

# Agenti care rezolva probleme

Catalin Stoean

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

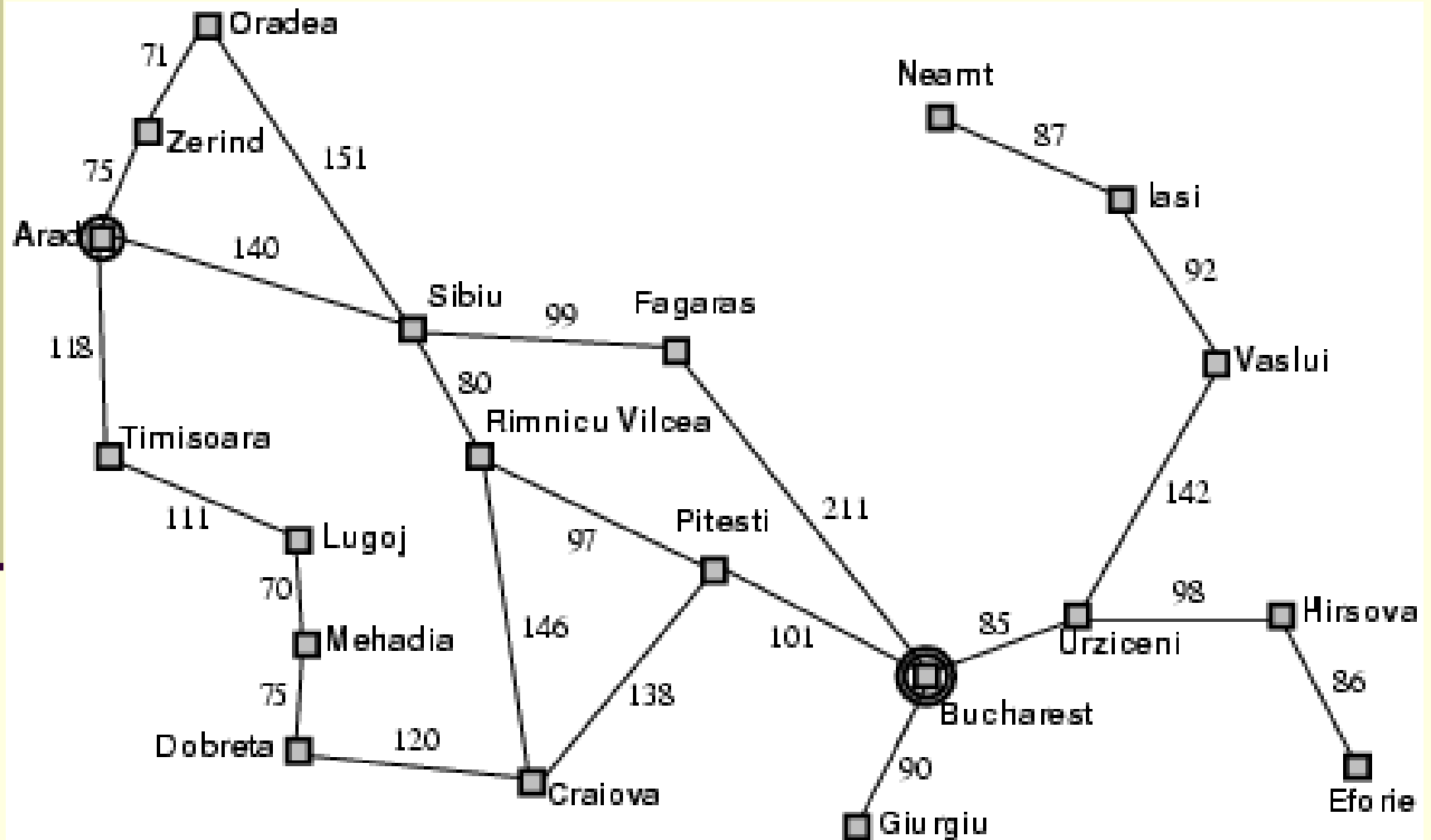
<http://inf.ucv.ro/~cstoean>

# 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 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*

# Algoritm cautare in latime

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

noduri = genereaza\_lista(genereaza\_nod(stare\_iniciala[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*

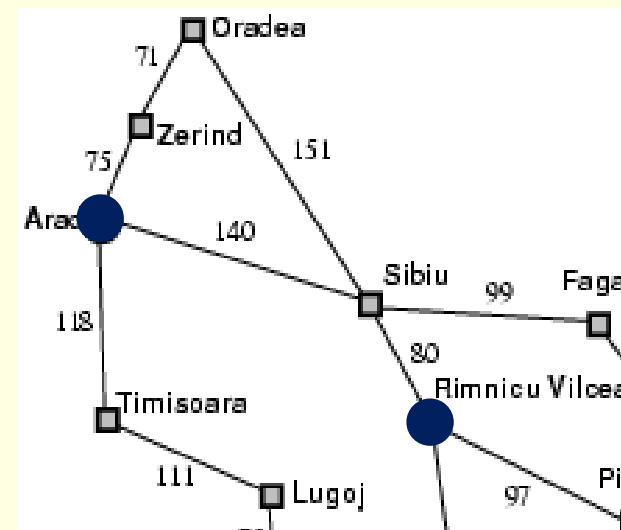
noduri = adauga(noduri, expandare(nod, adauga\_la\_sfarsit))

*Sfarsit cat timp*

# 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

**funcția** `cautare_cost_uniform(problema)` **intoarce** *soluție* sau **esec**  
`noduri = genereaza_lista(genereaza_nod(stare_initala[problema]))`

*Cat timp soluție negăsită și noduri  $\neq \emptyset$  execută*

`nod = scoate_din_fata(noduri)`

*Dacă* `testare_tinta[problema]` *se aplică la* `stare(nod)` *atunci*  
*soluție găsită*

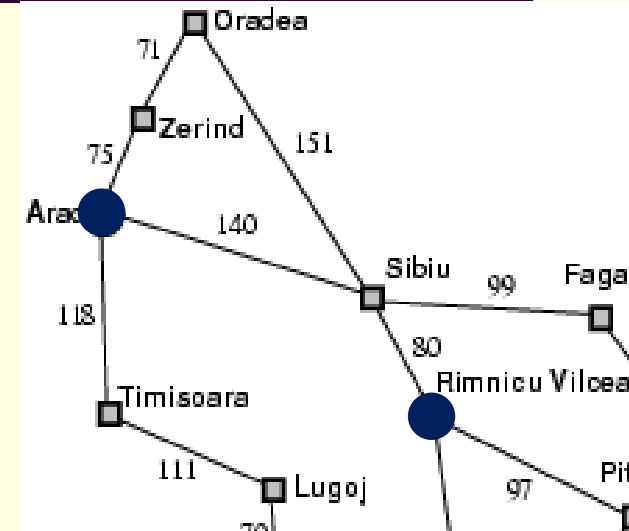
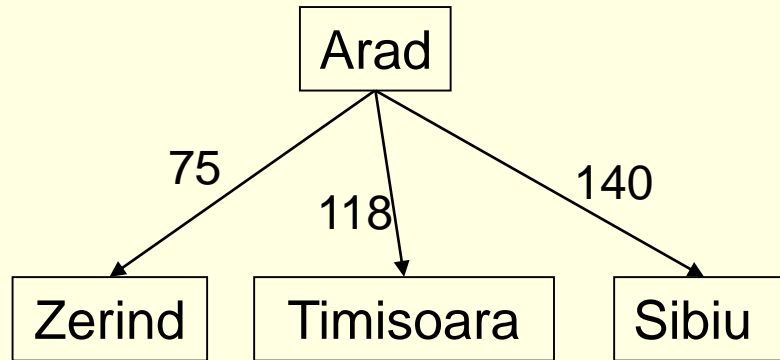
*Altfel*

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

*Sfârșit cat timp*

Se permit noduri duplicate, însă și ele sunt ordonate în funcție de cost.

# Cautarea cu cost uniform

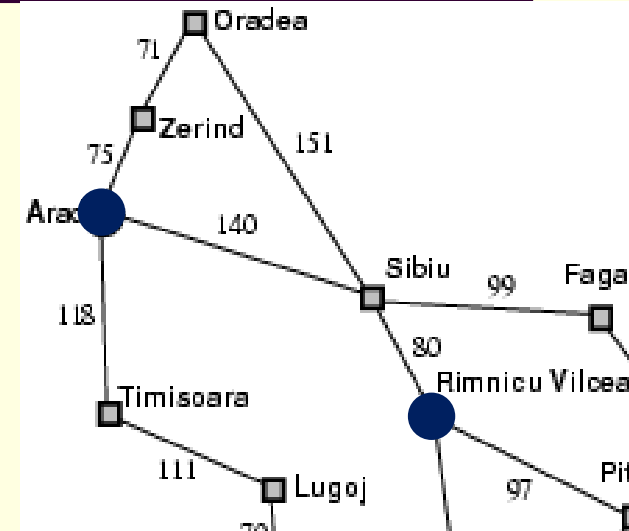
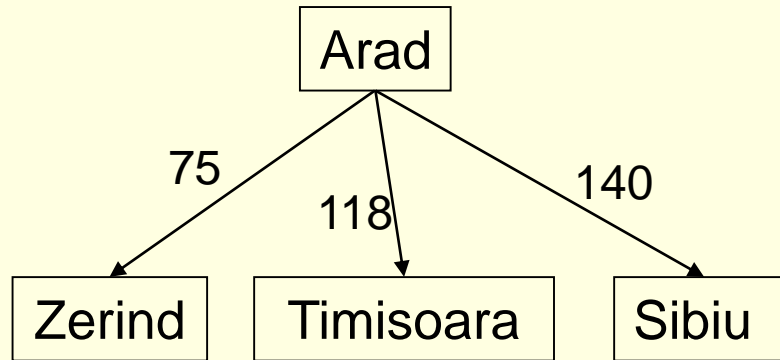


Fața	noduri	Sfarsit
Arad		
0		

Parcurgerea: Arad,



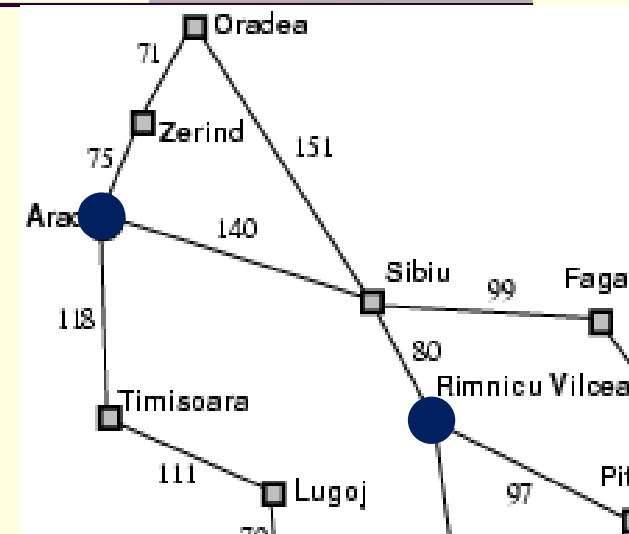
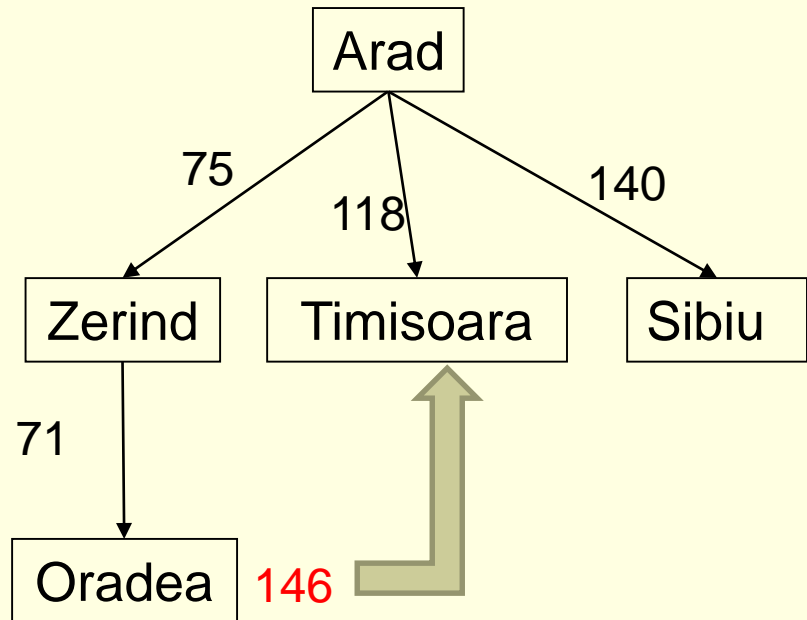
# Cautarea cu cost uniform



Fața	noduri		Sfarsit
Zerind	TM	SB	
75	118	140	

Parcurgerea: Arad,

# Cautarea cu cost uniform

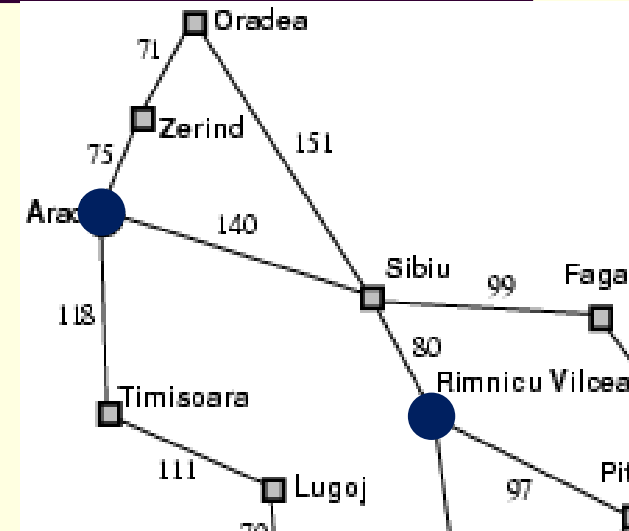
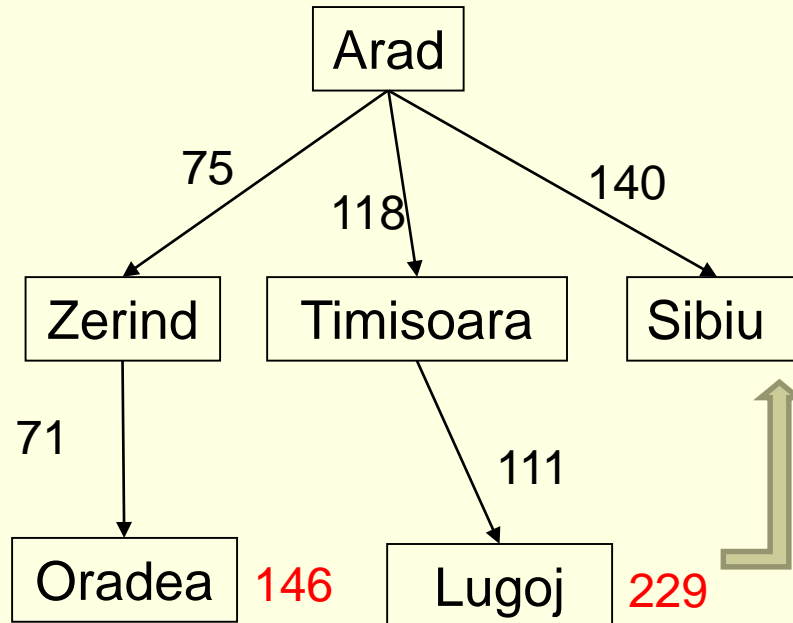


In total, de la Arad la Oradea.

