Metode de cautare informata

Catalin Stoean

catalin.stoean@inf.ucv.ro

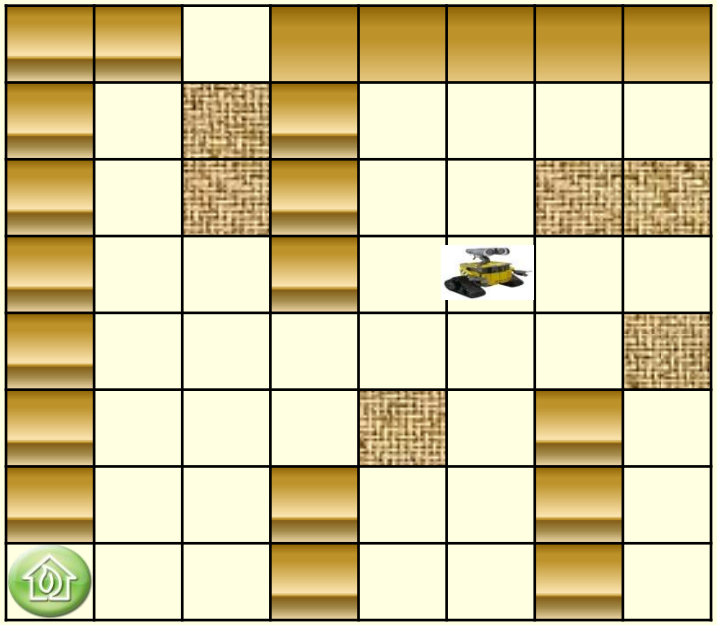
http://inf.ucv.ro/~cstoean

Cursul anterior

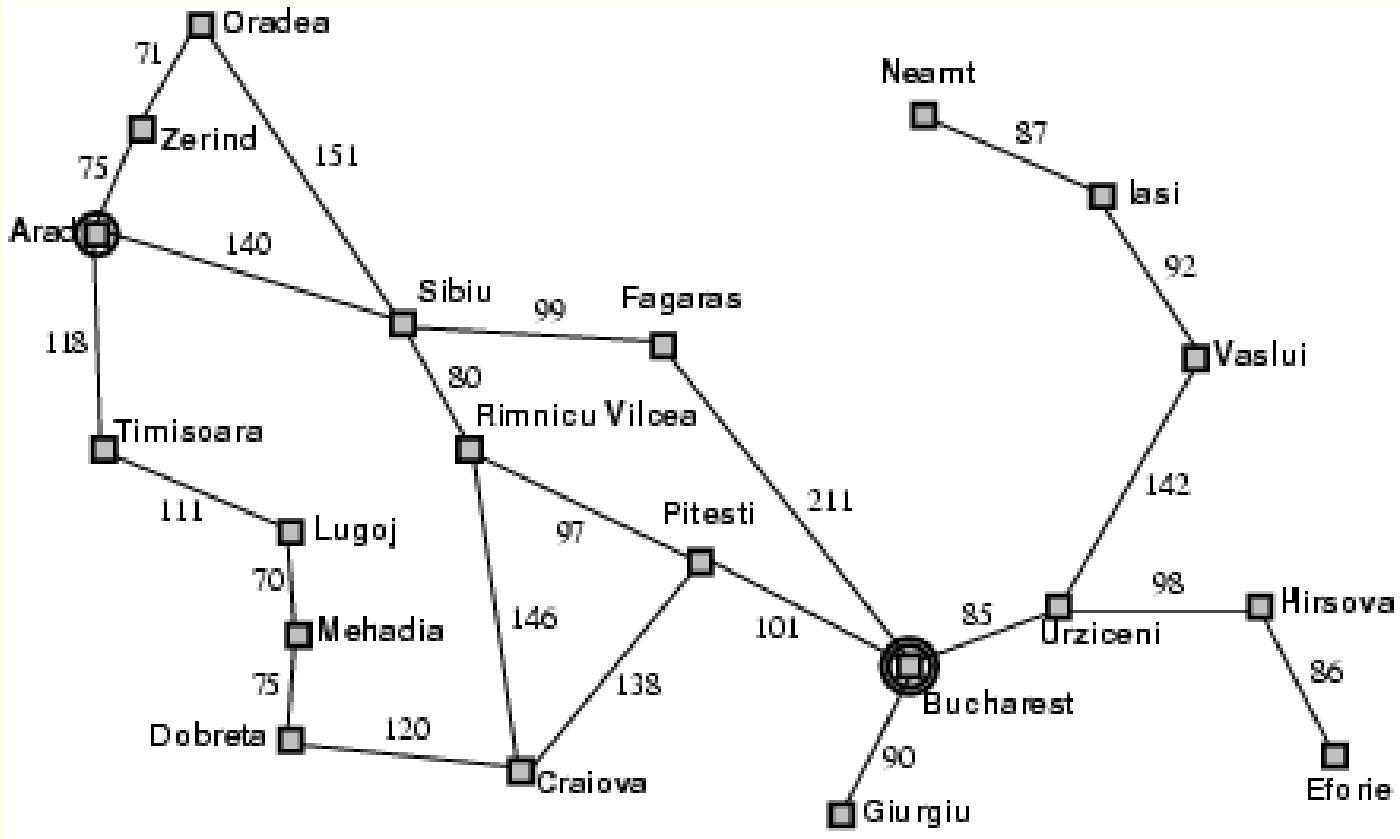
- Metode de **cautare neinformata** (blind search)
  - Solutiile la probleme sunt gasite prin generarea sistematica de stari noi si verificarea daca s-a ajuns la starea tinta (solutia problemei).
  - O strategie de cautare este data de **ordinea de expandare a nodurilor**.
  - In cele mai multe cazuri, acestea sunt ineficiente…
- In strategiile de **cautare informata**, se utilizeaza cunostinte specifice problemei pentru a gasi solutiile in mod eficient.

Cursul anterior

Apropos… tema?



Cautarea informata



Cautarea informata

- Folosind un algoritm general de cautare, cunostintele aditionale despre problema pot fi adaugate la functia care stabileste ce noduri vor fi expandate.
- Ideea este de a se utiliza o **functie de evaluare** $f(n)$ pentru fiecare nod n.
  - Ajuta la luarea de decizii in expandarea nodurilor.
  - Se ordoneaza nodurile astfel incat cel cu cea mai buna evaluare este expandat primul (strategia de cautare **intai cel mai bun**).

Strategia *intai cel mai bun*

Functia de evaluare

functia cautare_intai_best(*problema, eval*) ***intoarce*** solutie sau esec
noduri = genereaza_coada(genereaza_nod(stare_initiala[problema]))
f_coada = o functie care ordoneaza nodurile dupa *eval*

Cat timp este posibil *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, *f_coada*))
Sfarsit cat timp

Strategia *intai cel mai bun*

- Pentru a directiona cautarea, masura de evaluare trebuie sa incorporeze o masura a costului drumului de la starea curenta pana la starea tinta.

- Doua abordari:
  - Cautarea Greedy
  - Cautarea A*

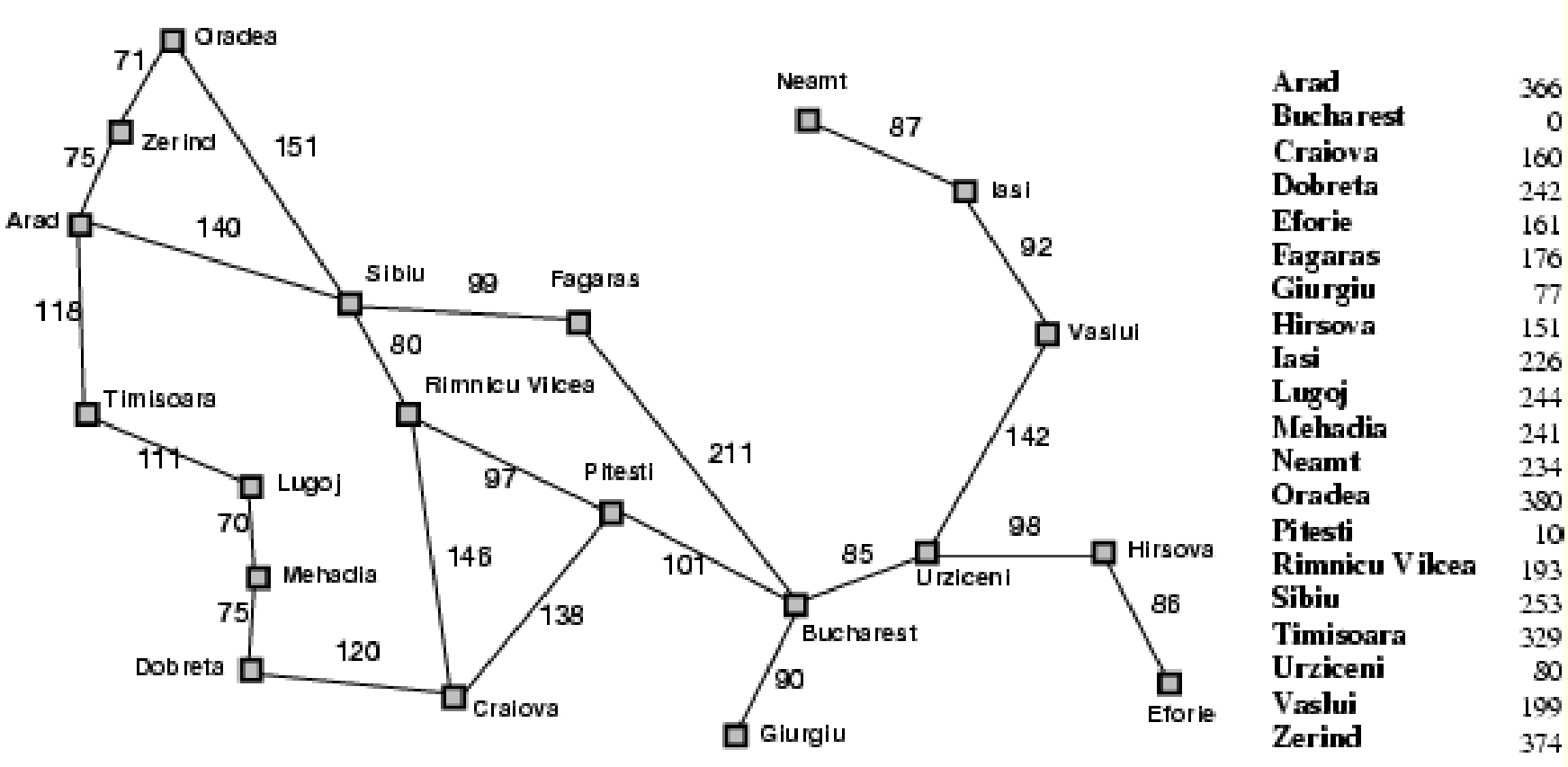
Cautarea Greedy

- Se bazeaza pe faptul ca trebuie minimizat costul de ajungere la nodul tinta.
- Concluzie: nodul care reprezinta starea care este cea mai aproape de starea tinta este intotdeauna expandat primul.
- La cele mai multe probleme, costul de a ajunge de la o stare la starea finala nu poate determinat exact, doar estimat.
- O functie care estimeaza astfel de costuri se numeste **functie euristica** si este notata de obicei cu *h*.

Cautarea Greedy

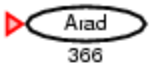
- ***h*(*n*)** = costul estimat pentru cel mai scurt drum de la nodul *n* pana la starea tinta.
  - Daca *n* este chiar nodul tinta, atunci *h*(*n*) = 0.
- O cautare *intai cel mai bun* care utilizeaza asemenea functie euristica se numeste **cautare greedy**.

