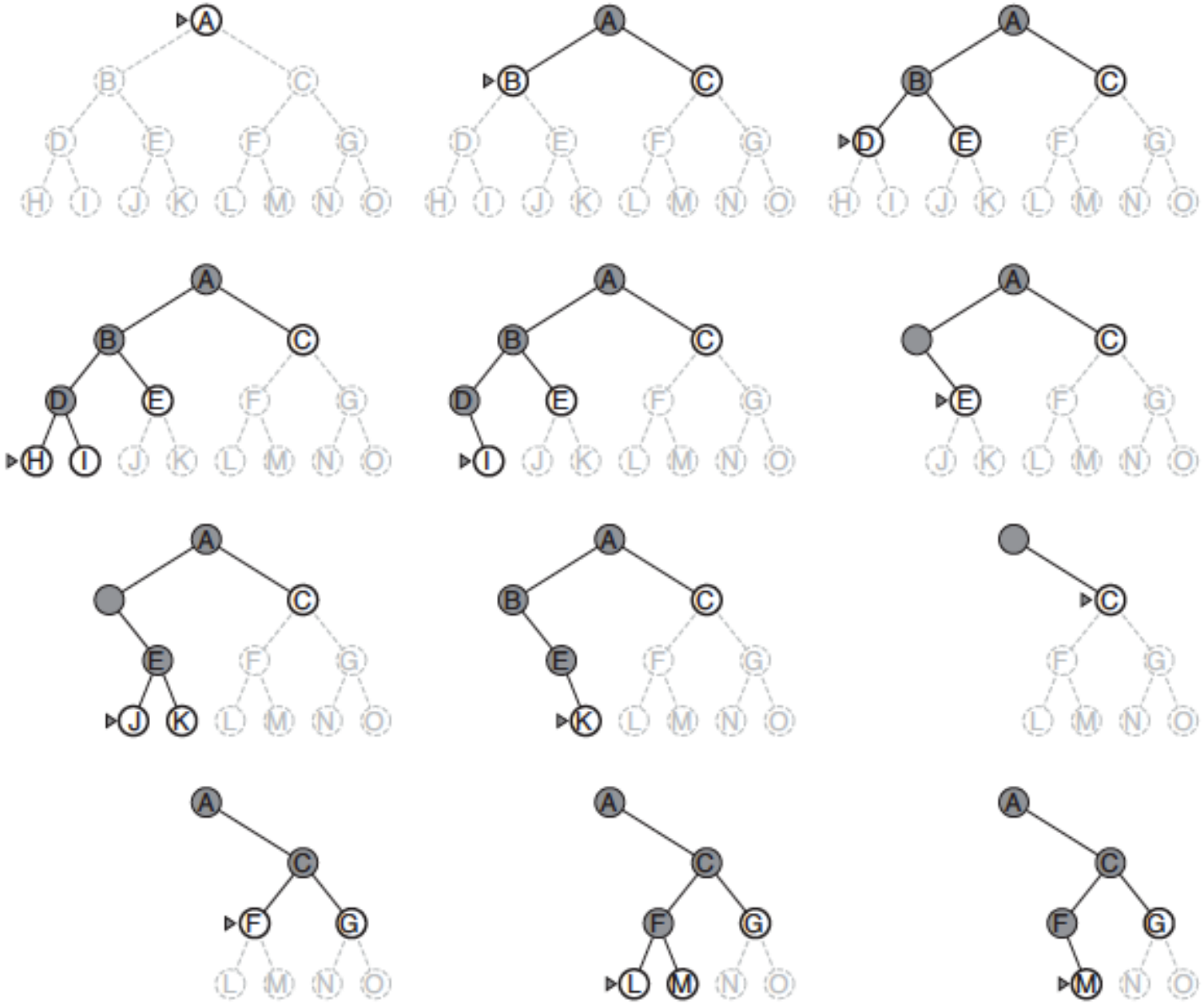
Strategie: expandează nodul curent de adâncime maximă

Frontiera este implementată cu o stivă (LIFO)