Fața	noduri	Sfarsit
TM	SB	Oradea
118	140	146

Parcurgerea: Arad, Zerind

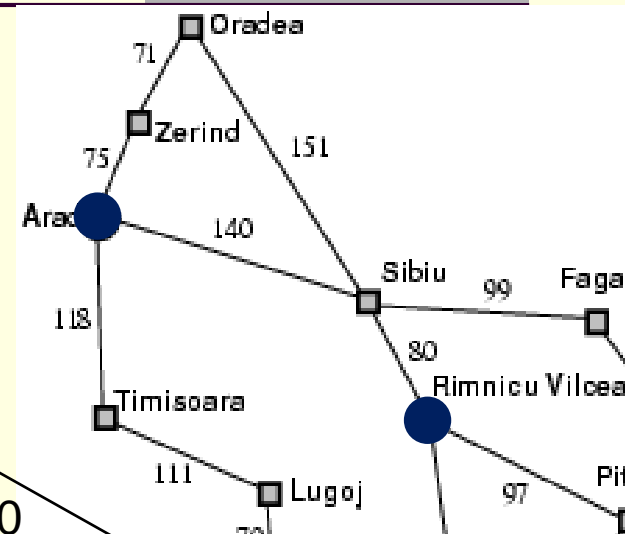
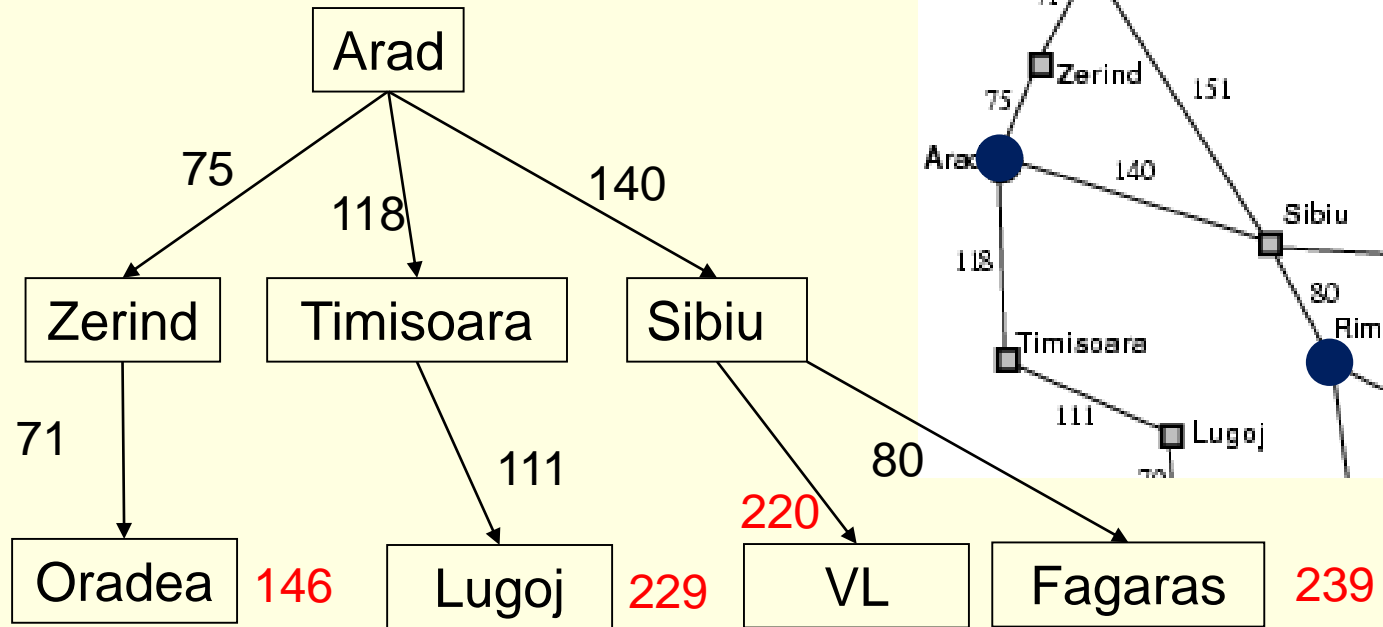
# Cautarea cu cost uniform



Fața	noduri		Sfarsit
SB	Oradea	Lugoj	
140	146	229	

Parcurgerea: Arad, Zerind, TM

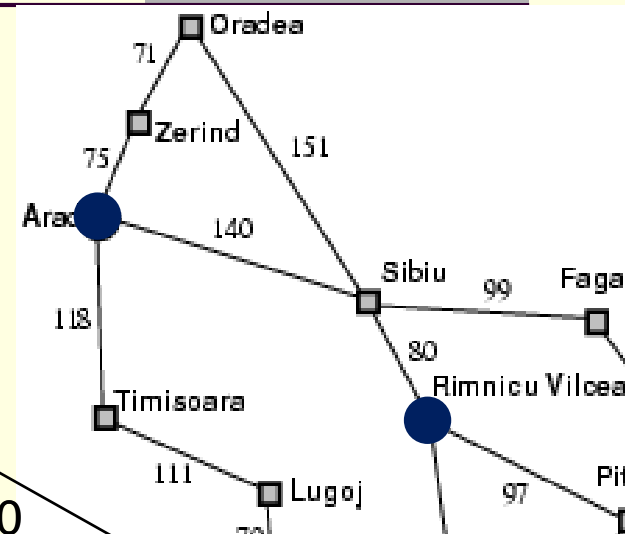
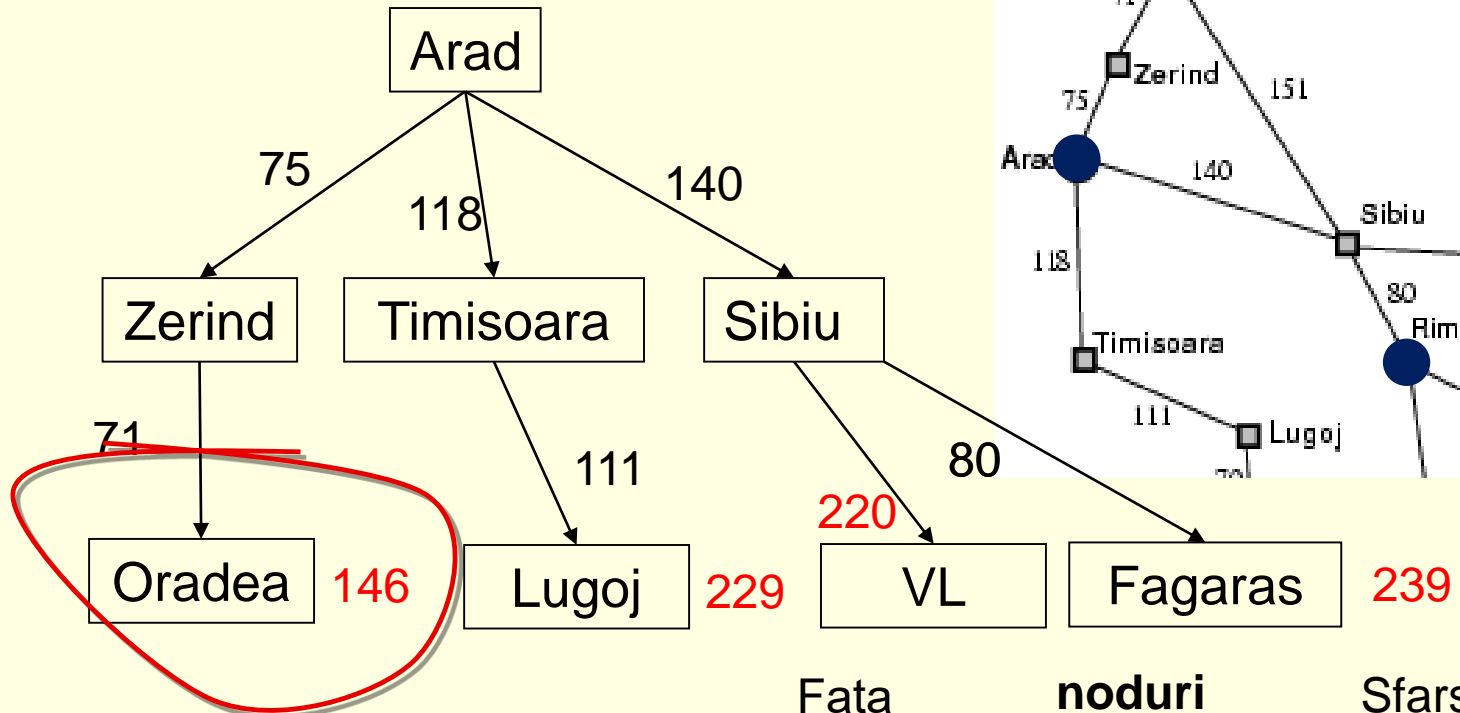
# Cautarea cu cost uniform



Fața	noduri			Sfarsit
Oradea	VL	Lugoj	Fagaras	
146	220	229	239	

Parcurgerea: Arad, Zerind, TM, SB

# Cautarea cu cost uniform

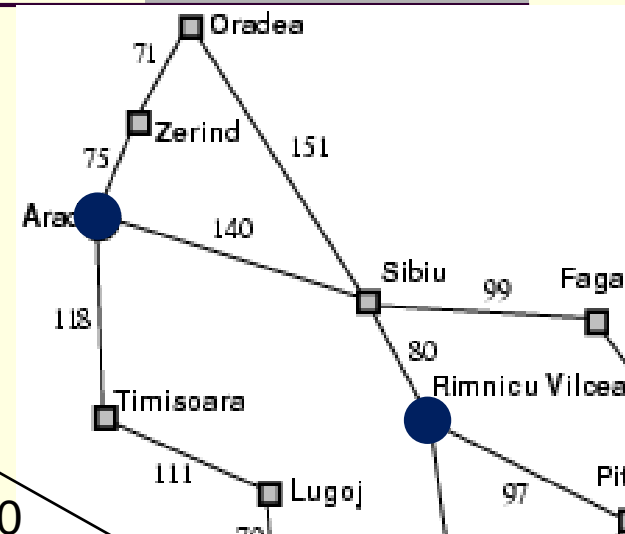
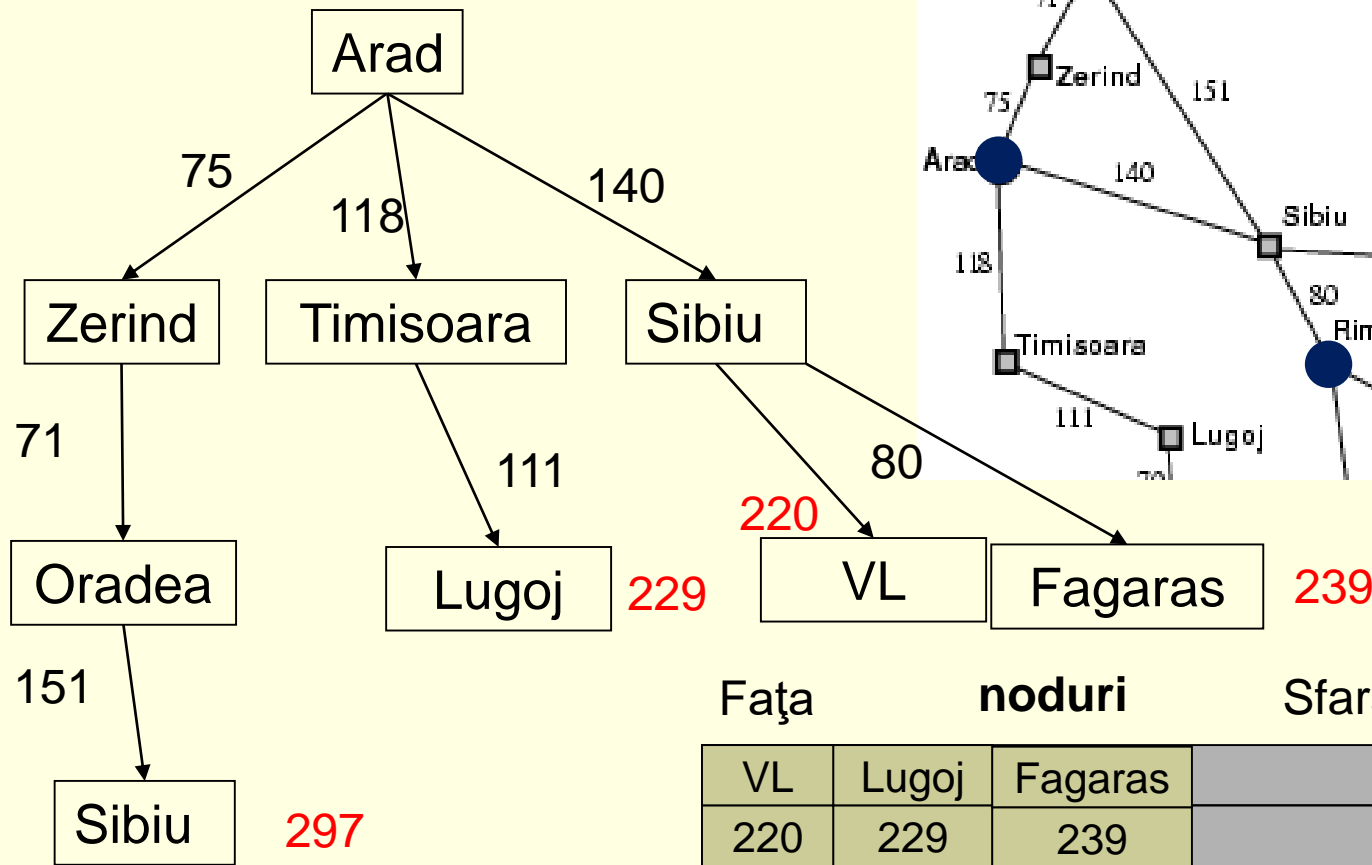


Fața	noduri			Sfarsit
Oradea	VL	Lugoj	Fagaras	
146	220	229	239	

Parcurgerea: Arad, Zerind, TM, SB

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

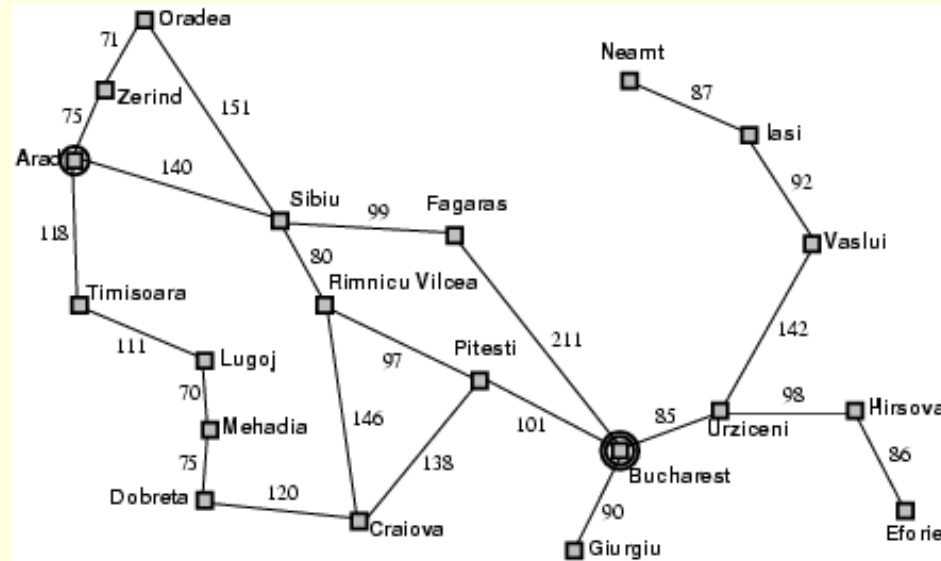
# Cautarea cu cost uniform



Nu adaugam SB pentru ca are o evaluare mai mare decat vechea valoare a orasului.

Parcurgerea: Arad, Zerind, TM, SB, Oradea, VL

# Exercitiu



Fața                      **noduri**                      Sfarsit

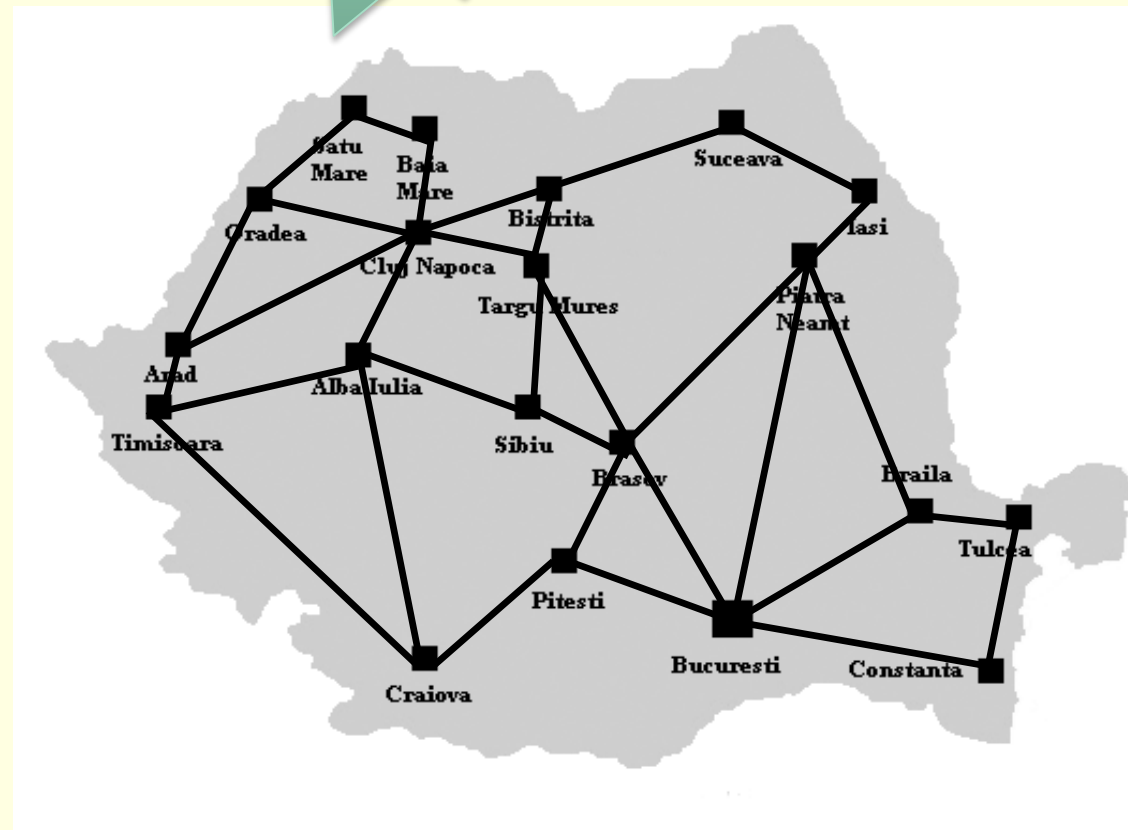


Gasiti o ruta de la Arad la Bucuresti folosind parcurgerea cu cost uniform. Desenati arborele, scrieti parcurgerea si continutul pentru noduri la fiecare pas.

