

# Agenti care rezolva probleme

Catalin Stoean

[catalin.stoean@inf.ucv.ro](mailto:catalin.stoean@inf.ucv.ro)

<http://inf.ucv.ro/~cstoean>

# Agenti care rezolva probleme

---

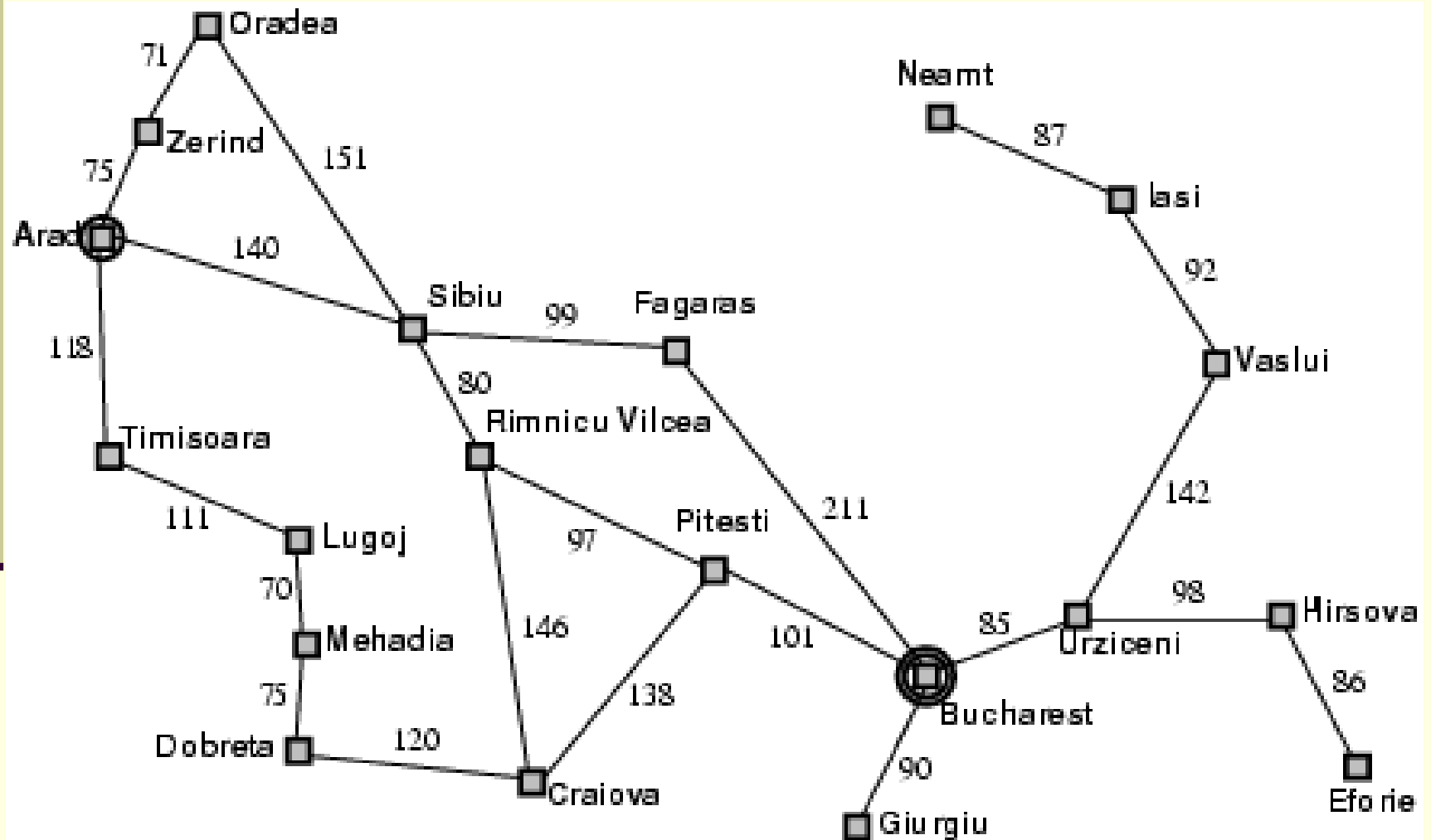
- Formularea problemelor
- Exemple de probleme
- Algoritmi de cautare standard

# Un agent *american*

---

- Vacanta in Romania – in Arad.
- In ziua urmatoare ii pleaca avionul din Bucuresti.
- **Formulara scopului:**
  - Ajungerea in Bucuresti
- **Formulara problemei:**
  - **Stari:** diverse orase
  - **Actiuni:** de a merge dintr-un oras in altul
- **Gasirea solutiei:**
  - O secventa de orase, de ex: Arad, Sibiu, Fagaras, Bucuresti.

# Un agent *american*



# Formulara problemelor

---

- **Formulara scopului** este primul lucru care trebuie stabilit de catre agent (ajungerea in Bucuresti).
- Intuitiv, **formulara problemei** este procesul de a decide ce actiuni si stari sunt considerate pentru a atinge scopul final.
- Daca *americanul* nu are nicio cunostinta aditionala despre Romania, cel mai bun lucru – alegerea unei actiuni in mod aleatoriu.
- Daca are o harta, poate examina diferite **secvente de actiuni** posibile care duc la starea finala, apoi alege cea mai buna secventa de actiuni.

# Formulara problemelor

---

- Procesul de a examina astfel de secvente de actiuni pentru alegerea celei mai bune dintre ele se numeste **cautare**.
- Un algoritm de cautare are ca intrare o problema si intoarce o **solutie** sub forma unei secvente de actiuni.
- Odata gasita solutia, se trece la faza de **executie**.
- O schema simpla pentru un agent consta in:
  - Formulare
  - Cautare
  - Executie

# Intelegerea problemei

---

- Un fermier are:
  - 20 de porci
  - 40 de vaci si
  - 60 de cai
- Cati cai are fermierul daca el considera ca si vacile sunt tot cai?

# Agenti care rezolva probleme

Perceptie

**functia** `agent_simplu_rezolvitor_de_probleme(p)` **intoarce** **actiune**

Persista la fiecare reapelare

- **s** – o secventa de actiuni initial vida
- **stare** – descriere a starii curente in care se afla lumea
- **g** – scop, initial nul
- **problema** - formulare

**stare** = `actualizeaza_stare(stare, p)`

daca **s** este vida *atunci*

**g** = `formulare_scop(stare)`

**problema** = `formulare_problema(stare, g)`

**s** = `cautare(problema)`

**actiune** = `recomandare(s, stare)`

**s** = `rest(s, stare)`

**intoarce** **actiune**



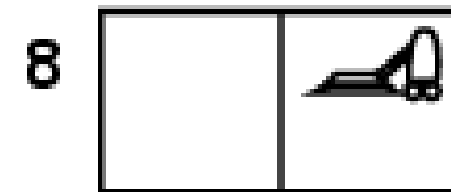
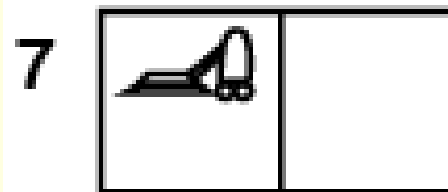
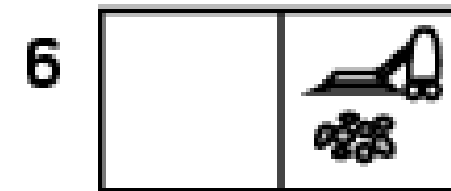
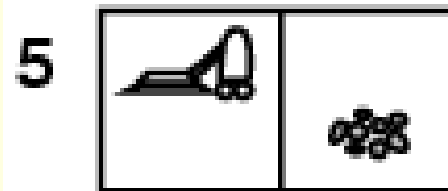
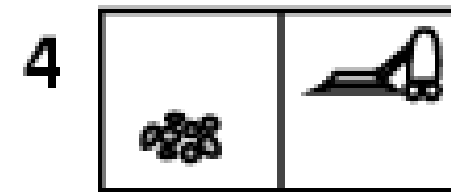
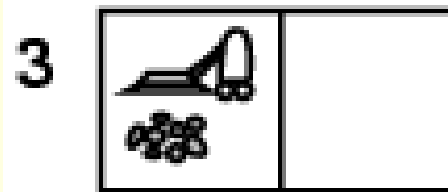
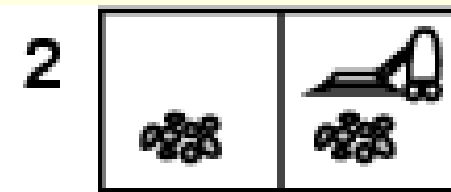
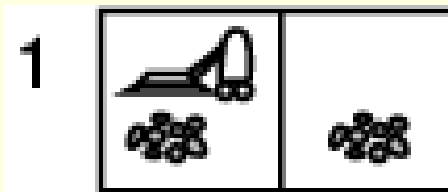
# Formulara problemelor

---

- Sunt patru tipuri de probleme:
  - **Cu o singura stare** (deterministe, observabile complet)
    - Agentul stie exact in ce stare se va gasi; solutia este o secventa;
  - **Cu mai multe stari** (neobservabil)
    - Agentul nu stie in ce stare se gaseste;
  - **Contingente** (nedeterminist, partial observabil)
    - Perceptorii aduc informatie noua despre starea curenta
  - **Explorative** (spatiul starilor necunoscut)

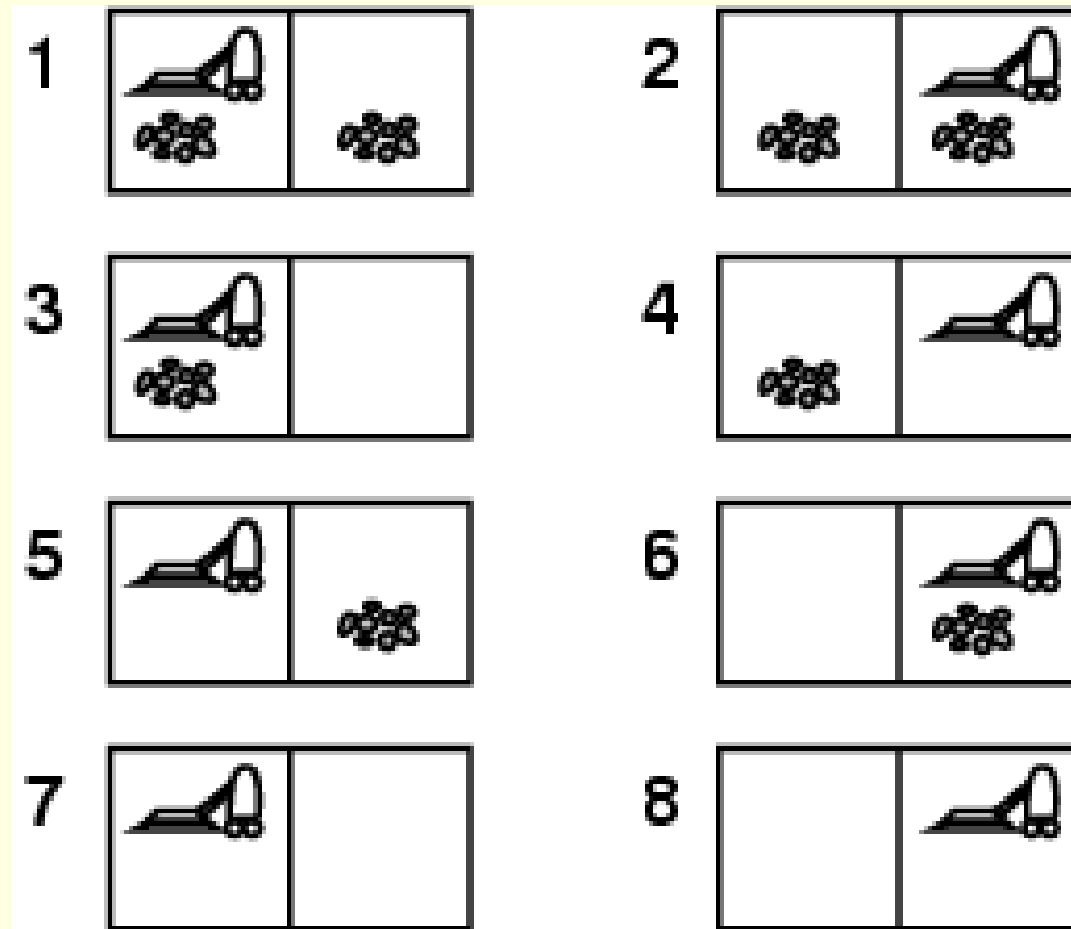
# Agentul aspirator

- Avem 2 locatii.
- In fiecare locatie
  - poate fi sau nu mizerie
  - poate sa fie aspiratorul sau nu
- Cele 8 stari posibile →
- Actiuni: *stanga*, *dreapta*, *aspira*.
- Scop: curatarea ambelor incaperi (starile 7 sau 8)



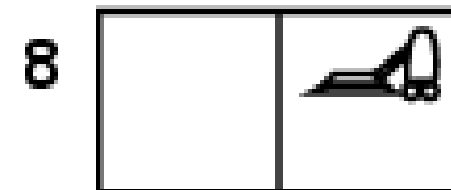
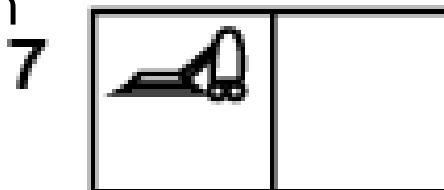
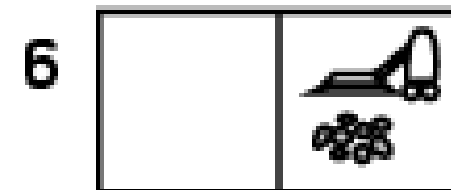
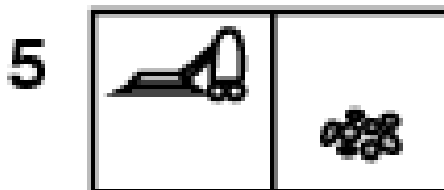
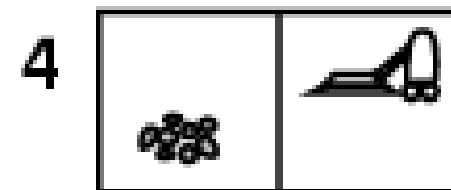
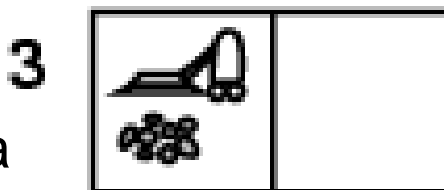
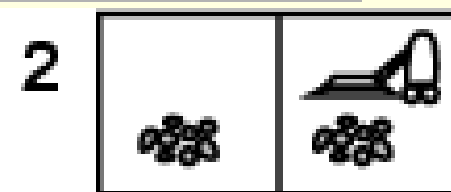
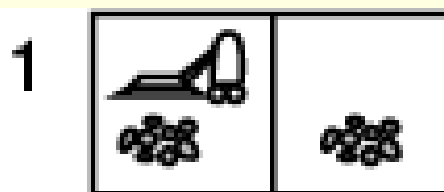
# Problema cu o singura stare (exemplu)

- Senzorii agentului ii spun exact starea in care se gaseste.
- Agentul stie exact ce efecte au actiunile sale.
- Exemplu: daca se gaseste in starea 5 si urmeaza secventa de actiuni [*Dreapta*, *Aspira*], se ajunge la atingerea scopului problemei.



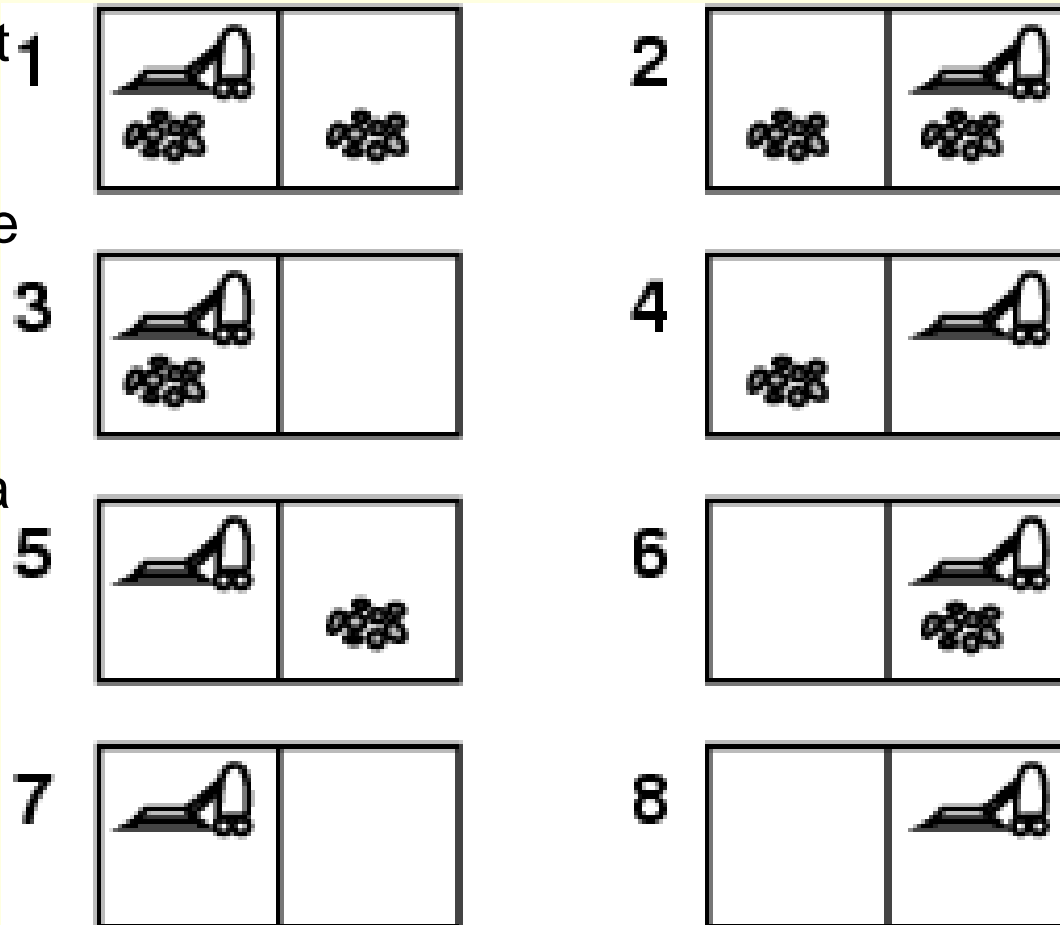
# Problema cu mai multe stari (exemplu)

- Agentul stie exact ce efecte au actiunile sale dar...
- Senzorii agentului au acces limitat fata de starea in care se gaseste.
- Poate sa nu aiba senzori deloc – stie doar ca in starea initiala se gaseste in una din starile {1, 2, ..., 8}
- Poate ajunge sa *rezolve* problema?...