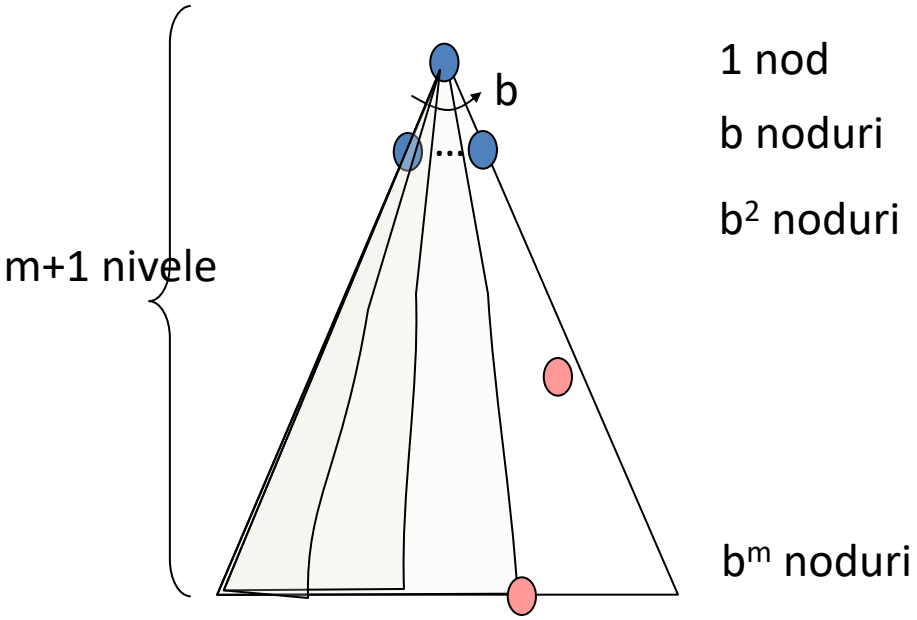


Căutare în adâncime pe un arbore binar

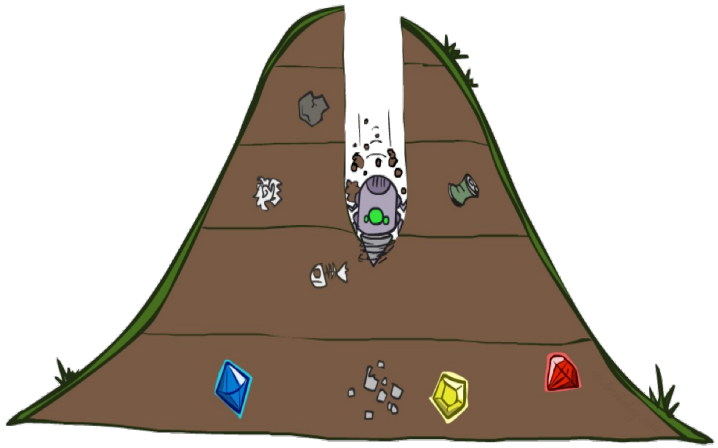


Proprietățile căutării în adâncime

- 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



