```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Male ulohy Prolog %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% 1. SPIRALA MATICE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Matici m*a muzeme reprezentovat jako seznam (delky m) radkovych seznamu
% (delek a) jejich prvku. Sestavte predikat otoc(+Mat,-OtMat), ktery otoci
% takto reprezentovanou matici o 90 stupnu proti smeru hodinovych rucicek.
% Na jeho zaklade sestavte predikat spirala(+Mat,-Sez), ktery "oloupa seznam
% z matice Mat do seznamu Sez". Postupujte pitom z leveho horniho rohu
% po smeru hodinovych rucicek.

% otoc(+Mat,-OtMat)
otoc(Mat,OtMat):-
otocAk(Mat,[],OtMat).

% getLine(+Mat,-Line,-NewMat)
getLine([],[],[]).
getLine([[Head|Tail]|MatTail],[Head|TailLine],[Tail|NewMatTail]):-
getLine(MatTail,TailLine,NewMatTail).

% otocAk(+Mat,+Ak,OtMat)
otocAk([],Ak,Ak).
otocAk([[]|_],Ak,Ak).
otocAk(Mat1,Ak,OtMat):-
getLine(Mat1,Line,NewMat1),
append([Line],Ak,NewAk),
otocAk(NewMat1,NewAk,OtMat).


% spirala(+Mat,-Sez)
spirala(Mat,Sez):-
spiralaAk(Mat,[],Sez).

% spiralaAk(+Mat,+Ak,-Sez)
spiralaAk([],Ak,Ak).
spiralaAk([Head|Tail],Ak,Sez):-
append(Ak,Head,NewAk),
otoc(Tail,NewMat),
spiralaAk(NewMat,NewAk,Sez).


% testy

sm_t1:-otoc([[1,2],[3,4],[5,6]],X), write(X).
sm_t2:-otoc([[1,2,3,4,5],[a,b,c,d,e],[x1,x2,x3,x4,x5]],X), write(X).
sm_t3:-spirala([[1,2,3],[4,5,6],[7,8,9]],X), write(X).
sm_t4:-spirala([[1,2,3,4],[5,6,7,8],[9,a,b,c]],X), write(X).
sm_t5:-spirala([[1,2,3],[4,5,6],[7,8,9],[a,b,c],[d,e,f]],X), write(X).

%% 2. TAUTOLOGY CHECKER %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Sestavte predikat taut(+Formule), ktery pokud je Formule spravne utvorena
% formule vyrokovï¿½o poctu:
% [spojky unarni: ~ (negace),
%  spojky binarni: & (konjunkce), # (disjunkce), => (implikace)
%  s obvyklymi prioritami, zavorky pro zmenu poradi vyhodnocovani,
%  vyrokove promenne - mala pismena]
% uspeje prave kdyz je tato formule tautologii (tautologie je formule, ktera
% nabyva hodnoty true nezavisle na ohodnoceni elementarnich formuli, ktere
% obsahuje).
% Napiste i predikaty, ktere zajisti pouziti prislusnych spojek jako
% operatoru.

% reseni s promennymi jako malymi pismenky nezname
% naznak reseni (=nefunkcni) s velkymi pismenky nasleduje
% pod nim v komentari jeste cizi tautology checker s jinou syntaxi
```

```
% taut(+Formule)
taut(Formule):-                         % formule je tautologie prave tehdy,
\+ sat(~(Formule)).                     % kdyz jeji negace neni splnitelna

sat(Formule):-                          % formule je splnitelna prave tehdy,
eval(Formule,1).                      % nejaka kombinace vede na 1

:-op(100,yfx,'=>').
:-op(200,yfx,'#').
:-op(300,yfx,'&').
:-op(400,fx,'~').

eval(V,V):-                             % vyhodnot promennou
var(V),
!,
evalute(V).

eval(V,V):-                             % ohodnot konstatny
evalute(V),
!.

eval(~E,R):-                            % vyhodnot negaci
!,
eval(E,V),
(
  (V=0,R=1)
  ;
  (V=1,R=0)
).

eval(A&B,R):-                           % ohodnot binarni operatory
!,
eval(A,Av),                           % vyhodnoceni podvyrazu nalevo
eval(B,Bv),                           % vyhodnoceni podvyrazu napravo
and(Av,Bv,R).                         % pouziti operatoru

eval(A#B,R):-                           % ohodnot binarni operatory
!,
eval(A,Av),                           % vyhodnoceni podvyrazu nalevo
eval(B,Bv),                           % vyhodnoceni podvyrazu napravo
and(Av,Bv,R).                         % pouziti operatoru

eval(A=>B,R):-                          % ohodnot binarni operatory
!,
eval(A,Av),                           % vyhodnoceni podvyrazu nalevo
eval(B,Bv),                           % vyhodnoceni podvyrazu napravo
imp(Av,Bv,R).                         % pouziti operatoru

% evalute udava mozne hodnoty promenne
evalute(0).
evalute(1).

% operatory
and(0,_,0):- !.
and(_,0,0):- !.
and(1,1,1):- !.

or(0,0,0).
or(1,_,1):-  !.
or(_,1,1):-  !.

imp(0,_,1):- !.
imp(1,V,V):- !.
```

```
% TAUTOLOGY CHECKER by Robert F. Staerk
%
% list([]).
% list([X|L]) :- list(L).
%
% member(X,[X|L]).
% member(X,[Y|L]) :- member(X,L).
%
% formula(p(X)).
% formula(neg(A)) :- formula(A).
% formula(and(A,B)) :- formula(A), formula(B).
% formula(or(A,B)) :- formula(A), formula(B).
%
% literal(p(X)).
% literal(neg(p(X))).
%
% literal_list([]).
% literal_list([A|I]) :-
%        literal(A),
%        literal_list(I).
%
% interpretation(I) :- literal_list(I), not incon(I).
%
% incon(I) :- member(p(X),I), member(neg(p(X)),I).
%
% valid(A) :- not satisfiable(neg(A)).
%
% satisfiable(A) :- true(A,[],I).
%
% true(p(X),I,[p(X)|I]) :- not member(neg(p(X)),I).
% true(neg(A),I,J) :- false(A,I,J).
% true(and(A,B),I,K) :- true(A,I,J), true(B,J,K).
% true(or(A,B),I,J) :- true(A,I,J).
% true(or(A,B),I,J) :- true(B,I,J).
%
% false(p(X),I,[neg(p(X))|I]) :- not member(p(X),I).
% false(neg(A),I,J) :- true(A,I,J).
% false(and(A,B),I,J) :- false(A,I,J).
% false(and(A,B),I,J) :- false(B,I,J).
% false(or(A,B),I,K) :- false(A,I,J), false(B,J,K).
%
% eval(p(X),I,1) :- member(p(X),I).
% eval(p(X),I,0) :- member(neg(p(X)),I).
% eval(neg(A),I,1) :- eval(A,I,0).
% eval(neg(A),I,0) :- eval(A,I,1).
% eval(and(A,B),I,1) :- eval(A,I,1), eval(B,I,1).
% eval(and(A,B),I,0) :- eval(A,I,0).
% eval(and(A,B),I,0) :- eval(B,I,0).
% eval(or(A,B),I,1) :- eval(A,I,1).
% eval(or(A,B),I,1) :- eval(B,I,1).
% eval(or(A,B),I,0) :- eval(A,I,0), eval(B,I,0).
%
% defined(p(X),I) :- member(p(X),I).
% defined(p(X),I) :- member(neg(p(X)),I).
% defined(neg(A),I) :- defined(A,I).
% defined(and(A,B),I) :- defined(A,I), defined(B,I).
% defined(or(A,B),I) :- defined(A,I), defined(B,I).