# Tema...

Un punct la  
examenul  
final!

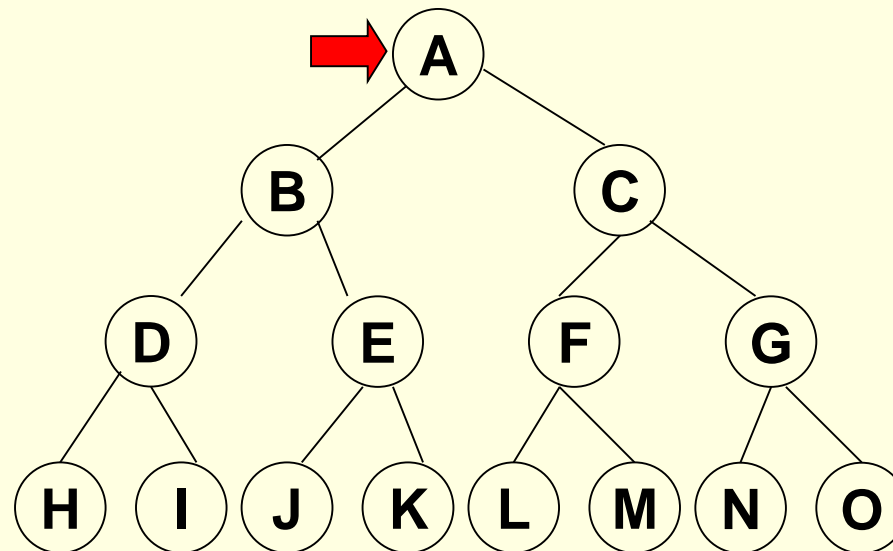
- Găsiți distanțele rutiere dintre orașele de pe harta din figura. Utilizați-le apoi pentru a implementa un algoritm de căutare cu cost uniform pentru a ajunge de la Oradea la Tulcea.





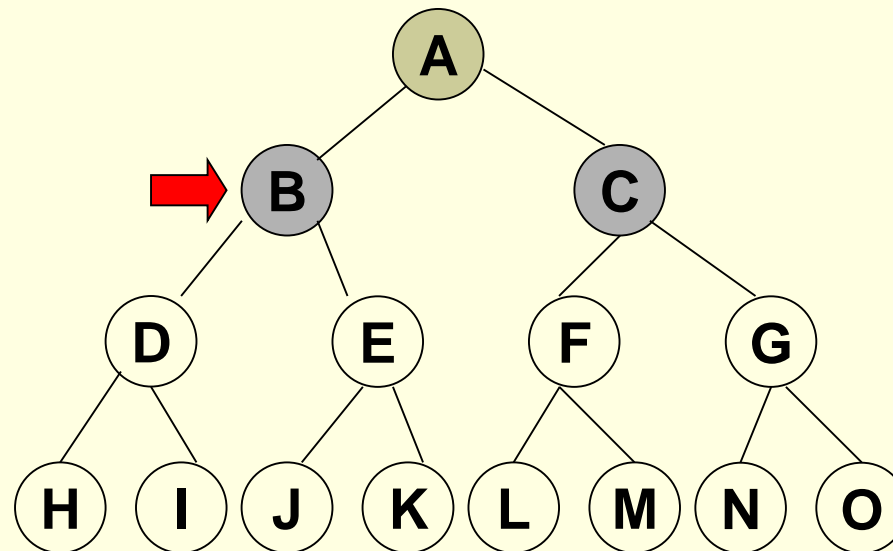
# Cautarea in adancime

- 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 nivelele mai putin adanci.



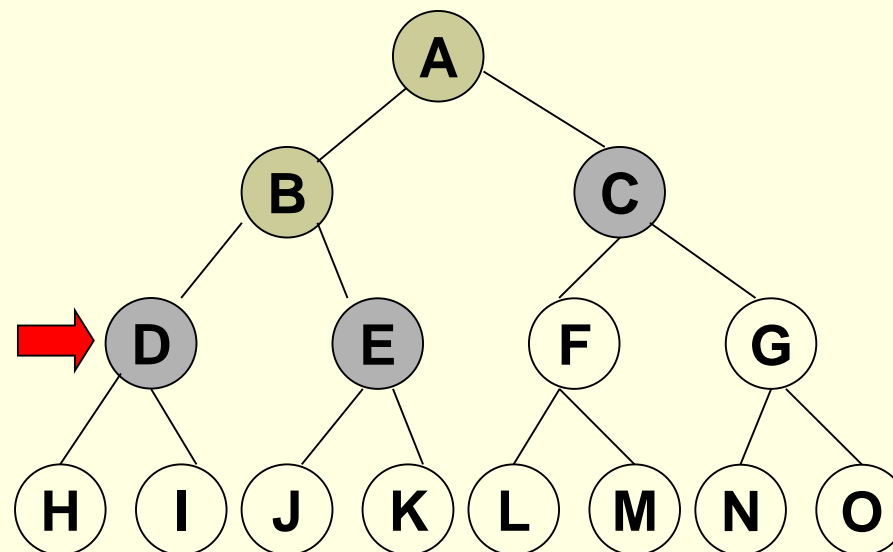
# Cautarea in adancime

- 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 nivelele mai putin adanci.



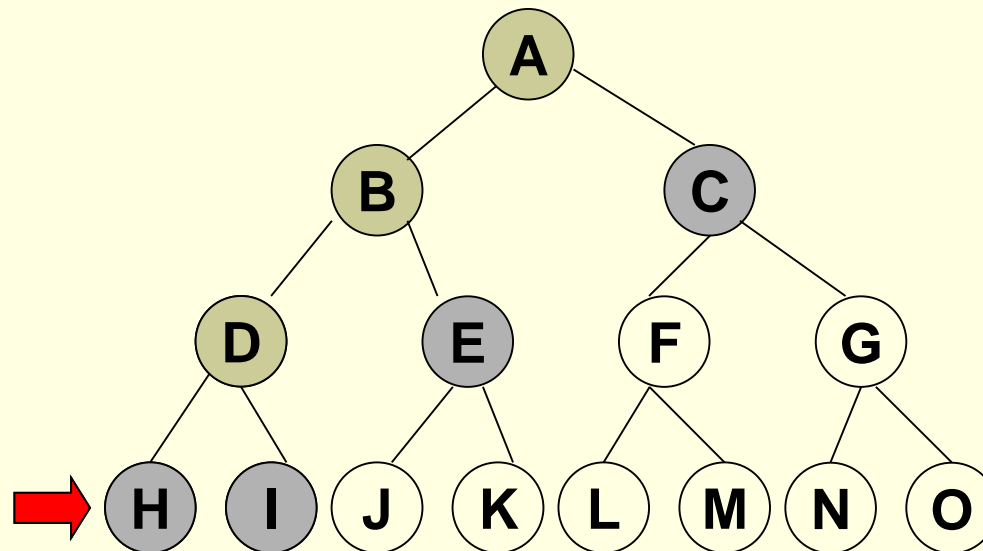
# Cautarea in adancime

- 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 nivelele mai putin adanci.



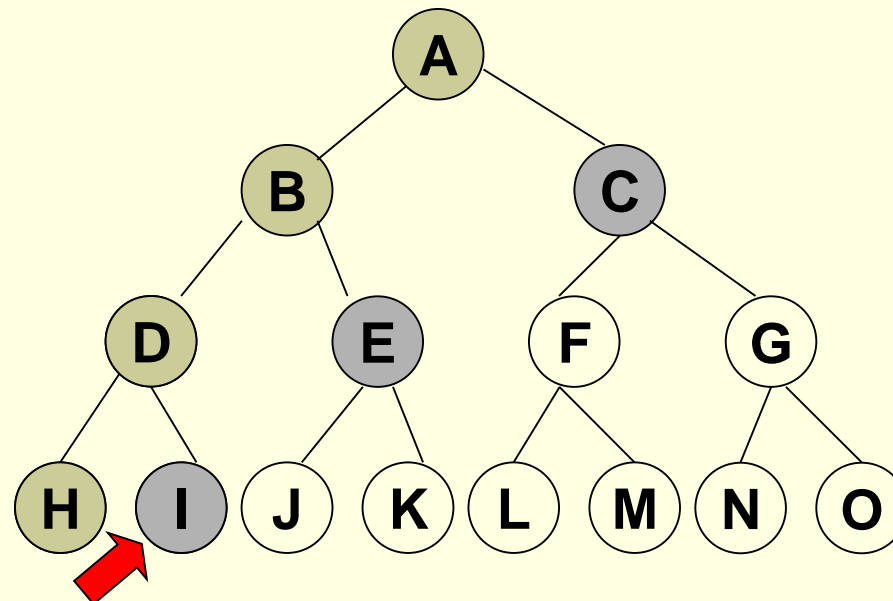
# Cautarea in adancime

- 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 nivelele mai putin adanci.



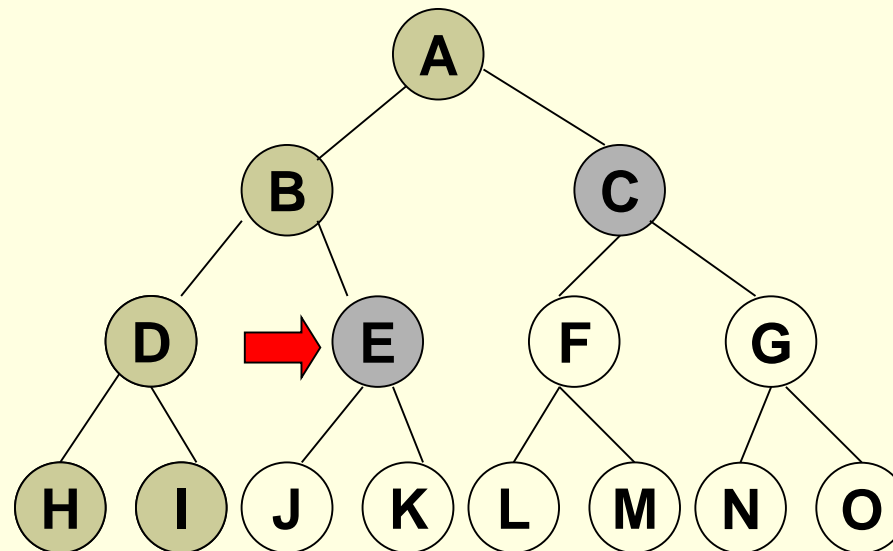
# Cautarea in adancime

- 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 nivelele mai putin adanci.



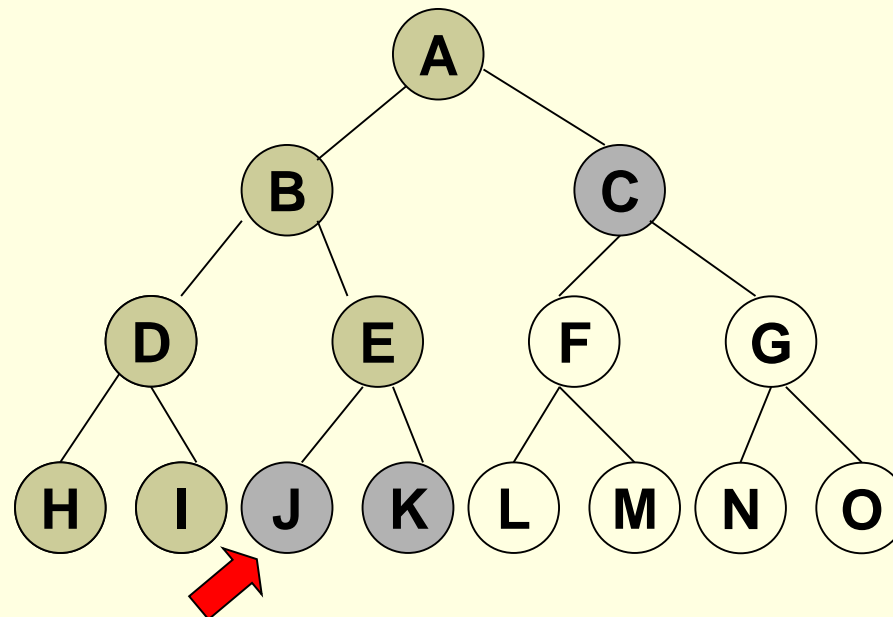
# Cautarea in adancime

- 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 nivelele mai putin adanci.



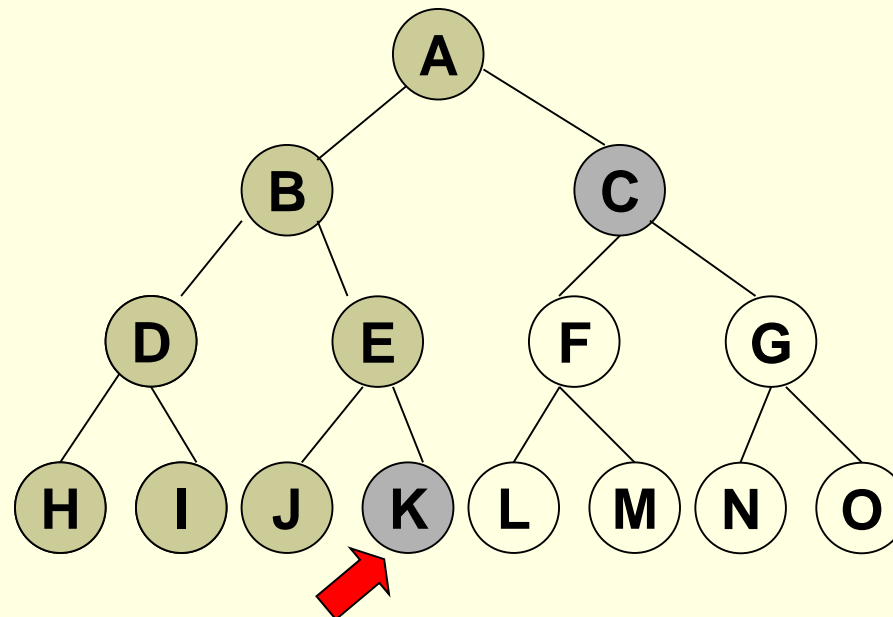
# Cautarea in adancime

- 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

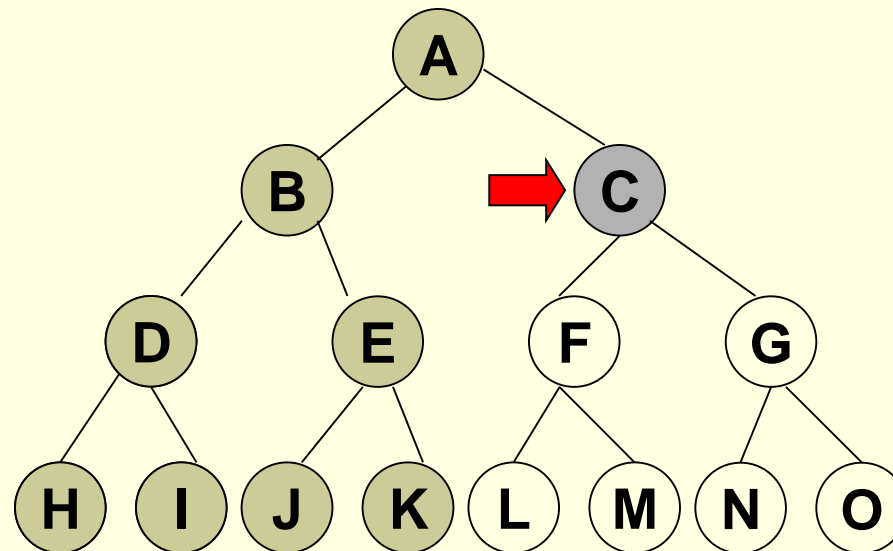
- Se expandeaza nodul radacina, apoi se merge pe un drum pana se ajunge la cel mai adanc nivel al arborelui.
- Numai cand se ajunge la final (la nodurile frunza), cautarea se intoarce si expandeaza noduri de la nivelele mai putin adanci.





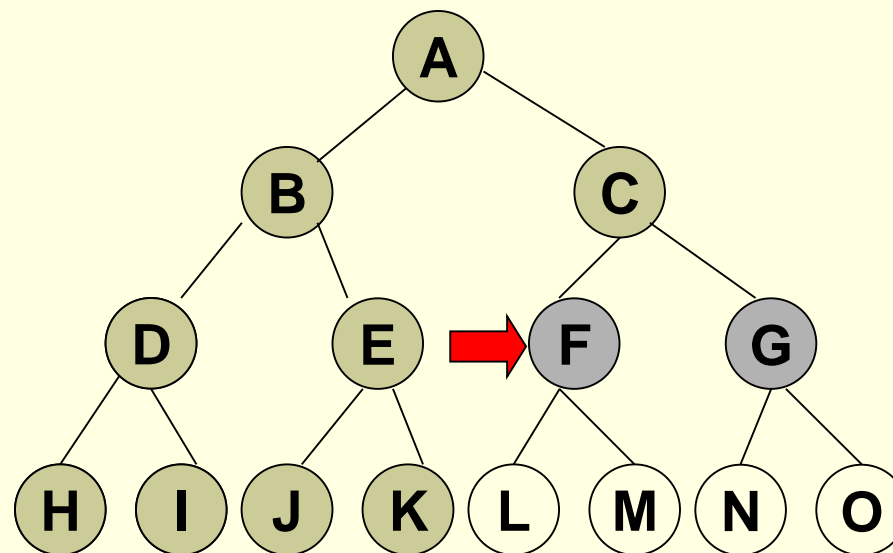
# Cautarea in adancime

- 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 nivelele mai putin adanci.



