Agenti care rezolva probleme

Catalin Stoean

catalin.stoean@inf.ucv.ro

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

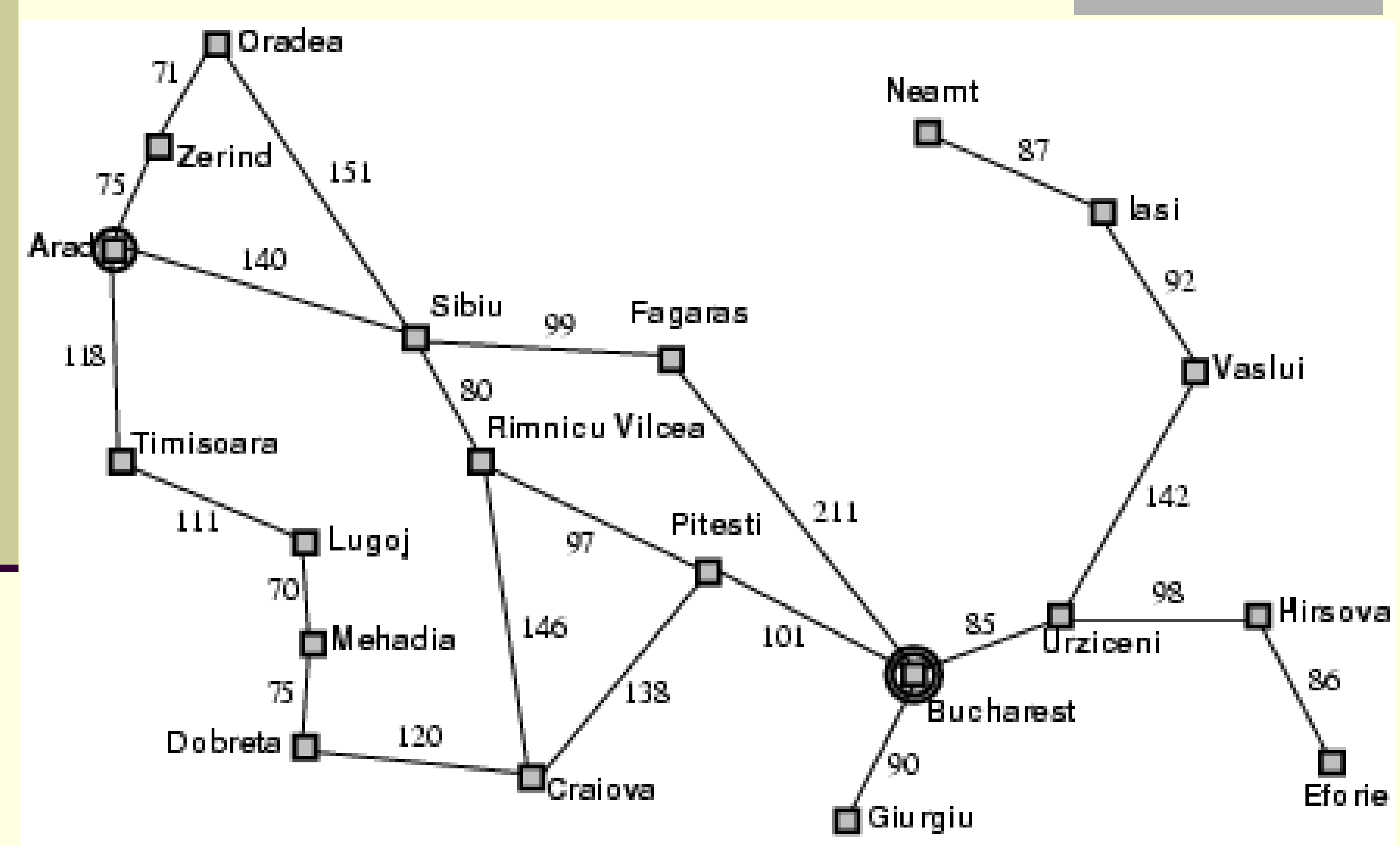
Agenti care rezolva probleme

- 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.
- **Formularea scopului:**
  - Ajungerea in Bucuresti
- **Formularea 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*



Formularea problemelor

- **Formularea scopului** este primul lucru care trebuie stabilit de catre agent (ajungerea in Bucuresti).
- Intuitiv, **formularea 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 aleator.
- Daca are o harta, poate examina diferite **secvente de actiuni** posibile care duc la starea finala, apoi alege cea mai buna secventa de actiuni.

Formularea 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?

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?
- Raspuns:

Fermierul are 60 de 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*)

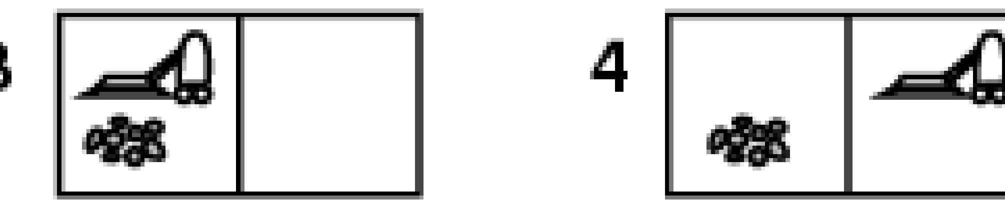
intoarce actiune

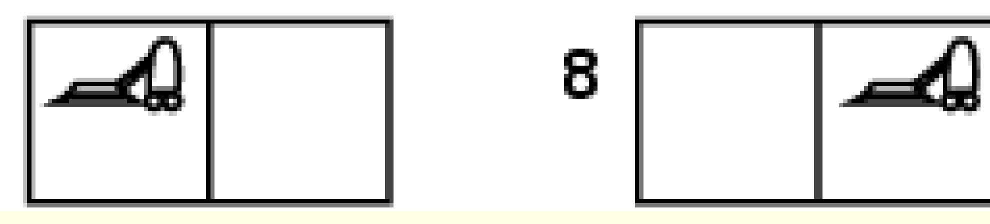
Formularea problemelor

- 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

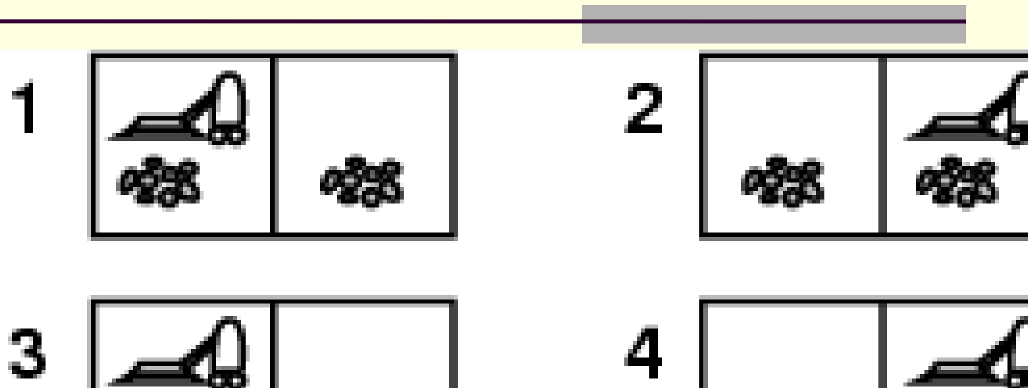
- Avem 2 locatii.
- In fiecare locatie
  - poate fi sau nu mizerie
  - poate sa fie aspiratorul sau nu
- Cele 8 stari posibile →
- Actiuni: *stanga, dreapta, aspira.*
- Scop: curatarea ambelor incaperi (starile 7 sau 8)

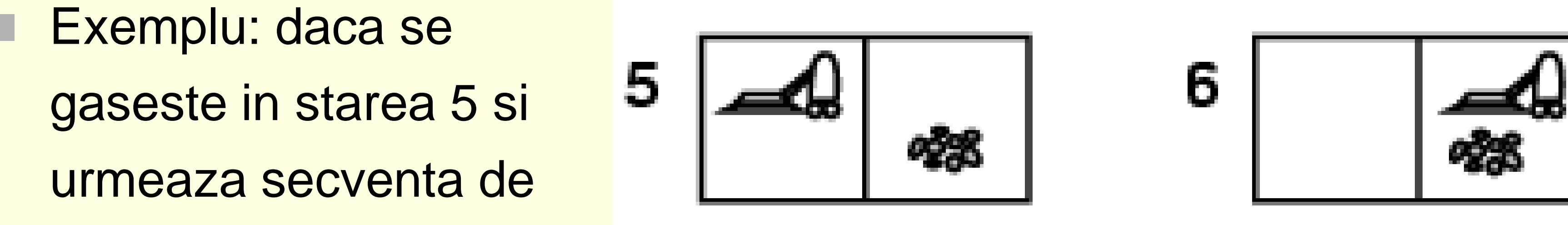




Problema cu o singura stare (exemplu)

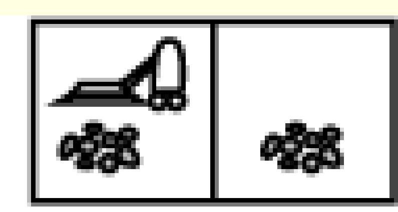
- Senzorii agentului ii spun exact starea in care se gaseste.
- Agentul stie exact ce efecte au actiunile sale.
- Exemplu: daca se gaseste in starea 5 si urmeaza secventa de actiuni [*Dreapta*, *Aspira*], se ajunge la atingerea scopului problemei.

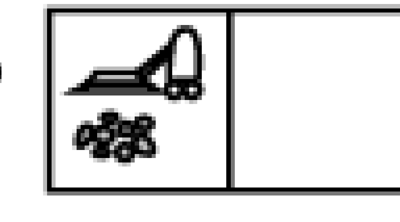
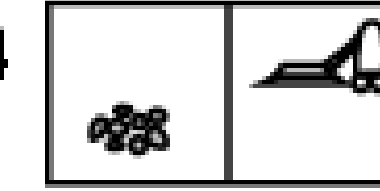
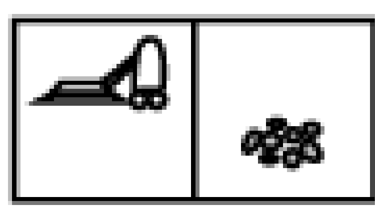
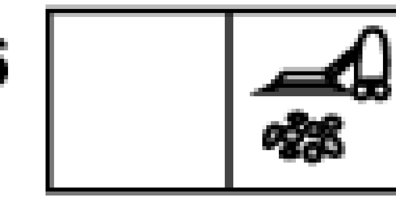
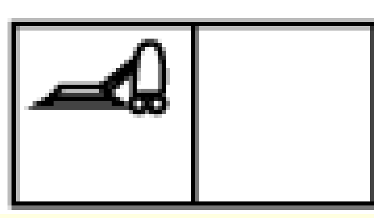
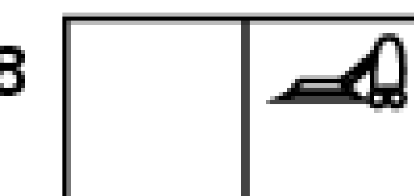




Problema cu mai multe stari (exemplu)

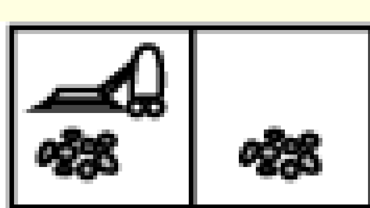
- Agentul stie exact ce efecte au actiunile sale dar…
- Senzorii agentului au acces limitat fata de starea in care se gaseste.
- Poate sa nu aiba senzori deloc – stie doar ca in starea initiala se gaseste in una din starile {1, 2, …, 8}
- Poate ajunge sa *rezolve* problema?...

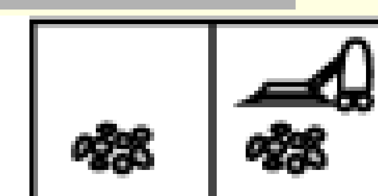
1

2

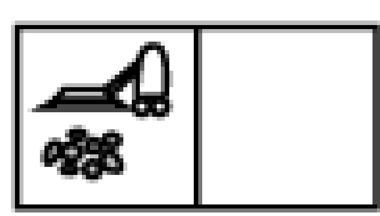
3

4

5

6

7

8


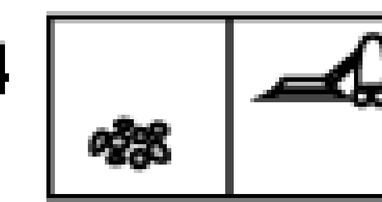
Problema cu mai multe stari (exemplu)

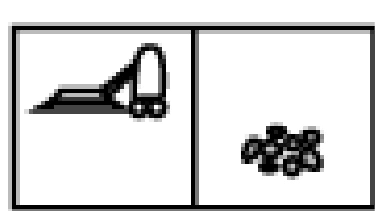
- **DA**, pentru ca stie ce efect au actiunile lui.
- Actiunea *Dreapta* il va face sa se gaseasca in una din situatiile {2, 4, 6, 8}.
- Ce efect va avea secventa de actiuni: [*Dreapta, Aspira, Stanga, Aspira*]?...
- Agentul trebuie sa rationeze in raport cu *multimi de stari* la care poate ajunge.

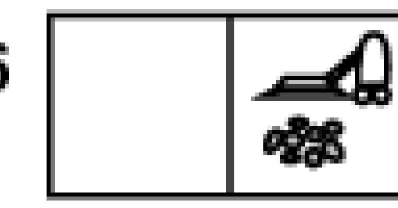
1


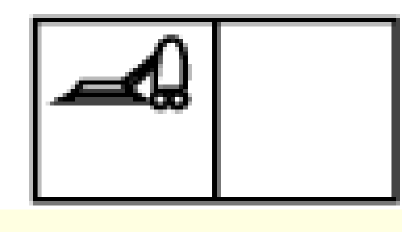
2


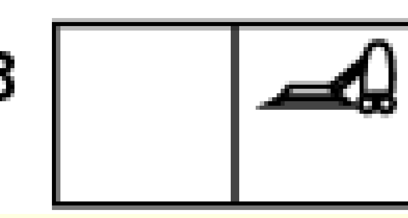
3


4


5


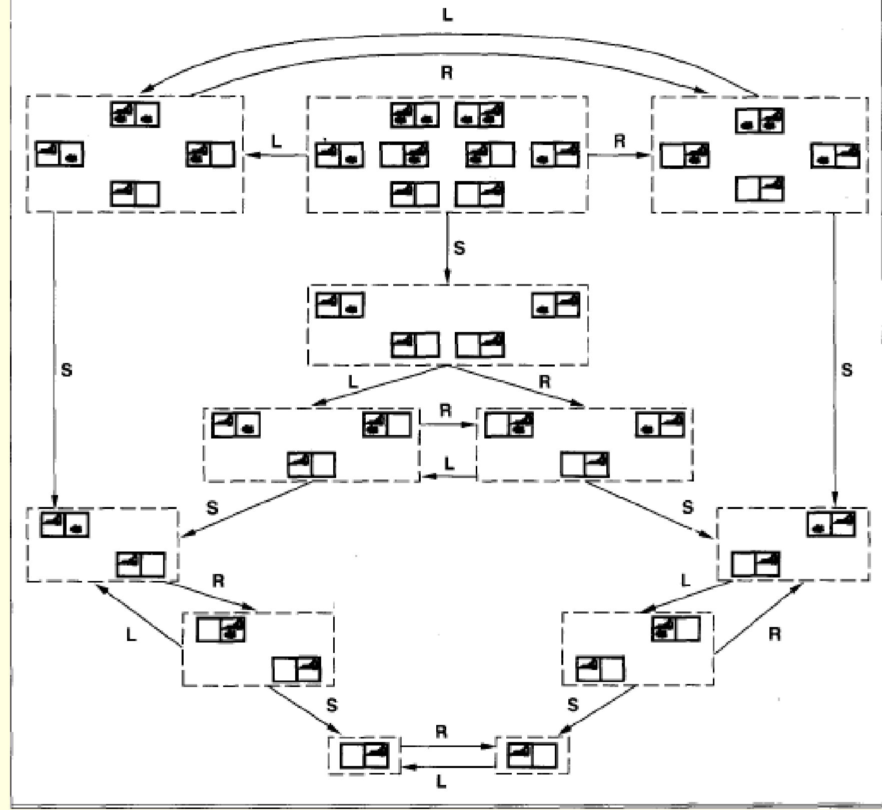
6


7


8


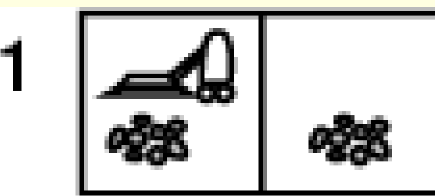
Problema cu mai multe stari (exemplu)

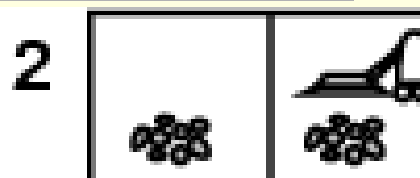
- Cand mediul nu este complet accesibil, agentul lucreaza cu multimi de stari in care poate ajunge, nu cu cate o singura stare.

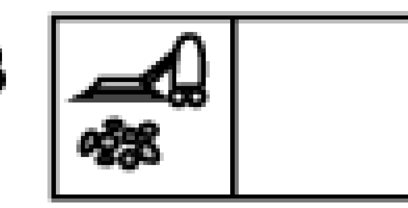


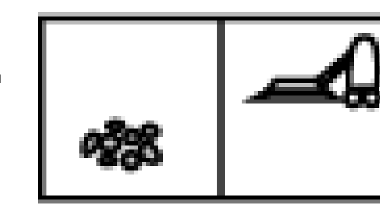
Problema contingenta (exemplu)

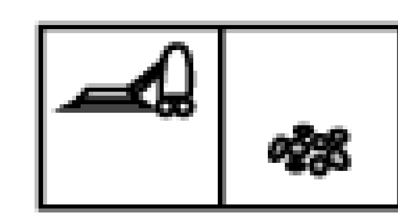
- Presupunem ca mediul este nedeterminist.
- Legile lui Murphy guverneaza mediul
  - aspirarea duce la depozitarea murdariei intr-un loc… care era complet curat…
- De exemplu, in starea 4, daca aspira se poate ajunge la 2 sau 4.

