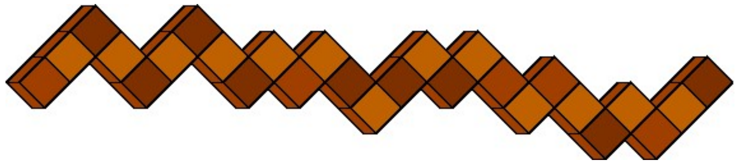
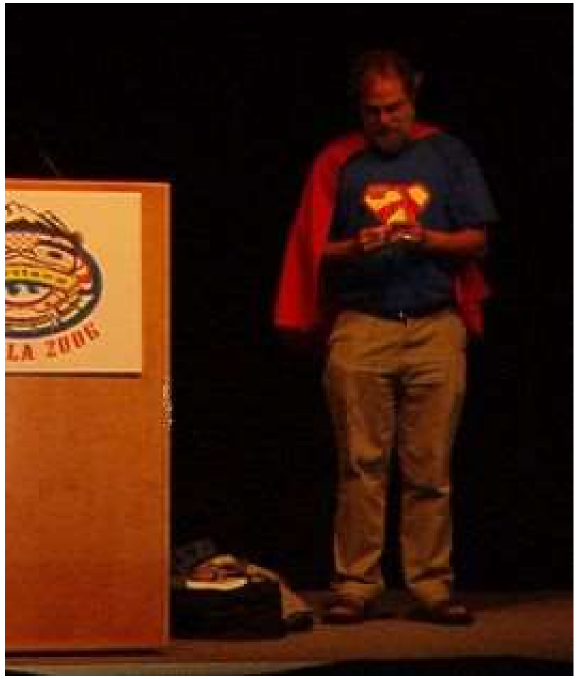
Funkcionálne programovanie 3

Peter Borovanský, KAI, I-18,
borovan(a)ii.fmph.uniba.sk

- parametrický polymorfizmus na príkladoch funkcionálov (map, filter, foldl, foldr)
- backtracking (ako príklad na list-comprehension)



Cvičenie:
- funkcionály (map, filter, foldr, …)
- backtracking v Haskelli,



Phil Wadler, λ-man

V týždni od 30.11.

Funkcia je hodnotou - zatiaľ len argumentom

učíme sa z Prelude.hs:
štandardná knižnica haskellu obsahuje množstvo:
- užitočných funkcií,
- vzorových funkcií.

- zober zo zoznamu tie prvky, ktoré spĺňajú bool-podmienku (test)

booleovská podmienka príde ako argument funkcie a má typ (a -> Bool):

```
filter          :: (a -> Bool) -> [a] -> [a]     -- ( ) nie sú zbytočné
filter p xs     = [ x | x <- xs, p x ]
```

```
> filter even [1..10]
[2,4,6,8,10]
```

- rozdeľ zoznam na zoznam menších, rovných a väčších prvkov ako prvý: riešenie menej efektívne ale v istom zmysle elegantnejšie

```
tripivot (x:xs)  = ( filter (<x) xs, filter (==x) xs, filter (>x) xs )
```

Funkcia (predikát) argumentom

učíme sa z Prelude.hs:

- ber zo zoznamu prvky, kým platí logická podmienka (test):

```
takeWhile                        :: (a -> Bool) -> [a] -> [a]
takeWhile p []                   = []
takeWhile p (x:xs) | p x         = x : takeWhile p xs
                   | otherwise   = []
```

> takeWhile (>0) [1,2,-1,3,4]
[1,2]

- vyhoď tie počiatočné prvky zoznamu, pre ktoré platí podmienka:

```
dropWhile                        :: (a -> Bool) -> [a] -> [a]
dropWhile p []                   = []
dropWhile p xs@(x:xs')  | p x    = dropWhile p xs'
                        | otherwise = xs
```

> dropWhile (>0) [1,2,-1,3,4]
[-1,3,4]

Príklad (porozdeľuj)

