



Logické programovanie 4

- backtracking a logické hádanky
 - "zebra problem" alias *kto chová rybičky ?*
- cesta v grafe - prehľadávanie stavového priestoru
- prehľadávanie grafu do šírky
 - druhorádové predikáty
- grafové algoritmy
 - jednot'ážka
 - farbenie grafu

Cvičenie

- grafové algoritmy
 - most, artikulácia grafu, súvislé komponenty, ...



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 $Xs[i]=X$

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

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

- **Dán pije čaj.**

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

- **Brit býva v červenom dome.**

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

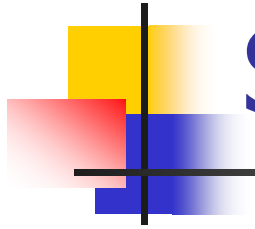
$\text{index}(\text{brit},N,I3), \text{index}(\text{cerveny},C,I3),$

- **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.



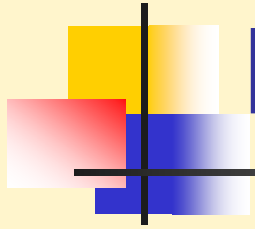
Susedia - 3

?- 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)

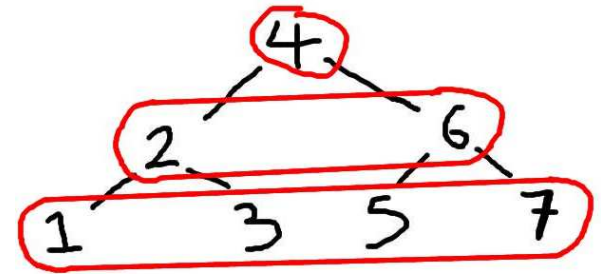
einstein(Houses, Fish_Owner) :-

```
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).

Reprezentácia grafu



<http://406notacceptable.com/java/breadth-first-tree-traversal-in-java/>

- reprezentácia grafu: hrana(X,Y)
hrana(a,b). hrana(c,a). hrana(c,b). hrana(c,d).
- ?- hrana(c,X).
X = a;
X = b;
X = d;
No
- iná reprezentácia grafu: susedia(c,[a,b,d])
- ako z prvej reprezentácie do druhej do druhej ??

s tým, čo vieme, to nejde ...
lebo backtracking...



Druho-rádové predikáty

- `bagof(Term, Cieľ, Bag)` –
Bag je *zoznam* inštancií Termu pre riešenia Cieľa
- `setof(Term, Cieľ, Set)` –
Set je *množina* inštancií Termu pre riešenia Cieľa

Príklady:

- `[1..N]`
`jednaAzN(N,List) :- bagof(X,between(1,N,X),List).`
- všetky kombinácie päťprvkovej množiny
`allCombinations(List) :- setof(C,comb(C,[1,2,3,4,5]),List).`



Existenčné premenné

?- bagof(X,(between(1,3,X),member(Y,[a,b,c])),Bag).

Y = a, Bag = [1, 2, 3] ;

Y = b, Bag = [1, 2, 3] ;

Y = c, Bag = [1, 2, 3].

% existuje také Y, že X patrí do 1..3 a Y do a..c

?- bagof(X,Y^(between(1,3,X),member(Y,[a,b,c])),Bag).

Bag = [1, 1, 1, 2, 2, 2, 3, 3, 3].

■ findall(Term, Ciel', Bag) –

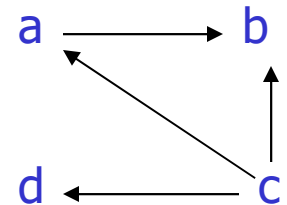
Bag je *zoznam* inštancií Termu pre riešenia Ciel'a, pričom všetky voľné premenné sú existenčne viazané

?- findall(X,(between(1,3,X),member(Y,[a,b,c])),Bag).

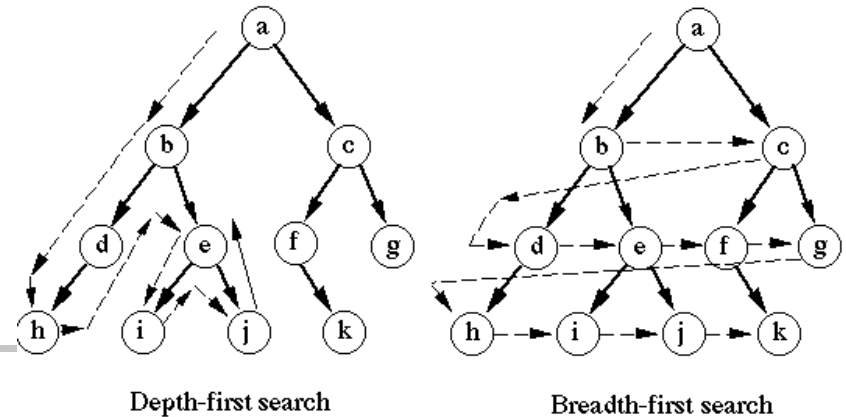
Bag = [1, 1, 1, 2, 2, 2, 3, 3, 3].

Susedia, stupeň vrchola

- súvislé komponenty grafu obsahujúci X
suvislyKomp(X,Comp):- setof(Y,Path[^]cesta(X,Y,[],Path),Comp).
- susedia(X, Neighbours) :- setof(Y,hrana(X,Y), Neighbours).
graph(G) :- setof((X,Neighbours), susedia(X, Neighbours),G).
- ?- graph(G).
G = [(a, [b]), (b, []), (c, [a, b, d]), (d, [])]
- stupenVrchola(X,N) :- susedia(X, Neighbours),
length(Neighbours,N).



Prehľadávanie grafu do šírky