%% 3. NEZAVISLA MNOZINA %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Je dan graf G=(V,E). Najdete libovolnou maximalni nezavislou mnozinu W.
% Nezavisla mnozina je takova podmnozina V, ze zadne jeji dva vrcholy nejsou
% spojeny hranou. Pozadavek maximality se nevztahuje na G, ale na W - tj.
% W nelze zvetsit, ale v G muze existovat jina nezavisla mnozina W', ktera je
% vetsi.
```

```
% reprezentace dat - vertex(V), edge(U,V)

% edgeG(+A,+B) - splneno, pokud hrana A-B v G
edgeG(A,B):- edge(A,B) ; edge(B,A).

% noedgeG(+A,+B) - splneno neni-li hrana A-B v G
noedge(A,B):- \+ edgeG(A,B).

% is_vertex(+Vs,+V) - splneno, pokud Vs a take [V|Vs] nezavisle mnoziny
is_vertex([],_).                        % konec rekurze
is_vertex([Head|Tail],V):-              % prvek lze pridat do nezavisle mnoziny
noedge(Head,V),                       % pokud neexistuje hrana s hlavou
is_vertex(Tail,V).                    % a neexistuje hrana s prvkem v tele

% insetG(-IS) - IS je nezavisla mnozina G
insetG(IS):-
vertex(V),                            % vyber 1. vrchol do nezavisle mnoziny
insetG([V],IS).                       % utvor nezavislou mnozinu obsahujici
                                      % tento vrchol

% insetG(+Vs,-IS) nezavisla mnozina obsahujici vrcholy Vs
insetG(Vs,IS):-
vertex(U),                            % vyber vrchol
\+ member(U,Vs),                      % ktery jeste neni ve Vs
is_vertex(Vs,U),                      % pokud lze pridat
insetG([U|Vs],IS).                    % rekurze pro vetsi mnozinu
insetG(Vs,Vs).                          % jiz neslo pridat -> napln vystupni IS

% testovaci data

vertex(a).
vertex(b).
vertex(c).
vertex(d).
vertex(e).
vertex(f).
vertex(g).
vertex(h).

edge(a,b).
edge(a,c).
edge(a,f).
edge(b,g).
edge(c,d).
edge(c,e).
edge(c,g).
edge(d,e).

% testy
is_t1:-insetG(W),write(W).

%% 4. KARTEZSKY SOUCIN GRAFU %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Jsou dany grafy G1=(V1,E1), G2=(V2,E2), vytvorte G=(V,E)=G1*G2, tedy E=V1*V2
% a ((u1,u2),(v1,v2)) in E, jestlize (u1,v1) in E1 a (u2,v2) in E2.

% priklad:
%
% G1=({a,b,c},{(a,b),(b,c)})                G2({1,2,3,4},{(1,2),(3,4)})
%
%      a                                        1 3
%      |                                        | |
%      b-c                                      2 4
%
%
```

```
% G=({a1,b1,c1,a2,b2,c2,a3,b3,c3,a4,b4,c4},{(a1,b2),(a2,b1),(b1,c2),(b2,c1),
%     (a3,b4),(a4,b3),(b3,c4),(b4,c3)})
%
%       a  * * * *
%           X   X
%       b  * * * *
%           X   X
%       c  * * * *
%
%          1 2 3 4
%

% reprezentace dat - vertexKP(G,V), edgeKP(G,U,V)

% edgeKPG(+G,?A,?B) - splneno, pokud hrana A-B v G
edgeKPG(G,A,B):- edgeKP(G,A,B) ; edgeKP(G,B,A).

% kp(+G1,+G2,-E) vrati hrany kartezskeho soucinu G1*G2, vrcholy jsou zrejme
kp(G1,G2,E):-
kp(G1,G2,[],E).

kp(G1,G2,KPE,E):-
edgeKPG(G1,U1,V1),
edgeKPG(G2,U2,V2),
\+ member(U1*U2-V1*V2,KPE),
\+ member(U1*V2-V1*U2,KPE),
append(KPE,[U1*U2-V1*V2],KPENew),
kp(G1,G2,KPENew,E).
kp(_,_,KPE,KPE).

% testovaci data

vertexKP(g1,a).
vertexKP(g1,b).
vertexKP(g1,c).
vertexKP(g2,x1).
vertexKP(g2,x2).
vertexKP(g2,x3).
vertexKP(g2,x4).

edgeKP(g1,a,b).
edgeKP(g1,b,c).
edgeKP(g2,x1,x2).
edgeKP(g2,x3,x4).

% testy
kp_t1:-kp(g1,g2,G),write(G).

%% 5. PREVOD N-ARNI -> BINARNI %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Sestavte funkci realizujici kanonickou reprezentaci obecneho stromu pomoci
% binarniho ("levy syn" = prvorozeny syn, "pravy syn" = mladsi bratr).


% reprezentace dat
%
% N-arni strom reprezentovan tN(Value,[Child]) nebo nilN
% binarni strom reprezentovan tB(Left,Value,Right) nebo nilB

% convNB(+TreeN,-TreeB) prevede N-arni strom na binarni

convNB(nilN,nilB).
convNB(TreeN,TreeB):-
convNBs([TreeN],TreeB).

convNBs([],nilB).
```

```
convNBs([tN(Value,[])],tB(nilB,Value,nilB)).

convNBs([tN(Value,Children)],tB(TreeB,Value,nilB)):-
convNBs(Children,TreeB).

convNBs([tN(Value,Children)|TNTail],tB(TreeB,Value,TreeB2)):-
convNBs(Children,TreeB),
convNBs(TNTail,TreeB2).


% testovaci stromy

treeN1(X):-
X=tN(10,
[
  tN(5,
  [
    tN(2,[]),
    tN(7,[])
  ]),
  tN(7,
  [
    tN(10,[])
  ]),
  tN(12,[]),
  tN(3,
  [
    tN(18,[]),
    tN(1,[]),
    tN(40,[])
  ])
]).

treeN2(X):-
X=tN(10,
[
  tN(5,
  [
    tN(4,[]),
    tN(1,[])
  ]),
  tN(7,[]),
  tN(2,[])
]).

% testy

cNB_t1:-treeN1(X),convNB(X,TB),write(TB).
cNB_t2:-treeN2(X),convNB(X,TB),write(TB).

%% 6. RIDKE POLYNOMY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Naprogramujte soucin ridkych polynomu reprezentovanych jako seznam dvojic
% prv(Koef,Exp), kde Koef je nenulovy koeficient u exponentu Exp.

% predpokladame rostouci seznam v reprezentaci polynomu

% insP(+Tuple,+List,-List2) zaradi Tuple do Listu na spravnou pozici
insP(T,[],[T]).
insP(prv(K1,E1),[prv(K2,E2)|Tail],[prv(K1,E1),prv(K2,E2)|Tail]):-
E1 P(i) }
% tzn. k P=[3,2,1,4] je O=[0,1,2,0].
% Sestavte:
%  a) predikat, ktery k dane permutaci udela vektor inverzi
%  b) predikat, ktery k vektoru inverzi udela permutaci
%  c) predikat, ktery urci, zda vektor je vektor permutace.
```

```
% countGreater(+X,+List,-Gr) spocte, kolik prvku v List je vetsich nez X
countGreater(_,[],0).
countGreater(X,[Head|Tail],Gr):-
countGreater(X,Tail,Gr1),
(
  (Head>X,!,Gr is Gr1+1)
  ;
  (Gr is Gr1)
).

% genList(+List,+From,-NumList) vygeneruje posloupnost prirozenych cisel
% od From delky Listu
genList([],_,[]).                          % konec rekurze
genList([_|Tail],From,[From|TailN]):-     % pridej prvek do hlavy
From1 is From+1,                         % s prvkem o jedna vetsim
genList(Tail,From1,TailN).               % se zarekurzi

% cast a)
% permVi(+P,-O) udela z permutace vektor inverzi
permVi(P,O):-
permVi(P,[],O).                          % zavolej verzi s prazdnym akumulatorem

