



Logické programovanie 3

- backtracking a nedeterministické programy (algebrogramy)
 - $SEND + MORE = MONEY$,
 - magické číslo,
 - 8-dám, ...
- nedeterministické konečné automaty inak
- cesta v grafe - prehľadávanie stavového priestoru
 - japonskí pltníci a misionári s kanibalmi
- logické hádanky
 - "zebra problem" alias *kto chová rybičky ?*

Cvičenie

- backtracking

Midterm

- total: 12.56/26
- fibo: 2.01/6
- foldy: 4.67/6
- kuchárka: 1.60/4
- kvíz: 3.42/7
- merge: 0.87/3

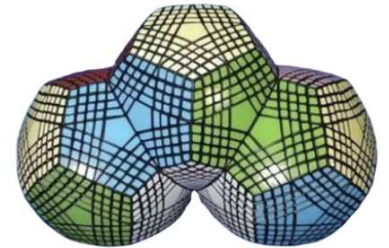
Math class



Math Homework



Math exam

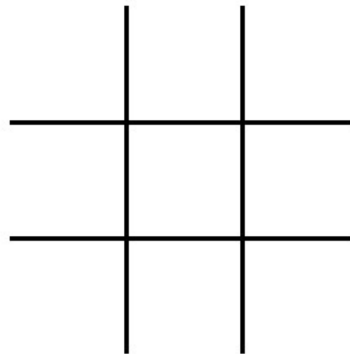




Bactracking

(ľahký úvod)

- vložte 6 kameňov do mriežky 3x3, tak aby v žiadnom smere (riadok, stĺpec, uhlopriečka) neboli tri.



- pri najivnom prehľadávaní - všetkých možností je $2^9 = 512$
- ak poznáme kombinácie bez opakovania - možností je už len 9 nad 6, teda 9 nad 3, čo je 84



Haskell to Prolog

- v Haskellu sme mali:

```
isOk :: [Int] -> Bool
```

```
isOk xs = not (subset' [0,1,2] xs) && not (subset' [3,4,5] xs) && not (subset' [6,7,8] xs) &&  
          not (subset' [0,3,6] xs) && not (subset' [1,4,7] xs) && not (subset' [2,5,8] xs) &&  
          not (subset' [0,4,8] xs) && not (subset' [2,4,6] xs)
```

- v Prologu nič ľahšie:

```
isOk(Xs):- not(subseq([0,1,2],Xs)), not(subseq([3,4,5], Xs)), not(subseq([6,7,8], Xs)),  
           not(subseq([0,3,6], Xs)), not(subseq([1,4,7], Xs)), not(subseq([2,5,8], Xs)),  
           not(subseq([0,4,8], Xs)), not(subseq([2,4,6], Xs)).
```

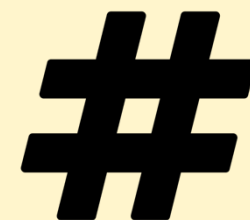
% - daj mi všetky 6 prvkové podmnožiny 0..8, že sú okay.

```
?- Cs=[_,_,_,_,_,_], comb(Cs,[0,1,2,3,4,5,6,7,8]), isOk(Cs).
```

```
?- Cs=[I1,I2,I3,I4,I5,I6],comb(Cs,[0,1,2,3,4,5,6,7,8]), isOk(Cs).
```

```
Cs = [0, 1, 3, 5, 7, 8];
```

```
Cs = [1, 2, 3, 5, 6, 7];
```



Prolog to eCLIPse

(constraint logic programming)

nie je hashtag ale **constraint**
obmedzenie, podmienka, vzťah

```
isOk(Xs):- Xs[1]+Xs[2]+Xs[3] #<3, Xs[4]+Xs[5]+Xs[6] #<3, Xs[7]+Xs[8]+Xs[9] #<3,
           Xs[1]+Xs[4]+Xs[7] #<3, Xs[2]+Xs[5]+Xs[8] #<3, Xs[3]+Xs[6]+Xs[9] #<3,
           Xs[1]+Xs[5]+Xs[9] #<3, Xs[3]+Xs[5]+Xs[7] #<3.
```

% pokus o cyklus v logickej paradigme vyzerá celkom tragicky...

```
isOk2(Xs):- (for(I,0,2), param(Xs) do Xs[1+3*I]+Xs[2+3*I]+Xs[3+3*I] #<3 ),
            (for(I,0,2), param(Xs) do Xs[1+I]+Xs[4+I]+Xs[7+I] #<3 ),
            Xs[1]+Xs[5]+Xs[9] #<3, Xs[3]+Xs[5]+Xs[7] #<3.
```

threeXthree(Cs) :-

dim(Cs,[9]),

Cs::0..1,

% 6 #= Cs[1] + Cs[2] + Cs[3] + Cs[4] + Cs[5] + Cs[6] + Cs[7] + Cs[8] + Cs[9],

6 #= sum(Cs[1..9]),

isOk2(Cs),

labeling(Cs), % backtrack

writeln(Cs).



?- threeXthree(Cs), fail.