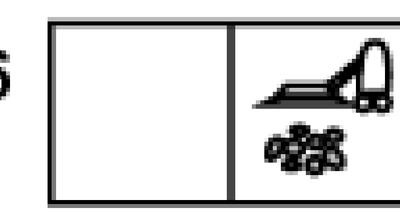


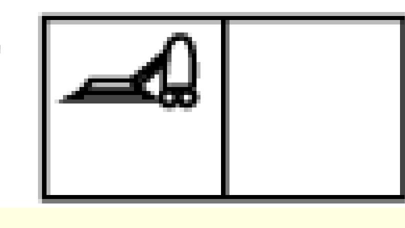


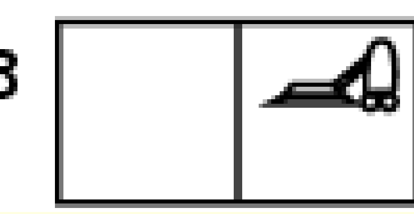






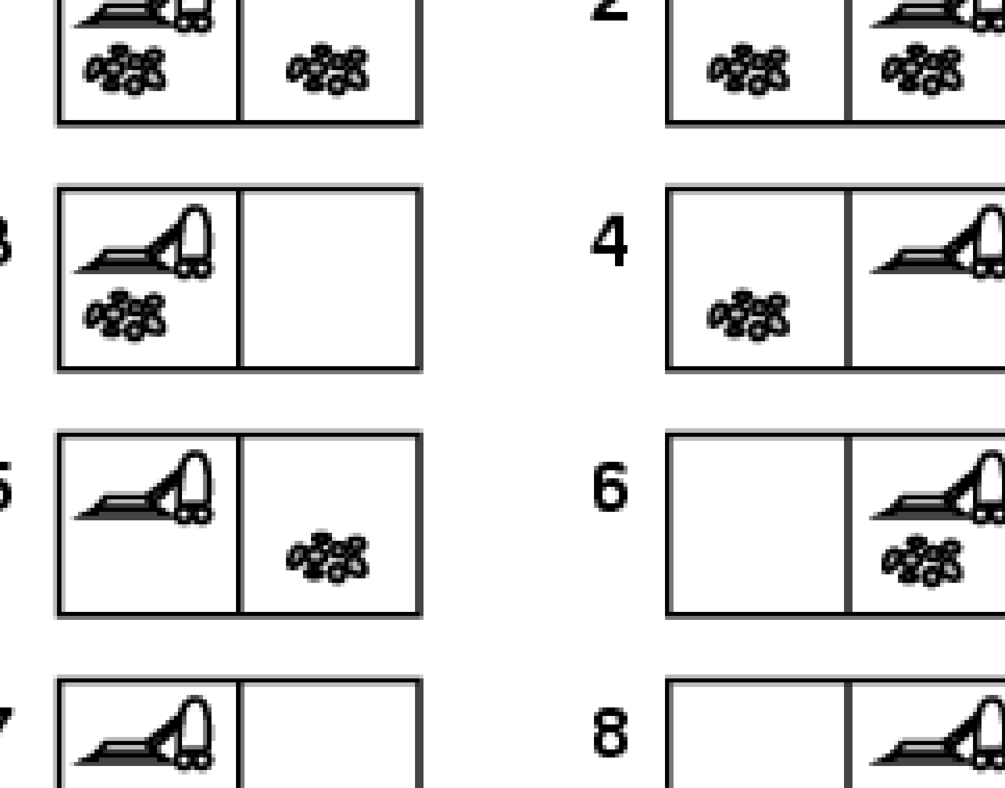






Problema contingenta (exemplu)

- Nu exista o secventa de actiuni unica ce reprezinta solutia problemei.
- Este nevoie de utilizare a senzorilor 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.

Formularea 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(Arad) = {Zerind, Sibiu, Timisoara})
  3. **Testarea tintei problemei** – se verifica daca starea curenta a atins tinta problemei (x = Bucuresti, 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 ≥ 0
- O solutie este o secventa de actiuni care merg de la starea initiala la starea tinta

Formularea problemelor

- Lumea reala este foarte complexa => spatiul starilor trebuie sa fie abstractizat pentru rezolvarea de probleme.
- (Abstract) Stare = multime de stari reale.
- (Abstract) Actiune = combinare 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

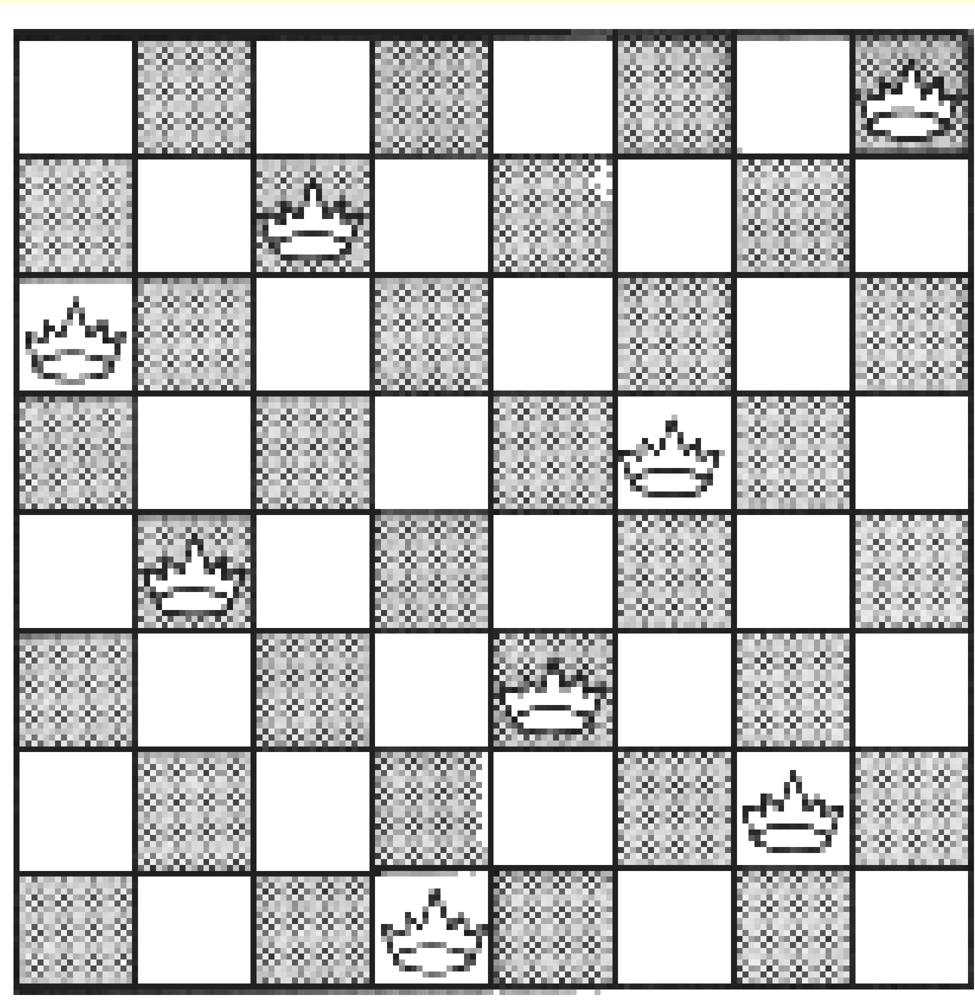
	1	2
3	4	5
6	7	8

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.

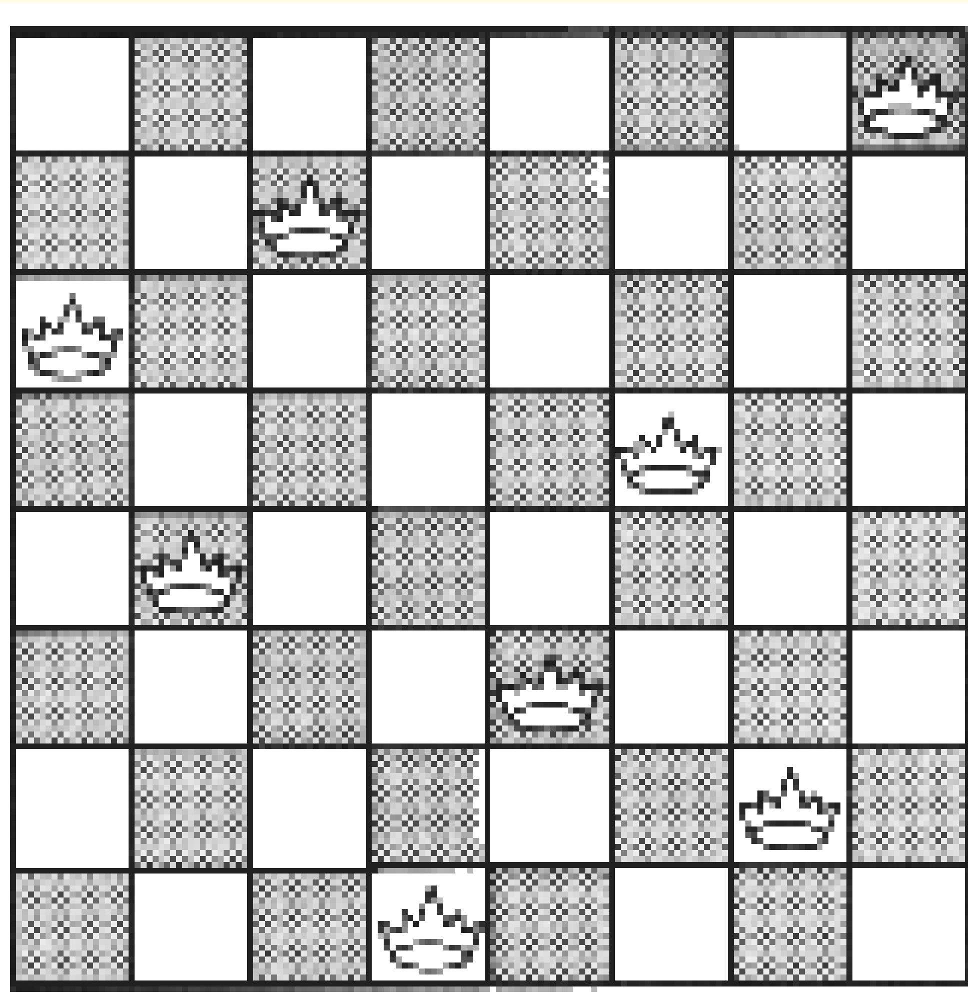
Problema celor 8 dame

- **Stari**: orice aranjament de 0 pana 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 actiuni)

- **Stari**: orice aranjament de 0 pana la 8 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**: 8 dame care nu se ataca pe tabla.
- **Costul drumului**: 0.
- 2057 posibilitati