permVi([],_,[]).                           % konec rekurze
permVi([HeadP|TailP],Ak,[Gr|TailO]):-     % do hlavy
countGreater(HeadP,Ak,Gr),               % spocti pocet vetsich prvku v Ak
permVi(TailP,[HeadP|Ak],TailO).          % a rekurze pro zbytek

% cast b)
% viPerm(+O,-P) udela z vektoru inverzi permutaci
viPerm(O,P):-
genList(O,1,List),                       % vygeneruj seznam 1..len(O)
reverse(List,ListR),                     % otoc jej
reverse(O,OR),                           % otoc vstupni vektor
viPerm(OR,ListR,[],P).                   % zavolej verzi s prazdnym akumulatorem

viPerm([],_,Ak,Ak).
viPerm([HeadO|TailO],List,Ak,P):-
nth0(HeadO,List,X),                      % najdi odpovidajici prvek
delete(List,X,ListNew),                  % vyskrtni jej ze senzamu cisel
viPerm(TailO,ListNew,[X|Ak],P).          % pridej ho k Ak a rekurze pro zbytek

% cast c)
% isPerm(+P) je splnen, pokud P je permutace
isPerm(P):-
genList(P,1,List),                       % vygeneruj cisla 1..len(P)
isPerm(P,List).                          % zavolej rozsirenou verzi

isPerm([],[]).                             % konec rekurze - uspech
isPerm([HeadP|TailP],List):-
delete(List,HeadP,ListNew),              % odeber prvni prvek permutace z Listu
isPerm(TailP,ListNew).                   % a rekurze pro zbytek

% testovaci permutace

vi_p1(X):-
X=[3,2,1,4].

vi_p2(X):-
X=[5,1,6,8,2,3,7,4].

vi_v1(X):-
X=[0,1,2,0].

vi_v2(X):-
X=[0,1,0,0,3,3,1,4].
```

```
% testy

vi_t1:-vi_p1(X),permVi(X,Y),write(Y).
vi_t2:-vi_p2(X),permVi(X,Y),write(Y).
vi_t3:-vi_v1(X),viPerm(X,Y),write(Y).
vi_t4:-vi_v2(X),viPerm(X,Y),write(Y).
vi_t5:-vi_p1(X),isPerm(X).
vi_t6:-vi_p2(X),isPerm(X).
vi_t7:-isPerm([1]).
vi_t8:-isPerm([1,2,9,8,4,1,6,3,5,7]).
vi_t9:-isPerm([2]).

%% 10. PREVOD REPREZENTACE PERMUTACE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Napiste predikat k prevodu permutace reprezentovane vektorem na reprezentaci
% cykly (napr. [1,3,2,4] -> [[1],[3,2],[4]])

% genZero(+P,List) vytvori seznam nul dlouhy len(P)
genZero([],[]).                         % konec rekurze
genZero([_|Tail],[0|TailL]):-           % pridej nulu
genZero(Tail,TailL).                   % a generuj telo

% check(+X,+List,-NewList) zaskrtne (nastavi na 1) prvek na pozici X v List
check(1,[_|Tail],[1|Tail]).
check(X,[Head|Tail1],[Head|Tail2]):-
X1 is X-1,
check(X1,Tail1,Tail2).

% rpcOne(+VP,+Pos,+List,-NewLost,+OneCyk,-Cyk) projde jeden cyklus,
% ktery obsahuje prvek na pozici Pos v permutaci VP se seznamem pruchodu
% List aktualnim cyklem OneCyk
rpcOne(_,Pos,List,List,Cyk,Cyk):-
nth1(Pos,List,1),                      % konec rekurze pokud dany prvek
!.                                     % zaskrtnut

rpcOne(VP,Pos,List,NewList,OneCyk,Cyk):-
check(Pos,List,List2),                 % zaskrtni navstiveny prvek
nth1(Pos,VP,X),                        % urci dalsi prvek
rpcOne(VP,X,List2,NewList,[X|OneCyk],Cyk).


% rpc(+VP,-Cyk) prevede vektor permutace na cykly
rpc(VP,Cyk):-
genZero(VP,ZeroList),                  % vygeneruj nulovy seznam
length(VP,VPLen),                      % zjisti delku vektoru
rpc(VP,1,VPLen,ZeroList,[],Cyk).      % a zavolej rozsirenou verzi predikatu

rpc(_,Pos,VPLen,_,Ak,Ak):-              % konec rekurze
Pos is VPLen+1.

rpc(VP,Pos,VPLen,List,Ak,Cyk):-
nth1(Pos,List,1),                      % pokud je prvek jiz hotov
!,
Pos1 is Pos+1,                         % jdi na nasledujici pozici
rpc(VP,Pos1,VPLen,List,Ak,Cyk).       % a rekurzi

rpc(VP,Pos,VPLen,List,Ak,Cyk):-
rpcOne(VP,Pos,List,NewList,[],NewCyk),% prvek neni hotov -> udelej cyklus
append(Ak,[NewCyk],NewAk),             % pridej cyklus do akumulatoru
Pos1 is Pos+1,                         % jdi na nasledujici pozici
rpc(VP,Pos1,VPLen,NewList,NewAk,Cyk). % a rekurzi

% testovaci permutace

rpc_p1(X):-
```

```
X=[1,3,2,4].

rpc_p2(X):-
X=[3,2,1,4].

rpc_p3(X):-
X=[5,1,6,8,2,3,7,4].


% testy

rpc_t1:-rpc_p1(X),rpc(X,Y),write(Y).
rpc_t2:-rpc_p2(X),rpc(X,Y),write(Y).
rpc_t3:-rpc_p3(X),rpc(X,Y),write(Y).

%% 11. POCET INVERSI %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Sestavte predikat inv(+Sez,-PocInv), ktery spocte, kolik inverzi obsahuje
% vstupni seznam cisel Sez. Inverze je dvojice A,B, kde AHead,
!,
CntNew is CntStart+1,
gtCnt(X,Tail,CntNew,Cnt).

gtCnt(X,[_|Tail],CntStart,Cnt):-
gtCnt(X,Tail,CntStart,Cnt).

% inv(+Sez,-PocInv)
inv(Sez,PocInv):-
inv(Sez,0,PocInv).

inv([],Cnt,Cnt).
inv([HeadSez|TailSez],Cnt,PocInv):-
gtCnt(HeadSez,TailSez,Cnt,CntNew),
inv(TailSez,CntNew,PocInv).


% testy

pi_t1:-inv([1,3,2],X),write(X).
pi_t2:-inv([1,2,3,4,5],X),write(X).
pi_t3:-inv([6,1,8,2,9,60,12,3],X),write(X).

%% THE END %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Prolog
=======
*  Sestavte predikáty hladiny(+Strom, -Seznam_Hladin) kde výstupní parametr je seznam
seznamu prvku na
   jednotlivých hladinách vstupního binárního stromu Strom a k nemu inverzní strom(-Strom,
+Seznam_Hladin).
   Nejedná se o binární vyhledávací strom, ale prvky na dané hladine jsou serazené (na
jejich poradí záleží).
   Takže pokud mám hladinu 3 (max. 4 prvky) [a,b,c], tak tu hladinu mohu rekonstruovat
   jako [nil a b c], [a nil b c], [a b nil c] nebo [a b c nil], nikoli jako [b a nil c].
   Pochopitelne to muže být jeden oboustranný predikát, bude-li efektivní
   (to inverzní musí vracet postupne všechny možnosti stromu)

Řešení:

   %hladiny(+- Strom, +- Hladiny)
   hladiny(nil, []).
   hladiny(tree(Levy, Hodnota, Pravy), [[Hodnota] | T]) :- hladiny(Levy, TL), hladiny(Pravy,
TP), bmerge(TL, TP, T).

   bmerge([], X, X).
   bmerge(X, [], X).
```

```
bmerge([X1|T1], [X2|T2],[X|T]) :- merge(X1, X2, X), bmerge(T1, T2, T).

merge([], X, X).
merge([X|T1], T2, [X|T]) :- merge(T1, T2, T).
```