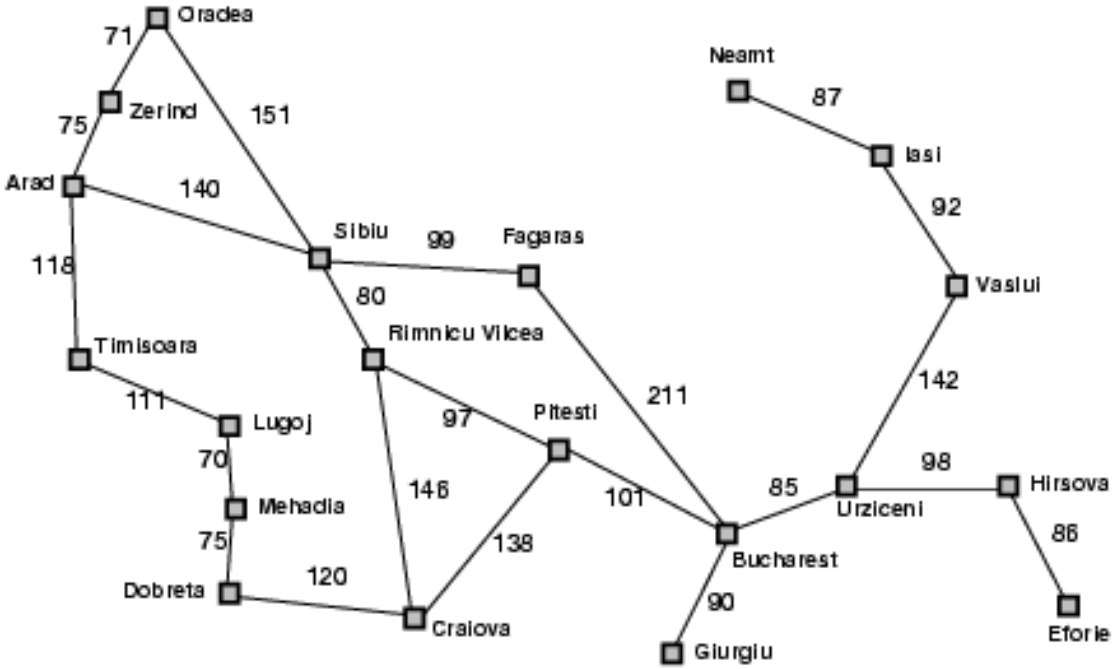
Avem distantele pana la Bucuresti



- $h(n)$ = distanta in linie dreapta de la orasul n pana la Bucuresti.

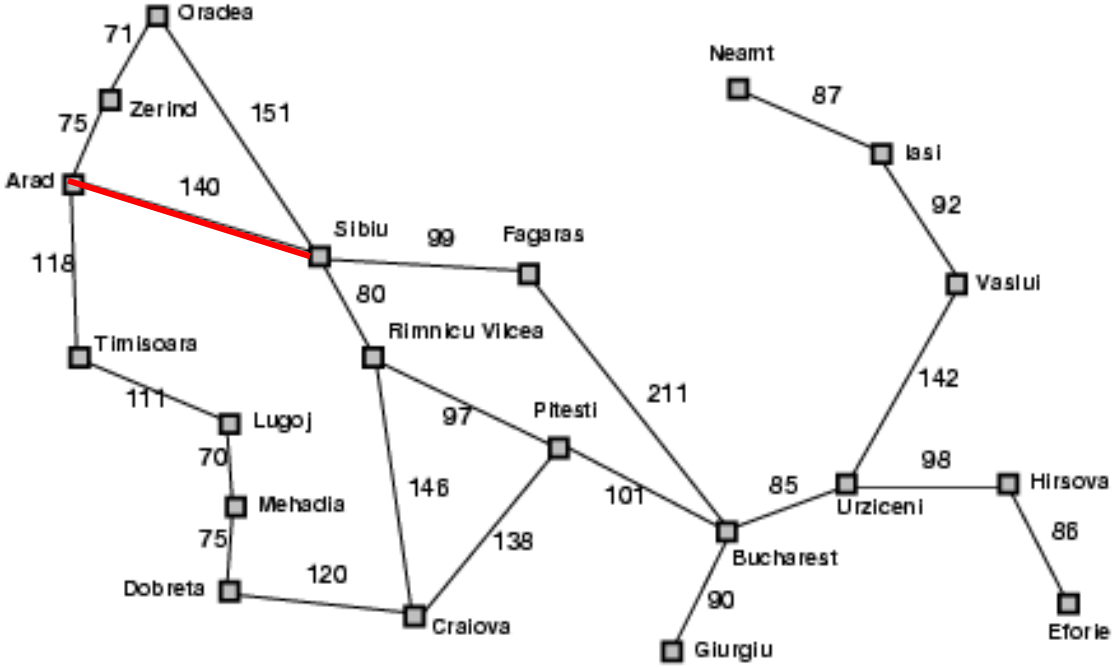
Cautarea Greedy





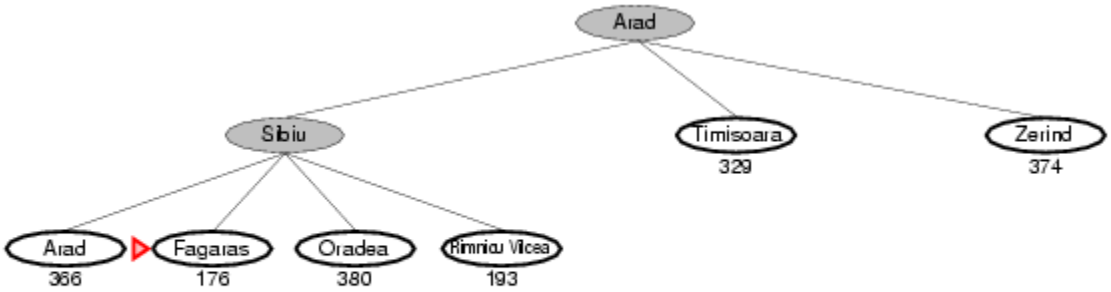
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