[(0, 1, 1, 1, 0, 1, 1, 1, 0)]

[(1, 1, 0, 1, 0, 1, 0, 1, 1)]

Send More Money

(algebrogram)

SEND
+ MORE
=====
MONEY

cifra(1).cifra(2).cifra(3).cifra(4).cifra(5).
cifra(6).cifra(7).cifra(8).cifra(9).
cifra(X):-between(1,9,X).

s(S,E,N,D,M,O,R,Y):

cifra0(D),

cifra0(E),D\=E,

Y is (D+E) mod 10,

Y\=E,Y\=D,

Pr1 is (D+E) // 10,

cifra0(N),N\=D,N\=E,N\=Y,

cifra0(R),R\=D,R\=E,R\=Y,R\=N,

E is (N+R+Pr1) mod 10,

Pr2 is (N+R+Pr1) // 10,

cifra0(O), O\=D,O\=E,O\=Y,O\=N,O\=R,

N is (E+O+Pr2) mod 10,

Pr3 is (E+O+Pr2) // 10,

cifra0(S), S\=D,S\=E,S\=Y,S\=N,S\=R,S\=O,

cifra0(M), M\=0,M\=D,M\=E,M\=Y,M\=N,M\=R,M\=O,M\=S,

O is (S+M+Pr3) mod 10,

M is (S+M+Pr3) // 10,

write(' '),write(S),write(E),write(N),write(D),nl,

write('+'),write(M),write(O),write(R),write(E),

nl,

write(M),write(O),write(N),write(E),write(Y),nl.

cifra0(0).

cifra0(X):-cifra(X).

?- s(S,E,N,D,M,O,R,Y).

9567

+1085

10652

Toto prepíšeme na cvičení

S = 9

E = 5

N = 6

D = 7

M = 1

O = 0

R = 8

Y = 2

VINGT+CINQ+CINQ=TRENTE

(algebrogram – moje riešenie)

VINGT
+ CINQ
+ CINQ
=====

TRENTE

```
alldiff([]).
```

```
alldiff([X|Xs]):- not(member(X,Xs)), alldiff(Xs).
```

```
%- scitovanie po stlpcoch
```

```
sumCol(Cifra1,Cifra2,Cifra3,Cifra,Prenos,NovyPrenos):-
```

```
    NovyPrenos is (Cifra1+Cifra2+Cifra3+Prenos)//10,
```

```
    Cifra is (Cifra1+Cifra2+Cifra3+Prenos) mod 10.
```

```
puzzle([V,I,N,G,T,C,Q,R,E]):-
```

```
    cifra(T),cifra(Q), alldiff([T,Q]), sumCol(0,T,Q,Q,E,Pr1), alldiff([E,T,Q]),
```

```
    cifra(G),cifra(N),alldiff([G,N,E,T,Q]), sumCol(Pr1,G,N,N,T,Pr2),
```

```
    cifra(I),alldiff([I,G,N,E,T,Q]), sumCol(Pr2,N,I,I,N,Pr3),
```

```
    cifra(C),alldiff([C,I,G,N,E,T,Q]), sumCol(Pr3,I,C,C,E,Pr4),
```

```
    cifra(V),alldiff([V,C,I,G,N,E,T,Q]), sumCol(Pr4,V,0,0,R,T)
```

```
    write(' '),write(V),write(I),write(N),write(G),write(T),nl,
```

```
    write(' '), write(C),write(I),write(N),write(Q),nl,
```

```
    write(' '), write(C),write(I),write(N),write(Q),nl,
```

```
    write(T),write(R),write(E),write(N),write(T),write(E),nl.
```

```
?- puzzle([V,I,N,G,T,C,Q,R,E]).
```

```
94851
```

```
6483
```

```
6483
```

```
-----
```

```
107817
```

VINGT+CINQ+CINQ=TRENTE

(algebrogram – iný prístup)

```
solve(V,I,N,G,T,C,Q,E,R) :-  
  select(T,[0,1,2,3,4,5,6,7,8,9], R1),  
  select(Q, R1, R2),  
  sumCol(T,Q,Q,E,0,Pr),  
  select(E, R2,R3),  
  select(G, R3,R4),  
  select(N, R4,R5),  
  sumCol(G,N,N,T,Pr,Pr2),  
  select(I, R5,R6),  
  sumCol(N,I,I,N,Pr2, Pr3),  
  select(C,R6,R7),  
  sumCol(I,C,C,E,Pr3,Pr4),  
  select(V,R7,R8),  
  sumCol(V,0,0,R,Pr4,Pr5),  
  T = Pr5,  
  select(R,R8,_), not(T = 0),  
  write(' '),write(V),write(I),write(N),write(G),write(T),nl,  
  write(' '),    write(C),write(I),write(N),write(Q),nl,  
  write(' '),    write(C),write(I),write(N),write(Q),nl,  
  write(T),write(R),write(E),write(N),write(T),write(E),nl.
```

```
% T∈[1..9],      R1=[1..9] \\ [T]  
% Q∈[1..9] \\ [T], R2=[1..9] \\ [T,Q]  
% 10*Pr+E = T+Q+Q+0
```

```
VINGT  
+ CINQ  
+ CINQ  
=====  
TRENTE
```

```
?- solve(V,I,N,G,T,C,Q,E,R).  
94851  
6483  
6483  
-----  
107817
```