```
cesta(X,Y) :- cestaDoSirky([X],[],Y).
```

```
cestaDoSirky([Y|_],_,Y).
```

```
cestaDoSirky([X|Xs],Navstivene,Y):-X\=Y,
```

```
% zober všetky dostupné vrcholy z X, kde si ešte nebol  
setof(Z,(hrana(X,Z), not(member(Z,Navstivene))),Nasledovnici),
```

```
% pridaj ich do fronty, na začiatok či koniec ?
```

```
%append(Nasledovnici,Xs,Xs1),          -- toto je do hĺbky
```

```
append(Xs,Nasledovnici,Xs1),          -- a toto do šírky
```

```
% opakuj v rekurzii kým vo fronte nie je vrchol kam smeruješ
```

```
cestaDoSirky(Xs1,[X|Navstivene],Y).
```

+ dostaneme najkratšiu cestu

- nevidíme ju (priamo)

Treba sa zamyslieť nad tým, ako si uchovať informácie pre rekonštrukciu cesty



Symbol rezu (cut)

Symbol rezu – označovaný predikátovým symbolom ! - predstavuje cieľ, ktorý je v triviálne splnený, má jedno riešenie, a pri pokuse o znovusplnenie je neúspešný. Okrem toho zakáže hľadať alternatívne riešenia pre:

- ciele vľavo od neho v klauzule, kde sa nachádza, a
- vo všetkých klauzulách nachádzajúcich sa pod touto klauzulou.

Príklad:

- Ak neplatí x , k symbolu ! sa výpočet nedostane.
- Ak sa podarí nájsť riešenie pre x , ! platí triviálne, hľadajú sa riešenia pre y .

Side-effect symbolu ! je to, že

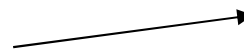
- iné riešenia pre x sa už nehľadajú
- ďalšie klauzuly pre b sa už nepoužívajú

Príklad:

foo:- a, b, c.

b :- ~~x~~!, y.

~~b :- z, w.~~





Príklad použitia !

(if-then-else)

- definujeme predikát, ktorý implementuje tradičný if-then-else
ifthenelse(C,T,E) :- C,T.
ifthenelse(C,T,E) :- not(C),E.

not(C) sa implementuje tak, že sa spustí výpočet C, a ak

- existuje riešenie pre C, not(C) neplatí,
- neexistuje riešenie pre C, not(C) platí.

tzv. negation as a failure.

Problém: predikát C sa vyhodnocuje dvakrát v prípade, ak neplatí C.

Riešenie (pomocou symbolu rezu):

ifthenelse(C,T,E) :- C,!,T.

ifthenelse(C,T,E) :- E.

Veľký problém: po sémantickej stránke je to *katastrofa*.



Príklady použitia symbolu rezu

príklad z generátora zoznamu

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

