Agenti care rezolva probleme

Catalin Stoean

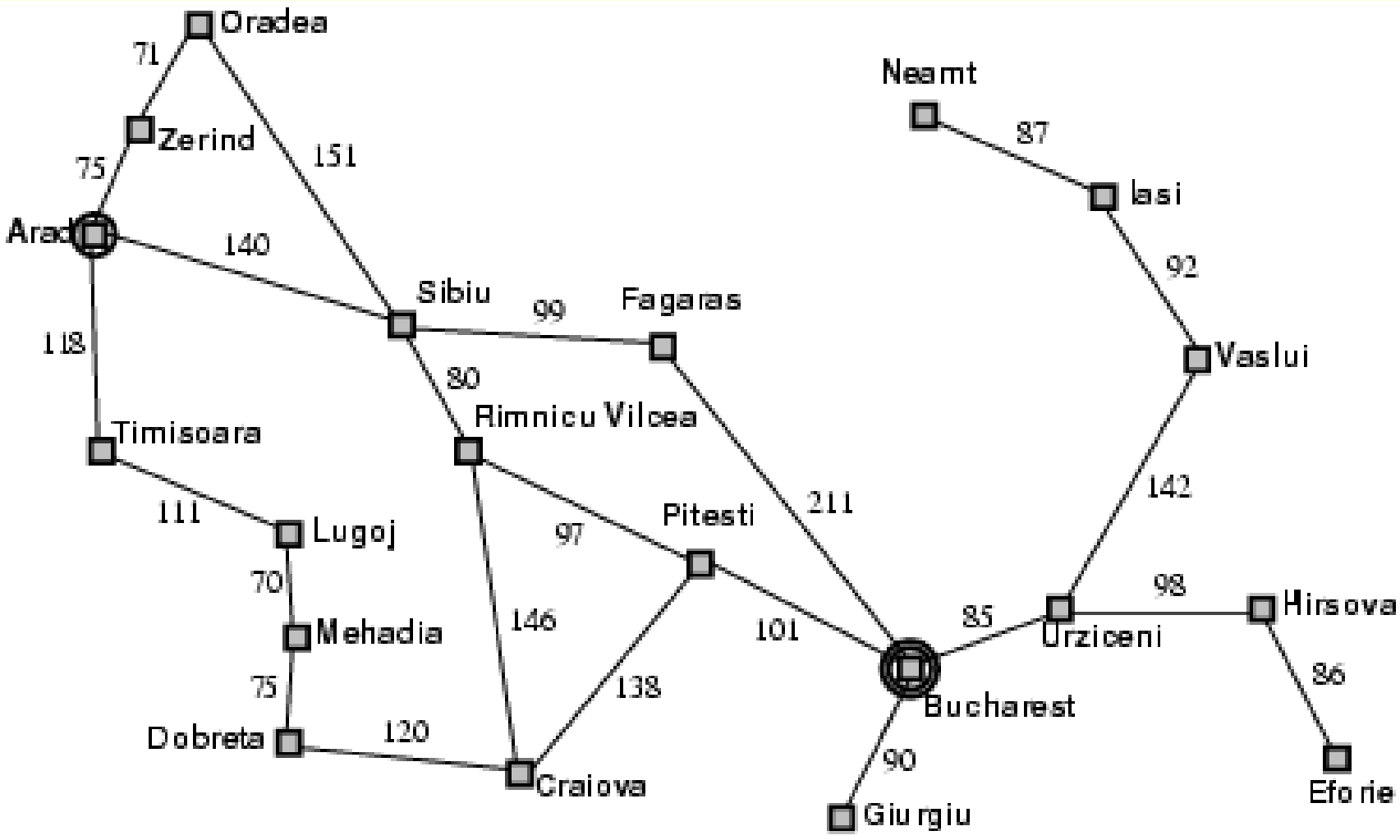
catalin.stoean@inf.ucv.ro

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

Un agent *american*

- Vacanta in Romania – in Arad.
- In ziua urmatoare ii pleaca avionul din Bucuresti.
- **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*



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*
    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

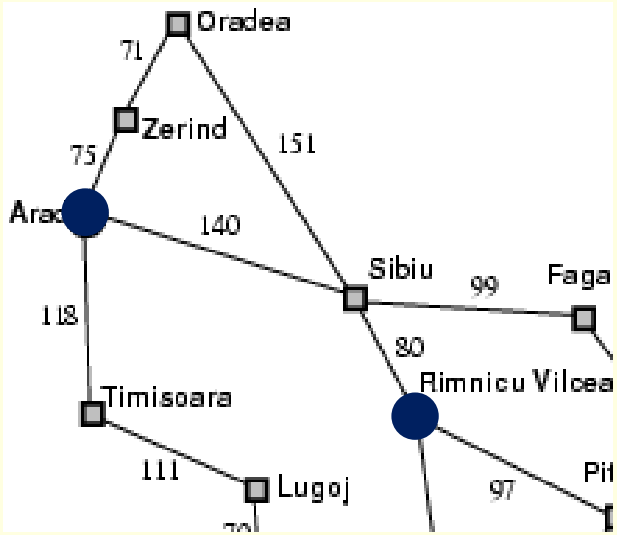
functia cautare_latime(*problema*) ***intoarce*** solutie sau esec
noduri = genereaza_lista(genereaza_nod(stare_initiala[problema]))

Cat timp solutie negasita si noduri ≠ ∅ *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.



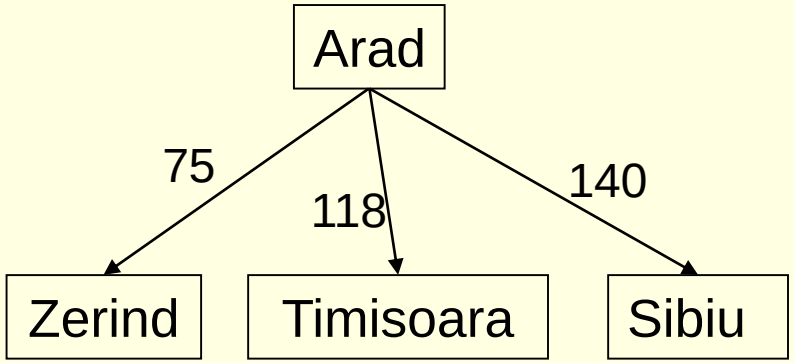
Algoritm cautare cu cost uniform

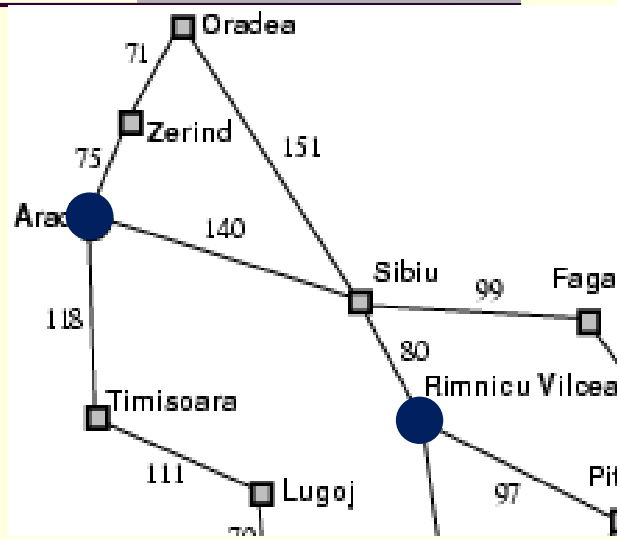
functia cautare_cost_uniform(*problema*) ***intoarce*** solutie sau esec
noduri = genereaza_lista(genereaza_nod(stare_initiala[problema]))

Cat timp solutie negasita si noduri ≠ ∅ *executa*
    nod = scoate_din_fata(noduri)
    Daca testare_tinta[problema] se aplica la stare(nod) *atunci*
        solutie gasita
    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

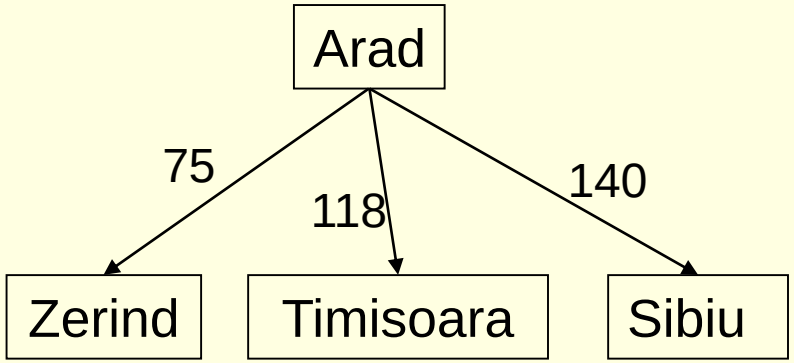


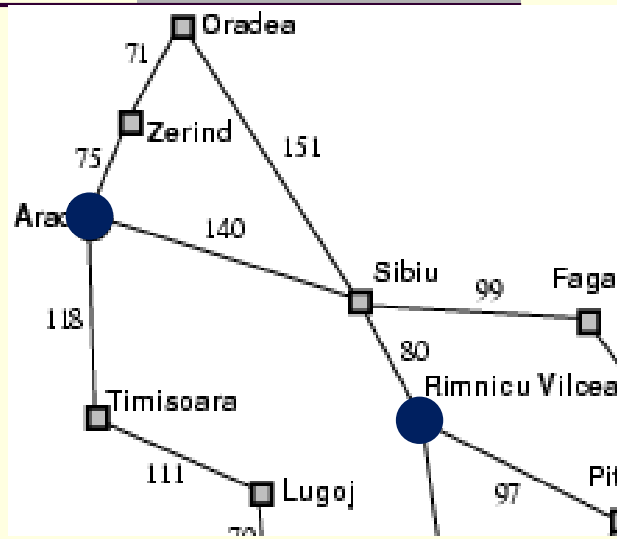


Faţa	**noduri**	Sfarsit
Arad		
0		

Parcurgerea: Arad,

Cautarea cu cost uniform

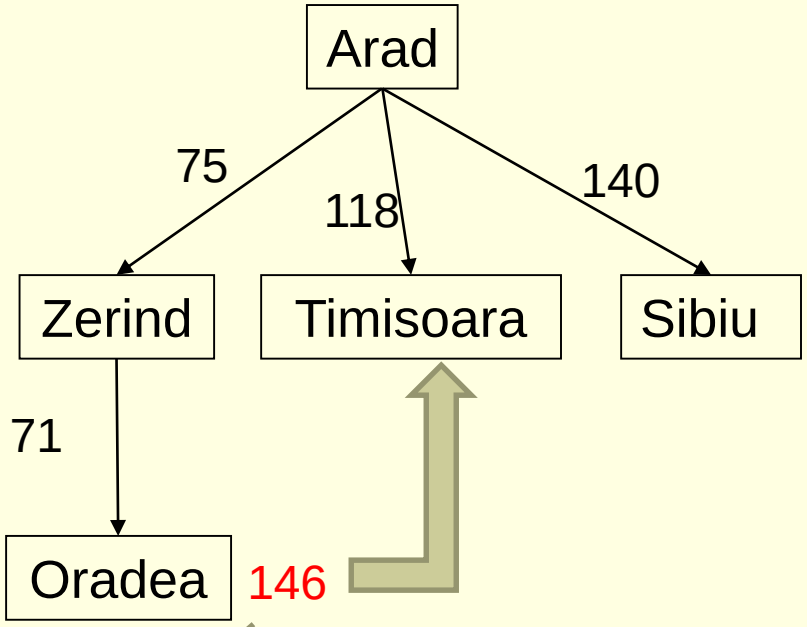


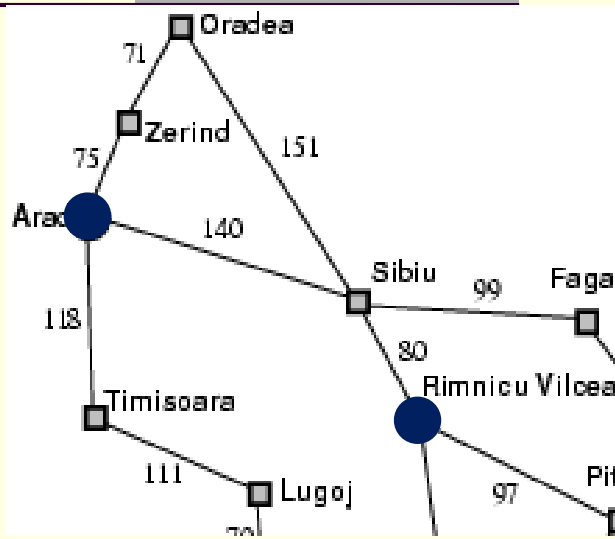


Faţa	**noduri**		Sfarsit
Zerind	TM	SB	
75	118	140	

Parcurgerea: Arad,

Cautarea cu cost uniform



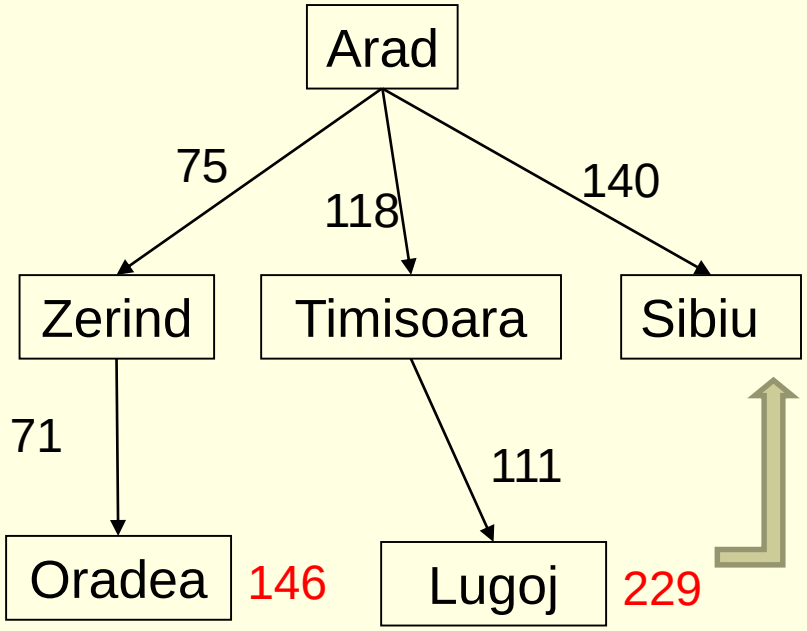


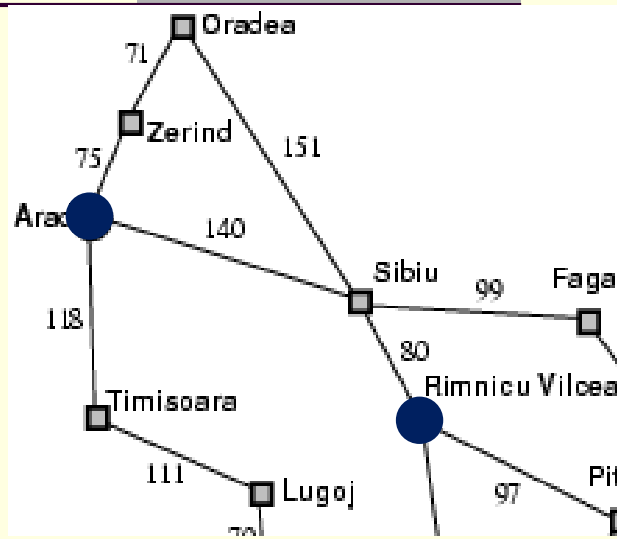
In total, de la Arad la Oradea.