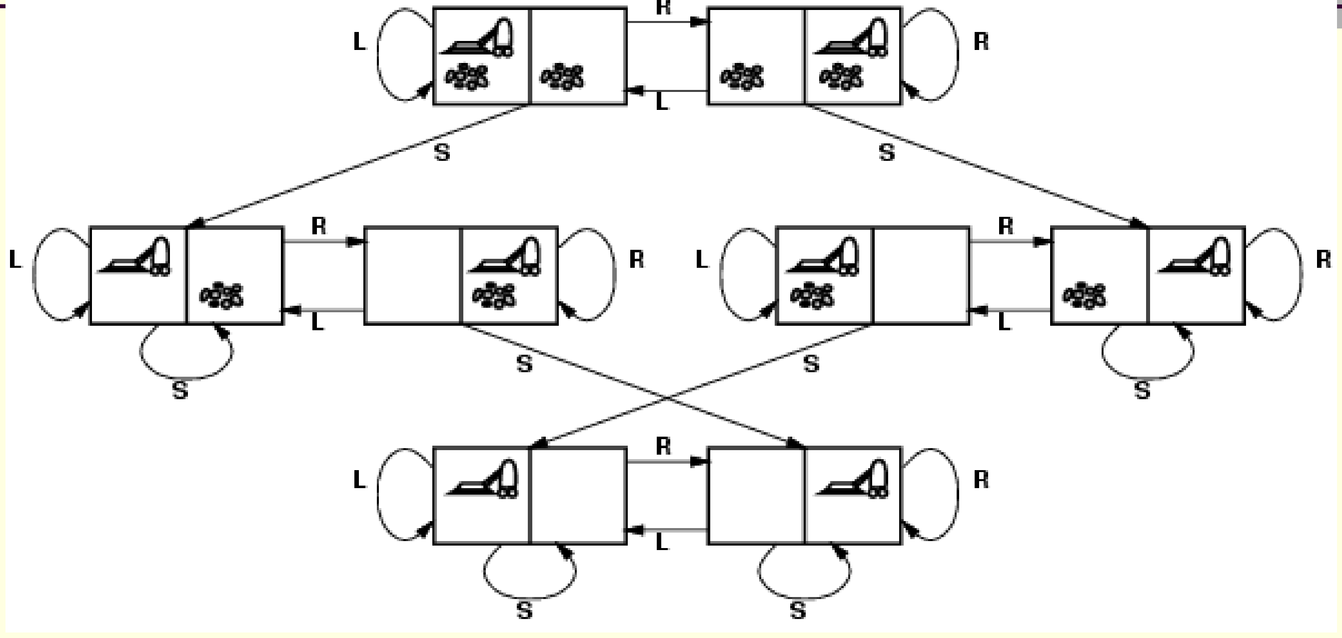


Criptaritmetica

```
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 singura 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.

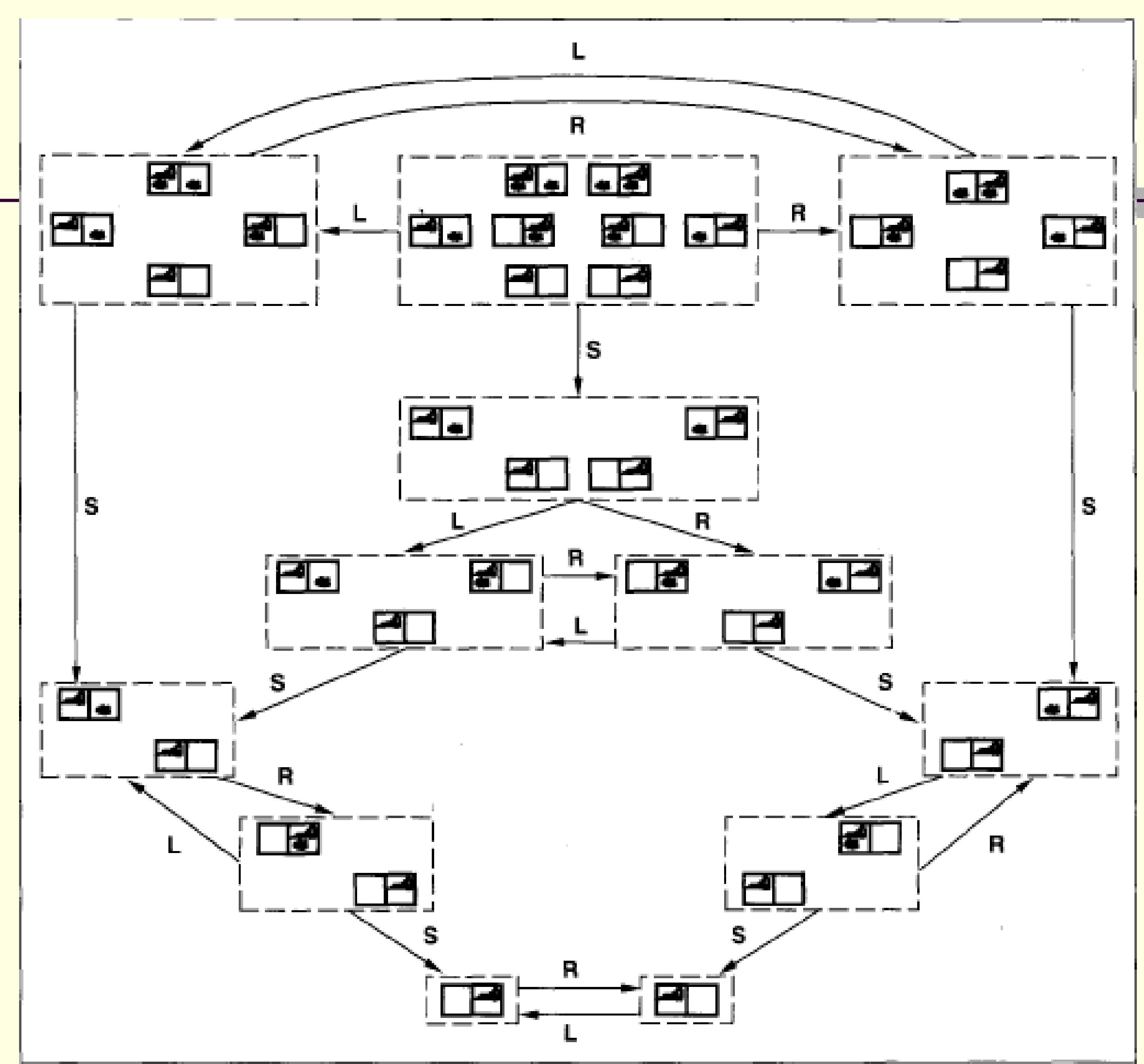
Aspiratorul cu stari multiple

- 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.

Catalin
Stoean

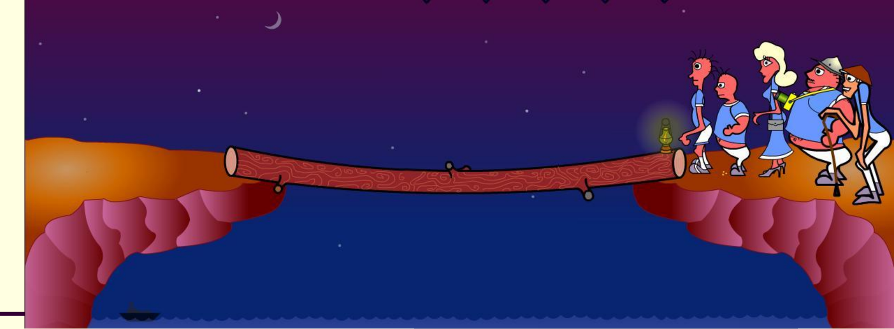


Misionarii si canibalii

- Trei misionari si trei canibali se afla de o parte a raului. Ei au o barca ce poate duce cel mult doi oameni. Gasiti o posibilitate sa traverseze toti de cealalta parte a raului cu conditia sa nu existe mai multi canibali decat misionari intr-o parte.

- **Stari**: secvente ordonate de 3 numere reprezentand numarul de misionari, canibali si barci de partea in care se aflau initial (3, 3, 1).
- **Actiuni**: mutarea unui misionar sau canibal sau 2 canibali, 2 misionari sau un misionar si un canibal de pe o parte pe alta.
- **Testarea tintei**: starea (0, 0, 0).
- **Costul drumului**: numarul de traversari.

Trecerea unui pod cu lampa - tema



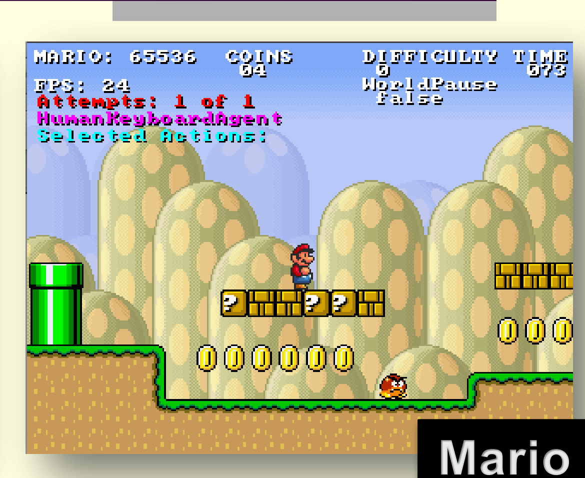
- Noapte, exista o singura lampa, 5 persoane trebuie sa treaca de pe un mal pe celalalt pe un pod care poate sustine doar doua persoane.
- Fiecare persoana trece podul cu o viteza diferita: 1 sec, 3 sec, 6 sec, 8 sec, 12 sec.
- Cand trec doua persoane, se deplaseaza ambele cu viteza celei care merge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

Alte platforme pentru algoritmi din IA

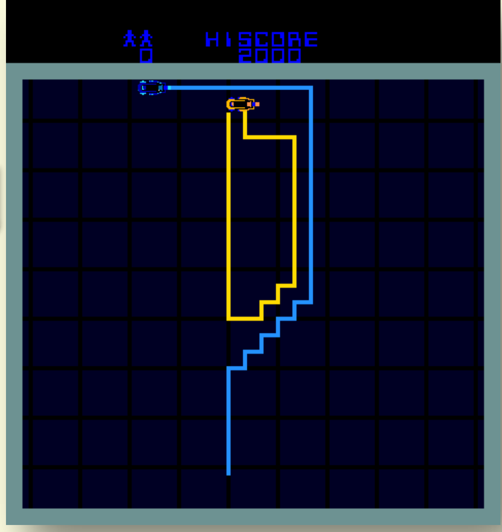
Robocode

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



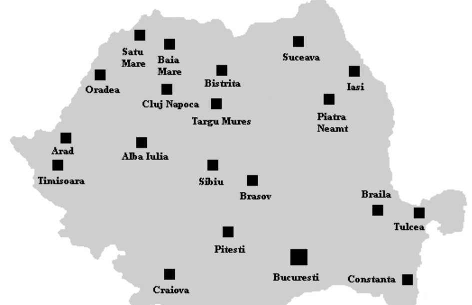
Mario



Tron

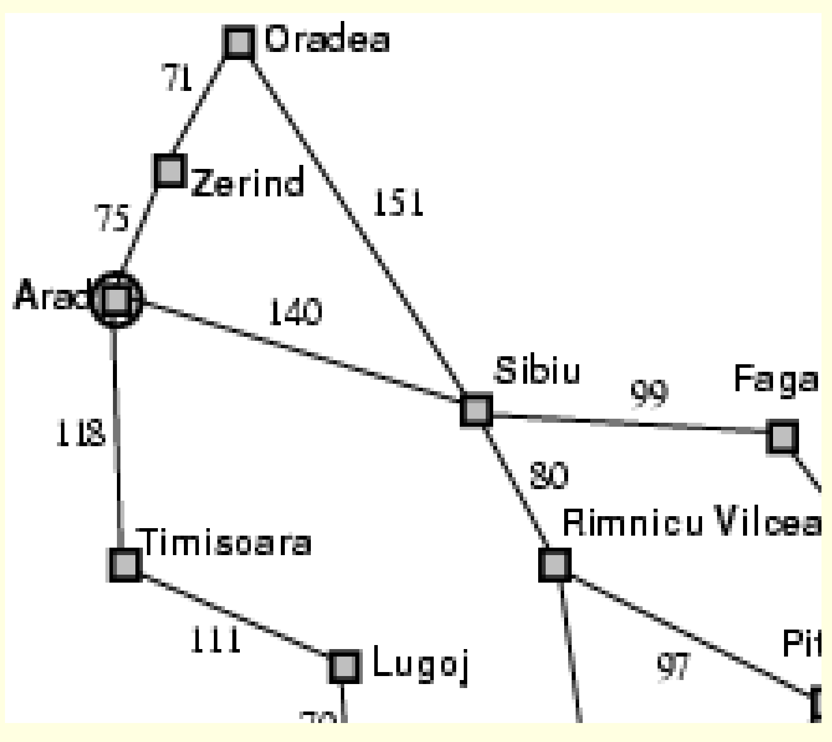
Torcs

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*.

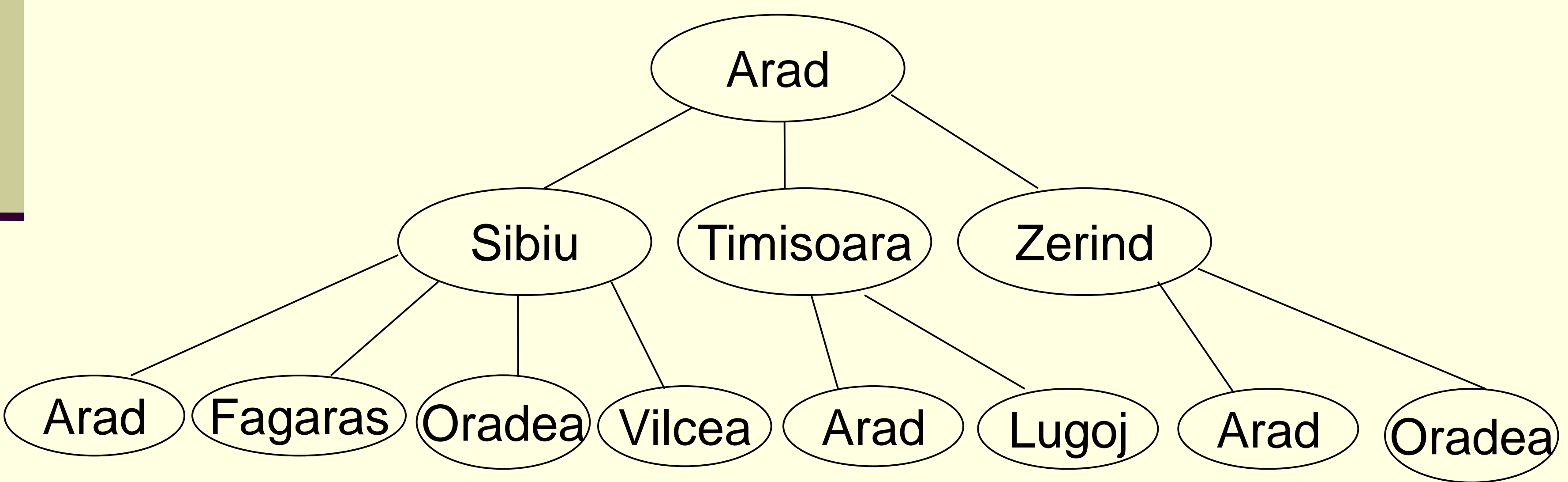
Algoritmi de cautare standard



- 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 ≠ ∅ *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 distinctie intre noduri si stari:
  - Starea reprezinta o configuratie a mediului;
  - Nodul contine informatii cu privire la structura arborelui de cautare.
- Colectia de noduri este implementata sub forma de **lista**. Operatii posibile:
  - genereaza_lista(*Elemente*) – creeaza o lista cu elementele date
  - goala(*lista*) – intoarce “adevarat” daca este goala lista
  - scoate_din_fata(*lista*) – intoarce elementul din fata
  - adauga(*Elemente, lista*) – introduce “Elemente”-le in “lista”

Algoritm general de cautare

functia cautare_generala(*problema*) ***intoarce*** solutie sau esec

noduri = genereaza_lista(genereaza_nod(stare_initiala[problema]))

Cat timp solutie negasita *si* noduri ≠ ∅ *executa*

Daca noduri = vida *atunci*

        intoarce esec

nod = scoate_din_fata(noduri)

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

        intoarce nod

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 repeta 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

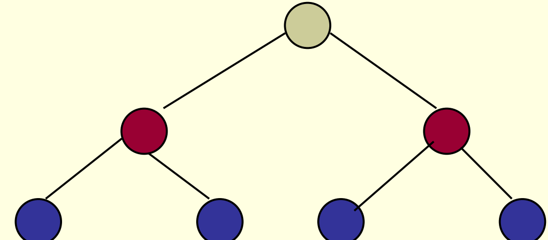
- Strategiile de cautare sunt analizate in functie de 4 criterii:
  - Completitudine
    - Garanteaza strategia gasirea unei solutii atunci cand exista una?
  - Complexitate temporala
    - Cat dureaza gasirea unei solutii?
  - Complexitatea spatiala
    - De cata memorie este nevoie pentru a face cautarea?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

Algoritmi de cautare standard