```
nAzJedna(N,[N|X]):-N>0,N1 is N-1,nAzJedna(N1,X).
```

pomocou !

```
nTo1(0,[]):-!.
```

```
nTo1(N,[N|X]):-N1 is N-1,nTo1(N1,X).
```

Minulé cvičenie

```
unary2bin(X,Y):-unary2bin(X,0,Y).
```

```
unary2bin(0, A, A):-!.
```

```
unary2bin(X,A,Y):-mod2(X,0),!,div2(X,X2),unary2bin(X2,o(A),Y).
```

```
unary2bin(X,A,Y):-div2(X,X2),unary2bin(X2,i(A),Y).
```




Zelené vs. červené !

Zelené rezy – jeho odstránenie nemení sémantiku programu, používame ho len pre zvýšenie efektívnosti – program je deterministickejší

`sign(X,-1) :- X<0, !.`

`sign(X, 0):-X=0, !.`

`sign(X, 1):-X>0, !.`

Červené rezy – menia význam programu

`not C :- C, !, fail.`

`not C.`

fail je predikát, ktorý nikdy neplatí

true je predikát, ktorý vždy platí, a má jediné riešenie

repeat je predikát, ktorý vždy platí a má nekonečne veľa riešení

`repeat.`

`repeat:-repeat.`



Cyklus repeat-fail

Zadaj dvojčiferné číslo z intervalu 10..99 a kým ho nezadáš, program ťa nepustí ďalej

```
cisloXX(XX) :- repeat, write('Zadaj dvociferne cislo '), read(XX),  
                (XX<100,XX>9, !, write(ok) ; write(zle), nl, fail).
```

ak prejde test $XX < 100, XX > 9$, znefunkční sa druhá vetva 'zle' aj alternatívy pre repeat

```
cisloXX1(XX) :- repeat, write('Zadaj dvociferne cislo '), read(XX),  
                (XX<100,XX>9 -> write(ok) , !  
                ; write(zle), nl, fail).
```

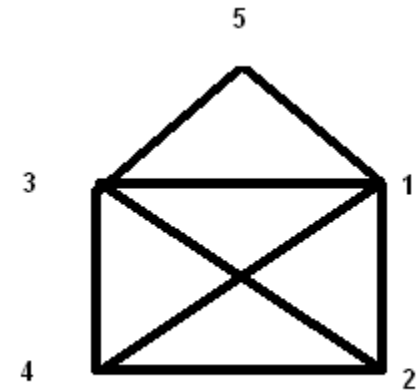
! ukrytý v $C \rightarrow T;E$ čo je syntax pre ifthenelse nemá vplyv na túto klauzulu, t.j. ak zadáme dvojčiferné číslo, systém chce ďalšie

Kaliningradské jednotáčky