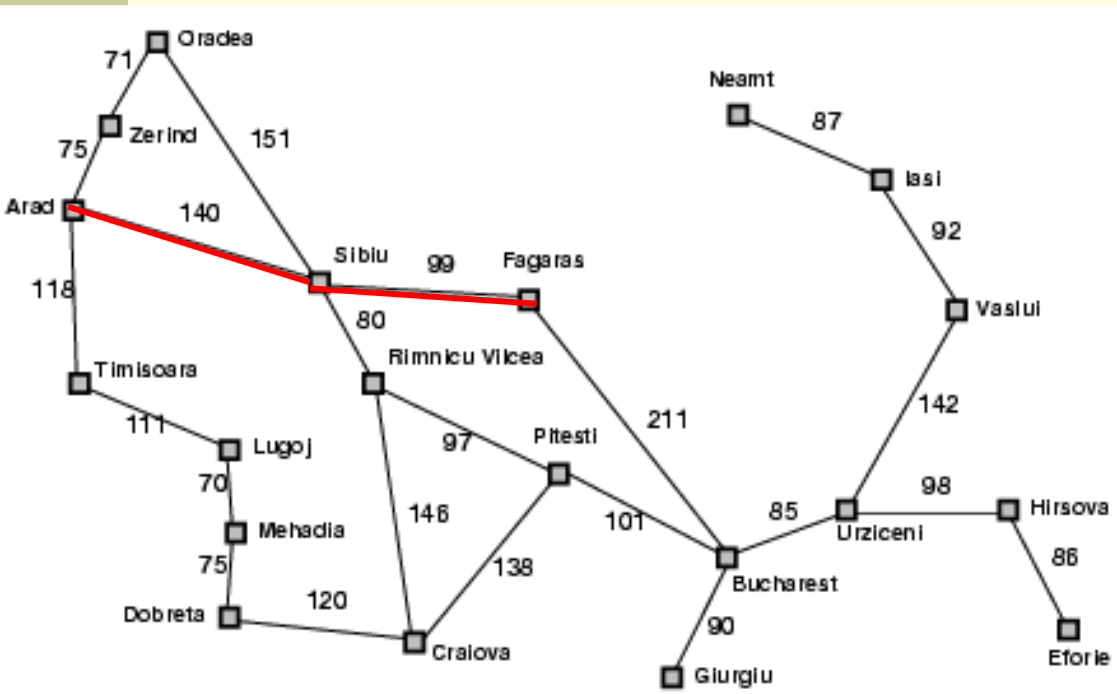
Cautarea Greedy



Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

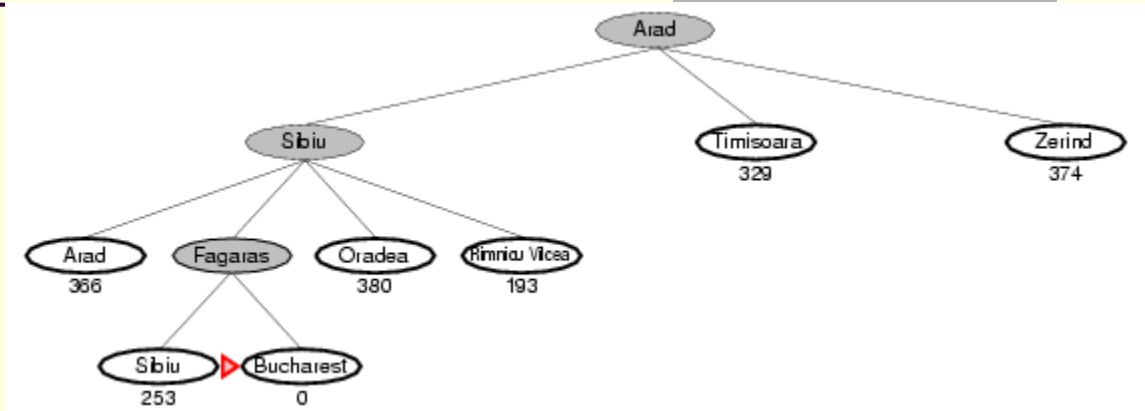
Cautarea Greedy

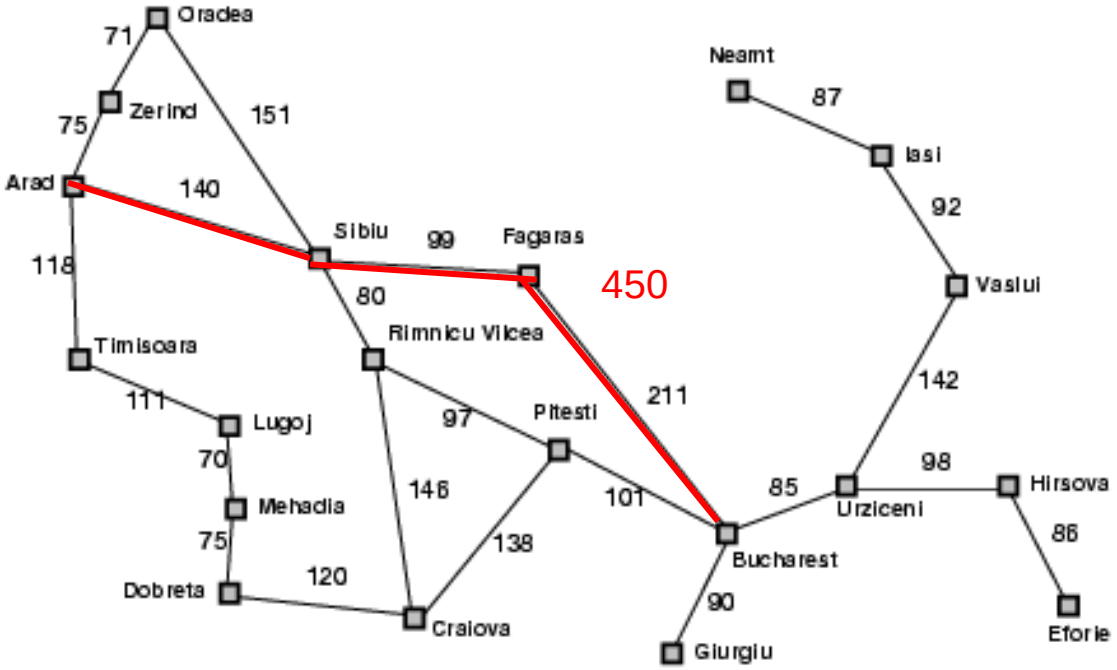




Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Cautarea Greedy

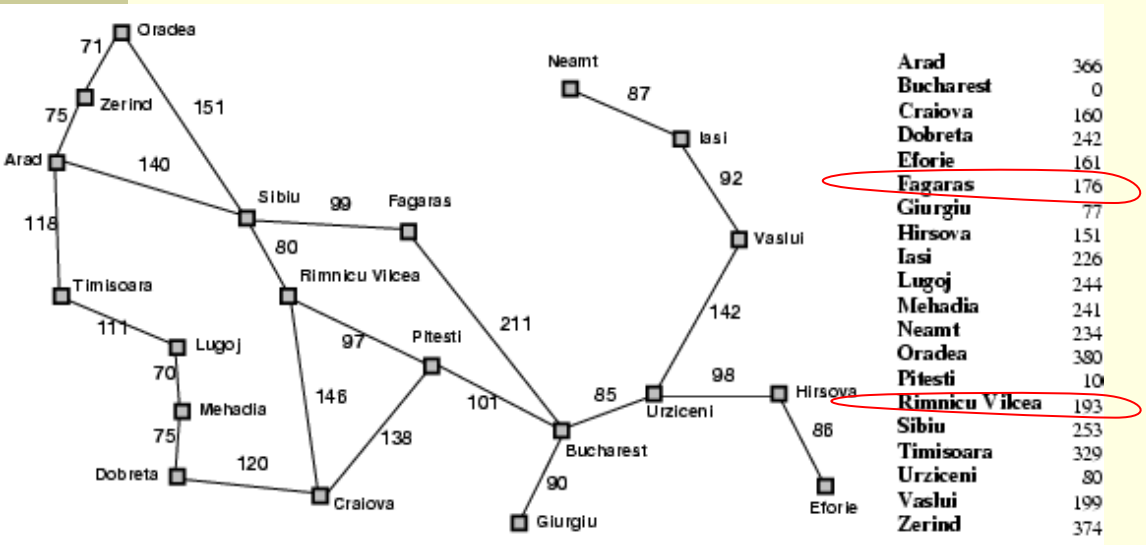




Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

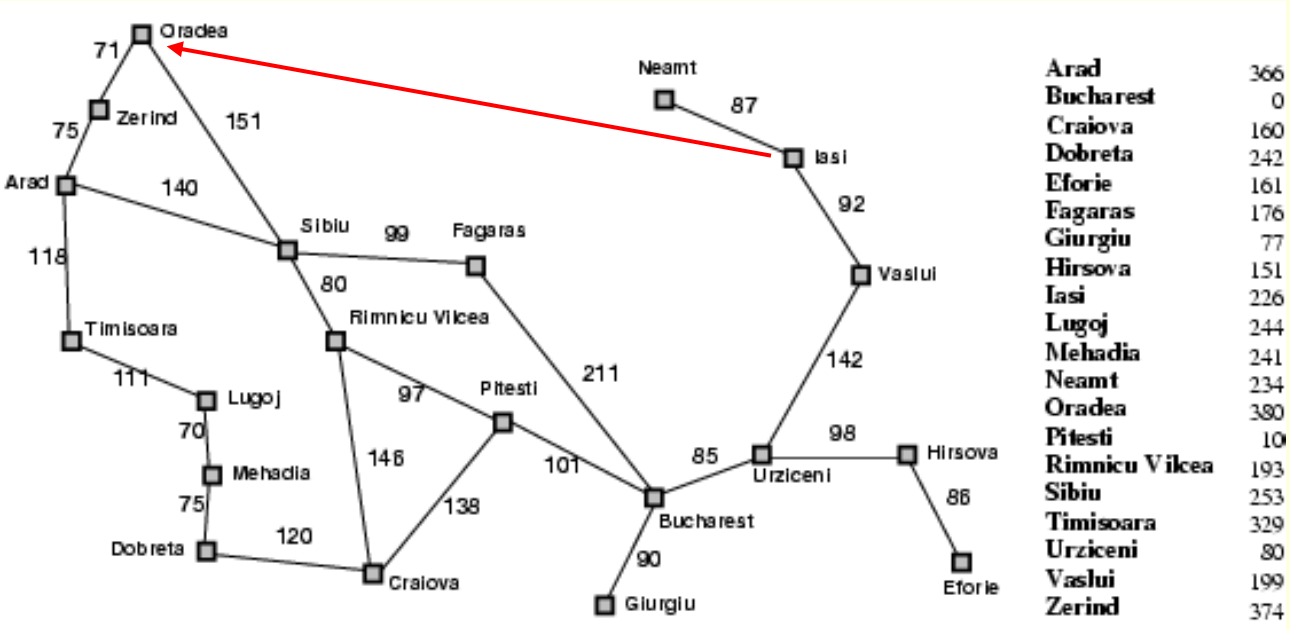
Cautarea Greedy

- Totusi, solutia nu este optimala.
  - Via Sibiu -> Fagaras -> Bucuresti este cu 32 km mai mult decat Sibiu -> Rimnicu Vilcea -> Pitesti -> Bucuresti.
  - De ce nu a fost aleasa calea mai convenabila?...
    - Pentru ca Fagaras este mai aproape de Bucuresti in linie dreapta decat Rimnicu Vilcea!



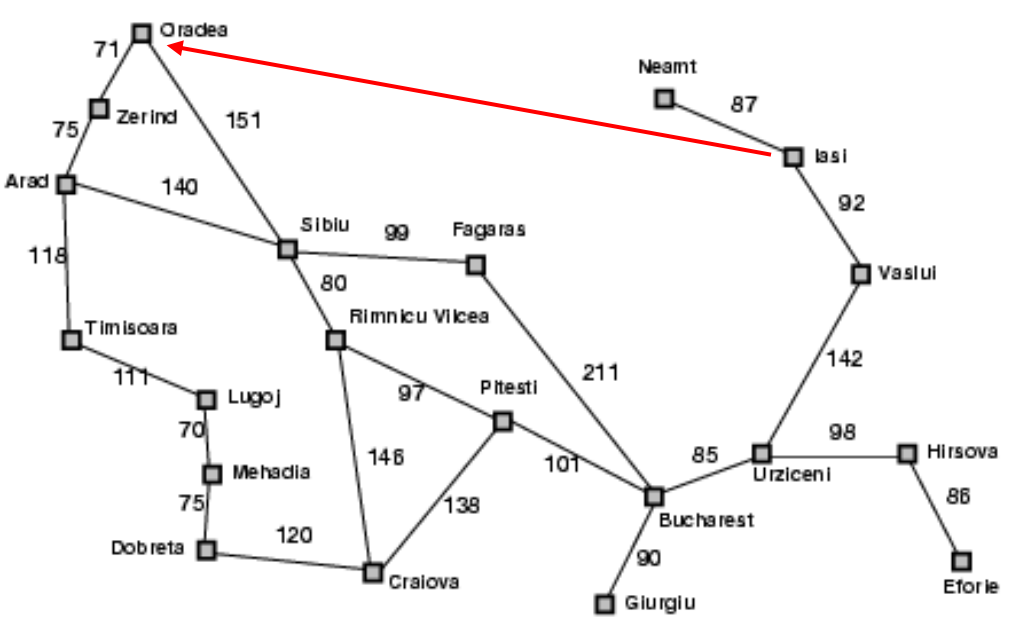
Cautarea Greedy in general gaseste solutia rapid insa nu gaseste intotdeauna solutia optimala.

Cautarea Greedy – dezavantaj…



- Cautarea Greedy poate avea starturi gresite.
- De exemplu, daca vrem sa ajungem din Iasi in Oradea, functia euristica va expanda intai nodul Neamt, insa acesta este un nod care se inchide => Iasi ->Neamt -> Iasi-> Neamt…

Cautarea Greedy – dezavantaj…



Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- Solutia ar fi sa se deplaseze in Vaslui, un nod care este mai departat de nodul tinta - conform cu euristica – si sa continue apoi cu Urziceni, Bucuresti etc.
- Daca nu formulam atent problema pentru a nu face pasi repetitivi, cautarea poate oscila la infinit intre Neamt si Iasi.

Cautarea Greedy

- Este similara cu cautarea in adancime
  - Ca si in acel caz, merge pe un singur drum pana la capat si, daca nu gaseste nodul tinta, se intoarce pentru a alege un alt drum.
- Ca si in cazul cautarii in adancime, algoritmul de cautare Greedy nu este **optimal** si este **incomplet** pentru ca o poate apuca pe un drum infinit si astfel sa nu se mai intoarca pentru a alege alte posibilitati.
- Complexitatea temporala si spatiala: $O(b^m)$, unde
  - b este numarul de noduri in care se expandeaza fiecare nod
  - m este adancimea maxima a spatiului de cautare.

Cautarea A*

- **Cautarea Greedy** minimizeaza costul estimat catre tinta ***h*(*n*)**, ceea ce face sa scada considerabil costul cautarii.
  - Nu este insa nici optimal, nici complet!
- **Cautarea cu cost uniform** minimizeaza costul drumului pana la momentul curent folosind ***g*(*n*)**.
  - Este optimal si complet insa poate fi ineficient!

- Prin combinarea celor doua strategii, vom obtine o functie care tine cont si de costul de pana acum in drumul considerat, si de costul estimat pana la tinta:

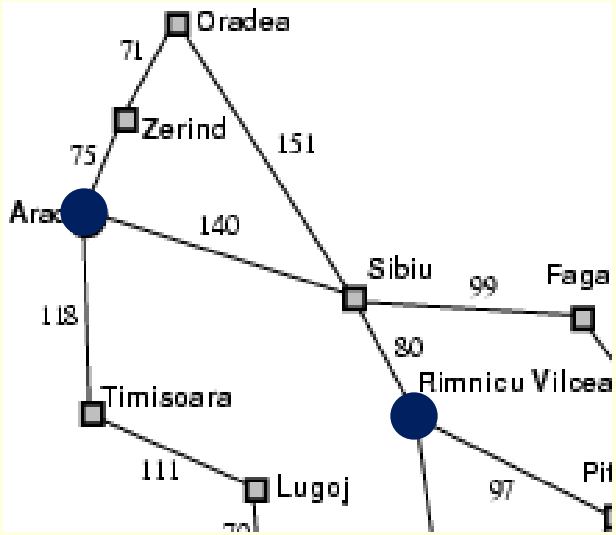
$$f(n) = g(n) + h(n)$$

Cum era…

Cautarea cu cost uniform

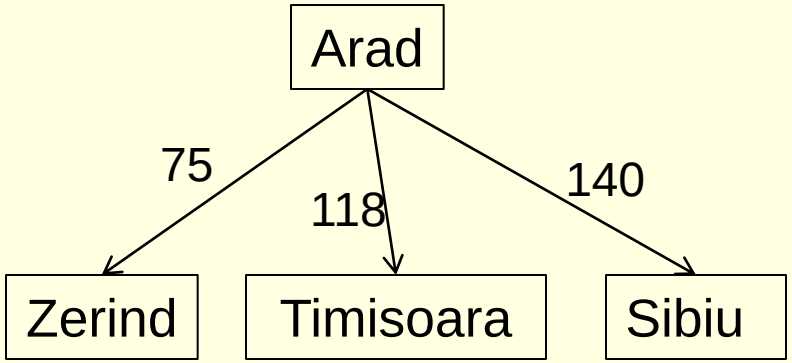
- Este echivalenta cu cautarea in latime daca toate costurile sunt egale.
- Extinde mereu nodul cu costul minim.
- Solutia cu cost minim va fi garantat gasita pentru ca daca exista o cale cu un cost mai mic aceasta este aleasa.

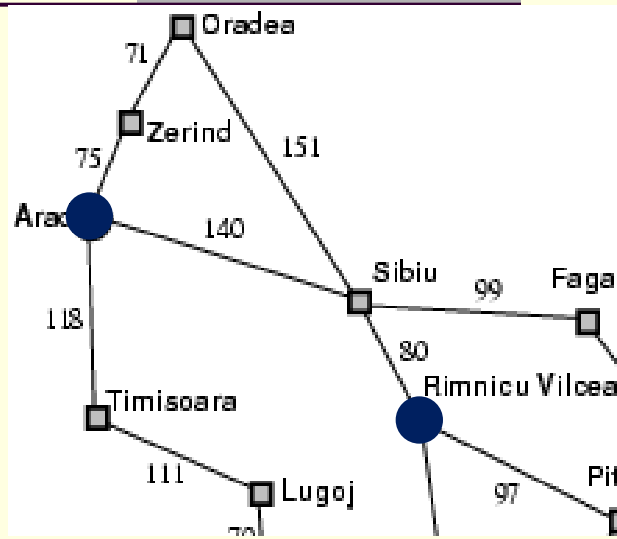
Vrem sa ajungem de la Arad la Rimnicu Vilcea.