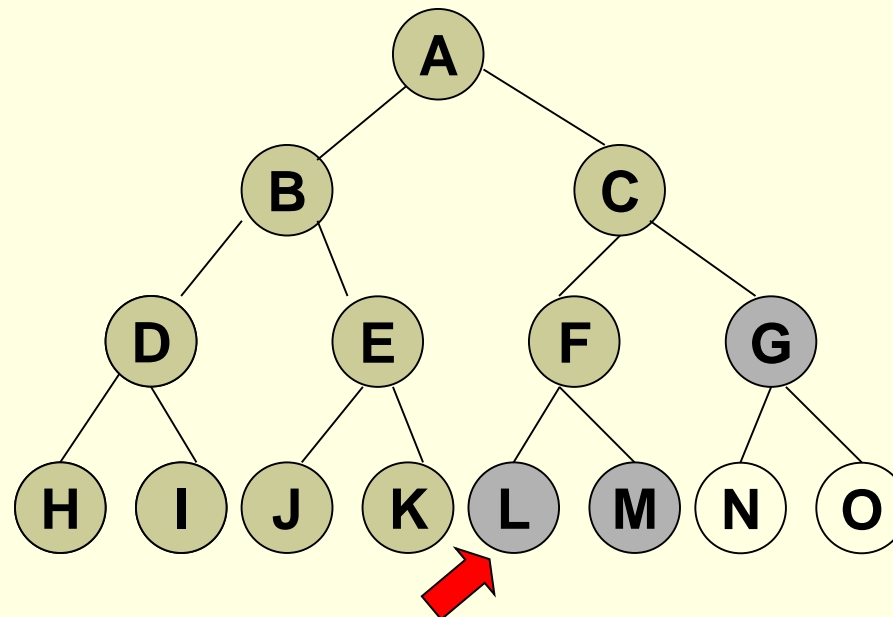
# Cautarea in adancime

- 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 nivelele mai putin adanci.



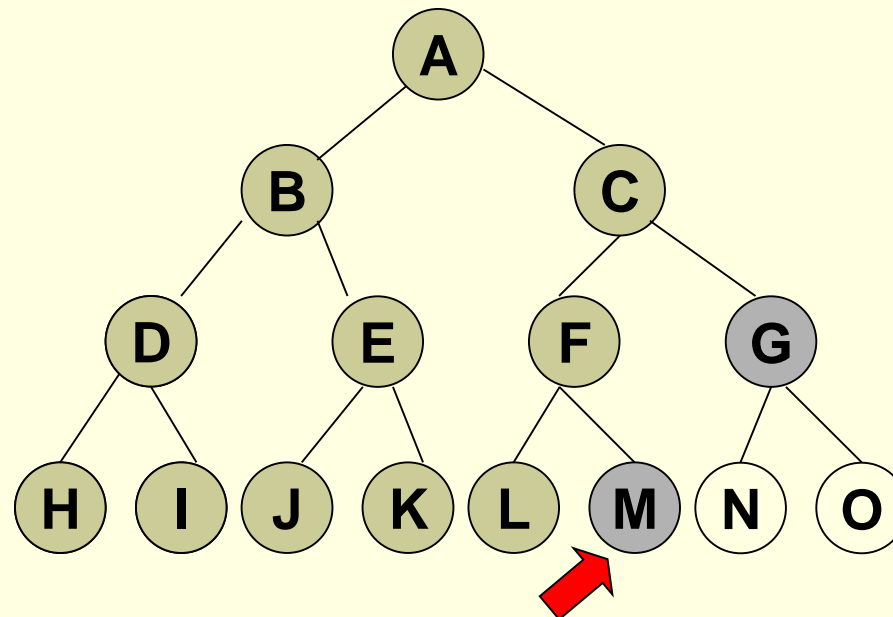
# Cautarea in adancime

- 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.



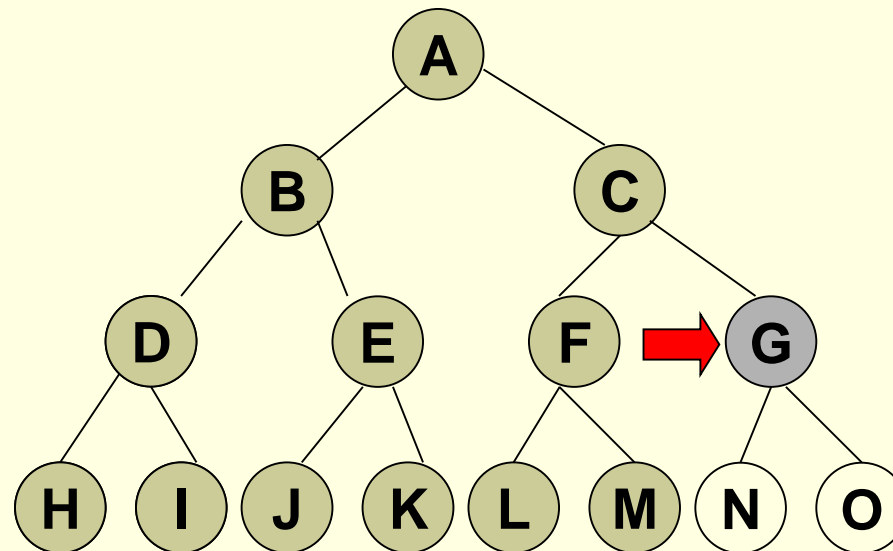
# Cautarea in adancime

- 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 nivelele mai putin adanci.



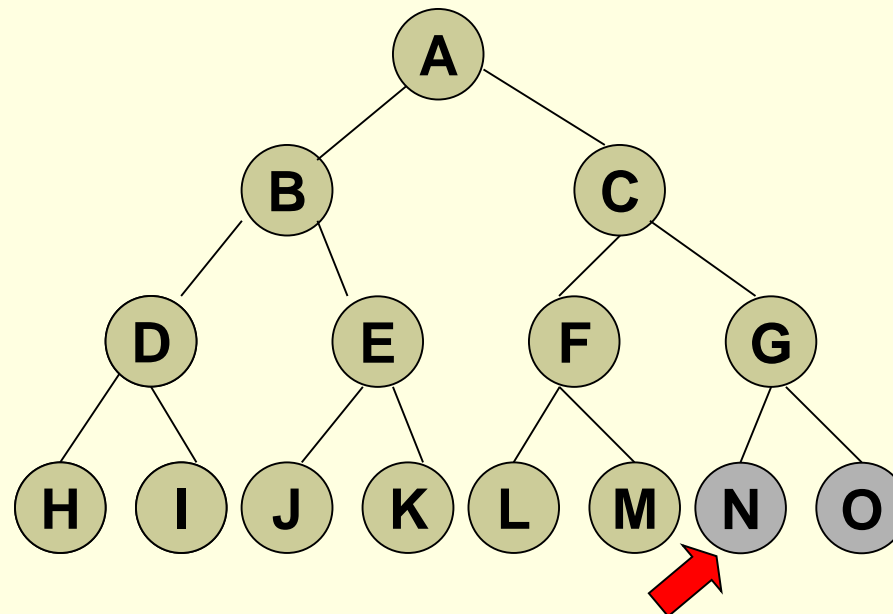
# Cautarea in adancime

- 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 nivelele mai putin adanci.



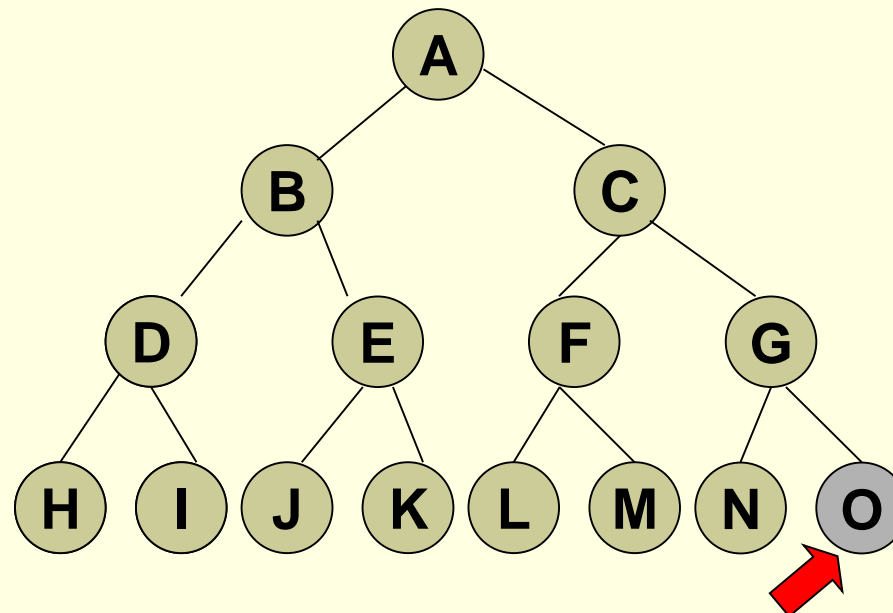
# Cautarea in adancime

- 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 nivelele 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 nivelele 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_iniciala[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*

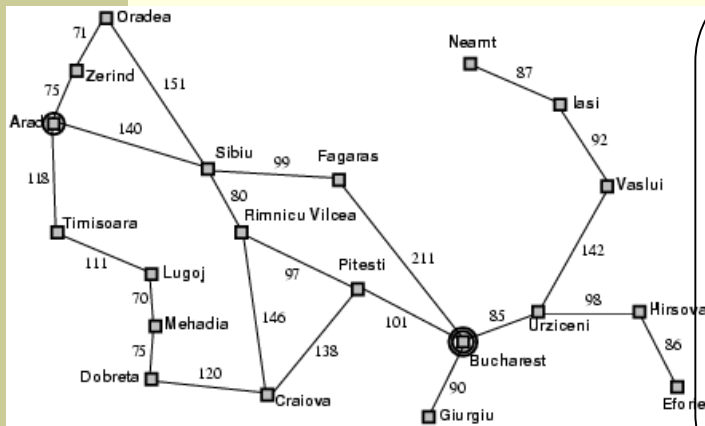
*Altfel*

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

*Sfarsit cat timp*



# Algoritm cautare in adancime



Arad

**functia** `cautare_adancime(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_inceput))`

*Sfarsit cat timp*

Fața

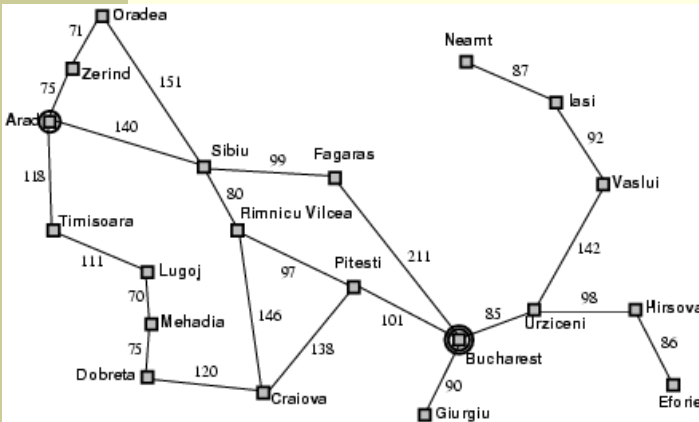
**noduri**

Sfarsit

Arad

Parcurgerea: Arad,

# Algoritm cautare in adancime



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

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

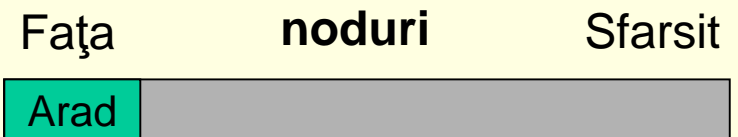
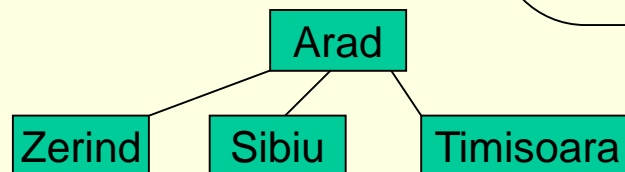
`nod = scoate_din_fata(noduri)`

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

*Altfel*

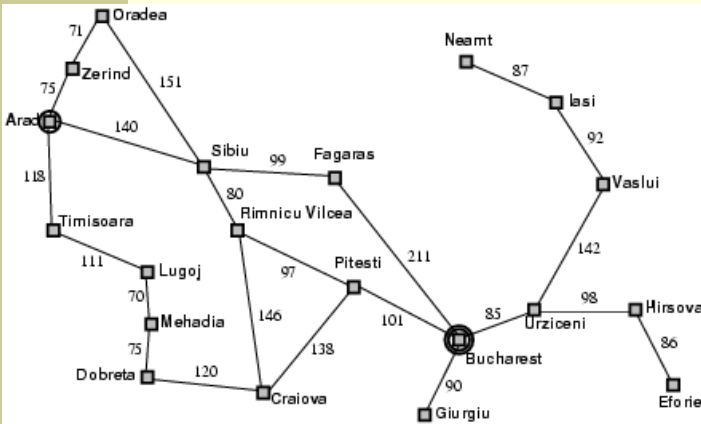
`noduri = adauga(noduri, expandare(nod, adauga_la_inceput))`

*Sfarsit cat timp*



Parcurgerea: Arad,

# Algoritm cautare in adancime



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

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

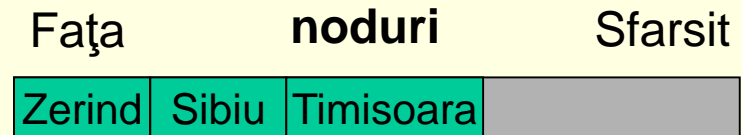
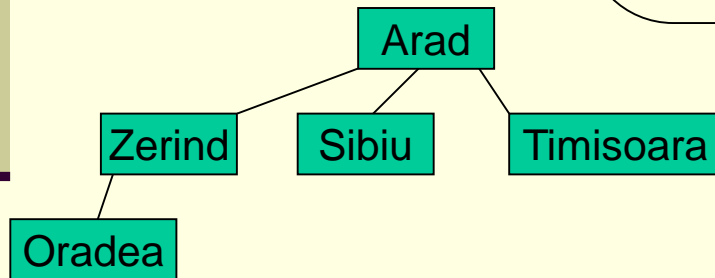
nod = scoate\_din\_fata(noduri)

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

*Altfel*

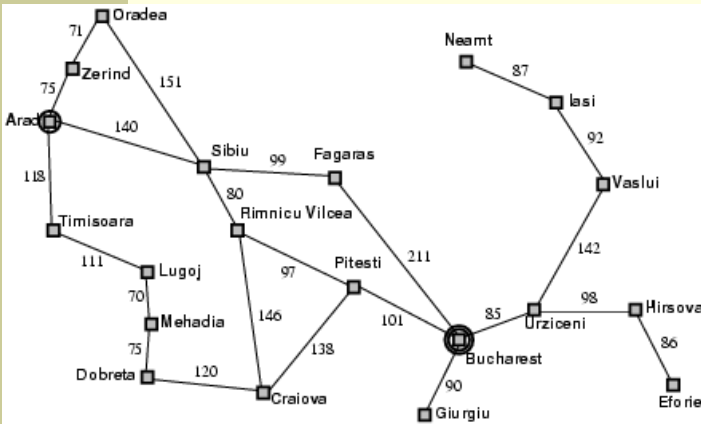
noduri = adauga(noduri, expandare(nod, adauga\_la\_inceput))

*Sfarsit cat timp*



Parcurgerea: Arad, Zerind,

# Algoritm cautare in adancime



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

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

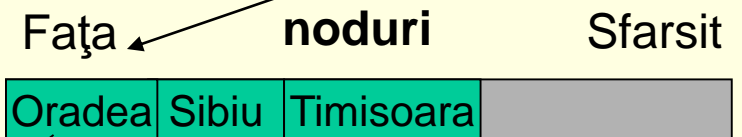
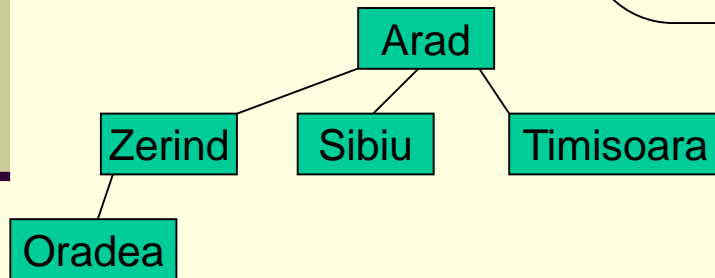
`nod = scoate_din_fata(noduri)`