```
neohr(1,2).
neohr(1,3).
neohr(1,4).
neohr(1,5).
```

```
neohr(2,3).
neohr(2,4).
```

```
neohr(3,4).
neohr(3,5).
```



Kaliningradský problém – prejsť všetky mosty a skončiť tam, kde sme začali

hrany(Hrany) :- setof((X,Y), **neohr(X,Y)**, Hrany).

jednotazka(X,Res) :- hrany(Hrany), jednotazka(X,Hrany,[X],Res).

jednotazka(X,[],C,Res):-Res = C.

jednotazka(X,Hrany,C,Res):-

(delete((X,Y),Hrany,Hrany1)

;

delete((Y,X),Hrany,Hrany1)),
jednotazka(Y,Hrany1,[Y|C],Res).

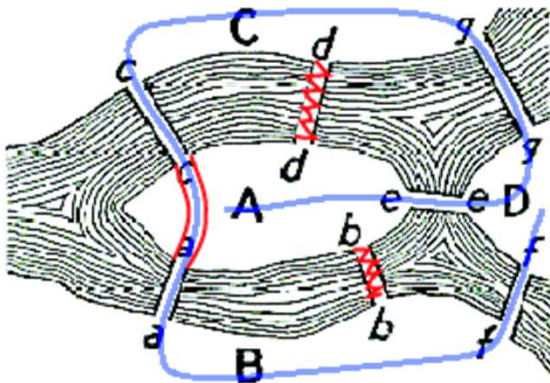


FIGURE 28. Geographic Map:
The Königsberg Bridges.

?- jednotazka(2,R).

R = [4, 2, 1, 5, 3, 1, 4, 3, 2] ;

R = [4, 2, 1, 3, 5, 1, 4, 3, 2] ;

R = [4, 1, 5, 3, 1, 2, 4, 3, 2] ;

Farbenie mapy



```
mapa(  
  [susedia(portugal, Portugal, [Spain]),  
    susedia(spain, Spain, [France,Portugal]),  
    susedia(france, France,  
              [Spain,Italy,Switzerland,Belgium,Germany,Luxemburg]),  
    susedia(belgium, Belgium, [France,Holland,Luxemburg,Germany]),  
    susedia(holland, Holland, [Belgium,Germany]),  
    susedia(germany, Germany,  
              [France,Austria,Switzerland,Holland,Belgium,Luxemburg]),  
    susedia(luxemburg,Luxemburg,[France,Belgium,Germany]),  
    susedia(italy, Italy, [France,Austria,Switzerland]),  
    susedia(switzerland,Switzerland,[France,Austria,Germany,Italy]),  
    susedia(austria, Austria, [Italy,Switzerland,Germany])  
]).
```

```
colors([green,red,blue,yellow]).
```

% zoznam farieb

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

% všetci sa nachádzajú v...

```
subset([X|Xs],Ys) :- member(X,Ys), subset(Xs,Ys).
```



Farbenie mapy

```
colorMap([],_).  
colorMap([susedia(_,Color,Neighbors)|Xs],Colors) :-  
    select(Color,Colors,Colors1),           % vyber farbu, ofarbi krajinu  
    subset(Neighbors,Colors1),               % prever, či susedia majú  
    colorMap(Xs,Colors).                     % iné farby a opakuj, kým []
```

?- mapa(M), colors(Cols), colorMap(M,Cols), write(M),nl.

```
[susedia(portugal, green, [red]),  
 susedia(spain, red, [green, green]),  
 susedia(france, green, [red, red, blue, red, yellow, blue]),  
 susedia(belgium, red, [green, green, blue, yellow]),  
 susedia(holland, green, [red, yellow]),  
 susedia(germany, yellow, [green, green, blue, green, red, blue]),  
 susedia(luxemburg, blue, [green, red, yellow]),  
 susedia(italy, red, [green, green, blue]),  
 susedia(switzerland, blue, [green, green, yellow, red]),  
 susedia(austria, green, [red, blue, yellow])]
```

Farbenie mapy2