<http://eclipseclp.org/>

Send More Money

(constraint logic programming)

```
:- lib(ic).
```

```
sendmore(Digits) :-
```

```
    Digits = [S,E,N,D,M,O,R,Y],
```

```
    Digits :: [0..9],           % obor hodnôt
```

```
    alldifferent(Digits),       % všetky prvky zoznamu musia byť rôzne, built-in
```

```
    S #\= 0, M #\= 0,          % úvodné cifry nemôžu byť 0
```

```
    (1000*S + 100*E + 10*N + D) + (1000*M + 100*O + 10*R + E)
```

```
        #= 10000*M + 1000*O + 100*N + 10*E + Y,
```

```
    labeling(Digits),           % generovanie možností, backtrack
```

```
    writeSolution(Digits).       % výpis riešenia
```

```
writeSolution([S,E,N,D,M,O,R,Y]) :-
```

```
    write(' '),write(S),write(E),write(N),write(D), nl,
```

```
    write('+'),write(M),write(O),write(R),write(E), nl,
```

```
    write(M), write(O),write(N),write(E),write(Y),nl.
```

SEND
+ MORE
=====
MONEY



Magické

- 381 je magické, lebo
3 je deliteľné 1,
38 je deliteľné 2,
381 je deliteľné 3.

?- umagic9([]).
381654729

- magicke(X):-magicke(X,0,0).

?- magicke([3,8,1]).
true.

- magicke([],_,_).
magicke([X|Xs],Cislo,N) :- Cislo1 is 10*Cislo+X,
N1 is N+1,
0 is Cislo1 mod N1,
magicke(Xs,Cislo1,N1).

- uplneMagicke(X) :- magicke(X), member(1,X), member(2,X), ...

?- uplneMagicke([3,8,1]). false.
?- uplneMagicke([3,8,1,6,5,4,7,2,9]) . true .

ifthenelse(C,T,E):-C->T;E
ifthen(C,T):-C->T

Ako nájdeme úplne magické

cifra(1).cifra(2).cifra(3).cifra(4).cifra(5).cifra(6).cifra(7).cifra(8).cifra(9).

- *technika "generuj a testuj"*

```
umag(X) :-    cifra(C1), cifra(C2), cifra(C3), cifra(C4), cifra(C5),  
              cifra(C6), cifra(C7), cifra(C8), cifra(C9),  
              uplneMagicke([C1,C2,C3,C4,C5,C6,C7,C8,C9]),  
              zoznamToInt2([C1,C2,C3,C4,C5,C6,C7,C8,C9],X).
```

- *technika backtracking*

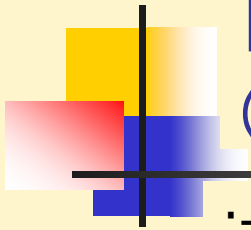
```
umagiccifra(X):- length(X,9) -> zoznamToInt2(X,Y),write(Y),nl  
;   
  cifra(C),not(member(C,X)),  
  append(X,[C],Y),  
  magicke(Y),  
  umagiccifra(Y).
```

if

then

else

?- umagic9([]).
381654729



Magické

(constraint logic programming)



<http://eclipseclp.org/>

```
:- lib(ic).
```

```
uplneMagicke(Digits):-
```

```
    Digits = [_,_,_,_,_,_,_,_],
```

```
    Digits :: [1..9],
```

```
    alldifferent(Digits),
```

```
    magicke(Digits),
```

```
    labeling(Digits).
```

% obor hodnôt

% všetky prvky zoznamu musia byť rôzne

% generovanie možností

```
magicke(X):-magicke(X,0,0).
```

```
magicke([],_,_).
```

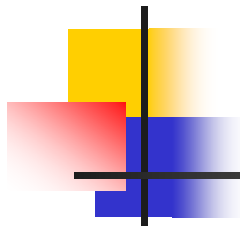
```
magicke([X|Xs],Cislo,N) :-
```

```
    Cislo1  $\# =$  10*Cislo+X,
```

```
    N1 is N+1,
```

```
    Cislo1 / N1  $\# =$  _,
```

```
    magicke(Xs,Cislo1,N1).
```



Master Mind

(hra Logic)

Hra MasterMind sa hráva vo viacerých verziách. Najjednoduchšia je taká, že hádate 4-ciferné číslo pozostávajúce z neopakujúcich sa čífiel od 1 do 6. Napríklad, ak hádate utajené číslo 4251, hádajúci položí dotaz 1234, tak dostane odpoveď, koľko čífiel ste uhádli (t.j. 3, lebo 1,2,4), a koľko je na svojom mieste (t.j. 1, lebo 2 je na "svojom" mieste v dotaze). Odpoveď je teda 3:1.