Cum era…

Cautarea cu cost uniform

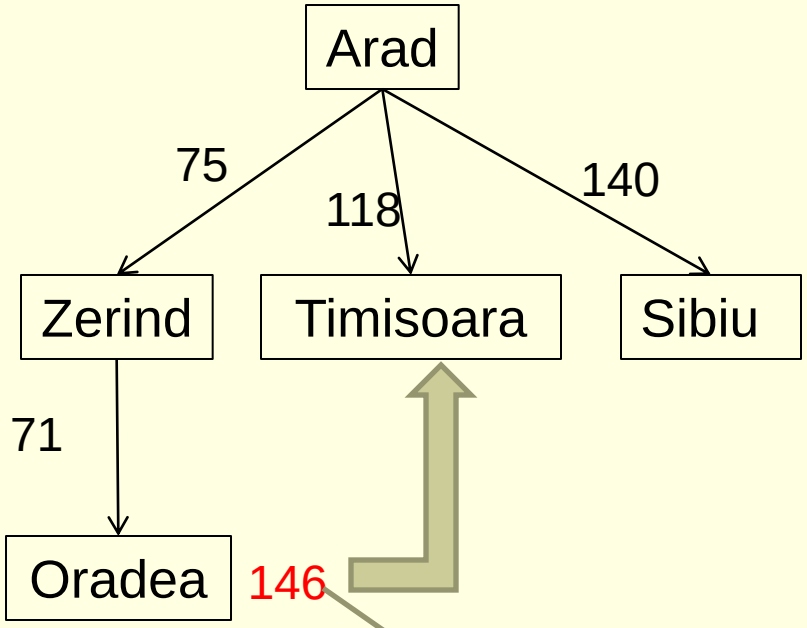


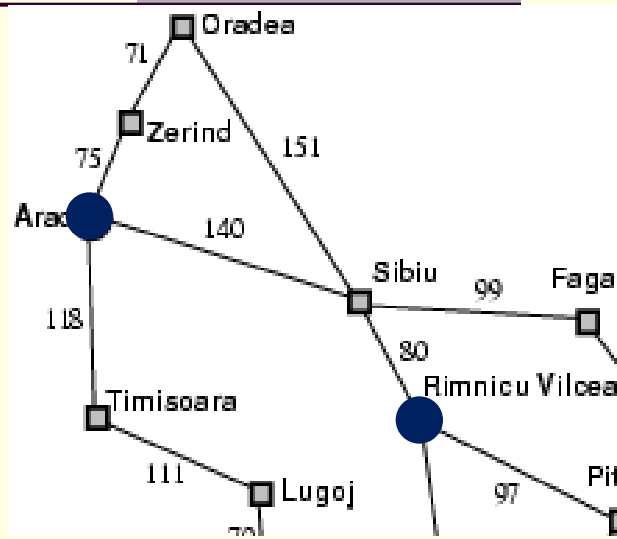


Descendentii sunt asezati in ordine crescatoare in functie de distanta fata de nodul curent.

Cum era…

Cautarea cu cost uniform

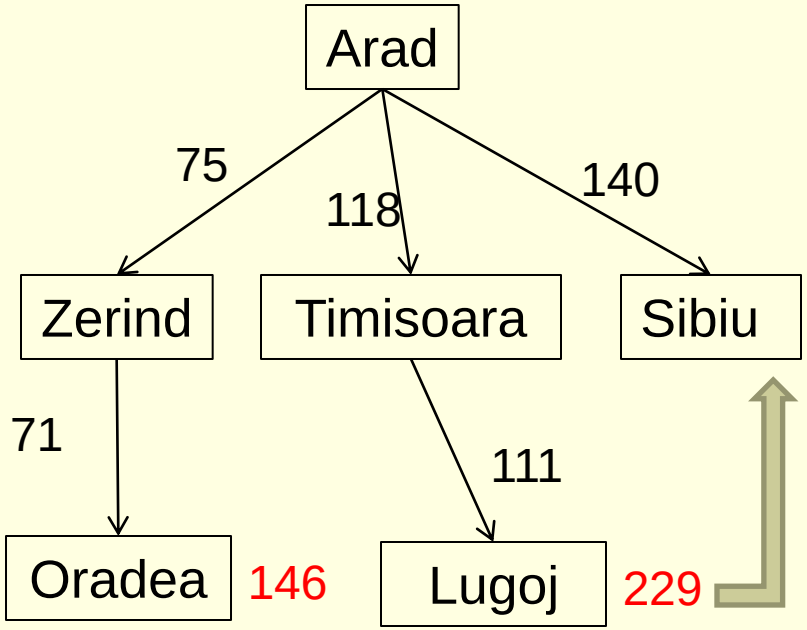


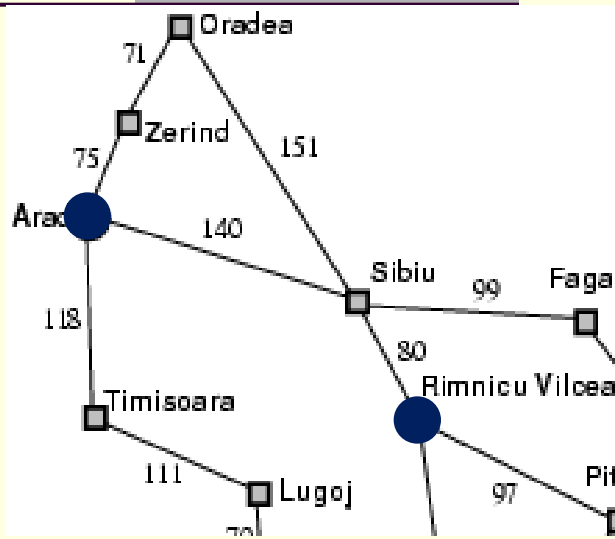


In total, de la Arad la Oradea.

Cum era…

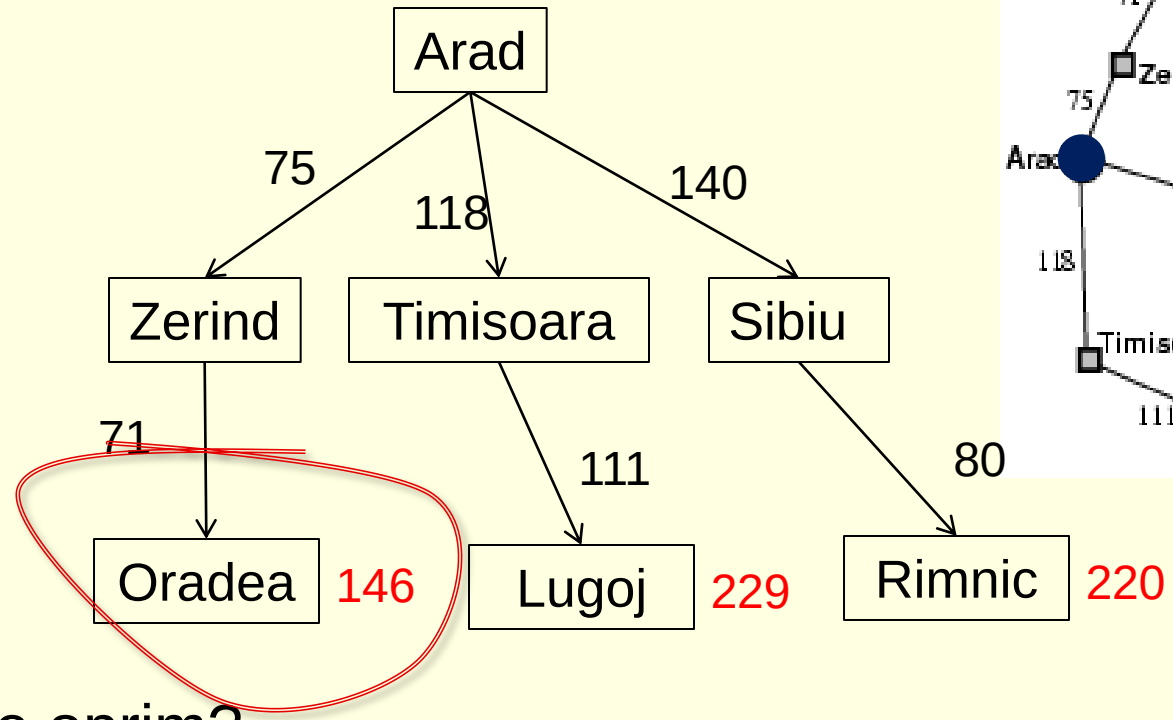
Cautarea cu cost uniform

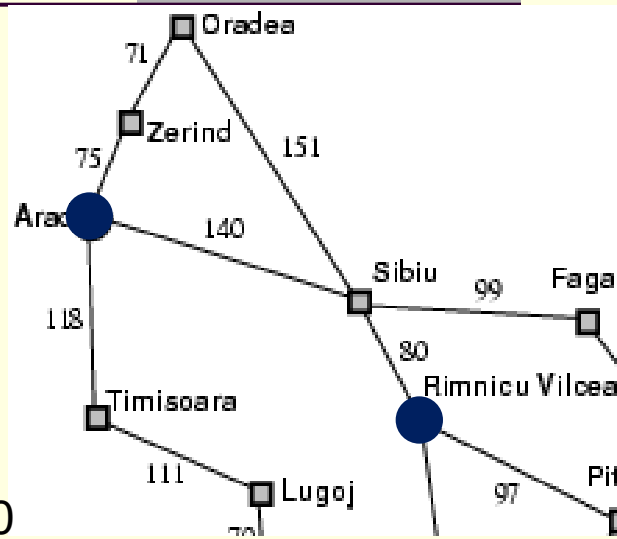




Cum era…

Cautarea cu cost uniform



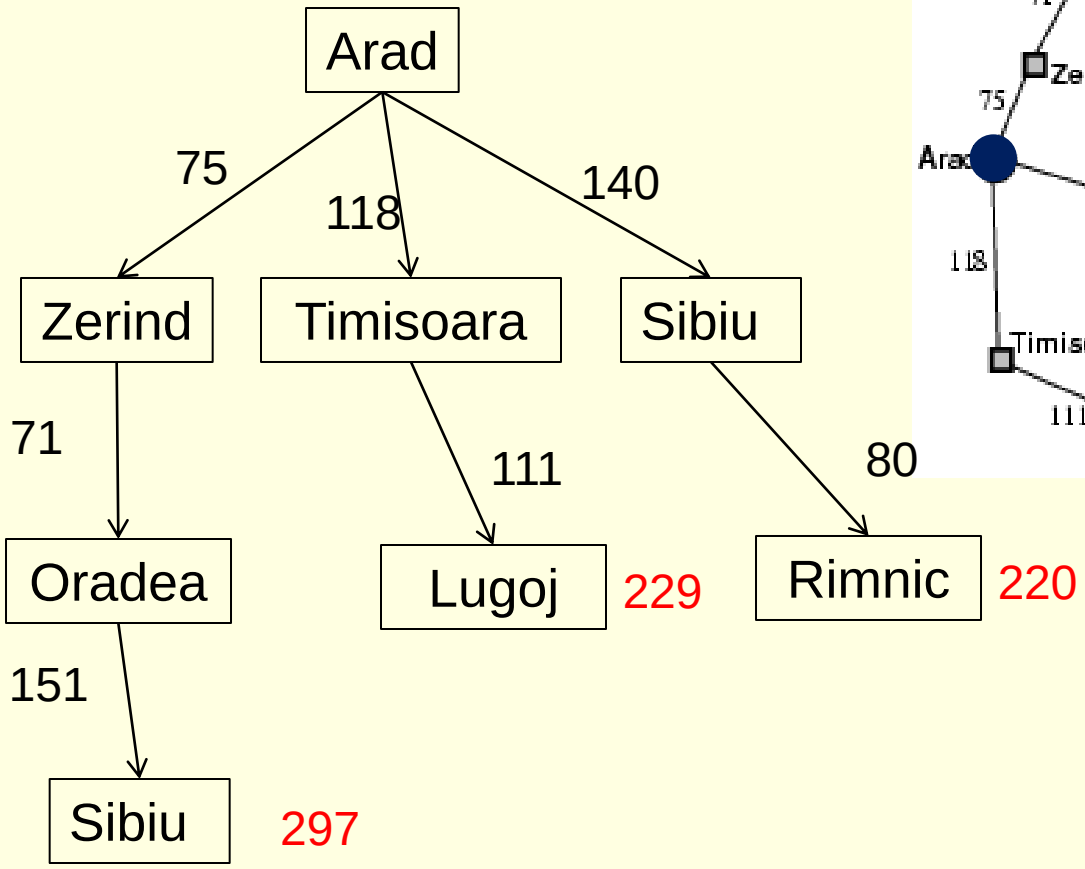


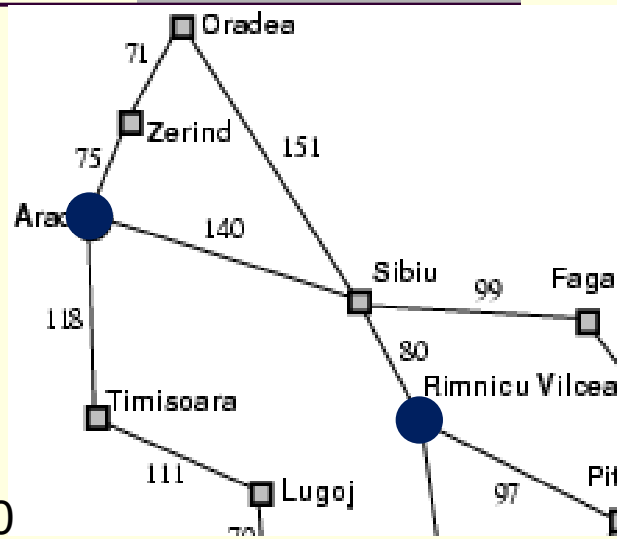
Ne oprim?