Faţa	**noduri**		Sfarsit
TM	SB	Oradea	
118	140	146	

Parcurgerea: Arad, Zerind

Cautarea cu cost uniform



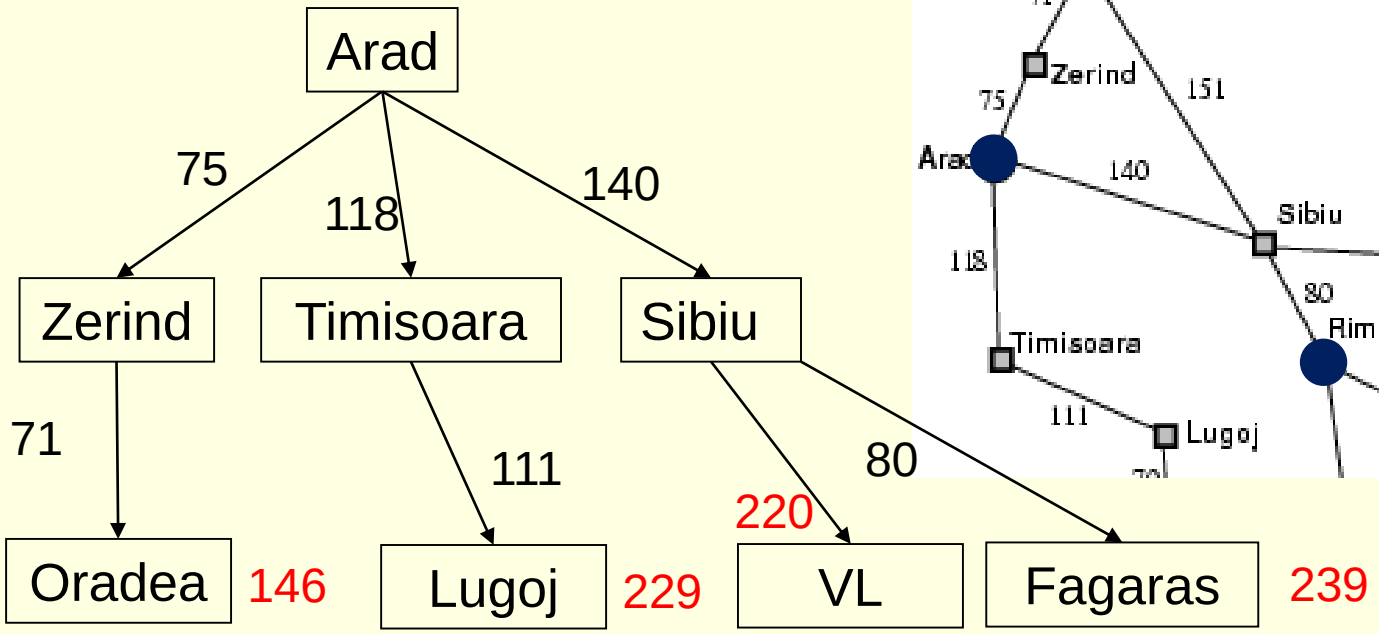


Faţa **noduri** Sfarsit

SB	Oradea	Lugoj	
140	146	229	

Parcurgerea: Arad, Zerind, TM

Cautarea cu cost uniform

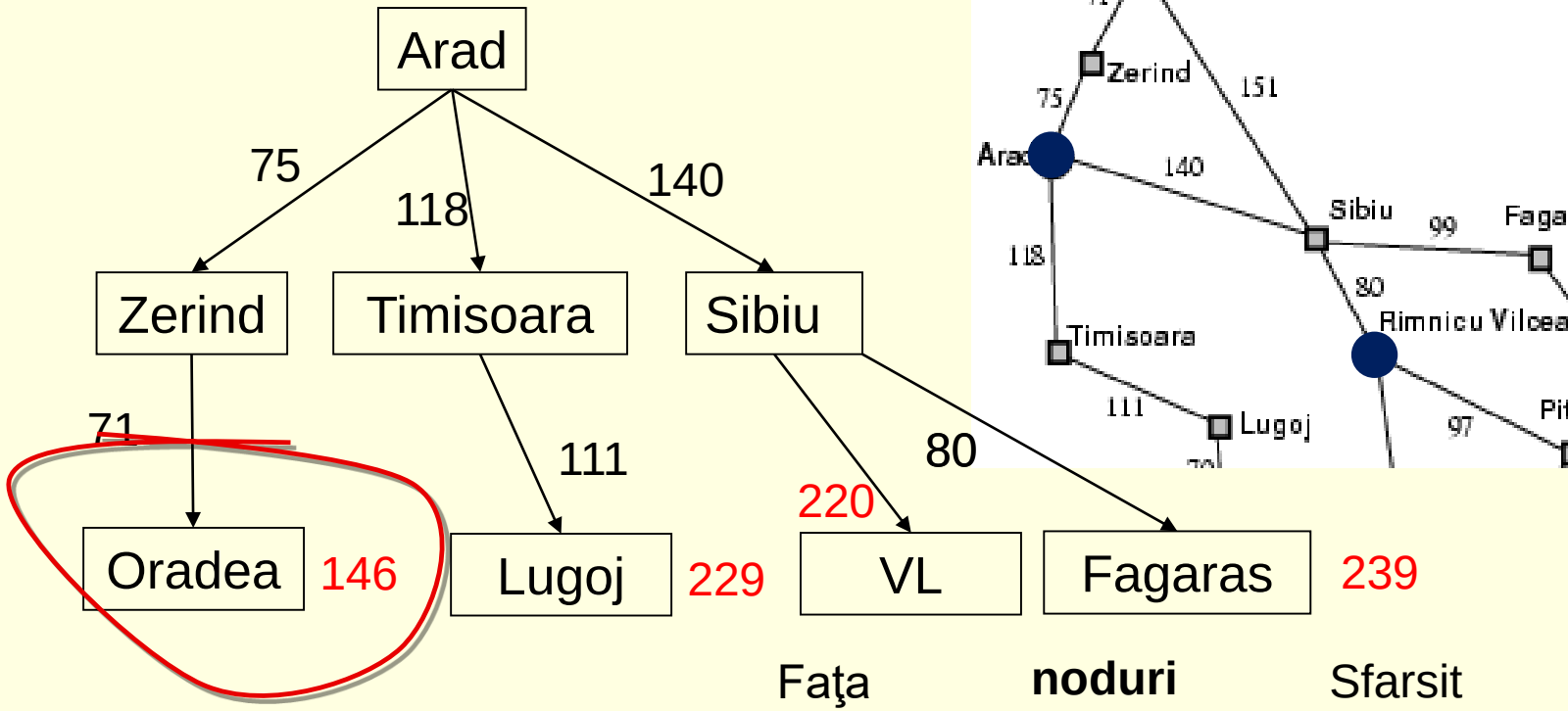


Faţa **noduri** Sfarsit

Oradea	VL	Lugoj	Fagaras	
146	220	229	239	

Parcurgerea: Arad, Zerind, TM, SB

Cautarea cu cost uniform



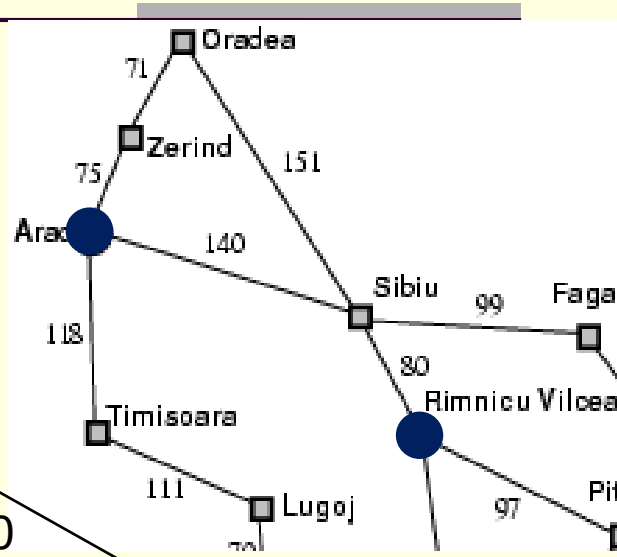
Faţa **noduri** Sfarsit

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



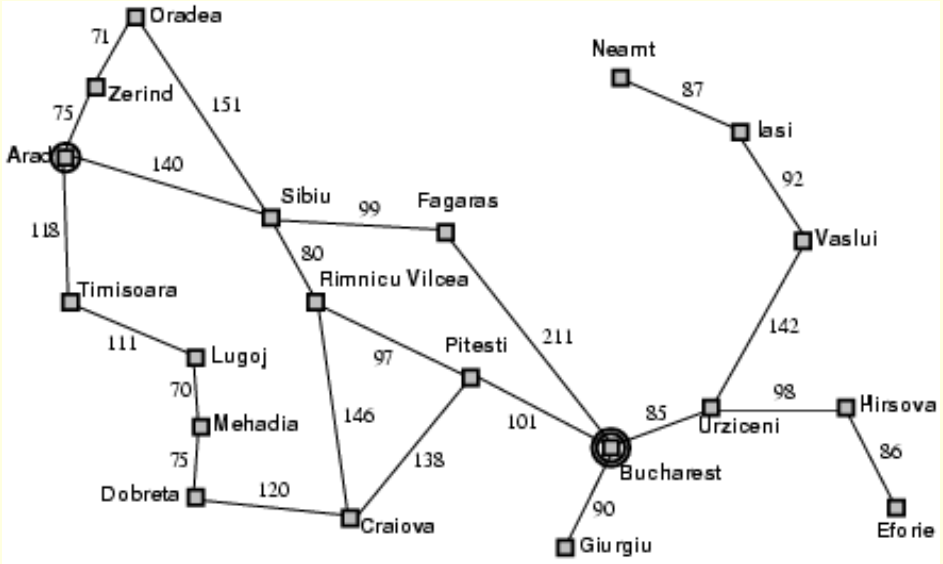
Nu adaugam SB pentru ca are o evaluare mai mare decat vechea valoare a orasului.

Faţa **noduri** Sfarsit

VL	Lugoj	Fagaras	
220	229	239	

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

Exercitiu



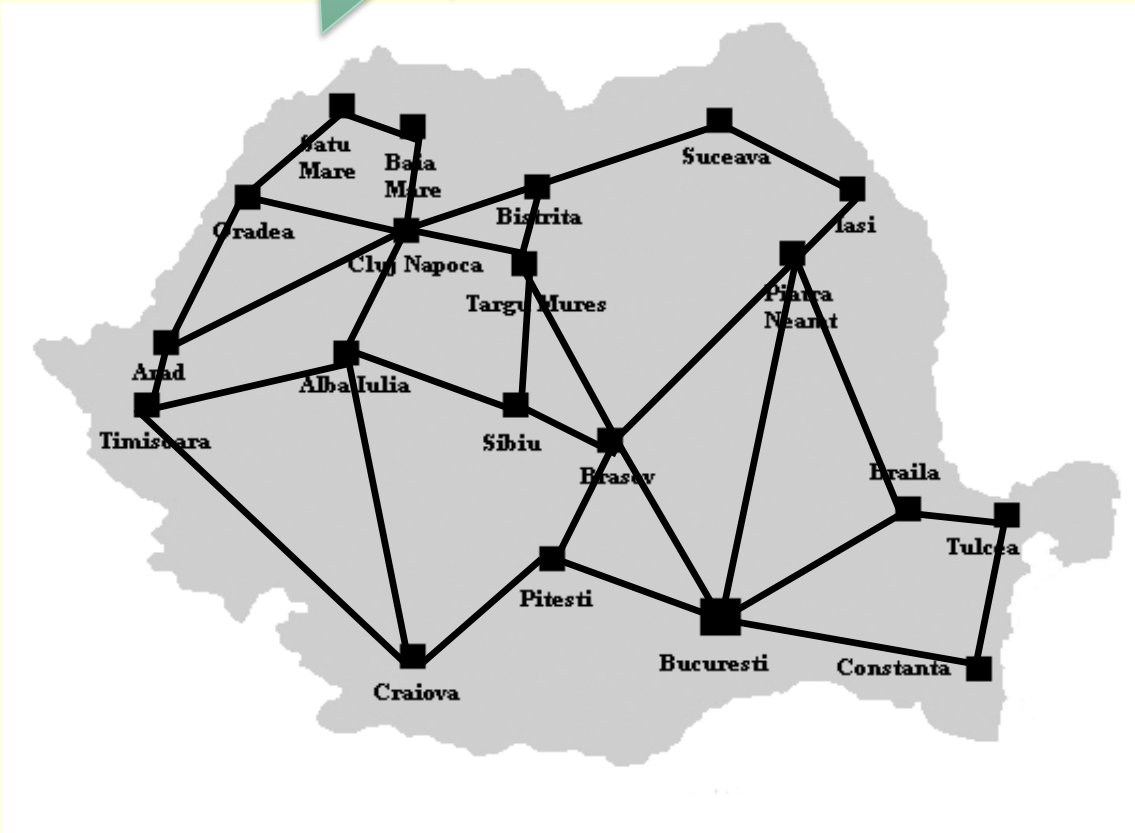
Faţa **noduri** Sfarsit

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…

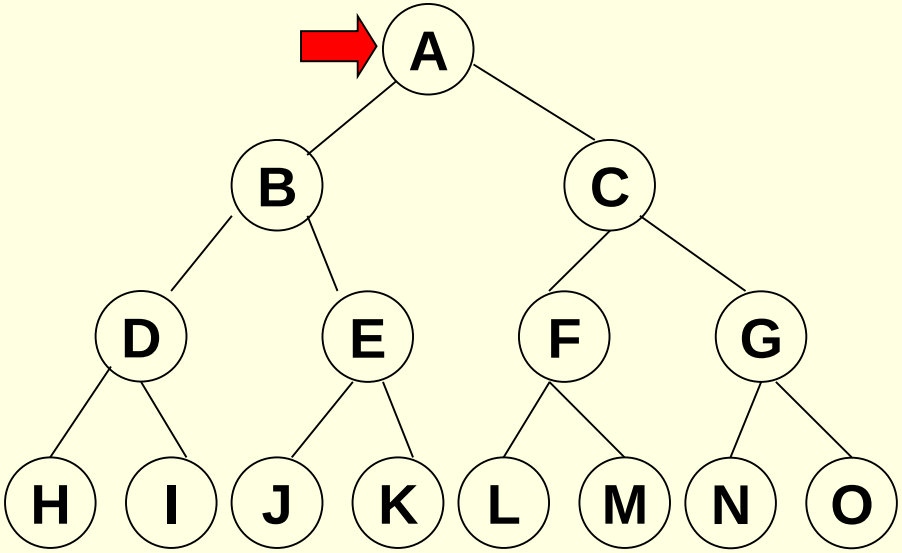
Un punct la examenul final!

- Gasiti distantele rutiere dintre orasele de pe harta din figura. Utilizati-le apoi pentru a implementa un algoritm de cautare cu cost uniform pentru a ajunge de la Oradea la Tulcea.



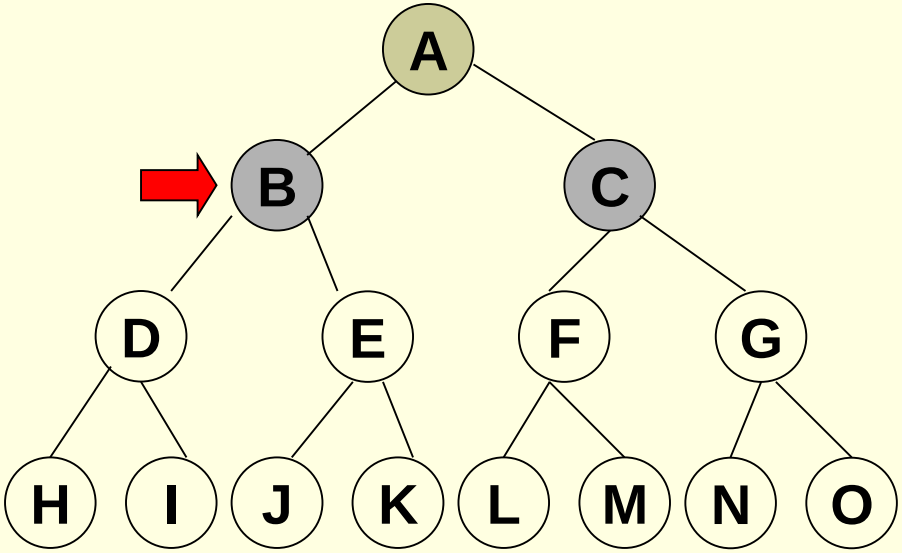
Cautarea in adancime

- 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 nivele mai putin adanci.