* Definujte predikát odpov(r1,r2) dvou proměnných, který pro každé dva seznamy (přirozených čísel a znaků * a ?) r1 a r2 uspěje pokud existuje "substituce jedno císla za žolík '?' a substituce posloupnosti čísel za znak '*'" takové, že dostanete stejné seznamy. Mužete předpokládat, že v každém ze seznamů, které jsou parametry, muže být nanejvýše jedna hvězdička.

Řešení #1

```
match(?,_).
match(_,?).
match(A,A). % :- number(A). pokud je libo :)

odpov([*|A], B) :- reverse(A, RA), reverse(B, RB), odpovt(RA, RB).
odpov(A, [*|B]) :- reverse(A, RA), reverse(B, RB), odpovt(RB, RA).
odpov([A1|A], [B1|B]) :- match(A1, B1), odpov(A, B).
odpov([], []).

odpovt([], _).
odpovt(_, [*|_]).
odpovt([A1|A], [B1|B]) :- match(A1, B1), odpovt(A, B).
```

Řešení #2

```
odpov(R1,R2) :-
       cutStart(R1,R2,R1c,R2c),  % kontroluji zda se shoduji zacatky obou seznamu, dokud nenarazim na hvezdicku,
                                 % vratim zbytky obou seznamu (hvezdicku ponechavam v seznamu)
       reverse(R1c,R1cr),        % otocim seznam
       reverse(R2c,R2cr),        % otocim seznam
       cutStart(R1cr,R2cr,_,_).  % orezavam, tentokrat konce
       % cutStart bud failuje a pak seznamy nemohou byt stejne nebo nezfailuje a vratil by seznamy:
       % 1. pripad:
       % ===========
       % L1=[*,neco, neco,...]
       % L2=[neco, neco,...,*]
       % coz je pripad, kdy bychom meli vratit "true" (
       % hvezdicka v L1 se predstavuje zacatek L2 az po hvezdicku; pro hvezdicku v L2 obdobne)
       %
       % 2. pripad:
       % ===========
       % L1=[cislo/*]
       % L2=[*/cislo]
       % meli bychom vratit "true"

% match
match(X,Y):-integer(X),integer(Y),X=:=Y.
match(X,Y):-(X == '?';Y=='?'),(integer(X);integer(Y)).

% cutStart(+Sezn1,+Sezn2,-OrezanySezn1,-OrezanySezn2)
/* pokud procedura zjisti, ze se seznamy lisi ve dvou prvcich, tak nenavratne failuje */
cutStart([],[],[],[]):-!.       % oba dva seznamy jsou stejne (s prihlednuti k vyznamu '?') a neobsahuji hvezdicku
cutStart([],L2,[],L2):-!,fail.  % jeden seznam je kratsi nez druhy; fail
cutStart(L1,[],L1,[]):-!,fail.  % dtto
```

```
   cutStart([H1|T1],[H2|T2],T3,T4):-match(H1,H2),!,cutStart(T1,T2,T3,T4). % dve stejna cisla
   cutStart([H1|_],[H2|_],_,_):-integer(H1),integer(H2),H1=\=H2,!,fail.
% dve ruzna cisla; konec
   cutStart([H1|T1],[H2|T2],[H1|T1],[H2|T2]):-(H1=='*';H2=='*'),!.
% konec na hvezdicce
```