Definujte predikát $mm(Utajene, Dotaz, X, Y)$, ktorý pre známe Utajene a Dotaz v tvare zoznamov $[4,2,5,1]$ a $[1,2,3,4]$ určí odpoveď $X:Y$, t.j., $X=3$ a $Y=1$.

Z rozohranej partie MasterMind ostal len zoznam dotazov a odpovedí hádajúceho vo formáte zoznamu, napr.
 $P = [dotaz([1,2,3,4],3,1), dotaz([4,3,2,1],3,2)]$. Definujte predikát $findMM(P, X)$, ktorý pre zadaný zoznam dotazov P nájde všetky možné utajené čísla X , ktoré vyhovujú odpovediam na tieto dotazy.
Napr. $X = [4,2,5,1]$ ale aj ďalšie.



Master Mind 1

Hádané číslo

[2,3,6,4]

[1, 2,3,4] 3:1,

[3, 2,1,5] 2:0

[6, 4,3,1] 3:0

MasterMind ... koľko cifier ste uhádli, a koľko je na svojom mieste

- predikát spočíta počet *právd* v zozname:

```
countTrue([],0).
```

```
countTrue([C|Cs],N) :- countTrue(Cs,N1),
```

```
(C -> N is N1+1
```

% ak pravda,+1

```
;
```

```
N is N1).
```

% inak nič

- mm([C1,C2,C3,C4],[Q1,Q2,Q3,Q4],X,Y) :-

```
C = [C1,C2,C3,C4],
```

```
countTrue([member(Q1,C),member(Q2,C),  
            member(Q3,C),member(Q4,C)],X),
```

```
countTrue([C1=Q1,C2=Q2,C3=Q3,C4=Q4],Y).
```

Master Mind ešte príde...



Master Mind 2

definujte predikát `findMM(P,X)`, ktorý pre zadaný zoznam dotazov `P` nájde všetky možné utajené čísla `X`, ktoré vyhovujú odpovediam.

```
findMM(Qs, Code) :-
    Code1 = [_,_,_,_],
    comb(Code1,[1,2,3,4,5,6]),
    perm(Code1,Code),
    checkMM(Qs,Code).
% Qs-zoznam dotazov s odpoveďami
% hádaš štvorciferné číslo
% ... 4-kombinácie množiny {1..6}
% ... a to rôzne poprehadzované
% ... že všetky dotazy platia
```

```
checkMM([],_).
checkMM([dotaz(Q,X,Y)|Qs],Code) :-
    mm(Q,Code,X,Y), checkMM(Qs,Code).
```

```
?-findMM([
    dotaz([1,2,3,4],3,1),
    dotaz([3,2,1,5],2,0),
    dotaz([6,4,3,1],3,0)],C).
```

```
C = [1, 6, 4, 2] ;
```

```
C = [2, 1, 6, 4] ;
```

```
C = [2, 3, 6, 4] ;
```

```
No
```



<http://eclipseclp.org/>

Master Mind

(constraint logic programming)

```
findMM(Qs, Code) :-  
    Code = [_,_,_,_],  
    Code :: [1..6],  
    alldifferent(Code),  
    labeling(Code),  
    checkMM(Qs,Code).
```

```
findMM([dotaz([1,2,3,4],3,1),dotaz([3,2,1,5],2,0),dotaz([6,4,3,1],3,0)],C),writeln(C),fail.
```

```
[1, 6, 4, 2]
```

```
[2, 1, 6, 4]
```

```
[2, 3, 6, 4]
```


8 dăm

```
queens:-queens(8,[]).
```

```
queens(N,Qs):-N==0->
```

```
    write(Qs), nl, fail
```

```
    ;
```

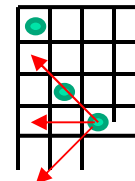
```
    q(Q),safe(Q,Q,Q,Qs),N1 is N-1,queens(N1,[Q|Qs]).
```

```
q(X):-between(1,8,X).
```

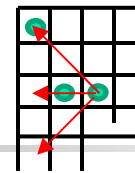
```
safe(_,_,_,[]).
```

```
safe(A,B,C,[D|Ds]):-
```

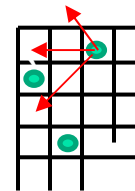
```
    A1 is A+1, C1 is C-1, A1\=D, B\=D, C1\=D, safe(A1,B,C1,Ds).
```



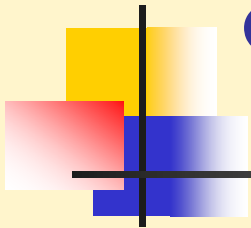
```
safe(3,3,3,[2,0]) :-  
    4,3,2 \= 2, ...
```



```
safe(1,1,1,[2,0]) :-  
    2,1,0 \= 2, ...
```



```
safe(0,0,0,[3,1]) :-  
    1,0,-1 \= 3,  
    safe(1,0,-1,[1]):-  
        2,0,-2 \= 1,  
        safe(2,0,-2,[]).
```