- Algoritmii studiati 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...

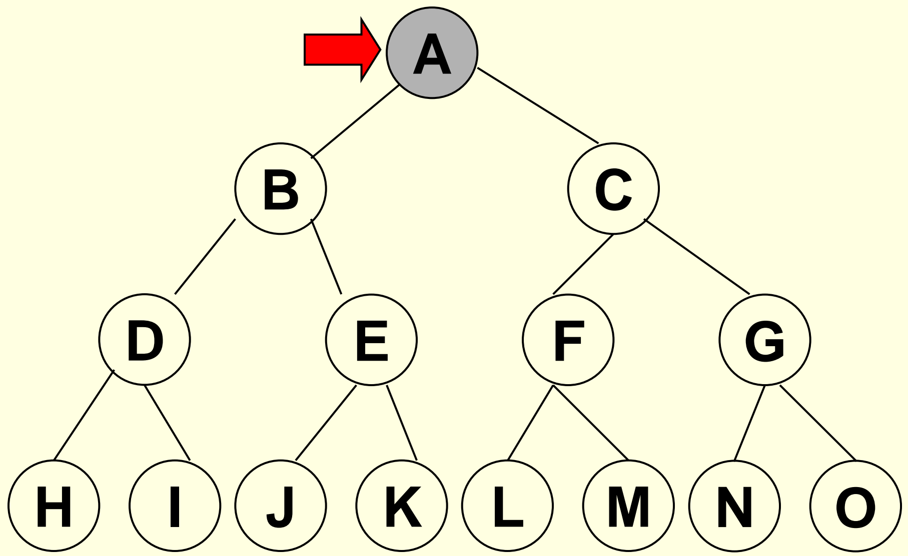
Cautarea in latime

- 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.

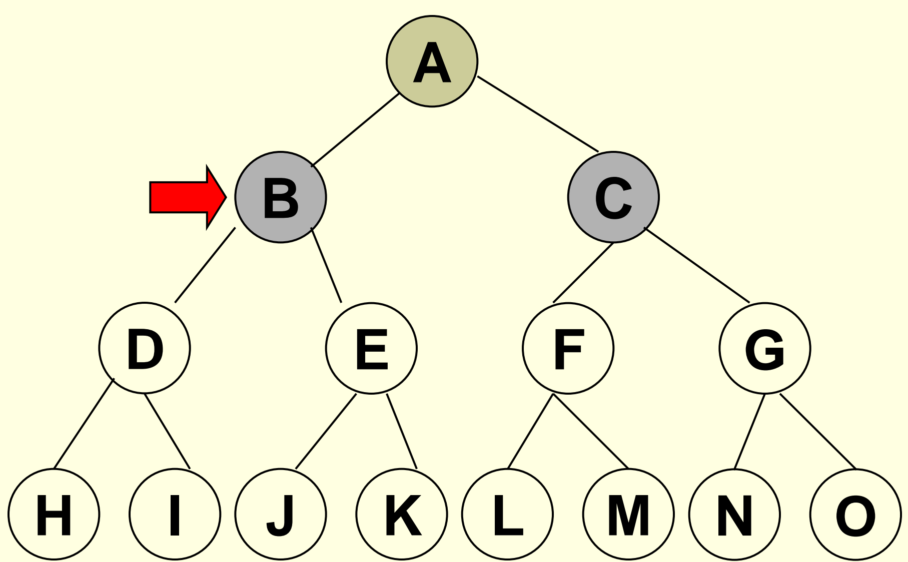


- Daca exista o solutie, algoritmul garanteaza ca o va gasi, iar daca exista mai multe solutii, algoritmul va gasi intai starea tinta aflata la cel mai mic numar de noduri.

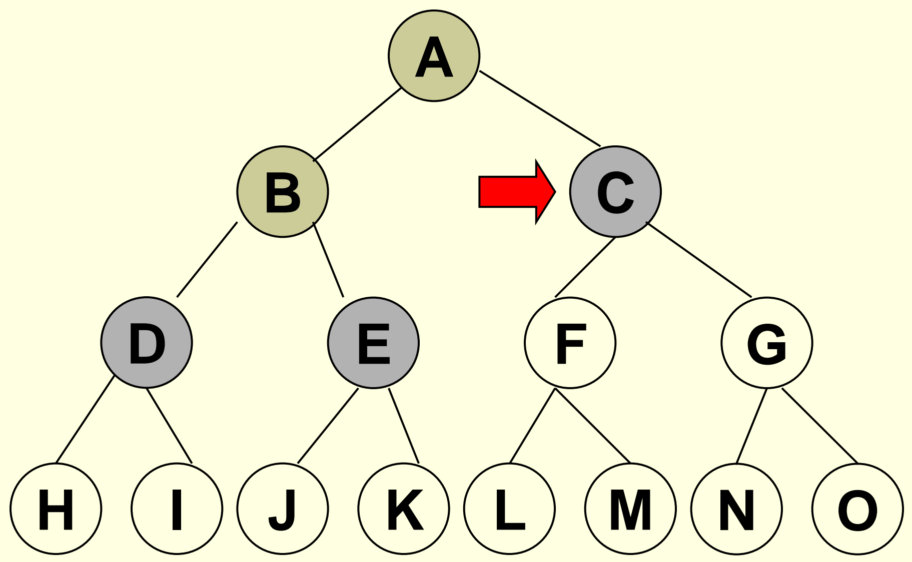
Cautarea in latime



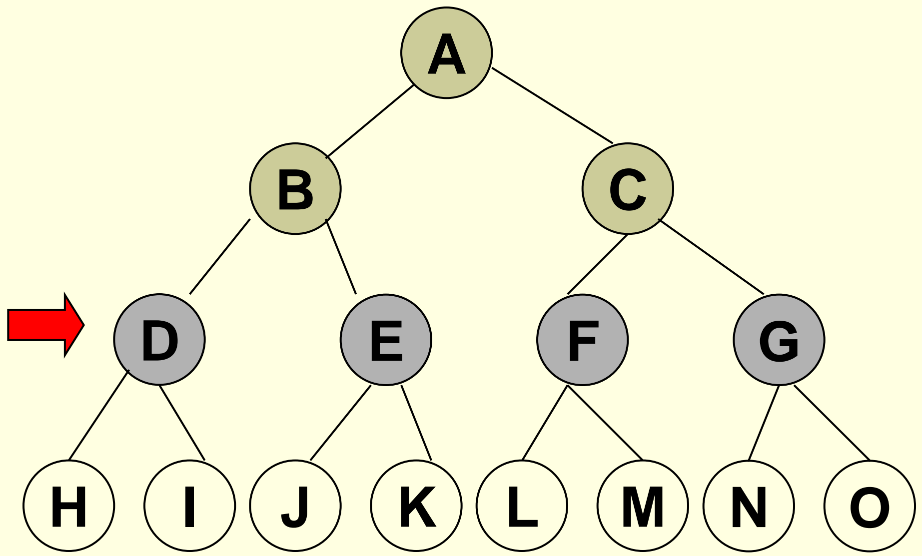
Cautarea in latime



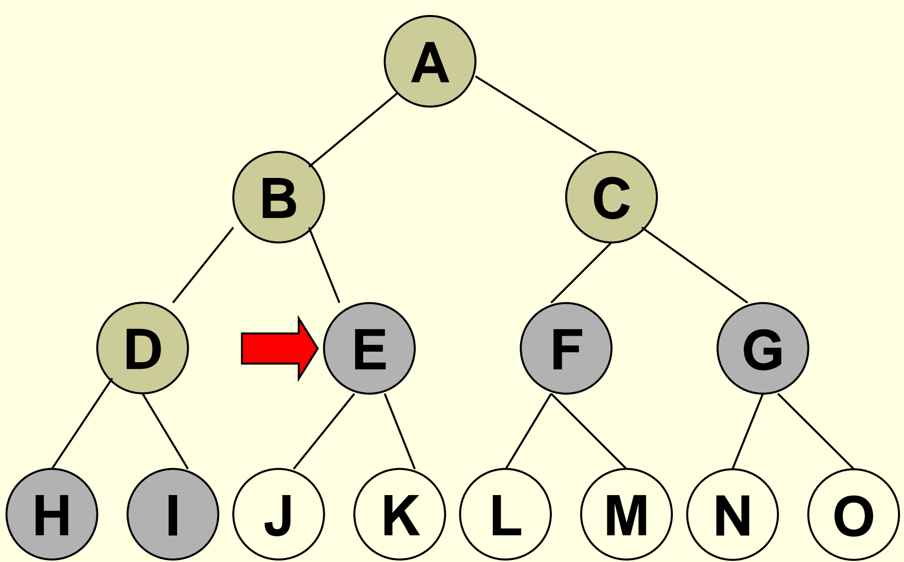
Cautarea in latime



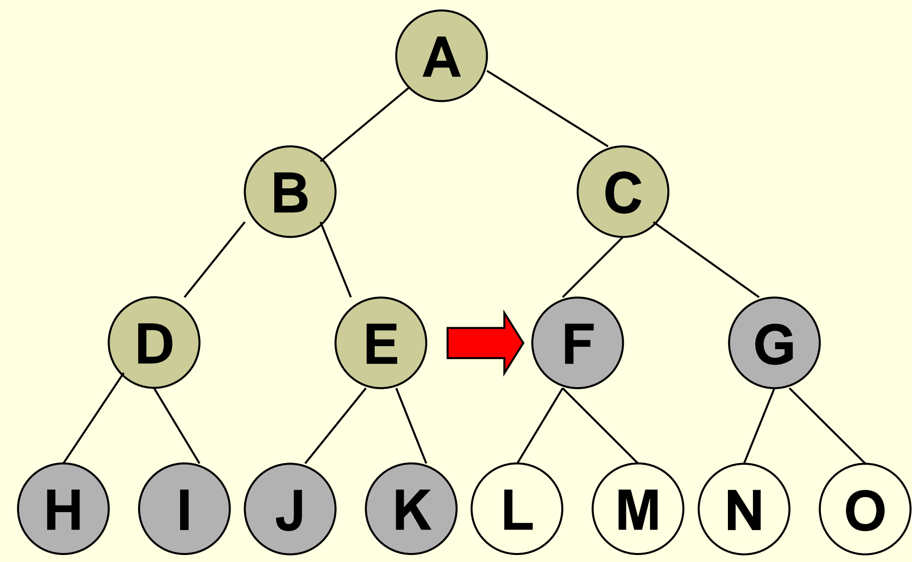
Cautarea in latime



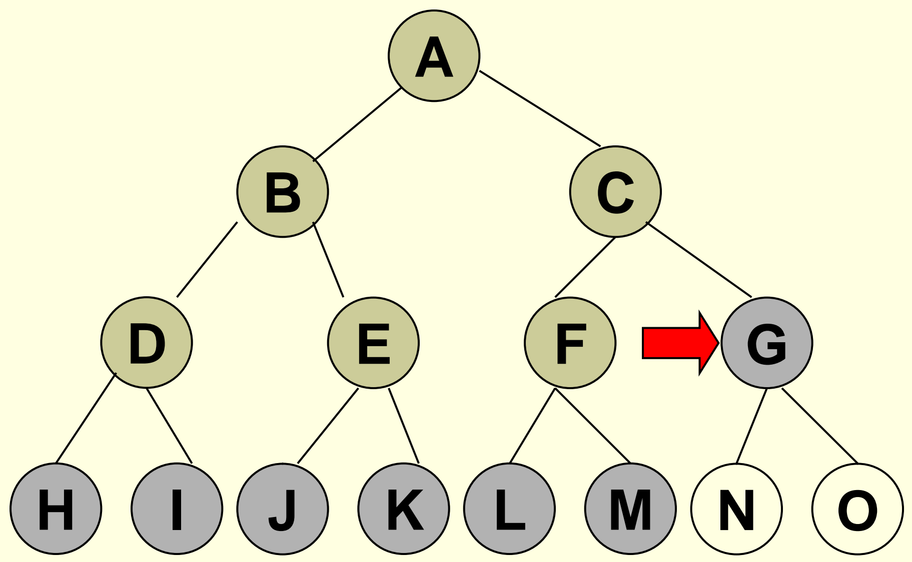
Cautarea in latime



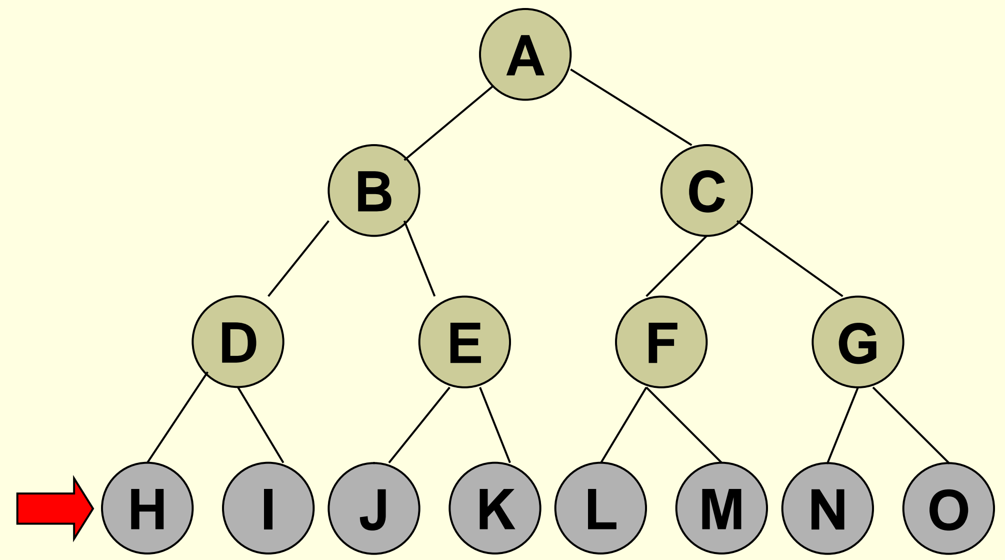
Cautarea in latime



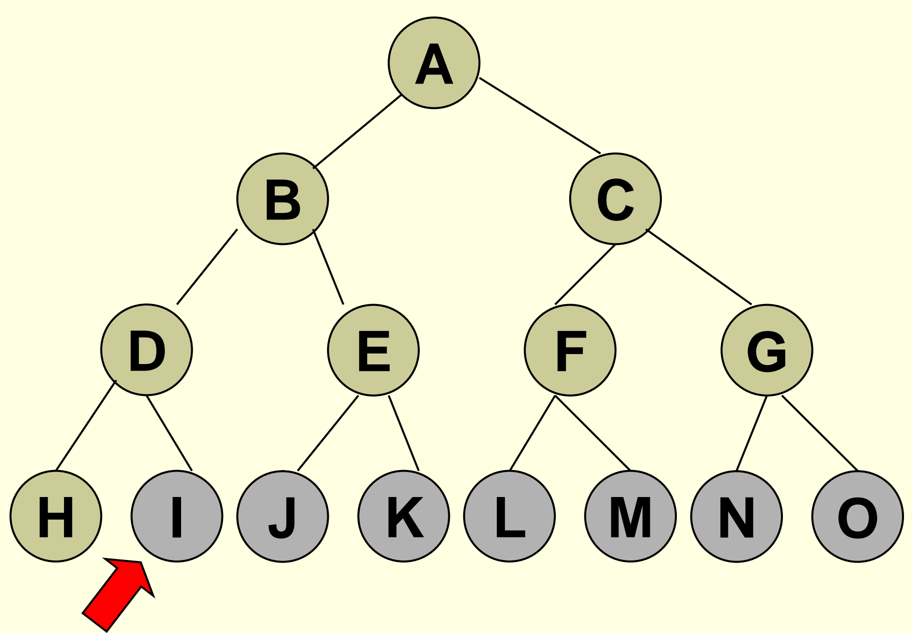
Cautarea in latime



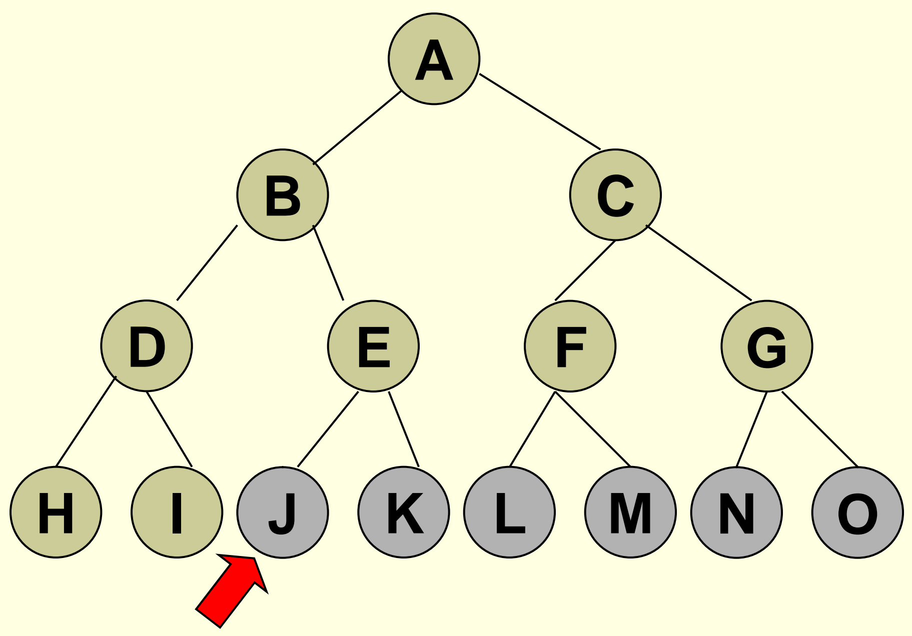
Cautarea in latime



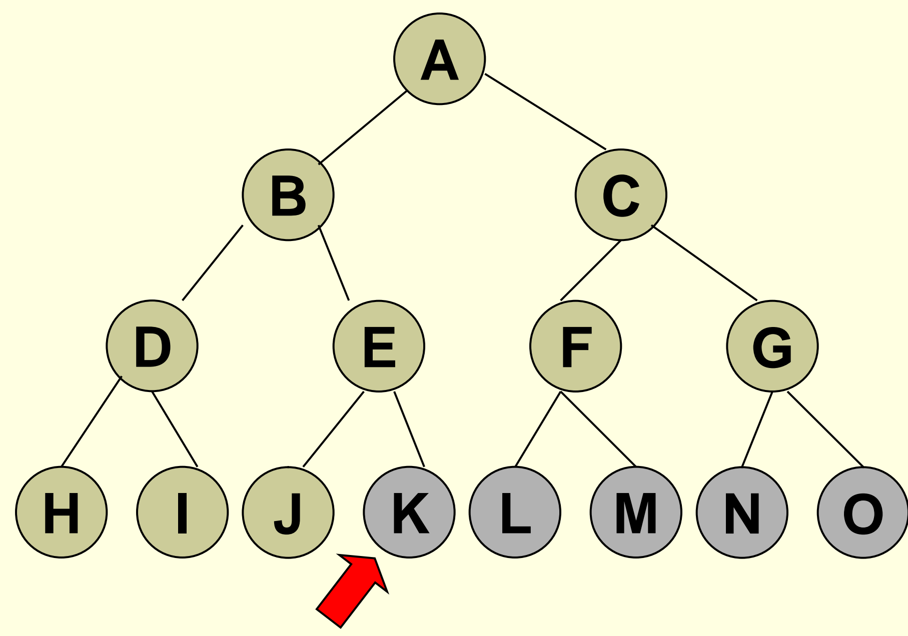
Cautarea in latime



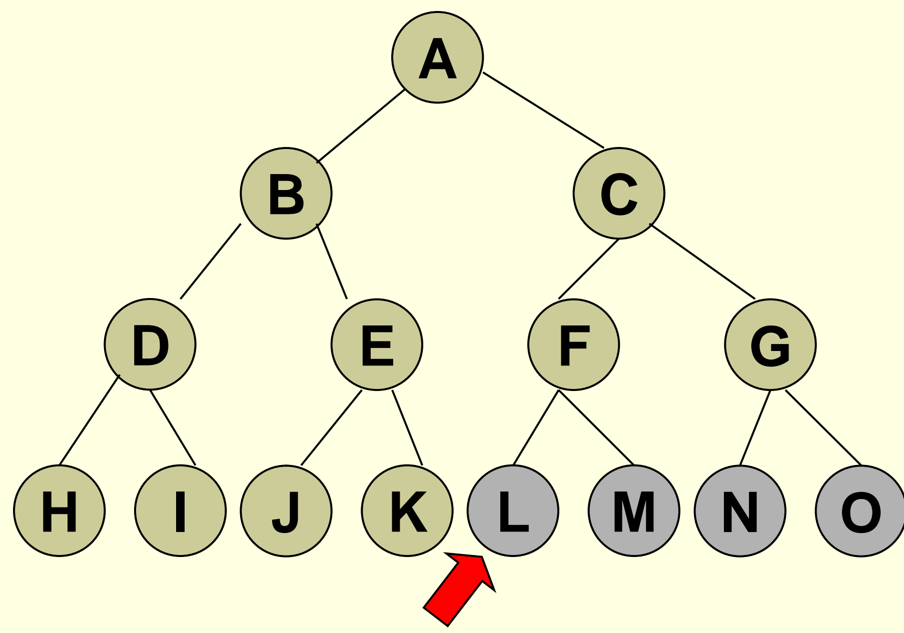
Cautarea in latime



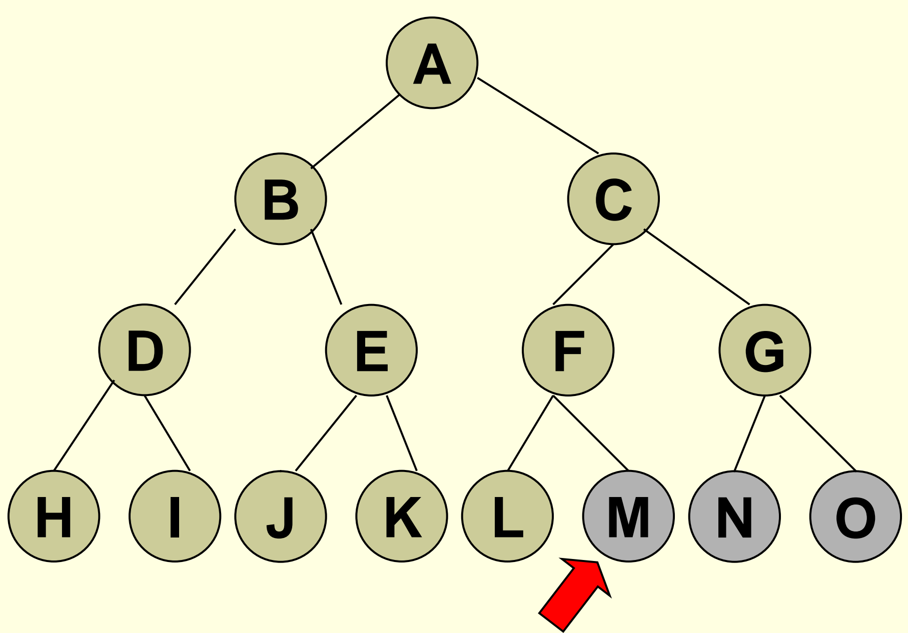
Cautarea in latime



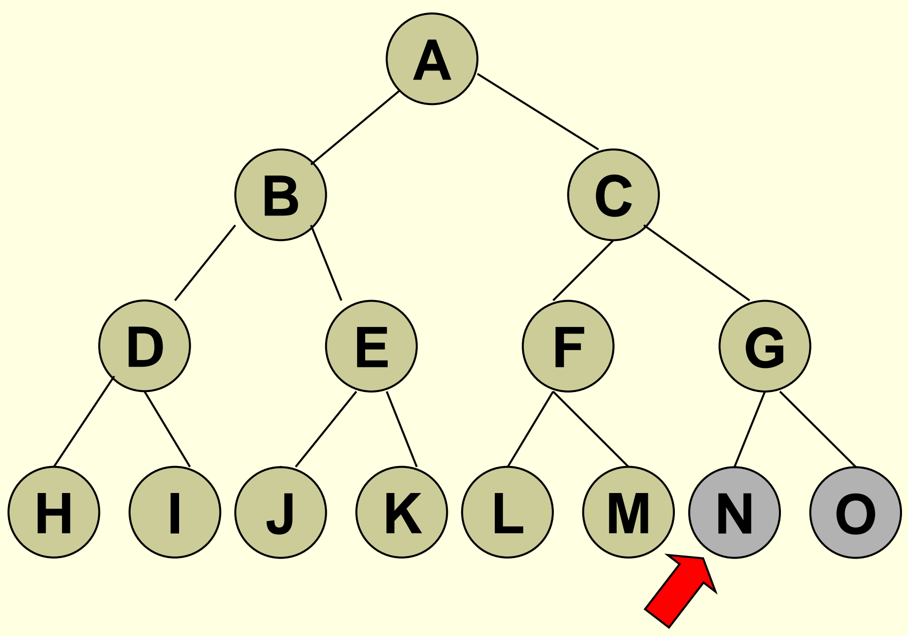
Cautarea in latime



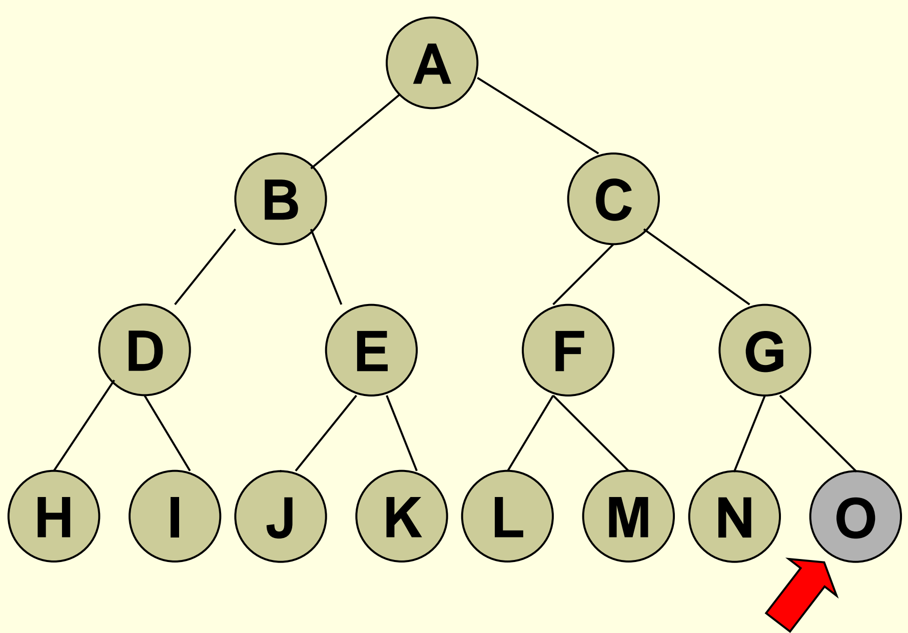
Cautarea in latime



Cautarea in latime



Cautarea in latime



Algoritm cautare in latime

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

Cat timp solutie negasita si noduri ≠ ∅ *executa*
Daca noduri = vida *atunci*
    intoarce esec
nod = scoate_din_fata(noduri)
Daca testare_tinta[problema] se aplica la stare(nod) *atunci*
    intoarce nod
Altfel
  noduri = adauga(noduri, expandare(nod, adauga_la_sfarsit))
Sfarsit cat timp

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:

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

- Complexitatea $O(b^d)$.
- Consideram un exemplu in care $b = 10$ si vom urmari diverse valori pentru *d*.
- Consideram ca se proceseaza 1000 de noduri pe secunda.
- Un nod se reprezinta pe 100 de bytes.

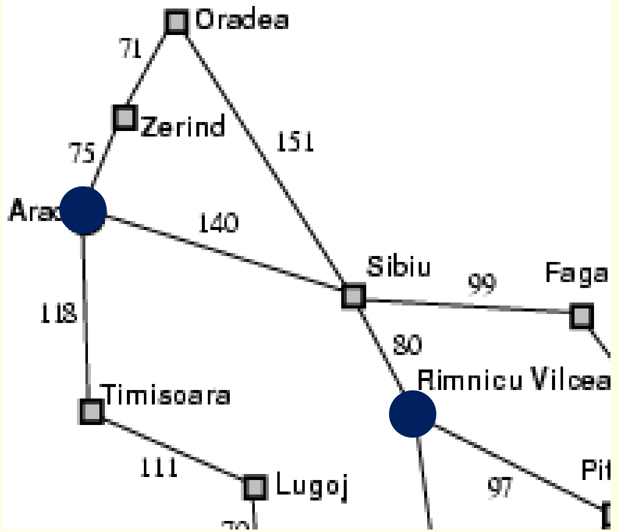
Cautarea in latime

Adancime	Noduri	Timp	Memorie
0	1	1 milisecunde	100 bytes
2	111	0.1 secunde	11 kilobytes
4	11 111	11 secunde	1 megabyte
6	10^6	18 minute	111 megabytes
8	10^8	31 ore	11 gigabytes
10	10^{10}	128 zile	1 terabyte
12	10^{12}	35 ani	111 terabytes
14	10^{14}	3 500 ani	11 111 terabytes

Cautarea cu cost uniform

- Este echivalenta cu cautarea in latime daca toate costurile sunt egale.
- Extinde mereu nodul cu costul minim.
- Solutia cu cost minim va fi garantat gasita pentru ca daca exista o cale cu un cost mai mic aceasta este aleasa.

Vrem sa ajungem de la Arad la Rimnicu Vilcea.



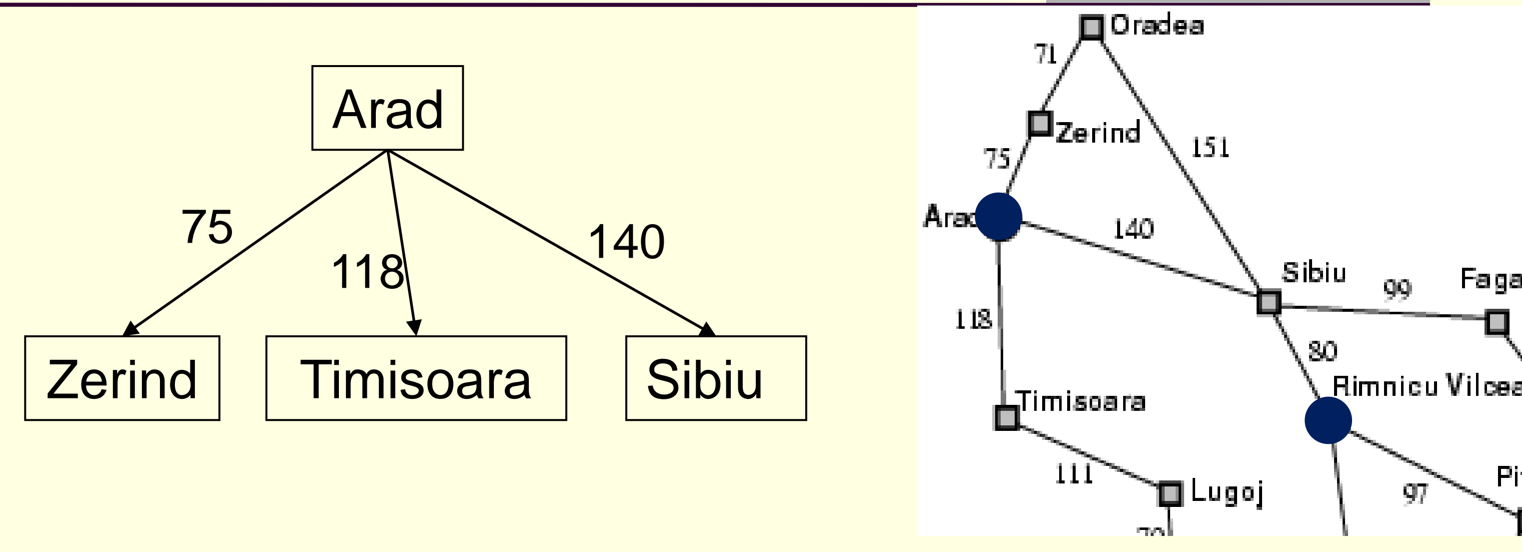
Algoritm cautare cu cost uniform

functia cautare_cost_uniform(*problema*) ***intoarce*** solutie sau esec
noduri = genereaza_lista(genereaza_nod(stare_initiala[problema]))

Cat timp solutie negasita si noduri ≠ ∅ *executa*
Daca noduri = vida *atunci*
    intoarce esec
nod = scoate_din_fata(noduri)
Daca testare_tinta[problema] se aplica la stare(nod) *atunci*
    intoarce nod
Altfel
  noduri = adauga(noduri, expandare(nod, ordoneaza_dupa_cost))
Sfarsit cat timp

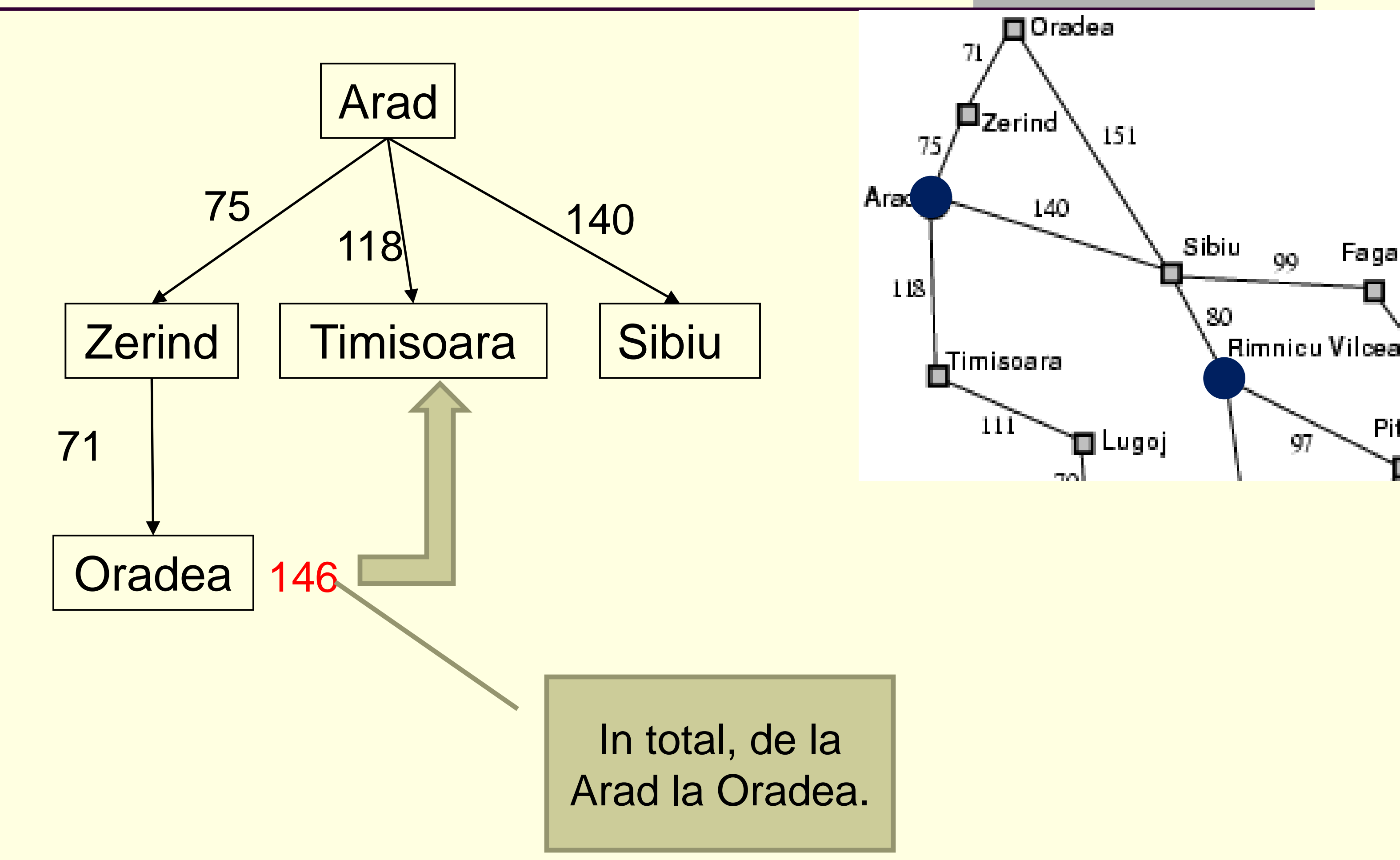
Se permit noduri duplicate, insa si ele sunt ordonate in functie de cost.

Cautarea cu cost uniform

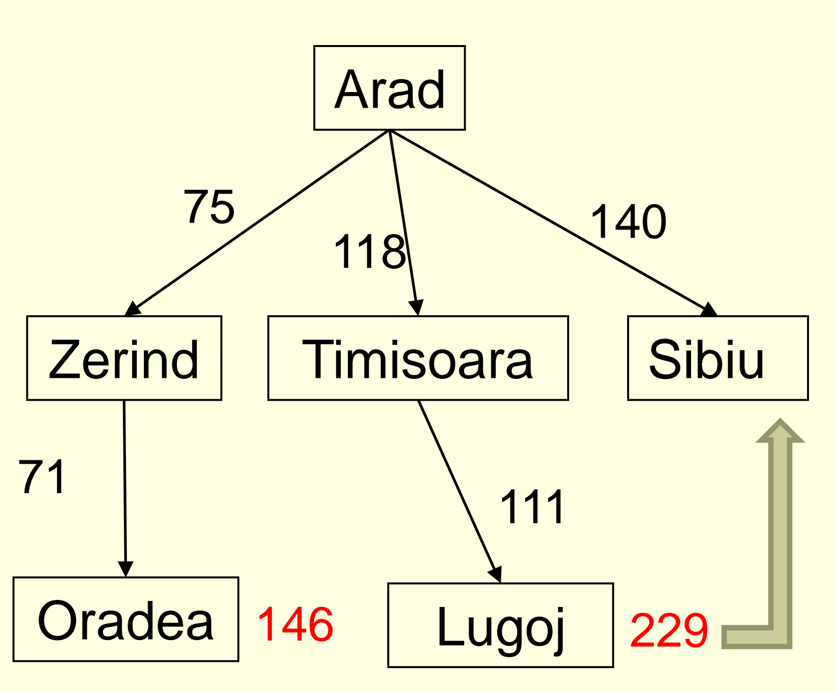


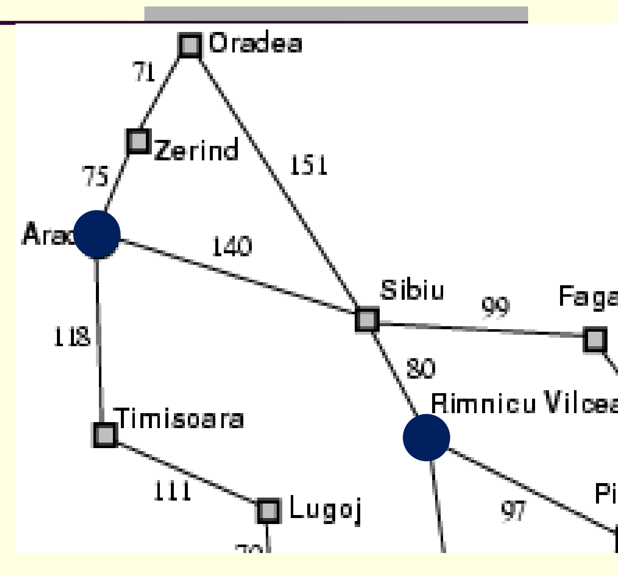
Descendentii sunt asezati in ordine crescatoare in functie de distanta fata de nodul curent.

