Tehnici de cautare informata: A*

Algoritmul A* se foloseste pentru a gasi un drum de cost minim de la un nod-start la un nod-scop intr-un graf cu muchii/arce ponderate (cu costuri).

Datele de intrare:

- graful (nodurile, muchiile/arcele impreuna cu costurile lor)
- nodul din care incepe cautarea (nodul-start)
- Un scop dat sub forma unei conditii pe care trebuie sa o indeplineasca nodul cautat (se poate oferi chiar nodul propriu-zis, conditia fiind relatia de egalitate cu acest nod). Vom numi mai departe nodul care indeplineste c.
- O estimare (euristica) a costului de la fiecare nod din graf la nodul (nodurile) scop.

Notatii:

- f - costul unui drum
- $\hat{f}$ - costul estimat al unui drum
- g(nod_c) - costul de la nodul start la un nod curent, nod_c, din drum
- h(nod_c) - costul de la nodul curent la un nod scop pe un anumit drum
- $\hat{h}$(nod_c) - costul estimat de la nodul curent la un nod scop

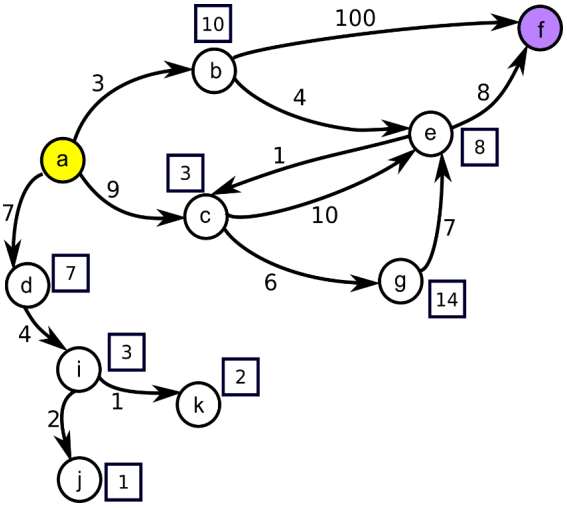
Pentru un drum dat D, avem formula: $f_D(nod\_c) = g_D(nod\_c) + h_D(nod\_c)$,

unde nod_c e un nod din drumul D.

Deoarece pe parcursul construirii arborelui de parcurgere nu cunoastem costul adevarat de la nodul curent la un nod scop (graful fiind descoperit pe masura ce e parcurs), ne vom folosi in algoritm de formula costului estimat:

$\hat{f}_D(nod\_c) = g_D(nod\_c) + \hat{h}_D(nod\_c)$

Consideram ca avem graful cu muchii ponderate, de mai jos:



Explicatii desen:

- informatia nodului („a”, „b”, „c”, …)

- nodul de start: „ a ”

- noduri scop: „ f ”

- costul muchiilor / arcelor dat de functia **g**:

g(„a”, „c”) = 9, **g**(„c”, „g”) = 6,

g(„g”, „e”) = 7,…

- valoarea functiei euristice $\hat{h}$ pentru fiecare nod (*numerele din patrate*):

$\hat{h}$(„a”) = ∞, $\hat{h}$(„f”) = 0,
$\hat{h}$(„g”) = 14, $\hat{h}$(„e”) = 8, …

- Fie drumul D = („a”, „c”, „g”, „e”).

Rezulta $\hat{f}_D(„e") = (9 + 6 + 7) + 8$, adica (suma costurilor muchiilor drumului D) + (valoarea functiei euristice pentru nodul „e”)

Pasii algoritmului

Se considera doua liste: OPEN (cu nodurile descoperite care inca nu au fost expandate) si CLOSED (cu nodurile descoperite si expandate).