8 dám

(constraint logic programming)

```
queens2:-queens2(8,[]).
```

```
queens2(N,Qs):-
```

```
    N==0 -> labeling(Qs),writeln(Qs),fail
```

```
    ;
```

```
    Q::1..8, safe(1,Q,Qs),N1 is N-1,queens2(N1,[Q|Qs]).
```

```
safe(_,_,[]).
```

```
safe(I,B,[A|Qs]):-I1 is I+1,
```

```
    B+I #\= A,
```

```
    B #\= A,
```

```
    B-I #\= A,
```

```
    safe(I1,B,Qs).
```

Kol'ko nerovnic vygeneruje tento program ?

sú to rovnice nad celými číslami

N:	Total	Unique
5:	10	2
6:	4	1
7:	40	6
8:	92	12
9:	352	46
10:	724	92
11:	2680	341
12:	14200	1787
13:	73712	9233
14:	365596	45752
15:	2279184	285053
16:	14772512	1846955
17:	95815104	11977939
18:	666090624	83263591
19:	4968057848	621012754
20:	39029188884	4878666808
21:	314666222712	39333324973



8 dám

(constraint logic programming)

queens(Board) :-

Size = 8,

dim(Board, [Size]),

Board[1..Size] :: 1..Size,

(for(I,1,Size), param(Board,Size) do

(for(J,I+1,Size), param(Board,I) do

Board[I] #\= Board[J],

Board[I] #\= Board[J]+J-I,

Board[I] #\= Board[J]+I-J

)

),

labeling(Board),

write(Board), nl.

% Board je vector veľkosti Size

% prvky Boardu sú 1..Size

% for i in 1..Size ... % param

% sprístupní Board, Size

% for j in i+1..Size ...

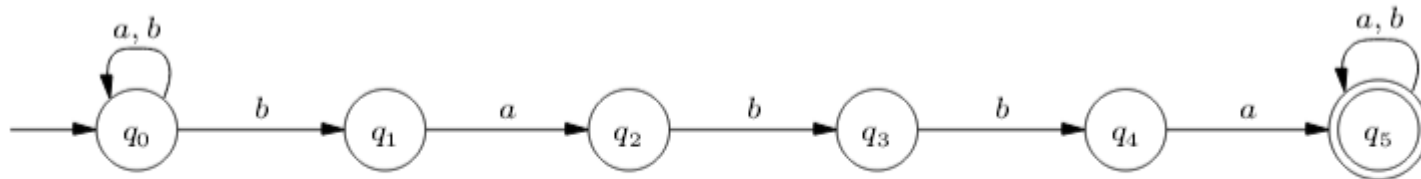
% param sprístupní Board, I

% param(X) deklaruje, že premennú

% z mimo cyklu možno použiť v cykle

N:	Total	Unique
5:	10	2
6:	4	1
7:	40	6
8:	92	12
9:	352	46
10:	724	92
11:	2680	341
12:	14200	1787
13:	73712	9233
14:	365596	45752
15:	2279184	285053
16:	14772512	1846955
17:	95815104	11977939
18:	666090624	83263591
19:	4968057848	621012754
20:	39029188884	4878666808
21:	314666222712	39333324973

Nedeterministický konečný automat



NKA pre jazyk $\{w \mid w \text{ obsahuje podslovo } babba\}$.

% --- prechodová funkcia δ :

next(q0,a,q0). next(q0,b,q0). next(q0,b,q1).

next(q1,a,q2). next(q2,b,q3). next(q3,b,q4).

next(q4,a,q5). next(q5,a,q5). next(q5,b,q5).

% --- počiatkový a množina koncových stavov

initial(q0).

finals([q5]).

% --- akceptovanie na NKA

accept(Ws) :- initial(IS), derivation(IS,Ws).

derivation(S,[]) :- finals(Fins), member(S,Fins).

derivation(S, [W|Ws]) :- next(S,W,S1), derivation(S1,Ws).

?- accept([b,a,b,b,a]).

true

?- accept([b,a,b,b,b,a,b,b,a]).

true



Jazyk akceptovaný NKA

%- generátor všetkých slov $\{a,b\}^*$, ale zlý...

```
word([]).  
word([a|Ws]):-word(Ws).  
word([b|Ws]):-word(Ws).
```

```
?- word(W).  
W = [] ;  
W = [a] ;  
W = [a, a] ;  
W = [a, a, a] ;  
W = [a, a, a, a] ;  
.....
```

%- generátor všetkých slov dĺžky k (K-vso nad množinou symbolov, teda a,b)

```
kword(0,[]).  
kword(K,[a|Ws]):-K>0,K1 is K-1,kword(K1,Ws).  
kword(K,[b|Ws]):-K>0,K1 is K-1,kword(K1,Ws).
```

```
?- kword(3,W).  
W = [a, a, a] ;  
W = [a, a, b] ;  
W = [a, b, a] ;  
W = [a, b, b] ;  
W = [b, a, a] ;  
W = [b, a, b] ;  
W = [b, b, a] ;  
W = [b, b, b] ;  
false.
```

% --- k-prvkové variácie s opakovaním, vso(K,Alphabet,Word)