Cautarea cu cost uniform

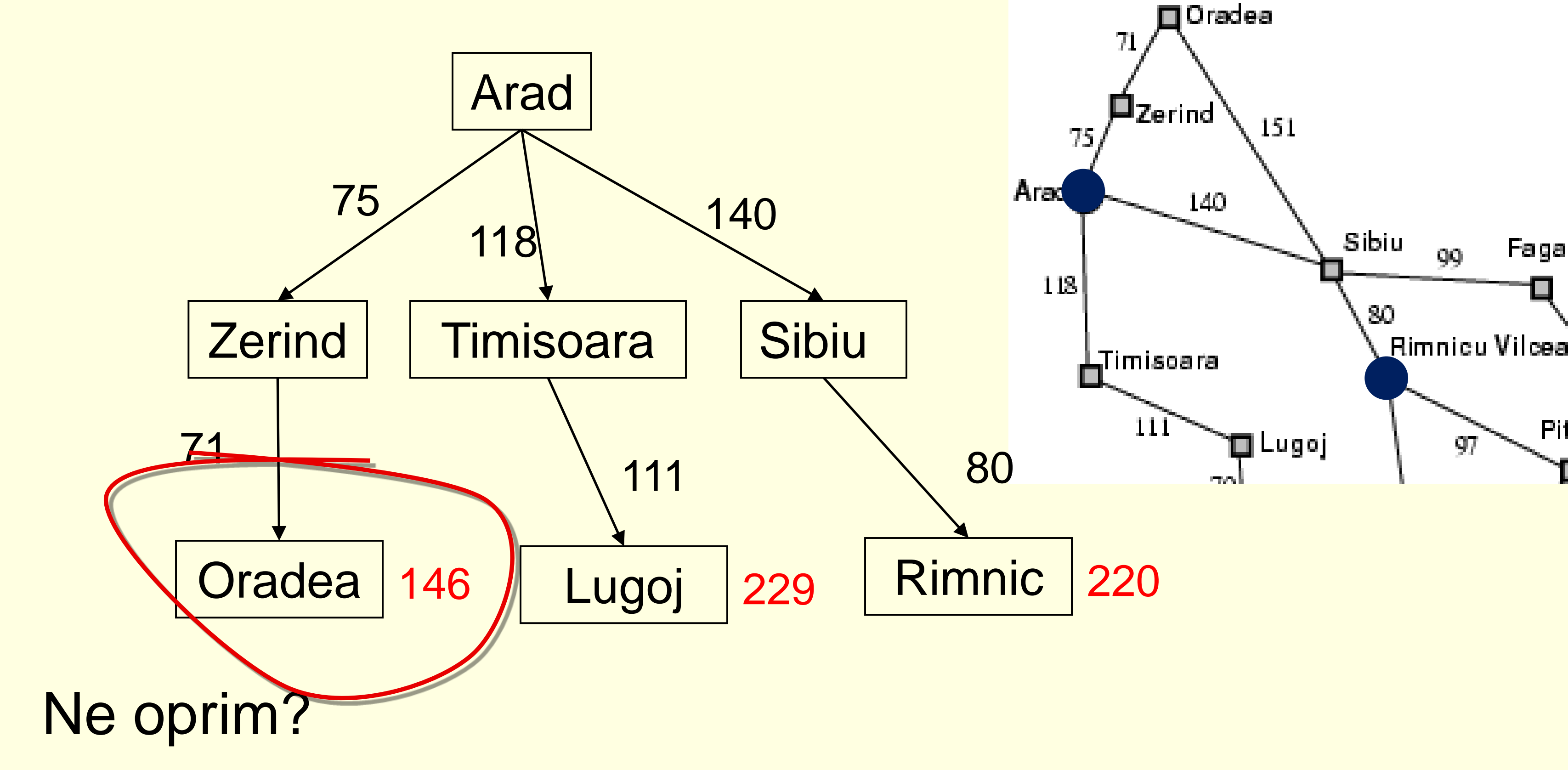


Cautarea cu cost uniform





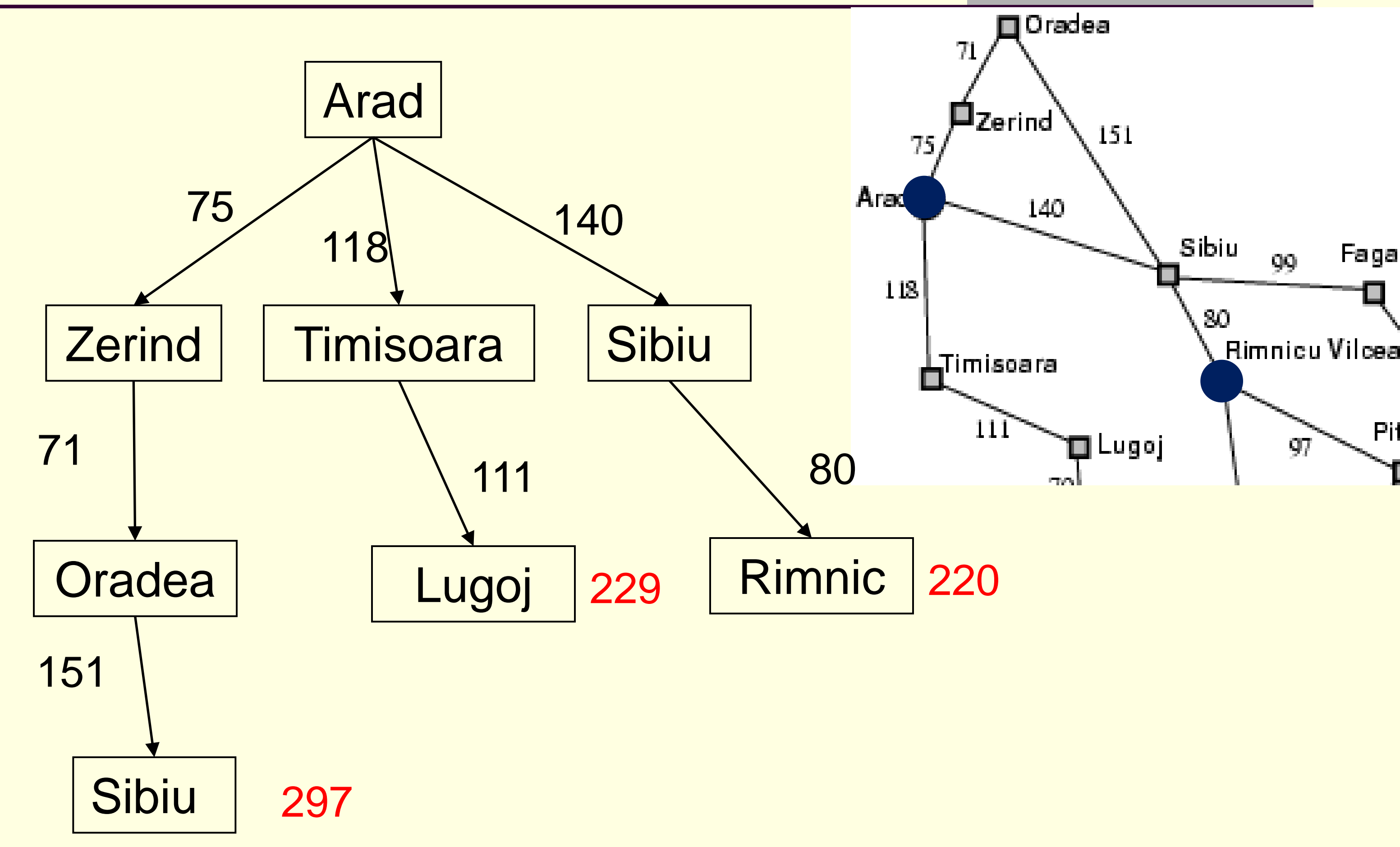
Cautarea cu cost uniform



Ne oprim?

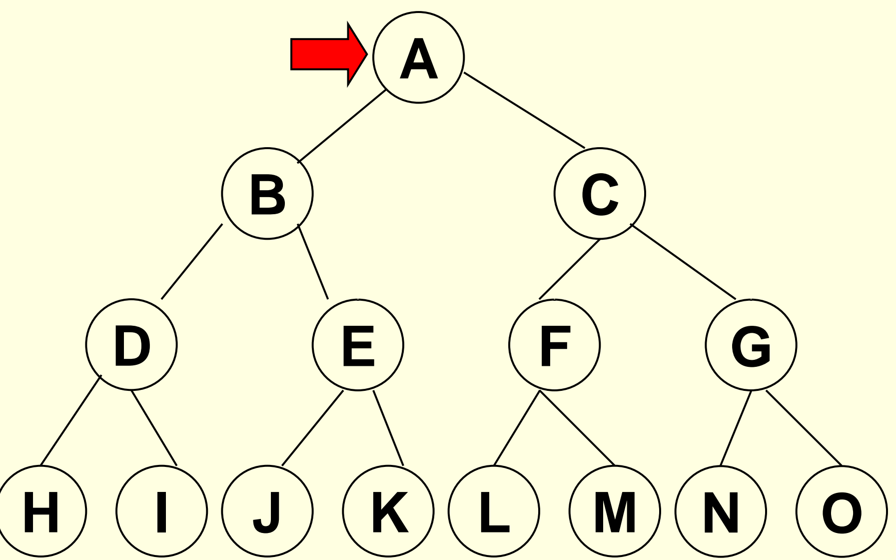
Nu, pana cand nu este depasita valoarea 220 pe toate rutele posibile.

Cautarea cu cost uniform



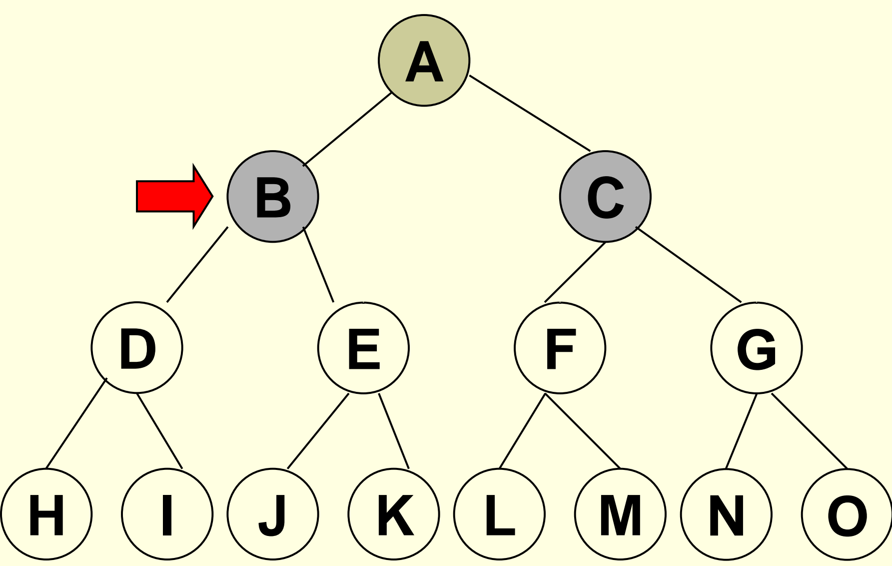
Cautarea in adancime

- Se expandeaza nodul radacina, apoi se merge pe un drum pana se ajunge la cel mai adanc nivel al arborelui.
- Numai cand se ajunge la final (la nodurile frunza), cautarea se intoarce si expandeaza noduri de la nivele mai putin adanci.



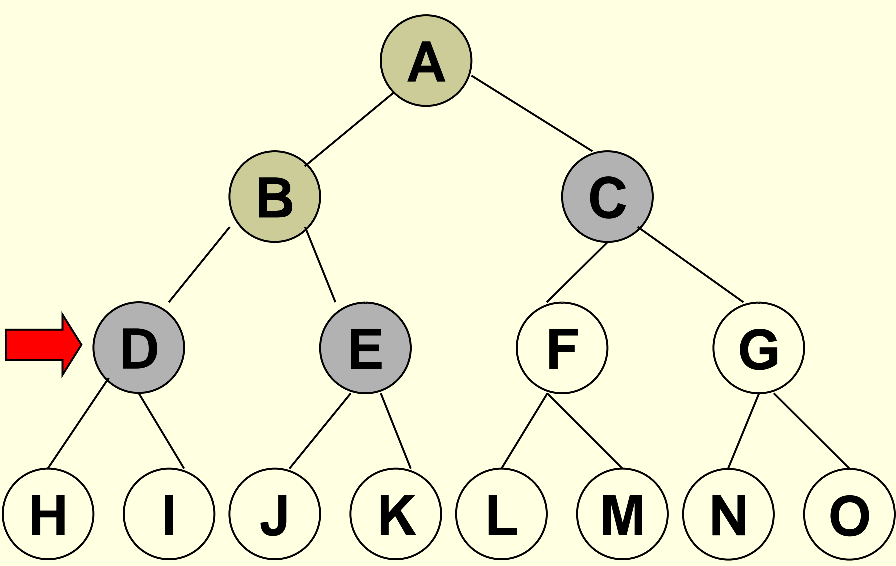
Cautarea in adancime

- Se expandeaza nodul radacina, apoi se merge pe un drum pana se ajunge la cel mai adanc nivel al arborelui.
- Numai cand se ajunge la final (la nodurile frunza), cautarea se intoarce si expandeaza noduri de la nivele mai putin adanci.



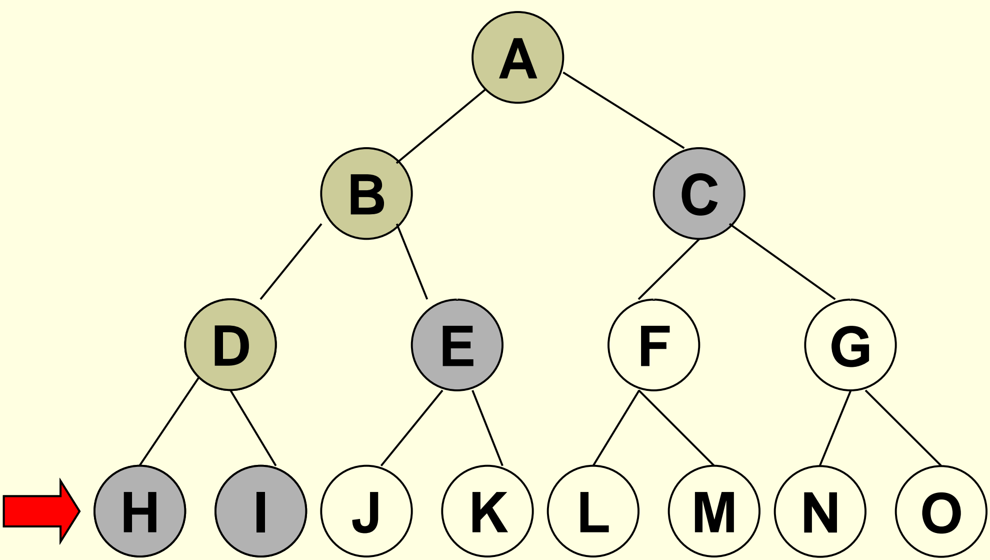
Cautarea in adancime

- Se expandeaza nodul radacina, apoi se merge pe un drum pana se ajunge la cel mai adanc nivel al arborelui.
- Numai cand se ajunge la final (la nodurile frunza), cautarea se intoarce si expandeaza noduri de la nivele mai putin adanci.



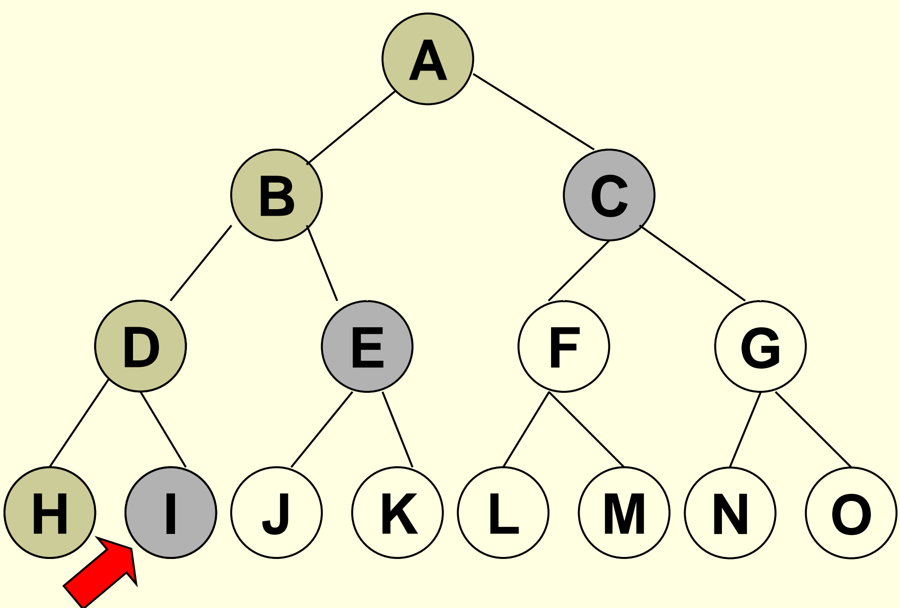
Cautarea in adancime

- Se expandeaza nodul radacina, apoi se merge pe un drum pana se ajunge la cel mai adanc nivel al arborelui.
- Numai cand se ajunge la final (la nodurile frunza), cautarea se intoarce si expandeaza noduri de la nivele mai putin adanci.



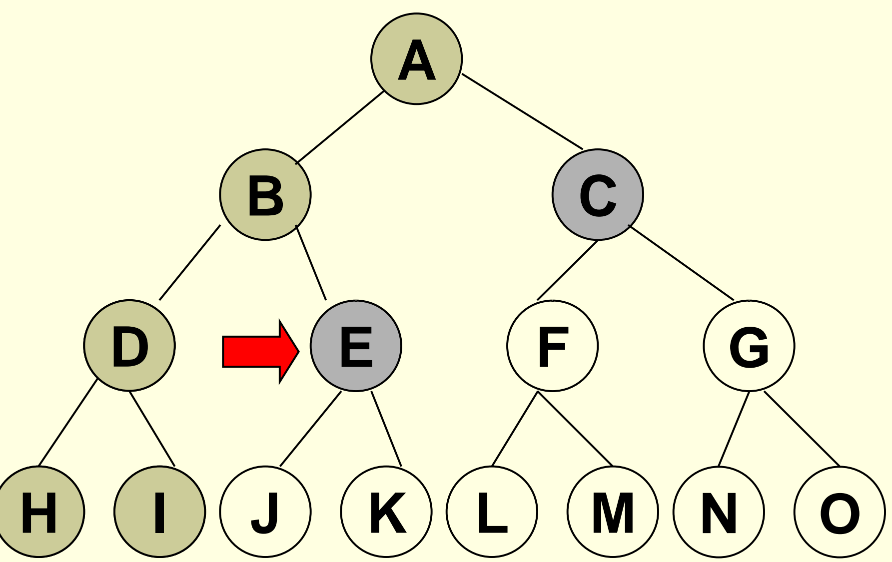
Cautarea in adancime

- Se expandeaza nodul radacina, apoi se merge pe un drum pana se ajunge la cel mai adanc nivel al arborelui.
- Numai cand se ajunge la final (la nodurile frunza), cautarea se intoarce si expandeaza noduri de la nivele mai putin adanci.



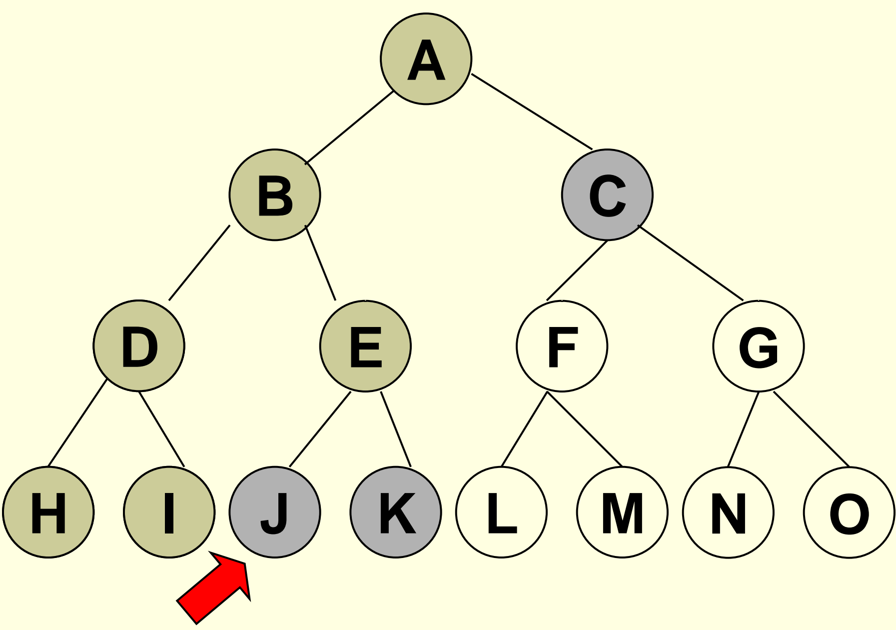
Cautarea in adancime

- Se expandeaza nodul radacina, apoi se merge pe un drum pana se ajunge la cel mai adanc nivel al arborelui.
- Numai cand se ajunge la final (la nodurile frunza), cautarea se intoarce si expandeaza noduri de la nivele mai putin adanci.



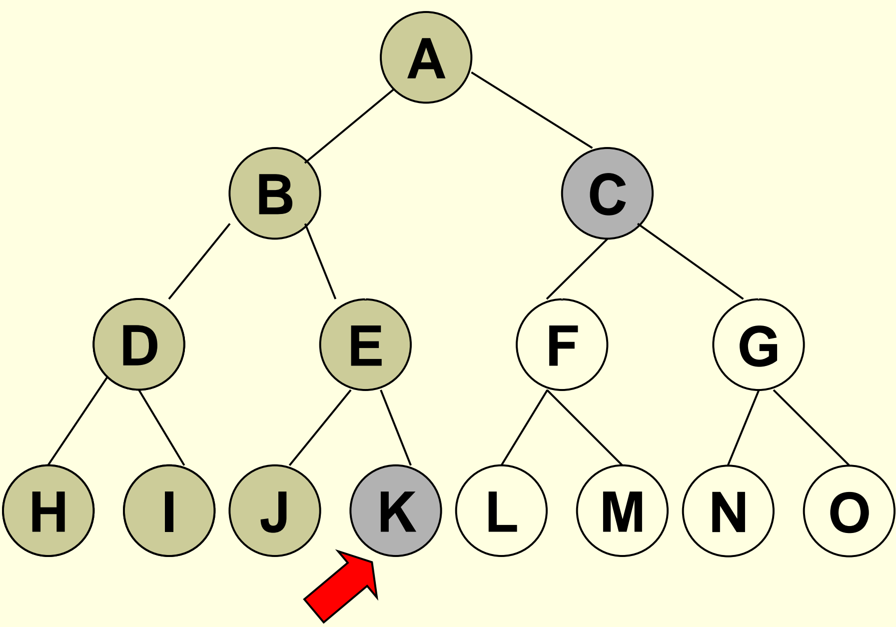
Cautarea in adancime

- Se expandeaza nodul radacina, apoi se merge pe un drum pana se ajunge la cel mai adanc nivel al arborelui.
- Numai cand se ajunge la final (la nodurile frunza), cautarea se intoarce si expandeaza noduri de la nivele mai putin adanci.



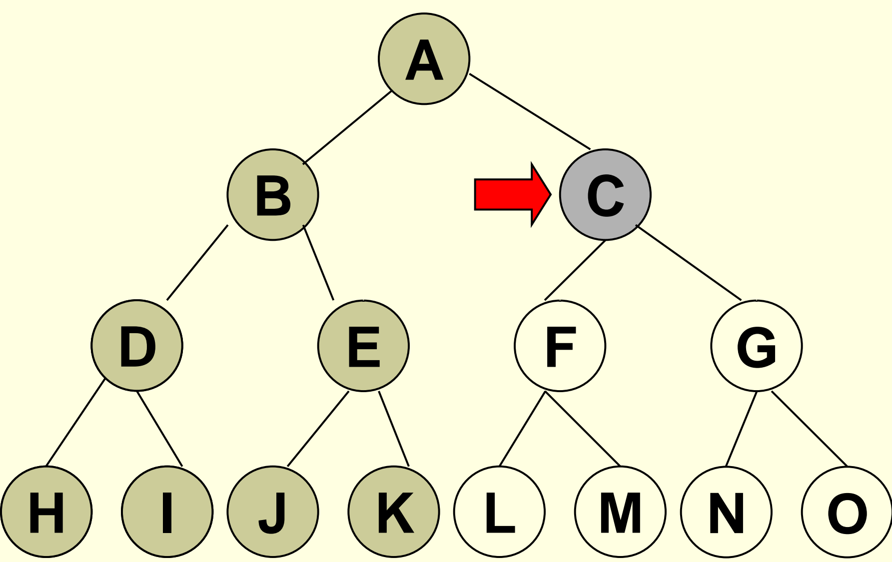
Cautarea in adancime

- Se expandeaza nodul radacina, apoi se merge pe un drum pana se ajunge la cel mai adanc nivel al arborelui.
- Numai cand se ajunge la final (la nodurile frunza), cautarea se intoarce si expandeaza noduri de la nivele mai putin adanci.



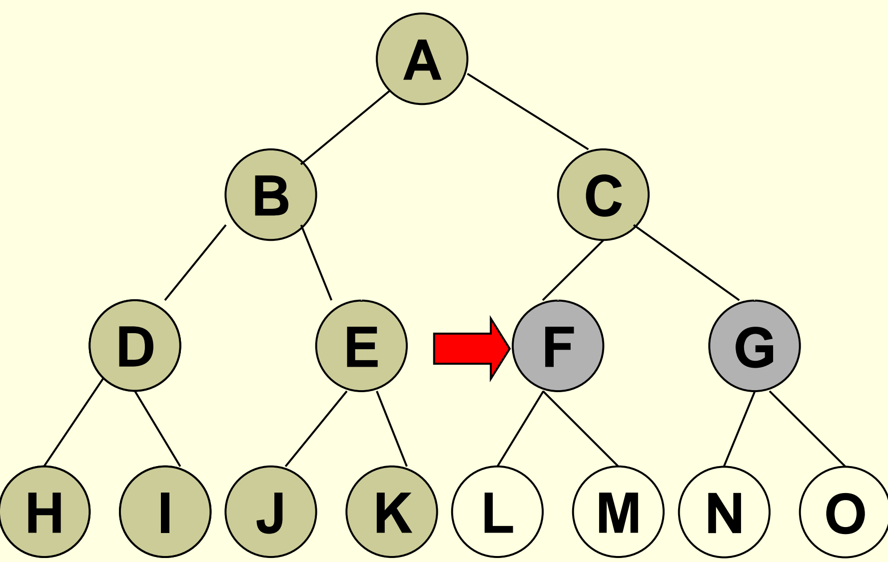
Cautarea in adancime

- Se expandeaza nodul radacina, apoi se merge pe un drum pana se ajunge la cel mai adanc nivel al arborelui.
- Numai cand se ajunge la final (la nodurile frunza), cautarea se intoarce si expandeaza noduri de la nivele mai putin adanci.



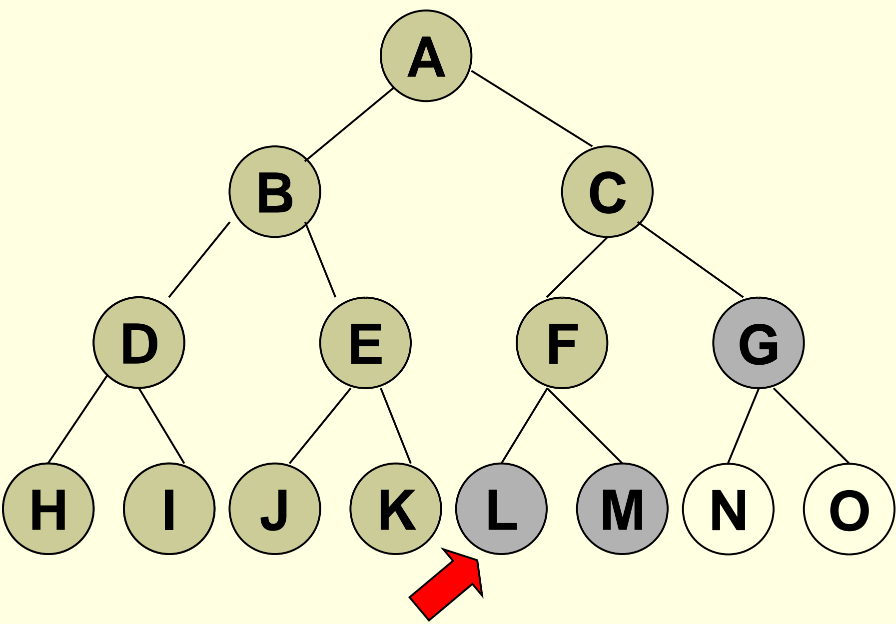
Cautarea in adancime

- Se expandeaza nodul radacina, apoi se merge pe un drum pana se ajunge la cel mai adanc nivel al arborelui.
- Numai cand se ajunge la final (la nodurile frunza), cautarea se intoarce si expandeaza noduri de la nivele mai putin adanci.



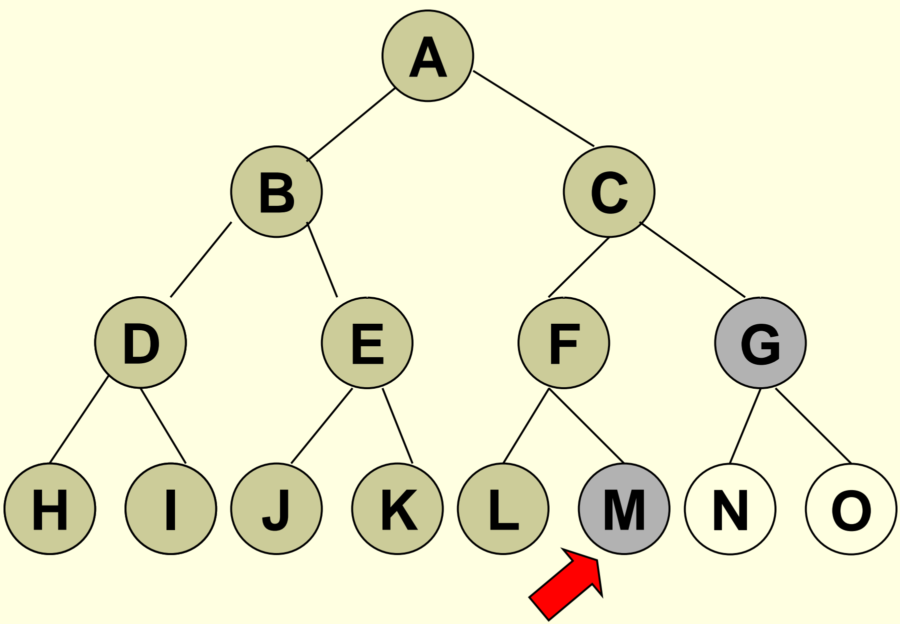