Pasii de mai jos sunt preluati din cartea „Aspecte ale Cautarii si Reprezentarii Cunostintelor in Inteligenta Artificiala” (BALCAN Maria Florina, HRISTEA Florentina, Editura Universitatii din Bucuresti, 2004):

1. In lista open se pune la inceput doar nodul de pornire.
2. Initial lista closed e vida
3. Cat timp lista open nu e vida se executa repetitiv pasii urmatori:
   - Se extrage primul nod, *n*, din lista open si se pune in closed.
   - Daca nodul *n* este nod scop, se opreste cautarea si se afiseaza drumul de la nodul-start pana la nodul *n*.
   - Se extinde nodul *n*, obtinand succesorii lui in graf. Nu se vor lua in considerare succesorii care se afla in drumul de la nodul start la *n*. Toti succesorii il au ca parinte pe n. Toti succesorii care nu se afla deja in open sau closed sunt inserati in lista open astfel incat aceasta sa fie in continuare ordonata crescator dupa f. Daca sunt doua noduri cu acelasi f, se aseaza inainte nodul cu g-ul mai mare.
   - Pentru succesorii care sunt deja in open sau closed, in cazul in care pentru drumul care trece prin n, s-a obtinut un f mai mic, li se schimba parintele la n, si li se actualizeaza f-ul, iar nodurile din open sunt repozitionate in lista astfel incat sa ramana ordonata crescator dupa f (si descrescator dupa g).
   - Pentru nodurile din closed (care au fost deja expandate) ar trebui refacut calculul pentru nodurile succesoare lor, prin urmare, cel mai simplu este sa le readaugam in open).

Implementare

Pentru implementare putem considera niste clase ajutatoare, care ar fi adaptate la particularitatile problemei curente rezolvate cu A*.

Clasa **NodParcurgere** reprezinta clasa prin care se memoreaza informatiile despre nodurile din arborele de parcurgere. Poate avea urmatoarele proprietati:

- nod - referinta catre nodul corespunzator din graf
- parinte - referinta catre nodul-parinte din arbore. Pentru radacina arborelui, parintele va avea valoarea None.

- g - costul de la radacina arborelui pana la nodul curent
- f - costul estimat pentru drumul care porneste de la radacina si trece prin nodul curent
- expandat - o proprietate optionala (booleana). O putem folosi in locul listei closed.

si urmatoarele metode:

- expandeaza - care va returna o lista cu toti succesorii posibili ai nodului curent
- test_scop - care testeaza daca nodul e nod scop

Clasa **Nod** se refera la nodurile efectiv aflate in graf si ar trebui sa aiba minim proprietatile:

- info: informatia nodului
- h: estimarea facuta pentru nod (valoarea functiei euristice pentru nod)

Clasa ***Problema*** care contine datele particulare ale problemei.

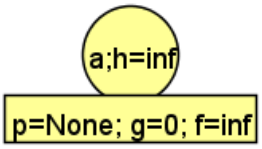
Pseudocod

```
Initial lista open e vida
Cream primul nod din arborele de parcurgere corespunzator nodului de start np_start
Punem nodul np_start in lista open
Cat timp (open nu e vida) repetam:
    extragem primul nod din open si il punem in variabila nod_curent
    daca nod_curent indeplineste conditia scop:
        afisam drum
        oprim cautarea
    expandez nod_curent
    adaug nod_curent in closed
    pentru fiecare succesor s:
        nod_nou = None
        daca s nu apartine drumului lui nod_curent:
            daca s e in open:
                    daca f-ul nodului din open e mai mare decat f-ul gasit pentru s sau
f-urile sunt egale si g-ul nodului din open e mai mic decat g-ul gasit pentru s:
                    scoate nodul vechi din open
                    seteaza pentru s parintele, g-ul si f-ul
                    seteaza nod_nou = s
            daca s a fost expandat(s e in closed):
                daca f-ul nodului din closed e mai mare decat f-ul gasit pentru s sau
f-urile sunt egale si g-ul nodului din closed e mai mic decat g-ul gasit pentru s:
                    sterge nodul din closed
                    seteaza pentru s parintele, g-ul si f-ul
                    ca sa adaug nodul in open (pentru a fi reexpandat) setez nod_nou = s
            altfel non_nou = s
            daca nod_nou contine un nod de inserat:
                pune nod_nou in open astfel incat open sa ramana ordonata
                    (crescator dupa f, apoi descrescator dupa g)
```