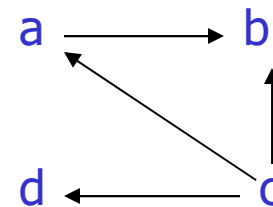
```
vso(0,Alphabet,[]).  
vso(K,Alphabet,[Symbol|Ws]):-K>0,K1 is K-1,  
member(Symbol, Alphabet), vso(K1,Alphabet,Ws).
```

% ---jazyk slov dĺžky max. 10 akceptovaný automatom

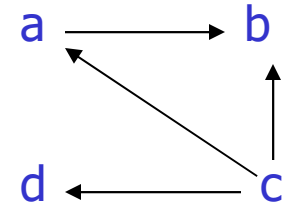
```
language(Word) :- between(0,10,Len), vso(Len,[a,b],Word), accept(Word).
```

Cesta v grafe

- majme graf definovaný predikátom hrana/2
hrana(a,b). hrana(c,a). hrana(c,b). hrana(c,d).
- predikát cesta/2 znamená, že medzi X a Y existuje postupnosť hrán
cesta(X, X).
cesta(X, Y) :- hrana(X, Z), cesta (Z, Y).
cesta(X, Y) :- (hrana(X, Z) ; hrana(Z, X)), cesta (Z, Y).
- ?-cesta(a,d).
no
- ?- cesta(a,d).
...



Cesta v cyklickom grafe



- `neohrana(X,Y) :- hrana(X,Y).`
`neohrana(X,Y) :- hrana(Y,X).`
- `cesta(X, X, _).`
`cesta(X, Y,C):-neohrana(X, Z), not member(Z,C), cesta (Z, Y,[Z|C]).`
- `?-cesta(a,d,[a]).`
yes
- `cesta(X, X, C, Res) :- Res = C.`
`cesta(X, Y,C,Res) :- neohrana(X, Z),`
`not member(Z,C), cesta (Z, Y,[Z|C], Res).`
- `?- cesta(a,d,[],Res).`
`Res = [a,c,d]`



Misionári a kanibali

(príklad použitia prehl'adávania grafu – stavového priestoru)



Traja misionári a traja kanibali sa stretli na jednom brehu rieky. Na brehu bola malá loďka, na ktorú sa zmestia maximálne dve osoby. Všetci sa chcú prepraviť na druhý breh, ale na žiadnom brehu nesmie nikdy zostať prevaha kanibalov nad misionármi, inak by mohlo dôjsť k tragédií. Akým spôsobom sa majú dostať na druhý breh?

<http://game-game.sk/18394/>



Misionári a kanibali

(príklad použitia prehľadávania grafu do hĺbky)

```
check(M,K) :- M = 0 ; K =< M.
```

```
check2(M,K) :- check(M,K), M1 is 3-M, K1 is 3-K, check(M1, K1).
```

```
init(state(3,3,l)).
```

```
final(state(0,0,r)).
```

```
% vľavo loďka, 3 misio a 3 canibs
```

```
% vpravo loďka, 3 misio a 3 canibs
```

```
% príklad prechodového pravidla
```

```
hrana(state(M,K,l), state(M1, K1, r)) :- check2(M,K),  
((M>1, M1 is M-2, K1 is K);  
 (M>0, M1 is M-1, K1 is K);  
 (K>0, M1 is M, K1 is K-1);  
 (M>0, K>0, M1 is M-1, K1 is K-1);  
 (K>1, M1 is M, K1 is K-2)),  
check2(M1,K1).
```

```
misio :- init(I), final(F), cesta(I,F,[],P), write(P).
```

```
[state(0, 0, r), state(1, 1, l), state(0, 1, r), state(0, 3, l), state(0, 2, r), state(2, 2, l),  
state(1, 1, r), state(3, 1, l), state(3, 0, r), state(3, 2, l), state(2, 2, r), state(3, 3, l)]
```

Japončící



<http://www.justonlinegames.com/games/river-iq-game.html>

- Pravidla hry jsou následující:
 1. Na voru se mohou vést najednou maximálně dvě osoby.
 2. Otec nemůže zůstat ani s jednou dcerou bez přítomnosti matky
 3. Matka nemůže zůstat ani s jedním synem bez přítomnosti otce
 4. Kriminálník (v pruhovaném obleku) nemůže zůstat ani s jedním členem rodiny bez přítomnosti policisty.
 5. Jen otec, matka a policista se umí plavit na voru.