Definujte porozdeluj :: (a -> Bool) -> [a] -> [[a]] , ktorá rozdelí zoznam na podzoznamy, v ktorých súvisle platí podmienka daná 1. argumentom

porozdeluj (>0) [1,2,0,3,4,5,-1,6,7] = [[1,2],[3,4,5],[6,7]]
porozdeluj (\x -> x `mod` 3 > 0) [1..10] = [[1,2],[4,5],[7,8],[10]].

```
porozdeluj p [] = []
porozdeluj p xs =
        (takeWhile p xs) :            -- prefix, kým platí p je prvým prvkom
            porozdeluj p              -- rekurzívne volanie na ďalšie prvky
                (dropWhile (\x -> (not (p x))) -- odstráň, kým neplatí p
                    (dropWhile p xs))          -- odstráň, kým platí p
```

```
Main> porozdeluj (>0) [1,2,0,0,3,4,-1,5]
[[1,2],[3,4],[5]]
```

(not . p)

Funktor map

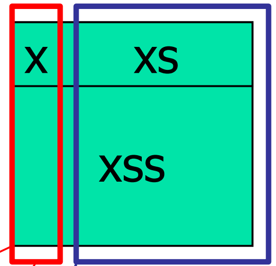
- funktor, ktorý aplikuje funkciu (1.argument) na všetky prvky zoznamu

```
map                    :: (a->b) -> [a] -> [b]
map f []               = []
map f (x:xs)           = f x : map f xs
-- alebo map f xs      = [f x | x <-xs]
```

- Príklad použitia:

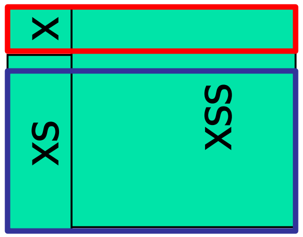
```
map (+1) [1,2,3,4,5]                          = [2,3,4,5,6]
map odd  [1,2,3,4,5]                          = [True,False,True,False,True]
and (map odd  [1,2,3,4,5])                    = False
all p xs = and (map p xs)     -- all p xs = p platí pre všetky prvky zoznamu xs

map head [ [1,0,0], [2,1,0], [3,0,1] ]        = [1, 2, 3]
map tail [ [1,0,0], [2,1,0], [3,0,1] ]        = [ [0,0], [1,0], [0,1] ]
map (0:) [[1],[2],[3]]                        = [[0,1],[0,2],[0,3]]
map (++[0]) [[1],[2],[3]]                     = [[1,0],[2,0],[3,0]]

                    \x->x++[0]
```

Transponuj maticu




```
-- transponuj pomocou map, nie list-comprehension

transponuj                 :: Matica -> Matica

transponuj []              = []
transponuj  ([]:xss)       = transponuj xss
transponuj ((x:xs):xss)    = (x:(map head xss)):
                                  (transponuj (xs:(map tail xss)))
```

```
-- riešenie z minulej prednášky
transpose []               = []
transpose ([] : xss)       = transpose xss
transpose ((x:xs) : xss)   = (x : [h | (h:t) <- xss]) :
                                  transpose (xs : [t | (h:t) <- xss])
```



Ďalšie (známe) funkcionály – (foldr – schéma rekurzie na zoznamoch)

```
foldr            :: (a -> b -> b) -> b -> [a] -> b
foldr f z []       = z
foldr f z (x:xs) = f x (foldr f z xs)

a : b : c : [] -> f a (f b (f c z))
```

```
    :                      f
   / \                    / \
  a   :    foldr f z     a   f
     / \   -------->        / \
    b   :                  b   f
       / \                    / \
      c  []                  c   z
```

```
Main> foldr (+) 0 [1..100]
5050
```

```
     :                  10*y+x
    / \                  / \
   1   :   foldr f z   1 10*y+x
      / \  -------->      / \
     2   :               2  10*y+x
        / \                  / \
       3  []                3   0
```

```
Main> foldr (\x y->10*y+x) 0 [1,2,3]
321
```

nepomenovaná analógia fcie: pom x y = 10*y+x

-- g je vnorená lokálna funkcia

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr f z = g
  where   g []     = z
          g (x:xs) = f x (g xs)
```

Ďalšie (známe) funkcionály -
(foldl – schéma iterácie na zoznamoch)