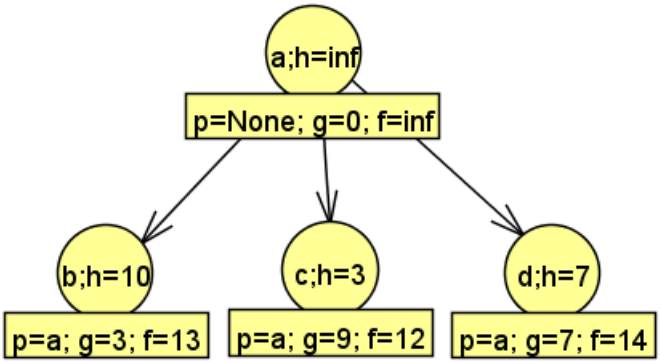
Explicatii detaliate pas cu pas (pt graful din desen)

Explicam si desenam cum va arata arborele de cautare la fiecare pas al algoritmului A*.
 Lista **open** contine nodurile frunza ale arborelui de cautare (mai exact acele noduri care nu au fost inca expandate) si in orice moment trebuie sa fie sortata (crescator dupa f, apoi descrescator dupa g).
 Lista **closed** contine nodurile interioare ale arborelui de cautare (care au fost deja expandate anterior) [sau frunze care au fost expandate, dar care nu au avut fii] si nu necesita niciun fel de sortare (nodurile se adauga mereu la final).
 La fiecare pas, algoritmul A* expandeaza primul nod din lista open (sau daca acesta este nod_scop, opreste cautarea si afiseaza drumul obtinut intre el si radacina arborelui).

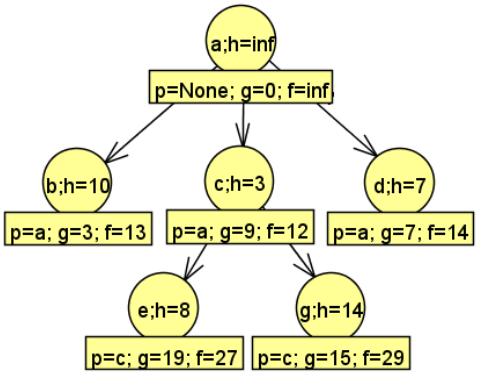
Pas 1:
Initializam listele open si closed.
-- **open** = [((a, h=inf), parinte=None, f=inf, g=0)]
-- **closed** = [ ]



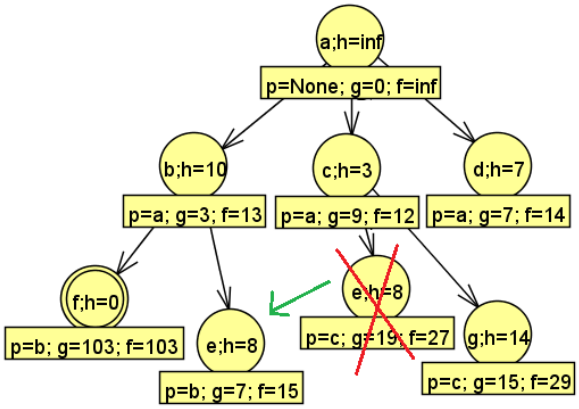
Pas 2:
Mutam nodul „a” din open in closed. Nodul „a” nu este nod_scop, deci il extindem (fiii „b”, „c”, „d” sunt desenati pe arbore in ordinea in care sunt gasiti folosind lista de muchii, dar in lista open ei vor fi introdusi astfel incat toata lista sa ramana sortata corect).
-- **open** = [((c, h=3), parinte=a, f=12, g=9),
((b, h=10), parinte=a, f=13, g=3),
((d, h=7), parinte=a, f=14, g=7)]
-- **closed** = [((a, h=inf), parinte=None, f=inf, g=0)]



Pas 3:
Mutam nodul „c” din open in closed. Nodul „c” nu este nod_scop, deci il extindem (fiii „e”, „g” se adauga in open, respectand sortarea).
-- **open** = [((b, h=10), parinte=a, f=13, g=3),
((d, h=7), parinte=a, f=14, g=7),
((e, h=8), parinte=c, f=27, g=19),
((g, h=14), parinte=c, f=29, g=15)]
-- **closed** = [((a, h=inf), parinte=None, f=inf, g=0),
((c, h=3), parinte=a, f=12, g=9)]