japonci – stavy na l'avobrehu

```
%--- [boat,father,mother,sons,daughters,policeman,criminal]
```

```
%--- next(state1, state2)
```

```
next([1,1,M,S,D,P,C], [0,0,M,S,D,P,C]).
```

```
next([1,F,1,S,D,P,C], [0,F,0,S,D,P,C]).
```

```
next([1,F,M,S,D,1,C], [0,F,M,S,D,0,C]).
```

```
next([1,1,1,S,D,P,C], [0,0,0,S,D,P,C]).
```

```
. . . . .
```

```
next([1,1,M,S,D,P,C], [0,0,M,SX,D,P,C]) :- between(1,2,S), succ(SX,S).
```

```
next([1,F,M,S,D,1,C], [0,F,M,SX,D,0,C]) :- between(1,2,S), succ(SX,S).
```

```
next([1,F,M,S,D,1,C], [0,F,M,S,DX,0,C]) :- between(1,2,D), succ(DX,D).
```

```
. . . . .
```



japonci – kritické situácie

`valid_father([_,F,M,_,D,_,_]) :- F = M; (F = 1, D = 0); (F = 0, D = 2).`

`valid_mother([_,F,M,S,_,_,_]) :- F = M; (M = 1, S = 0); (M = 0, S = 2).`

`valid_criminal([_,F,M,S,D,P,C]) :- C = P;
 (C = 1, F = 0, M = 0, S = 0, D = 0)
 ;
 (C = 0, F = 1, M = 1, S = 2, D = 2).`

`valid(S) :- valid_father(S), valid_mother(S), valid_criminal(S).`



japonci – riešenie

?- solve.	[boat,father,mother,sons,daughters,policeman,criminal]
[1, 1, 1, 2, 2, 1, 1]	počiatočný stav
[0, 1, 1, 2, 2, 0, 0]	P+C ->
[1, 1, 1, 2, 2, 1, 0]	<- P
[0, 1, 1, 1, 2, 0, 0]	P+S ->
[1, 1, 1, 1, 2, 1, 1]	<- P+C
[0, 0, 1, 0, 2, 1, 1]	F+S ->
[1, 1, 1, 0, 2, 1, 1]	<- F
[0, 0, 0, 0, 2, 1, 1]	F+M ->
[1, 0, 1, 0, 2, 1, 1]	<- M
[0, 0, 1, 0, 2, 0, 0]	P+C ->
[1, 1, 1, 0, 2, 0, 0]	<- F
[0, 0, 0, 0, 2, 0, 0]	M+F ->
[1, 0, 1, 0, 2, 0, 0]	<- M
[0, 0, 0, 0, 1, 0, 0]	M+D->
[1, 0, 0, 0, 1, 1, 1]	<- P+C
[0, 0, 0, 0, 0, 0, 1]	P+D->
[1, 0, 0, 0, 0, 1, 1]	<- P
[0, 0, 0, 0, 0, 0, 0]	P+C ->



Susedia

(zebra problem)

Tento kvíz údajne vymyslel Albert Einstein a údajne ho 98% ľudí vôbec nevyrieši.

Je rada piatich domov, pričom každý má inú farbu. V týchto domoch žije päť ľudí rôznych národností. Každý z nich chová iné zviera, rád pije iný nápoj a fajčí iné cigarety.

1. Brit býva v červenom dome.
 2. Švéd chová psa.
 3. Dán pije čaj.
 4. Zelený dom stojí hneď naľavo od bieleho.
 5. Majiteľ zeleného domu pije kávu.
 6. Ten, kto fajčí Pall Mall, chová vtáka.
 7. Majiteľ žltého domu fajčí Dunhill.
 8. Človek z prostredného domu pije mlieko.
 9. Nór býva v prvom dome.
 10. Ten, kto fajčí Blend, býva vedľa toho, kto chová mačku.
 11. Ten, kto chová kone, býva vedľa toho, kto fajčí Dunhill.
 12. Ten, kto fajčí Blue Master, pije pivo.
 13. Nemec fajčí Prince.
 14. Nór býva vedľa modrého domu.
 15. Ten, kto fajčí Blend, má suseda, ktorý pije vodu.
- Kto chová rybičky? (patríte medzi tie 2%) ?



Susedia - 1

domy sú v rade indexované 1..5

% dom	1	2	3	4	5
% narod	N1	N2	N3	N4	N5
% zviera	Z1	Z2	Z3	Z4	Z5
% napoj	P1	P2	P3	P4	P5
% fajci	F1	F2	F3	F4	F5
% farba	C1	C2	C3	C4	C5

susedia(N,Z,P,F,C) :-

N=[N1,N2,N3,N4,N5], perm([brit,sved,dan,nor,nemec],N),
N1=nor, %- Nór býva v prvom dome

P=[P1,P2,P3,P4,P5], perm([caj,voda,pivo,kava,mlieko],P),
P3=mlieko, ... %- Človek z prostredného domu pije mlieko



Susedia - 2

z minulej prednášky:

predikát **index**(X,Xs,I), ktorý platí, ak $X_{si} = X$

$\text{index}(X, [X|_], 1)$.

$\text{index}(X, [_|Ys], I) :- \text{index}(X, Ys, I1), I \text{ is } I1+1$.