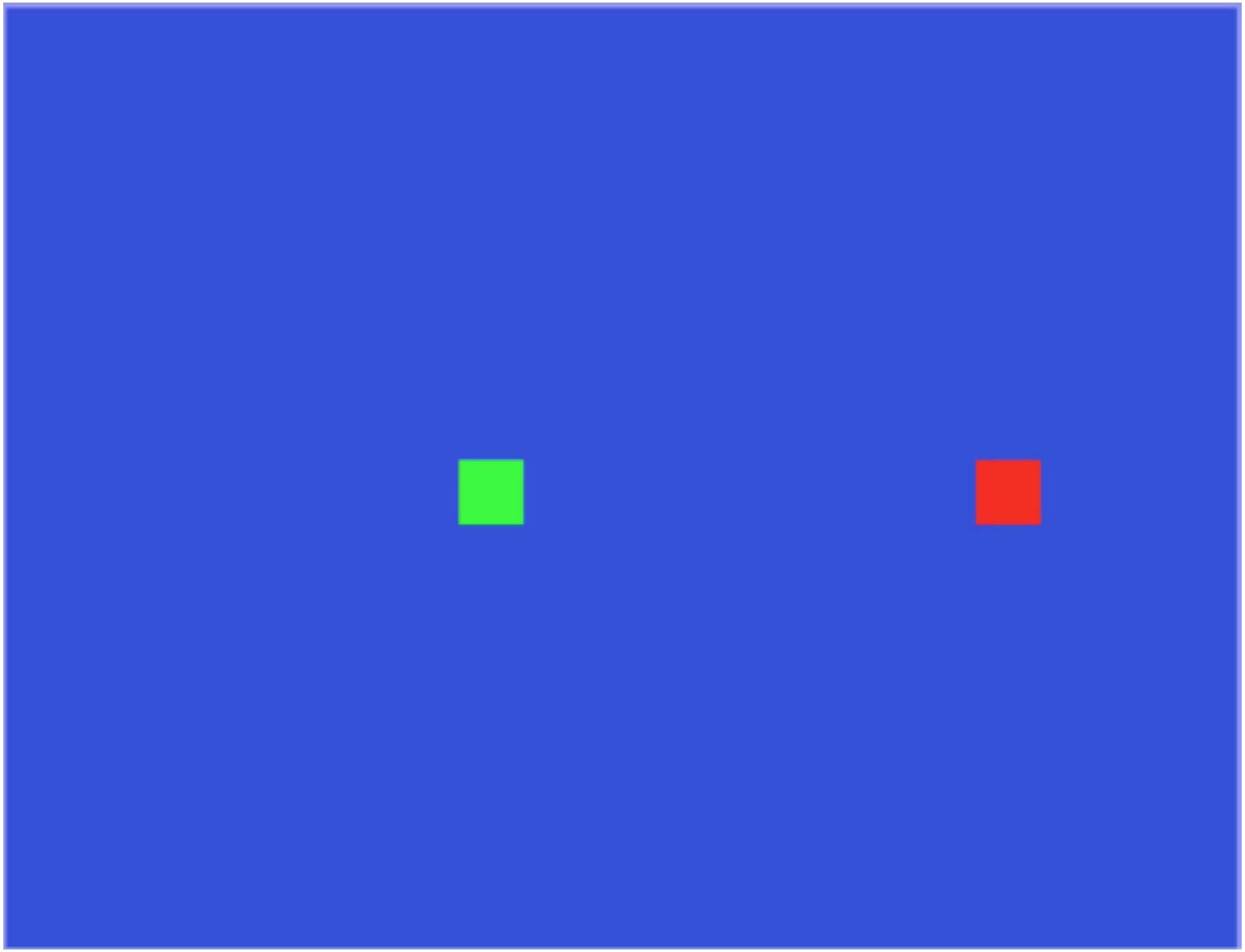
BF vs DF



- Când BF găsește o soluție mai bună decât DF?

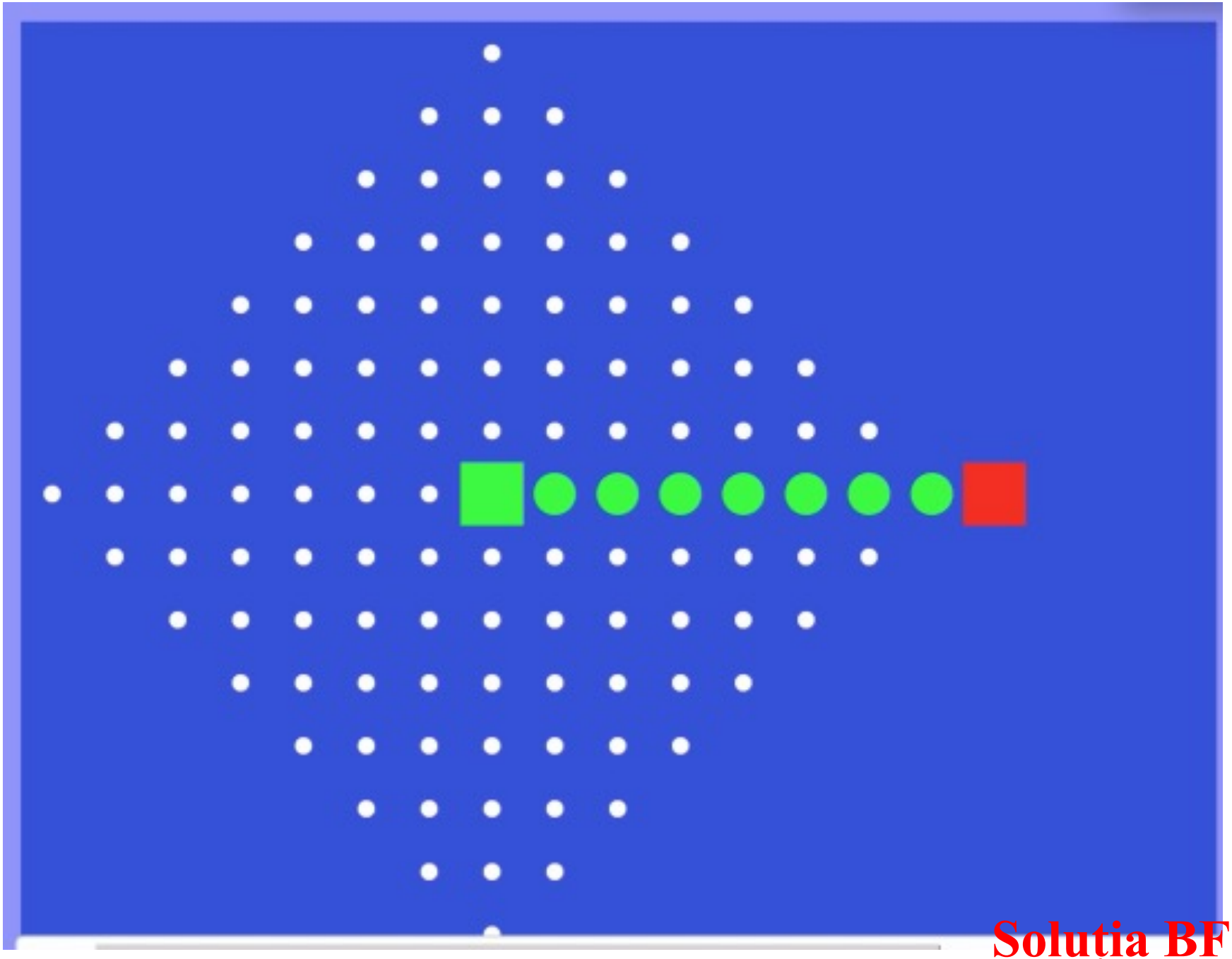
- Când DF găsește o soluție mai bună decât BF?