* Prevest permutaci zadanou ve tvaru "v(7,[4,3,5,6,2,7,1])" (tedy prvni cifra udava rozsah hodnot v permutaci,
zde od 1 do 7, pak nasleduje seznam, kde prvek na pozici i se zobrazi na cislo na te pozici) do zapisu ve
tvaru cyklu "c(7,[[1,4,6,7],[2,3,4]])" , tedy nepsat jednoprvkove cykly.

Řešení

```
   preved(v(L, P), c(L, CP)) :-
      ohodnot(1, P, O),
      spoj(O, CP),
      !.

   ohodnot(X, [X|T], Z) :-
      L1 is X + 1,
      ohodnot(L1, T, Z).
   ohodnot(L, [H|T], [[L,H]|Z]) :-
      L1 is L + 1,
      ohodnot(L1, T, Z).
   ohodnot(_, [], []).

   spoj([H|Z], O) :-
      last(H, Last),
      member([Last|X], Z),
      append(H,X,V),
      delete(Z, [Last|X], Z1),
      spoj([V|Z1], O).
   spoj([H|Z], [X|O]) :-
      append(X, [_], H), % odstrani posledni prvek
      spoj(Z, O).
   spoj([], []).
```

Řešení #2

Popis:
Vezmu prvni prvek permutace, hledam pro nej cyklus, vratim permutaci, kde prvky,
na kterych jsem byl, jsou 0. cyklus hledam naivne - cili vezmu dany prvek x, pridam
do seznamu, k nemu xty, ap. dokud nezjistim, ze tam ten prvek uz v cyklu je

Kód:

```
    perm_cykly(v(0,_), c(0,[])).
    perm_cykly(v(N,P), c(N,C)) :- perm(P, P, C).

    %perm(+[Permutace], +[Permutace], -[[Cyklus]])
    perm(_, [], []).
    %hledame postupne cykly pres vsechny prvky permutace
    perm(P, [HP|TP], [C|CS]) :- najdi_cyklus(HP, P, [], C, P2), !, perm(P2, TP, CS).
    %vratil se prazdny seznam, ten v cyklech nechceme
    perm(P, [_|TP], CS) :- perm(P, TP, CS).

    %najdi_cyklus(+Prvek, +[Permutace], +[SetridenySeznam], -[Cyklus], -[0Permtuce])
    najdi_cyklus(X, P, C1, C, P3) :- nty(X,P,NT,P2), vloz(NT, C1, C2), !, najdi_cyklus(NT,
P2, C2, C, P3).
    najdi_cyklus(_, P, [], [], P) :- !, fail.  % vkladali jsme 0 -> prvek uz je v min. cyklu
    najdi_cyklus(_, P, [_], [], P) :- !, fail. % identita -> prazdny seznam
    najdi_cyklus(_, P, C, C, P).               % prvek uz je v cyklu, konec

    %vloz(+Prvek, +SetridenySeznam, -SetridenySeznam), vklada prvek > 0 do usp. sezn.
    vloz(0, _, _) :- !, fail.
```

```
    vloz(X, [H|T], [H|T1]) :- X > H, !, vloz(X, T, T1).
    vloz(X, T, [X|T]).

    %nty(+N, +Seznam, -Nty, -Seznam), vrati Nty a vrati seznam, kde na Ntem prvku je 0
    nty(1, [H|T], H, [0|T]).
    nty(N, [H|T], X, [H|T1]) :- N1 is N - 1, nty(N1, T, X, T1).
```