- **Dán pije čaj.**

$\text{index}(\text{dan}, N, I2), \text{index}(\text{caj}, P, I2),$

- **Brit býva v červenom dome.**

$C = [C1, C2, C3, C4, C5], \text{perm}([\text{cerveny}, \text{biely}, \text{modry}, \text{zlty}, \text{zeleny}], C),$

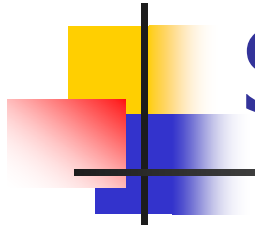
$\text{index}(\text{brit}, N, I3), \text{index}(\text{cerveny}, C, I3),$

- **Ten, kto fajčí Blend, býva vedľa toho, kto chová mačku.**

$F = [F1, F2, F3, F4, F5], \text{perm}([\text{pallmall}, \text{dunhill}, \text{prince}, \text{blend}, \text{bluemaster}], F),$

$\text{index}(\text{blend}, F, I10), \text{index}(\text{macka}, Z, I11), \text{vedla}(I10, I11),$

vedla(I,J) :- I is J+1 ; J is I+1.



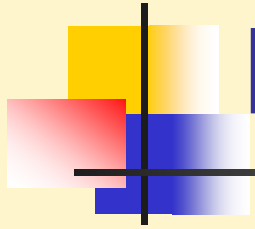
Susedia - 3

?- susedia(N,Z,P,F,C).

N = [nor,	dan,	brit,	nemec,	sved]
Z = [macka,	kon,	vtak,	rybicky,	pes]
P = [voda,	caj,	mlieko,	kava,	pivo]
F = [dunhill,	blend,	pallmall,	prince,	bluemaster]
C = [zlty,	modry,	cerveny,	zeleny,	biely] ;

No

Kto chová rybičky?



Kto rybičky

?- susedia(N, Z, P, F, C).
N = [3, 5, 2, 1, 4]
Z = [5, 3, 1, 2, 4]
P = [2, 4, 3, 5, 1]
F = [3, 1, 2, 5, 4]
C = [3, 5, 4, 1, 2]



<http://eclipseclp.org/>

susedia(N,Z,P,F,C):-


N = [Brit,Sved,Dan,Nor,Nemec], N :: 1..5, alldifferent(N),
Z = [Pes,Vtak,Macka,Kon,Rybicky], Z :: 1..5, alldifferent(Z),
P = [Caj,Kava,Mlieko,Pivo,Vodu], P :: 1..5, alldifferent(P),
F = [Pallmall,Dunhill,Blend,Bluemaster,Prince], F :: 1..5, alldifferent(F),
C = [Cerveny,Biely,Zeleny,Zlty,Modry], C :: 1..5, alldifferent(C),

Brit # = Cerveny, % Brit býva v červenom dome.
Biely # = Zeleny+1, % Zelený dom stojí hned nalavo od bieleho.
Mlieko # = 3, % Clovek z prostredného domu pije mlieko.
Nor # = 1, % Nór býva v prvom dome.
abs(Blend-Macka) # = 1, % Ten, kto fajčí Blend, býva vedľa chová macku.

labeling(N), labeling(C), labeling(Z), labeling(P), labeling(F).

Susedia

(iné riešenie, iná reprezentácia)



```
Houses = [      [N1,Z1,F1,P1,C1], % 1.dom [národ,zviera,fajčí,pije,farba]
                [N2,Z2,F2,P2,C2], % 2.dom
                [N3,Z3,F3,P3,C3], % 3.dom
                [N4,Z4,F4,P4,C4], % 4.dom
                [N5,Z5,F5,P5,C5] ].% 5.dom
```

ako vyjadríme fakt, že:

- **nór býva v prvom dome**
Houses = [[norwegian, _, _, _, _] | _]
- **človek z prostredného domu pije mlieko**
Houses = [_, _, [_, _, _, milk, _], _, _]
- **dán pije čaj**
member([dane, _, _, tea, _], Houses)
- **v susednom dome od ... definujeme pomocné predikáty next_to, iright**
next_to(X, Y, List) :- iright(X, Y, List) ; iright(Y, X, List).
iright(L, R, [L, R | _]).
iright(L, R, [_ | Rest]) :- iright(L, R, Rest).



Susedia (alias zebra problem)

einstein(Houses, Fish_Owner) :-

```
Houses = [[norwegian, _, _, _, _], _, [_, _, _, milk, _], _, _],  
member([brit, _, _, _, red], Houses),  
member([swede, dog, _, _, _], Houses),  
member([dane, _, _, tea, _], Houses),  
iright([_, _, _, green], [_, _, _, white], Houses),  
member([_, _, _, coffee, green], Houses),  
member([_, bird, pallmall, _, _], Houses),  
member([_, _, dunhill, _, yellow], Houses),  
next_to([_, _, dunhill, _, _], [_, horse, _, _, _], Houses),  
next_to([_, _, blend, _, _], [_, cat, _, _, _], Houses),  
next_to([_, _, blend, _, _], [_, _, _, water, _], Houses),  
member([_, _, bluemaster, beer, _], Houses),  
member([german, _, prince, _, _], Houses),  
next_to([norwegian, _, _, _, _], [_, _, _, blue], Houses),
```

```
member([Fish_Owner, fish, _, _, _], Houses). % kto chová rybičky ?
```

?- einstein(Houses, Fish_Owner).