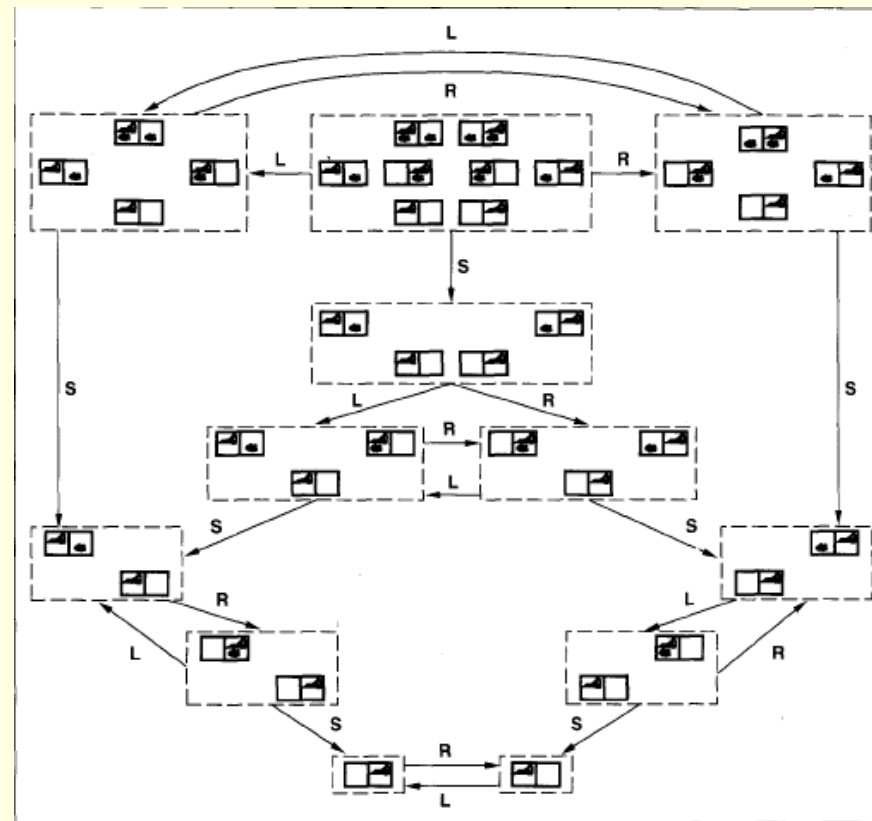
# Problema cu mai multe stari (exemplu)

- **DA**, pentru ca stie ce efect au actiunile lui.
- Actiunea *Dreapta* il va face sa se gaseasca in una din situatiile {2, 4, 6, 8}.
- Ce efect va avea secventa de actiuni: [*Dreapta*, *Aspira*, *Stanga*, *Aspira*]?...
- Agentul trebuie sa rationeze in raport cu *multimi de stari* la care poate ajunge.



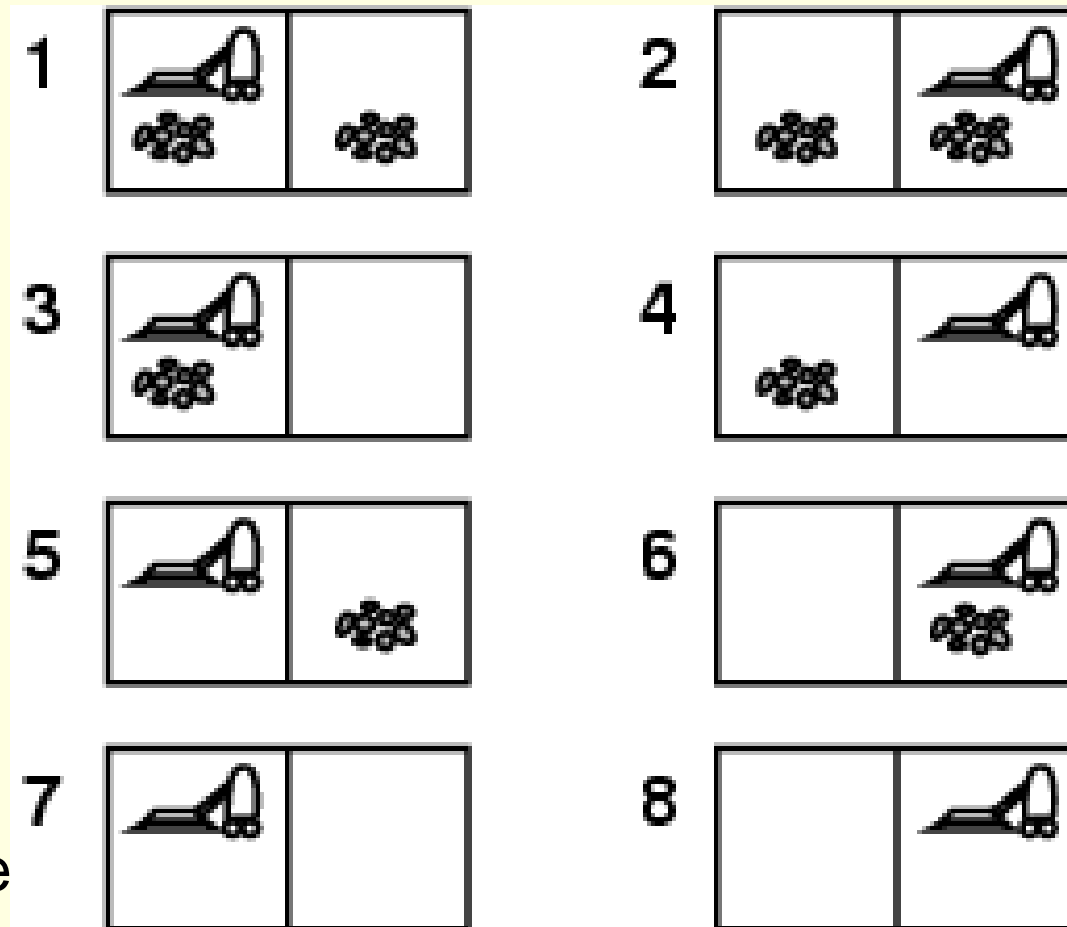
# Problema cu mai multe stări (exemplu)

- Când mediul nu este complet accesibil, agentul lucrează cu mulțimi de stări în care poate ajunge, nu cu câte o singură stare.



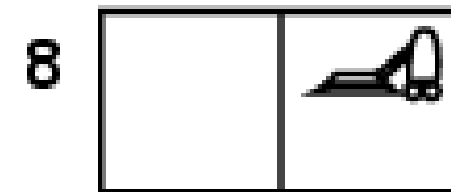
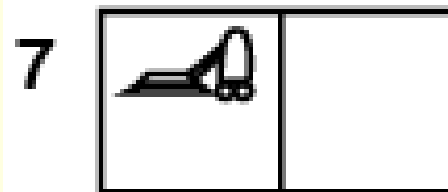
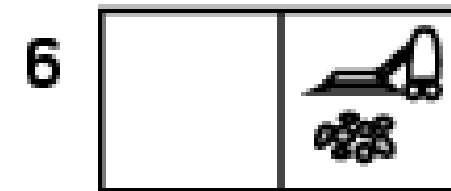
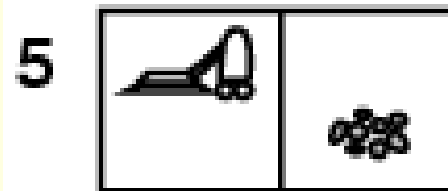
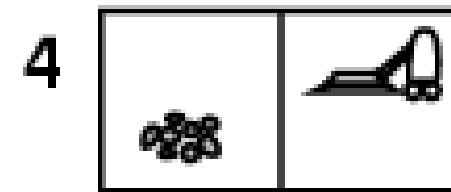
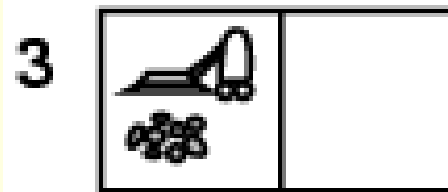
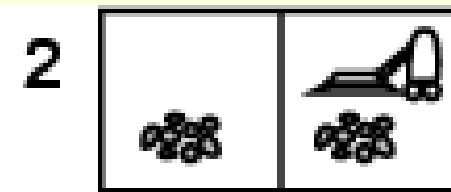
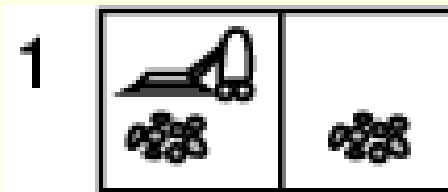
# Problema contingenta (exemplu)

- Presupunem ca mediul este nedeterminist.
- Legile lui Murphy guverneaza mediul
  - aspirarea duce la depozitarea murdariei intr-un loc... care era complet curat...
- De exemplu, in starea 4, daca aspira se poate ajunge la 2 sau 4.



# Problema contingenta (exemplu)

- Nu exista o secventa de actiuni unica ce reprezinta solutia problemei.
- Este nevoie de utilizare a senzorialor in cursul fazei de executie:
  - *Aspira numai daca este mizerie in casuta curenta.*





# Probleme explorative

---

- Agentul trebuie sa experimenteze, sa descopere gradual care sunt efectele actiunilor lui si ce tipuri de stari exista.
- Exemplu: agentul american in Arad, fara o harta sau fara alte cunostinte despre Romania.
- Cautarea se aplica si in cazul acestor probleme, dar mediul in care apar problemele explorative este lumea reala.

# Formulara problemelor

- O problema se defineste prin patru puncte:
  1. **Starea initiala** in care se afla agentul (de exemplu, Arad).
  2. **Actiuni** sau **functia succesor**  $S(x)$  – fiind data o stare  $x$ ,  $S(x)$  intoarce multimile de stari in care se poate ajunge din  $x$  printr-o singura actiune ( $S(\text{Arad}) = \{\text{Zerind}, \text{Sibiu}, \text{Timisoara}\}$ )
  3. **Testarea tintei problemei** – se verifica daca starea curenta a atins tinta problemei ( $x = \text{Bucuresti}$ ,  $\text{sah\_mat}(x)$ )
  4. Functia de **cost al drumului** –
    - calculeaza un cost  $g$  pentru drumul curent (suma distantelor, numarul actiunilor executate etc).
    - $c(x, y)$  – costul pasului, presupus sa fie  $\geq 0$
- O solutie este o secventa de actiuni care merg de la starea initiala la starea tinta

# Formulara problemelor

---

- Lumea reala este foarte complexa => spatiul starilor trebuie sa fie abstractizat pentru rezolvarea de probleme.
- (Abstract) Stare = multime de stari reale.
- (Abstract) Actiune = combinatie complexa de actiuni reale
  - Ex: de la Arad la Sibiu: drumuri bifurcate, stopuri etc.
- Fiecare actiune abstracta ar trebui sa fie mai "usoara" decat actiunea/actiunile in problema originala.

# Exemple de probleme

---

- Probleme tip joc
  - Ilustreaza diverse metode de rezolvare de probleme
- Probleme din viata reala
  - Sunt mai dificile
  - Sunt mult mai de interes

# Puzzle cu 8 valori

3	8	4
5	1	7
6		2

Starea initiala

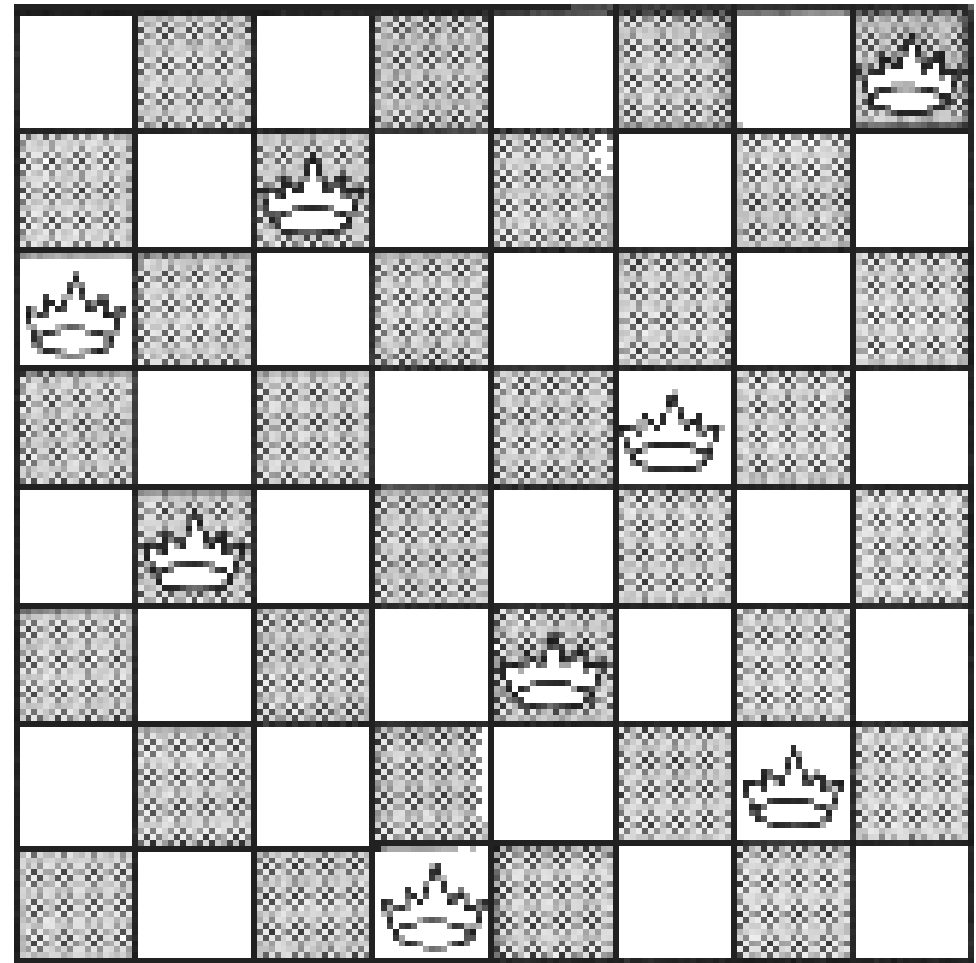
1	2	3
4	5	6
7	8	

Starea tinta

- **Stari:** este descrisa locatia fiecarei cifre in una din cele 9 casute.
- **Actiuni:** casuta goala se misca la stanga, dreapta, sus sau jos.
- **Testarea tintei:** starea se gaseste in configuratia din dreapta.
- **Costul drumului:** fiecare pas are costul 1, deci costul drumului este dat de numarul de mutari.

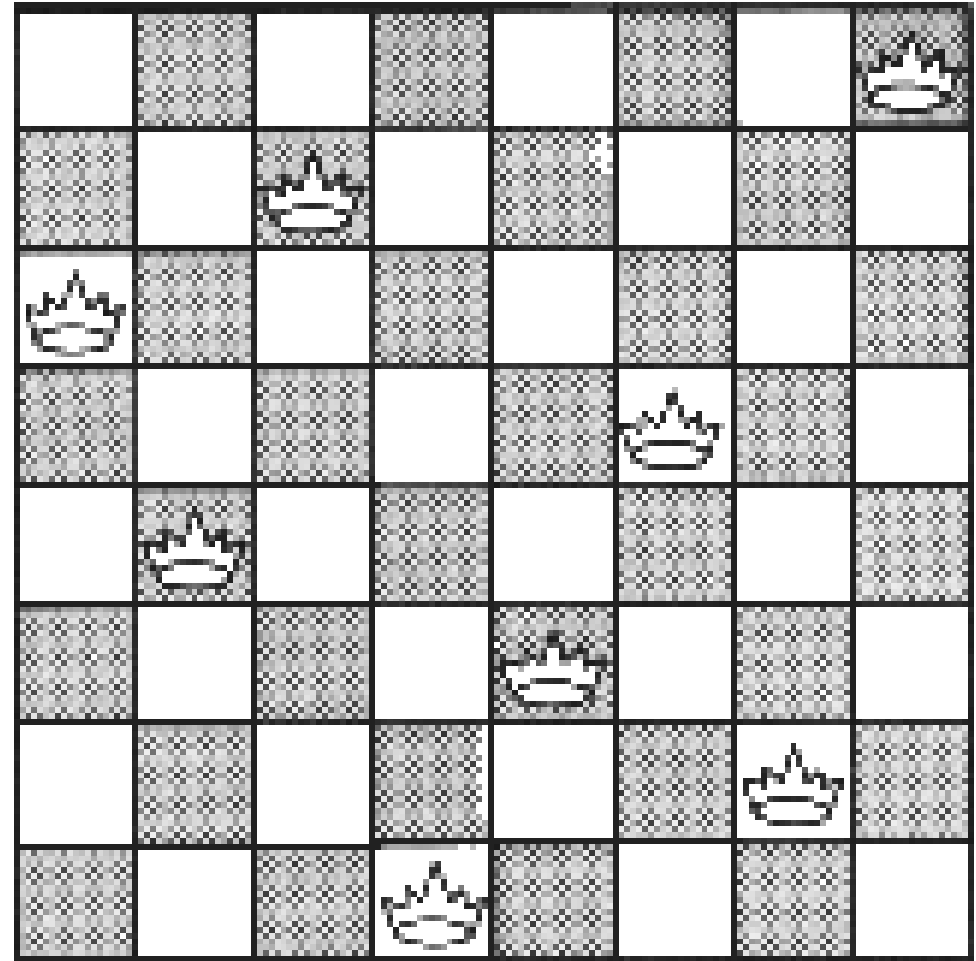
# Problema celor 8 dame

- **Stari:** orice aranjament de 0 până la 8 dame care nu se ataca.
- **Actiuni:** adauga o dama la orice patrat.
- **Testarea tintei:** 8 dame care nu se ataca pe tabla.
- **Costul drumului:** 0.
- $64^8$  posibilitati...