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

*Altfel*

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

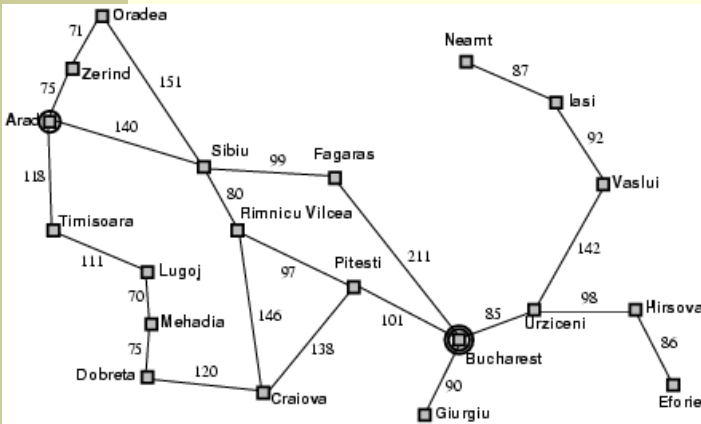
*Sfarsit* `cat timp`



Parcurgerea: Arad, Zerind,

**Adaugarea se face prin față!**

# Algoritm cautare in adancime



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

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

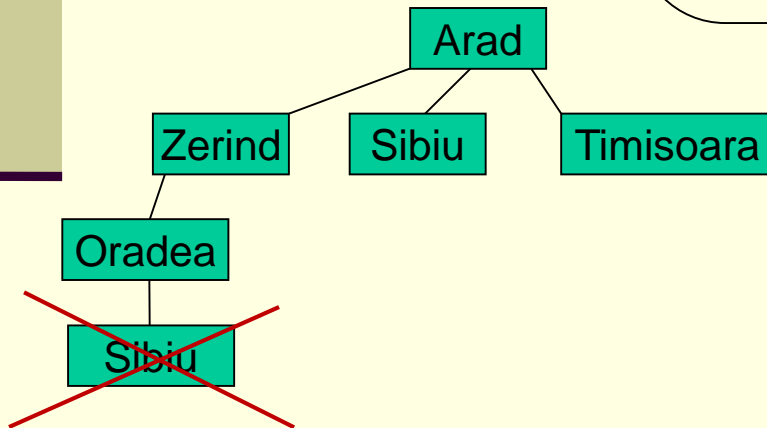
nod = scoate\_din\_fata(noduri)

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

*Altfel*

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

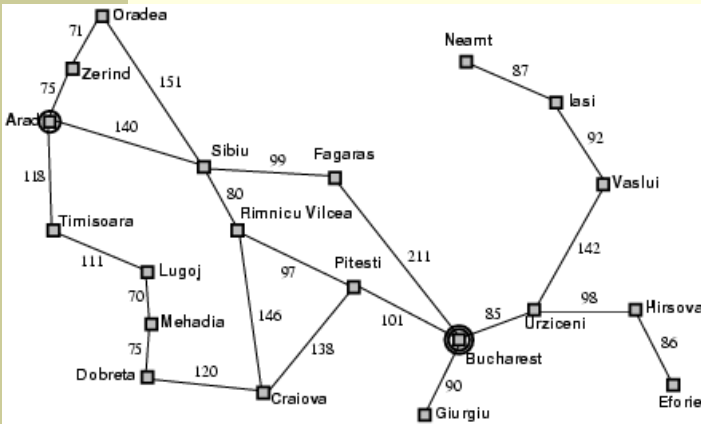
*Sfarsit* cat timp



Fața	noduri	Sfarsit
Oradea	Sibiu	Timisoara

Parcurgerea: Arad, Zerind, Oradea

# Algoritm cautare in adancime



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

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

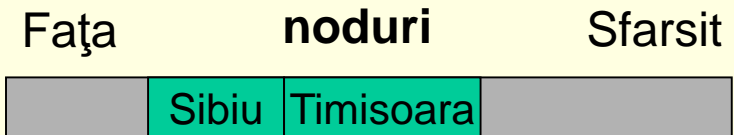
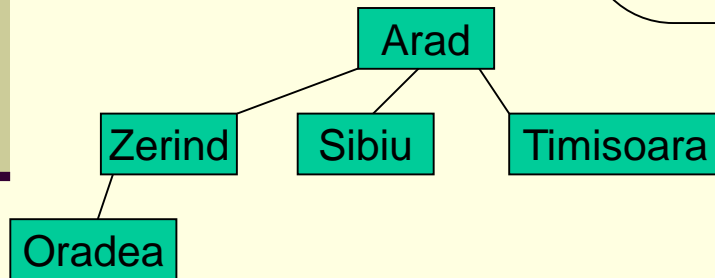
nod = scoate\_din\_fata(noduri)

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

*Altfel*

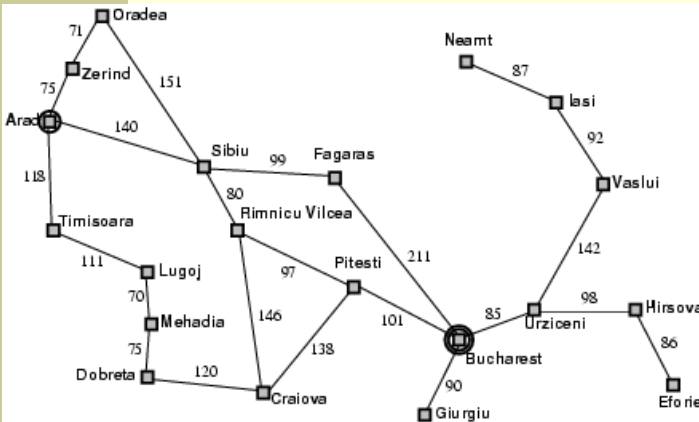
noduri = adauga(noduri, expandare(nod, adauga\_la\_inceput))

*Sfarsit* cat timp



Parcurgerea: Arad, Zerind,  
 Oradea, Sibiu

# Algoritm cautare in adancime



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

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

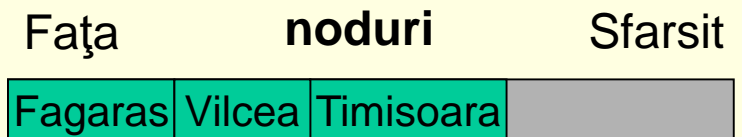
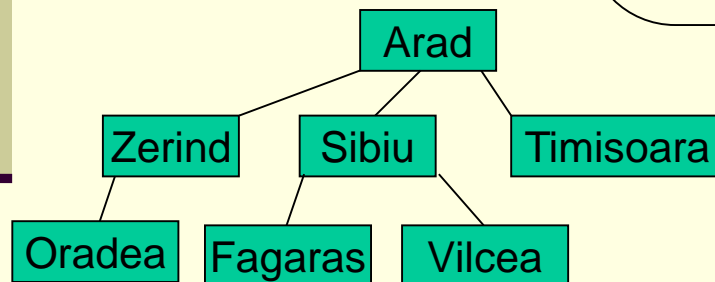
`nod = scoate_din_fata(noduri)`

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

*Altfel*

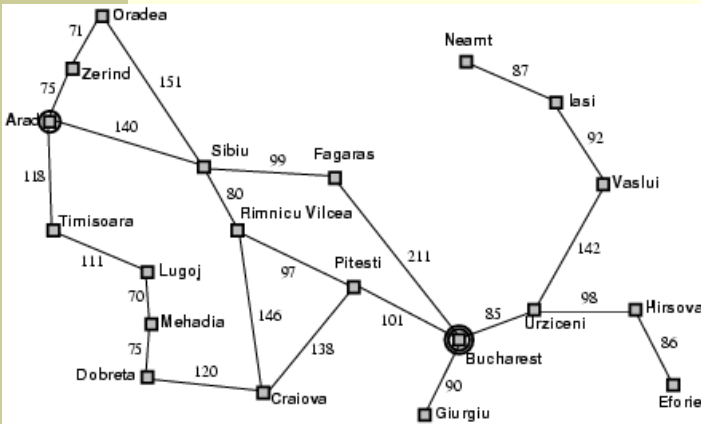
`noduri = adauga(noduri, expandare(nod, adauga_la_inceput))`

*Sfarsit cat timp*



Parcurgerea: Arad, Zerind,  
Oradea, Sibiu

# Algoritm cautare in adancime



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

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

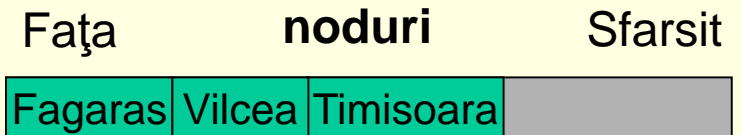
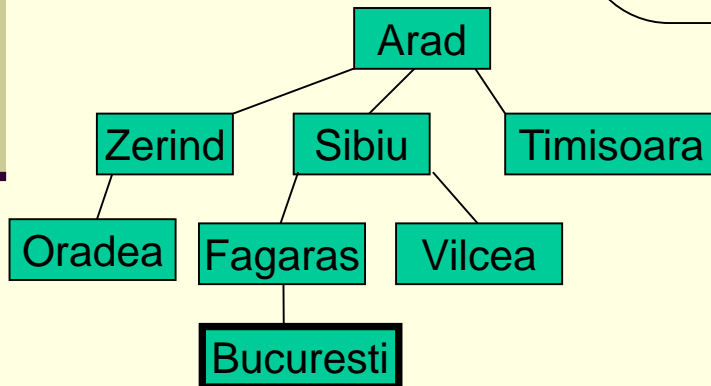
nod = scoate\_din\_fata(noduri)

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

*Altfel*

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

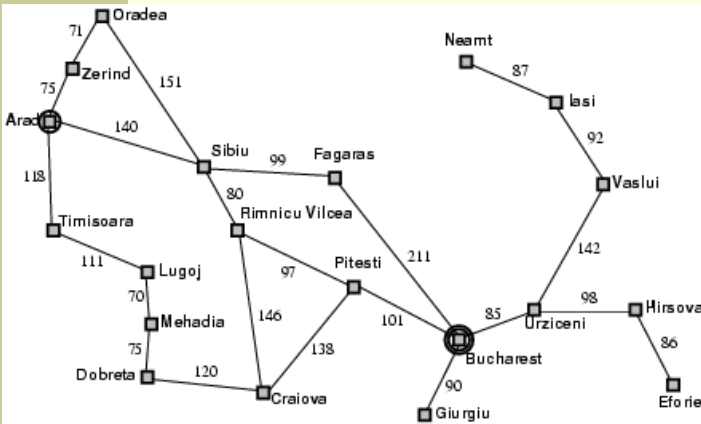
*Sfarsit* cat timp



Parcurgerea: Arad, Zerind, Oradea, Sibiu, Fagaras



# Algoritm cautare in adancime



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

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

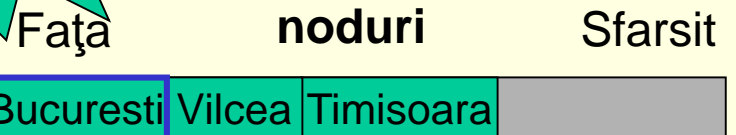
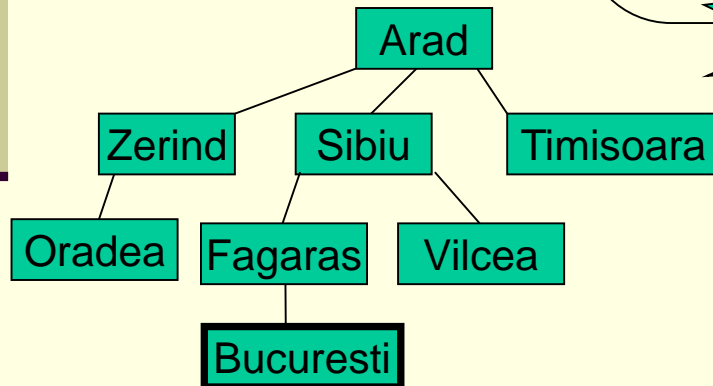
`nod = scoate_din_fata(noduri)`

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

*Altfel*

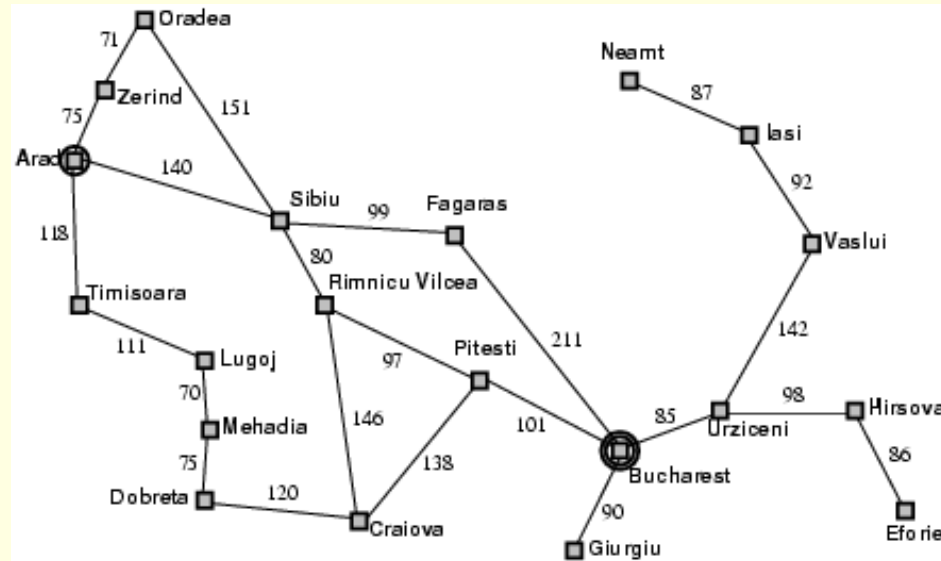
`noduri = adauga(noduri, expandare(nod, adauga_la_inceput))`

*Sfarsit* `cat timp`



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

# Exercitiu



Fața                      **noduri**                      Sfarsit



Gasiti o ruta de la Timisoara la Pitesti folosind parcurgerea in adancime.

Desenati arborele, scrieti parcurgerea si continutul pentru noduri la fiecare pas. Alegeti descendenti in ordine alfabetica

**Parcurgerea:** Timisoara, ..., Pitesti

# Exercitiu

- Considerăm problema găsirii unei rute în figura de mai jos de la start la stop.
- Agentul se mută un pătrat la fiecare pas vertical sau orizontal.
- Nu se poate deplasa în pătratele hasurate.
- Etichetați cu litere în ordine alfabetică pătratele dacă se utilizează o căutare în adâncime, iar ordinea operațiilor este: sus, stânga, dreapta și jos.

			stop		
		start			

# 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 adancime arborele complet care duce la rezolvarea problemei. Numerotati nodurile in ordinea in care au fost vizitate.

# 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`  $\neq \emptyset$  *executa*

`nod = scoate_din_fata(noduri)`

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

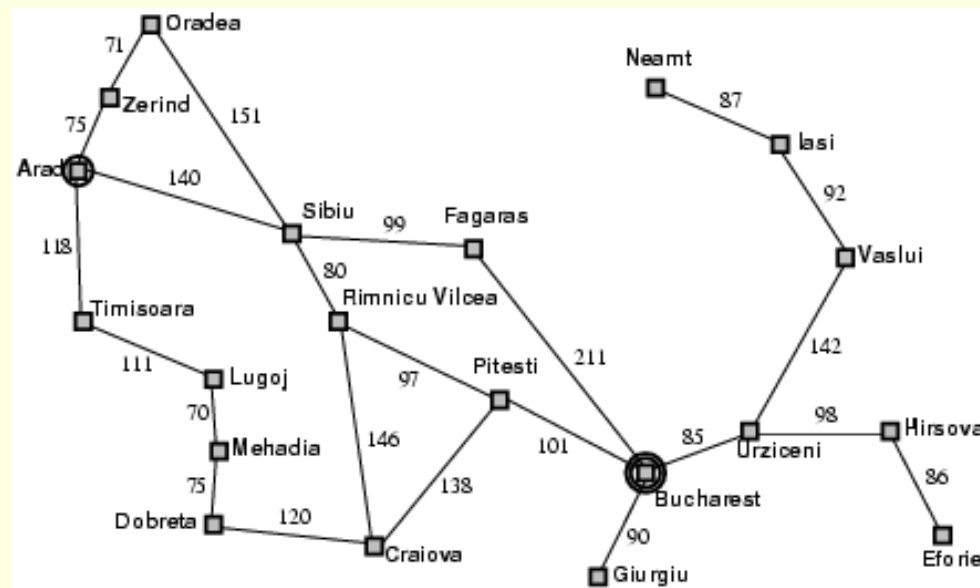
*Altfel*

`noduri = adauga(noduri, expandare(nod, adauga_la_inceput_daca_`  
`limita_permite))`

*Sfarsit cat timp*



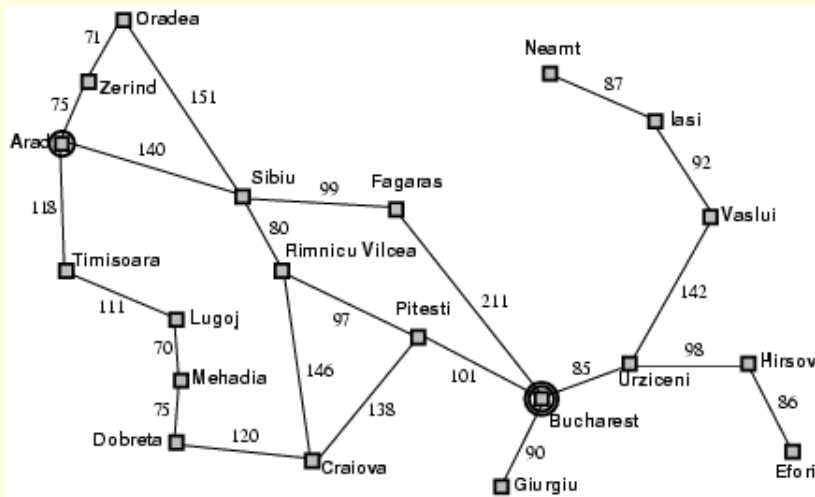
# Cautarea limitata in adancime



Stabilim limita de adancime egala cu 3.