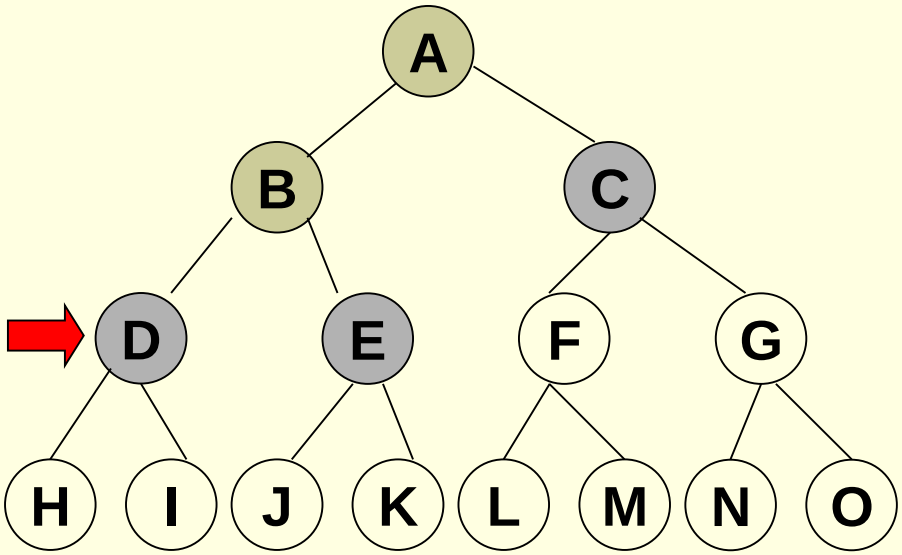
Cautarea in adancime

- 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.



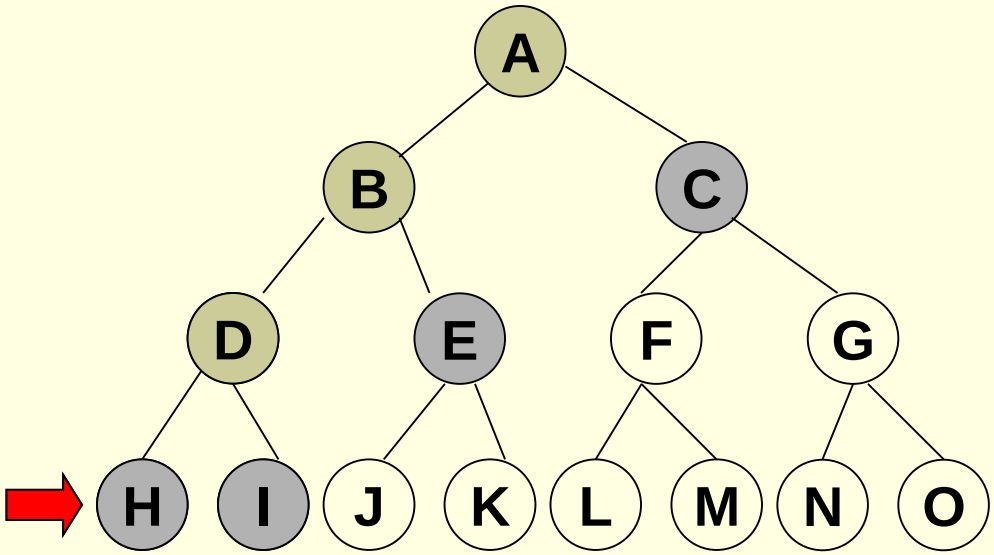
Cautarea in adancime

- 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.



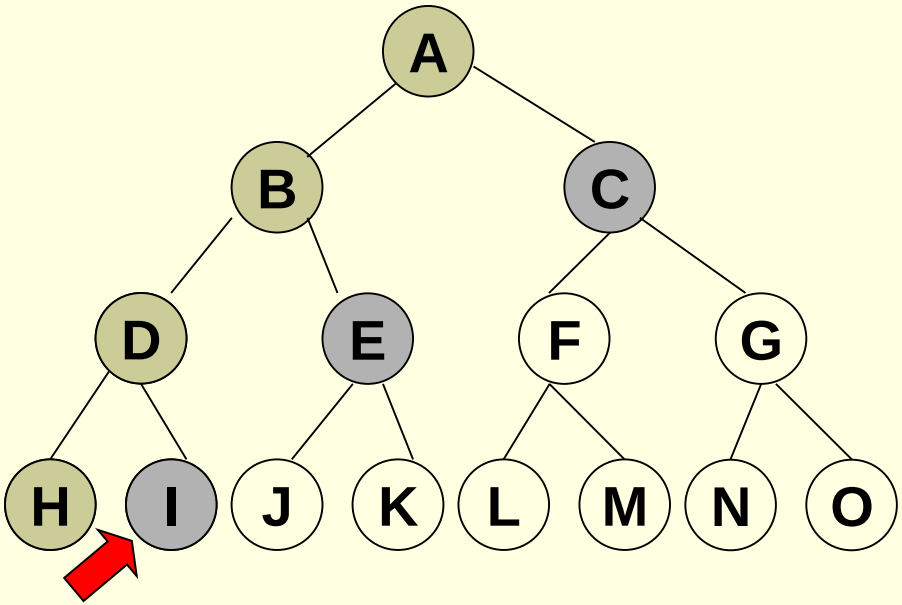
Cautarea in adancime

- 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.



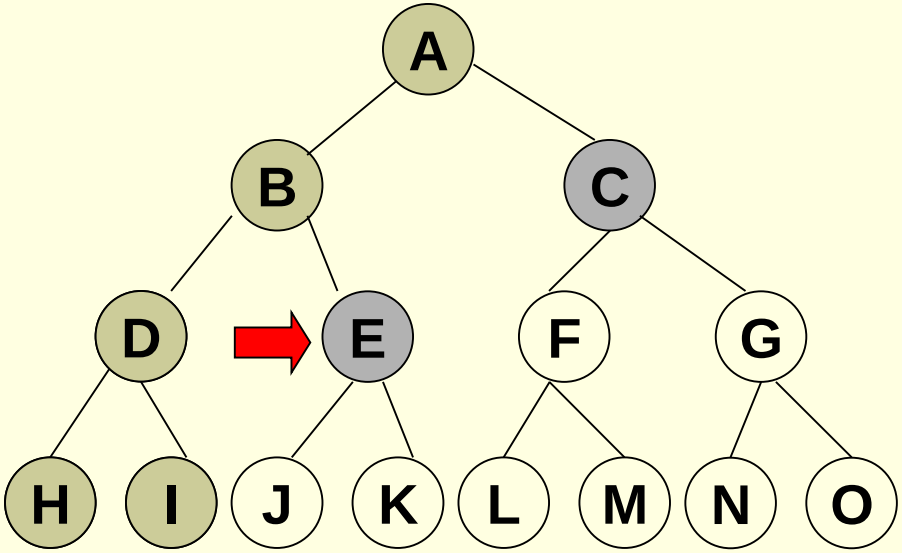
Cautarea in adancime

- 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.



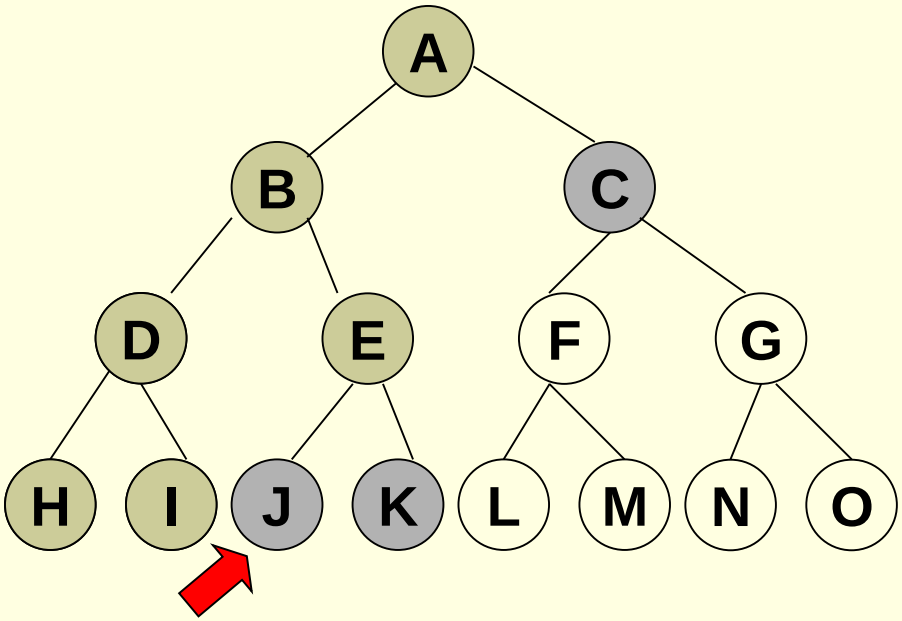
Cautarea in adancime

- 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.



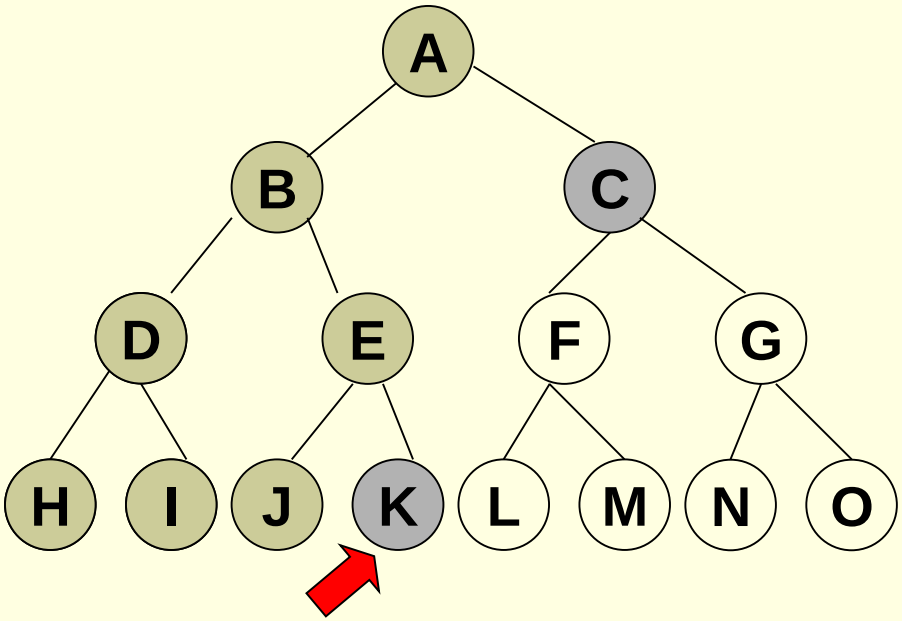
Cautarea in adancime

- 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.



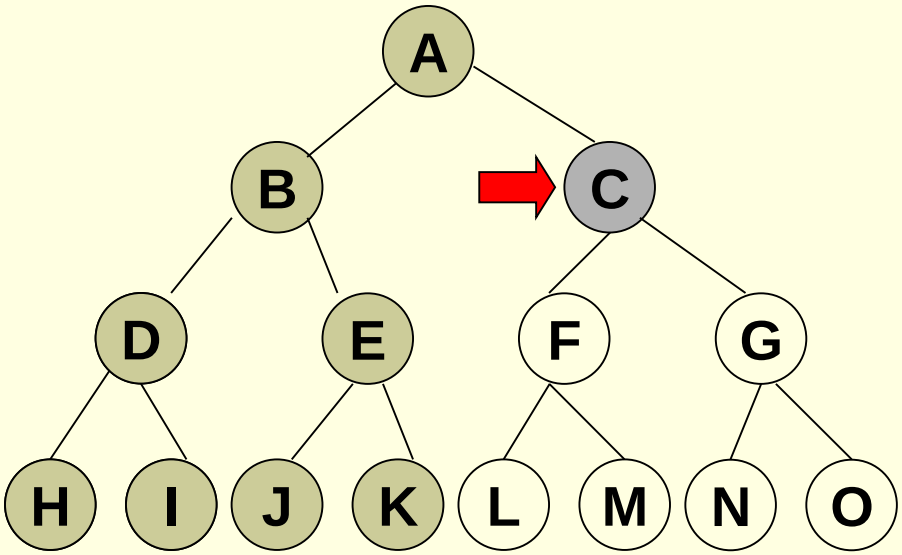
Cautarea in adancime

- 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.



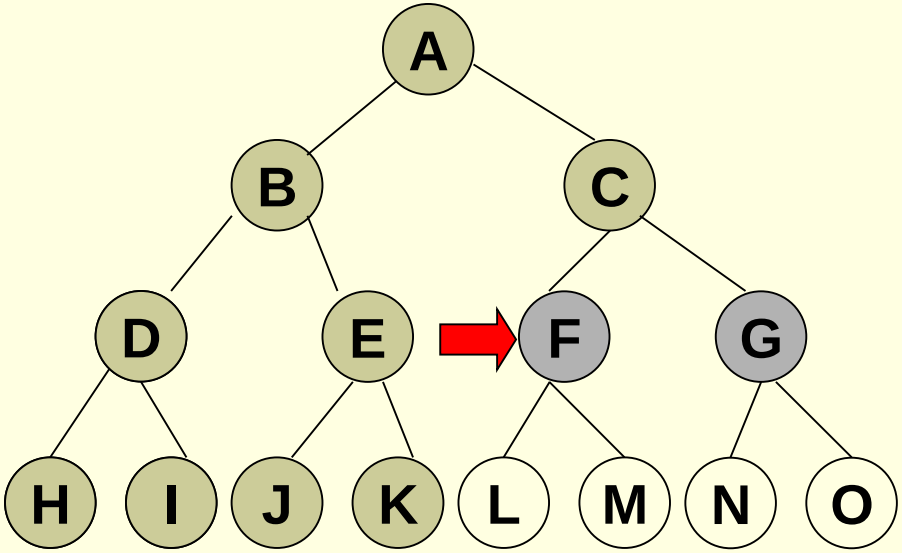
Cautarea in adancime

- 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.



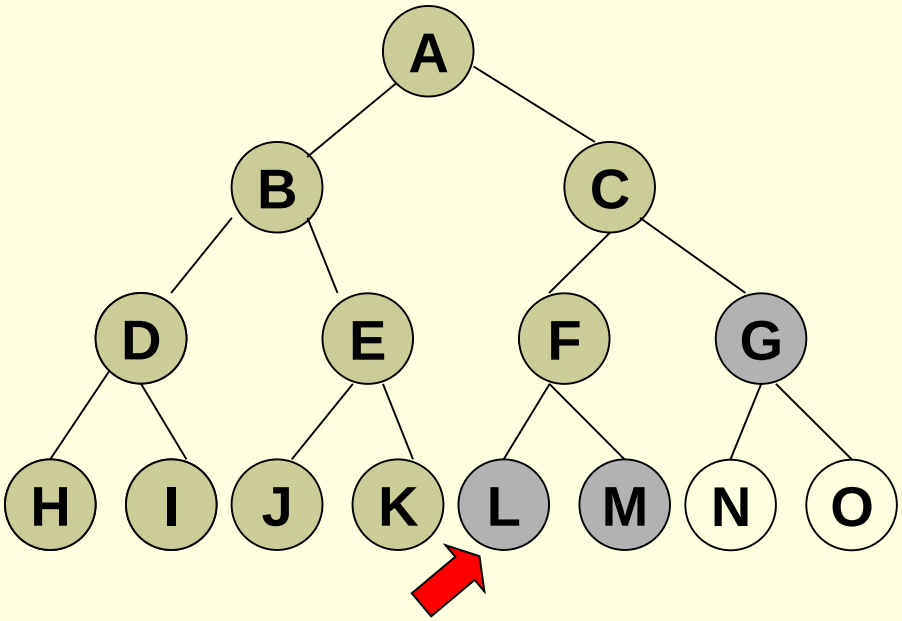
Cautarea in adancime

- 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.