# Problema celor 8 dame (alte acțiuni)

- **Stari:** orice aranjament de 0 până la 8 dame care nu se atacă.
- **Acțiuni:** adăuga o damă pe coloana cea mai din stânga a.i. să nu fie atacată de alta damă.
- **Testarea țintei:** 8 dame care nu se atacă pe tablă.
- **Costul drumului:** 0.
- 2057 posibilități



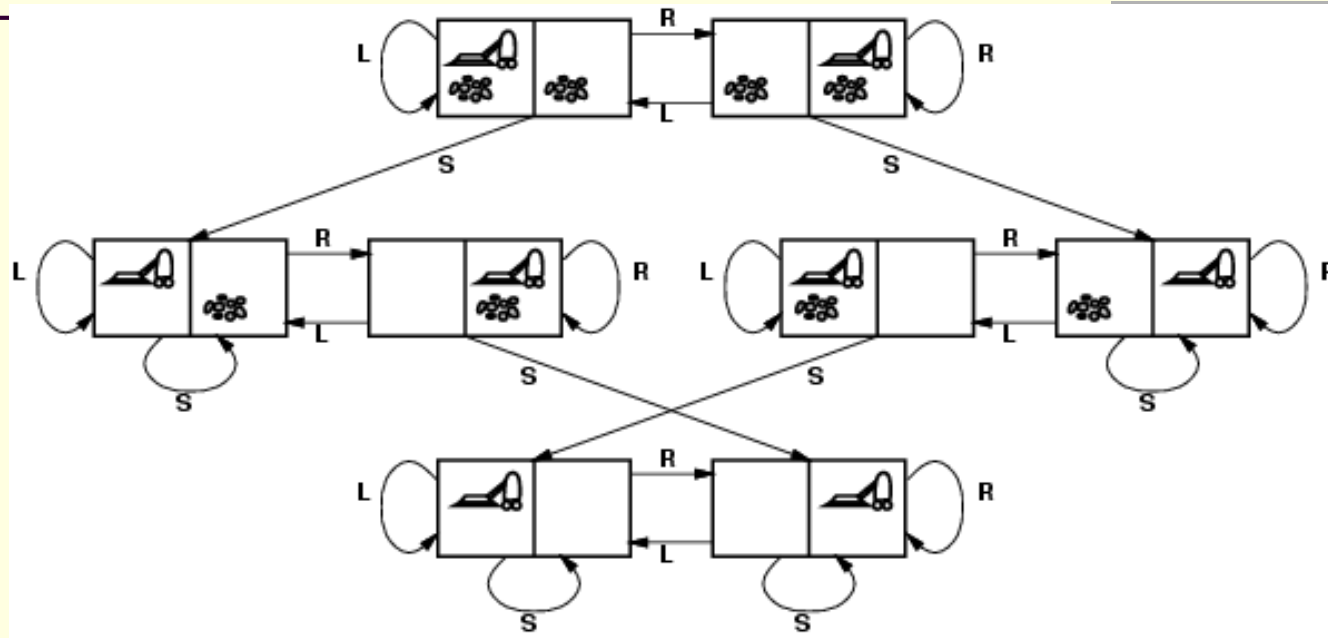
# Criptaritmetica

FORTY+	Solutie:	29786	F = 2, O = 9, R = 7 etc
TEN		850	
TEN		850	
-----		-----	
SIXTY		31486	

- **Stari:** un puzzle criptaritmatic cu litere inlocuite de cifre.
- **Actiuni:** inlocuieste toate aparitiile unei litere cu o cifra care nu apare deja in puzzle.
- **Testarea tintei:** puzzle-ul contine numai cifre iar suma este corecta.
- **Costul drumului:**0. Toate solutiile sunt egale ca insemnatate.



# Aspiratorul cu o singură stare



- **Stari:** una din cele 8 stari.
- **Actiuni:** mutare stanga, mutare dreapta, aspirare.
- **Testarea tintei:** toate locatiile sunt curate.
- **Costul drumului:** fiecare actiune consta 1.

# Aspiratorul cu stari multiple

---

- Agentul nu are senzori, insa tot trebuie sa curete toate locatiile.
  - **Stari:** submultimi din cele 8 stari.
  - **Actiuni:** mutare stanga, mutare dreapta, aspirare.
  - **Testarea tintei:** toate locatiile sunt curate.
  - **Costul drumului:** fiecare actiune consta 1.
- 
- Multimea de stari initiale consta din toate cele 8 situatii pentru ca agentul nu are senzori.

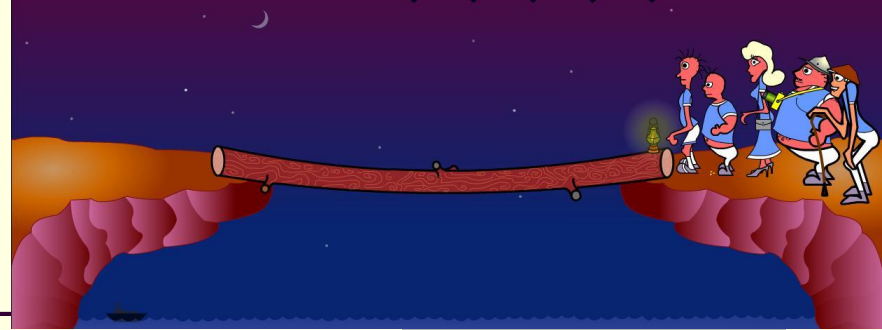


# Misionarii și canibalii



- Trei misionari și trei canibali se afla de o parte a raului. Ei au o barcă ce poate duce cel mult doi oameni. Găsiți o posibilitate să traverseze toți de cealaltă parte a raului cu condiția să nu existe mai mulți canibali decât misionari într-o parte.
- <https://www.learn4good.com/kids-games/puzzle/boat.htm>
- <http://www.novelgames.com/en/missionaries/>
- **Stari:** secvențe ordonate de 3 numere reprezentând numărul de misionari, canibali și bărci de partea în care se aflau inițial (3, 3, 1).
- **Actiuni:** mutarea unui misionar sau canibal sau 2 canibali, 2 misionari sau un misionar și un canibal de pe o parte pe alta.
- **Testarea țintei:** starea (0, 0, 0).
- **Costul drumului:** numărul de traversări.

# Trecerea unui pod cu lampa - tema



- Noapte, exista o singura lampa, 5 persoane trebuie sa treaca de pe un mal pe celalalt pe un pod care poate sustine doar doua persoane.
- Fiecare persoana trece podul cu o viteza diferita: 1 sec, 3 sec, 6 sec, 8 sec, 12 sec.
- Cand trec doua persoane, se deplaseaza ambele cu viteza celei care merge mai incet.
- **Lampa tine doar 30 sec!**
  - <http://www.inwebson.com/demo/cross-the-bridge/>

# Alte platforme pentru algoritmi din IA



**Robocode**

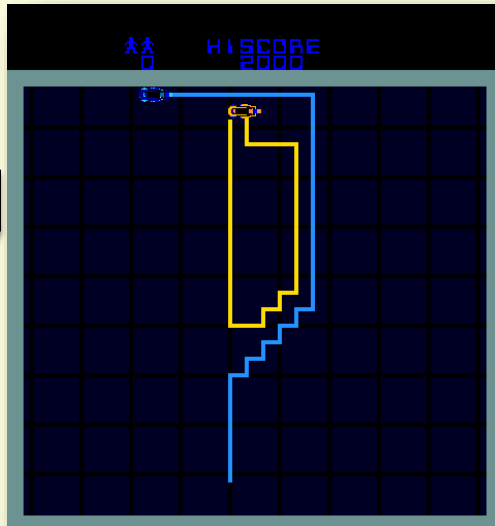
**Sudoku**

5	3		7					
6			1	9	5			
	9	8					6	
8			6					3
4			8	3				1
7			2					6
	6					2	8	
			4	1	9			5
			8				7	9



**Mario**

**Tron**



**Torcs**

# Si mai bine: intelegeti dezvoltarea platformei

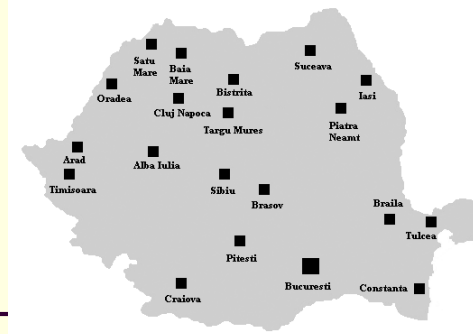
Let's make 16 classic games in C++/SFML!





# Probleme din viata reala

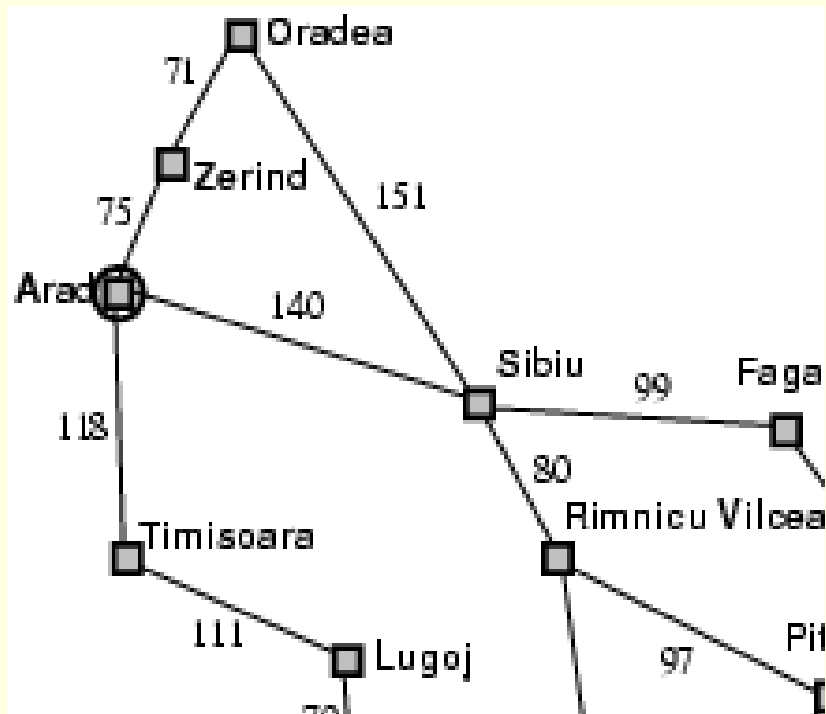
---



- Algoritmi de gasire de rute:
  - Rutarea in retele de calculatoare
  - Sisteme de planificare pentru transportul aerian
- Problema comis-voiajorului
  - Sa se gaseasca cel mai scurt tur astfel incat sa se viziteze fiecare oras exact o data plecand si terminand din/in acelasi oras.
  - Spre deosebire de problemele cu gasire de rute, aici trebuie retinute orasele *vizitate*.



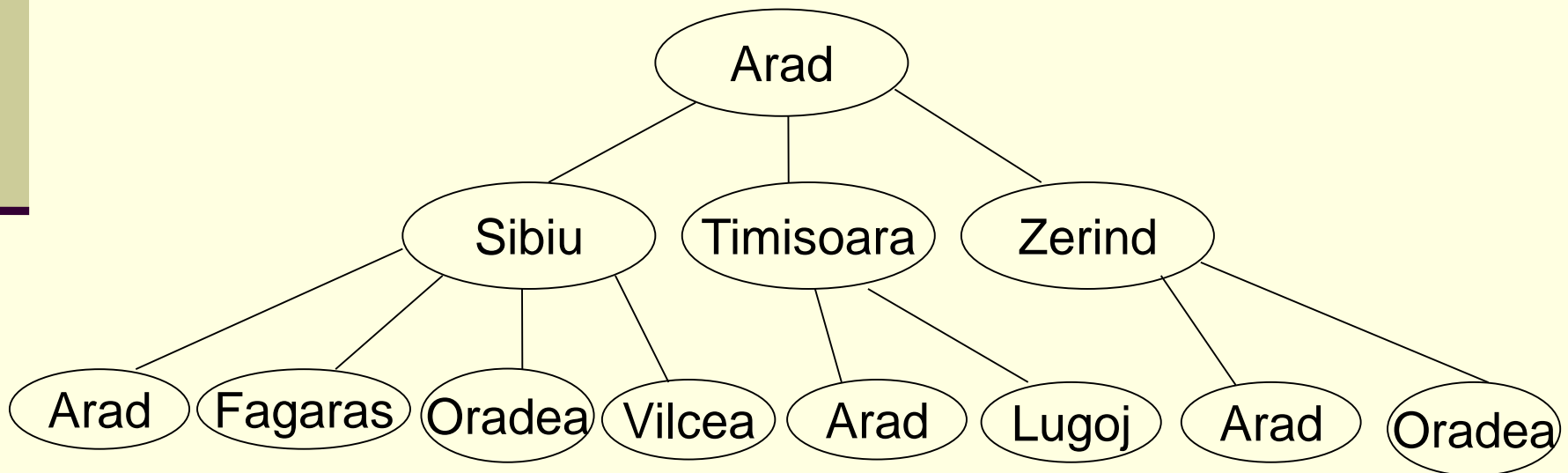
# Algoritmi de cautare standard



- Pornim din Arad.
- Posibilitati?
- Zerind, Sibiu, Timisoara...
- Pe care il alegem?
- Depinde de **strategia de cautare** folosita.

# Algoritmi de cautare standard

- Procesul de cautare poate fi construit sub forma unui arbore de cautare.
- Radacina arborelui de cautare este un nod de cautare care corespunde cu starea initiala.



# Algoritm general de cautare

**functia** *cautare\_generala*(*problema*, *strategie*)

**intoarce** *solutie* sau *esec*

Initializeaza arborele de cautare folosind starea initiala a problemei.

*Cat timp* *solutie* negasita si *noduri*  $\neq \emptyset$  *executa*

*Daca* nu mai sunt posibilitati de incercat *atunci*

**intoarce** *esec*

Alege un nod posibil in concordanta cu strategia

*Daca* nodul contine o stare tinta *atunci*

**intoarce** *solutia* corespunzatoare

*Altfel* gaseste toate posibilitatile ce pornesc din acest nod si  
adauga-le la arborele de cautare

*Sfarsit cat timp*

# Componentele unui nod

---

- Starea din spatiul starilor careia ii corespunde nodul;
- Nodul parinte (nodul din arborele de cautare care a generat nodul curent);
- Actiunea care a fost aplicata pentru a se ajunge la nodul curent;
- Adancimea nodului - numarul de noduri prin care s-a trecut de la nodul radacina pana la nodul curent;
- Costul drumului de la starea initiala pana la nodul curent.

# Algoritmi de cautare standard

---

- Se face distincție între noduri și stări:
  - Starea reprezintă o configurație a mediului;
  - Nodul conține informații cu privire la structura arborelui de căutare.
- Colectia de noduri este implementată sub forma de **lista**.  
Operații posibile:
  - `genereaza_lista(Elemente)` – creează o listă cu elementele date
  - `goala(lista)` – întoarce “adevărat” dacă este goală listă
  - `scoate_din_fata(lista)` – întoarce elementul din față
  - `adauga(Elemente, lista)` – introduce “Elemente”-le în “listă”

# Algoritm general de cautare

**functia** cautare\_generala(*problema*) **intoarce** solutie sau **esec**  
noduri = genereaza\_lista(genereaza\_nod(stare\_initala[problema]))

*Cat timp* solutie negasita si noduri  $\neq \emptyset$  *executa*

nod = scoate\_din\_fata(noduri)

*Daca* testare\_tinta[problema] se aplica la stare(nod) *atunci*  
*solutie gasita*

*Altfel*

noduri = adauga(noduri, expandare(nod, actiuni[problema]))

*Sfarsit cat timp*

# Algoritmi de cautare standard

---

- Mai avem in vedere la algoritmul anterior:
  - Ce noduri au fost vizitate (pentru a evita ciclurile)
  - Pentru fiecare nod retinem nodul parinte
    - Pentru a intoarce la final solutia plecand de la destinatie inapoi catre start.
- Daca strategia o cere:
  - Adancimea nodului curent (radacina este la adancimea 0)
  - Costul de la nodul de start pana la cel curent
    - Cosul nodului de start este 0, apoi la nodul curent insumam costul nodului parinte cu costul dintre nodul parinte si cel curent.

# Algoritmi de cautare standard

---

- Strategiile de cautare sunt analizate in functie de 4 criterii:
  - Completitudine
    - Garanteaza strategia gasirea unei solutii atunci cand exista una?
  - Complexitate temporala
    - Cat dureaza gasirea unei solutii?
  - Complexitatea spatiala
    - De cata memorie este nevoie pentru a face cautarea?
  - Optimalitate
    - Gaseste strategia solutia cea mai buna calitativ cand exista mai multe solutii diferite?



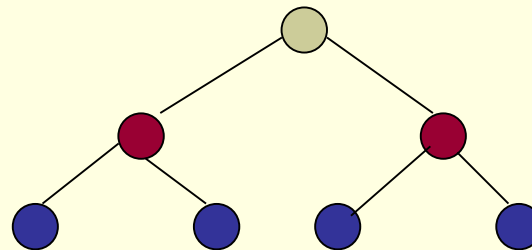
# Algoritmi de cautare standard

---

- Algoritmii studiatii in acest curs (cel de azi) folosesc doar **cautarea neinformata** (blind search).
  - Nu exista informatii despre numarul de pasi sau despre costul drumului de la starea curenta pana la starea tinta.
  - Pot doar distinge o stare tinta de o stare care nu este tinta.
  - (Agentul american) Orice cale din Arad are aceeasi importanta.
- **Cautarea informata** (heuristic search) presupune utilizarea de informatii aditionale (data viitoare).
  - (Agentul american) Se poate folosi informatia ca trebuie sa mearga spre sud-est...

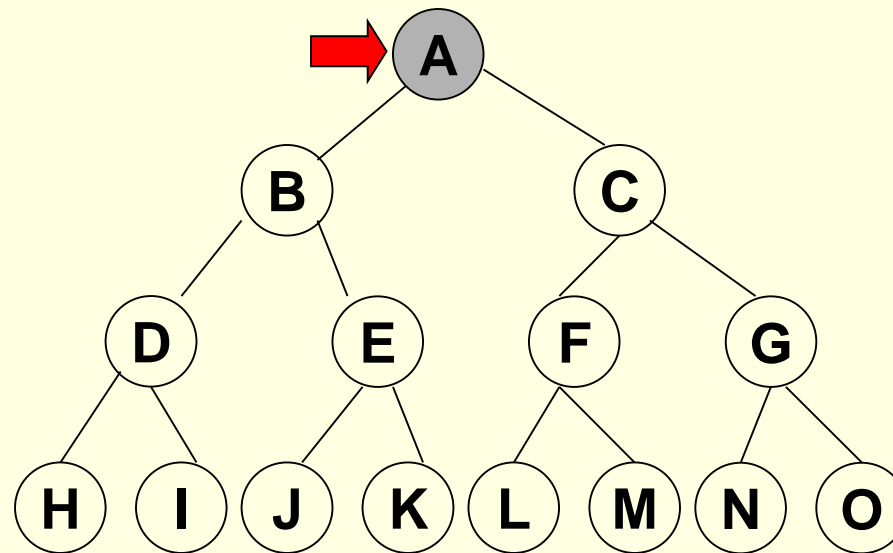
# Cautarea in latime

- Nodul radacina este expandat intai, apoi toate celelalte noduri sunt expandate, apoi toti succesorii lor s.a.m.d.
- Sunt considerate toate drumurile de lungime 1, apoi toate drumurile de lungime 2 etc.

