Laborator 7

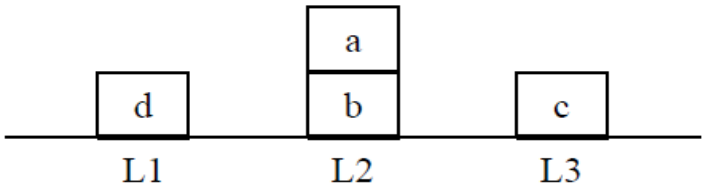
Problema Mutarii Blocurilor

Sa consideram ca avem la dispozitie M blocuri, depozitate pe un teren. Pe suprafata terenului exista un numar de N locatii de depozitare. Pentru o mai buna utilizare a spatiului, blocurile pot fi asezate unele peste altele, in stive. In fiecare locatie de depozitare exista cate o astfel de stiva, eventual vida. Un bloc poate fi mutat din locul sau numai daca el se afla in varful unei stive, si poate fi asezat numai deasupra unei alte stive (eventual vide).

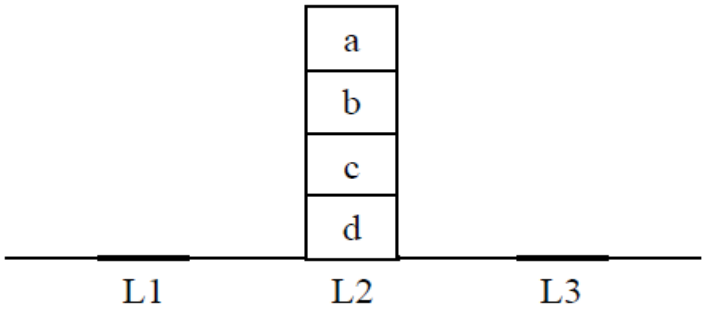
Pentru doua configuratii date C1, C2 ale asezarii celor M blocuri in cele N locatii, sa se tabileasca daca si cum este posibil sa se ajunga in configuratia C2, plecand din configuratia C1.

Exemplu: M = 4, N = 3;

C1:



C2:



In Prolog, reprezentam o stare a problemei (o configuratie) prin intermediul unei liste de liste C. Fiecare lista din C corespunde unei stive de blocuri si este ordonata astfel incat blocul din varful stivei corespunde capului listei. Stivele vide sunt reprezentate prin liste vide. In cazul exemplului anterior, termenii Prolog corespunzatori lui C1, respectiv C2 sunt: [[d],[a,b],[c]] si [[ ],[a,b,c,d],[ ]].

```
% Cautare de tip breadth-first
rezolva_b(NodInitial,Solutie):- breadthfirst([[NodInitial]],Solutie).
breadthfirst([[Nod|Drum]|_],[Nod|Drum]):-scop(Nod).
breadthfirst([Drum|Drumuri],Solutie):- extinde(Drum,DrumNoi),
                                       concat(Drumuri,DrumNoi,Drumuri1),
                                       breadthfirst(Drumuri1,Solutie).
extinde([Nod|Drum],DrumNoi):- bagof([NodNou,Nod|Drum],
                                    (s(Nod,NodNou),
                                    \+ (membru(NodNou,[Nod|Drum]))), DrumNoi),!.
extinde(_,[]).
%Cautare de tip depth-first cu mecanism de detectare
%ciclurilor
rezolva(Nod,Solutie):-depthfirst([],Nod,Solutie).
```

```
depthfirst(Drum, Nod,[Nod|Drum]):-scop(Nod).
depthfirst(Drum,Nod,Solution):-  s(Nod,Nod1),\+ (membru(Nod1,Drum)),
                                    depthfirst([Nod|Drum],Nod1,Solution).
%Cautare de tip iterative deepening
cale(Nod,Nod,[Nod]).
cale(PrimNod,UltimNod,[UltimNod|Drum]):- cale(PrimNod,PenultimNod,Drum),
                                          s(PenultimNod,UltimNod),
                                          \+(membru(UltimNod,Drum)).
depth_first_iterative_deepening(Nod,Solutie):- cale(Nod,NodScop,Solutie),
                                          scop(NodScop),!.

 %Predicatele specifice problemei mutarii blocurilor
 s(Lista_stive,Lista_stive_rez):- membru(X,Lista_stive),X=[Varf|_],
                                  det_poz_el(Lista_stive,N,X),
                                  sterg_la_n(Lista_stive,Lista_stive_inter,N),
                                  membru(Y,Lista_stive),det_poz_el(Lista_stive,N1,Y),N1\==N,
                                  adaug_la_n(Varf,Lista_stive_inter,Lista_stive_rez,N1).
 initial([[d],[a,b],[c]]).
 scop([[],[a,b,c,d],[]]).

 pb_bf:-tell('C:\\bloc_mut_ies_bf.txt'), initial(S),rezolva_b(S,Solutie),afisare(Solutie),told.
 pb_df:-tell('C:\\bloc_mut_ies_df.txt'), initial(S),rezolva(S,Solutie),afisare(Solutie),told.
```