```
mapa([
  susedia(portugal,Portugal,[Spain]),
  susedia(spain,Spain,[Portugal,Andorra,France]),
  susedia(Andorra,Andorra,[Spain,France]),
  susedia(france,France,[Spain,Andorra,Monaco,Italy,Switzerland,Germany,Luxembourg,Belgium,United_kingdom]),
  susedia(united_kingdom,United_kingdom,[France,Belgium,Netherlands,Denmark,Norway,Iceland,Ireland]),
  susedia(Ireland,Ireland,[United_kingdom,Iceland]),
  susedia(monaco,Monaco,[France]),
  susedia(italy,Italy,[France,Greece,Albania,Montenegro,Croatia,Slovenia,Austria,Switzerland,San_marino]),
  susedia(san_marino,San_marino,[Italy]),
  susedia(switzerland,Switzerland,[France,Italy,Austria,Germany,Liechtenstein]),
  susedia(liechtenstein,Liechtenstein,[Switzerland,Austria]),
  susedia(germany,Germany,[France,Switzerland,Austria,Czech_republic,Poland,Sweden,Denmark,Netherlands,Belgium,Luxembourg]),
  susedia(belgium,Belgium,[France,Luxembourg,Germany,Netherlands]),
  susedia(netherlands,Netherlands,[Belgium,Germany,United_kingdom]),
  susedia(luxembourg,Luxembourg,[France,Germany,Belgium]),
  susedia(austria,Austria,[Italy,Slovenia,Hungary,Slovakia,Czech_republic,Germany,Switzerland,Liechtenstein]),
  susedia(slovenia,Slovenia,[Italy,Croatia,Hungary,Austria]),
  susedia(croatia,Croatia,[Italy,Montenegro,Bosnia,Serbia,Hungary,Slovenia]),
  susedia(bosnia,Bosnia,[Croatia,Montenegro,Serbia]),
  susedia(montenegro,Montenegro,[Croatia,Italy,Albania,Serbia,Bosnia]),
  susedia(albania,Albania,[Italy,Greece,Macedonia,Serbia,Montenegro]),
  susedia(greece,Greece,[Italy,Cyprus,Bulgaria,Macedonia,Albania]),
  susedia(cyprus,Cyprus,[Greece]),
  susedia(macedonia,Macedonia,[Albania,Greece,Bulgaria,Serbia]),
  susedia(bulgaria,Bulgaria,[Macedonia,Greece,Romania,Serbia]),
  susedia(serbia,Serbia,[Montenegro,Albania,Macedonia,Bulgaria,Romania,Hungary,Croatia,Bosnia]),
  susedia(romania,Romania,[Serbia,Bulgaria,Hungary,Moldova,Ukraine]),
  susedia(hungary,Hungary,[Slovenia,Croatia,Serbia,Romania,Slovakia,Austria,Ukraine]),
  susedia(slovakia,Slovakia,[Austria,Hungary,Poland,Czech_republic,Ukraine]),
  susedia(czech_republic,Czech_republic,[Germany,Austria,Slovakia,Poland]),
  susedia(poland,Poland,[Germany,Czech_republic,Slovakia,Sweden,Ukraine,Lithuania,Belarus]),
  susedia(denmark,Denmark,[United_kingdom,Germany,Sweden,Norway]),
  susedia(sweden,Sweden,[Norway,Denmark,Germany,Poland,Finland]),
  susedia(norway,Norway,[United_kingdom,Denmark,Sweden,Finland,Iceland]),
  susedia(finland,Finland,[Sweden,Norway]),
  susedia(Iceland,Iceland,[Ireland,United_kingdom,Norway]),
  susedia(ukraine,Ukraine,[Slovakia,Moldova,Poland,Belarus,Hungary,Romania]),
  susedia(moldova,Moldova,[Ukraine,Romania]),
  susedia(belarus,Belarus,[Poland,Ukraine,Lithuania,Latvia]),
  susedia(lithuania,Lithuania,[Poland,Belarus,Latvia]),
  susedia(estonia,Estonia,[Latvia]),
  susedia(latvia,Latvia,[Estonia,Belarus,Lithuania])
]).
```



colors([green,red,blue,yellow]).

% zoznam farieb

subset([],_).

% všetci sa nachádzajú v...

subset([X|Xs],Ys) :- member(X,Ys), subset(Xs,Ys).

Farbenie mapy2