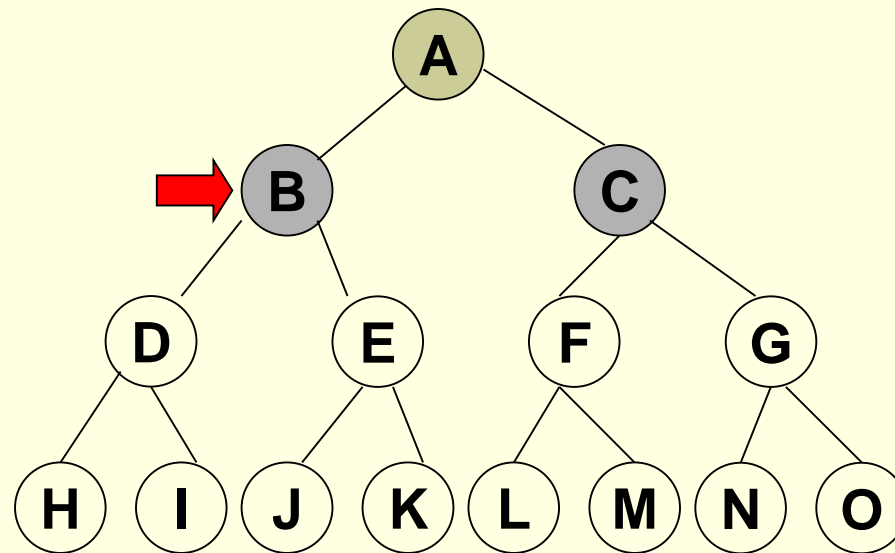


- Daca exista o solutie, algoritmul garanteaza ca o va gasi, iar daca exista mai multe solutii, algoritmul va gasi intai starea tinta aflata la cel mai mic numar de noduri.

