Mesterséges Intelligencia III.

gyakorlat

KIRÁLYTÚRA

Készítette:

Szathmáry László
IV. PTM szakos hallgató

Feladat: egy általam tetszőlegesen kiválasztott feladat implementálása a Prolog programozási nyelv segítségével.

Királytúra

*Adott egy N*M-es tábla. A tábla egy tetszőleges mezőjébe egy királyt helyezünk. A feladat a tábla összes mezőjének a bejárása úgy, hogy minden mezőt csak egyszer érinthetünk, és a király bármely szomszédos mezőre léphet a sakk szabályainak megfelelően.*

A klasszikus eset a 3*3-as tábla lenne, de annyiban módosítottam a feladaton, hogy tetszőleges „téglalap" esetén is ki lehessen próbálni.
A feladatot úgy oldottam meg, hogy a program a lehetséges megoldások közül az optimálisat keresse meg. Ez azt jelenti, hogy az egyes lépésekhez költséget rendelünk:

```
balra_k(2).
fel_balra_k(1).
fel_k(4).
fel_jobbra_k(2).
jobbra_k(3).
le_jobbra_k(2).
le_k(4).
le_balra_k(1).
```

A tábla mezőit egyértelműen azonosítani kell tudni ⇒ a mezőkhöz egy-egy betűt fogunk rendelni:

```
   0 1 2
  +-+-+-+
 0|a|b|c|
  +-+-+-+
 1|d|e|f|
  +-+-+-+
 2|g|h|i|
  +-+-+-+
```

Ezt Prolog-ban a következőképpen tehetjük meg:

```
mezo([0,0],a).
mezo([0,1],b).
mezo([0,2],c).
mezo([1,0],d).
mezo([1,1],e).
mezo([1,2],f).
mezo([2,0],g).
mezo([2,1],h).
mezo([2,2],i).
```

A „sakktábla" tetszőleges téglalap lehet:

```
meret_x(2).                    % fuggoleges iranyban
meret_y(2).                    % vizszintes iranyban
```

A könnyebb áttekinthetőség kedvéért egy 2*2-es táblával fogunk dolgozni.

Szükségünk lesz egy „globális változóra” is (erről majd később):

```
tempGY([]).
```

A program működésének áttekintése:

Optimális keresést akarunk végrehajtani, s ehhez egy keresőgráfot fogunk előállítani. A keresőgráfban csúcsok vannak, melyek zártak vagy nyíltak lehetnek. Zárt: közbenső csúcs, már túlhaladtunk rajtuk; nyílt: levélelem, itt folytatjuk majd a keresést.
A gráf elemeit egy listában fogom tárolni. Egy csúcs egy „objektum”, melyet pl. rekord segítségével tudnánk leírni. Mivel Prolog-ban nincs rekord, ezért egy listába fogjuk összefogni a „rekordelemeket”, így a „gráf” egy olyan lista lesz, melynek elemei listák (rekordok).
Pl. egy listaelem a következőképpen nézhet ki:

```
[b,[a,b],3]
```

Első elem: hol van most a király, melyik mezőn.
Második elem: a kezdőpozíciótól az aktuális pozícióba milyen úton jutott el a király.
Harmadik elem: az eddigi út összköltsége.

A program indításkor megkérdezi a felhasználótól, hogy az hova szeretné tenni a királyt. Ha az „a” mezőre tesszük, akkor a keresőgráf iniciális állapota így fog kinézni:

```
[[a,[a],0]]
```

Vagyis a gráfban (külső lista) eleddig egyetlen csúcs van.

A keresőalgoritmus működése:

A nyílt csúcsok közül választani kell, hogy melyiket terjesszük ki. Optimális keresésről lévén szó, az eddig legkisebb költségű csúcsot terjesztjük ki, ott reméljük a legolcsóbb megoldást. A kiterjesztés művelete azt jelenti, hogy az adott csúcsból minden lehetséges irányba továbblépünk. Itt több dolgot is regisztrálni kell: az új csúcsok esetén meg kell határozni az új költséget; az új csúcs esetén kell egy visszamutató az ősre, hogy majd a megoldást rekonstruálni tudjuk (erre szolgál a „rekord” 2. eleme, ami az eddig megtett utat rögzíti); az új csúcsokat fel kell venni az adatbázisba (azaz a keresőgráfba), nyílttá kell őket tenni, míg a kiterjesztett csúcsot zárttá kell tenni (ezt törölni fogjuk).
Ha egyetlenegy nyílt csúcs sincs, amelyet ki tudnánk terjeszteni: sikertelen vég.
A kiterjesztésre kiválasztott csúcsot teszteljük, hogy célcsúcs-e. Ha az, akkor sikeres vég: a megtett út ott van a második elemben, míg a harmadik eleme a „rekordnak” az összköltséget tartalmazza.