```
diff(X,[ ]).
diff(X,[Y | Ys]):-X#\=Y,diff(X,Ys).
```

```
Countries = [Portugal, Spain, ...],
Countries :: [1..4],                % obor hodnot
diff(Portugal,[Spain]),
diff(Spain,[Portugal,Andorra,France]),
diff(Andorra,[Spain,France]),
labeling(Countries),                % generovanie moznosti
write(Countries),nl.                % vypis riesenia
```

Portugal = 1
Spain = 2
Andorra = 1
France = 3
United_kingdom = 1
Ireland = 2
Monaco = 1
Italy = 1
San_marino = 2
Switzerland = 2
Liechtenstein = 1
Germany = 1
Belgium = 2
Netherlands = 3

Luxembourg = 4
Austria = 3
Slovenia = 2
Croatia = 3
Bosnia = 1
Montenegro = 2
Albania = 3
Greece = 2
Cyprus = 1
Macedonia = 1
Bulgaria = 3
Serbia = 4
Romania = 2
Hungary = 1

Slovakia = 2
Czech_republic = 4
Poland = 3
Denmark = 2
Sweden = 4
Norway = 3
Finland = 1
Iceland = 4
Ukraine = 4
Moldova = 1
Belarus = 1
Lithuania = 2
Estonia = 1
Latvia = 3



Hra NIM



http://www.archimedes-lab.org/game_nim/nim.html

jednokopový NIM: dvaja hráči postupne berú z kopy 1,2 alebo 3 zápalky, vyhráva ten, kto berie poslednú zápalku (prehráva, kto nemá čo brať)

Rekurzívna definícia predikátov:

- vitaznaPozicia1NIM/1,
- prehravajucaPozicia1NIM/1,
- spravnyTah1NIM

vitaznaPozicia1NIM(N):-between(1,3,N).

vitaznaPozicia1NIM(N):-

tah1NIM(N,N1),

prehravajucaPozicia1NIM(N1). % existuje správny ťah z N do N1,
% že konfigurácia N1 je prehrávajúca

tah1NIM(N,N1):-

between(1,3,Tah),

Tah=<N, N1 is N-Tah.

% existuje konkrétny Tah

% v súlade s pravidlami jednokopového

% NIMu



Hra NIM - prehrávajúca

```
prehravajucaPozicia1NIM(0).                % nemá čo brať
prehravajucaPozicia1NIM(N):-                % pre ľubovoľný ťah z N do N1,
    bagof(vitaznaPozicia1NIM(N1), tah1NIM(N,N1), All),
    forall(All).                             % nová konfigurácia je víťazná

forall([]).                                  % platia všetky podciele zoznamu
forall([G|Gs]):-G, forall(Gs).