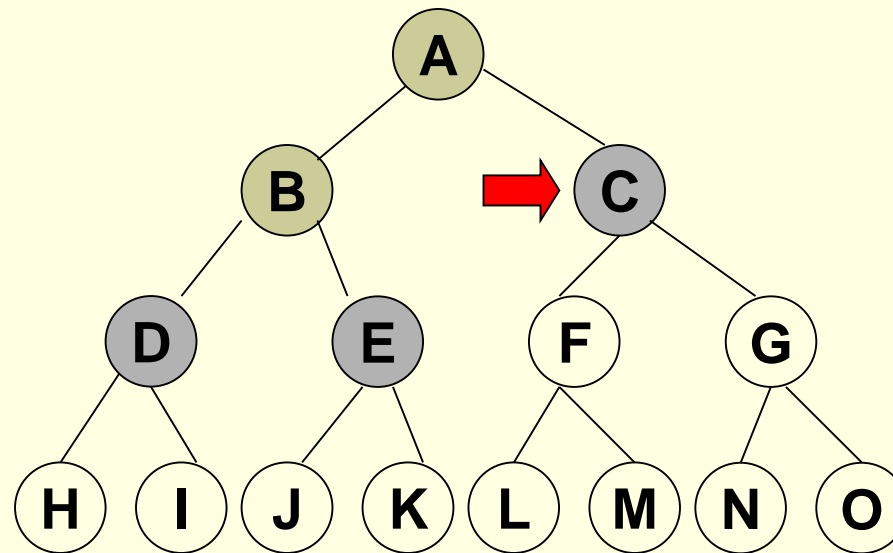
# Cautarea in latime



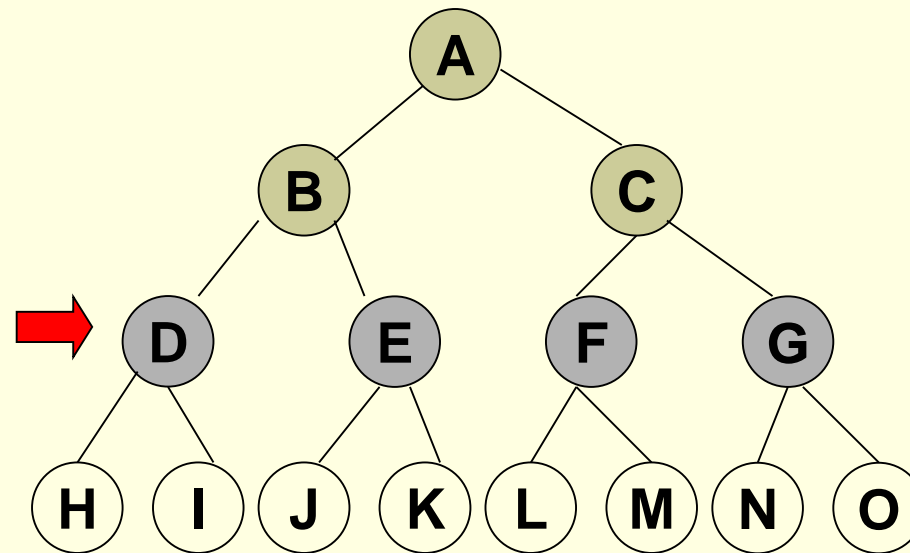
# Cautarea in latime



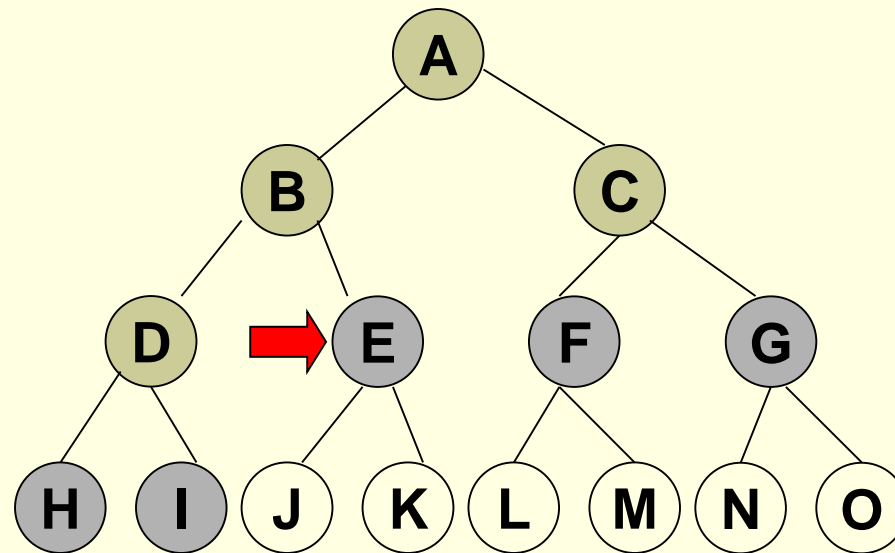
# Cautarea in latime



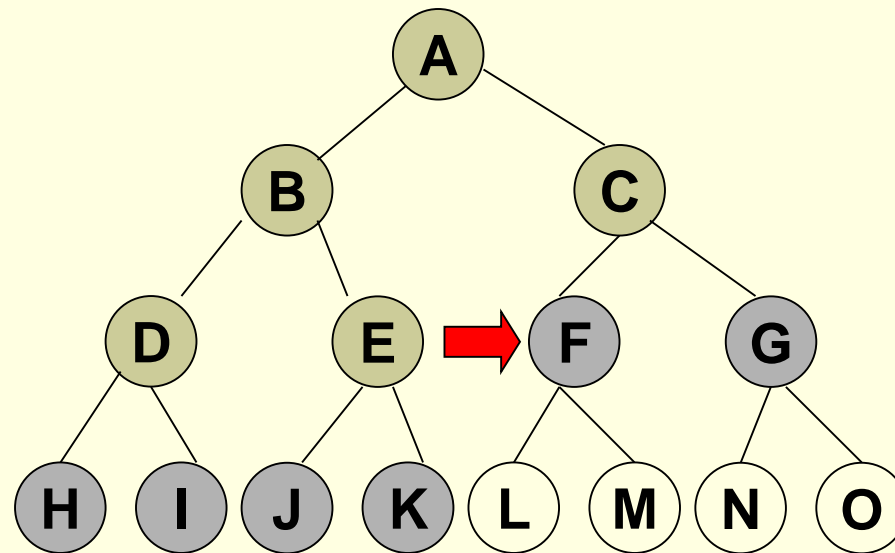
# Cautarea in latime



# Cautarea in latime

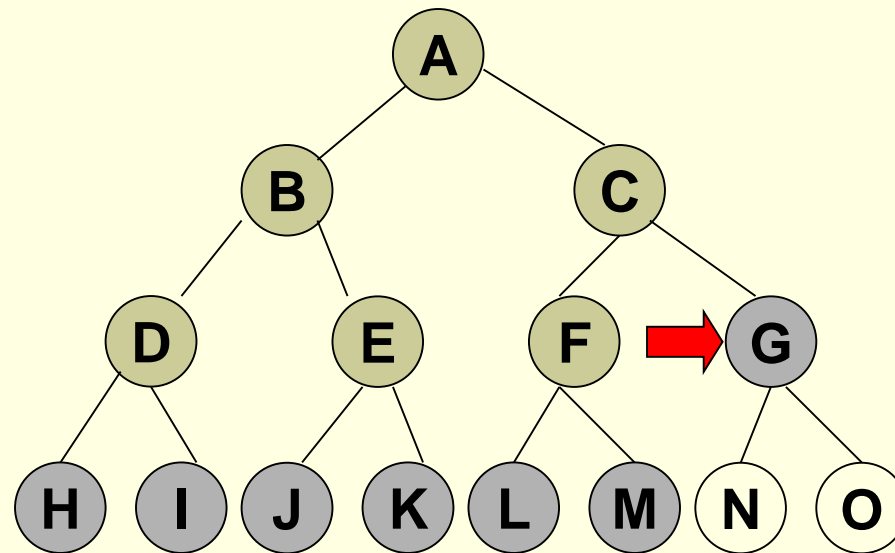


# Cautarea in latime

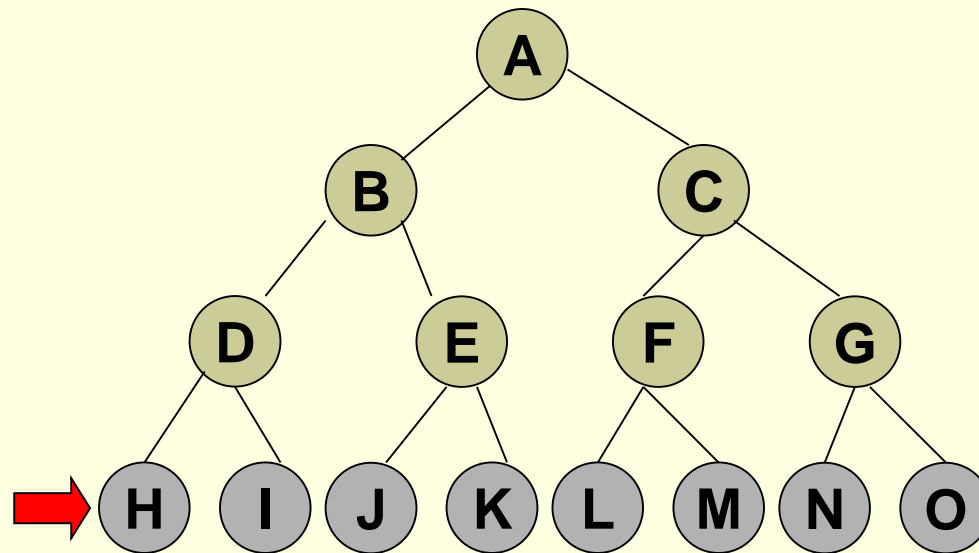




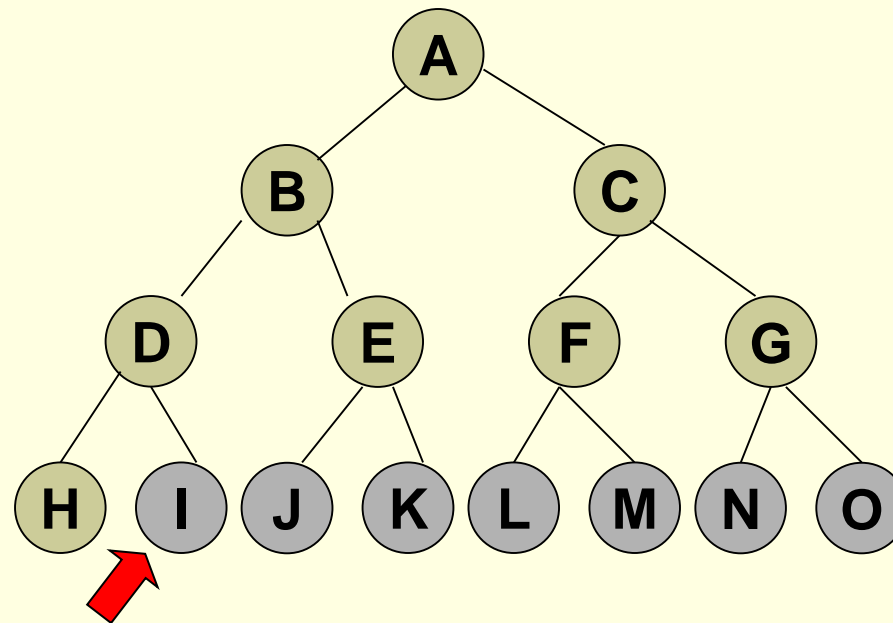
# Cautarea in latime



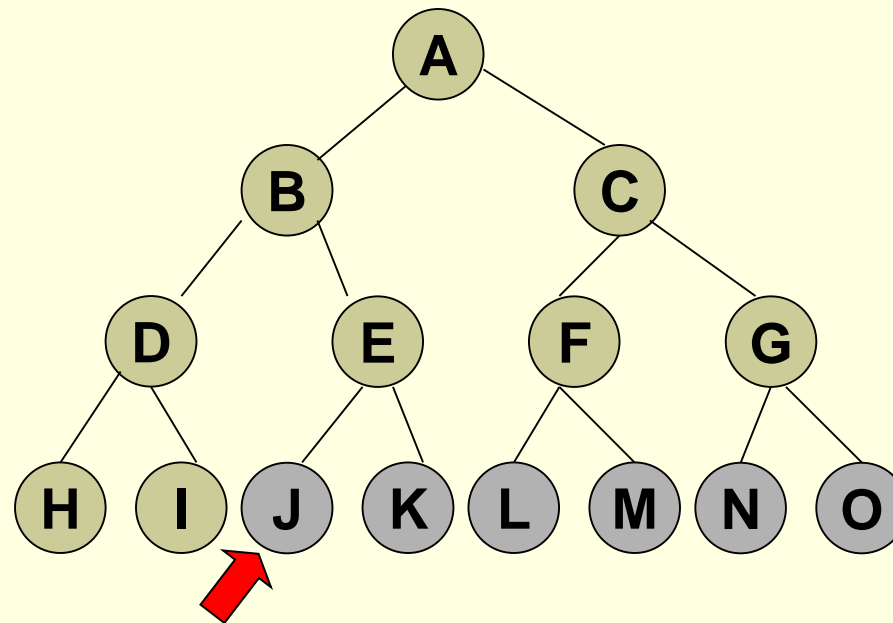
# Cautarea in latime



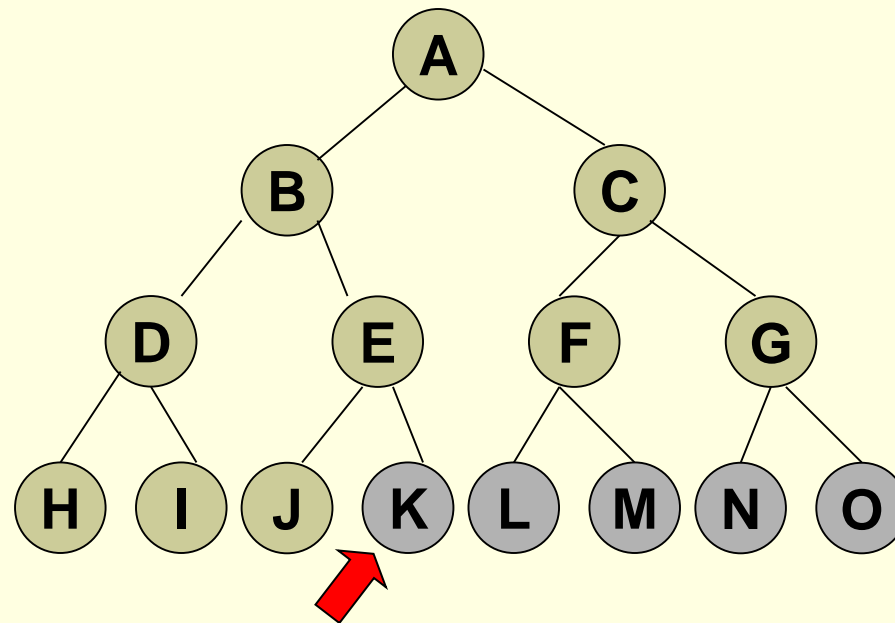
# Cautarea in latime



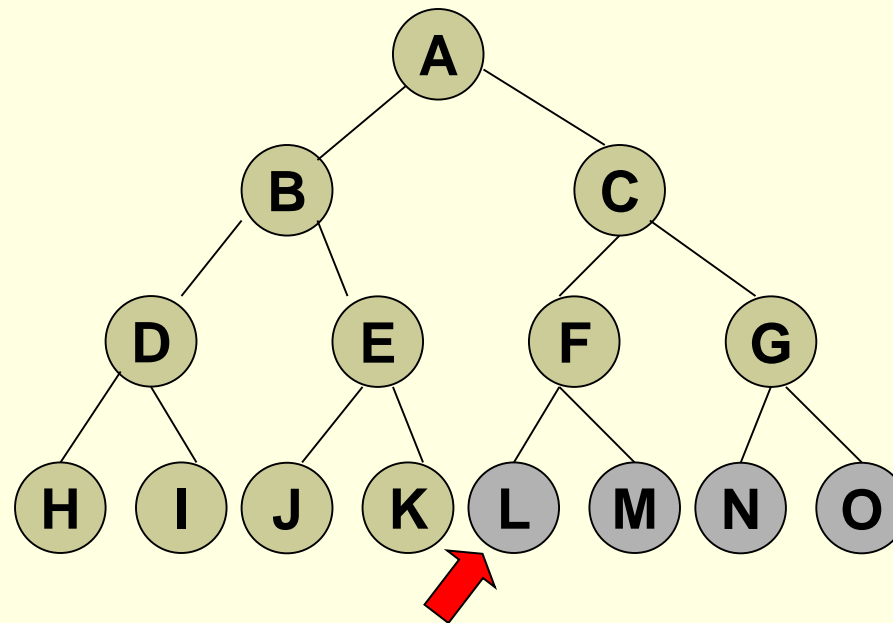
# Cautarea in latime



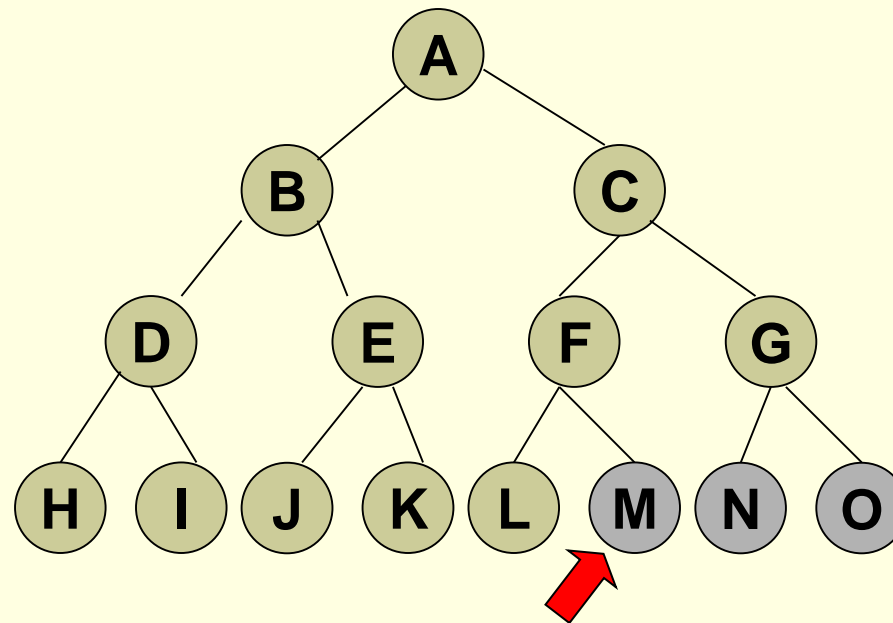
# Cautarea in latime



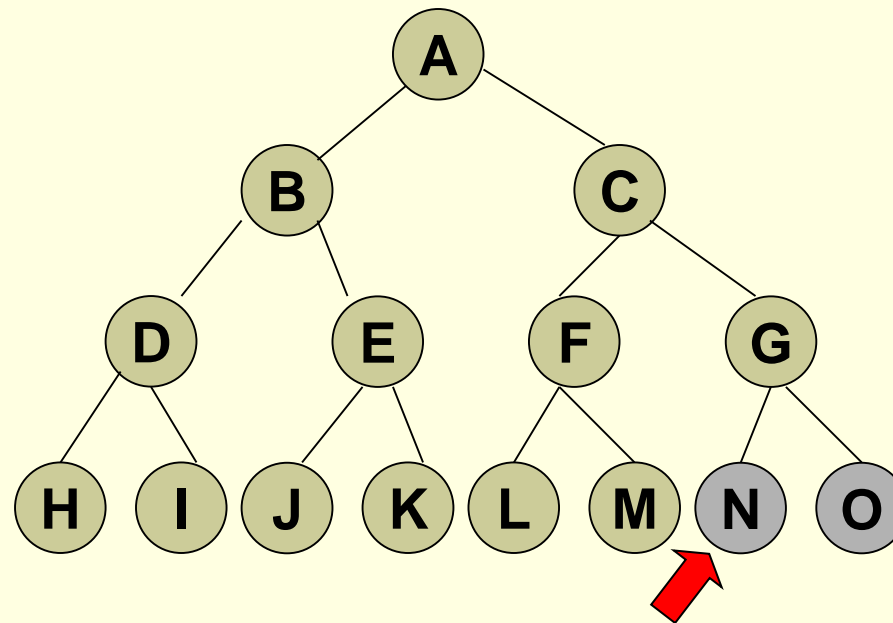
# Cautarea in latime



# Cautarea in latime

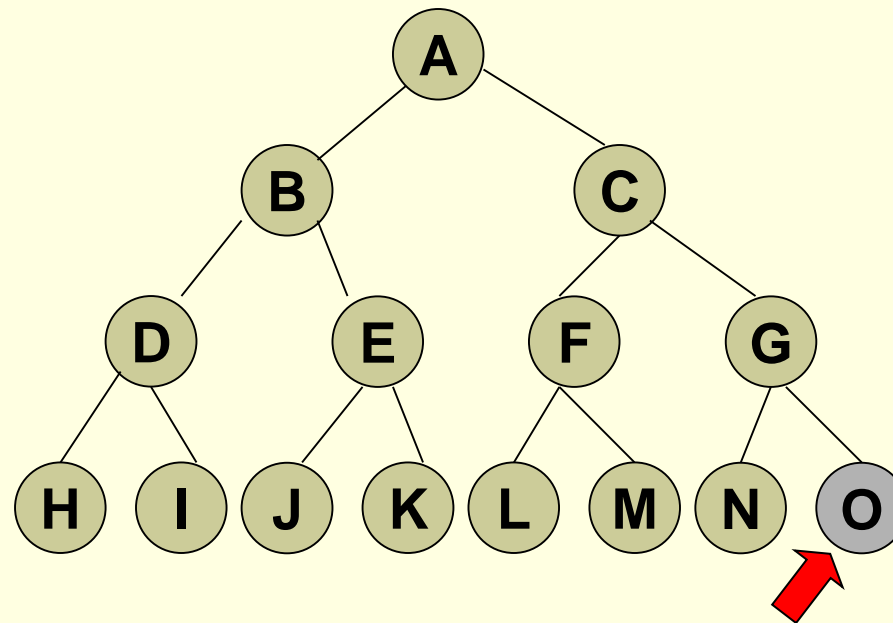


# Cautarea in latime





# Cautarea in latime



# Algoritm cautare in latime

**functia** cautare\_latime(problema) **intoarce** solutie sau esec

noduri = genereaza\_lista(genereaza\_nod(stare\_initala[problema]))

*Cat timp* solutie negasita si noduri  $\neq \emptyset$  *executa*

nod = scoate\_din\_fata(noduri)

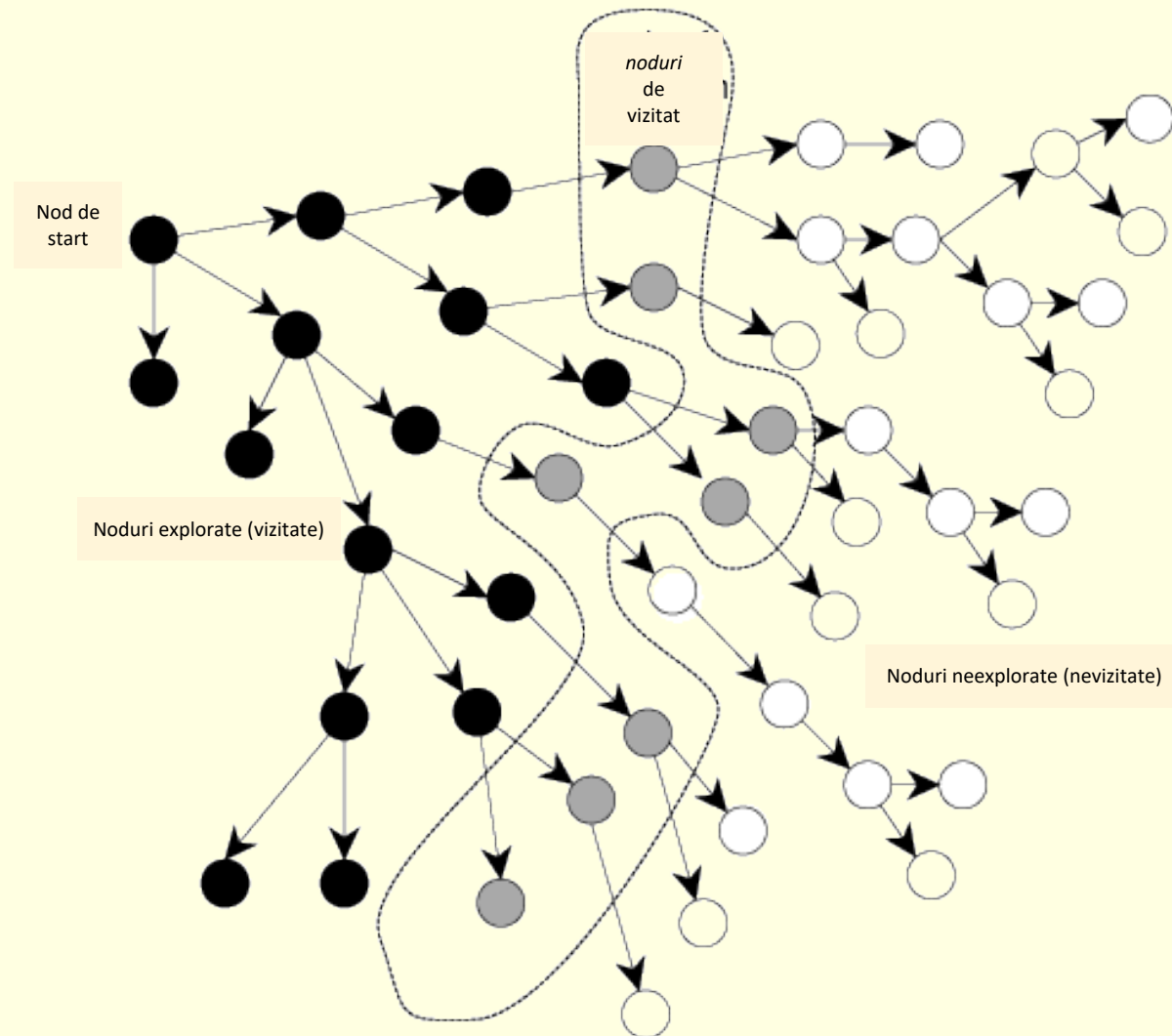
*Daca* testare\_tinta[problema] se aplica la stare(nod) *atunci*  
*solutie gasita*

*Altfel*

noduri = adauga(noduri, expandare(nod, adauga\_la\_sfarsit))

*Sfarsit cat timp*

# Lista *noduri* contine nodurile ce urmeaza sa fie vizitate (din marcaj)

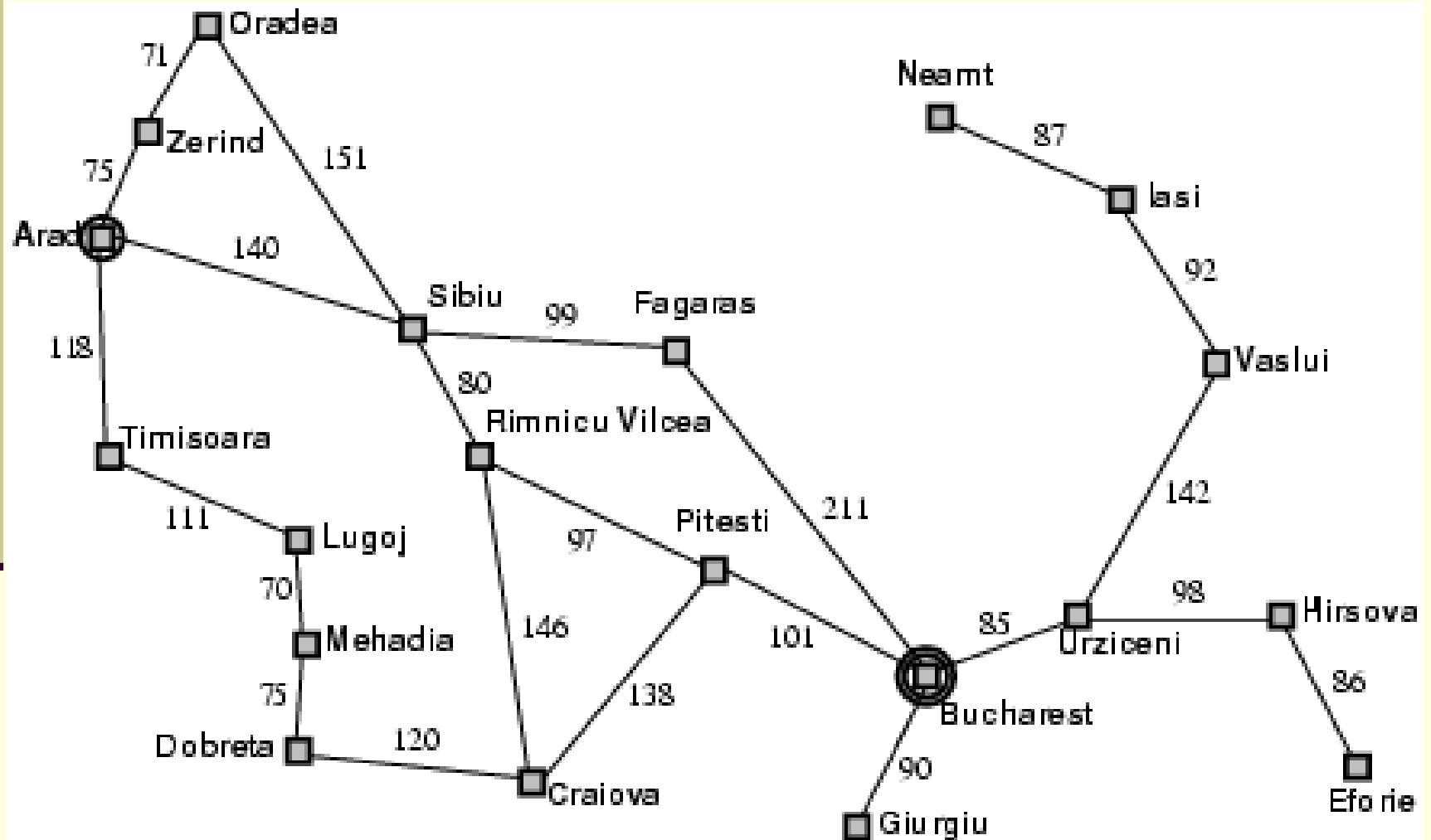


# Problema de rutare: un agent *american*

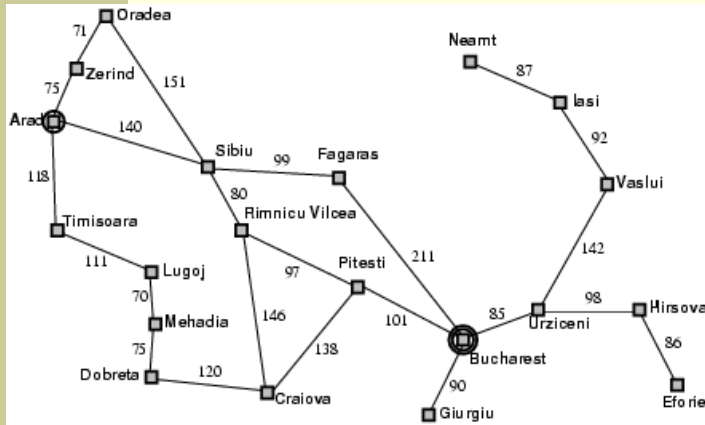
---

- Vacanta in Romania – in Arad.
- In ziua urmatoare ii pleaca avionul din Bucuresti.
- **Formulara scopului:**
  - Ajungerea in Bucuresti
- **Formulara problemei:**
  - **Stari:** diverse orase
  - **Actiuni:** de a merge dintr-un oras in altul
- **Gasirea solutiei:**
  - O secventa de orase, de ex: Arad, Sibiu, Fagaras, Bucuresti.

# Un agent *american*



# Algoritm cautare in latime



**functia** *cautare\_latime*(problema) **intoarce** solutie sau esec  
 noduri = genereaza\_coada(genereaza\_nod(stare\_initala[problema]))

*Cat timp* solutie negasita si noduri  $\neq \emptyset$  *executa*

nod = scoate\_din\_fata(noduri)

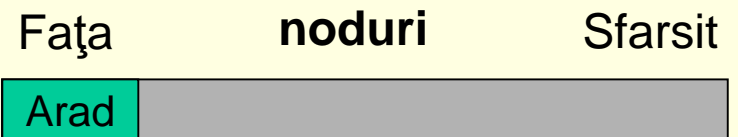
*Daca* testare\_tinta[problema] se aplica la stare(nod) *atunci*  
 solutie gasita

*Altfel*

noduri = adauga(noduri, expandare(nod, adauga\_la\_sfarsit))

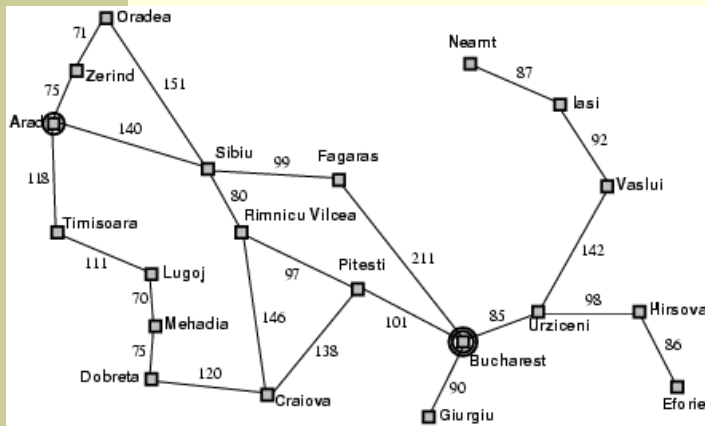
*Sfarsit cat timp*

Arad



Parcurgerea: Arad,

# Algoritm cautare in latime



**functia** `cautare_latime(problema)` **intoarce** `solutie` sau `esec`  
`noduri = genereaza_coada(genereaza_nod(stare_initiala[problema]))`

*Cat timp* `solutie` negasita si `noduri`  $\neq \emptyset$  *executa*

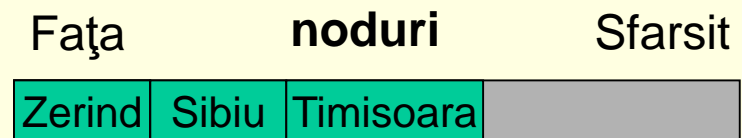
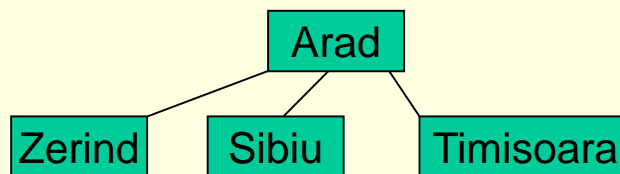
`nod = scoate_din_fata(noduri)`