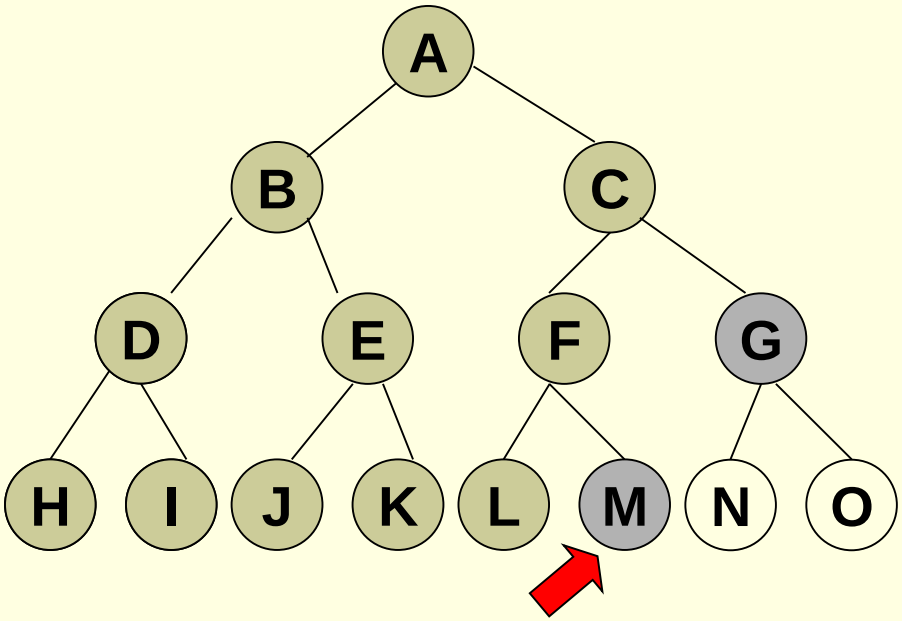
Cautarea in adancime

- 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.



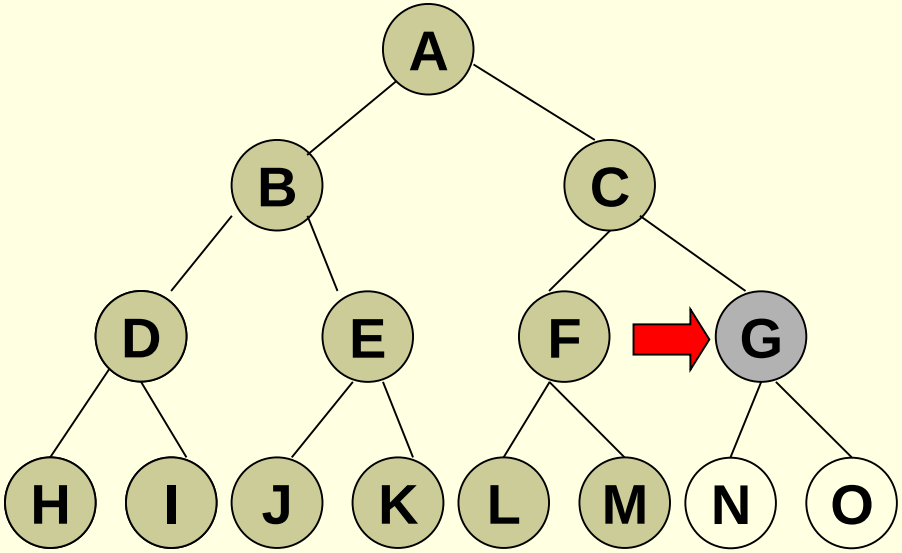
Cautarea in adancime

- 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.



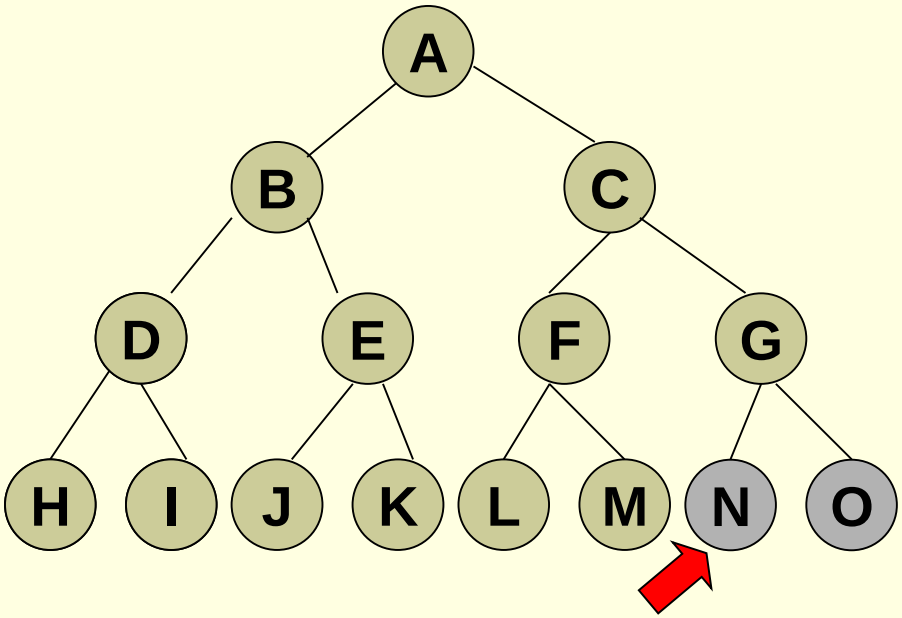
Cautarea in adancime

- 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.



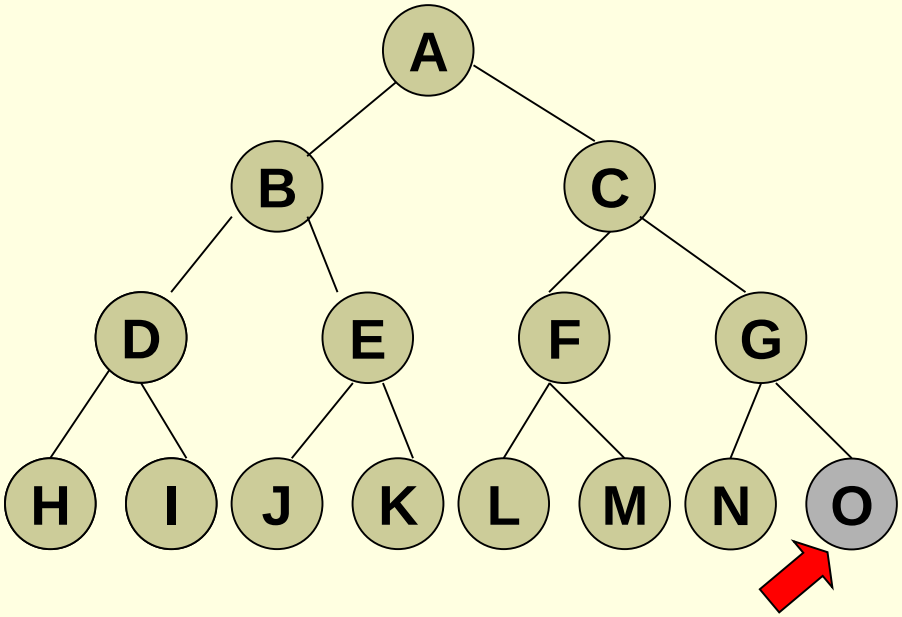
Cautarea in adancime

- 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.



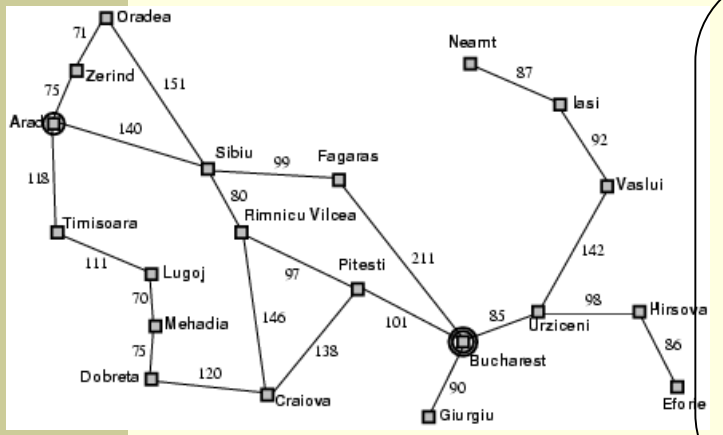
Parcurgerea in adancime: A, B, D, H, I, E, J, K, C, F, L, M, G, N, O.

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*
    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



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

Cat timp solutie negasita *si* noduri ≠ ∅ *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

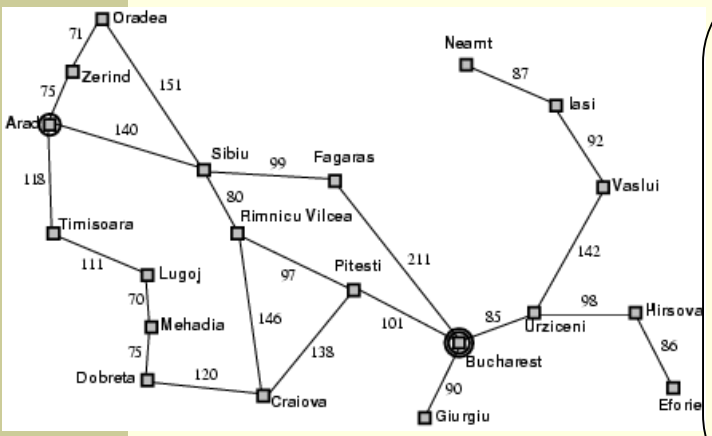
Arad

Faţa **noduri** Sfarsit

Arad

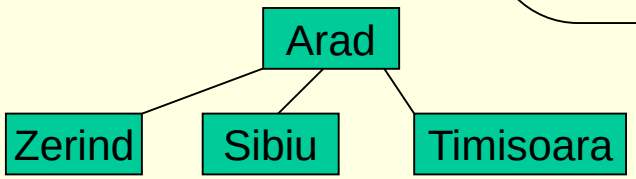
Parcurgerea: Arad,

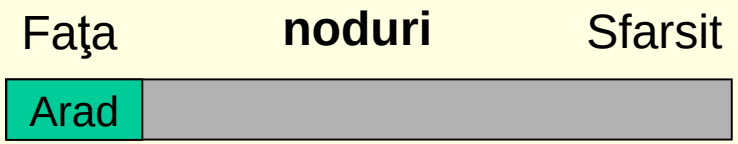
Algoritm cautare in adancime



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

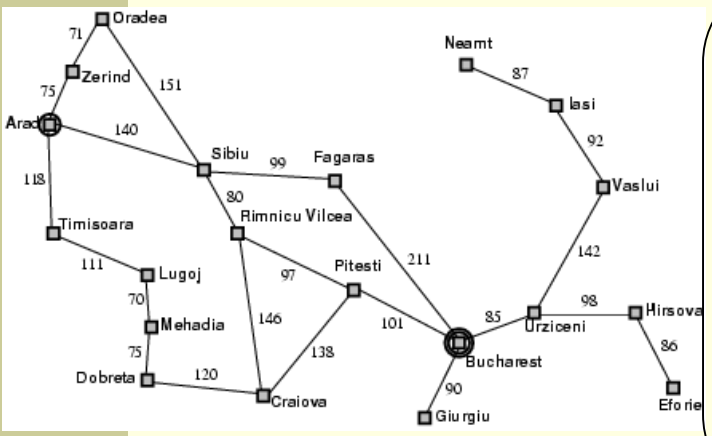
Cat timp solutie negasita *si* noduri ≠ ∅ *executa*
nod = scoate_din_fata(noduri)
Daca testare_tinta[problema] se aplica la stare(nod) *atunci*
solutie gasita
Altfel
noduri = adauga(noduri, expandare(nod, adauga_la_inceput))
Sfarsit cat timp





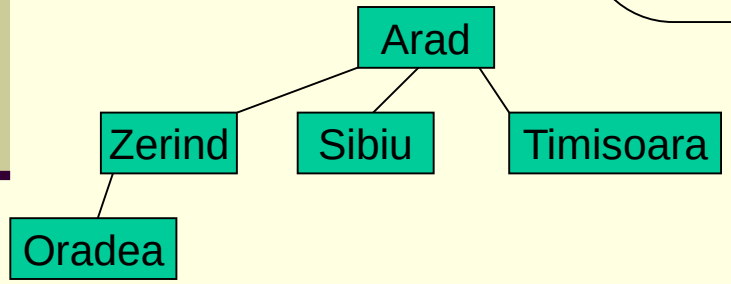
Parcurgerea: Arad,

Algoritm cautare in adancime



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

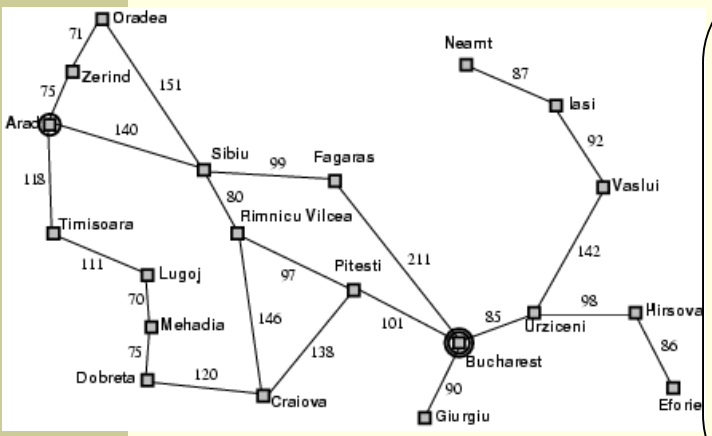
Cat timp solutie negasita *si* noduri ≠ ∅ *executa*
nod = scoate_din_fata(noduri)
Daca testare_tinta[problema] se aplica la stare(nod) *atunci*
solutie gasita
Altfel
noduri = adauga(noduri, expandare(nod, adauga_la_inceput))
Sfarsit cat timp



Faţa **noduri** Sfarsit

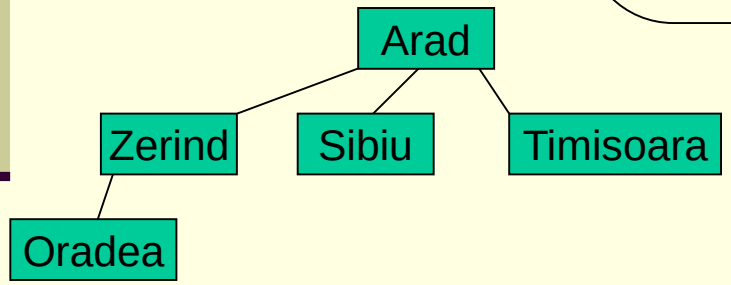
Parcurgerea: Arad, Zerind,

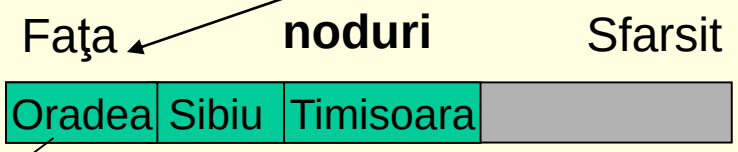
Algoritm cautare in adancime



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

Cat timp solutie negasita *si* noduri ≠ ∅ *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

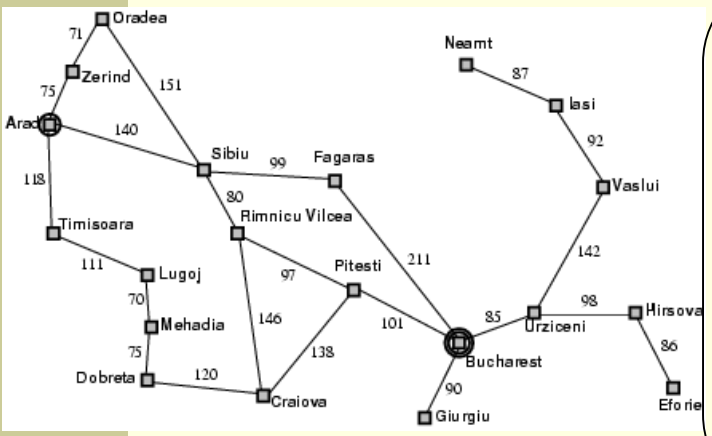




Parcurgerea: Arad, Zerind,

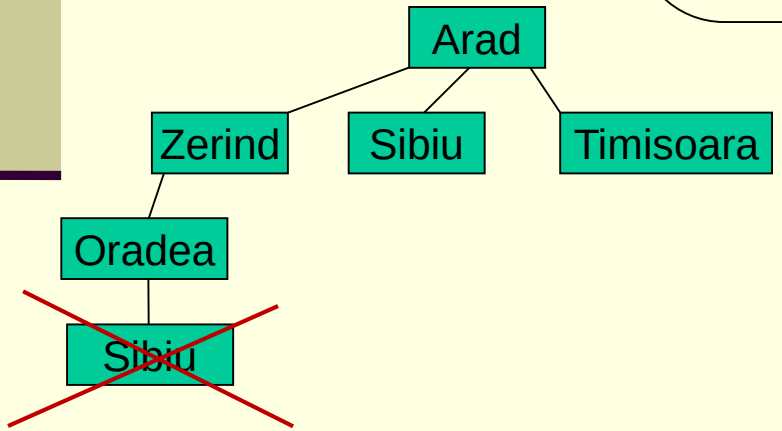
Adaugarea se face prin faţă!