spravnyTah1NIM(N):-tah1NIM(N,N1), prehravajucaPozicia1NIM(N1),
    write('nechaj ='), write(N1),nl.
```

zlé pozície sú tvaru $4*k$ ($k>0$), t.j. 4, 8, 12, ... a víťazná stratégia je:
ak super zoberie Z zápaliek, ja beriem $4-Z$, ak som vo víťaznej pozícii,
ak som v prehrávajúcej pozícii, beriem čo najmenej (snáď sa pomýli...).



Hra NIM - trace

Ak si dáme vypisovať, pre ktoré ciele sa forall dokazuje, že platia, vidíme, že overovanie, že pozícia N je víťazná, sa vykonáva mnohokrát.

?- prehravajúcaPozicia1NIM(8).

[vitaznaPozicia1NIM(7), vitaznaPozicia1NIM(6), vitaznaPozicia1NIM(5)]

[vitaznaPozicia1NIM(5), vitaznaPozicia1NIM(4), vitaznaPozicia1NIM(3)]

[vitaznaPozicia1NIM(3), vitaznaPozicia1NIM(2), vitaznaPozicia1NIM(1)]

[vitaznaPozicia1NIM(2), vitaznaPozicia1NIM(1)]

[vitaznaPozicia1NIM(1)]

[]

[vitaznaPozicia1NIM(4), vitaznaPozicia1NIM(3)]

[vitaznaPozicia1NIM(2), vitaznaPozicia1NIM(1), vitaznaPozicia1NIM(0)]

[vitaznaPozicia1NIM(1), vitaznaPozicia1NIM(0)]

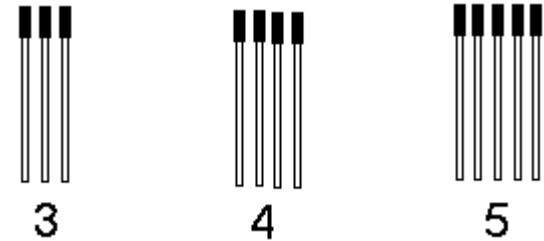
[vitaznaPozicia1NIM(0)]

[vitaznaPozicia1NIM(1), vitaznaPozicia1NIM(0)]

[vitaznaPozicia1NIM(0)]

.

3-kopový NIM



<http://www.mathematische-basteleien.de/nimgame.html>

pravidlá (<http://en.wikipedia.org/wiki/Nim>):

- hráč berie z ľubovoľnej kopy (ale len jednej) ľub.počet zápaliek
- vyhráva ten, kto berie poslednú (t.j. prehráva ten, kto už nemá čo brať).

predikáty:

vitaznaPozicia3NIM/1,
prehravajucaPozicia3NIM/1,
spravnyTah3NIM

%- pravidlá závislé na hre (konfigurácia a povolené ťahy):

vitaznaPozicia3NIM([A,0,0]):-A>0.

vitaznaPozicia3NIM([0,B,0]):-B>0.

vitaznaPozicia3NIM([0,0,C]):-C>0.



Hra 3-NIM - prehrávajúca

%- toto je nezávislé na hre

```
vitaznaPozicia3NIM(N):-tah3NIM(N,N1), % existuje správny ťah z N do N1,  
    lemma(prehravajucaPozicia3NIM(N1)).% že N1 je prehrávajúca
```

```
prehravajucaPozicia3NIM([0,0,0]).
```

```
prehravajucaPozicia3NIM(N):- % pre ľub.ťah, nová konfigurácia je víťazná  
    bagof(lemma(vitaznaPozicia3NIM(N1)), tah3NIM(N,N1), All),  
    forall(All).
```

```
spravnyTah3NIM(N):-tah3NIM(N,N1),  
    lemma(prehravajucaPozicia3NIM(N1)),  
    write('nechaj ='), write(N1),nl.
```

[1,2,3] ☹

- [0,2,3] ☺ -> [0,2,2] ☹
- [1,1,3] ☺ -> [1,1,0] ☹
- [1,0,3] ☺ -> [1,0,1] ☹
- [1,2,2] ☺ -> [0,2,2] ☹
- [1,2,1] ☺ -> [1,0,1] ☹
- [1,2,0] ☺ -> [1,1,0] ☹



Hra 3-NIM – tabelizácia

%- toto je nezávislé na hre

```
vitaznaPozicia3NIM(N):-tah3NIM(N,N1), % existuje správny ťah z N do N1,  
    lemma(prehravajucaPozicia3NIM(N1)).% že N1 je prehrávajúca
```

```
prehravajucaPozicia3NIM([0,0,0]).
```

```
prehravajucaPozicia3NIM(N):- % pre ľub.ťah, nová konfigurácia je víťazná  
    bagof(lemma(vitaznaPozicia3NIM(N1)), tah3NIM(N,N1), All),  
    forall(All).
```

% lemma(P) je predikát logicky ekvivalentný P. Ale výpočtovo optimálnejší: ak sa nám raz podarilo P dokázať, zapamätáme si to ako fakt **P:-true.**, a keď opätovne dokazujeme P, tak sa pozrieme, či sme už P dokazovali.