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

Cum era…

Cautarea cu cost uniform

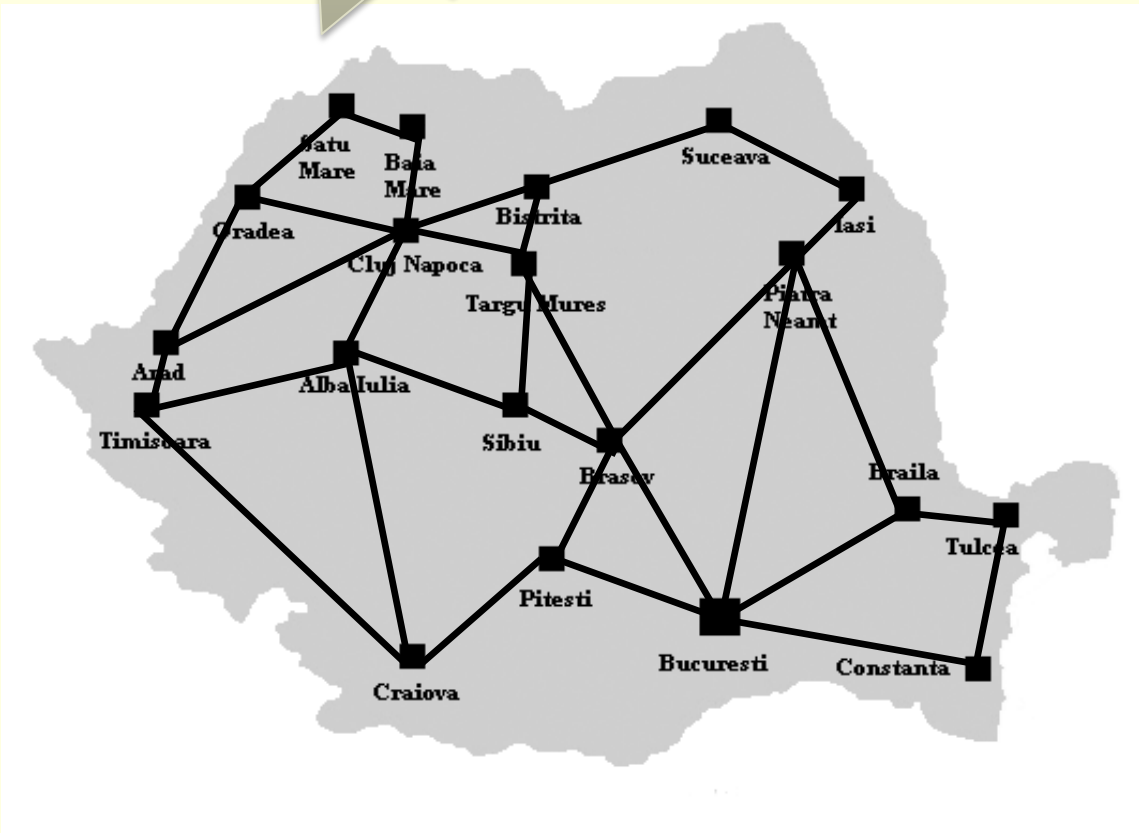




O noua tema…

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.



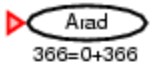
Cautarea A*

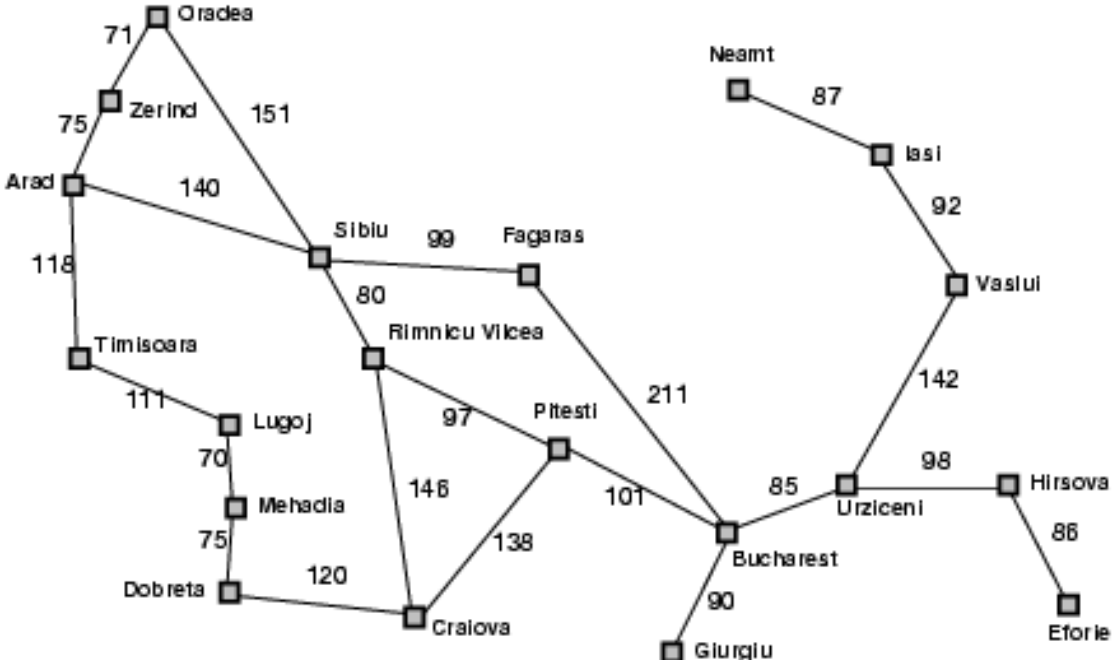
- Cum $g(n)$ da costul drumului de la starea initiala pana la nodul n, iar $h(n)$ estimeaza costul celui mai convenabil drum de la n pana la tinta…

$f(n)$ = costul estimat al celei mai convenabile solutii prin n.

- Este demonstrabil faptul ca algoritmul care foloseste cautarea A* este **complet** si **optimal** cu conditia ca functia h **nu supraestimeaza** costul de ajungere la solutie.

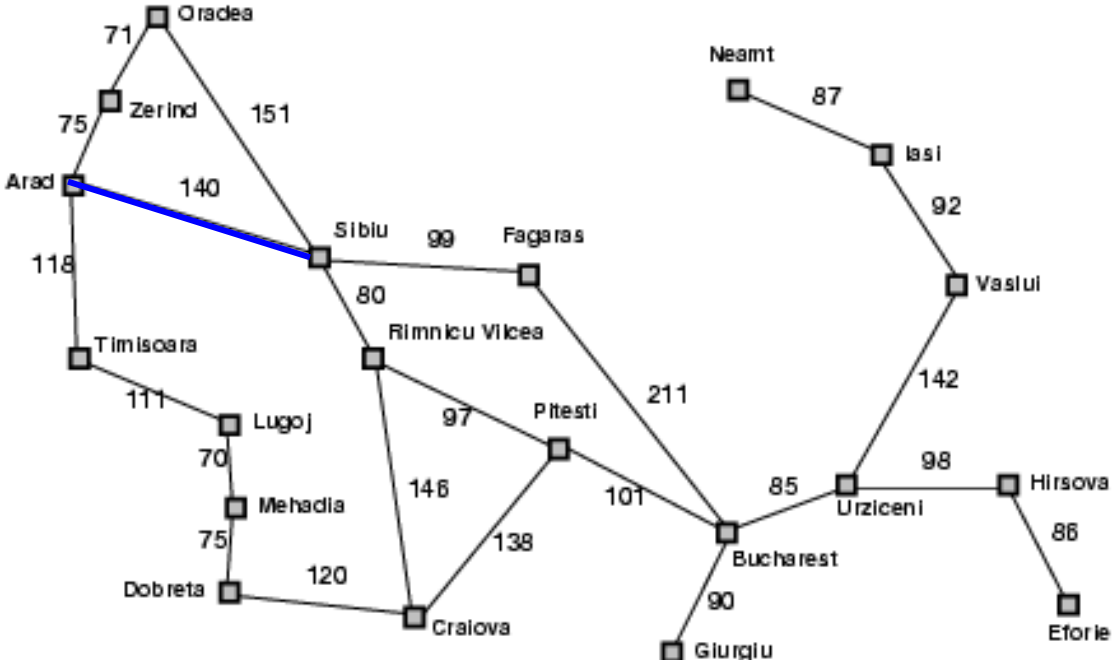
Cautarea A*





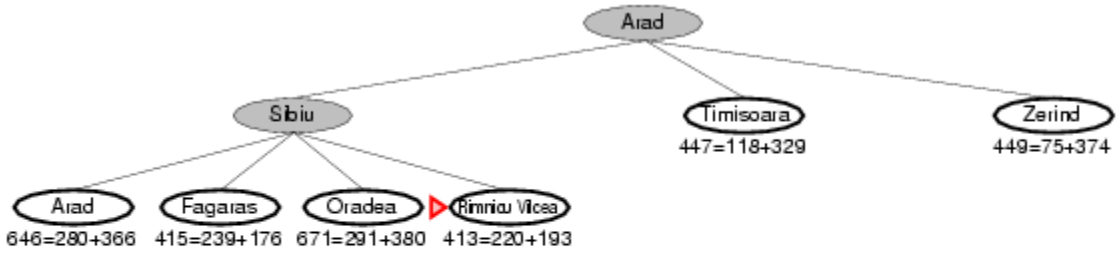
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