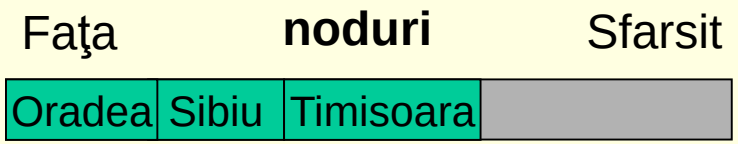
Algoritm cautare in adancime



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

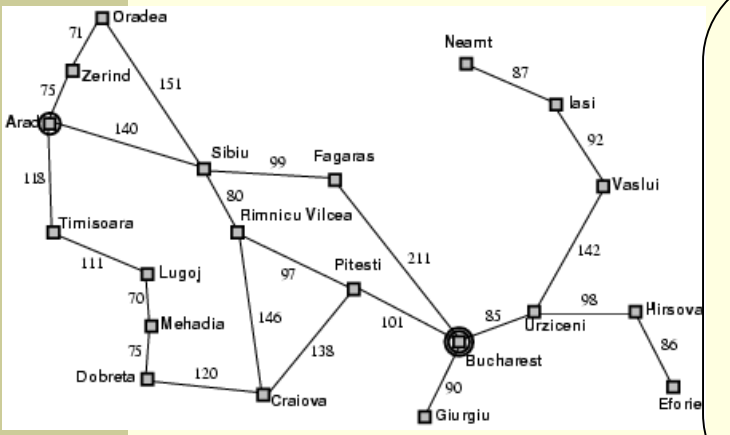
Cat timp solutie negasita *si* noduri ≠ ∅ *executa*
nod = scoate_din_fata(noduri)
Daca testare_tinta[problema] se aplica la stare(nod) *atunci*
solutie gasita
Altfel
noduri = adauga(noduri, expandare(nod, adauga_la_inceput))
Sfarsit cat timp





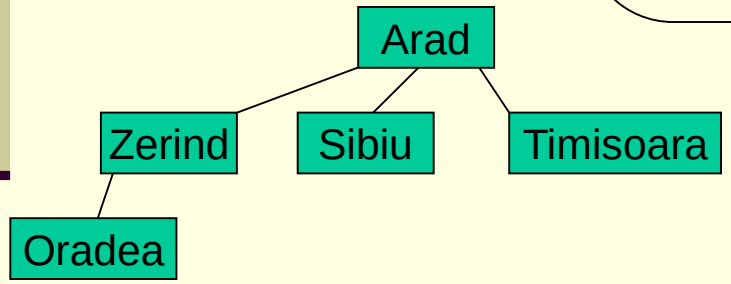
Parcurgerea: Arad, Zerind, Oradea

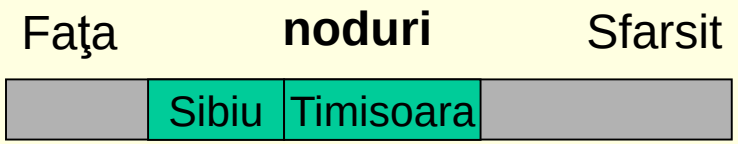
Algoritm cautare in adancime



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

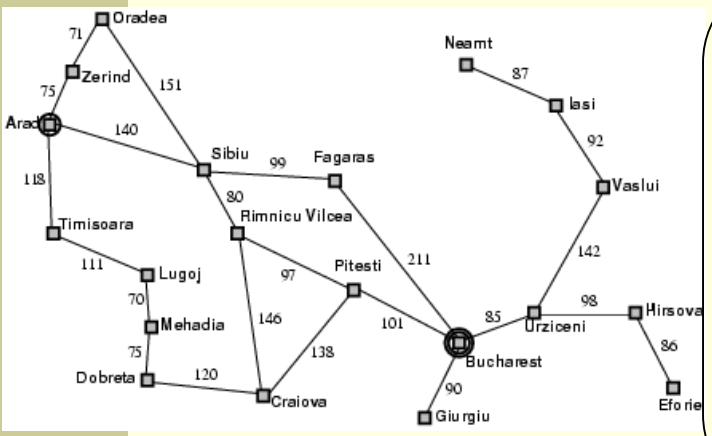
Cat timp solutie negasita *si* noduri ≠ ∅ *executa*
nod = scoate_din_fata(noduri)
Daca testare_tinta[problema] se aplica la stare(nod) *atunci*
solutie gasita
Altfel
noduri = adauga(noduri, expandare(nod, adauga_la_inceput))
Sfarsit cat timp





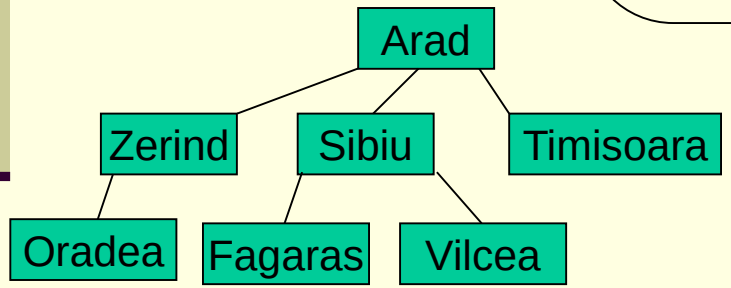
Parcurgerea: Arad, Zerind, Oradea, Sibiu

Algoritm cautare in adancime



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

Cat timp solutie negasita *si* noduri ≠ ∅ *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

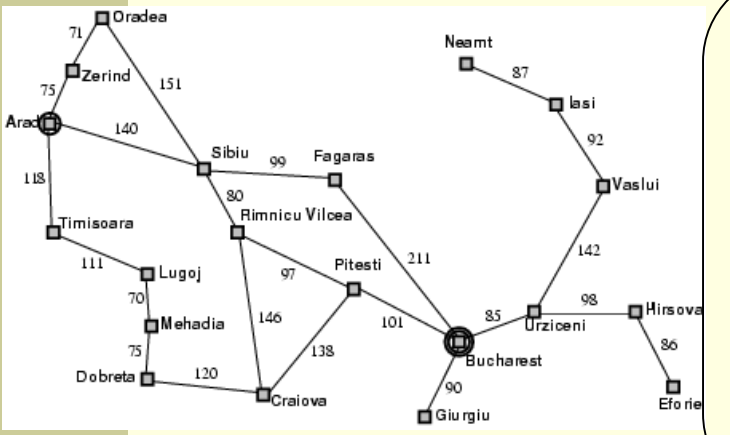


Faţa **noduri** Sfarsit

Fagaras	Vilcea	Timisoara	

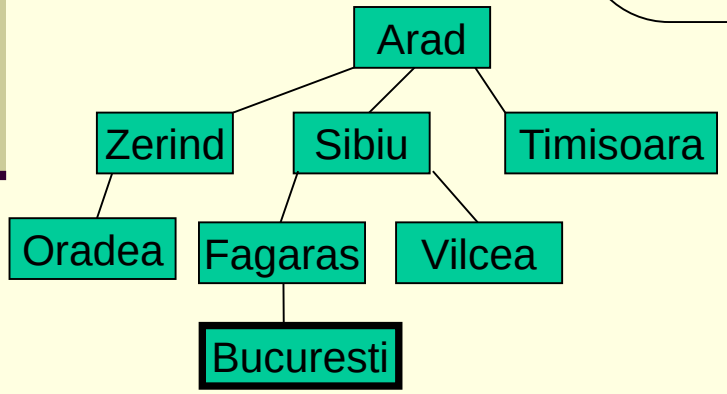
Parcurgerea: Arad, Zerind, Oradea, Sibiu

Algoritm cautare in adancime



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

Cat timp solutie negasita *si* noduri ≠ ∅ *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

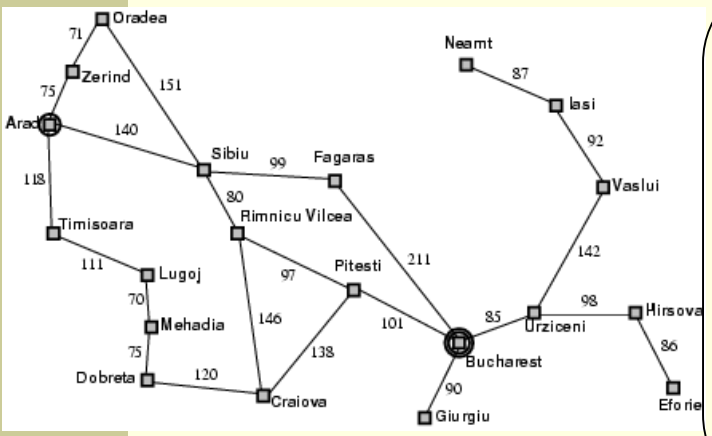


Faţa **noduri** Sfarsit

Fagaras	Vilcea	Timisoara	

Parcurgerea: Arad, Zerind, Oradea, Sibiu, Fagaras

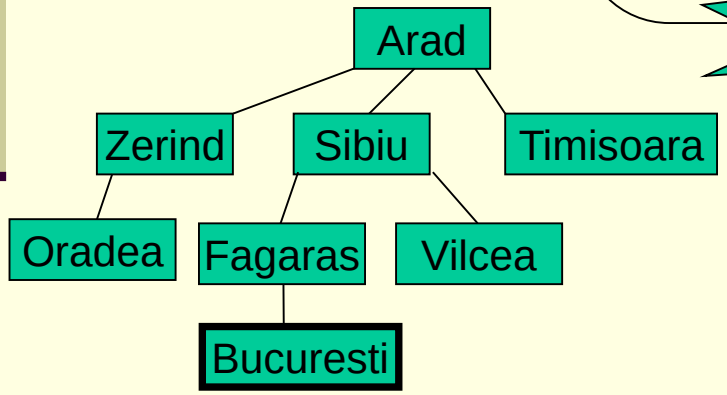
Algoritm cautare in adancime



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

Cat timp solutie negasita *si* noduri ≠ ∅ *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 daca
Sfarsit cat timp

Nod tinta!

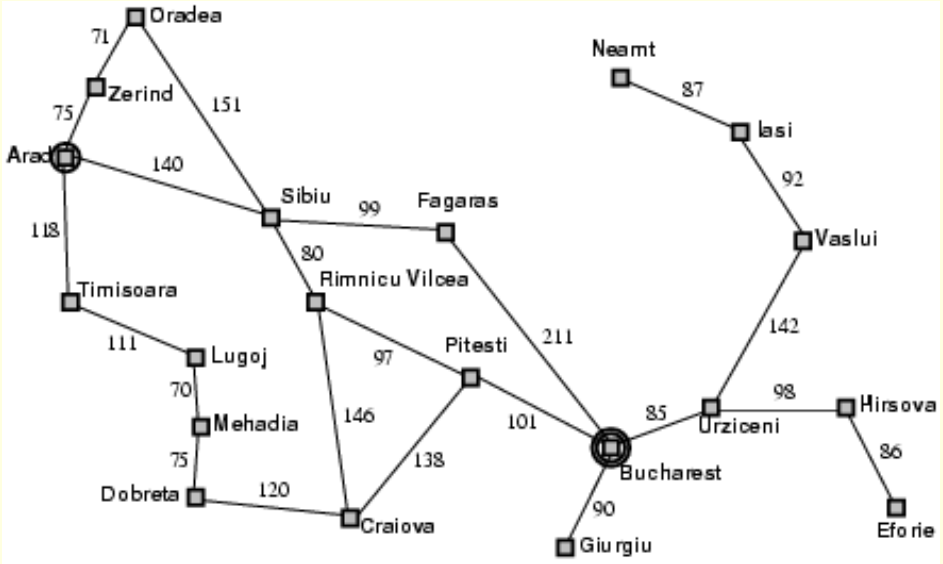


Faţa **noduri** Sfarsit

Bucuresti	Vilcea	Timisoara	

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

Exercitiu



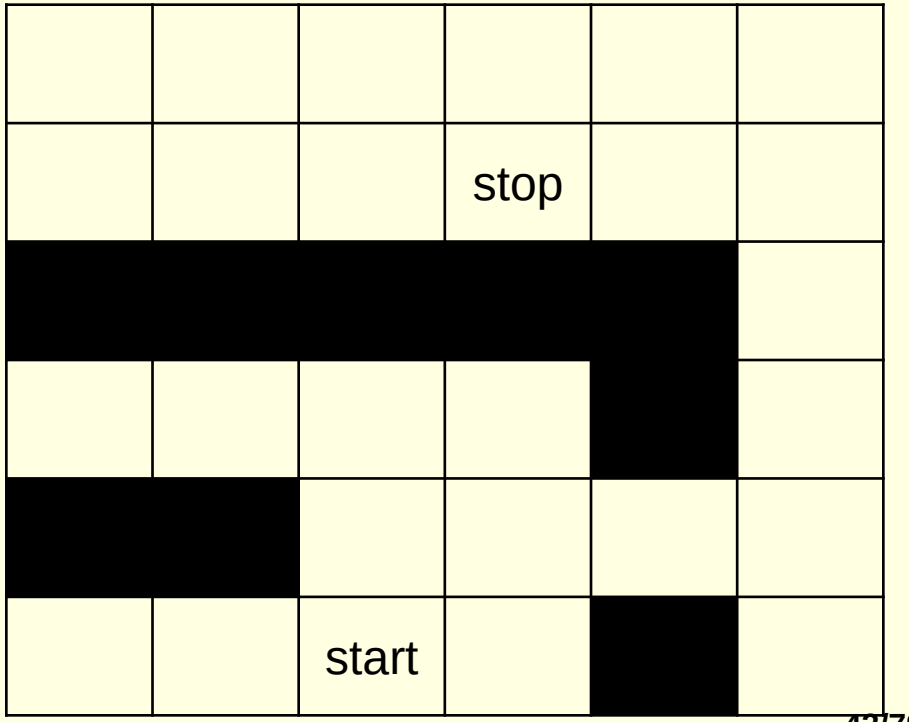
Faţa **noduri** Sfarsit

Gasiti o ruta de la Timisoara la Pitesti folosind parcurgerea in adancime. Desenati arborele, scrieti parcurgerea si continutul pentru noduri la fiecare pas. Alegeti descendentii in ordine alfabetica

Parcurgerea: Timisoara, …, Pitesti

Exercitiu

- Consideram problema gasirii unui rute in figura de mai jos de la start la stop.
- Agentul se muta un patrat la fiecare pas vertical sau orizontal.
- Nu se poate deplasa in patratele hasurate.
- Etichetati cu litere in ordine alfabetica patratele daca se utilizeaza o cautare in adancime, iar ordinea operatiilor este: sus, stanga, dreapta si jos.