```
:-dynamic prehravajucaPozicia3NIM/1, vitaznaPozicia3NIM/1.
```

```
lemma(P):-clause(P,true),!  
    ;  
    P, assertz(P:-true).
```

3-NIM a XOR

Prehrávajúca konfigurácia
je taká, že XOR jednotlivých
kôp je 0.

$$3 = 011_2$$

$$4 = 100_2$$

$$5 = 101_2$$

----- XOR

$$010_2$$

správny ťah $3-2 = 1$

$$3 = 0011_2$$

$$7 = 0111_2$$

$$9 = 1001_2$$

----- XOR

$$1101_2$$

správny ťah $9-5 = 4$

$$1 = 001_2$$

$$4 = 100_2$$

$$5 = 101_2$$

----- XOR

$$000_2$$

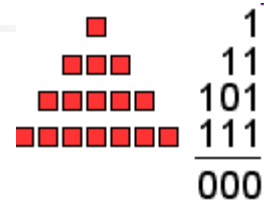
$$3 = 0011_2$$

$$7 = 0111_2$$

$$9 = 0100_2$$

----- XOR

$$0000_2$$



<http://www.numericana.com/answer/games.htm>



Práca s „databázou“

- `asserta(Clause)`, `assertz(Clause)` – pridá klauzulu do databázy, `asserta` na začiatok, `assertz` na koniec procedúry,
- `clause(Head, Body)` – hľadá klauzulu s porovnateľnú s `Head:-Body`,
- `retract(Clause)` – odstráni klauzulu z databázy,
- `retractall(Clause)` – odstráni všetky klauzuly z databázy.

Predikáty spôsobujú side-effect: po backtracku sa ich účinky NEODSTRÁNIA, t.j. pridaná klauzula sa nevymaže a vymazaná sa nepridá.

?-otec(jonatan, izidora).

No.

?-asserta(otec(jonatan,izidora)).

Yes.

?- otec(jonatan,izidora).

Yes.

?-retractall(otec(_, _)).

Yes.

?- otec(jonatan,izidora).

No.

Hurá, máme globálnu
premennú (databázu)



Tabelizácia

```
lemma(P):-write('zistujem '), write(P), nl, fail.
```

```
lemma(P):-(clause(P,true),write('nasiel '), write(P), nl, !)
```

```
;
```

```
(P,write('dokazal '), write(P), nl,  
assertz(P:-true)).
```

% deklarácia, že predikát fib/2 bude modifikovaný.

:-dynamic fib/2.

```
fib(N,1):-N<2,!
```

```
fib(N,F):-N1 is N-1, N2 is N-2,
```

```
lemma(fib(N1,F1)),
```

```
lemma(fib(N2,F2)),
```

```
F is F1+F2.
```

```
?- fib(5,N).
```

```
zistujem fib(4, _G342)
```

```
zistujem fib(3, _G345)
```

```
zistujem fib(2, _G348)
```

```
zistujem fib(1, _G351)
```

```
dokazal fib(1, 1)
```

```
zistujem fib(0, _G357)
```

```
dokazal fib(0, 1)
```

```
dokazal fib(2, 2)
```

```
zistujem fib(1, _G369)
```

```
nasiel fib(1, 1)
```

```
dokazal fib(3, 3)
```

```
zistujem fib(2, _G381)
```

```
nasiel fib(2, 2)
```

```
dokazal fib(4, 5)
```

```
zistujem fib(3, _G393)
```

```
nasiel fib(3, 3)
```

```
N = 8.
```