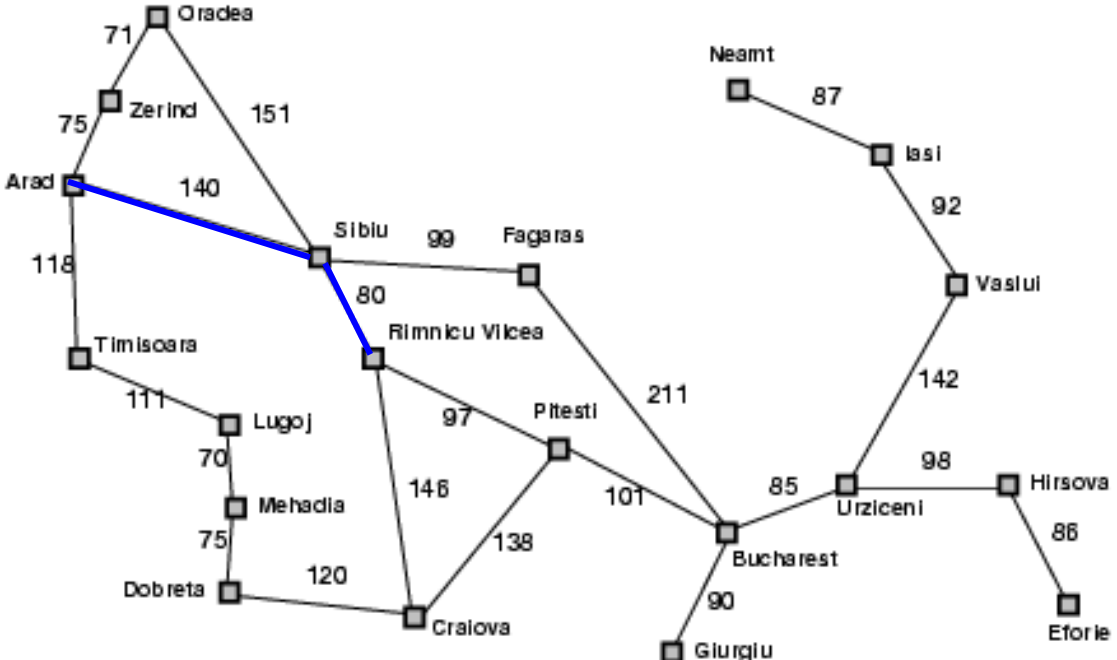
Cautarea A*



Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

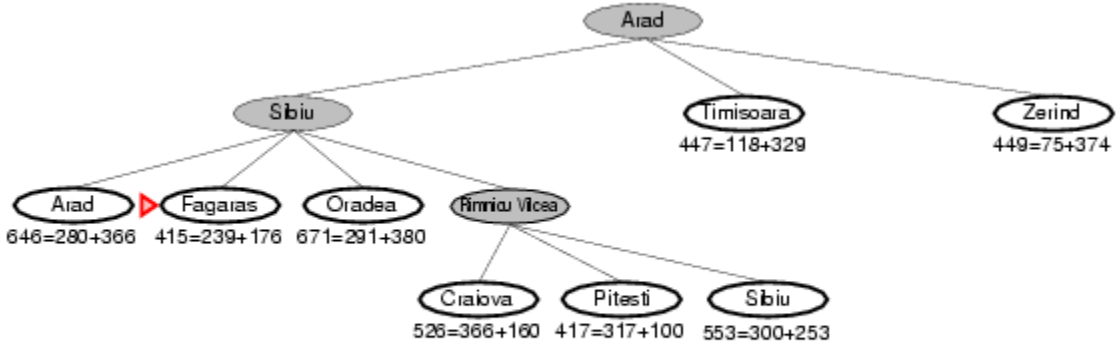
Cautarea A*

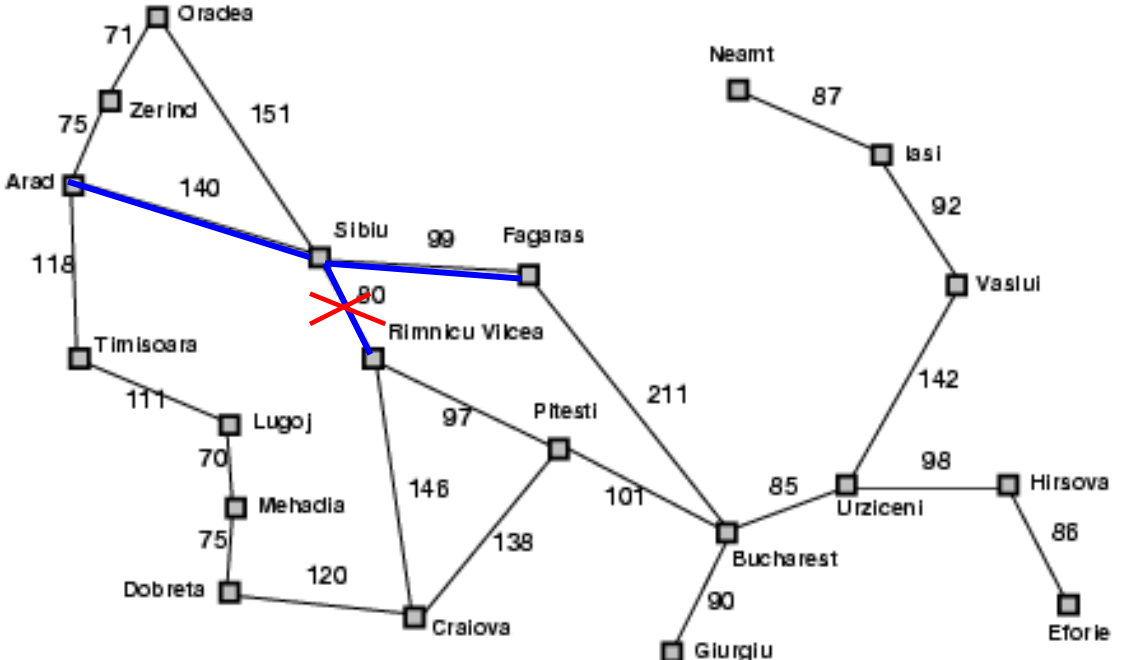




Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

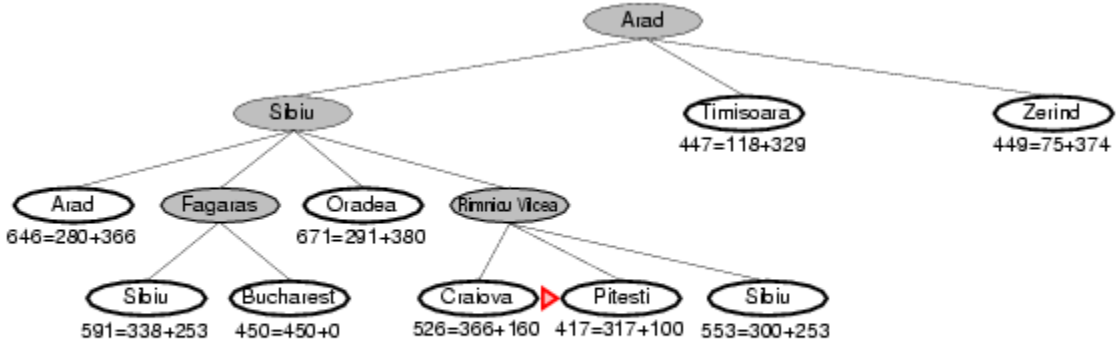
Cautarea A*

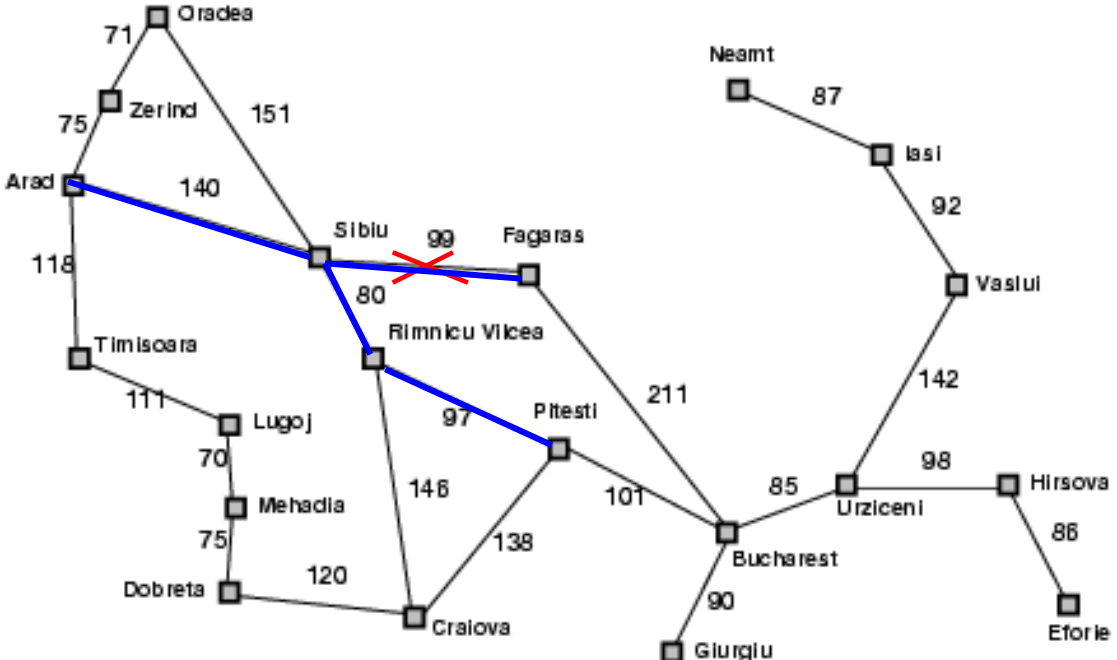




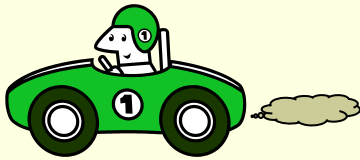
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Cautarea A*

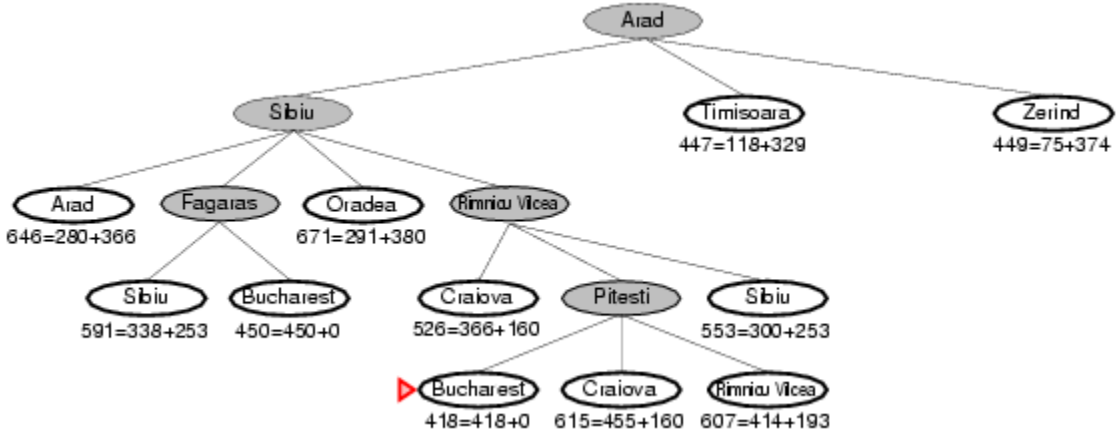


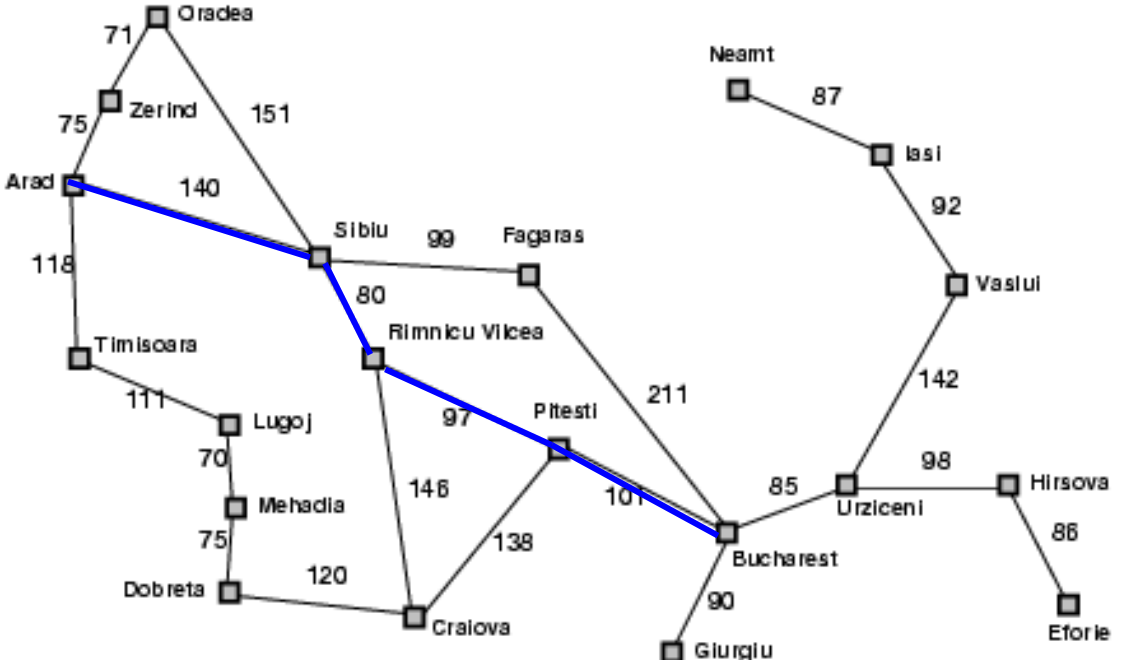


Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



Cautarea A*





Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Cautarea A*

- Complexitatea spatiala: toate nodurile sunt retinute in memorie.
- Complexitatea temporala: $O(b^m)$.
- Este complet si optimal daca h nu supraestimeaza costul pana la starea tinta.

Functii euristice

- Pentru problema de gasire de rute o functie euristica potrivita este cea aleasa, distanta directa dintre 2 orase.
- Alegerea functiilor euristice depinde de la o problema la alta.
- De alegerea functiei euristice depind complexitatile temporala si spatiala.
- Alegerea potrivita a unei functii euristice se face dupa o buna studiere a spatiului starilor problemei.

Functii euristice

Stare curenta

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

Stare tinta

1	2	3
4	5	6
7	8	

- Cum factorul de expandare este aproximativ 3, intr-o cautare la o adancime de 20 de nivele, se ajunge sa fie parcurse aproximativ 3^{20} stari = $3.5 * 10^9$.
- Avem nevoie de o functie euristica cu proprietatea ca nu supraestimeaza numarul de pasi pana la starea tinta.