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

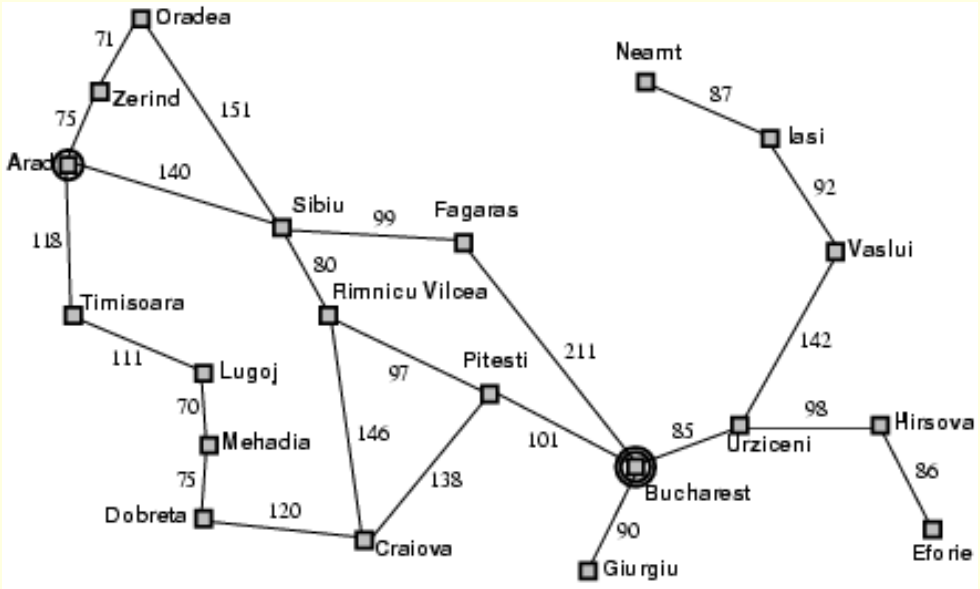
- Impune o margine superioara pentru lungimea unui drum.
- Se poate utiliza la probleme unde stim la ce adancime maxima trebuie sa gasim solutia
  - Ex: avem 20 de orase, ne aflam in orasul A, solutia trebuie sa se gaseasca la maxim 19 pasi.
- Daca *l* este limita de adancime stabilita, atunci complexitatile:
  - 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 ≠ ∅ *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