Cautarea in adancime

- Se expandeaza nodul radacina, apoi se merge pe un drum pana se ajunge la cel mai adanc nivel al arborelui.
- Numai cand se ajunge la final (la nodurile frunza), cautarea se intoarce si expandeaza noduri de la nivele mai putin adanci.



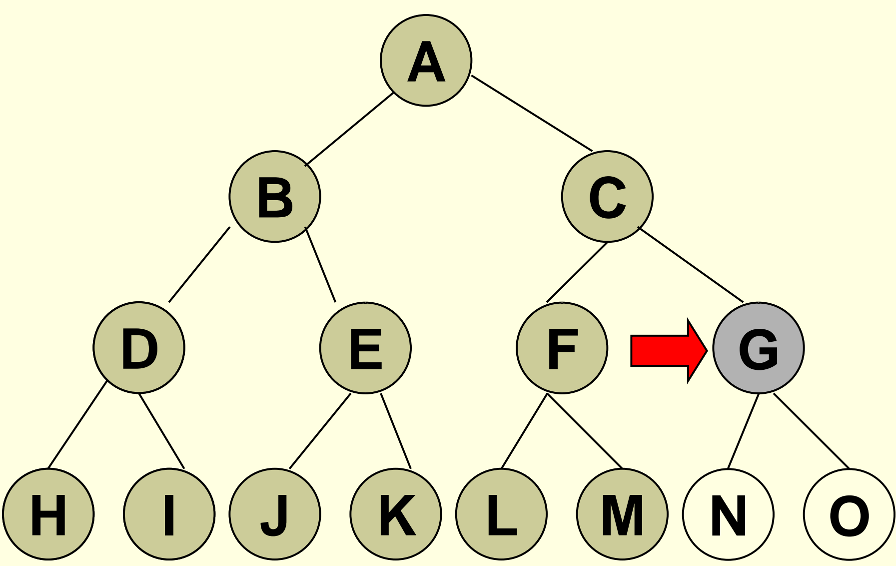
Cautarea in adancime

- Se expandeaza nodul radacina, apoi se merge pe un drum pana se ajunge la cel mai adanc nivel al arborelui.
- Numai cand se ajunge la final (la nodurile frunza), cautarea se intoarce si expandeaza noduri de la nivele mai putin adanci.



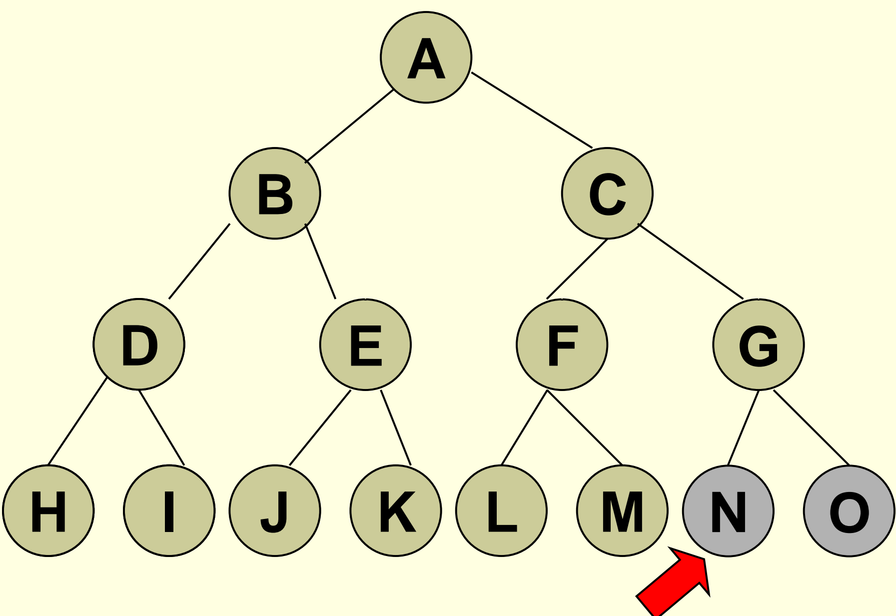
Cautarea in adancime

- Se expandeaza nodul radacina, apoi se merge pe un drum pana se ajunge la cel mai adanc nivel al arborelui.
- Numai cand se ajunge la final (la nodurile frunza), cautarea se intoarce si expandeaza noduri de la nivele mai putin adanci.



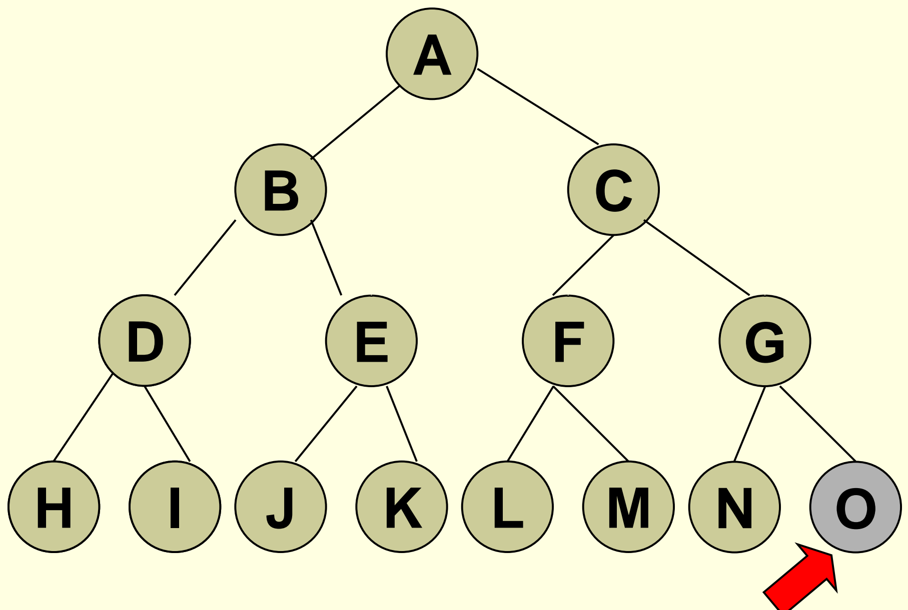
Cautarea in adancime

- Se expandeaza nodul radacina, apoi se merge pe un drum pana se ajunge la cel mai adanc nivel al arborelui.
- Numai cand se ajunge la final (la nodurile frunza), cautarea se intoarce si expandeaza noduri de la nivele mai putin adanci.



Cautarea in adancime

- Se expandeaza nodul radacina, apoi se merge pe un drum pana se ajunge la cel mai adanc nivel al arborelui.
- Numai cand se ajunge la final (la nodurile frunza), cautarea se intoarce si expandeaza noduri de la nivele mai putin adanci.



Parcurgerea in adancime: A, B, D, H, I, E, J, K, C, F, L, M, G, N, O.

Algoritm cautarea in adancime

functia cautare_adancime(*problema*) ***intoarce*** solutie sau esec
noduri = genereaza_lista(genereaza_nod(stare_initiala[problema]))

Cat timp solutie negasita si noduri ≠ ∅ *executa*
Daca noduri = vida *atunci*
    intoarce esec
nod = scoate_din_fata(noduri)
Daca testare_tinta[problema] se aplica la stare(nod) *atunci*
    intoarce nod
Altfel
  noduri = adauga(noduri, expandare(nod, adauga_la_inceput))
Sfarsit cat timp

Cautarea in adancime

- Nu necesita multa memorie – stocheaza un singur drum de la radacina la o frunza impreuna cu nodurile neexpandate.
- Daca fiecare nod genereaza *b* noduri si adancimea maxima este *m*, cautarea in adancime va stoca la un moment dat maximum *bm* noduri (fata de b^d, in cazul cautarii in latime).
  - Pentru $d = 12$, cand cautarea in latime necesita 111 terabytes, pentru cautarea in adancime este nevoie doar de 12 kilobytes.
- Complexitatea temporala – $O(b^m)$.
  - Daca sunt multe solutii, sunt sanse sa fie gasita mai rapid una decat in cazul cautarii in latime.

Cautarea in adancime

- Dezavantaj: se poate bloca daca porneste pe un drum gresit.
- Poate nimeri pe un drum infinit si nu va gasi astfel solutii care se pot gasi pe un alt drum la o distanta mica de radacina; intr-un astfel de caz nu va intoarce nici o solutie!
- Poate gasi o solutie care este mult mai costisitoare in comparatie cu alte solutii existente.
- **A nu se folosi la arbori care au adancimi foarte mari!**

Cautarea limitata in adancime

- Impune o margine superioara pentru lungimea unui drum.
- Se poate utiliza la probleme unde stim la ce adancime maxima trebuie sa gasim solutia
  - Ex: avem 20 de orase, ne aflam in orasul A, solutia trebuie sa se gaseasca la maxim 19 pasi.
- Daca l este limita de adancime stabilita, atunci complexitatile:
  - Pentru timp: $O(b^l)$
  - Pentru spatiu: $O(bl)$.

Algoritm cautarea limitata in adancime