```
foldl            :: (a -> b -> a) -> a -> [b] -> a
foldl f z []        =  z
foldl f z (x:xs) =  foldl f (f z x) xs

a : b : c : [] -> f (f (f z a) b) c
```

```
   :                     f
  / \                   / \
 a   :   foldl f z     f   c
    / \   ------>     / \
   b   :             f   b
      / \           / \
     c  []         z   a
```

```
Main> foldl (+) 0 [1..100]
5050

Main> foldl (\x y->10*x+y) 0 [1,2,3]
123
```

```
   :                          10*x+y
  / \                         /    \
 1   :   foldl f z     10*x+y       3
    / \   ------>       /    \
   2   :          10*x+y      2
      / \          /    \
     3  []        0      1
```

Vypočítajte

- foldr max (-999) [1,2,3,4]
  foldl max (-999) [1,2,3,4]

- foldr (_ -> \y ->(y+1)) 0 [3,2,1,2,4]
  foldl (\x -> _ ->(x+1)) 0 [3,2,1,2,4]

- foldr (-) 0 [1..100] =

(1-(2-(3-(4-...-(100-0))))) = 1-2 + 3-4 + 5-6 + ... + (99-100) = -50

- foldl (-) 0 [1..100] =

  (...(((0-1)-2)-3) ... - 100) = -5050

Funkcia je hodnotou

- [a->a] je zoznam funkcií typu a->a
  napríklad: [(+1),(+2),(*3)] je [\x->x+1,\x->x+2,\x->x*3]

- čo je foldr (.) id [(+1),(+2),(*3)] ??
  akého je typu
  foldr (.) id [(+1),(+2),(*3)] 100
  foldl (.) id [(+1),(+2),(*3)] 100

[a->a]
303
???

lebo skladanie fcií je asociatívne:

- ((f . g) . h) x = (f . g) (h x) = f (g (h x)) = f ((g . h) x) = (f . (g . h)) x

- mapf :: [a->b] -> [a] -> [b]   -- rovnako dlhé zoznamy
  mapf [] _ = []
  mapf _ [] = []
  mapf (f:fs) (x:xs) = (f x):mapf fs xs

[11,22,33]

mapf [(+1),(+2),(+3)] [10,20,30]

Kvíz

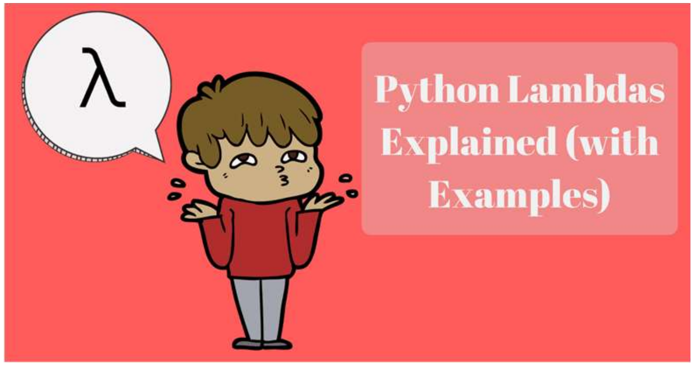
foldr (:) [] xs =   xs

foldr (:) ys xs =   xs++ys

foldr ? ? xs = reverse xs

foldl ? ? xs = reverse xs

Python Kvíz




```
print(map(lambda x: x*x, [1,2,3,4,5]))
print(list(map(lambda x: x*x, [1,2,3,4,5])))
print(list(filter(lambda y:y>10,map(lambda x: x*x, [1,2,3,4,5]))))
```

<map object at 0x037
[1, 4, 9, 16, 25]
[16, 25]