*Daca* `testare_tinta[problema]` se aplica la `stare(nod)` *atunci*  
`solutie gasita`

*Altfel*

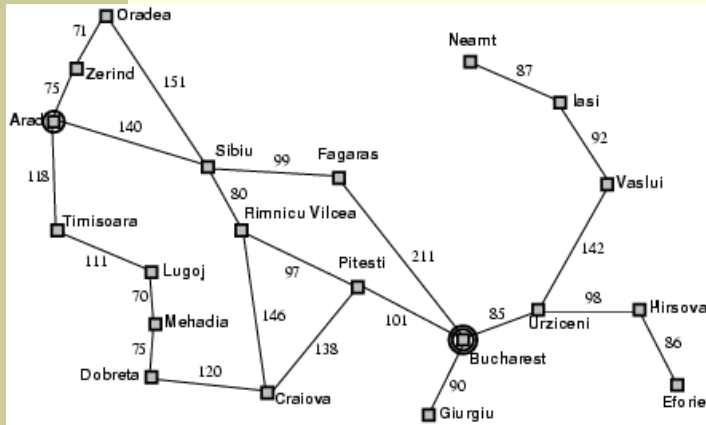
`noduri = adauga(noduri, expandare(nod, adauga_la_sfarsit))`

*Sfarsit cat timp*



Parcurgerea: Arad, Zerind

# Algoritm cautare in latime



**functia** *cautare\_latime*(problema) **intoarce** solutie sau esec  
 noduri = genereaza\_coada(genereaza\_nod(stare\_initiala[problema]))

*Cat timp* solutie negasita si noduri  $\neq \emptyset$  *executa*

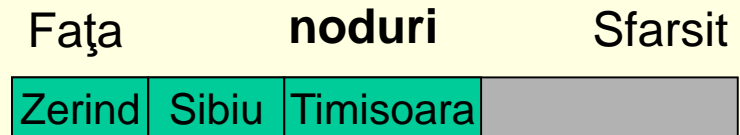
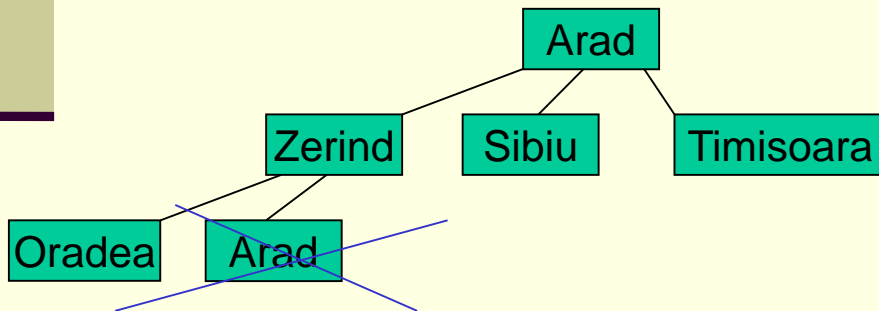
nod = scoate\_din\_fata(noduri)

*Daca* testare\_tinta[problema] se aplica la stare(nod) *atunci*  
 solutie gasita

*Altfel*

noduri = adauga(noduri, expandare(nod, adauga\_la\_sfarsit))

*Sfarsit cat timp*

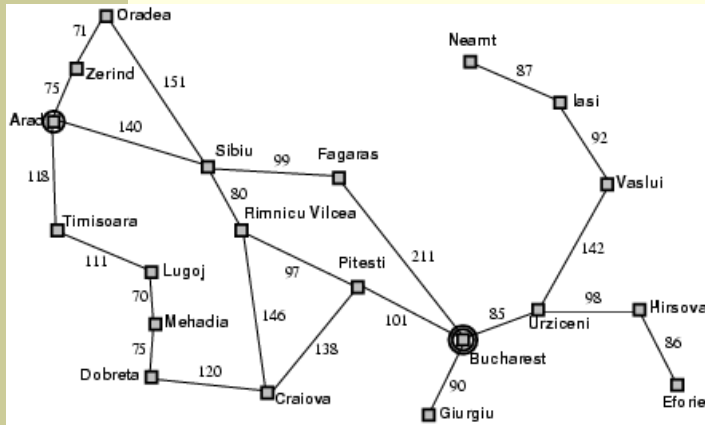


Parcurgerea: Arad, Zerind

A mai aparut acest nod!



# Algoritm cautare in latime



**functia** *cautare\_latime*(problema) **intoarce** solutie sau esec  
 noduri = genereaza\_coada(genereaza\_nod(stare\_initala[problema]))

*Cat timp* solutie negasita si noduri  $\neq \emptyset$  *executa*

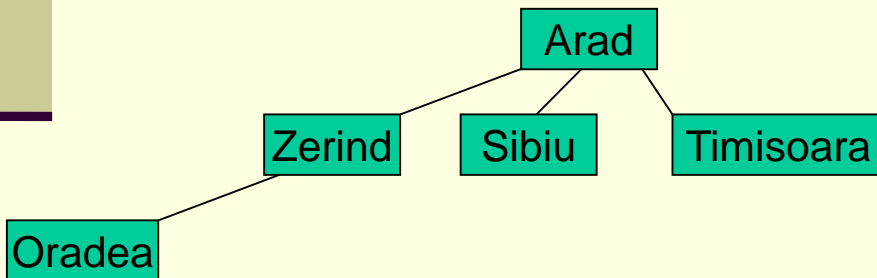
nod = scoate\_din\_fata(noduri)

*Daca* testare\_tinta[problema] se aplica la stare(nod) *atunci*  
 solutie gasita

*Altfel*

noduri = adauga(noduri, expandare(nod, adauga\_la\_sfarsit))

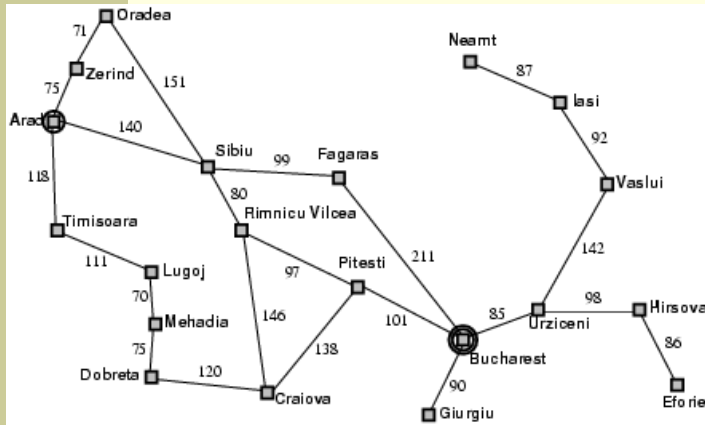
*Sfarsit cat timp*



Fața	noduri	Sfarsit
Sibiu	Timisoara	Oradea

Parcurgerea: Arad, Zerind, Sibiu

# Algoritm cautare in latime



**functia** *cautare\_latime*(problema) **intoarce** solutie sau esec  
 noduri = genereaza\_coada(genereaza\_nod(stare\_initala[problema]))

*Cat timp* solutie negasita si noduri  $\neq \emptyset$  *executa*

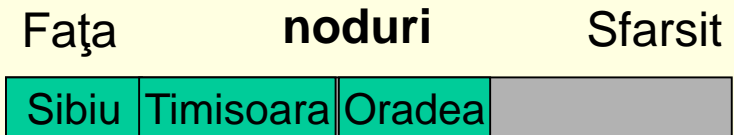
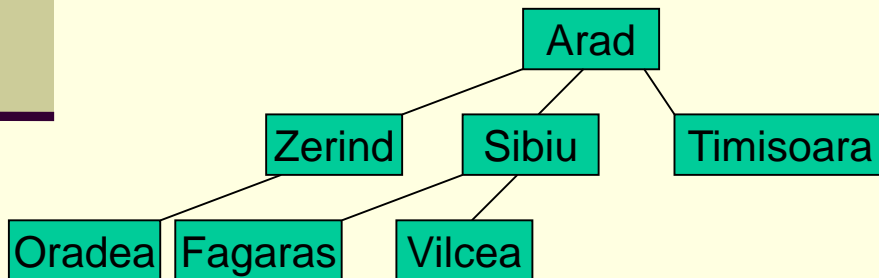
nod = scoate\_din\_fata(noduri)

*Daca* testare\_tinta[problema] se aplica la stare(nod) *atunci*  
 solutie gasita

*Altfel*

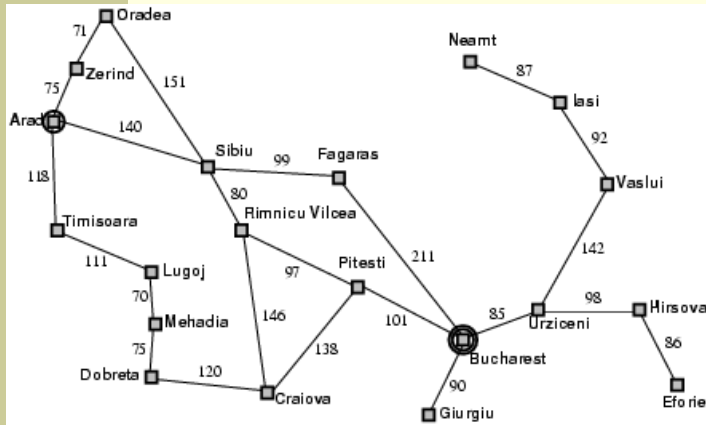
noduri = adauga(noduri, expandare(nod, adauga\_la\_sfarsit))

*Sfarsit cat timp*



Parcurgerea: Arad, Zerind, Sibiu

# Algoritm cautare in latime



**functia** *cautare\_latime*(problema) **intoarce** solutie sau esec  
 noduri = genereaza\_coada(genereaza\_nod(stare\_initala[problema]))

*Cat timp* solutie negasita si noduri  $\neq \emptyset$  *executa*

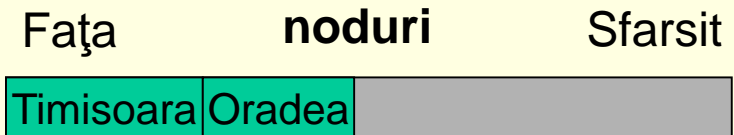
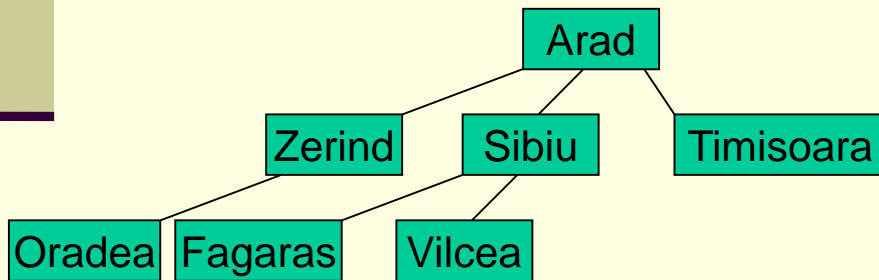
nod = scoate\_din\_fata(noduri)

*Daca* testare\_tinta[problema] se aplica la stare(nod) *atunci*  
 solutie gasita

*Altfel*

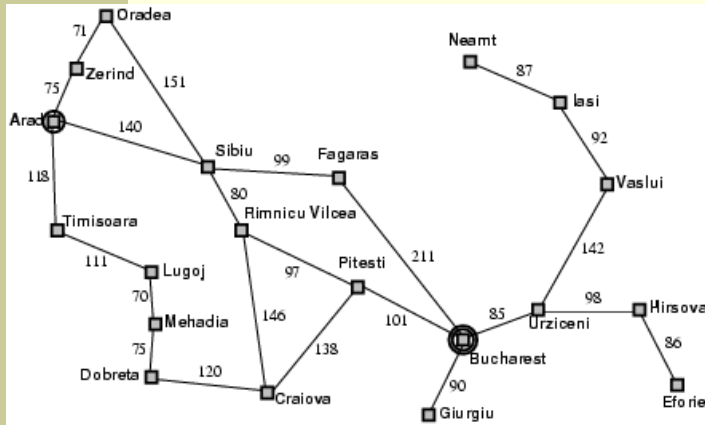
noduri = adauga(noduri, expandare(nod, adauga\_la\_sfarsit))

*Sfarsit cat timp*



Parcurgerea: Arad, Zerind, Sibiu, Timisoara

# Algoritm cautare in latime



**functia** *cautare\_latime*(problema) **intoarce** solutie sau **esec**  
 noduri = genereaza\_coada(genereaza\_nod(stare\_initala[problema]))

*Cat timp* solutie negasita si noduri  $\neq \emptyset$  *executa*

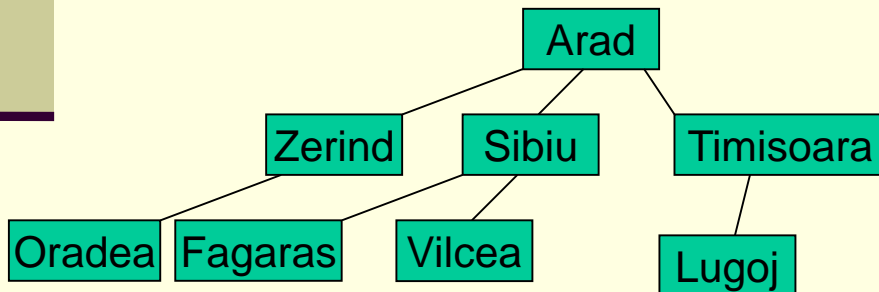
nod = scoate\_din\_fata(noduri)

*Daca* testare\_tinta[problema] se aplica la stare(nod) *atunci*  
 solutie gasita

*Altfel*

noduri = adauga(noduri, expandare(nod, adauga\_la\_sfarsit))

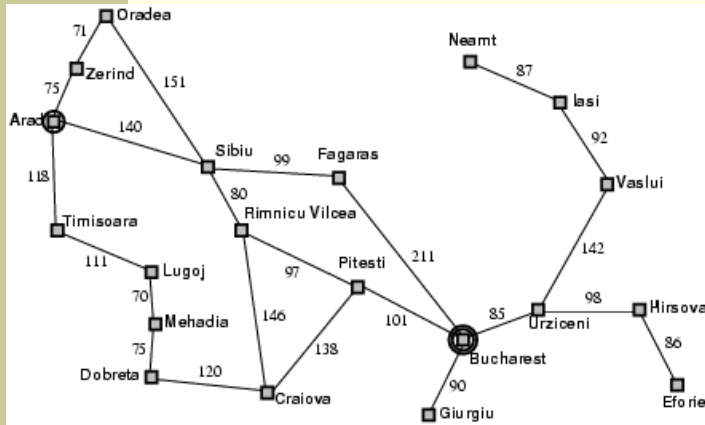
*Sfarsit cat timp*



Fața	noduri	Sfarsit
	Timisoara Oradea Fagaras Vilcea	

Parcurgerea: Arad, Zerind, Sibiu, Timisoara

# Algoritm cautare in latime



**functia** *cautare\_latime*(problema) **intoarce** solutie sau esec  
 noduri = genereaza\_coada(genereaza\_nod(stare\_initala[problema]))

*Cat timp* solutie negasita si noduri  $\neq \emptyset$  *executa*

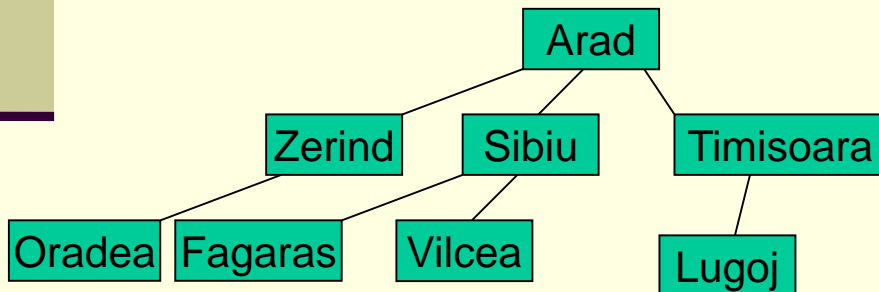
nod = scoate\_din\_fata(noduri)

*Daca* testare\_tinta[problema] se aplica la stare(nod) *atunci*  
 solutie gasita

*Altfel*

noduri = adauga(noduri, expandare(nod, adauga\_la\_sfarsit))

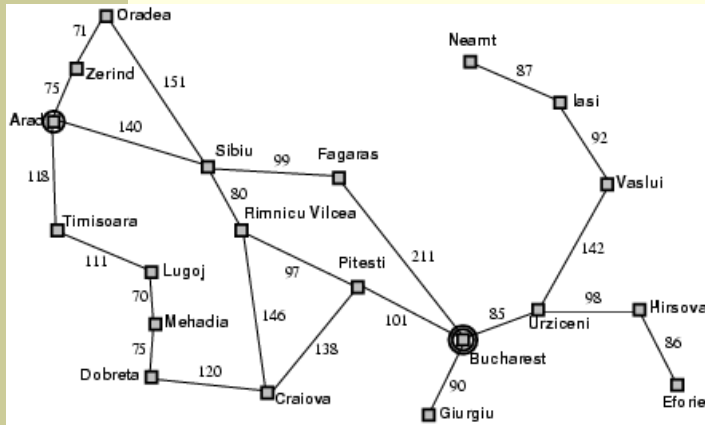
*Sfarsit cat timp*



Fața	noduri	Sfarsit		
Oradea	Fagaras	Vilcea	Lugoj	

Parcurgerea: Arad, Zerind, Sibiu, Timisoara, Oradea

# Algoritm cautare in latime



**functia** *cautare\_latime*(problema) **intoarce** solutie sau esec  
 noduri = genereaza\_coada(genereaza\_nod(stare\_initala[problema]))

*Cat timp* solutie negasita si noduri  $\neq \emptyset$  *executa*

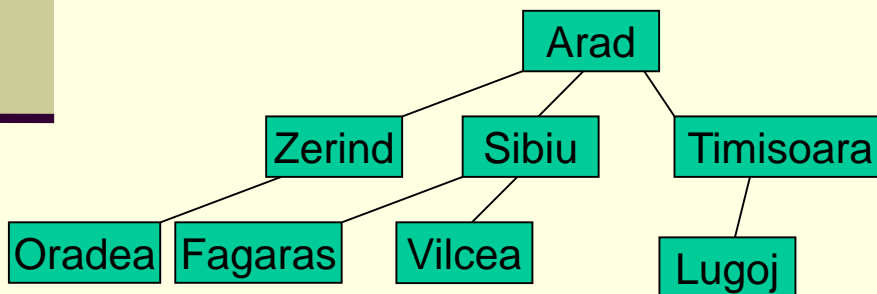
nod = scoate\_din\_fata(noduri)