Řešení #3

```
    % +N ... pocet permutovanych prvku
    % +Perm ... permutace, seznam, pr. [4,3,2,1] pro N = 4
    % -Cycles ... vrati: [[1, 4], [2, 3]]
    perm2cycles(N,Perm,Cycles):-
            addPos(N,Perm,PermPairs),
            place(PermPairs,CyclesPairs),
            findCycles(CyclesPairs,Cycles).

    % addPos(+N, +Perm, -Result)
    % Prevede permutaci [4,3,2,1] na [[1,4],[2,3],[3,2],[4,1]], tj.
    % mame tedy vzdy dvojici [x,f(x)]
    addPos(N,Perm,Result):-addPos(1,N,Perm,Result).
    addPos(N,N,[P],[[N,P]]).
    addPos(I,N,[H|PermTail],[[I,H]|Result]):-I1 is I+1, addPos(I1,N,PermTail,Result).

    % place(+PermPairs,-CyclesPairs)
    % Seradi za sebou dvojice [x,f(x)] timto zpusobem:
    % CyclesPairs = [ [x,f(x)], [f(x),y],[y,x], [h,i],[i,j], ...]
    % tj. vytvarime co nejdelsi posloupnosti
    place(PermPairs,CyclesPairs):-place(PermPairs,[],CyclesPairs).
    place([],L,L).
    place([[First,Second]|T],Tmp,Result):-placeWork([First,Second],Tmp,Tmp2),
place(T,Tmp2,Result).

    placeWork([F1,S1],[],[[F1,S1]]).
    placeWork([F1,S1],[[F2,S2]|T],[[F1,S1],[F2,S2]|T]):-S1=:=F2.
    placeWork([F1,S1],[[F2,S2]|T],[[F2,S2],[F1,S1]|T]):-F1=:=S2.
    placeWork([F1,S1],[[F2,S2]|T],[[F2,S2]|R]):-placeWork([F1,S1],T,R).

    % findCycles(+CyclesPairs,-Cycles)
    % pr. CyclesPairs = [ [x,f(x)], [f(x),y],[y,x], [h,i],[i,j], ...]
    % predikat nalezne cykly. Prvni z prikladu je [x,f(x)], [f(x),y],[y,x].
    findCycles([],[]).
    findCycles([[Start,Start]|RestCyclesPairs],R):-findCycles(RestCyclesPairs,R). %
odstraneni jednoprvkovych cyklu
    findCycles([[Start,Last]|TCyclesPairs],[Cyclus|R]):-Start=\=Last,
            findCycle(Start,Last,TCyclesPairs,Cyclus,RestCyclesPairs),
            findCycles(RestCyclesPairs,R).

    % findCycle(+Start,+Last,+CyclesPairs,-Cyclus,-RestOfCyclesPairs)
    % Start ... hodnota prvku, kterym jsme zacinali a ktery kdyz nalezneme
    %           znovu, tak mame cyklus
    % Last  ... hodnota prvku, na ktery chceme napojovat
    % CyclesPairs ... seznam dvojic [x,f(x)]
    findCycle(Start,Last,[[F1,S1]|RestCyclesPairs],[Last|R],Return):-Last=:=F1,
            S1=\=Start,
            findCycle(Start,S1,RestCyclesPairs,R,Return).

    findCycle(Start,Last,[[F1,S1]|RestCyclesPairs],[Last,Start],RestCyclesPairs):- % uspesny
konec cyklu
            Last=:=F1,
            S1=:=Start.
```

* Vytvorit predikat, ktery

% a) prijme 2 permutace v zapisu cyklu a vrati jejich soucin (tedy slozeni permutaci)

Řešení by Martin Všetička:

```
   soucin(c(N,C1), c(N,C2), c(N,C3)):-      % soucin dvou permutaci
      sezn(N,1,Res1),
      sloz(N,C1,C2,Res1,P_tmp),
      perm_cykly(v(N,P_tmp),c(N,C3)).

   sloz(_,[],_,Res1,Res1).                  % zpracuje v kazdem kroku jeden cyklus prvni
permutace do vysledne permutace
   sloz(N,[[H|T]|Z],C2,Rs1,Rs):-
      zarad(H,[H|T],C2,Rs1,Rs2),
      sloz(N,Z,C2,Rs2,Rs).

   % v prvni permutaci se H1 zobrazi na Start
   zarad(Start,[H],C2,Rs1,Rs2):- fnd(Start, C2, X), umisti(1,H, X, Rs1, Rs2).
   % v prvni permutaci se H1 zobrazi na H2
   zarad(Start,[H1,H2|L],C2,Rs1,Rs2):- fnd(H2,C2,X), umisti(1,H1, X, Rs1, Rs3), zarad(Start,
[H2|L],C2,Rs3,Rs2).

   umisti(N,N,X,[_|T],[X|T]).
   umisti(I,N,X,[H|T],[H|Rs]):-I1 is I+1,umisti(I1,N,X,T,Rs).

   fnd(E,[[H|T1]|_],X):-fnd0(E,H,[H|T1],X). % na co se zobrazi prvek E v druhe permutaci
   fnd(E,[_|T],X):-fnd(E,T,X).

   fnd0(E,X,[E],X).
   fnd0(E,_,[H1,H2|_],H2):-E==H1.
   fnd0(E,H,[H1,H2|T1],X):-E\=H1,fnd0(E,H,[H2|T1],X).

   %sezn(+N, 1, -[1..N]).  ... vrati seznam N nul
   sezn(N, N, [N]).
   sezn(N, X, [0|T]) :- X < N, X1 is X + 1, sezn(N, X1, T).
```

Řešení by Mus:

```
   %POSTUP: brute-force cykly->permut->soucin->permut->cykly

   %soucinc(+Cykly, +Cykly, -SoucinCyklu)
   soucinc(c(N,C1), c(N,C2), c(N,C3)) :- cykly_perm(v(N,P1),c(N,C1)),  % vrati permutaci
jakozto seznam pres prvni parametr
              cykly_perm(v(N,P2),c(N,C2)),
              soucinp(P1, P2, P3),
              perm_cykly(v(N,P3), c(N,C3)).

   %soucinp (+Permutace1, +Permutace2, -Slozena) ... slozeni dvou permutaci ulozenych v
seznamech
   soucinp([], _, []).
   soucinp([P1|T1], P2, [S|TS]) :- nty(P1,P2,S), soucinp(T1, P2, TS).

   %nty(+N, +Seznam, -Nty, -Seznam), vrati Nty a vrati seznam, kde na Ntem prvku je 0
   nty(1, [H|_], H).
   nty(N, [_|T], X) :- N1 is N - 1, nty(N1, T, X).

   %cykly_perm(-Permutace, +Cykly)
   cykly_perm(v(0,_), c(0,[])).
   cykly_perm(v(N,P), c(N,C)) :- sezn(N, 1, P1), cykly_init(C, P1, P).

   %cykly_init(+Cykly, +TempPermutace, -Permutace)
   cykly_init([], P, P).
   cykly_init([C|TC], P1, P) :- cykl(C,C,1,P1,P2), cykly_init(TC, P2, P).

   %cykl(+Cyklus, +Cyklus, 1, +TempPermutace, -PermutaceDleCyklu)
   cykl([HC|_], [C|TC], N, [H|TP], [H|T2]) :- N < C, !, N1 is N + 1, cykl([HC], [C|TC], N1,
```

```
TP, T2).
   cykl([HC|_], [_], _, [_|TP], [HC|TP]).
   cykl([HC|_], [_,C2|TC], N, [_|TP], [C2|T2]) :- N1 is N + 1, cykl([HC], [C2|TC], N1, TP,
T2).

   %sezn(+N, 1, -[1..N]).  ... vrati seznam cisel od 1 do N
   sezn(N, N, [N]).
   sezn(N, X, [X|T]) :- X < N, X1 is X + 1, sezn(N, X1, T).
```

* Čísla reprezentujeme jako seznamy čísel jejich dvojkových zápisů. Sestavte predikát, který realizuje násobení čísel.