Sa se gaseasca o ruta de la Arad la Bucuresti folosind cautarea limitata in adancime.

# Cautarea limitata in adancime



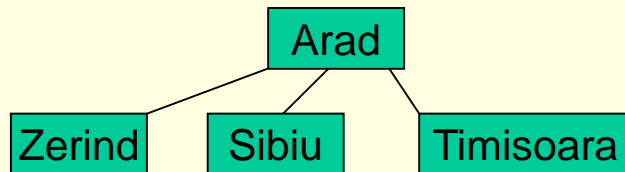
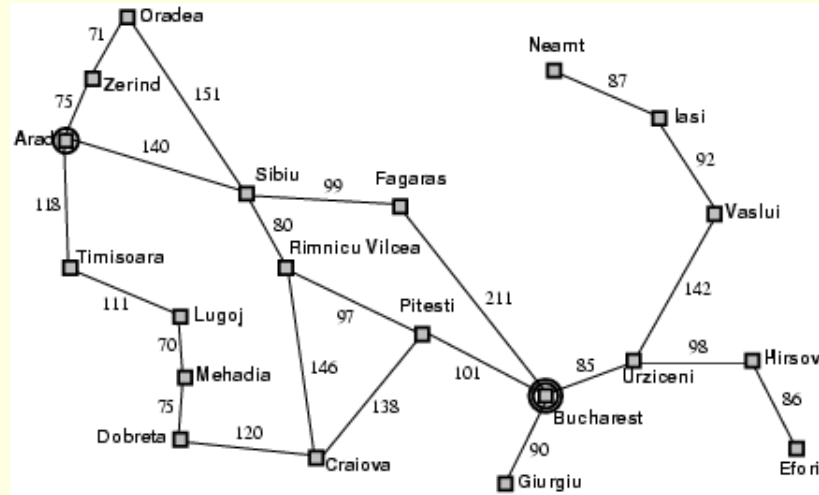
Arad

Fața	noduri	Sfarsit
Arad		
0		

Parcurgerea: Arad,

Adancime: 0

# Cautarea limitata in adancime

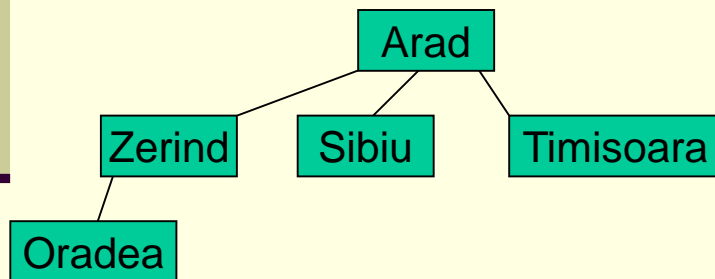
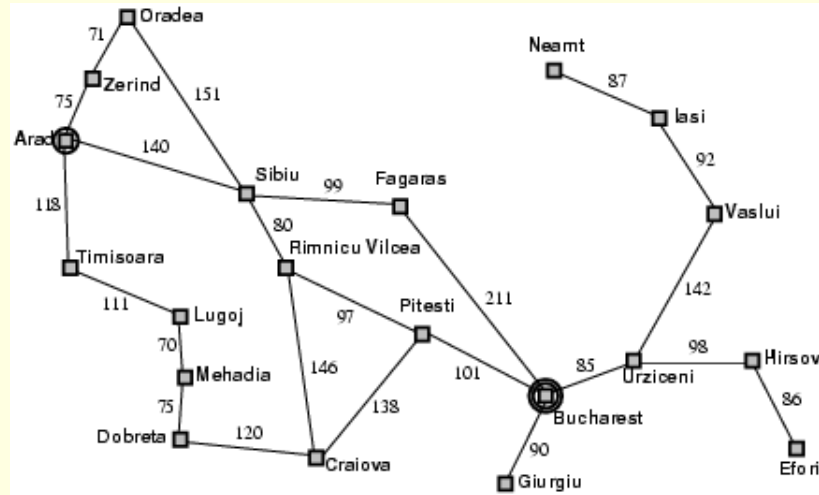


Fața	noduri	Sfarsit
Arad		
0		

Parcurgerea: Arad,

Adancime: 0

# Cautarea limitata in adancime

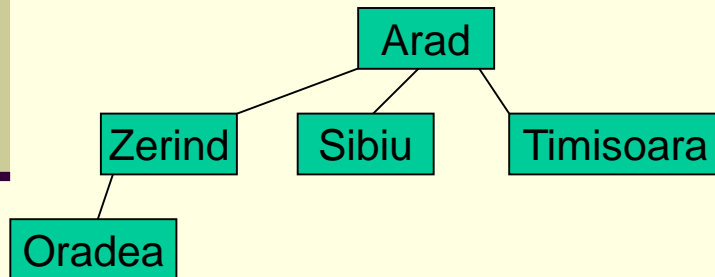
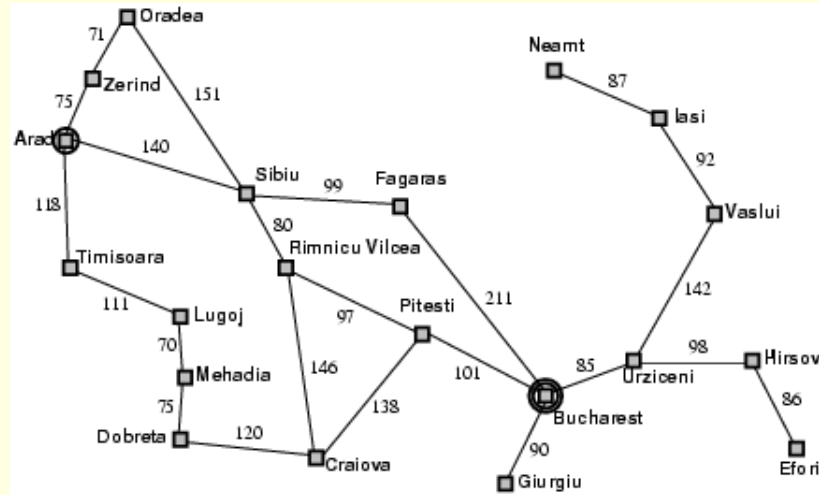


Fața	noduri			Sfarsit
Zerind	Sibiu	Timisoara		
1	1	1		

Parcurgerea: Arad, Zerind,

Adancime: 1

# Cautarea limitata in adancime

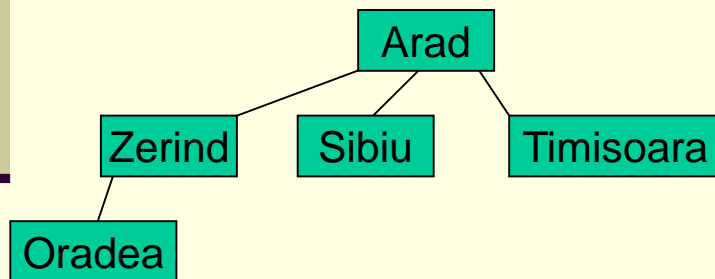
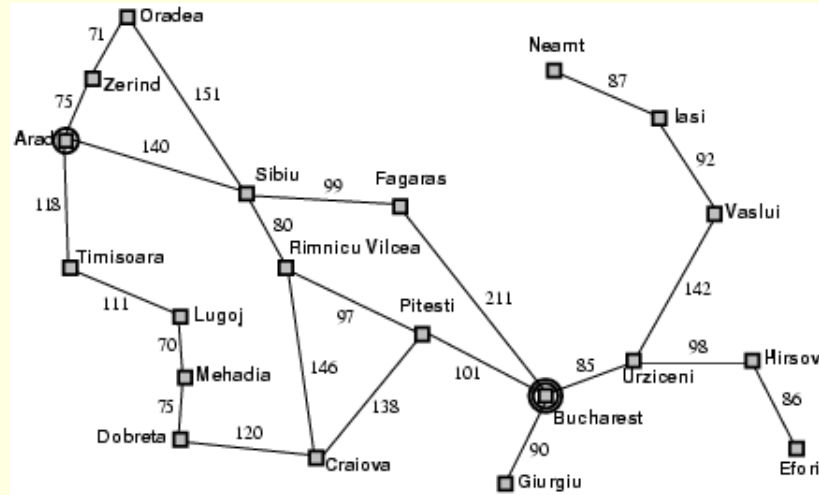


Fața	noduri		Sfarsit
Oradea	Sibiu	Timisoara	
2	1	1	

Parcurgerea: Arad, Zerind, Oradea

Adancime: 2

# Cautarea limitata in adancime

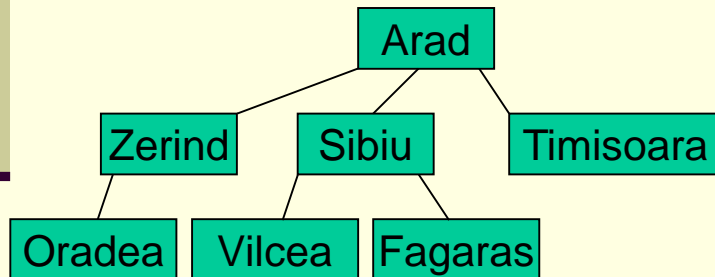
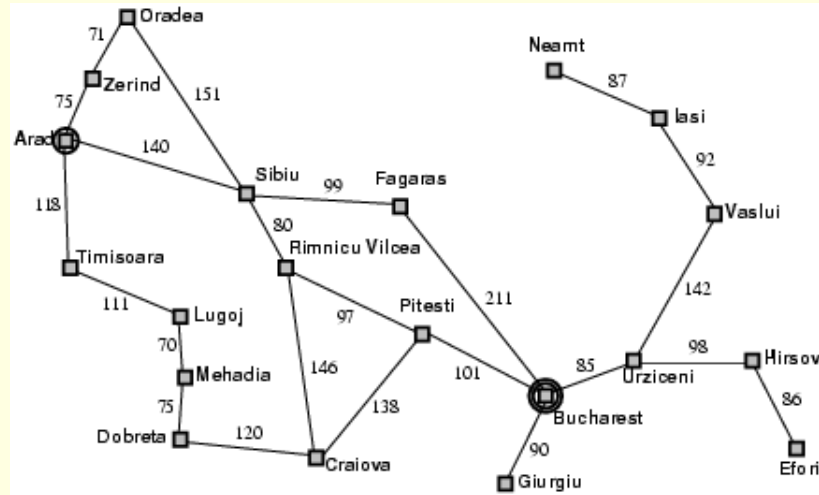


Fața	noduri	Sfarsit
Sibiu	Timisoara	
1	1	

Parcurgerea: Arad, Zerind, Oradea, Sibiu

Adancime: 1

# Cautarea limitata in adancime

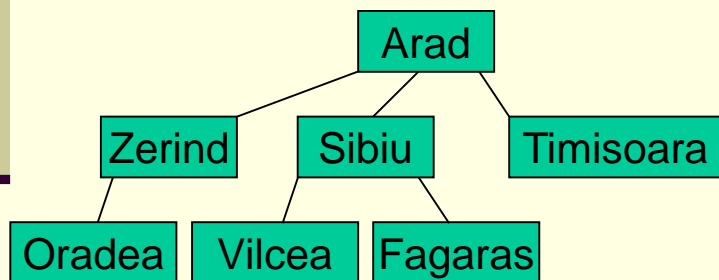
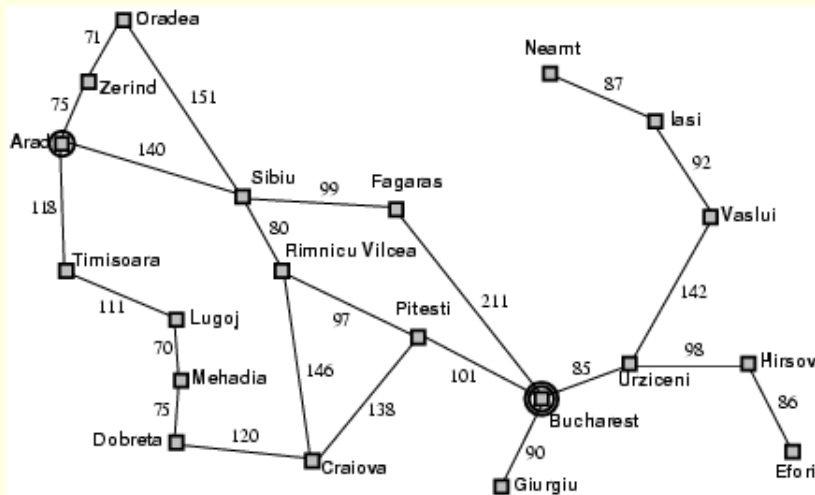


Fața	noduri	Sfarsit
Sibiu	Timisoara	
1	1	

Parcurgerea: Arad, Zerind, Oradea, Sibiu

Adancime: 1

# Cautarea limitata in adancime



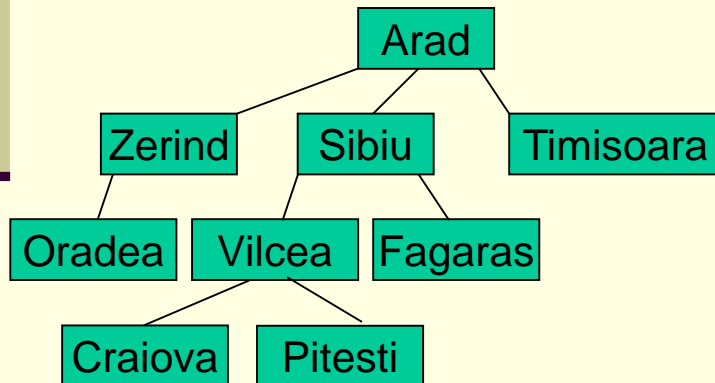
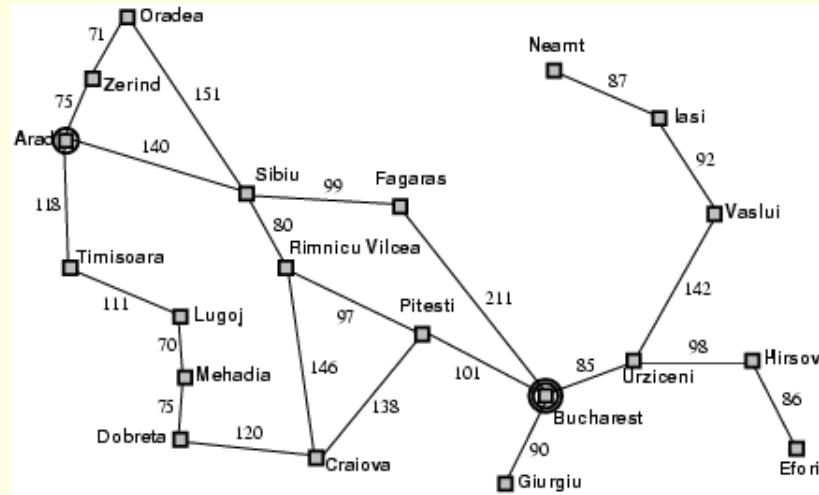
Fața	noduri			Sfarsit
Vilcea	Fagaras	Timisoara		
2	2	1		

Parcursul: Arad, Zerind, Oradea, Sibiu, Vilcea

Adancime: 2



# Cautarea limitata in adancime

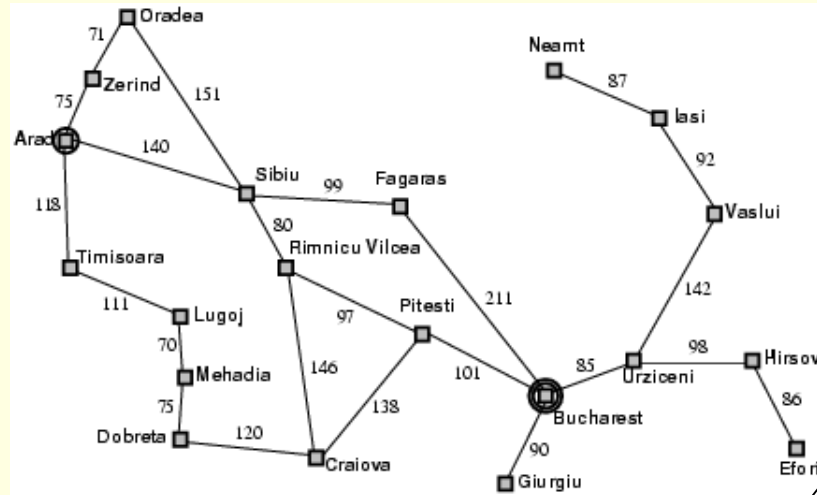


Fața	noduri			Sfarsit
Vilcea	Fagaras	Timisoara		
2	2	1		

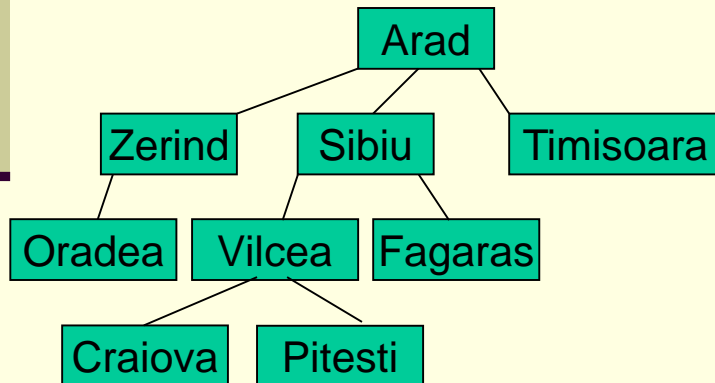
Parcursirea: Arad, Zerind, Oradea, Sibiu, Vilcea

Adancime: 2

# Cautarea limitata in adancime



Nu este nodul tinta!!