*Daca* testare\_tinta[problema] se aplica la stare(nod) *atunci*  
 solutie gasita

*Altfel*

noduri = adauga(noduri, expandare(nod, adauga\_la\_sfarsit))

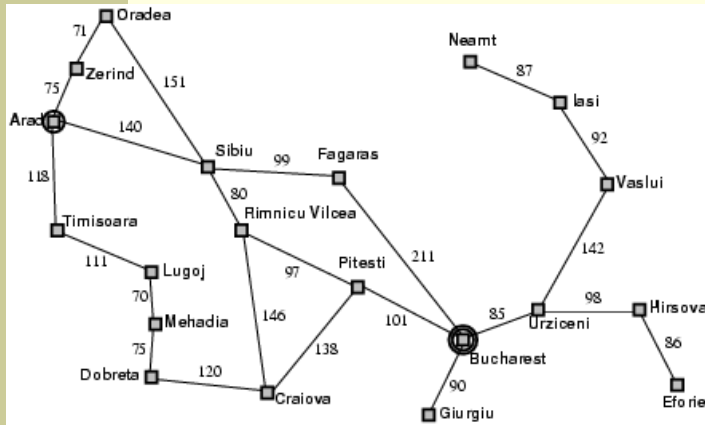
*Sfarsit cat timp*



Fața	noduri	Sfarsit
Fagaras	Vilcea	Lugoj

Parcurgerea: Arad, Zerind, Sibiu,  
 Timisoara, Oradea, Fagaras

# Algoritm cautare in latime



**functia** `cautare_latime(problema)` **intoarce** `solutie` sau `esec`  
`noduri = genereaza_coada(genereaza_nod(stare_initala[problema]))`

*Cat timp* `solutie` negasita si `noduri`  $\neq \emptyset$  *executa*

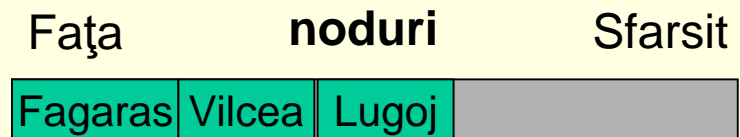
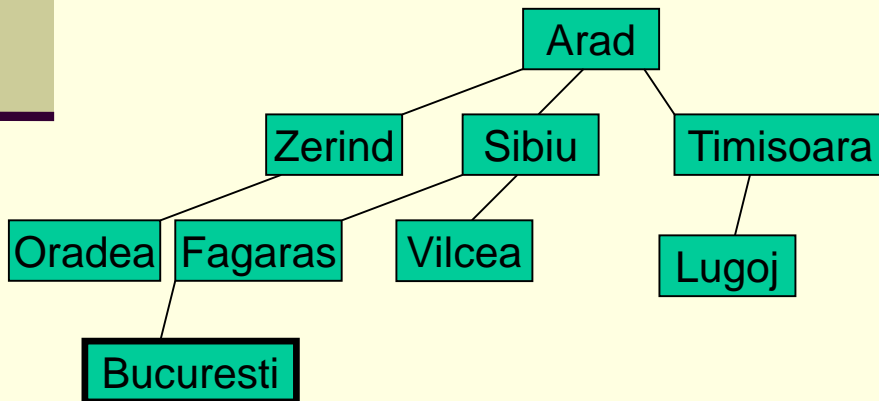
`nod = scoate_din_fata(noduri)`

*Daca* `testare_tinta[problema]` se aplica la `stare(nod)` *atunci*  
`solutie gasita`

*Altfel*

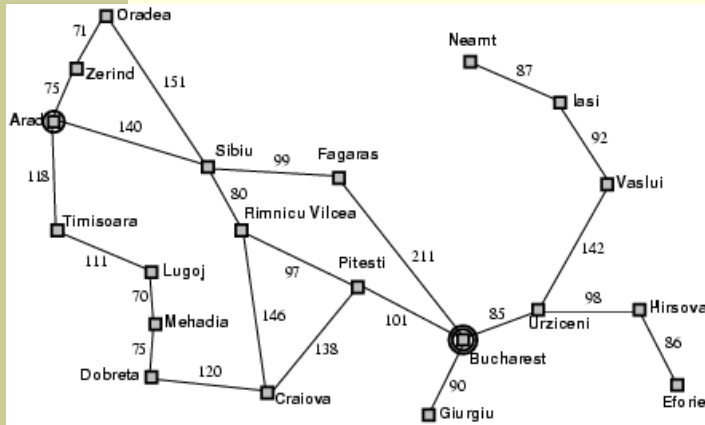
`noduri = adauga(noduri, expandare(nod, adauga_la_sfarsit))`

*Sfarsit cat timp*



Parcurgerea: Arad, Zerind, Sibiu,  
 Timisoara, Oradea, Fagaras

# Algoritm cautare in latime



**functia** *cautare\_latime*(problema) **intoarce** solutie sau esec  
 noduri = genereaza\_coada(genereaza\_nod(stare\_initiala[problema]))

*Cat timp* solutie negasita si noduri  $\neq \emptyset$  *executa*

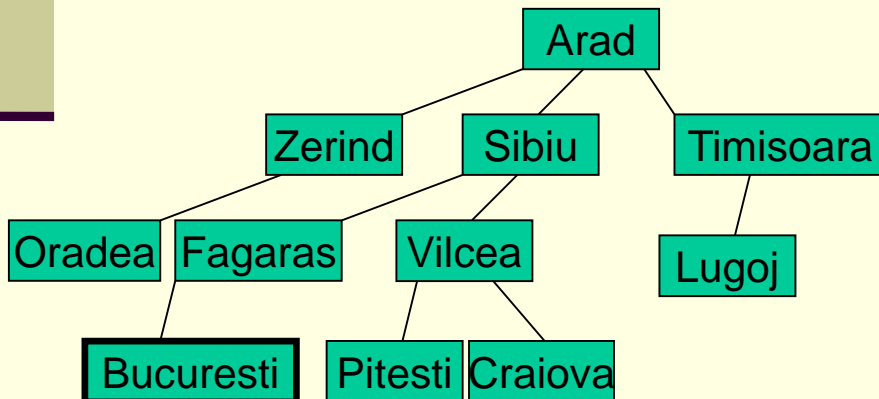
nod = scoate\_din\_fata(noduri)

*Daca* testare\_tinta[problema] se aplica la stare(nod) *atunci*  
 solutie gasita

*Altfel*

noduri = adauga(noduri, expandare(nod, adauga\_la\_sfarsit))

*Sfarsit cat timp*

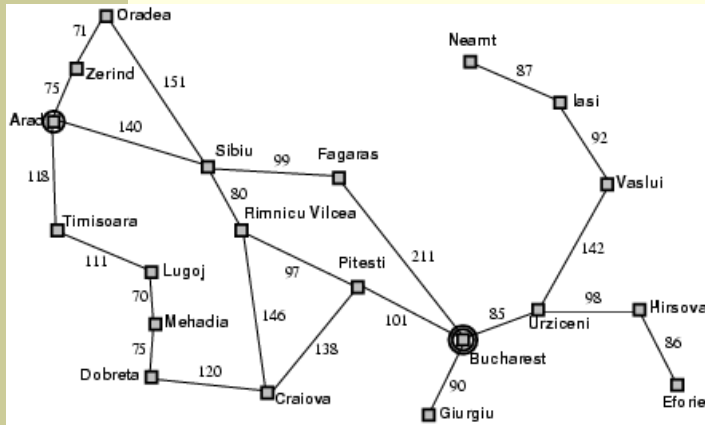


Fața	noduri	Sfarsit
Vilcea	Lugoj	Bucuresti

Parcurgerea: Arad, Zerind, Sibiu, Timisoara, Oradea, Fagaras, Vilcea



# Algoritm cautare in latime



**functia** *cautare\_latime*(problema) **intoarce** solutie sau esec  
 noduri = genereaza\_coada(genereaza\_nod(stare\_initala[problema]))

*Cat timp* solutie negasita si noduri  $\neq \emptyset$  *executa*

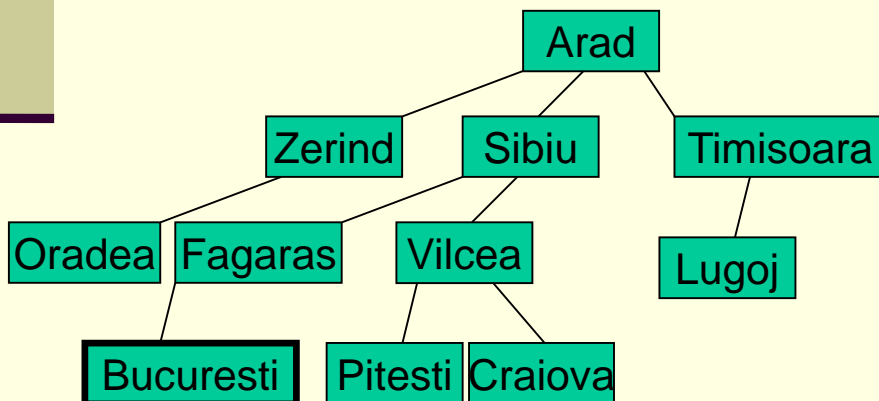
nod = scoate\_din\_fata(noduri)

*Daca* testare\_tinta[problema] se aplica la stare(nod) *atunci*  
 solutie gasita

*Altfel*

noduri = adauga(noduri, expandare(nod, adauga\_la\_sfarsit))

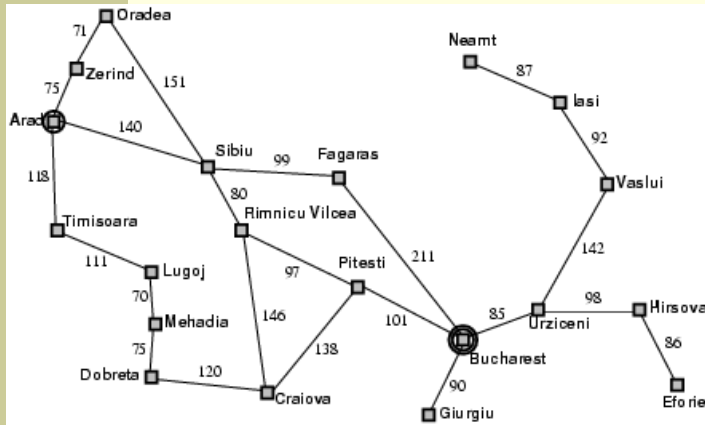
*Sfarsit cat timp*



Fața	noduri	Sfarsit
Lugoj	Bucuresti	Pitesti
Craiova		

Parcurgerea: Arad, Zerind, Sibiu, Timisoara, Oradea, Fagaras, Vilcea, Lugoj

# Algoritm cautare in latime



**functia** `cautare_latime(problema)` **intoarce** `solutie` sau `esec`  
`noduri = genereaza_coada(genereaza_nod(stare_initiala[problema]))`

*Cat timp* `solutie` negasita si `noduri`  $\neq \emptyset$  *executa*

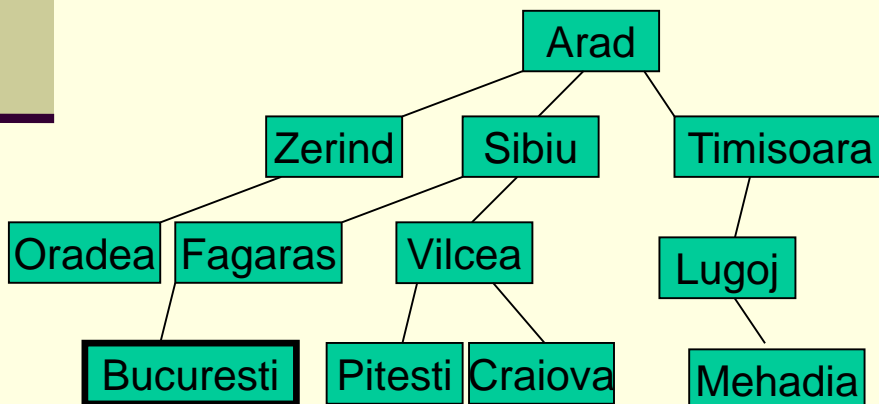
`nod = scoate_din_fata(noduri)`

*Daca* `testare_tinta[problema]` se aplica la `stare(nod)` *atunci*  
`solutie gasita`

*Altfel*

`noduri = adauga(noduri, expandare(nod, adauga_la_sfarsit))`

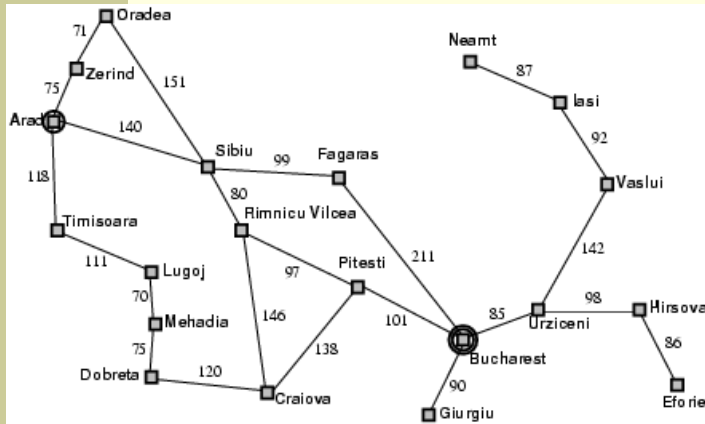
*Sfarsit cat timp*



Fața	noduri	Sfarsit
Lugoj	Bucuresti	Pitesti
	Craiova	

Parcurgerea: Arad, Zerind, Sibiu, Timisoara, Oradea, Fagaras, Vilcea, Lugoj

# Algoritm cautare in latime



*functia* cautare\_latime(problema) **intoarce** solutie sau esec  
 noduri = genereaza\_coada(genereaza\_nod(stare\_initiala[problema]))

*Cat timp* solutie negasita si noduri  $\neq \emptyset$  *executa*

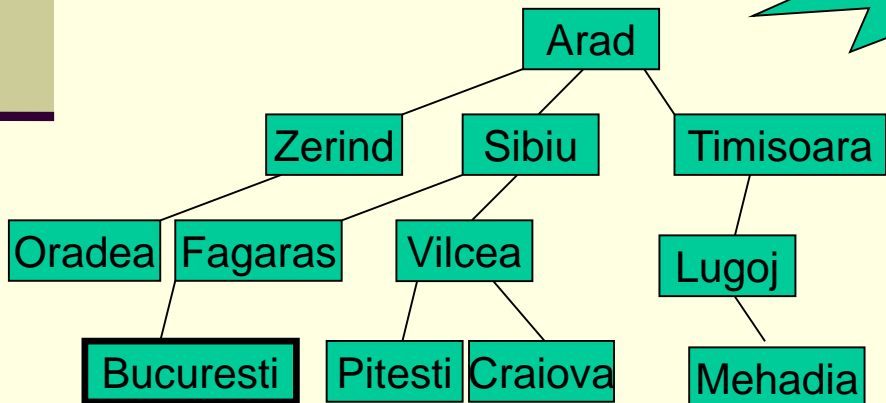
nod = scoate\_din\_fata(noduri)

*Daca* testare\_tinta[problema] se aplica la stare(nod) *atunci*  
 solutie gasita

*Altfel*

noduri = adauga(noduri, expandare(nod, adauga\_la\_sfarsit))

*Sfarsit* cat timp



Fața	noduri	Sfarsit
	Bucuresti   Pitesti   Craiova   Mehadia	

Parcurgerea: Arad, Zerind, Sibiu, Timisoara, Oradea, Fagaras, Vilcea, Lugoj, **Bucuresti!**

# Cautarea in latime

---

- Algoritmul satisface criteriile:
  - Completitudine
  - Optimalitate, cu conditia ca orice operatiune sa aiba acelasi cost.
- De verificat complexitatile...
  - Presupunem ca fiecare stare poate fi expandata la  $b$  alte stari.
  - Nodul radacina genereaza  $b$  noduri la primul nivel, si fiecare din acestea genereaza cate  $b$  noduri => pe al doilea nivel avem  $b^2$  noduri.

# Cautarea in latime

- Daca solutia problemei se gaseste la lungimea  $d$  atunci numarul maxim de noduri expandate pentru gasirea solutiei va fi:

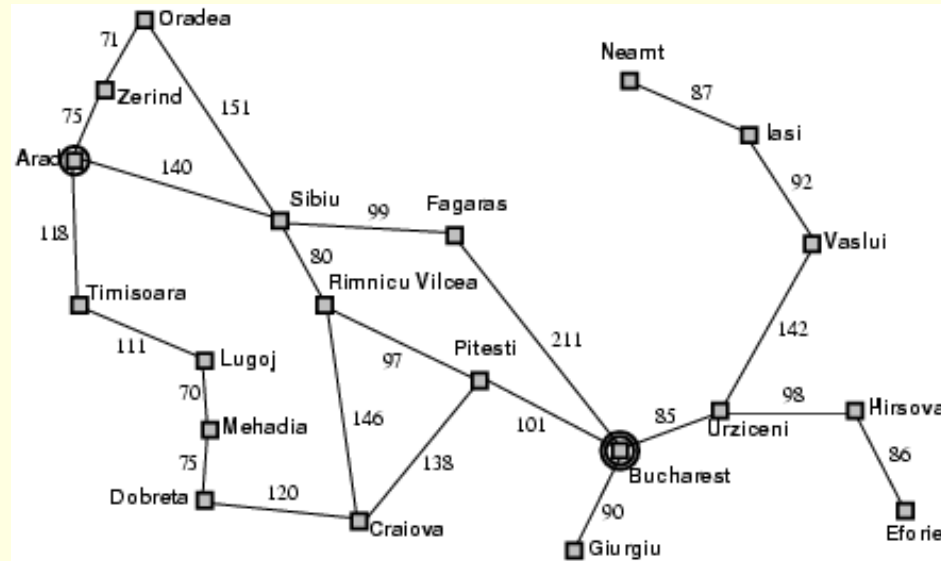
$$1 + b + b^2 + b^3 + \dots + b^d$$

- Complexitatea  $O(b^d)$ .
- Consideram un exemplu in care  $b = 10$  si vom urmari diverse valori pentru  $d$ .
- Consideram ca se proceseaza 1 milion de noduri pe secunda.
- Un nod se reprezinta pe 1000 de bytes.