Ugyanez tehát Prolog-ban:

```
start(Csucsok) :-
  ures(Csucsok),
  write('Nem letezik megoldasa a feladatnak.'), nl.
```

Azaz ha nincs kiterjeszthető nyílt csúcs: sikertelen vég. Különben:

```
start(Csucsok) :-
  nem_ures(Csucsok),
  minelem(Csucsok,Min),
  teszt_minelem(Min),
  kiterjeszt(Min),
  tempGY(GY), retract(tempGY(_)), asserta(tempGY([])),
  osszefuz(Csucsok,GY,Temp),
  torol(Min,Temp,UjCsucsok),
  start(UjCsucsok).
```

Legelőször az iniciális állapottal hívjuk meg (az előző példánál maradva tehát [[a,[a],0]] – val). Megkeressük a minimális elemet, kiterjesztjük (a kiterjesztés során több új csúcsot is találhatunk, ezeket fogjuk a globális változóban összegyűjteni, amit a GY nevű változóba lekérdezünk. Miután GY-ben benne vannak az új csúcsok, ezt a globális változót „kinullázzuk". Az új csúcsokat felvesszük, a kiterjesztett csúcsot töröljük, majd kezdjük az egészet elölről.

Szükség van néhány listakezelő műveletre is:

```
nem_eleme(X,Y) :- not(eleme(X,Y)).

eleme(X,[X|Farok]).
eleme(X,[Fej|Farok]) :- eleme(X,Farok).

minelem([X],X).
minelem([[Poz,Ut,Koltseg]|S], Z) :- minelem(S, [MinA,MinB,MinK]),
      Koltseg =< MinK, Z = [Poz,Ut,Koltseg].
      minelem([[Poz,Ut,Koltseg]|S], Z) :- minelem(S, [MinA,MinB,MinK]),
      MinK < Koltseg, Z = [MinA,MinB,MinK].

nem_ures(L) :- not(ures(L)).

ures([]).

hossz([], 0).
hossz([X|T], N) :- hossz(T,Z), N is Z+1.

osszefuz([],Ys,Ys).
osszefuz([X|Xs],Ys,[X|Zs]) :- osszefuz(Xs,Ys,Zs).

torol(X,[X|Xs],Xs).
torol(X,[Y|Ys],[Y|Zs]) :- torol(X,Ys,Zs).

%--- csucs tesztelese, miszerint vegallapot-e ----------------

nem_vegallapot(Csucs) :- not(vegallapot(Csucs)).

vegallapot([Poz,Ut,Koltseg]) :-
  hossz(Ut,UtHossz), meret_x(MX), meret_y(MY), N is MX*MY, UtHossz = N.

teszt_minelem(Csucs) :-
  nem_vegallapot(Csucs).

teszt_minelem([Poz,Ut,Koltseg]) :-
  vegallapot([Poz,Ut,Koltseg]),
  write('Megtalaltam az optimalis megoldast: '), write(Ut), nl,
  write('Ennek osszkoltsege: '), write(Koltseg), nl,
  fail.
```

Egy csúcs kiterjesztése úgy történik, hogy minden lehetséges irányba megpróbálunk továbblépni. Az előfeltételek mondják meg, hogy egy bizonyos irányba léphetünk-e vagy sem. Itt a tábla szélét kell figyelni (nem léphetünk le róla elvégre), ill. egy mezőt csak egyszer érinthetünk! Ha valamelyik irányba „szabad a pálya”, akkor azt az új csúcsot felvesszük az adatbázisba. A `fail` azért kell mindegyik végére, mert minden irányba végig kell próbálni, azaz nem állhatunk meg, ha valamelyik irányba sikerült ⇒ tovább kell próbálkozni.

```
kiterjeszt(Mit) :-
  balra_ef(Mit,UjPoz,LepesKoltseg),
  beszur(Mit,UjPoz,LepesKoltseg),
  fail.

kiterjeszt(Mit) :-
  fel_balra_ef(Mit,UjPoz,LepesKoltseg),
  beszur(Mit,UjPoz,LepesKoltseg),
  fail.

kiterjeszt(Mit) :-
  fel_ef(Mit,UjPoz,LepesKoltseg),
  beszur(Mit,UjPoz,LepesKoltseg),
  fail.

kiterjeszt(Mit) :-
  fel_jobbra_ef(Mit,UjPoz,LepesKoltseg),
  beszur(Mit,UjPoz,LepesKoltseg),
  fail.

kiterjeszt(Mit) :-
  jobbra_ef(Mit,UjPoz,LepesKoltseg),
  beszur(Mit,UjPoz,LepesKoltseg),
  fail.

kiterjeszt(Mit) :-
  le_jobbra_ef(Mit,UjPoz,LepesKoltseg),
  beszur(Mit,UjPoz,LepesKoltseg),
  fail.

kiterjeszt(Mit) :-
  le_ef(Mit,UjPoz,LepesKoltseg),
  beszur(Mit,UjPoz,LepesKoltseg),
  fail.

kiterjeszt(Mit) :-
  le_balra_ef(Mit,UjPoz,LepesKoltseg),
  beszur(Mit,UjPoz,LepesKoltseg),
  fail.

kiterjeszt(_).
```