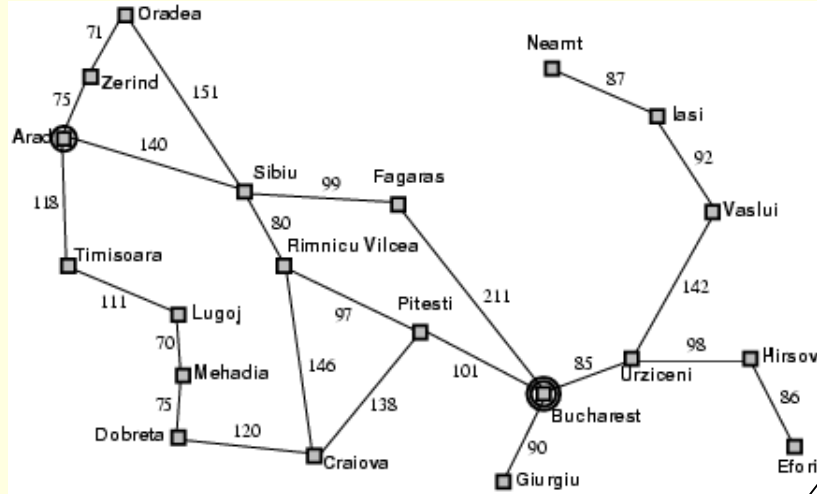


Fața	noduri	Sfarsit	
Craiova	Pitesti	Fagaras	Timisoara
3	3	2	1

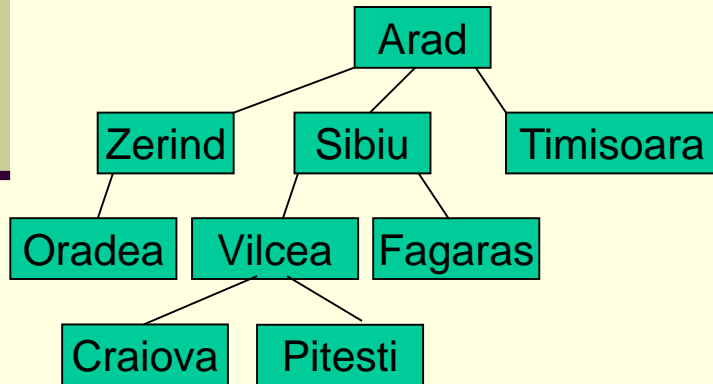
Parcursul: Arad, Zerind, Oradea, Sibiu, Vilcea, Craiova

Adancime: 3

# Cautarea limitata in adancime



Nu este nodul tinta!!

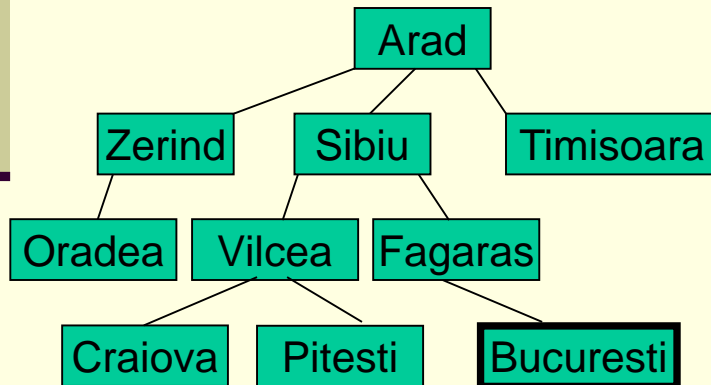
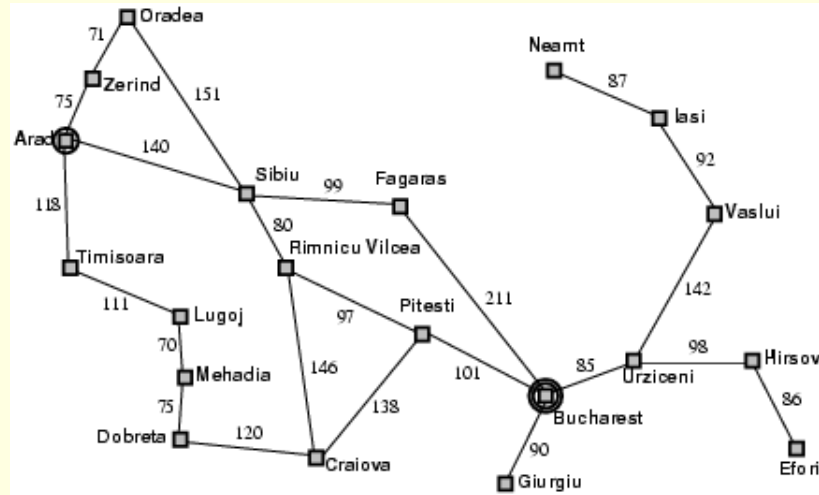


Fața	noduri			Sfarsit
Pitesti	Fagaras	Timisoara		
3	2	1		

Parcursirea: Arad, Zerind, Oradea, Sibiu, Vilcea, Craiova, Pitesti

Adancime: 3

# Cautarea limitata in adancime

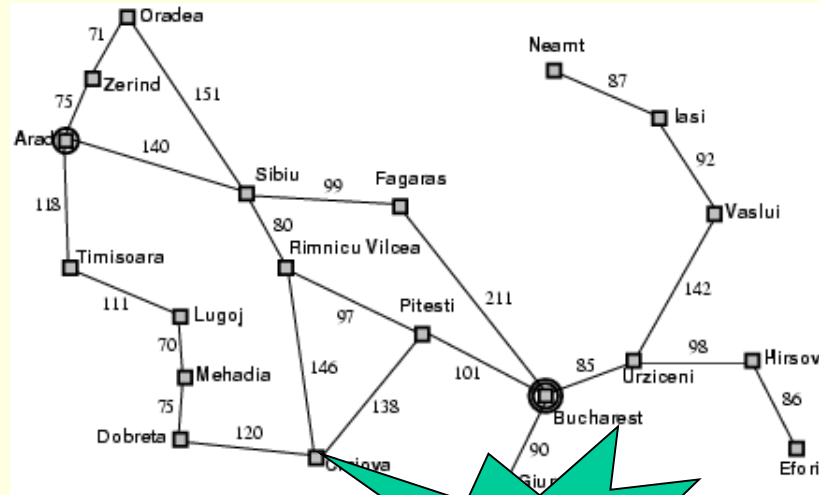


Fața	noduri	Sfarsit
Fagaras	Timisoara	
2	1	

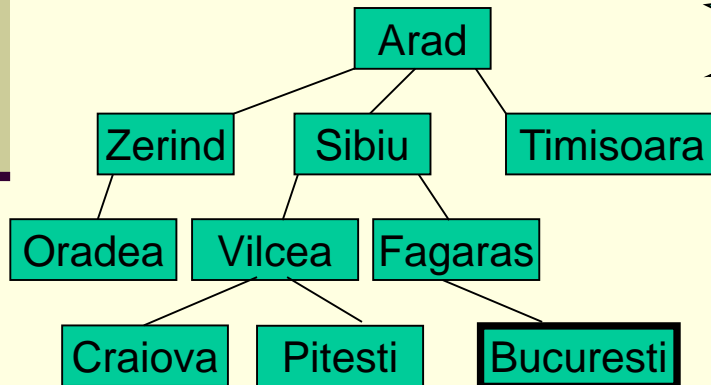
Parcursul: Arad, Zerind, Oradea, Sibiu, Vilcea, Craiova, Pitesti, Fagaras

**Adancime: 2**

# Cautarea limitată în adâncime



**Nod tinta!**

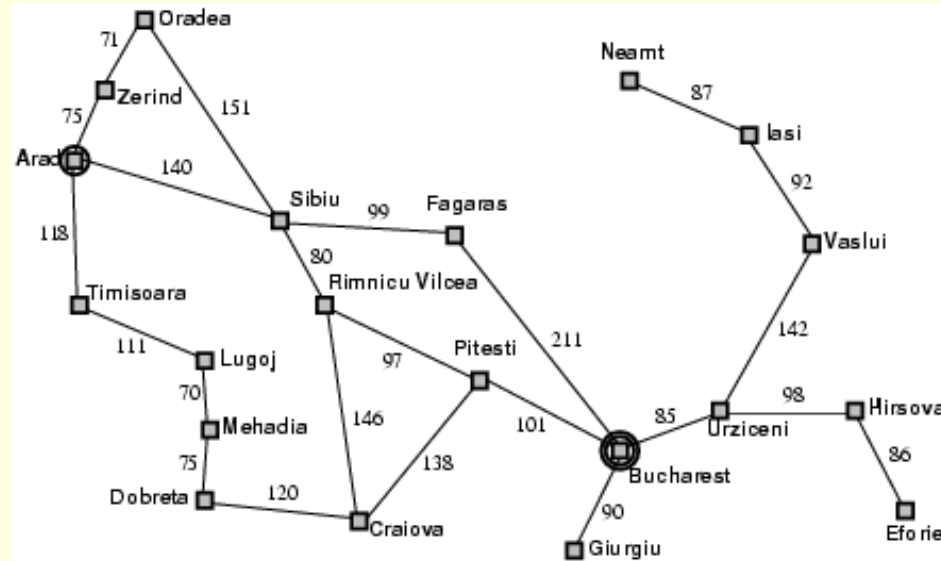


Fața	noduri	Sfarsit
Bucuresti	Timisoara	
3	1	

Parcursul: Arad, Zerind, Oradea, Sibiu, Vilcea, Craiova, Pitesti, Fagaras, Bucuresti.

**Adâncime: 3**

# Exercitiu



Fața                      **noduri**                      Sfarsit



**Parcurerea:** Bucuresti, ...,  
Rm. Vilcea

Gasiti o ruta de la Bucuresti la Rimnicu Vilcea folosind parcurerea limitata in adancime cu limita 3.  
Desenati arborele, scrieti parcurerea si continutul pentru noduri la fiecare pas. Alegeti descendenti in ordine alfabetica.