# Cautarea în latime

Adâncime	Noduri	Timp	Memorie
2	110	0.11 milisecunde	107 kilobytes
4	11 110	11 milisecunde	10.6 megabytes
6	$10^6$	11 secunde	1 gigabyte
8	$10^8$	2 minute	103 gigabytes
10	$10^{10}$	3 ore	10 terabytes
12	$10^{12}$	13 zile	1 petabyte
14	$10^{14}$	3.5 ani	99 petabytes
16	$10^{16}$	350 ani	10 exabytes

# Exercitiu



Fața                      **noduri**                      Sfarsit

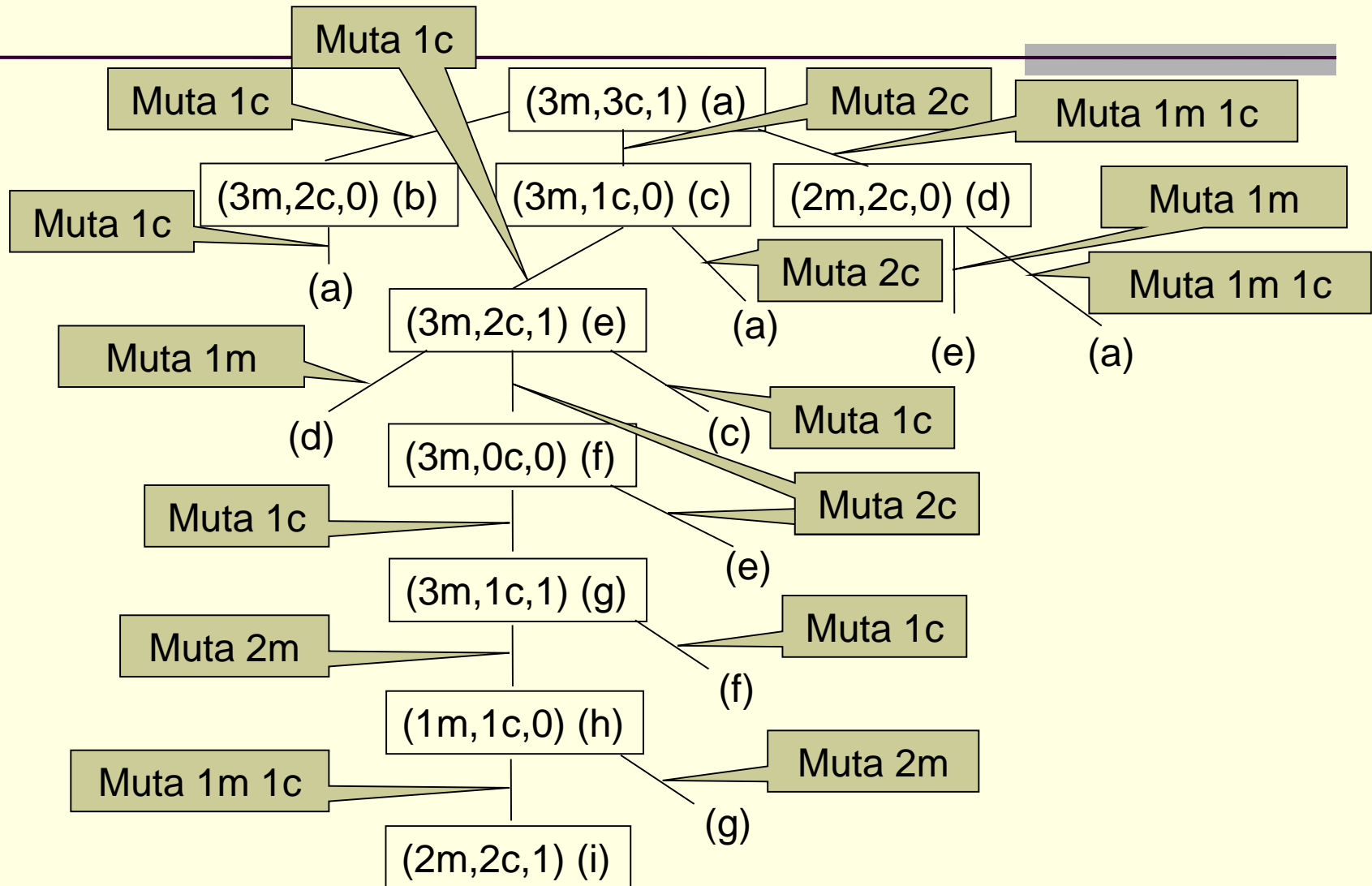


Gasiti o ruta de la Bucuresti la Sibiu folosind parcurgerea in latime.

Desenati arborele, scrieti parcurgerea si continutul pentru noduri la fiecare pas.

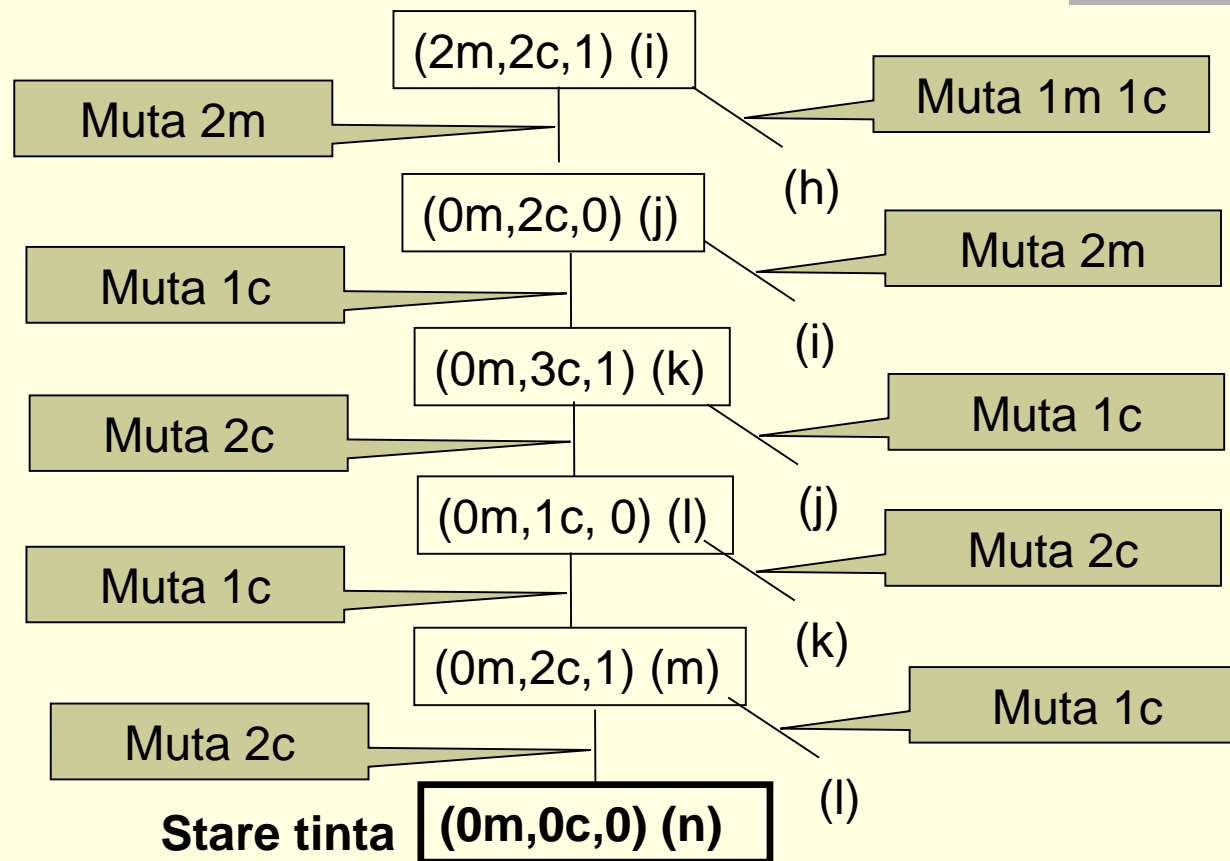
**Parcurgerea:** Bucuresti, ..., Rm. Vilcea

# Misionarii si canibalii





# Misionarii și canibalii



# Puzzle cu 8 valori

	2	3
1	4	6
7	5	8

Starea initiala

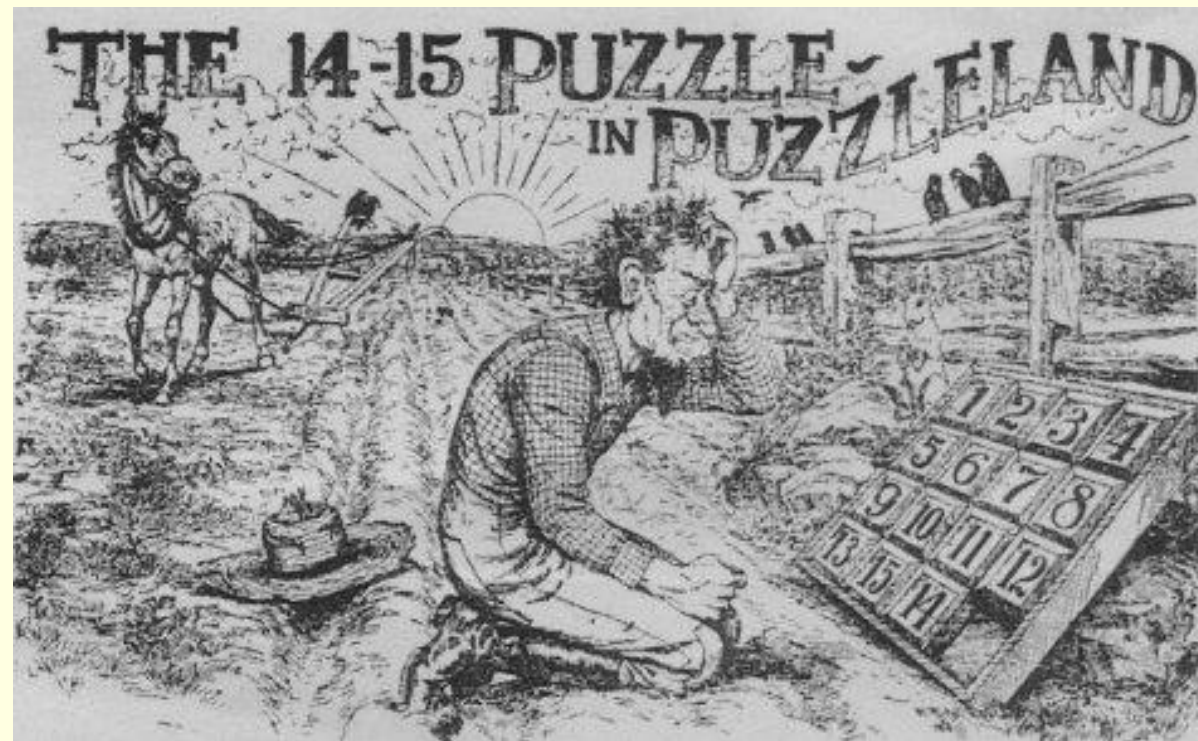
1	2	3
4	5	6
7	8	

Starea tinta

- **Stari:** este descrisa locatia fiecarei cifre in una din cele 9 casute.
- **Actiuni:** casuta goala se misca la stanga, dreapta, sus sau jos.
- **Testarea tintei:** starea se gaseste in configuratia din dreapta.
- **Costul drumului:** fiecare pas are costul 1, deci costul drumului este dat de numarul de mutari.

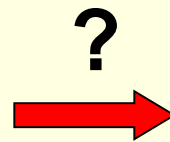
# Puzzle cu 15 valori

- Sam Loyd a introdus în 1878 15-puzzle-ul.
- A oferit un premiu de 1 000 \$ din proprii bani pentru cel care rezolva problema în situația în care doar 14 și 15 sunt inversate, restul fiind aliniate crescător.



# Puzzle cu 15 valori

1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

**Nimeni nu a castigat premiul!**

# Puzzle cu 8 valori

---

- Cate stari posibile exista?
  - $9! = 362\,880$  stari
- Numai la **jumatate** din aceste stari se poate ajunge din orice stare data.

# Posibilitatea de a atinge scopul final...

- O valoare  $j$  apare dupa o valoare  $i$  daca
  - $j$  apare pe acelasi rand, dar in dreapta lui  $i$
  - $j$  se afla sub linia lui  $i$
- Pentru  $i = 1, 2, \dots, 8$ , fie  $n_i$  numarul de aparitii ale lui  $j$  ( $j < i$ ) dupa  $i$ .
- $N = n_2 + n_3 + \dots + n_8$

	2	3
1	4	6
7	5	8

$$n_2 = 1$$

$$n_6 = 1$$

$$n_3 = 1$$

$$n_7 = 1$$

$$n_4 = 0$$

$$n_8 = 0$$

$$n_5 = 0$$

$$N = 4$$

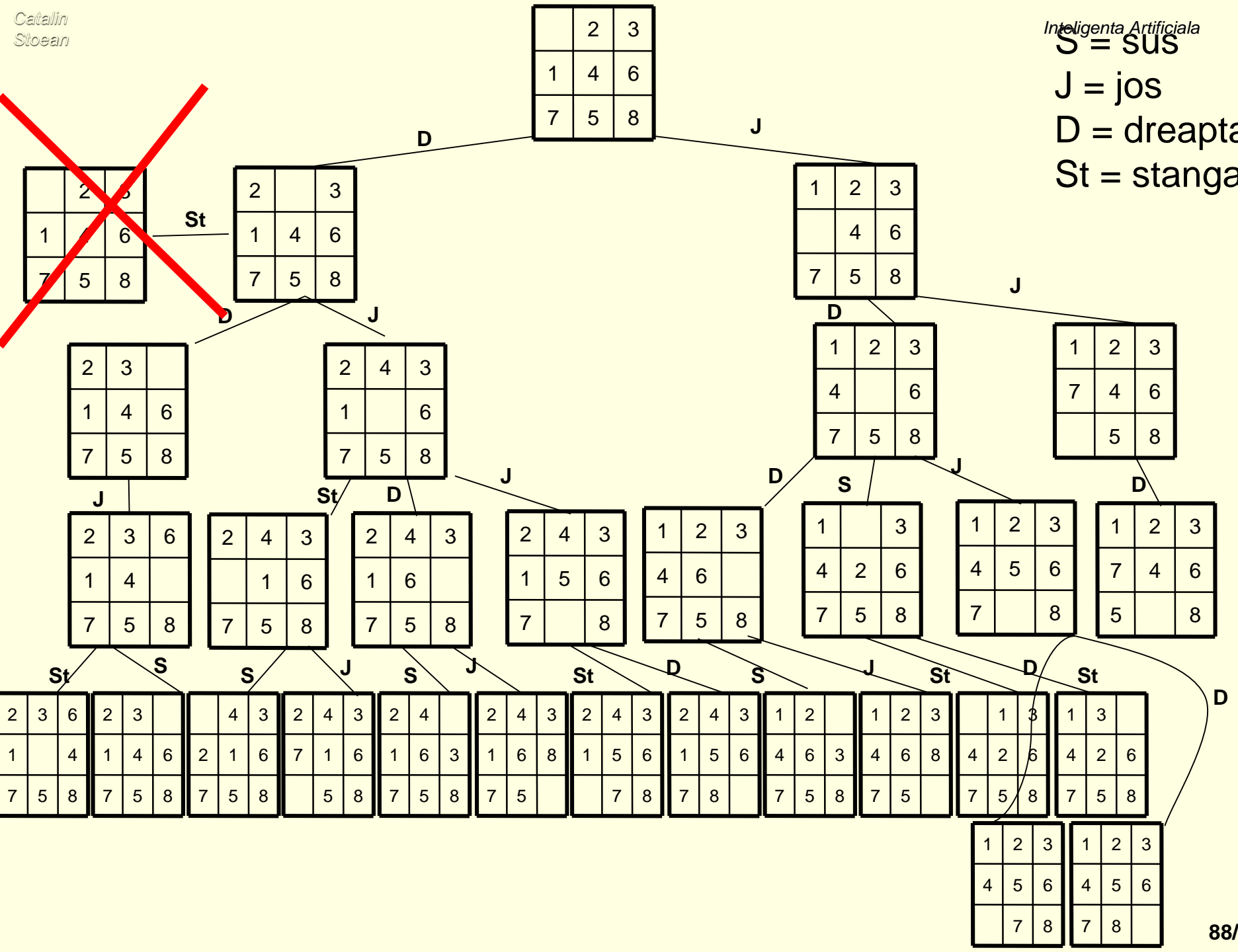
Pentru  $N$  par, puzzle-ul cu 8 valori este rezolvabil!

S = sus

J = jos

D = dreapta

St = stanga



# Exercitiu - Problema celor 4 dame

---

- **Stari:** orice aranjament de 0 pana la 4 dame care nu se ataca.
- **Actiuni:** adauga o dama pe coloana cea mai din stanga a.i. sa nu fie atacata de alta dama.
- **Testarea tintei:** 4 dame care nu se ataca pe tabla.
- **Costul drumului:** 0.
  
- Pornind de la o tabla de 4x4 goala si folosind datele problemei de mai sus, sa se construiasca printr-o cautare in latime arborele complet care duce la rezolvarea problemei. Numerotati nodurile in ordinea in care au fost vizitate.



# Recapitulare 1/3

---

- Am studiat metode pe care un agent le poate utiliza cand nu este clar care actiune imediata trebuie urmata.
- In astfel de cazuri, agentul poate considera posibile secvente de actiuni; acest proces se numeste **cautare**.
- Inainte de a cauta solutii, un agent trebuie sa formuleze o tinta (un scop) si sa foloseasca apoi aceasta tinta pentru a formula problema.
- O **problema** consta din 4 parti:
  - O **stare initiala**
  - O **multime de actiuni**
  - O **functie care testeaza** daca starea curenta este chiar **tinta**
  - O **functie de cost** al drumului.

# Recapitulare 2/3

---

- Mediul problemei este reprezentat prin **spatiul starilor**.
- Un drum prin spatiul starilor de la starea initiala la starea tinta este o **solutie**.
- In viata reala, cele mai multe probleme sunt prost-definite; dupa analiza, multe probleme pot fi transpuse intr-un spatiu al starilor.
- Un **algoritm general de cautare** poate fi folosit pentru rezolvarea oricarei probleme.
- Algoritmii de cautare sunt analizati in functie de **completitudine**, **optimalitate**, **complexitatea timpului**, **complexitatea spatiului**.
- Complexitatea depinde de  $b$  (numarul de noduri in care se expandeaza un nod) si de  $d$  (adancimea la care se gaseste cea mai apropiata solutie).

# Recapitulare 3/3

---

- **Cautarea in latime** gaseste solutia care se afla cel mai aproape de nodul radacina.
  - Complet
  - Optim, daca fiecare actiune are acelasi cost
  - Complexitatea temporala si spatiala:  $O(b^d)$