Řešení by Martin Všetička:

```
   binAdd(B1,B2,R):-binAdd(B1,B2,0,R1),reverse(R1,R).
   binAdd([],[],0,[]).
   binAdd([],[],1,[1]).
   binAdd([],[H2|T2],Rem,R):-binAdd([H2|T2],[],Rem,R).
   binAdd([H1|T1],[],0,[H1|T1]).
   binAdd([H1|T1],[],1,[H3|R]):-
      count(H1+1,H3,Rem2),
      binAdd(T1,[],Rem2,R).
   binAdd([H1|B1],[H2|B2],Rem,[H3|R]):-
      count(H1+H2+Rem,H3,Rem2),
      binAdd(B1,B2,Rem2,R).
   count(Sum,Show,Remainder):-Remainder is Sum//2, Show is Sum mod 2.


   % cisla se zadavaji odzadu, vysledek je jiz popredu
   binMult(B1,B2,R):-binMult(B1,B2,[],R1),reverse(R1,R).
   binMult(_,[],R,R).
   binMult(B1,[0|B2],M,R):-binMult([0|B1],B2,M,R).
   binMult(B1,[1|B2],M,R):-binAdd(M,B1,0,M1), binMult([0|B1],B2,M1,R).
```

Zdroje: http://www.binarymath.info/multiplication-division.php

Řešení z http://prgs.xf.cz/:

```
   %binarni cisla reprezentujeme jako seznam jednicek a nul

   %mult(+A, +B, -AxB)
   %v predikatu mult jsou v seznamu cisla serazene od nejvyssi vahy po nejmensi
   %cislo 12 = 1100 v programu [1,1,0,0]
   mult(A,B, Res):-
     reverse(A,RA),
     reverse(B,RB),
        nasob(RA, RB, RRes),
        reverse(RRes, Res).

   %reverse(+L, -ReversedL).
   reverse(L, RL):-reverse(L, [], RL).
   reverse([H|T], Acc, Res):-reverse(T, [H|Acc], Res).
   reverse([], Acc, Acc).

   %nasob(+A, +B, -AxB)
   %v seznamu jsou cisla serazene od nejmensi vahy po nejvetsi
   %cislo 12 = 1100 v programu [0,0,1,1]
   nasob(A, B, Res):-
        nasob(A, B, 0, [], Res).

   %nasob(+A, +B, +PosunitieBdolava, +DocasnySucet, -Res)
   nasob(_, [], _, Acc, Acc).
```

```
nasob(A, [0|B], N, Acc, Res):-
     N2 is N+1,
     nasob(A, B, N2, Acc, Res).
nasob(A, [1|B], N, Acc, Res):-
     posun(A, N, A2),
     scitaj(A2, Acc, 0, Tmp),
     N2 is N+1,
     nasob(A, B, N2, Tmp, Res).

%posun(+A, +PocetMist, -Aposunute_o_pocet_mist_doleva)
posun(A, 0, A):-!.
posun(A, N, [0|X]):-
     N2 is N-1,
  posun(A, N2, X).

%scitaj(+A, +B, +Prechod, -Res):-
scitaj([X|A],[Y|B], P, [Z|R]):-
     bs(X, Y, V, 0),
  bs(V, P, Z, P2),
  scitaj(A, B, P2, R).
scitaj([1|A], [1|B], P, [P|R]):-
  scitaj(A, B, 1, R).
scitaj([], B, 0, B).
scitaj(A, [], 0, A).
scitaj([], B, 1, R):-
   !,
   scitaj([1], B, 0, R).
scitaj(A, [], 1, R):-
   scitaj(A, [1], 0, R).

%bs(+C1, +C2, -Vysl, -Prechod)
%binarni scitani dvou binarnich cisel
bs(0,0,0,0).
bs(0,1,1,0).
bs(1,0,1,0).
bs(1,1,0,1).
```

*   Sestavte predikát natretiny(+Seznam,-Prvni,-Druhy,-Treti),

    který rozdělí vstupní seznam na tři seznamy přibližně stejné délky (Zřetězení seznamů Prvni,
    Druhy a Treti je seznam Seznam, délky seznamů Prvni, Druhy a Treti se mohou lišit nejvýše o 1).
    Při jeho konstrukci nesmíte použít žádnou aritmetiku (ani predikát length).

Řešení by Martin Všetička:

```
%na3(+Seznam,-Prvni,-Druhy,-Treti)
na3(X,R1,R2,R3):-fst3(X,X,R1,R),hlfs(R,R,R2,R3).

%hlfs(+Seznam,+Seznam,-Prvni,-Druhy)
hlfs([],T2,[],T2).
hlfs([_],[H|T2],[H],T2).
hlfs([_,_|T1],[H|T2],[H|Hlf],R):-hlfs(T1,T2,Hlf,R).

%fst3(+Seznam,+Seznam,-Tretina,-Zbytek)
fst3([],T2,[],T2).
fst3([_],[H|T2],[H],T2).
fst3([_,_],[H|T2],[H],T2).
fst3([_,_,_|T1],[H|T2],[H|Thrd],R):-fst3(T1,T2,Thrd,R).
```

Řešení z http://prgs.xf.cz:

```
%rozdeleni seznamu na tretiny
natretiny(L, L1, L2, L3):-
  tretiny(L, L1, L23),
  na2(L23, L2, L3).

%na2(+L, -L1, -L2).
%rozdeleni seznamu na dve poloviny
na2(L, L1, L2):-
  na2(L,L,L1,L2).
na2(L, [], [], L).
na2(L, [_], [], L).
na2([H|L],[_,_|T],[H|X],Y):-
  na2(L,T,X,Y).

%prvnitretina(+L123, -L1, -L23).
%rozdeleni seznamu na prvni tretinu a zbytek
tretiny(L, L1, L23):-
  tretiny(L, L, L1, L23).
tretiny(L, [], [], L).
tretiny(L, [_], [], L).
tretiny(L, [_,_], [], L).
tretiny([H|L],[_,_,_|T],[H|X],Y):-
  tretiny(L, T, X, Y).
```

* Máte k dispozici predikát modry/1, který uspěje, pokud argument je modrý. Sestavte predikát

    natretiny(+Seznam, -Prvni, -Druhy, -Treti),

který rozdělí efektivním způsobem vstupní seznam na tři seznamy obsahující přibližně stejně modrých prvků
(Zřetězení seznamu Prvni, Druhy a Treti je seznam Seznam, počty modrých prvků v seznamech Prvni, Druhy a Treti
se mohou lišit nejvýše o 1). Při jeho konstrukci nesmíte použít žádnou aritmetiku (ani predikát length).