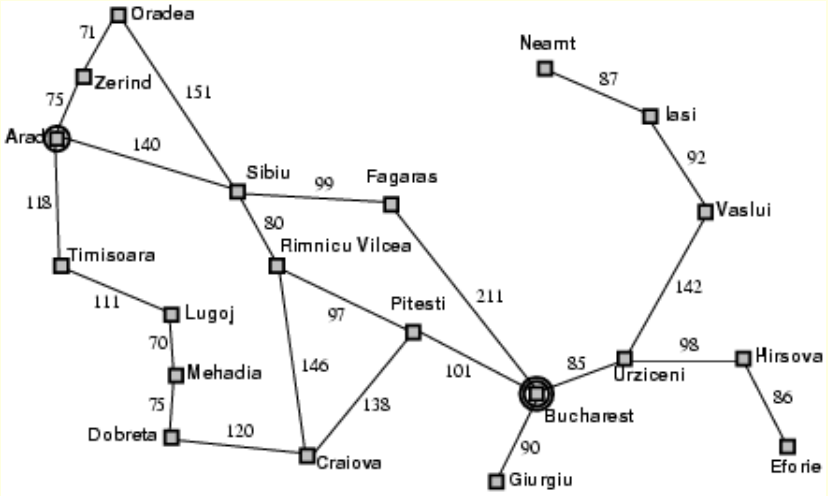
Cautarea limitata in adancime



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



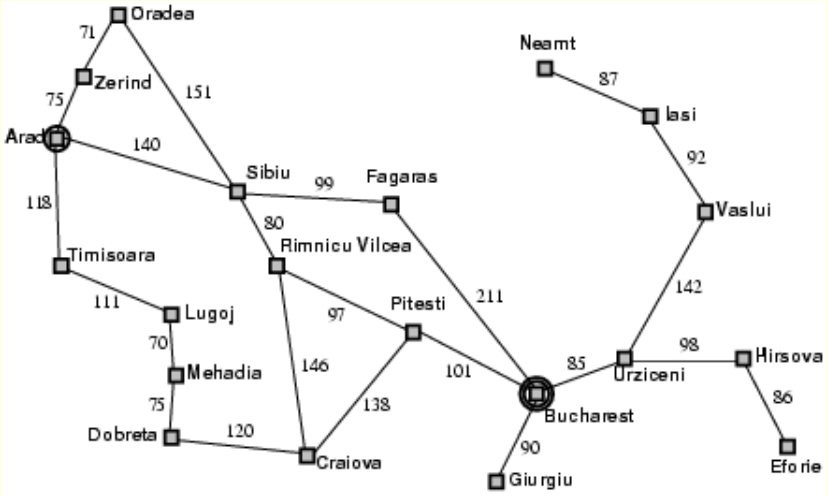
Arad

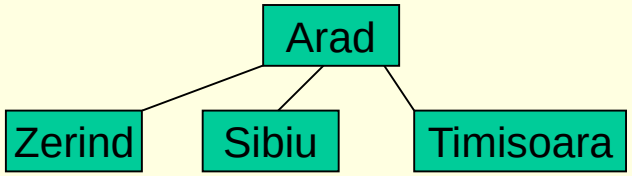
Faţa	**noduri**	Sfarsit
Arad		
0		

Parcurgerea: Arad,

Adancime: 0

Cautarea limitata in adancime





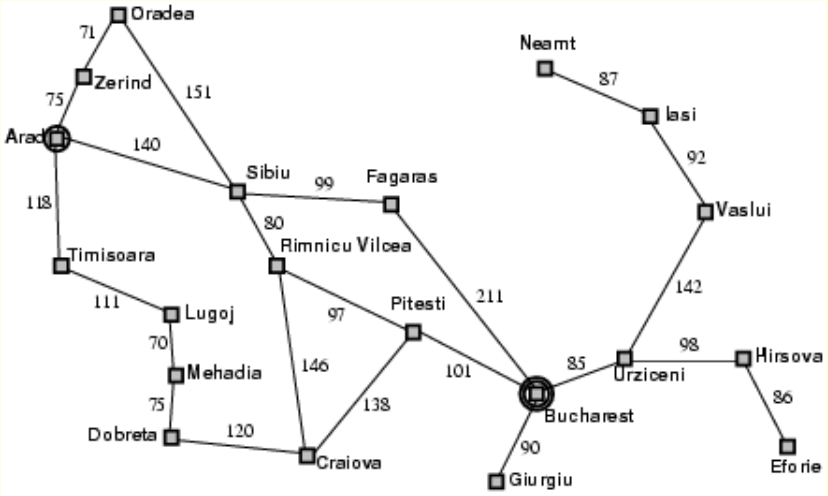
Faţa **noduri** Sfarsit

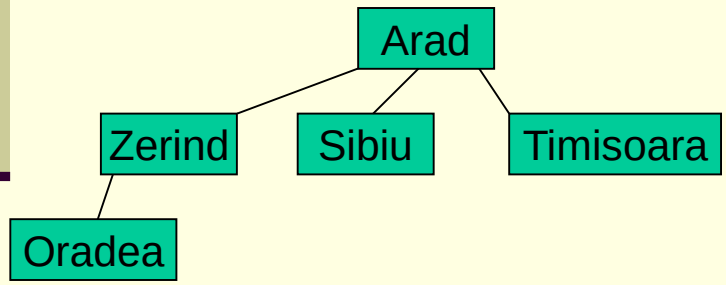
Arad	
0	

Parcurgerea: Arad,

Adancime: 0

Cautarea limitata in adancime





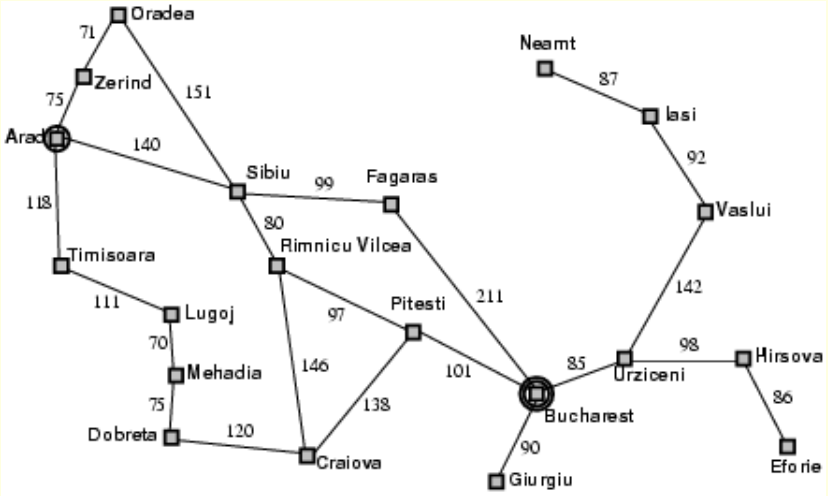
Faţa **noduri** Sfarsit

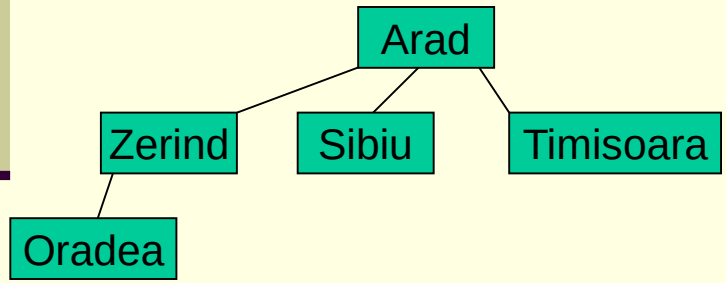
Zerind	Sibiu	Timisoara	
1	1	1	

Parcurgerea: Arad, Zerind,

Adancime: 1

Cautarea limitata in adancime





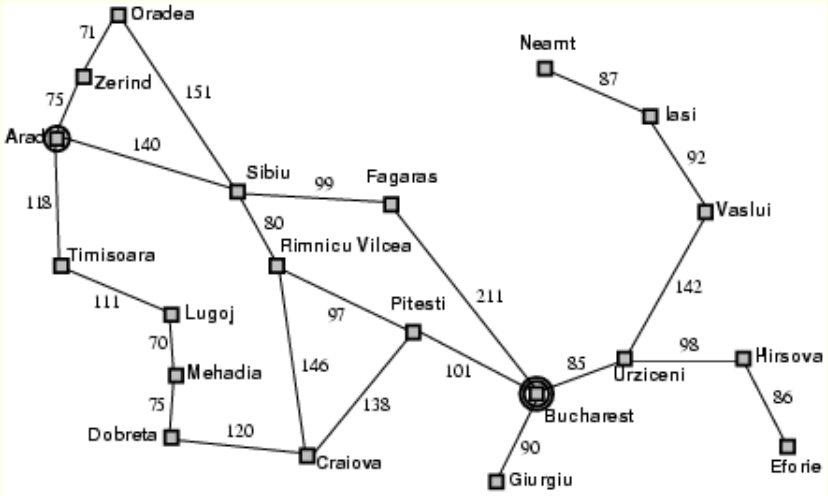
Faţa **noduri** Sfarsit

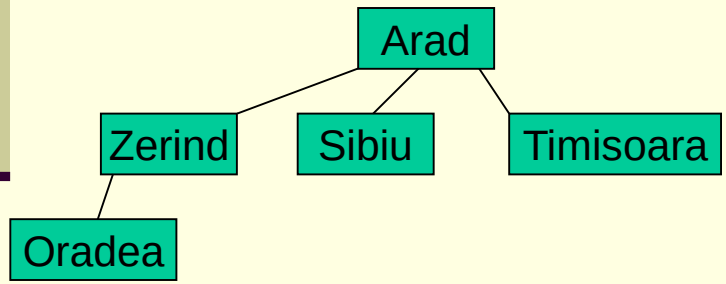
Oradea	Sibiu	Timisoara	
2	1	1	

Parcurgerea: Arad, Zerind, Oradea

Adancime: 2

Cautarea limitata in adancime





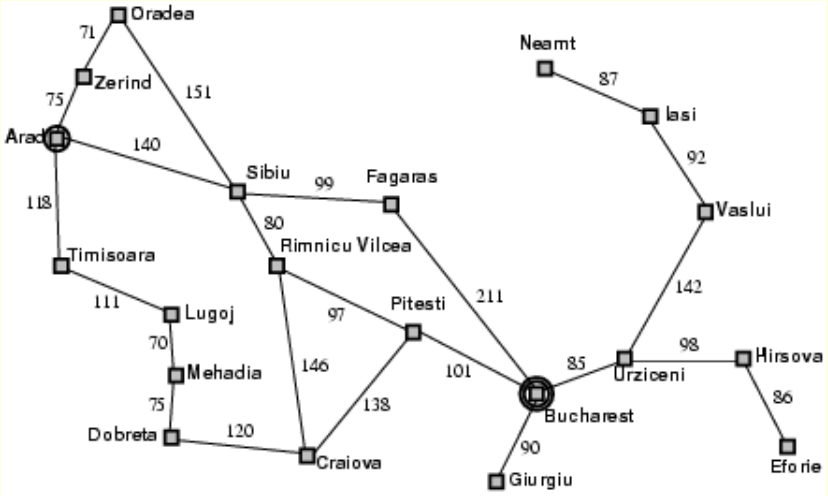
Faţa **noduri** Sfarsit

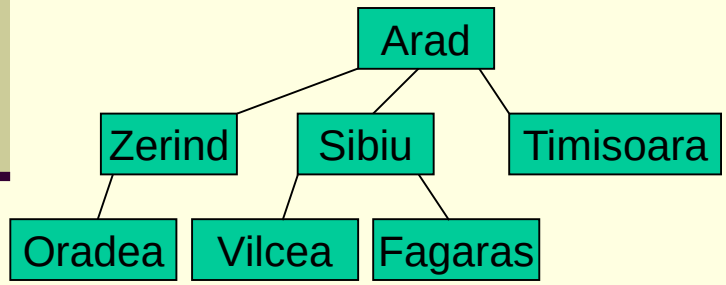
Sibiu	Timisoara	
1	1	

Parcurgerea: Arad, Zerind, Oradea, Sibiu

Adancime: 1

Cautarea limitata in adancime





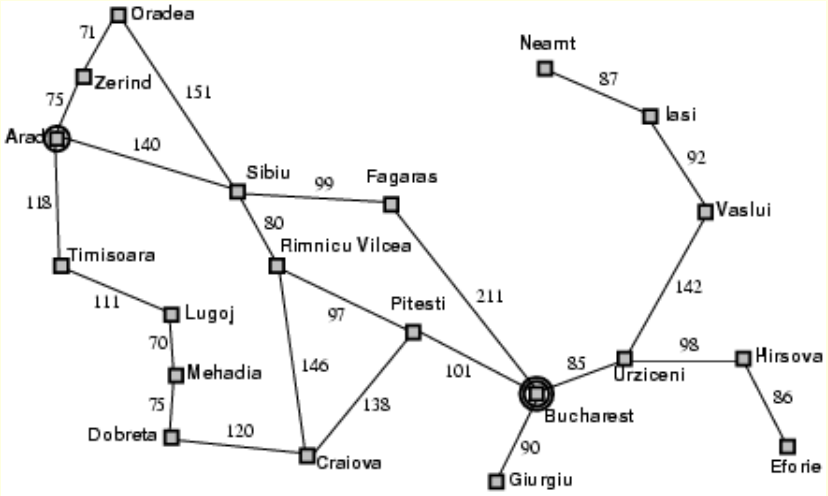
Faţa **noduri** Sfarsit

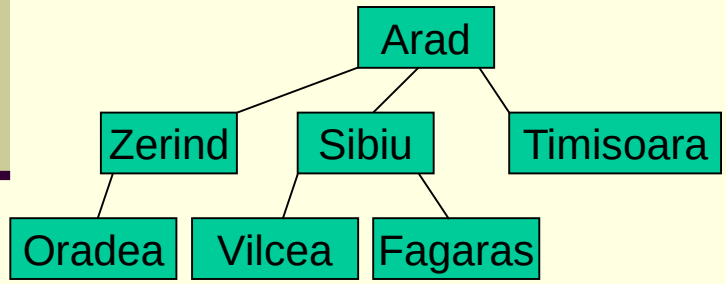
Sibiu	Timisoara	
1	1	

Parcurgerea: Arad, Zerind, Oradea, Sibiu

Adancime: 1

Cautarea limitata in adancime





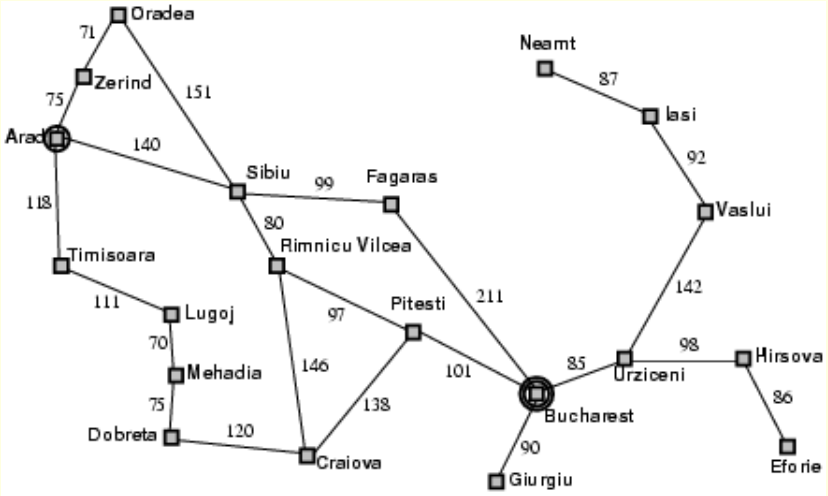
Faţa **noduri** Sfarsit

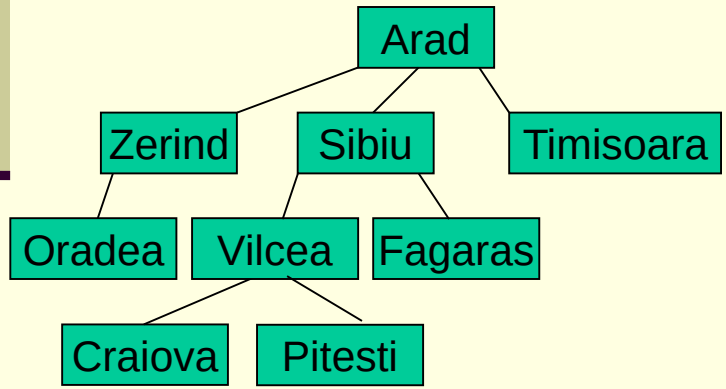
Vilcea	Fagaras	Timisoara	
2	2	1	

Parcurgerea: Arad, Zerind, Oradea, Sibiu, Vilcea

Adancime: 2

Cautarea limitata in adancime





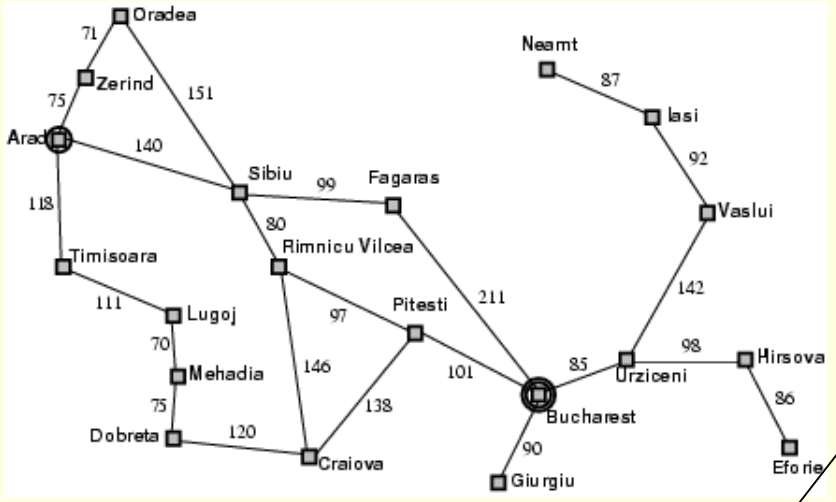
Faţa **noduri** Sfarsit

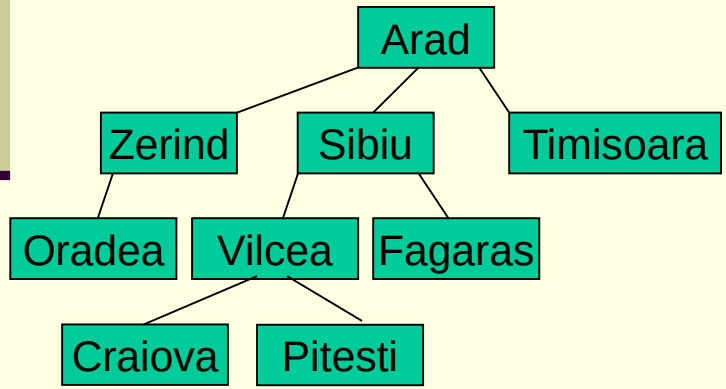
Vilcea	Fagaras	Timisoara	
2	2	1	

Parcurgerea: Arad, Zerind, Oradea, Sibiu, Vilcea

Adancime: 2

Cautarea limitata in adancime





Nu este nodul tinta!!

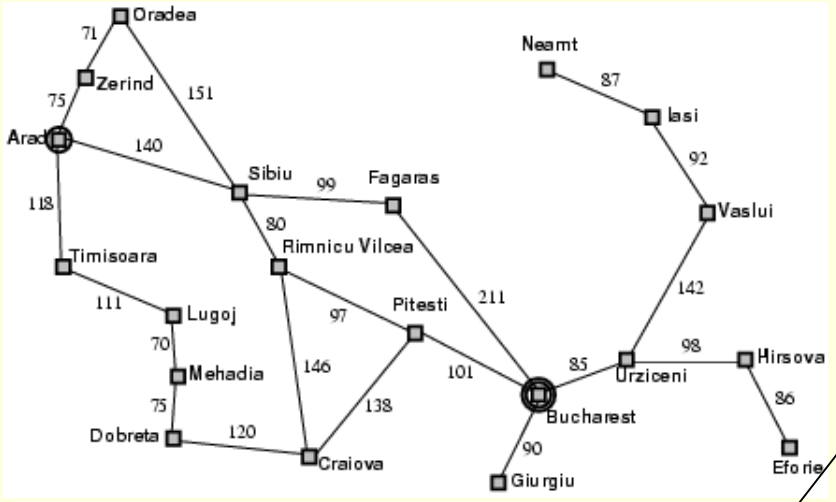
Faţa **noduri** Sfarsit

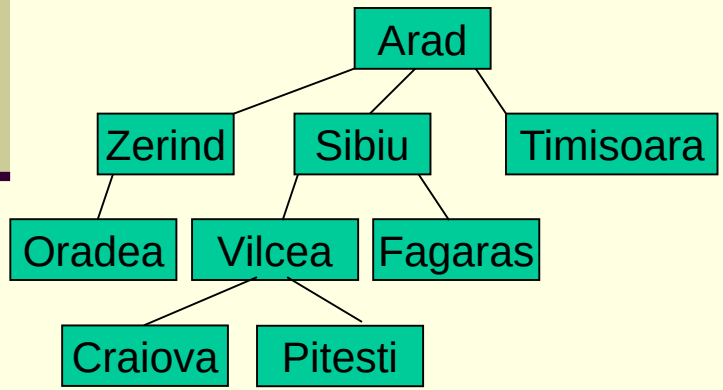
Craiova	Pitesti	Fagaras	Timisoara	
3	3	2	1	

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

Adancime: 3

Cautarea limitata in adancime





Nu este nodul tinta!!

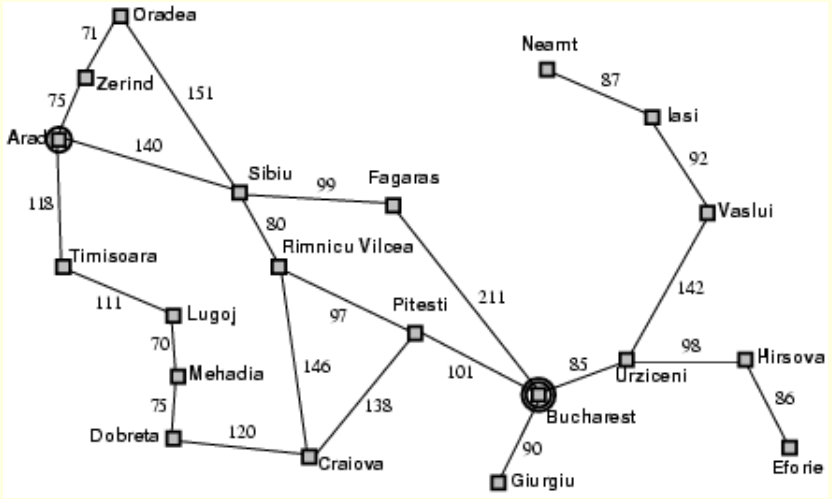
Faţa **noduri** Sfarsit

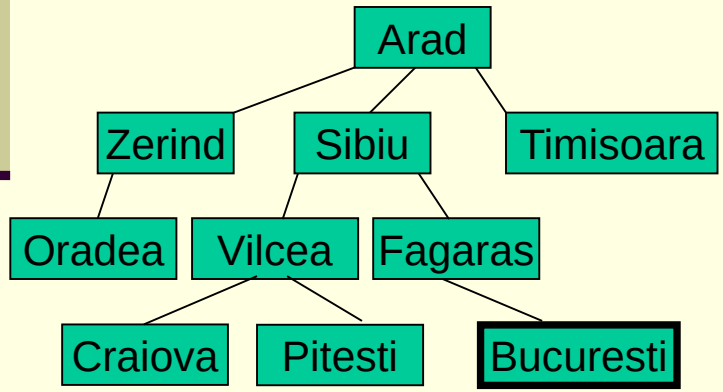
Pitesti	Fagaras	Timisoara	
3	2	1	

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

Adancime: 3

Cautarea limitata in adancime





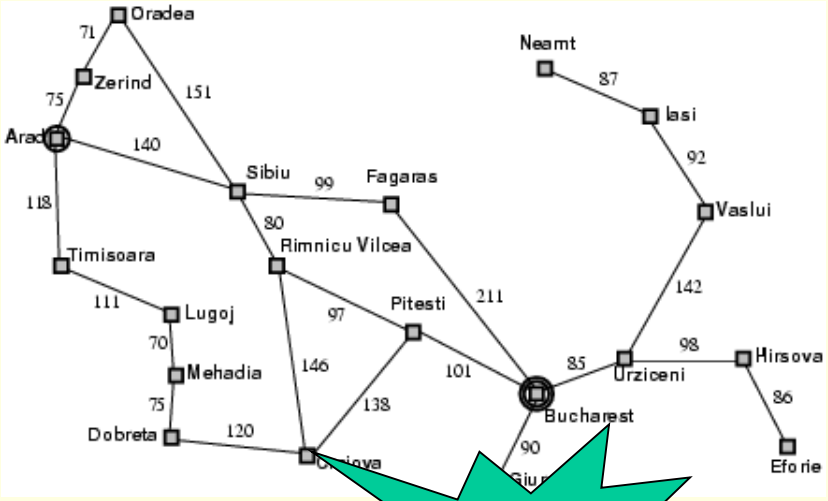
Faţa **noduri** Sfarsit

Fagaras	Timisoara	
2	1	

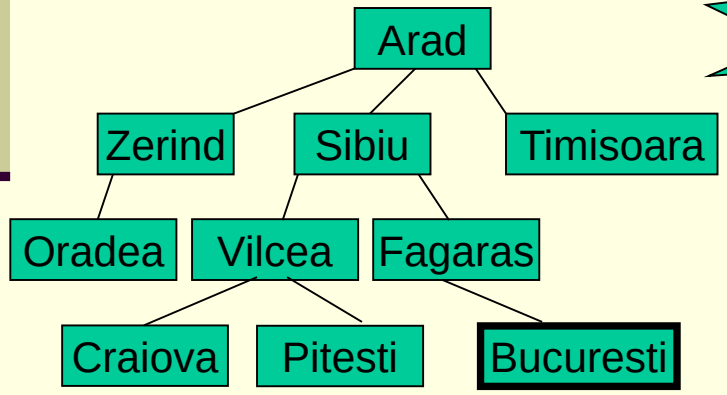
Parcurgerea: Arad, Zerind, Oradea, Sibiu, Vilcea, Craiova, Pitesti, Fagaras

Adancime: 2

Cautarea limitata in adancime



Nod tinta!