# 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  $\infty$  executa*

*Daca `cautare_adancime_limitata`(problema, adancime)*

*gaseste solutia atunci*

***intoarce solutia***

*Sfarsit daca*

*Sfarsit pentru*

# Cautarea cu adancime iterativa

---



Limita 0

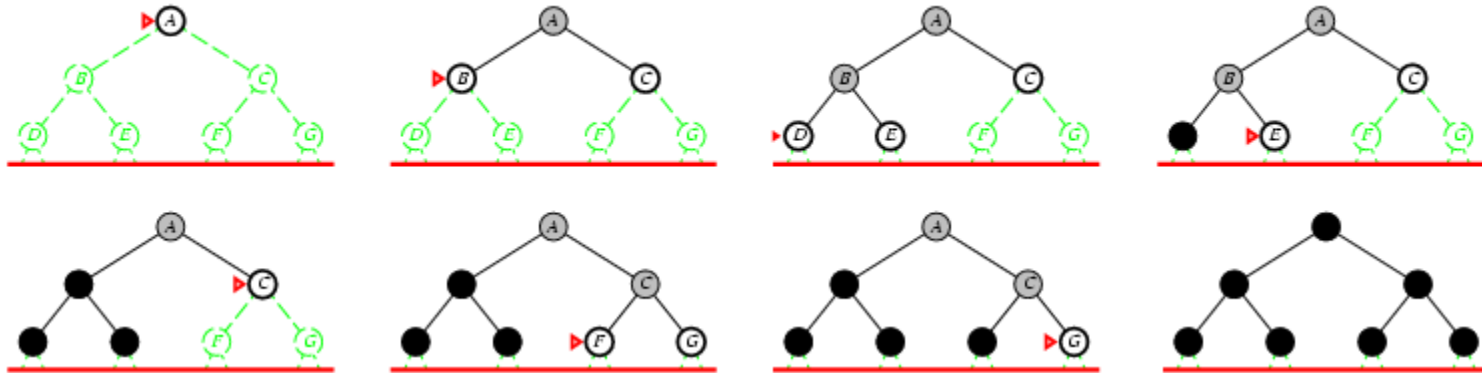


# Cautarea cu adancime iterativa



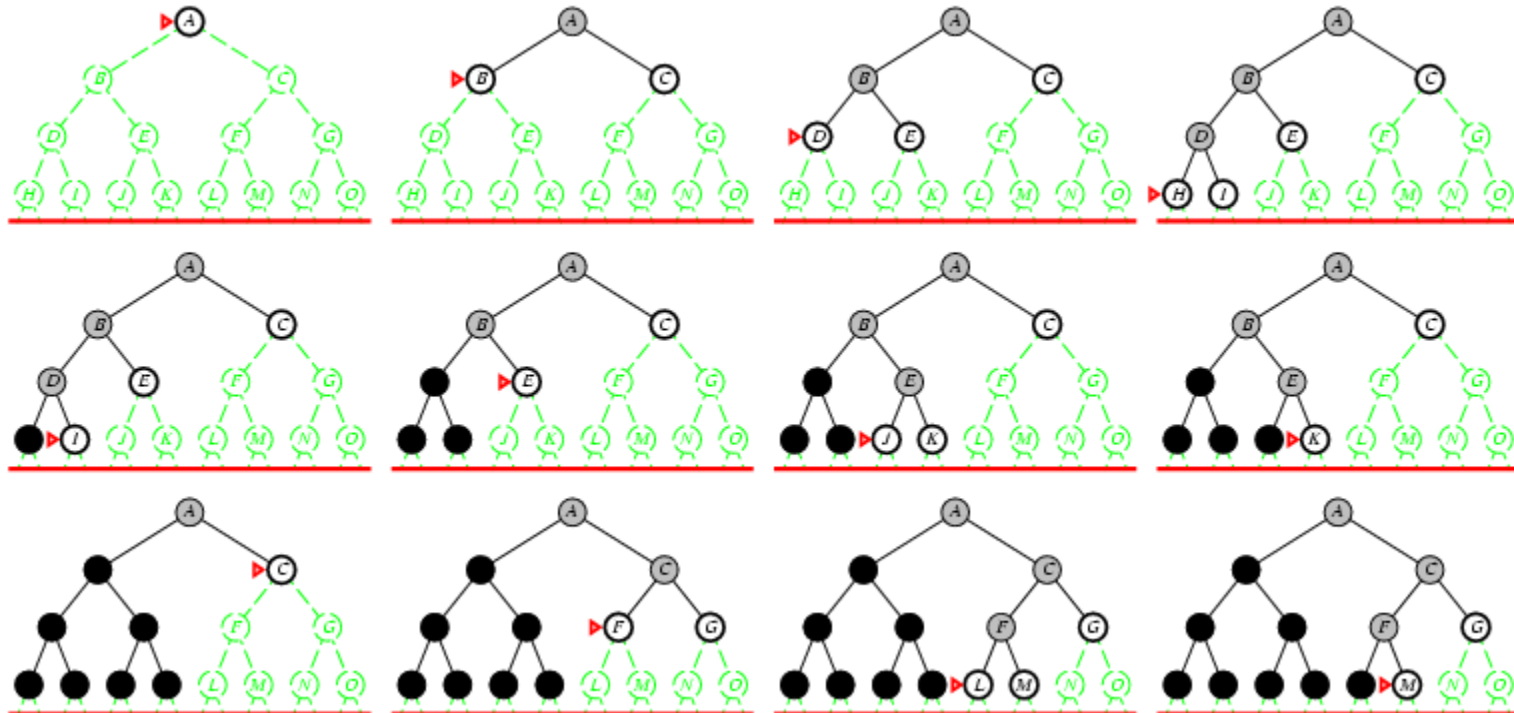
Limita 1

# Cautarea cu adancime iterativa



Limita 2

# Cautarea cu adancime iterativa



Limita 3

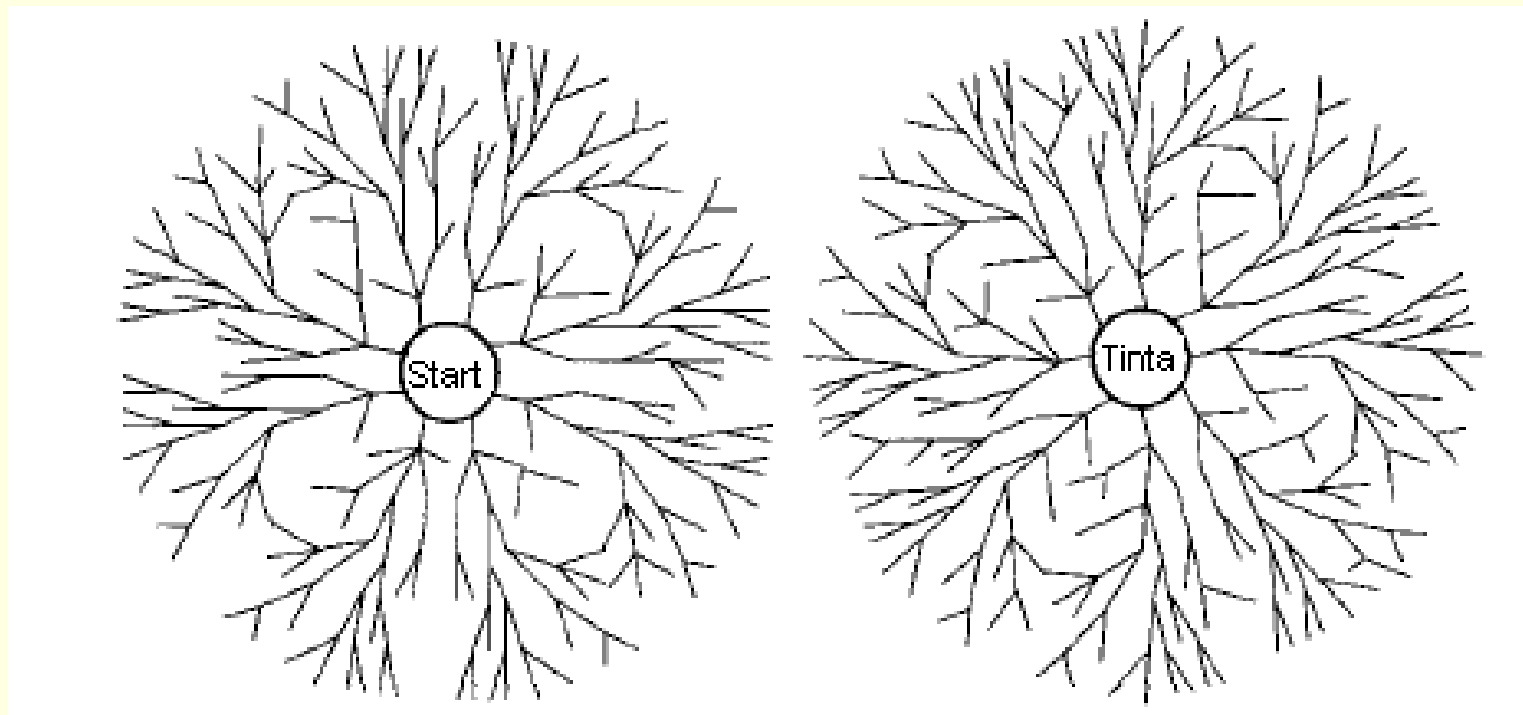
# Cautarea cu adancime iterativa

---

- 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 bidirecțională

- Ideea este de a cauta în același timp pornind de la starea inițială și de la starea țintă cu scopul de a întâlni cele două căutări la mijloc.



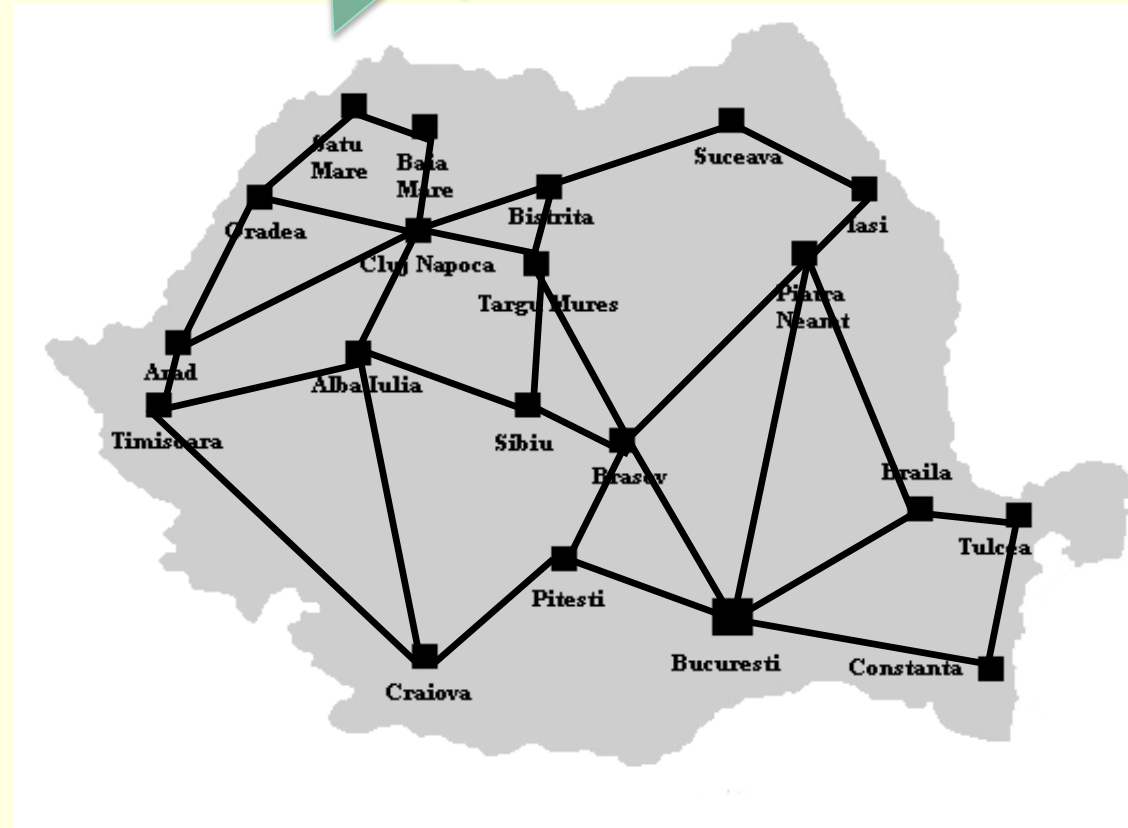
# Cautarea bidirectionala

- 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?

# Tema...

Un punct la  
examenul  
final!

- Implementați un algoritm de căutare bidirecțională pentru harta alăturată a. i. să avem combinații din următoarele căutări:
  - Latime
  - Adâncime
- Numărați câte orașe distincte trec prin lista “noduri”.



# Exercitiu

- Consideram distantele directe dintre 9 puncte date ca in tabelul de mai jos. Cazul in care intr-o casuta din tabel nu este trecuta nicio valoare semnifica faptul ca nu exista un drum direct intre cele doua puncte (de exemplu, intre A si C). Sa se aplice cautarea bidirectionala pentru gasirea drumului de la J la A, astfel incat din J sa se porneasca cu cautarea cu cost uniform, iar din A cu cautarea in latime: reprezentati arborii si scrieti ruta obtinuta in final.

	A	B	C	D	E	F	G	H	J
A	0	10	-	-	7	-	-	-	-
B	10	0	2	-	-	5	-	-	-
C	-	2	0	10	-	4	11	-	-
D	-	-	10	0	12	7	-	-	-
E	7	-	-	12	0	4	-	-	-
F	-	5	4	7	4	0	-	8	-
G	-	-	11	-	-	-	0	-	10
H	-	-	-	-	-	8	-	0	15
J	-	-	-	-	-	-	10	15	0



# 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	$b^d$	$b^d$	<b><math>bm</math></b>	<b><math>bl</math></b>	<b><math>bd</math></b>	$b^{d/2}$
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.

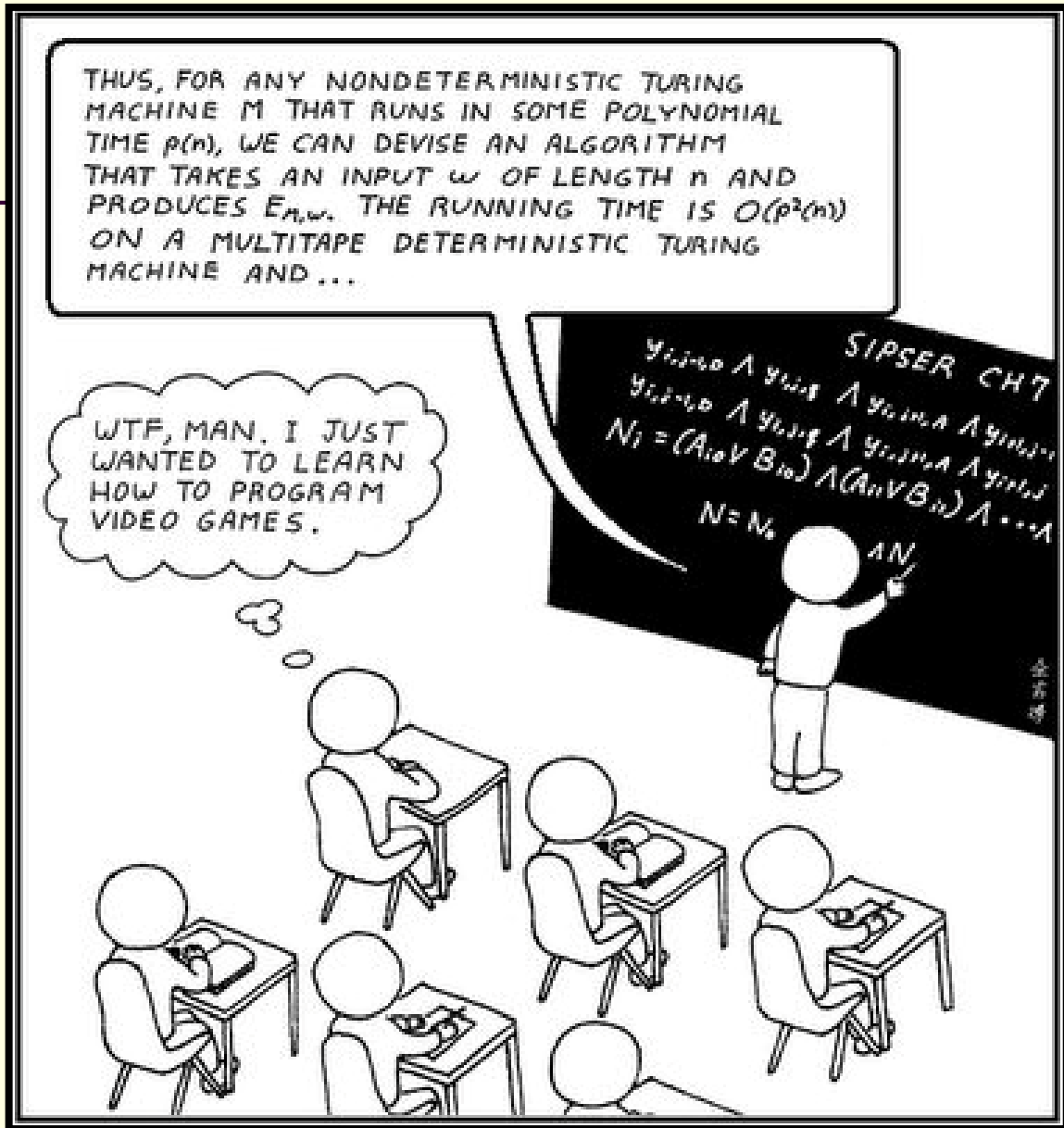
# Evitarea starilor de repetare

---

- 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.

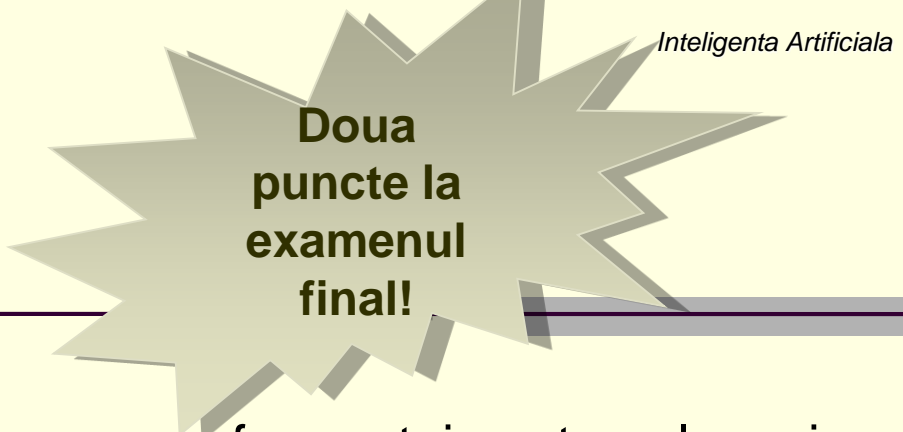
# Satisfacerea constrangerilor

- Intr-o problema cu satisfacere de constrangeri, starile sunt definite prin valorile pe care le iau o multime de **variabile**, iar in testarea tinteii 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 monstru si 0 altfel. Al treilea devine 1 daca intra intr-o celula unde se gaseste o comoara si 0 altfel.
- **Actiuni:** mergi in fata, intoarce la dreapta cu  $90^\circ$ , la stanga cu  $90^\circ$ , impusca, oprire.
  - O impuscatura tinteste spre celula din fata exploratorului, iar daca aceasta contine un monstru, il ucide.

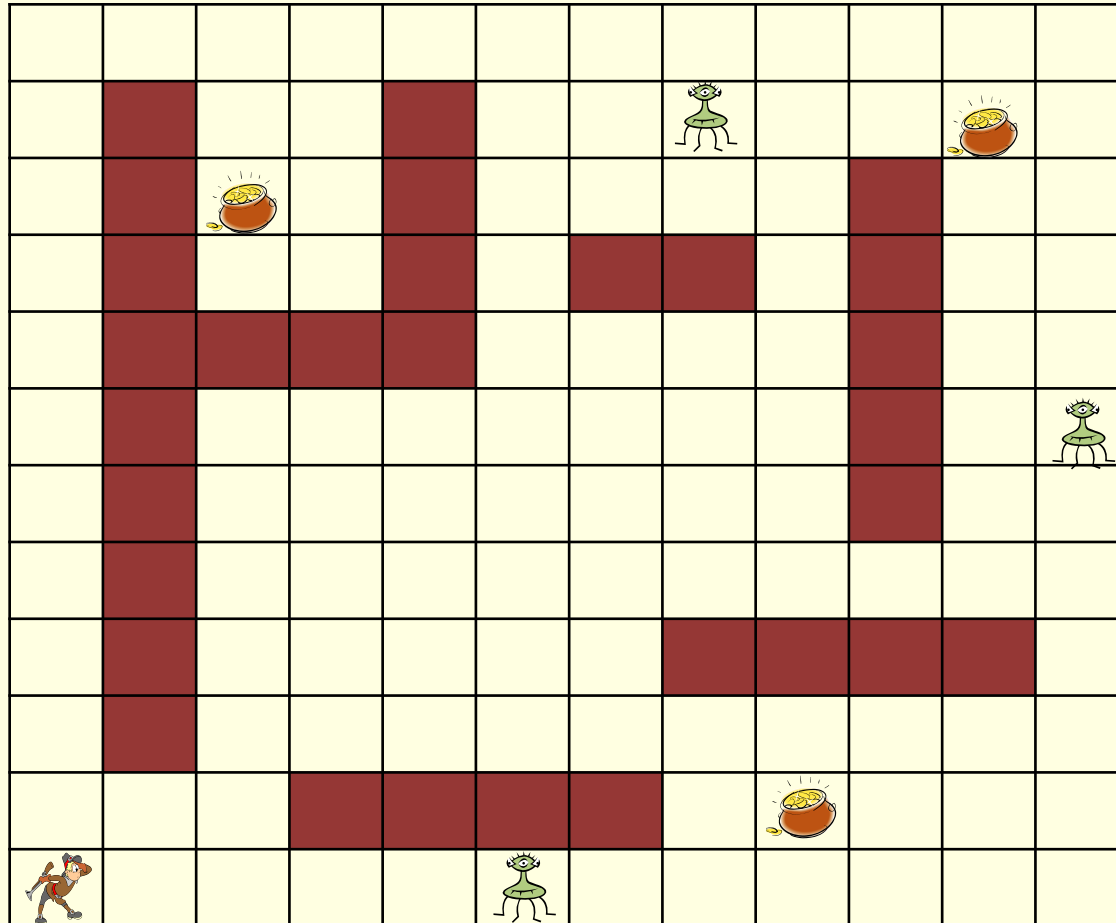
# Tema 2/3

---

- **Tinta:**
  1. Sa gaseasca comoara, sa nu intre in celula unui monstru viu. Masura de performanta 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.
  2. Aplicam cautarea in latime pentru a gasi o comoara.
- **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

			
Explorator	Comoara	Monstru	Locatia



# Recapitulare 1/2

- **Cautarea in latime** gaseste solutia care se afla cel mai aproape de nodul radacina.
  - Complet
  - Optim, daca fiecare actiune are acelasi cost
  - Complexitatea temporală și spațială:  $O(b^d)$
- **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 temporală  $O(b^m)$  și pe cea spațială  $O(bm)$ , unde  $m$  este adancimea maxima. Daca arborele este de adancime foarte mare sau infinita, aceasta cautare este nepractica.



# Recapitulare 2/2

---

- **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 spatiaza sunt minimizezate.
- **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 spatiaza  $O(bd)$ .
- **Cautarea bidirectionala** reduce complexitatea temporala foarte mult inza nu este aplicabila in orice caz.