BF vs DF



BF vs DF

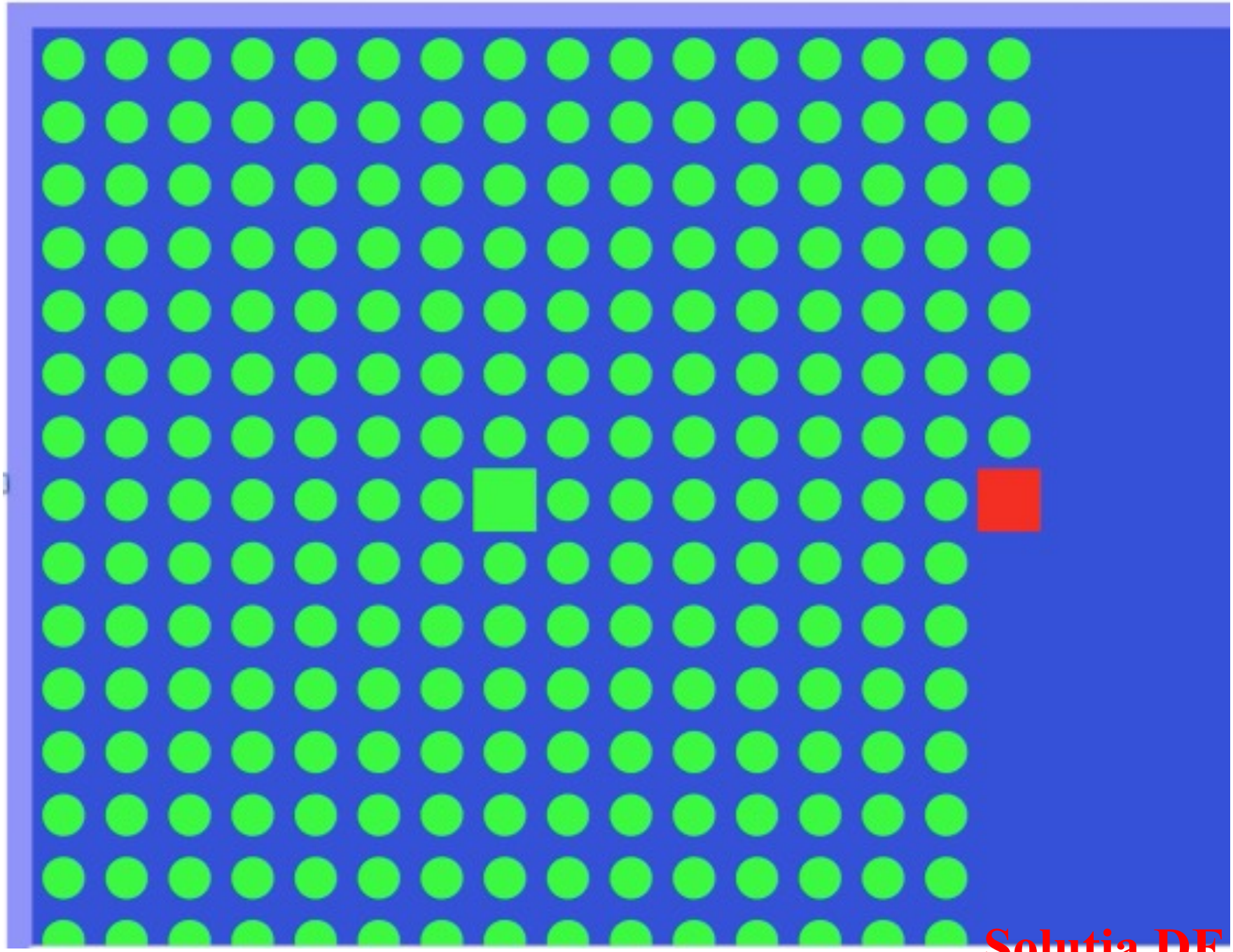
Soluția BF



Soluția BF

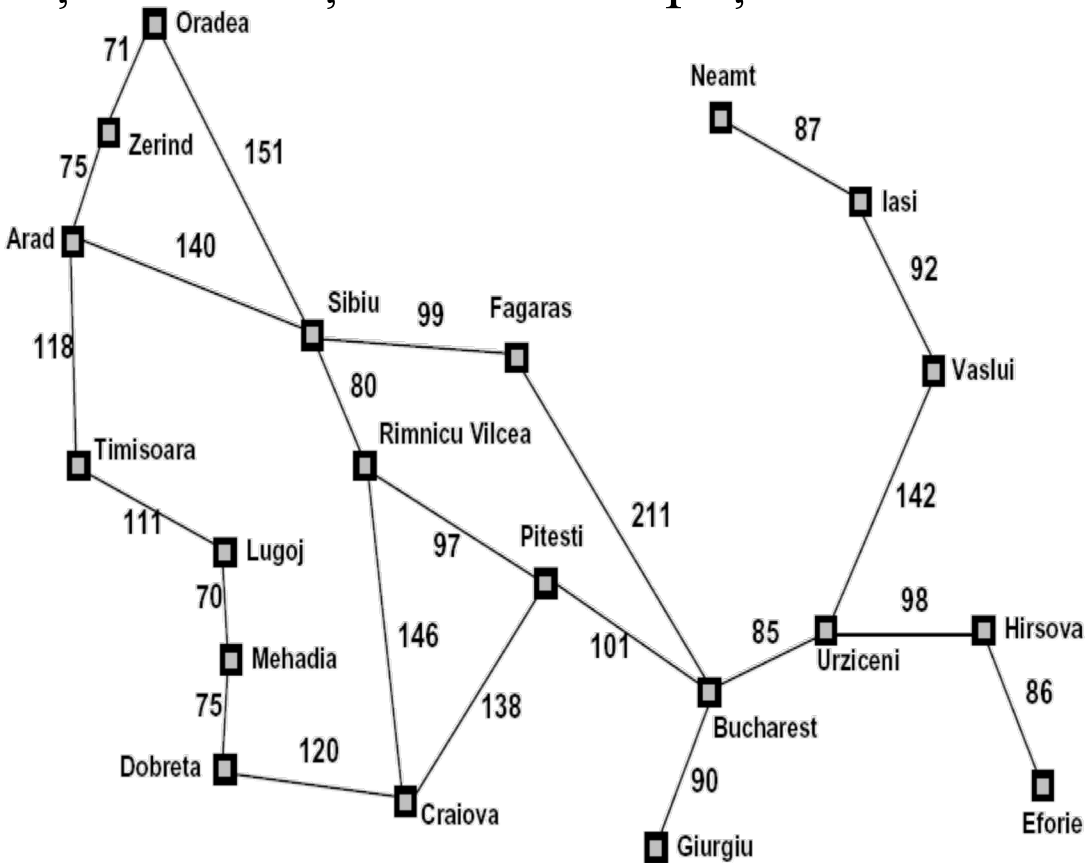
BF vs DF

Soluția DF


Soluția DF

Căutare în adâncime limitată

- căutarea în adâncime poate eșua pentru spațiu de stări infinit
- harta are 20 de orașe
- adâncimea maximă poate fi considerată 19
- pot ajunge din orice oraș în alt oraș în maxim 9 pași
- diametru = 9
- limitează DF la 9



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