pb_df_id:-tell('C:\\block_mut_ies_df_id_.txt'), initial(S),
depth_first_iterative_deepening(S,Solutie),
afisare(Solutie),told.

Interogarea Prologului se face apeland predicatele **pb_bf daca se doreste o cautare de tip** breadth-first, **pb_df pentru o cautare de tip depth-first cu mecanism de detectare a** ciclurilor si, respectiv **pb_df_id pentru o cautare de tip iterative deepening. Iata un** exemplu de executie:
?- pb_df_id.

% Predicatul sterg_la_n(Lista_stive,N,X) este folosit pentru stergerea capului stivei de pe pozitia N din lista de stive Lista_stive.
% Predicatul adaug_la_n(Lista_stive,N,X) este folosit pentru adaugarea elementului X in capului stivei din lista de stive Lista_stive, de pe pozitia N.
% Predicatul rezolva_b(NodInitial,Solutie) este adevarat daca Solutie este un drum (in ordine inversa) de la nodul NodInitial la o stare scop, drum obtinut folosind cautarea de tip breadth-first.
% Predicatul breadthfirst(Drumuri,Solutie) este adevarat daca un drum din multimea de drumuri candidate numita "Drumuri" poate fi extins la o stare scop; Solutie este un asemenea drum.
% Predicatul rezolva(Nod,Solutie) este adevarat daca Solutie este un drum aciclic (in ordine inversa) intre nodul Nod si o stare scop.

% Predicatul depthfirst(Drum,Nod,Solutie) este adevarat daca, extinzand calea [Nod|Drum] catre o stare scop, obtinem drumul Solutie. Semnificatia argumentelor sale este: Nod este o stare de la care trebuie gasita o cale catre o stare scop, Drum este o cale (o lista de noduri) intre starea initiala si Nod, Solutie este Drum extins via Nod catre o stare scop.

% Predicatul cale(Nod1,Nod2,Drum) este adevarat daca Drum este o cale aciclica intre nodurile Nod1 si Nod2, in spatiul starilor; calea Drum este reprezentata ca o lista de noduri in ordine inversa.

% Pentru afisare, determinam mai intai cea mai mare lungime a unei stive din L si afisam nivel cu nivel,incepand de la ultimul, blocurile din stivele aflate in lista L.

Pentru bf, rezulta:

```
 a
d  b  c
==========

  a  d
  b  c
==========
```

```
    d
a  b  c
==========

b    d
a    c
==========
```

```
b
a  d  c
==========

b  c
a  d
==========
```

```
  b
  c
a  d
==========

  a
  b
  c
  d
==========
```

OBS!

tab(+N)

tab(+Stream,+N)

N spaces are output onto Stream. N may be an arithmetic expression.