Řešení od univ z fora (http://forum.matfyz.info/memberlist.php?mode=viewprofile&u=3483):

```
% natretiny(+,-,-,-)
natretiny(Seznam,Prvni,Druhy,Treti):-                  % hlavni predikat

   prvni_tretina(Seznam,Prvni,Zbytek),
   napoloviny(Zbytek,Druhy,Treti).

% prvni_tretina(+,-,-)
prvni_tretina(Seznam,Prvni,Zbytek):-tretina2(Seznam,Seznam,Prvni,Zbytek).

% tretina2(+,+,-,-)
tretina2(L1,L2,P,Zb):-

   odeber3modre(L1,L1Zb),!,             % pokusim se odebrat tri modre prvky, pokud
neuspeje, cela klauzule tretina2 zfailuje
   odeber1modry(L2,PZb,P,L2Zb),       % na tri nalezene modre mi pripada jeden, ktery
umistim do prvni tretiny
   tretina2(L1Zb,L2Zb,PZb,Zb).

tretina2(_,L2,[],L2).

% odeber1modry(+,+,-,-)
odeber1modry([H|L],PZb,[H|L2],Zb):-

   modry(H),!,L2=PZb,Zb=L;             % odeberu modry prvek
   odeber1modry(L,PZb,L2,Zb).          % .. nebo pokracuju ve zbytku seznamu

% odeber2modre(+,-)
```

```
odeber2modre([H|L],Zb):-

   modry(H),!,odeber1modry(L,_,_,Zb);  % odeberu dva modre prvky
   odeber2modre(L,Zb).                 % .. nebo pokracuju ve zbytku seznamu

% odeber3modre(+,-)
odeber3modre([H|L],Zb):-

   modry(H),!,odeber2modre(L,Zb);      % odeberu tri modre prvky
   odeber3modre(L,Zb).                 % .. nebo pokracuju ve zbytku seznamu


% napoloviny(+,-,-)
napoloviny(L,L1,L2):- polovina2(L,L,L1,L2).

% polovina2(+,+,-,-)
polovina2(L1,L2,P,Zb):-

   odeber2modre(L1,L1Zb),!,            % stejny princip, najdu dva modre, pak jeden hned
muzu zaradit do prvni poloviny
   odeber1modry(L2,PZb,P,L2Zb),
   polovina2(L1Zb,L2Zb,PZb,Zb).

polovina2(_,L2,[],L2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% TEST

:-op(100,xfy,->).

modry(m).

s([m,m,a,m,m,b,m,m]).
s([a,m,b,m,c,m,d,m,e,m,f]).
s([a,b,c,m,m,d,e,f,m,g,h,i,m,j,k,l]).

test:-s(X),natretiny(X,L1,L2,L3),write(X->L1+L2+L3),nl,fail.
```

* Sestavte proceduru-y, pro kódování a dekódování (dlouhého) seznamu pomocí šifry "Monte Christo".
   Text se zakóduje do čtvercové matice 4x4. Kódovacím klíčem je čtvercová mřížka stejných rozměrů s vhodně vyřezanými otvory. Text se vypisuje do otvorů ve mřížce postupně po řádcích, vždy do každého otvoru jedno písmeno. Pak se mřížka otočí o 90Â° doprava a další text se opět vypisuje do otvorů. Toto se opakuje celkem čtyřikrát. Po zaplnění celé matice se mřížka odstraní a obsah matice se vypíše po řádcích na výstup. Je-li šifrovaná zpráva delší než jeden čtverec (tj. delší než 4x4 písmen), rozdělí se na více úseků, každý o délce jednoho čtverce. Napište kódovací a dekódovací proceduru, musíte otestovat, zda je mřížka připustná.

Řešení by dobre_rano:

```
   % oboustranny predikat
   zasifruj(Text, Klic1, Sifra):-
     len(Text, Len), Len=16,
     take4(Text,T1, Zb1), works(T1,Klic1, Sifra),
     otoc(Klic1, Klic2), take4(Zb1,T2, Zb2), works(T2, Klic2, Sifra),
     otoc(Klic2, Klic3), take4(Zb2,T3, Zb3), works(T3, Klic3, Sifra),
     otoc(Klic3, Klic4), works(Zb3, Klic4, Sifra).

   % works(T, Klic, Sifra) ... provadi samotne (de)sifrovani
   works([],_,_).
   works([HP|TP], [HK|TK], [HS|TS]):-
     HK=1,!,HP=HS,works(TP, TK, TS);          % pokud se objevi policko, do ktereho mame
psat
     works([HP|TP], TK, TS).                  % pokud ne.

   % Natvrdo napsane otoceni mrizky 4x4.
   otoc([A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13,A14,A15,A16],
[A13,A9,A5,A1,A14,A10,A6,A2,A15,A11,A7,A3,A16,A12,A8,A4]).

   % Pro vsechny rotace klicove mrizky plati, ze dohromady musi vyrezana policka vsech
rotaci klicove mrizky pokryt celou mrizku,
   % tudiz pokud si seznamy rotaci klice napiseme nad sebe, tak ve sloupci musi byt vzdy
prave jedna jednicka
   % reprezentujici zapisovane policko tabulky, jinak by dochazelo k prepisum pri kodovani a
klic by nebyl pripustny.
   zkontroluj(Klic1):-
     otoc(Klic1, Klic2), otoc(Klic2, Klic3), otoc(Klic3, Klic4),
     zkontroluj(Klic1, Klic2, Klic3, Klic4).

   zkontroluj([],[],[],[]).
   zkontroluj([H1|T1], [H2|T2], [H3|T3], [H4|T4]):-
     (H1=:=1,H2=:=0,H3=:=0,H4=:=0;
     H1=:=0,H2=:=1,H3=:=0,H4=:=0;
     H1=:=0,H2=:=0,H3=:=1,H4=:=0;
     H1=:=0,H2=:=0,H3=:=0,H4=:=1),
     zkontroluj(T1,T2,T3,T4).

   % Vezmi ctyri prvky ze seznamu
   take4(Text,Ctyri,Zbytek):-
     takeN(4, Text, Ctyri, Zbytek).

   take4(0, L, [], L).
   take4(_, [], [],[]).
   take4(N, [H|L], [H|Zb], Rest):-
     N > 0, N1 is N -1,
     takeN(N1, L, Zb, Rest).

   len([],0).
   len([H|T], N):-
     len(T,N1), N is N1+1.
```