functia cautare_adancime_limitata(*problema*) ***intoarce*** solutie sau esec
noduri = genereaza_lista(genereaza_nod(stare_initiala[problema]))

Cat timp solutie negasita si noduri ≠ ∅ *executa*
Daca noduri = vida *atunci*
    intoarce esec
nod = scoate_din_fata(noduri)
Daca testare_tinta[problema] se aplica la stare(nod) *atunci*
    intoarce nod
Altfel
  noduri = adauga(noduri, expandare(nod, adauga_la_inceput_daca_
    limita_permite))
Sfarsit cat timp

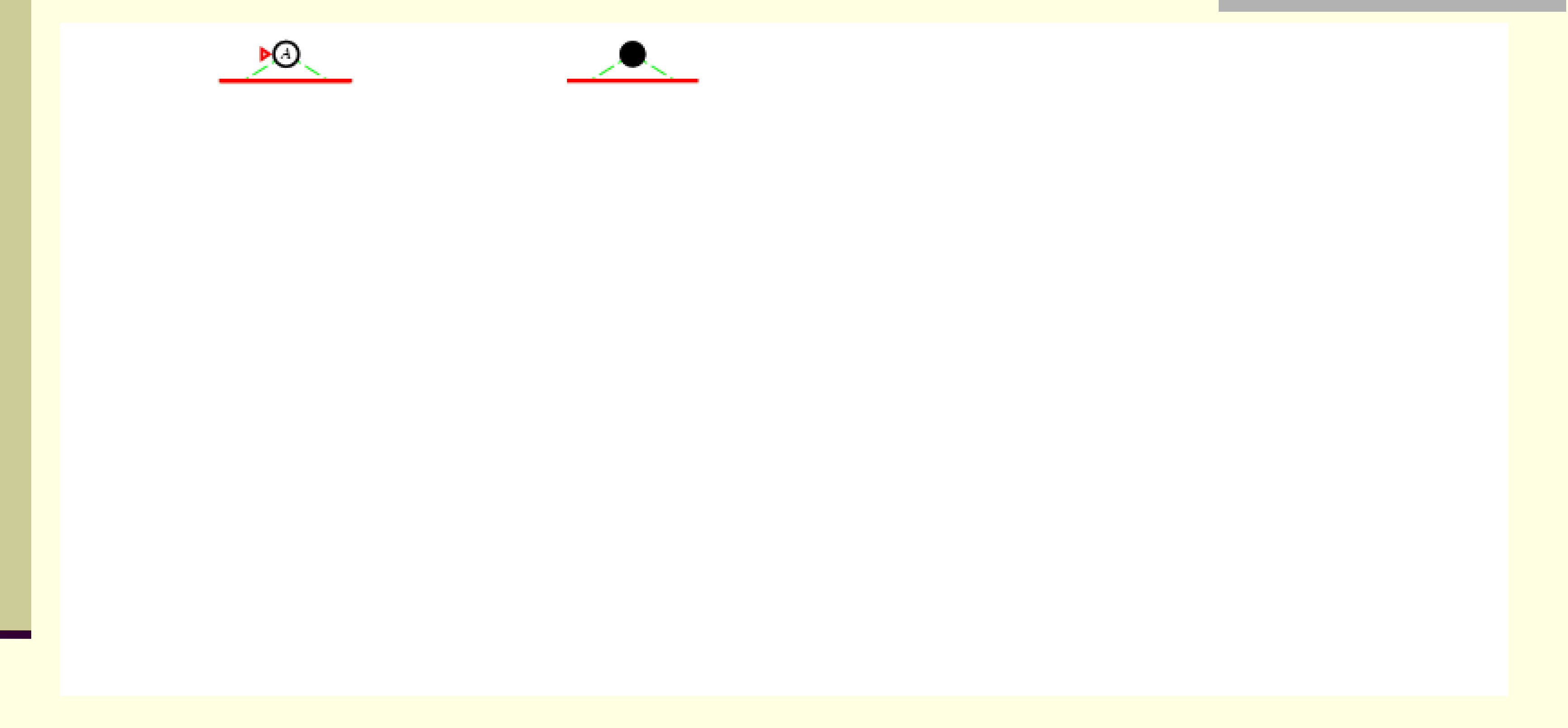
Cautarea cu adancime iterativa

- Este greu de stabilit o limita in adancime pana la care sa se mearga.
- Cautarea cu adancime iterativa alege limita de a merge in adancime iterativ, incepand cu 0, 1, 2 s.a.m.d.

functia cautare_adancime_iterativa(*problema*) ***intoarce*** solutie

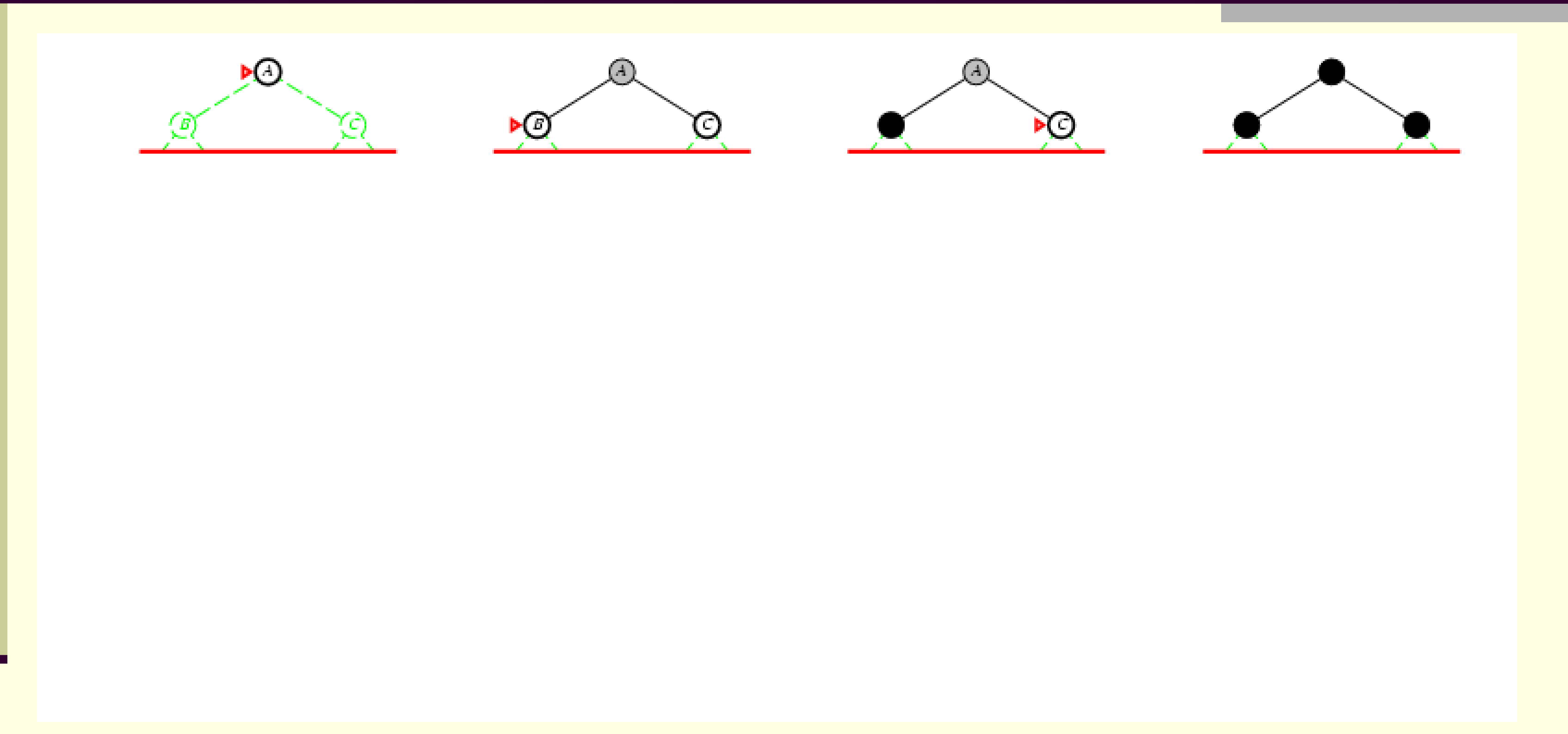
Pentru adancime = 0 *pana la* ∞ *executa*
Daca cautare_adancime_limitata(*problema*, adancime)
    gaseste solutia *atunci*
    intoarce solutia
Sfarsit daca
Sfarsit pentru

Cautarea cu adancime iterativa



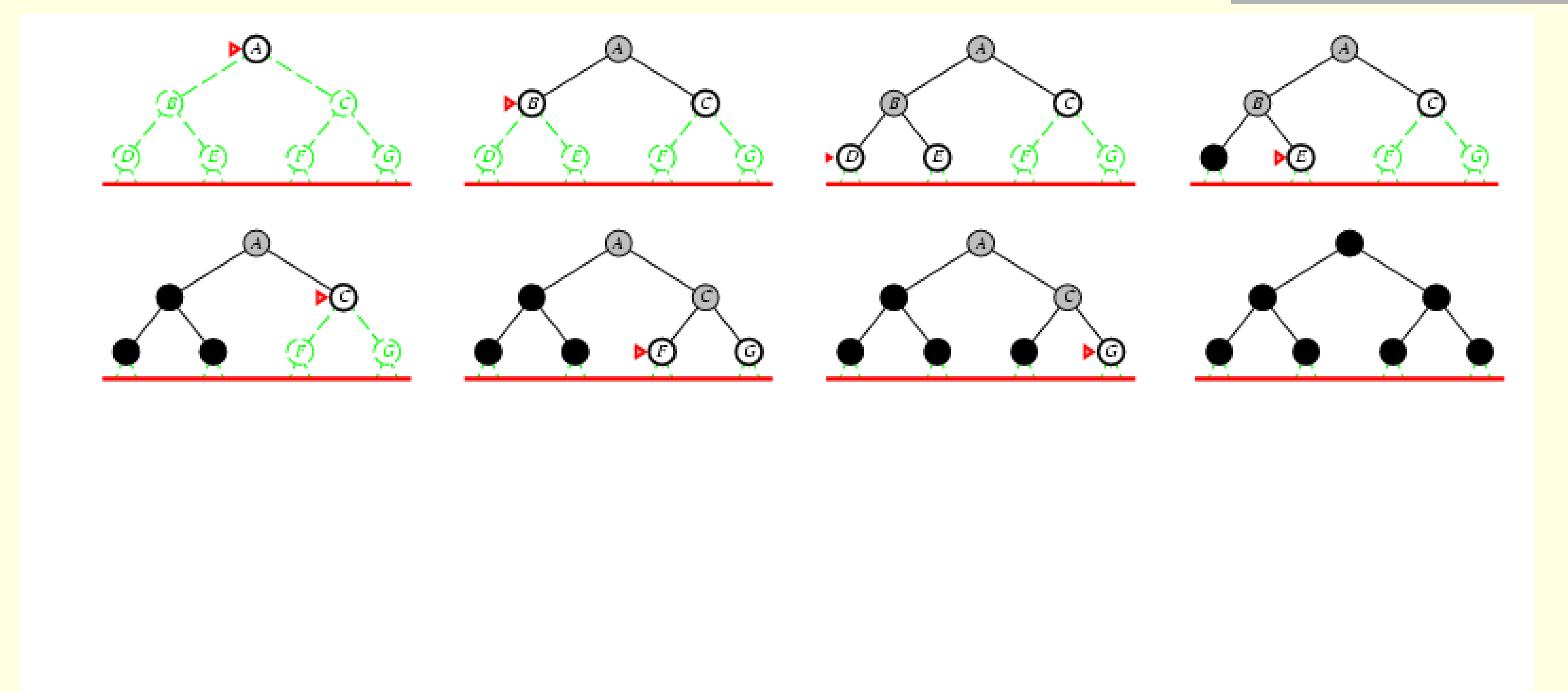
Limita 0

Cautarea cu adancime iterativa



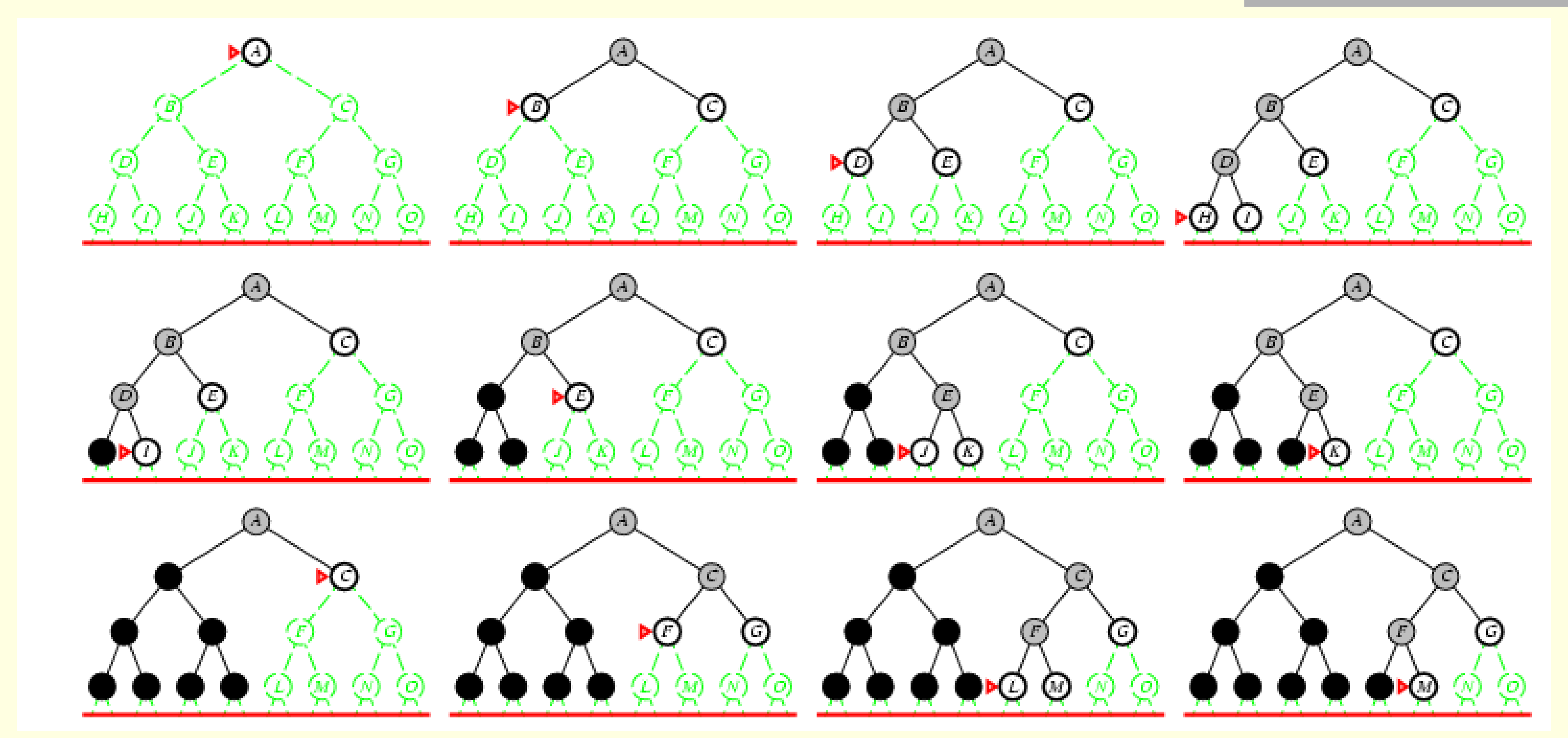
Limita 1

Cautarea cu adancime iterativa



Limita 2

Cautarea cu adancime iterativa



Limita 3

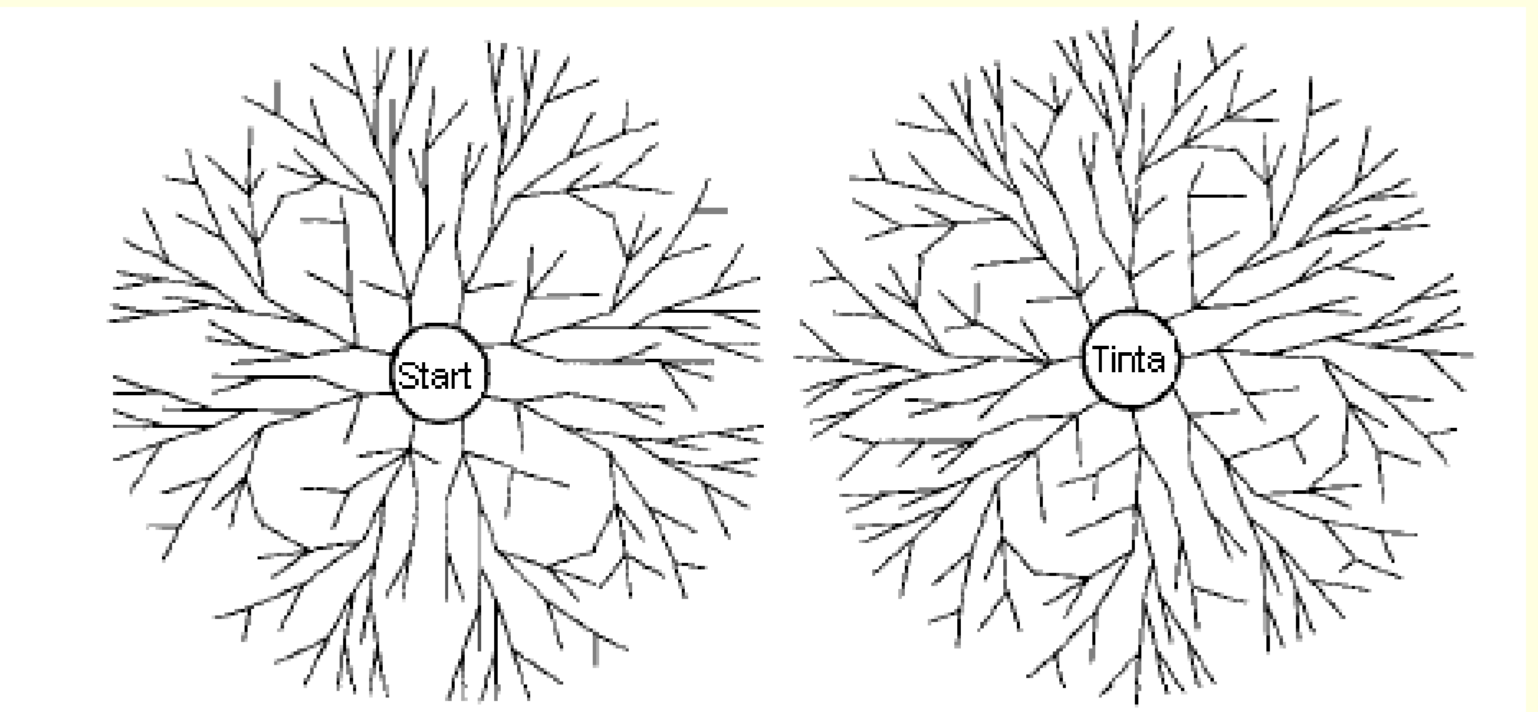
Cautarea cu adancime iterativa

- Este optim si complet ca algoritmul de cautare in latime, dar necesita memorie putina ca algoritmul de cautare in adancime.
- Costul – unele stari sunt expandate de mai multe ori.

- Acest tip de cautare este preferat cand spatiul de cautare este foarte mare si nu se cunoaste adancimea solutiei.

Cautarea bidirectionala

- Ideea este de a cauta in acelasi timp pornind de la starea initiala si de la starea tinta cu scopul de a intalni cele doua cautari la mijloc.



Cautarea bidirectionala

- Daca fiecare nod se expandeaza in *b* alte noduri si o solutie se gaseste la adancimea *d*, aceasta va fi gasita in $O(2b^{d/2}) = O(b^{d/2})$ pasi, ceea ce este mult mai rapid decat la cautarea in latime/adancime.
- Pare foarte buna, dar este complicat de implementat...
- Situatii ce trebuie tratate:
  - Gasirea predecesorilor unui nod;
  - Daca sunt mai multe solutii (stari tinta)? Ex. sah mat…
  - Trebuie verificat daca un nou nod se gaseste in lista celeilalte cautari – daca a fost deja parcurs.
  - Ce fel de cautare se utilizeaza pentru fiecare directie?

Comparatii intre strategii de cautare

Criteriu	In latime	Cost uniform	In adancime	Limitata in adancime	Adancime iterativa	Bidirectionala
Timp	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Spatiu	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optim	Da	Da	Nu	Nu	Da	Da
Complet	Da	Da	Nu	Da, daca $l > d$	Da	Da

- *b* este numarul de noduri in care se expandeaza fiecare nod;
- *d* este adancimea la care se gaseste o solutie;
- *l* este limita de adancime stabilita;
- *m* este adancimea maxima din arbore.

Evitarea starilor de repetare

- Unele probleme pot fi transpuse sub forma de arbori infiniti (ex: gasirea de rute, problema misionarilor si canibalilor etc.).
- Acest lucru trebuie evitat printr-o serie de reguli care trebuie respectate:
  - Pentru un nod, sa nu existe posibilitatea de a se intoarce in nodul parinte – expandarea unui nod sa nu contina nodul parinte!
  - Sa nu se creeze drumuri cu cicluri in ele – prin expandare sa nu apara noduri care au fost gasite la noduri predecesor.