Ez az utolsó sor azért felelős, hogy ha már minden irányt végigpróbáltunk, akkor sikerrel visszatérjünk (ez ui. mindig igaz lesz).

```
beszur([Poz,Ut,Koltseg],UjPoz,LepesK) :-
  tempGY(GY),
  osszefuz(Ut,[UjPoz],UjUt),
  UjKoltseg is Koltseg + LepesK,
  osszefuz(GY, [[UjPoz,UjUt,UjKoltseg]], UjGY),
  retract(tempGY(_)), asserta(tempGY(UjGY)).
```

A globális változóban gyűlnek az új csúcsok, ahogy minden irányba próbálkozunk. Először üres, majd ahogy megyünk végig a 8 irányon, s találunk egy új csúcsot, akkor azt felvesszük ide. Azaz lekérdezzük az értékét GY-be, GY-hez hozzáfűzzük az új csúcsot, majd ezt az új GY-t „visszaírjuk”.

Az egyes előfeltételek:

```
balra_ef([Poz,Ut,Koltseg],UjPoz,LepesK) :-
  mezo([X,Y],Poz),
  Ym1 is Y-1,
  mezo([X,Ym1],UjPoz),
  Y >= 1,
  nem_eleme(UjPoz,Ut),
  balra_k(LepesK).

fel_balra_ef([Poz,Ut,Koltseg],UjPoz,LepesK) :-
  mezo([X,Y],Poz),
  Xm1 is X-1, Ym1 is Y-1,
  mezo([Xm1,Ym1],UjPoz),
  X>=1, Y>=1,
  nem_eleme(UjPoz,Ut),
  fel_balra_k(LepesK).

fel_ef([Poz,Ut,Koltseg],UjPoz,LepesK) :-
  mezo([X,Y],Poz),
  Xm1 is X-1,
  mezo([Xm1,Y],UjPoz),
  X>=1,
  nem_eleme(UjPoz,Ut),
  fel_k(LepesK).

fel_jobbra_ef([Poz,Ut,Koltseg],UjPoz,LepesK) :-
  mezo([X,Y],Poz),
  Xm1 is X-1, Yp1 is Y+1,
  meret_y(MY),
  mezo([Xm1,Yp1],UjPoz),
  X>=1, Y=<MY-2,
  nem_eleme(UjPoz,Ut),
  fel_jobbra_k(LepesK).

jobbra_ef([Poz,Ut,Koltseg],UjPoz,LepesK) :-
  mezo([X,Y],Poz),
  Yp1 is Y+1,
  meret_y(MY),
  mezo([X,Yp1],UjPoz),
  Y=<MY-2,
  nem_eleme(UjPoz,Ut),
  jobbra_k(LepesK).
```

```
le_jobbra_ef([Poz,Ut,Koltseg],UjPoz,LepesK) :-
  mezo([X,Y],Poz),
  Xp1 is X+1, Yp1 is Y+1,
  meret_x(MX), meret_y(MY),
  mezo([Xp1,Yp1],UjPoz),
  X=<MX-2, Y=<MY-2,
  nem_eleme(UjPoz,Ut),
  le_jobbra_k(LepesK).

le_ef([Poz,Ut,Koltseg],UjPoz,LepesK) :-
  mezo([X,Y],Poz),
  Xp1 is X+1,
  meret_x(MX),
  mezo([Xp1,Y],UjPoz),
  X=<MX-2,
  nem_eleme(UjPoz,Ut),
  le_k(LepesK).

le_balra_ef([Poz,Ut,Koltseg],UjPoz,LepesK) :-
  mezo([X,Y],Poz),
  Xp1 is X+1, Ym1 is Y-1,
  meret_x(MX),
  mezo([Xp1,Ym1],UjPoz),
  X=<MX-2, Y>=1,
  nem_eleme(UjPoz,Ut),
  le_balra_k(LepesK).
```

Azaz: ne menjünk le a tábláról, s ne lépjünk olyan mezőre, ahol már jártunk (ez az Ut-ban van letárolva).