```
from functools import reduce
print(reduce((lambda x, y: x * y), [1, 2, 3, 4]))
```

24

```
print(reduce((lambda x, y: x + y), [1, 2, 3, 4]))
print(reduce((lambda x, y: x - y), [1, 2, 3, 4]))
```

10
-8

```
def compose(f, g):
        return lambda x: f(g(x))
print(compose( lambda x: x+1, lambda x: x*3 )(10))
```

31

```
def composeMany(*fs):
        return reduce(compose, fs)
print(composeMany(lambda x:x+1, lambda x:x+2, lambda x:x*3)(10))
```

33

lambdas.hs

All Things Pythonic

Guido van Rossum: The fate of reduce() in Python 3000, (r.2005)

- Python aquired lambdas, reduce(), filter() and map() thanks a Lisp hacker

- despite of the PR value, I think these features should be cut from Python 3

- **Update: lambda, filter and map will stay (the latter two with small changes, returning iterators instead of lists). Only reduce will be removed from the 3.0 standard library. You can import it from functools.**

https://www.artima.com/weblogs/viewpost.jsp?thread=98196

Priemerný prvok

Ak chceme vypočítať aritmetický priemer (a-priemer) prvkov zoznamu, matice, ... potrebujeme poznať ich súčet a počet. Ako to urobíme na jeden prechod štruktúrou pomocou foldr/foldl ? …počítame dvojicu hodnôt, súčet a počet:

- priemerný prvok zoznamu

```
priemer  xs = sum/count where (sum, count) = sumCount xs
sumCount xs = foldr (\x -> \(sum, count) -> (sum+x, count+1)) (0, 0) xs
```

- priemerný prvok matice

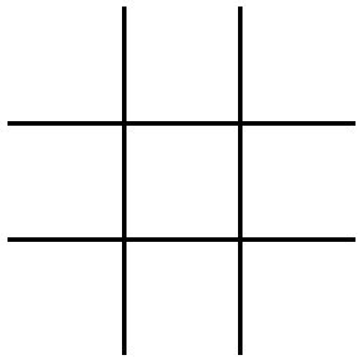
  je a-priemer a-priemerov riadkov matice a-priemerom hodnôt matice ?

```
sumCount'          :: [[Float]] -> (Float,Float)
sumCount' xs =
    foldr (\x -> \(sum, count)->scitaj (sumCount x) (sum, count)) (0, 0) xs
    where scitaj (a,b) (c,d) = (a+c, b+d)

priemer' :: [[Float]] -> Float
priemer' = uncurry (/) . sumCount'
```

```
uncurry :: (a->b->c) -> (a,b) -> c
uncurry f (a,b) = f a b
curry    :: ((a,b) -> c) -> (a->b->c)
curry g a b     = g (a,b)
```

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

Riešenie „brute force“
(s ním prežijete len pri smiešne jednoduchých úlohách)

```
subset' ys xs = and (map (\x -> elem x xs) ys)
subset'' xs ys = all (`elem` ys) xs

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)
```

vygenerujeme všetky podmnožiny [0..8] a vyberieme správne dĺžky 6

```
powerSet          :: [Int] -> [[Int]]
powerSet  []      = [[]]
powerSet  (x:xs)  = map (x:) ps ++ ps  where ps = powerSet  xs

solve3x3 ' = filter (\x -> 6 == length x) (filter isOk (powerSet [0..8]))
```

Riešenie „kombinatorické“

(s ním prežijete len pri smiešne jednoduchých úlohách)

generujeme len 6 prvkové kombinácie [0..8] a z nich vyberieme správne

```
kbo                 :: [Int] -> Int -> [[Int]]
kbo _ 0             = [[]]
kbo [] _            = []
kbo (x:xs) k        = [x:ys | ys <- kbo xs (k-1)] ++ kbo xs k

solve3x3''          = filter isOk (kbo [0..8] 6)
```

generujeme len správne 6 prvkové kombinácie [0..8]

Backtracking = testujeme správnosť riešenia už počas jeho tvorby