  - Sa nu se genereze o stare care a mai fost intalnita anterior.

Satisfacerea constrangerilor

- Intr-o problema cu satisfacere de constrangeri, starile sunt definite prin valorile pe care le iau o multime de variabile, iar in testarea tintei sunt specificate o serie de constrangeri care trebuie respectate de catre valori.
- In problema damelor,
  - Variabilele - locatiile in care se gasesc cele 8 dame
  - Constrangerile - nu trebuie sa se afle 2 dame pe aceeasi linie, coloana sau diagonala.
- Constrangerile
  - Unare – referitoare la o singura variabila (ex: prima cifra la criptaritmetica trebuie sa fie diferita de 0)
  - Binare – se refera la perechi de variabile (ex: dame)

Tema 1/3

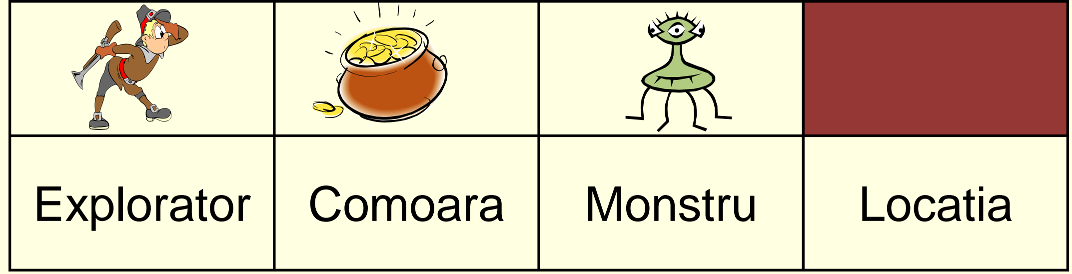
Doua puncte la examenul final!

- Implementati un mediu de masura a performantei pentru o lume in care se gaseste un explorator. Lumea este descrisa astfel:
- **Perceptori**: Agentul explorator primeste un vector de 3 elemente la fiecare mutare. Primul element, un senzor de atingere, este 1 daca exploratorul s-a lovit de ceva si 0 altfel. Al doilea devine 1 daca intra in celula unui monstrul si 0 altfel. Al treilea devine 1 daca intra intr-o celula unde se gaseste o comoara si 1 altfel.
- **Actiuni**: mergi inainte, intoarce la dreapta cu 90°, intoarce la stanga cu 90°, impusca, oprire.
  - O impuscatura tinteste spre celula din fata exploratorului, iar daca aceasta contine un monstru, il ucide.

Tema 2/3

- **Tinta**: Sa gaseasca comoara, sa nu intre in celula unui monstru viu. Masura de performata este de 100 de puncte pentru fiecare comoara gasita, 50 de puncte pentru fiecare monstru ucis, -25 de puncte pentru fiecare foc tras, -1 punct pentru fiecare actiune facuta, -1000 de puncte daca exploratorul intra intr-o celula cu un monstru viu.
- **Mediul**: Consta intr-o tabela de celule. Unele celule contin obstacole, altele un monstru, o comoara sau nimic. Cu fiecare actiune „mergi inainte", exploratorul se muta un patrat cu exceptia cazului cand este un obstacol in calea sa, moment in care ramane pe loc dar senzorul de atingere se face 1. O comanda „oprire" termina simularea.

Tema 3/3



Recapitulare 1/4

- 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 2/4

- 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 3/4

- **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: $O(b^d)$
- **Cautarea cu cost uniform** expandeaza mai intai nodul cu costul minim. Este complet, optim (si cand actiunile au costuri diferite), complexitatile sunt aceleasi.
- **Cautarea in adancime** expandeaza mai intai un drum de la radacina pana la frunze. Nu este nici complet, nici optim si are complexitatea temporala $O(b^m)$ si pe cea spatiala $O(bm)$, unde m este adancimea maxima. Daca arborele este de adancime foarte mare sau infinita, aceasta cautare este nepractica.

Recapitulare 4/4

- **Cautarea limitata in adancime** stabileste o limita la cat de adanc poate merge cautarea in adancime. Daca limita este egala chiar cu d, atunci complexitatea temporala si spatiala sunt minimizate.
- **Cautarea iterativa in adancime** foloseste cautarea limitata in adancime cu limite care cresc pana cand se ajunge la tinta. Este complet, optimal, cu complexitatea temporala $O(b^d)$ si cea spatiala $O(bd)$.
- **Cautarea bidirectionala** reduce complexitatea temporala foarte mult insa nu este aplicabila in orice caz.