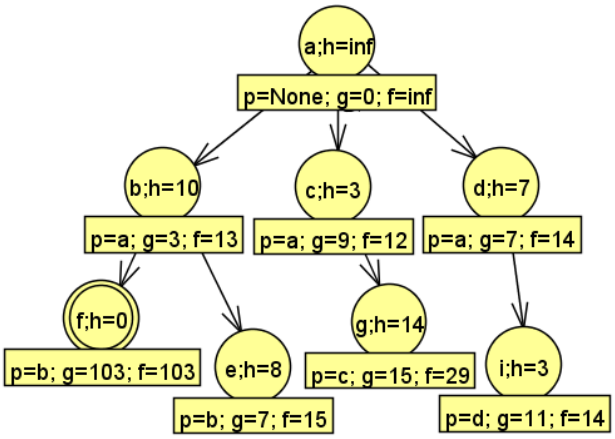


Pas 4:
Mutam nodul „b” din open in closed. Nodul „b” nu este nod_scop, deci il extindem (fii „e” si „f”).
 Nodul „e” ***se afla deja in lista open*** (cu parinte=c, f=27, g=19), dar pentru ca f-ul calculat pe drumul gasit acum este mai bun (f=15 < 27) vom actualiza informatiile pentru nodul

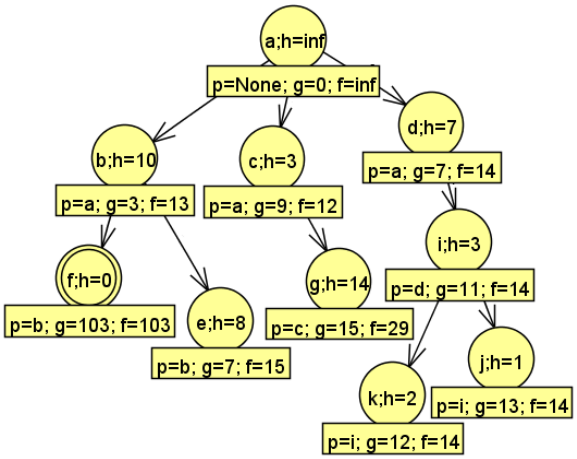


„e” (parinte=b, f=15, g=7) si ii vom schimba pozitia in lista open astfel incat sa se pastreze sortarea corecta.
-- **open** = [((d, h=7), parinte=a, f=14, g=7),
((e, h=8), parinte=b, f=15, g=7),
((g, h=14), parinte=c, f=29, g=15),
((f, h=0), parinte=b, f=103, g=103)]
-- **closed** = [((a, h=inf), parinte=None, f=inf, g=0),
((c, h=3), parinte=a, f=12, g=9),
((b, h=10), parinte=a, f=13, g=3)]

Pas 5:
Mutam nodul „d” din open in closed. Nodul „d” nu este nod_scop, deci il extindem (fiul „i” se adauga in open, respectand sortarea).
-- **open** = [((i, h=3), parinte=d, f=14, g=11),
((e, h=8), parinte=b, f=15, g=7),
((g, h=14), parinte=c, f=29, g=15),
((f, h=0), parinte=b, f=103, g=103)]
-- **closed** = [((a, h=inf), parinte=None, f=inf, g=0),
((c, h=3), parinte=a, f=12, g=9),
((b, h=10), parinte=a, f=13, g=3),
((d, h=7), parinte=a, f=14, g=7)]