```
kbo'                :: [Int] -> Int -> [[Int]]
kbo' _ 0            = [[]]
kbo' [] _           = []
kbo' (x:xs) k       = [x:ys | ys <- kbo' xs (k-1), isOk (x:ys)] ++ kbo' xs k

solve3x3            = kbo' [0..8] 6
```

Backtracking
magické číslo

```
?  381654729
2| 38
3| 381
4| 3816
5| 38165
6| 381654
7| 3816547
8| 38165472
9| 381654729
```

číslo s neopakujúcimi sa ciframi 1..9, ktorého prvých i-cifier je delitelných i

```
-- (čiastočným) riešením je zoznam cifier od konca, t.j. [3,2,1] je číslo 123
type Riesenie = [Int]                    -- typ riešenia

-- nájdi všetky magické riešenia, ktoré obsahujú predpísané cifry
magicBacktrack          :: [Int] -> [Riesenie]        -- hľadáme zoznam
magicBacktrack []= [[]]                               -- riešení
magicBacktrack cifry    = [ c:ciastocneRiesenie     |
                            c<-cifry,
                            ciastocneRiesenie<-magicBacktrack (cifry \\ [c]),
                            jeMagicke (c:ciastocneRiesenie) ]

-- jeMagicke je test, ktorý overí konzistentnosť nového čiastočného riešenia
```

Backtracking
jeMagické

```
jeMagicke                       :: Riesenie -> Bool
jeMagicke cs                    = jeMagicke' 1 0 (reverse cs)

jeMagicke'                      :: Int->Int->Riesenie -> Bool
jeMagicke' _ _ []               = True
jeMagicke' i stareCislo (cifra:cs)
                                = noveCislo `mod` i == 0
                                        &&
                                  jeMagicke' (i+1) noveCislo cs
                                where
                                  noveCislo = 10*stareCislo+cifra
```

```
Main> magicBacktrack [1,2,3,4,5,6,7,8,9]
[[9,2,7,4,5,6,1,8,3]]
```

t.j. jediné riešenie je
381654729

opäť jeMagicke

Definujme pomocou schém rekurzie foldr a iterácie foldl

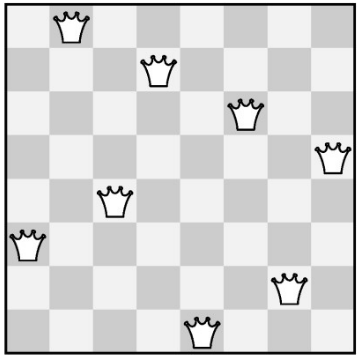
foldr:

```
jeMagicke''         :: Riesenie -> Bool
jeMagicke'' xs      = res
        where (_,_,res) =
        foldr (\cifra -> \(count, cislo, result) ->
                          (count+1, 10*cislo+cifra,
                          result  && ( (10*cislo+cifra) `mod` (count+1) == 0)))
              (0, 0, True)
              xs
```

foldl: na cvičení...

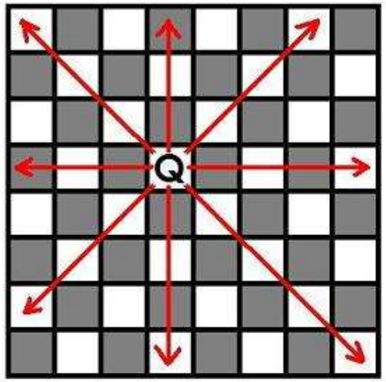
```
Main> jeMagicke'' [9,2,7,4,5,6,1,8,3]
True
```

Backtracking
8 dám na šachovnici



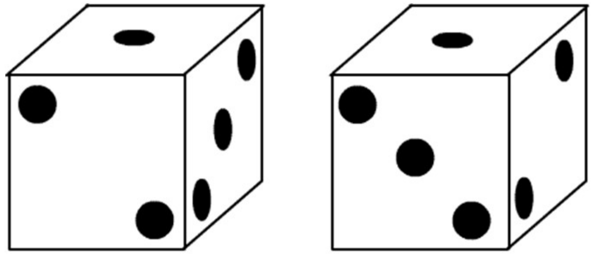
```
-- definujeme typ popisujúci riešenie problému
type RiesenieDam = [Int]           -- pozície dám v stĺpcoch 1,2,3, ..., n
                                   -- príklad riešenia [3,1,6,2,5,7,4,0]
                                   -- hľadáme permutáciu [0..N-1], [0..7]
-- definujeme rekurzívnu funkciu, ktorá vráti zoznam všetkých riešení
damyBacktrack          :: Int -> [RiesenieDam] -- všetky riešenia,
                       -- argument Int určuje, ako ďaleko v "riešení" sme