Functii euristice

Stare curenta

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

Stare tinta

1	2	3
4	5	6
7	8	

- h_1 = numarul de cifre care sunt gresit pozitionate.

h_1

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

= 5

Functii euristice – distanta Manhattan

Stare curenta

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

Stare tinta

1	2	3
4	5	6
7	8	

- h_2 = suma mutarilor necesare pentru ca toate cifrele sa ajunga la starea tinta ca si cum toate celelalte casute ar fi goale.
- Deplasarile se pot face numai pe orizontala si verticala.

h_2

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

= 0 + 0 + 1 + 1 + 0 + 1 + 1 + 2 = 6

Efectul functiilor euristice asupra performantei algoritmului

- **Factorul efectiv de ramificare** *b** afecteaza in mod direct performanta algoritmului.
- Daca numarul total de noduri expandate de catre A* pentru o anumita problema este *N* si solutia se gaseste la o adancime *d* atunci b* este **factorul de ramificare** (in cate noduri se expandeaza fiecare nod) pe care l-ar avea un arbore uniform de adancime *d* care ar contine *N* noduri:

$$N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$

- O euristica este cu atat mai bine definita cu cat face ca numarul de noduri in care se expandeaza fiecare nod sa fie, in medie, cat mai mic.
- *b** trebuie sa fie cat mai aproape de 1!

Dominare functii euristice

- Daca avem doua functii euristice adimisibile h_1 si h_2 cu proprietatea ca $h_2(n) > h_1(n)$ pentru orice nod n atunci spunem ca h_2 **domina** h_1.
  - h_2 este functia euristica mai buna decat h_1 pentru cautare.
- Au fost generate aleator 100 de probleme, fiecare cu solutii aflate la adancimea 2, 4, …, 24 pentru a fi rezolvate cu A* utilizand pe rand, h_1 si h_2.
- In toate situatiile, h_2 este mai buna decat h_1.

Dominare functii euristice

	Costul cautarii		Factorul efectiv de ramificare	
d	**A*(h_1)**	**A*(h_2)**	**A*(h_1)**	**A*(h_2)**
2	6	6	1.79	1.79
4	13	12	1.48	1.45
8	39	25	1.33	1.24
12	227	73	1.42	1.24
16	1301	211	1.45	1.25
24	39 135	1 641	1.48	1.26

Folosind cautarea in adancime iterativa, pentru d = 12, costul cautarii este 364 404, iar factorul efectiv de ramificare 2.78.

Dominare functii euristice

- Asadar, A* folosind h_2 expandeaza mai putine noduri in medie decat A* cu h_1.
- Concluzia: este intotdeauna mai bine sa se utilizeze functii euristice care intorc valori mai mari, doar sa nu supraestimeze valoarea efectiva.

Relaxarea problemelor

- O problema cu mai putine restrictii asupra actiunilor posibile duce la o **relaxare** a problemei initiale.
- Costul unei solutii optimale obtinut pentru o problema *relaxata* reprezinta o functie euristica admisibila pentru problema originala.
- Daca regulile de la puzzle-ul cu 8 valori sunt relaxate astfel incat o cifra se poate muta oriunde dintr-o singura deplasare, atunci $h_1(n)$ da solutia cea mai scurta.
- Daca regulile sunt relaxate astfel incat o cifra se poate muta la orice casuta **adiacenta** atunci $h_2(n)$ da solutia cea mai scurta.

Functii euristice

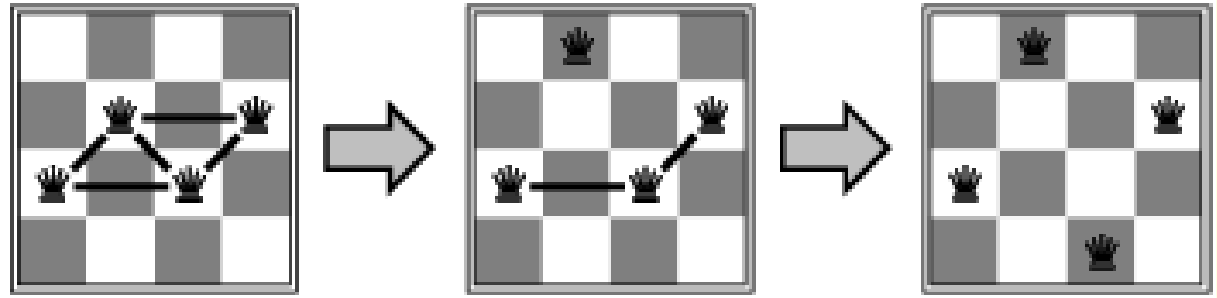
- De multe ori este nevoie sa tinem cont de anumite **caracteristici** ale starii curente care contribuie la evaluarea functiei euristice.
- De exemplu, ce caracteristici ar fi elocvente in cazul functiei euristice pentru jocul de sah? Chiar daca starea tinta este sah mat, se poate tine cont de…
  - Numarul de piese pe care il au fiecare din cei doi jucatori.
  - Numarul de piese atacate de catre piesele adversarului.
    - Tipul pieselor pe care le are fiecare din cei doi jucatori.
    - s.a.m.d.

Algoritmi de cautare locala

- In multe probleme de optimizare, **drumul** catre tinta este irelevant.
- Spatiul starilor este o multime de configuratii complete, o multime de **potentiale solutii**.
- La unele probleme, trebuie gasite configuratii (solutii) care sa satisfaca contrangeri, de exemplu in problema damelor.
- In astfel de cazuri, putem folosi **algoritmi de cautare locala**.
- In astfel de situatii avem o singura **stare curenta** careia i se aduc **imbunatatiri**.

Exemplu: problema celor *n* dame

- Problema este de a pune *n* dame pe o tabla *n* × *n* in asa fel incat doua dame sa nu se afle pe aceeasi line, pe aceeasi coloana sau pe aceeasi diagonala.



- **Stari**: 4 dame in 4 coloane ($4^4 = 256$ de stari posibile)
- **Actiuni**: muta o dama pe coloana
- **Stare tinta**: sa nu se atace damele reciproc
- ***h*(*n*)** = numarul de perechi de dame care se ataca reciproc

Hill climbing

- Este ca si cand ai urca un munte, este ceata foarte deasa si ai avea amnezie.
- Este vorba de o miscare continua inspre valori mai bune, mai mari (de aici, *urcusul pe munte*).
- Algoritmul nu mentine un arbore de cautare, prin urmare, pentru fiecare nod se retine numai starea pe care o reprezinta si evaluarea sa.

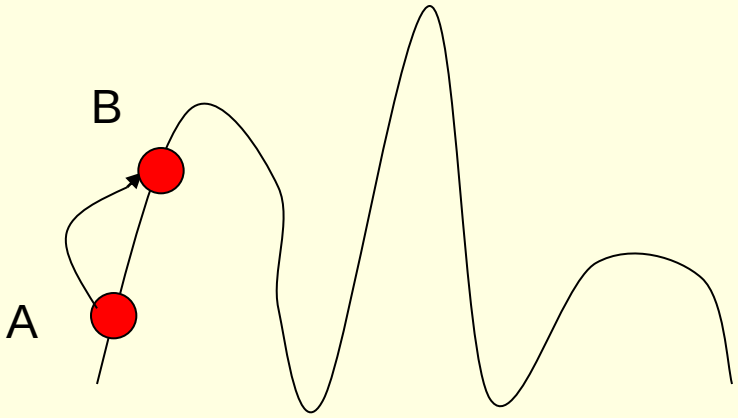
Hill climbing

functia hill_climbing(*problema*) ***intoarce*** o solutie
Se pastreaza la fiecare reapelare: nodul *curent* si nodul *urmator.*

curent = genereaza_nod(stare_initiala[problema])

Cat timp este posibil *executa*
    urmator = succesorul cel mai bun al nodului *curent*
    Daca eval(*urmator*) < eval(*curent*) *atunci*
        intoarce *curent*
    curent = urmator
Sfarsit cat timp

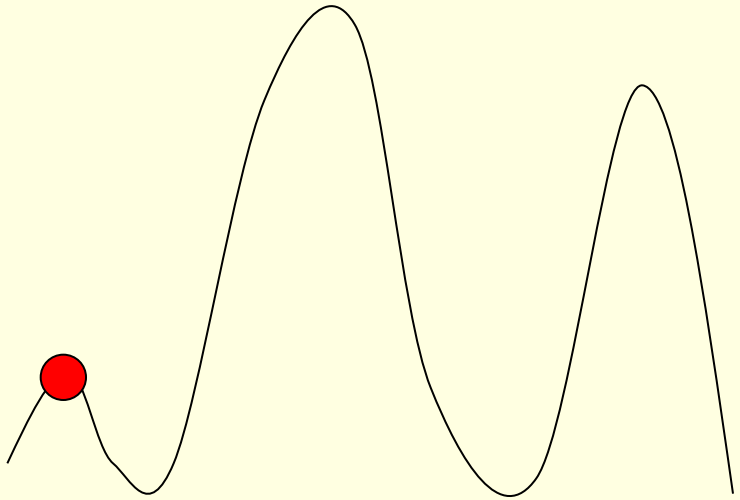
Hill climbing



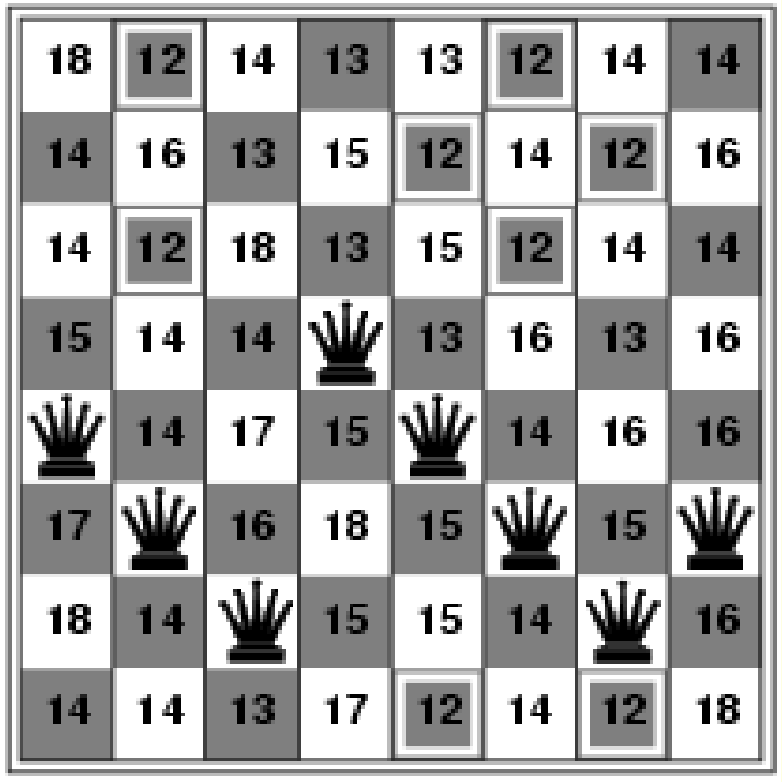
- Din starea curenta A => starea B.
- Atunci cand sunt mai multi succesori care sunt toti la fel de buni, algoritmul il alege pe unul in mod aleator.

Dezavantaje hill climbing

- **Maxime locale**: este vorba de un varf care este mai mic decat cel mai inalt varf din spatiul starilor. Cand se ajunge la maxime locale, algoritmul se opreste pentru ca nu mai poate *cobori* dealul.
  - Solutia gasita poate fi foarte slaba!

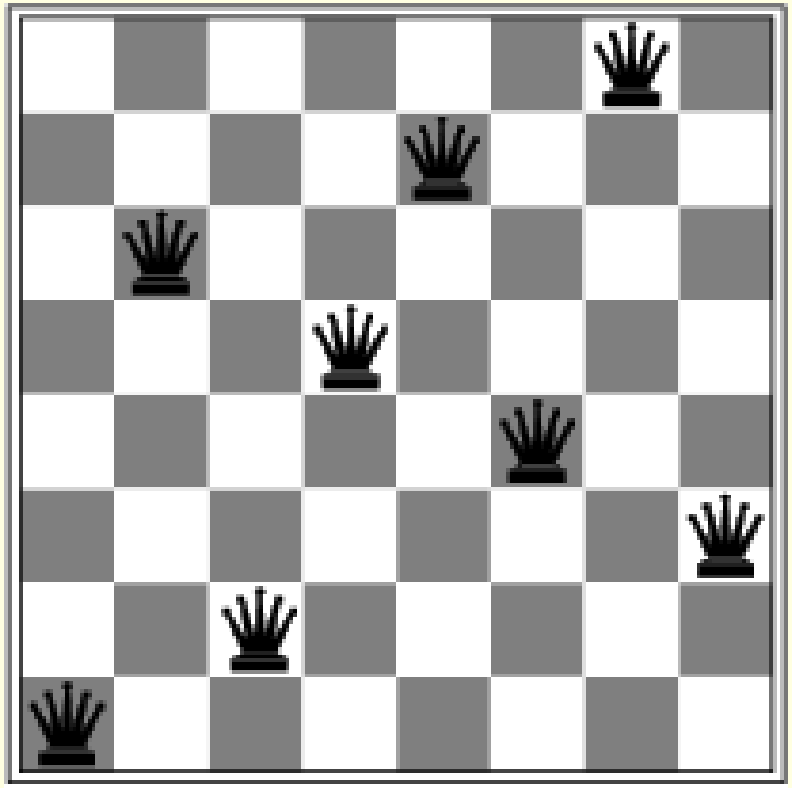


Hill-climbing - problema celor 8 dame



- *h* = numarul de perechi de dame care se ataca reciproc direct sau indirect.
- *h = 17* pentru starea de mai sus.