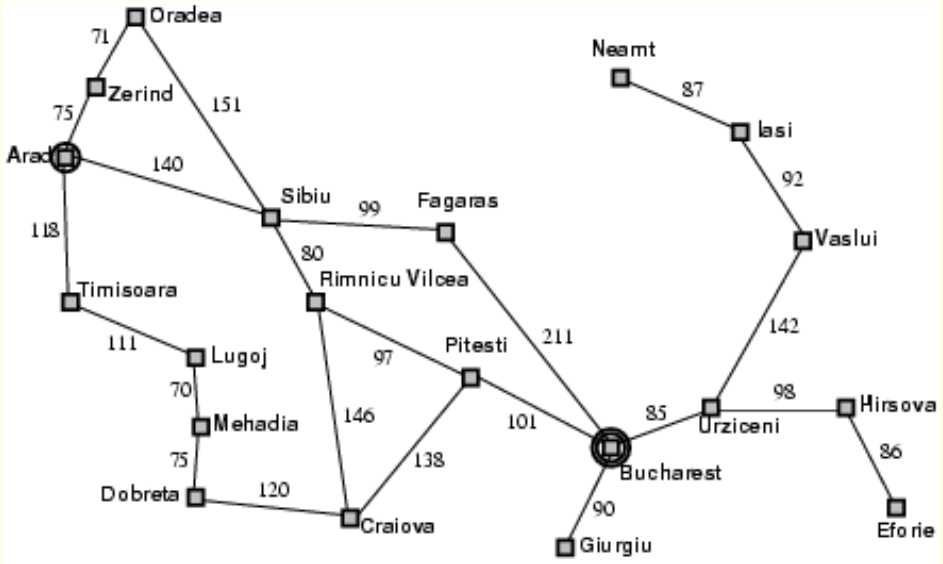
Faţa **noduri** Sfarsit

Bucuresti	Timisoara	
3	1	

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

Adancime: 3

Exercitiu



Faţa **noduri** Sfarsit

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

Parcurgerea: Bucuresti, …, Rm. Vilcea

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

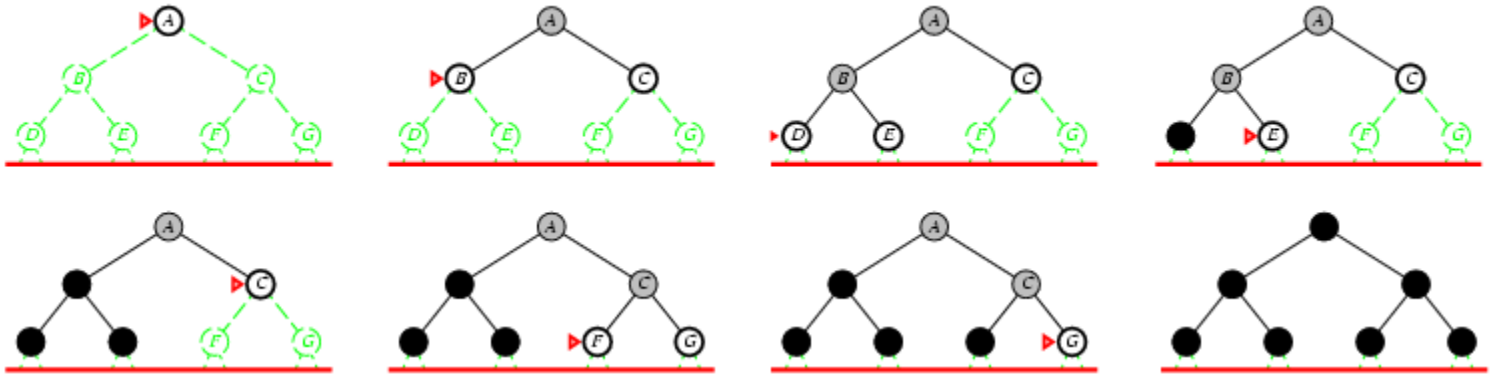
Cautarea cu adancime iterativa

Limita 0

Cautarea cu adancime iterativa

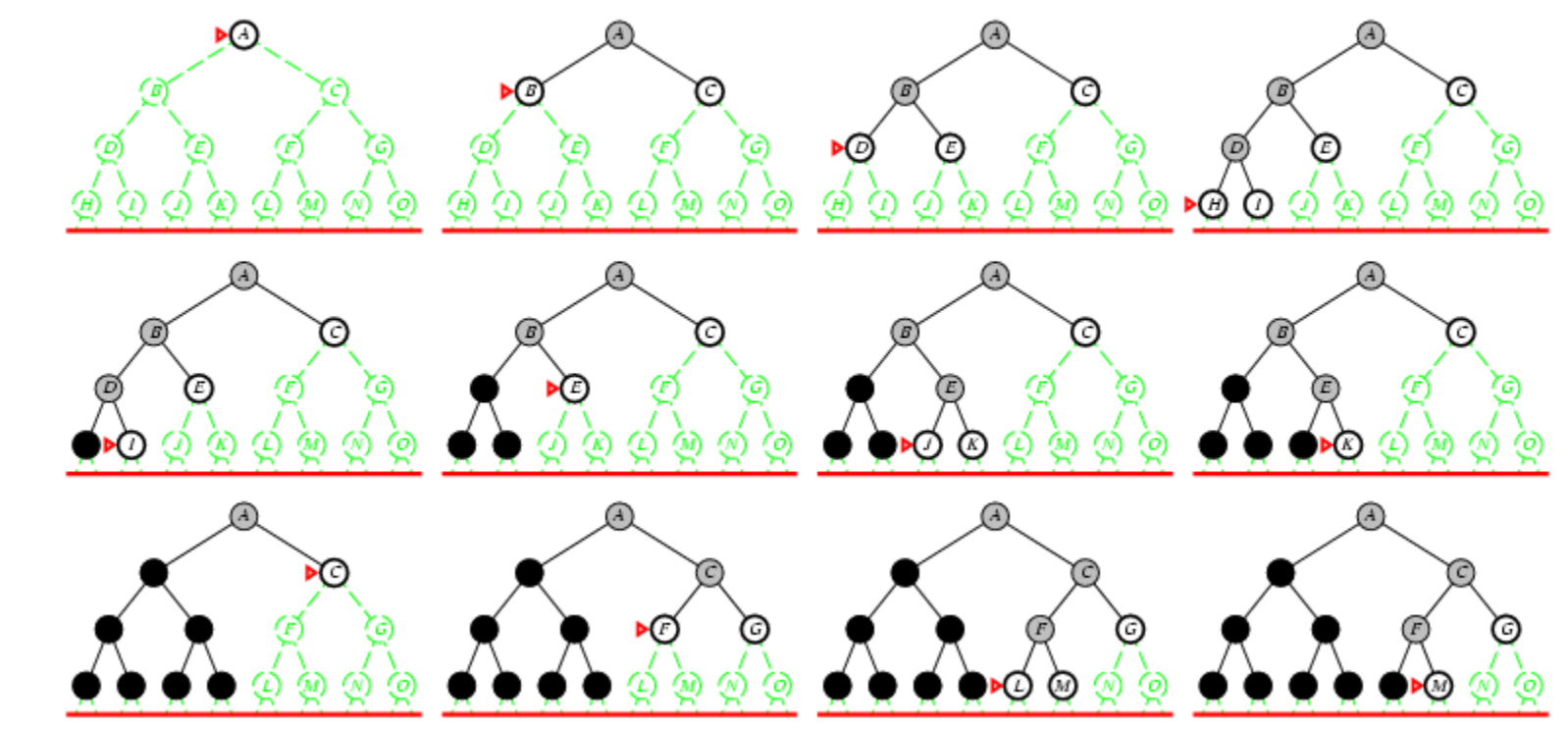
Limita 1

Cautarea cu adancime iterativa



Limita 2

Cautarea cu adancime iterativa



Limita 3

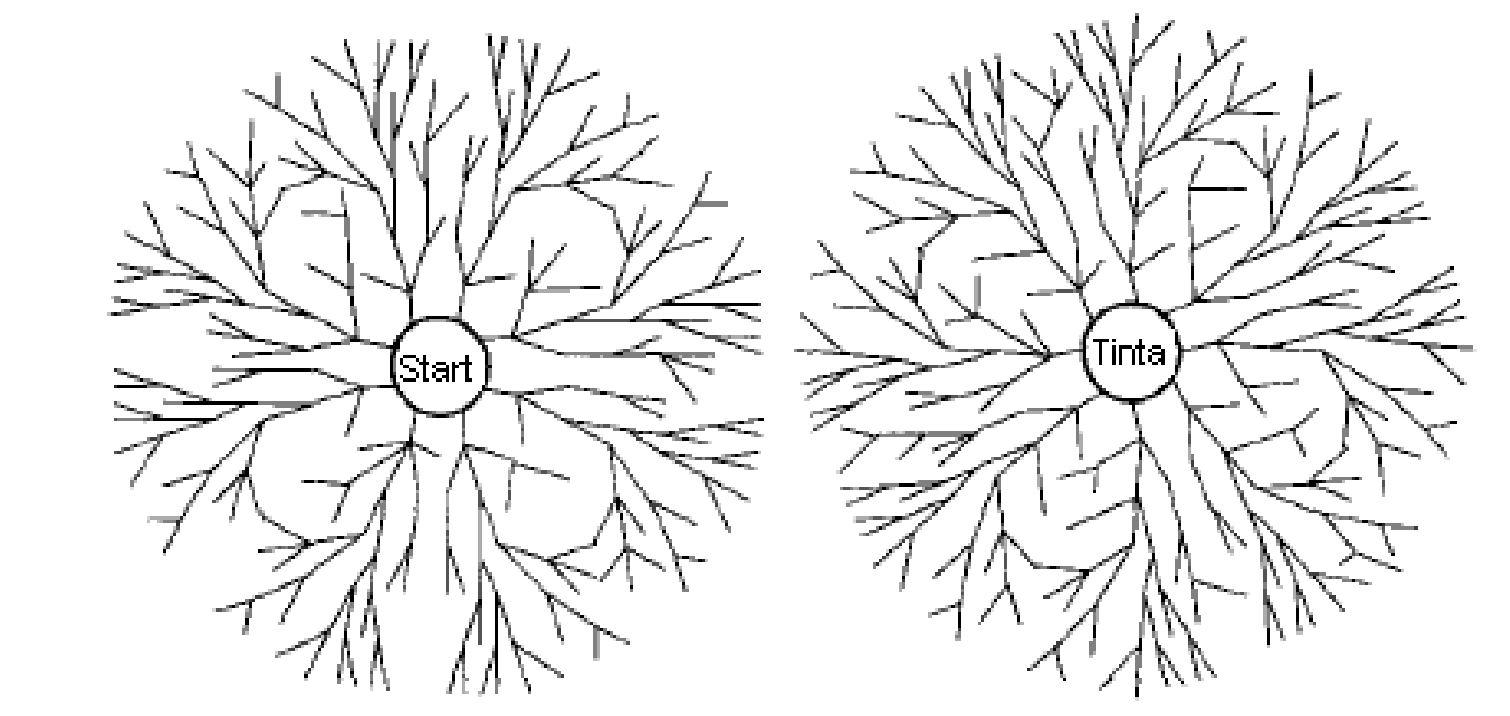
Cautarea cu adancime iterativa

- 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	Bidirectio nala
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

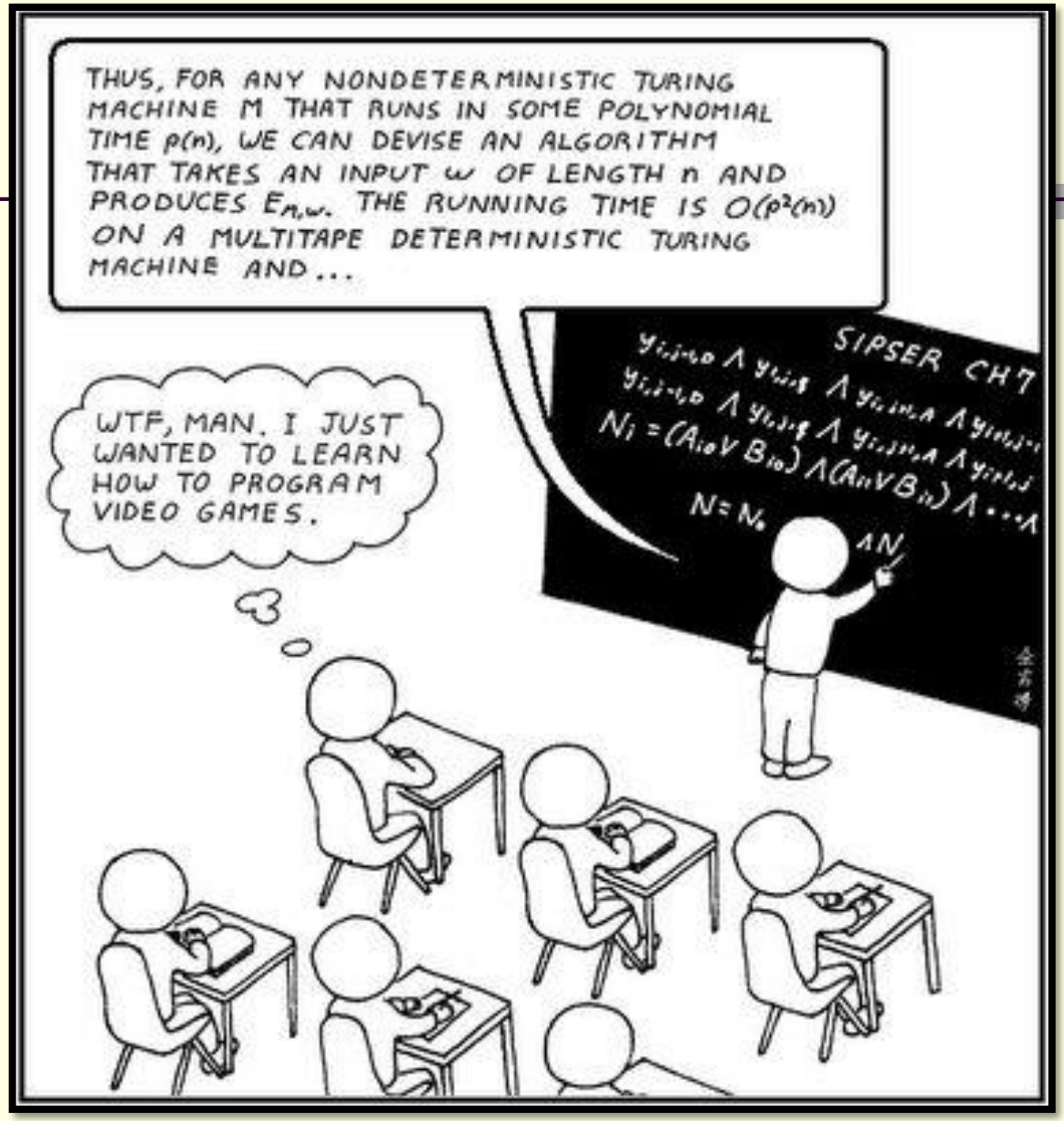
- *b* este numarul de noduri in care se expandeaza fiecare nod;
- *d* este adancimea la care se gaseste o solutie;
- *l* este limita de adancime stabilita;
- *m* este adancimea maxima din arbore.

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)


THUS, FOR ANY NONDETERMINISTIC TURING MACHINE M THAT RUNS IN SOME POLYNOMIAL TIME p(n), WE CAN DEVISE AN ALGORITHM THAT TAKES AN INPUT w OF LENGTH n AND PRODUCES $E_{M,w}$. THE RUNNING TIME IS $O(p^2(n))$ ON A MULTITAPE DETERMINISTIC TURING MACHINE AND...
SIPSER CH 7
$N_i = (A_{i0} \vee B_{i0}) \wedge (A_{i1} \vee B_{i1}) \wedge \cdots \wedge$
$N = N_0$
$\wedge N$
WTF, MAN. I JUST WANTED TO LEARN HOW TO PROGRAM VIDEO GAMES.

Tema 1/3

Doua puncte la examenul final!

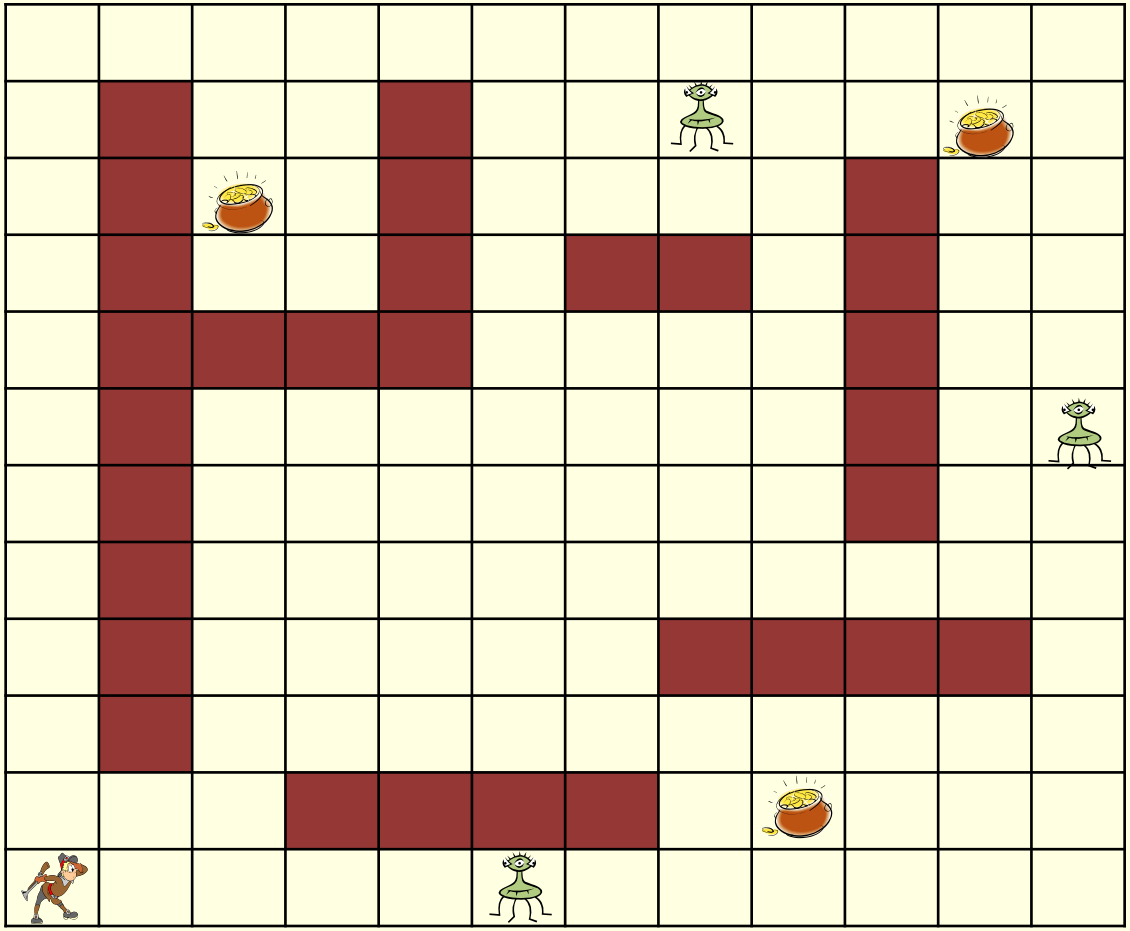
- Implementati un mediu de masura a performantei pentru o lume in care se gaseste un explorator. Lumea este descrisa astfel:
- **Perceptori**: Agentul explorator primeste un vector de 3 elemente la fiecare mutare. Primul element, un senzor de atingere, este 1 daca exploratorul s-a lovit de ceva si 0 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 0 altfel.
- **Actiuni**: mergi in fata, intoarce la dreapta cu 90º, la stanga cu 90º, 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

			
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 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 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 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.