damyBacktrack 0        = [[]]      -- jedno triviálne riešenie pre 0x0
damyBacktrack (n+1)    = [          -- začiatok list-comprehension
    dama:ciastocneRiesenie |       -- nové riešenie daj do zoznamu riešení

        ciastocneRiesenie <- damyBacktrack n, -- rekurzívne volanie nájde
                                              -- čiastočné riešenie
        dama <- [0..7],                       -- rozšírime čiastočné riešenie
                                              -- o ďalšiu dámu na nové r.
        damyOk n dama ciastocneRiesenie]      -- otestuj, či je to ešte riešenie
```

Backtracking
(ťažké: orezávanie stromu možností)

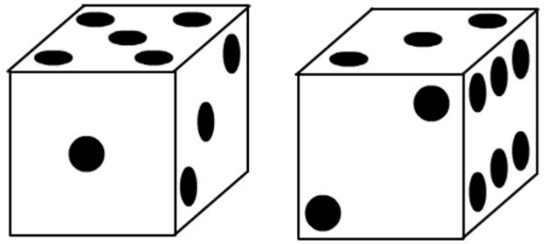



```
damyOk                          :: Int -> Int -> RiesenieDam -> Bool

damyOk n ndama ciastocneRiesenie          = and [    -- dámy sú ok, ak žiadna
    not (damaVOhrozeni ndama stlpec ciastocneRiesenie) |   -- z nich nie je
    stlpec <- [0..(n-1)] ]                  -- v ohrození, pre všetky už položené dámy


damaVOhrozeni                   :: Int-> Int-> RiesenieDam->Bool
                                                  -- dáma je v ohrození
damaVOhrozeni ndama stlpec ciastocneRiesenie  =   -- od dámy v stĺpec
    (ndama==ciastocneRiesenie!!stlpec) ||        -- ak sú v rovnakom riadku
    (abs(ndama - ciastocneRiesenie!!stlpec)==stlpec+1)  -- alebo diagonále