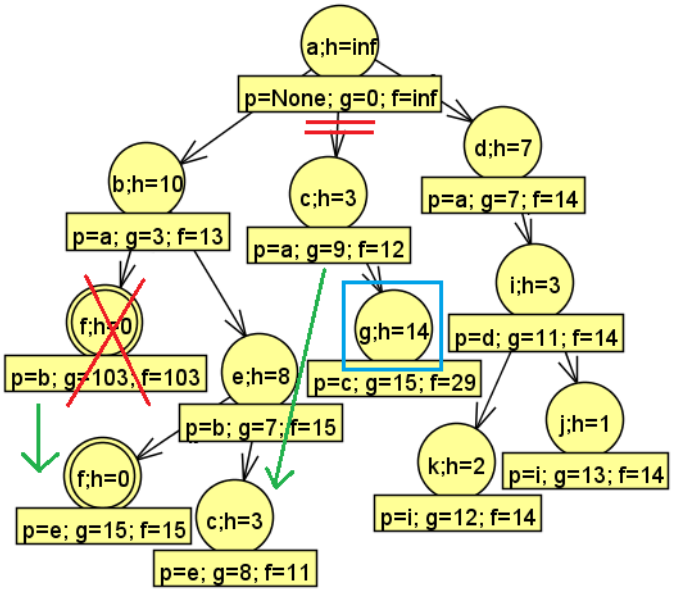
Pas 6:
Mutam nodul „i” din open in closed. Nodul „i” nu este nod_scop, deci il extindem (fiii „k” si „j” se adauga in open, respectand sortarea). Observam ca f(„k”) = f(„j”) = 14, de aceea se aplica ***al doilea criteriu de sortare*** si nodul „j” se pune inaintea nodului „k” (pentru ca g(„j”)=13 > g(„k”)=12).
-- **open** = [((j, h=1), parinte=i, f=14, g=13),
((k, h=2), parinte=i, f=14, g=12),
((e, h=8), parinte=b, f=15, g=7),
((g, h=14), parinte=c, f=29, g=15),
((f, h=0), parinte=b, f=103, g=103)]
-- **closed** = [((a, h=inf), parinte=None, f=inf, g=0),
((c, h=3), parinte=a, f=12, g=9),
((b, h=10), parinte=a, f=13, g=3),
((d, h=7), parinte=a, f=14, g=7),
((i, h=3), parinte=d, f=14, g=11)]



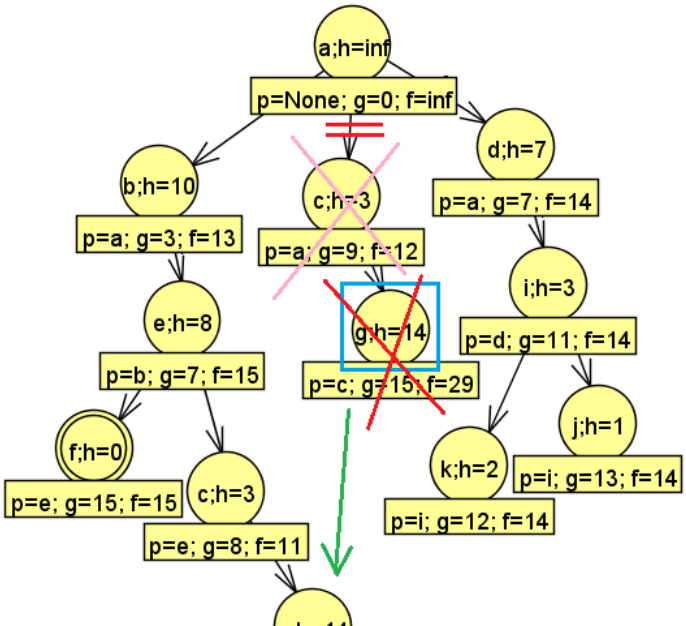
Pas 7:
Mutam nodul „j” din open in closed. Nodul „j” nu este nod_scop, deci incercam sa il extindem, dar nu are niciun fiu.
-- **open** = [((k, h=2), parinte=i, f=14, g=12), ((e, h=8), parinte=b, f=15, g=7),
((g, h=14), parinte=c, f=29, g=15), ((f, h=0), parinte=b, f=103, g=103)]
-- **closed** = [((a, h=inf), parinte=None, f=inf, g=0), ((c, h=3), parinte=a, f=12, g=9),
((b, h=10), parinte=a, f=13, g=3), ((d, h=7), parinte=a, f=14, g=7),
((i, h=3), parinte=d, f=14, g=11), ((j, h=1), parinte=i, f=14, g=13)]

Pas 8:
Mutam nodul „k” din open in closed. Nodul „k” nu este nod_scop, deci incercam sa il extindem, dar nu are niciun fiu.
-- **open** = [((e, h=8), parinte=b, f=15, g=7),
((g, h=14), parinte=c, f=29, g=15), ((f, h=0), parinte=b, f=103, g=103)]
-- **closed** = [((a, h=inf), parinte=None, f=inf, g=0), ((c, h=3), parinte=a, f=12, g=9),
((b, h=10), parinte=a, f=13, g=3), ((d, h=7), parinte=a, f=14, g=7),
((i, h=3), parinte=d, f=14, g=11), ((j, h=1), parinte=i, f=14, g=13),
((k, h=2), parinte=i, f=14, g=12)]