A program indítása a következő „eljárás” segítségével történik:

```
main :-
  write('Mesterseges Intelligencia III. - gyakorlat'), nl,
  write('KIRALYTURA - keszitette Szathmary Laszlo'), nl,
  nl,
  write('A sakktabla:'), nl,
  nl,
  write('    0 1 2'), nl,
  write('   +-+-+-+'), nl,
  write('  0|a|b|c|'), nl,
  write('   +-+-+-+'), nl,
  write('  1|d|e|f|'), nl,
  write('   +-+-+-+'), nl,
  write('  2|g|h|i|'), nl,
  write('   +-+-+-+'), nl,
  nl,
  meret_x(Fuggoleges), meret_y(Vizszintes),
  write('Ebbol mi most vizszintes iranyban '), write(Vizszintes),
  write(' mezot,'), nl,
  write('fuggoleges iranyban pedig '), write(Fuggoleges),
  write(' mezot fogunk felhasznalni.'), nl,
  nl,
  write('Melyik mezorol indul a kiraly? (egy betut kerek): '), read(Mezo),
  ellenoriz(Mezo), !,
  start([[Mezo,[Mezo],0]]).
```

```
ellenoriz(Betu) :-
  mezo([X,Y],Betu),
  meret_x(XM), meret_y(YM),
  X<XM, Y<YM.

ellenoriz(Betu) :-
  write('Hiba: a '''), write(Betu),
  write(''' mezo kivul esik a megengedett teruleten!'), nl,
  fail.
```

Mi most speciel egy 2*2-es táblát adtunk meg. A felhasználótól bekérjük, hogy hová szeretné helyezni a királyt. Ha rossz adatot ad meg (pl. 2*2-es táblán kívüli mezőt), akkor azt nem szabad elfogadni: erre szolgál az `ellenoriz(Mezo).`

Futási eredmény:

```
A sakktabla:

    0 1 2
   +-+-+-+
  0|a|b|c|
   +-+-+-+
  1|d|e|f|
   +-+-+-+
  2|g|h|i|
   +-+-+-+

Ebbol mi most vizszintes iranyban 2 mezot,
fuggoleges iranyban pedig 2 mezot fogunk felhasznalni.

Melyik mezorol indul a kiraly? (egy betut kerek): a.
Megtalaltam az optimalis megoldast: [a,e,d,b]
Ennek osszkoltsege: 6
```

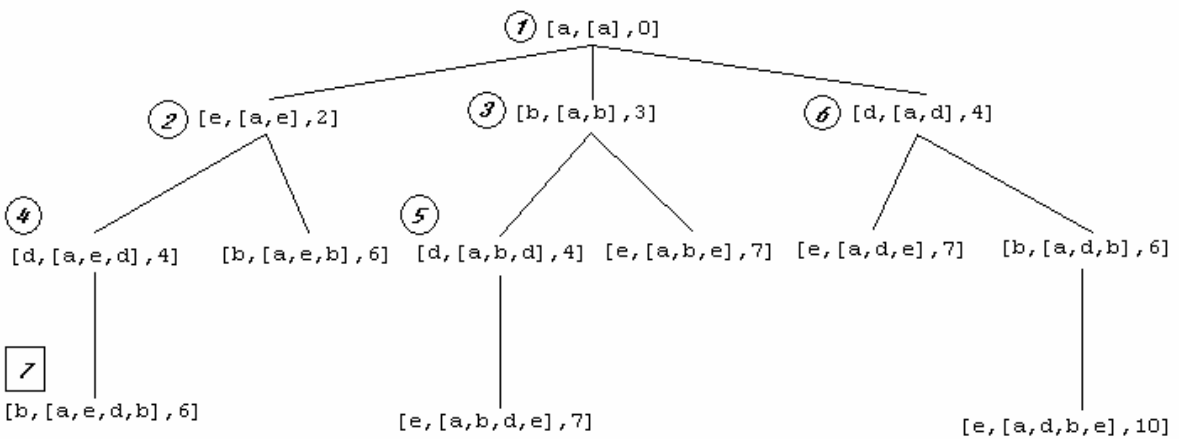
Egy hibás mező esetén:

```
Melyik mezorol indul a kiraly? (egy betut kerek): c.
Hiba: a 'c' mezo kivul esik a megengedett teruleten!
```

A következő ábra a keresőgráf növekedését mutatja az előző példa alapján. Egy-egy csúcs a következő információt hordozza:

- király jelenlegi pozíciója
- az eddig megtett út
- az eddig megtett út költsége

A csúcsok melletti bekarikázott szám a kiterjesztések sorrendjét tükrözi. A 7-essel jelölt csúcs célcsúcs, ez lesz a megoldás.


1 [a,[a],0]
2 [e,[a,e],2]
3 [b,[a,b],3]
6 [d,[a,d],4]
4
[d,[a,e,d],4]
[b,[a,e,b],6]
5
[d,[a,b,d],4]
[e,[a,b,e],7]
[e,[a,d,e],7]
[b,[a,d,b],6]
7
[b,[a,e,d,b],6]
[e,[a,b,d,e],7]
[e,[a,d,b,e],10]