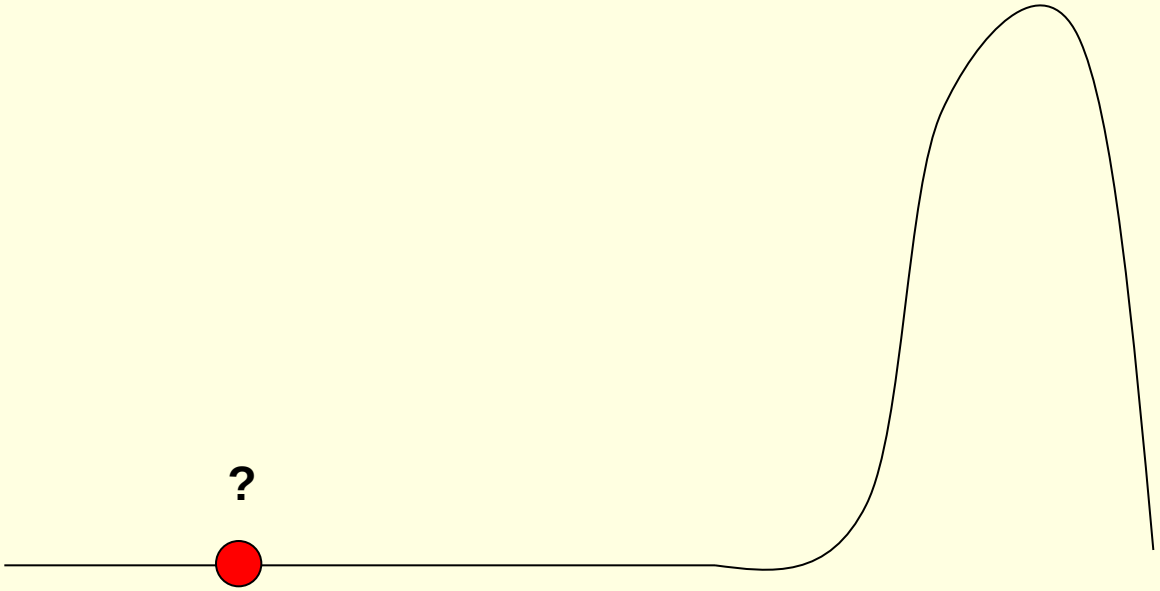
Hill-climbing problema celor 8 dame



- Un minim local unde *h = 1*

Dezavantaje hill climbing

- **Platouri** – o zona din spatiul de cautare in care functia de evaluare are valori constante.
- Cautarea va merge in aceste cazuri la intamplare.



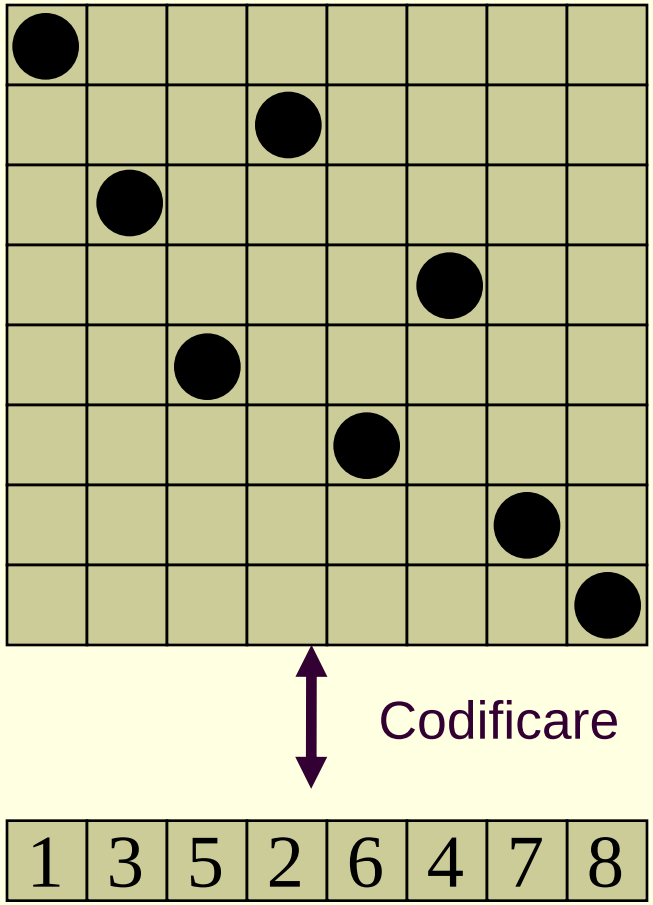
Hill climbing cu restart aleator

- Cand apar astfel de situatii in care cautarea nu realizeaza nici un progres, un lucru bun ar fi sa se reinceapa cautarea de la un alt punct de start.
- **Hill climbing cu restart aleator** face o serie de cautari folosind hill climbing cu porniri din diverse stari aleatoare.
- Fiecare rulare dureaza pana cand cautarea nu mai inregistreaza imbunatatiri sau a trecut un numar de iteratii.
- Cel mai bun rezultat din fiecare cautare este retinut.
- Se poate repeta pentru un numar fixat de iteratii sau se poate continua pana cand cel mai bun rezultat obtinut nu a mai fost imbunatatit de mai multe generatii.

Reprezentarea pentru problema damelor

Potenţială soluţie:
o configuraţie a celor 8 dame

Cromozomul:
o permutare a primelor 8 cifre.



Reprezentarea pentru problema damelor

- Considerăm pentru început o configuraţie pe care o generăm aleator.

Cum?

Initializarea unei posibile solutii

```
functie initializare() intoarce configuratie
    M = {1, 2, 3, 4, 5, 6, 7, 8}
    i = 0
    Cat timp (lungime(M) > 0) executa
        g = (int)(lungime(M) * random(1)) //vezi slide-ul urm pt C
        configuratie[i++] = M[g]
        Scoate elementul M[g] din multimea M
        lungime(M)--;
    Sfarsit cat timp
intoarce configuratie
```

Intoarce un numar aleator in intervalul [0, 1]

Cum generam in limbajul *C* un intreg *g* aleator intre 0 si *n*-1

```
include <time.h>
include <stdlib.h>
int main(){
…
time_t t;
srand((unsigned) (time(&t)));
int g = rand() % n;
…
}
```

Evaluarea unei configuratii

```
functie evaluare(configuratie) intoarce numar_erori
    erori = 0;
    Pentru i = 1 pana la 7 executa
        Pentru j = i + 1 pana la 8 executa
            Daca (|configuratie[i] - configuratie[j]| == |i - j|) atunci
                erori++
            Sfarsit daca
        Sfarsit pentru
    Sfarsit pentru
Intoarce erori
```

Schimbarea pentru problema damelor

- O mică variaţie într-o permutare:
  - Se aleg două valori în mod aleator (5 şi 7 în imaginea din stânga).
  - Poziţiile celor două valori sunt interschimbate.

Schimbarea pentru problema damelor

```
functie perturbare(configuratie) intoarce configuratie
    x = (int)(lungime(configuratie) * random(1))
    y = (int)(lungime(configuratie) * random(1))
    Cat timp (x == y) executa
        y = (int)(lungime(configuratie) * random(1))
    Sfarsit cat timp
    temp = configuratie[x]
    configuratie[x] = configuratie[y]
    configuratie[y] = temp
intoarce configuratie
```

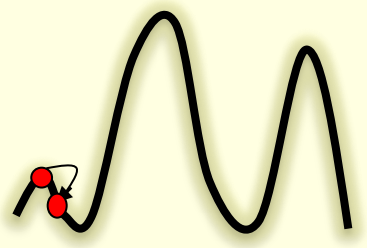
Hill climbing pentru problema celor 8 dame

1. configuratie = initializare();
2. Pentru un numar de iteratii
    1. eval_curent = evaluare(configuratie)
    2. configuratie1 = perturbare(configuratie)
    3. if(eval(configuratie1) < eval(configuratie))
        1. configuratie = configuratie1
3. Sfarsit pentru
4. Afisare (configuratie)

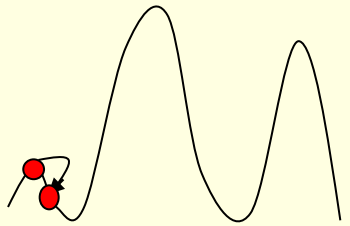
Nu uitati ca valorile lui *eval* trebuie minimizate!!

Transformati acest algoritm intr-un hill climbing cu restart aleator!

Simulated annealing



Simulated annealing



- In loc de a reincepe cautarea dintr-o noua stare generata aleator, aceasta poate sa si *coboare de pe munte* pentru a scapa de maxime locale – acest lucru este permis de catre **simulated annealing**.
- In loc sa fie alese cele mai bune actiuni, aceasta alege si miscari in mod aleator.
- Daca actiunea imbunatateste situatia, atunci este intotdeauna executata.
  - Altfel, algoritmul alege o actiune cu o anumita probabilitate mai mica decat 1.
  - Aceasta probabilitate descreste exponential cu cat de rea este actiunea (cu ΔE, care spune cu cat este inrautatita solutia).

Simulated annealing

- *T* este un parametru folosit pentru a determina probabilitatea de a selecta actiunea mai proasta.
- Cu cat *T* este mai mare, cu atat actiunile proaste au sanse mai mari sa fie selectate.
- Cand *T* tinde la 0, algoritmul devine din ce in ce mai mult precum hill climbing.
- Apare un parametru de intrare, *planificare,* cu rolul de a determina valoarea lui *T* in raport cu cate iteratii au avut loc.

Simulated annealing

functia simulated_annealing(*problema, planificare*) ***intoarce*** o solutie
Se pastreaza la fiecare reapelare: nodul *curent* si nodul *urmator.*

curent = genereaza_nod(stare_initiala[problema])

Pentru t = 1 *pana la* ∞ *executa*
T = planificare[t]
urmator = un succesor al nodului *curent* ales aleator
ΔE = *eval*(*urmator*) - eval(*curent*)
Daca $\Delta E > 0$ *atunci curent = urmator*
Altfel curent = urmator cu probabilitatea $e^{\Delta E/T}$
Sfarsit pentru

Aplicatii la probleme cu constrangeri

- La problema celor 8 dame, o stare initiala are toate cele 8 dame pe tabla, iar un operator muta o dama dintr-un patrat in altul.
- Algoritmii care rezolva astfel de probleme se numesc metode **euristice de reparare** pentru ca repara inconsistentele din configuratia curenta.
- In alegerea unei noi valori pentru o variabila, o **euristica cu conflicte minime** este cea mai potrivita – se urmareste selectarea unei valori care sa duca la minimizarea numarului de conflicte cu celelalte variabile.
- Aceste euristici sunt capabile sa rezolve problema celor un milion de dame in medie in mai putin de 50 de pasi.

Recapitulare 1/2

- **Cautarea *intai cel mai bun*** este doar un arbore general de cautare in care nodurile de cost minim (in functie de o anumita masura) sunt expandate mai intai.
- Daca minimizam costul estimat pentru a ajunge la tinta, $h(n)$, avem **cautare greedy**.
  - Timpul de cautare este redus in comparatie cu algoritmii *neinformati*, insa nu este nici complet, nici optimal.
- Minimizarea lui $f(n) = g(n) + h(n)$ combina avantajele cautarii cu cost uniform cu cele ale cautarii greedy.
- Daca evitam starile repetate si nu supraestimam $h(n)$, avem **cautarea A***.

Recapitulare 2/2

- A* este complet, optimal, insa complexitatile pentru timp si spatiu sunt inca mari.
- Complexitatea temporala a algoritmilor euristici depinde de calitatea functiei euristice folosite.
- Algoritmii bazati pe imbunatatiri iterative tin in memorie o singura stare, dar se pot bloca in maxime locale.
- Simulated annealing ofera o cale de a scapa de maxime locale si este complet si optimal daca variabila *planificare* are un proces lung si gradual de scadere.
- Pentru probleme cu satisfacere de constrangeri, euristicile care ordoneaza variabilele pot aduce imbunatatiri foarte mari ale performantei.