Pas 9:
Mutam nodul „e” din open in closed. Nodul „e” nu este nod_scop, deci il extindem (fiii „f” si „c”).
 Nodul „f” ***se afla deja in lista open*** (cu parinte=b, f=103, g=103), dar pentru ca f-ul calculat pe drumul gasit acum este mai bun (f=15 < 103) vom actualiza informatiile pentru nodul „f” (parinte=e, f=15, g=15) si ii vom schimba pozitia in lista open astfel incat sa se pastreze sortarea corecta (nodul „f” se muta inaintea nodului „g”).
 Nodul „c” ***se afla deja in lista closed*** (cu parinte=a, f=12, g=9), dar pentru ca f-ul calculat pe drumul gasit acum este mai bun (f=11 < 12) vom actualiza informatiile pentru nodul „c” (parinte=e, f=11, g=8) si il *vom muta inapoi din closed in open* (pentru ca la pasii urmatori, cand nodul „c” *va fi extins din nou*, informatiile tuturor descendentilor sai (nodul „g”) se vor putea actualiza tinand cont de acest drum mai bun gasit pana in nodul „c”). Nodul „c” a fost inserat in open la pozitia corecta conform sortarii (este primul din open).



-- **open** = [((c, h=3), parinte=e, f=11, g=8),
((f, h=0), parinte=e, f=15, g=15),
((g, h=14), parinte=c, f=29, g=15)]
-- **closed** =
[((a, h=inf), parinte=None, f=inf, g=0),
((b, h=10), parinte=a, f=13, g=3),
((d, h=7), parinte=a, f=14, g=7),
((i, h=3), parinte=d, f=14, g=11),
((j, h=1), parinte=i, f=14, g=13),
((k, h=2), parinte=i, f=14, g=12),
((e, h=8), parinte=b, f=15, g=7)]



Pas 10:

Mutam nodul „c” din open in closed. Nodul „c” nu este nod_scop, deci il extindem (fiii „e” si „g”).
 Nodul „e” ***face parte din drumul de la radacina la tatal sau*** („a”, „b”, „e”, „c”), asa ca *nu vom extinde acest drum*, pentru *a nu crea un circuit* in arbore.
 Nodul „g” ***se afla deja in lista open*** (cu parinte=c, f=29, g=15), dar pentru ca f-ul calculat pe drumul gasit acum este mai bun (f=28 < 29) vom actualiza informatiile pentru nodul „g” (parinte=c, f=28, g=14) si ii vom schimba pozitia in lista open astfel incat sa se pastreze sortarea corecta (de fapt va ramane pe aceeasi pozitie).
-- **open** = [((f, h=0), parinte=e, f=15, g=15),
((g, h=14), parinte=c, f=28, g=14)]
-- **closed** =
[((a, h=inf), parinte=None, f=inf, g=0),
((b, h=10), parinte=a, f=13, g=3),
((d, h=7), parinte=a, f=14, g=7),
((i, h=3), parinte=d, f=14, g=11),
((j, h=1), parinte=i, f=14, g=13),
((k, h=2), parinte=i, f=14, g=12),
((e, h=8), parinte=b, f=15, g=7),
((c, h=3), parinte=e, f=11, g=8)]

Pas 11:
Mutam nodul „f” din open in closed. Nodul „f” ***este nod_scop***, deci *oprim cautarea* si *afisam drumul* de la radacina (nod_start „a”) la nodul scop „f” (parcurgandu-l in sens invers cu ajutorul proprietatii „parinte”):
 drumul de cost minim este („a”, „b”, „e”, „f”) cu costul f=15.
Drum_cost_min = [((a, h=inf), parinte=None, f=inf, g=0), ((b, h=10), parinte=a, f=13, g=3),
((e, h=8), parinte=b, f=15, g=7), ((f, h=0), parinte=e, f=15, g=15)]