Main> damy
[[3,1,6,2,5,7,4,0],[4,1,3,6,2,7,5,0],[2,4,1,7,5,3,6,0],[2,5,3,1,7,4,6,0]
```

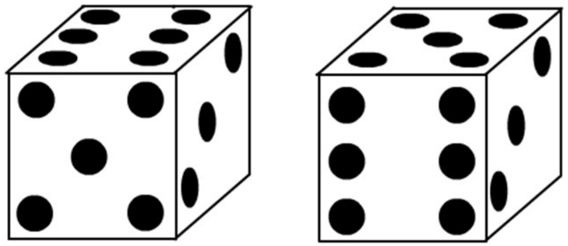
Kubológia
(v teórii)



Otestujte sa, toto je ktorá ?



A toto ?



- pravé hracie kocky majú súčet protiľahlých 7
- už asi vieme, že existujú dve
  - left-handed a
  - right-handed
- kocky

ZOBERME LEN JEDEN Z TYPOV LH/RH, ľubovoľne

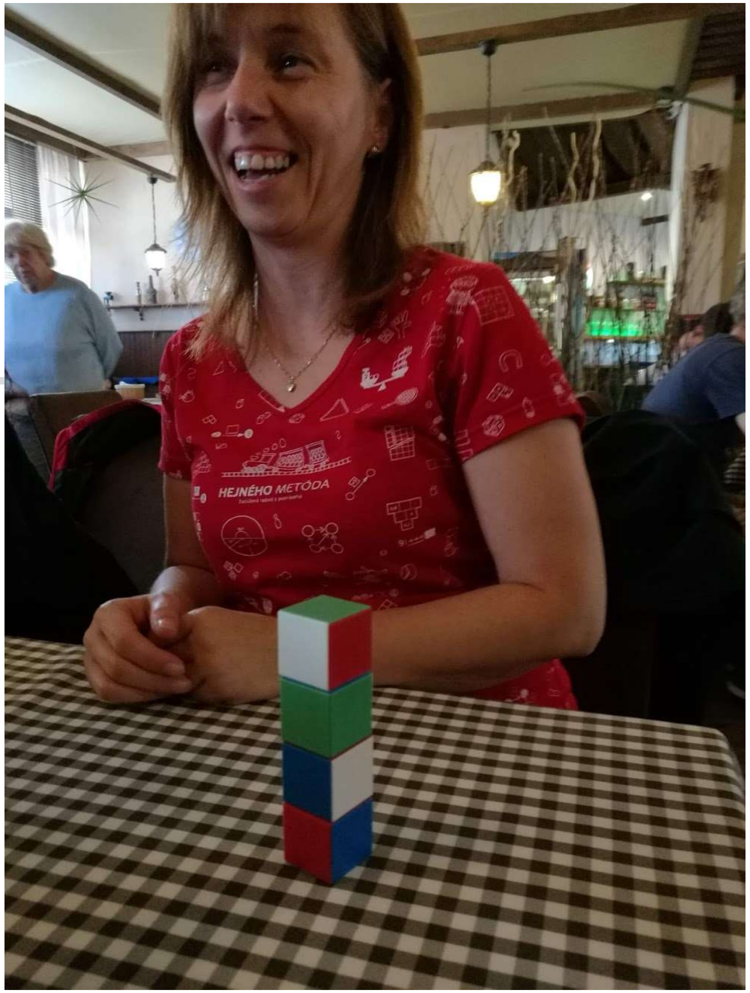
- koľko je možných natočení kocky ?
- koľko kociek vieme postaviť na seba,
  aby žiadne bočné steny takejto veže kociek
  neobsahovali dve rovnaké čísla.
- vieme

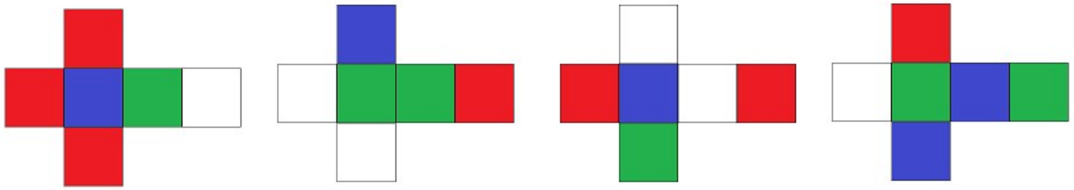
  3? 4? 5? 6? 7?

Kto bude vedieť zajtra odpoveď, má body

Kubológia
(v praxi)

- Instant Insanity (okamžité šialenstvo)
- https://en.wikipedia.org/wiki/Instant_Insanity



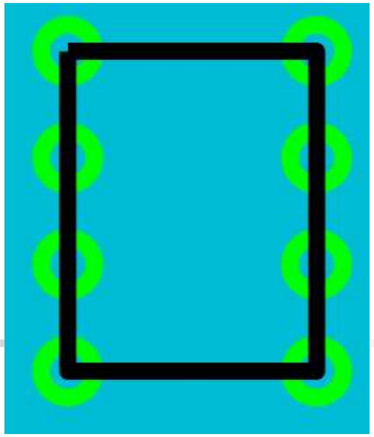


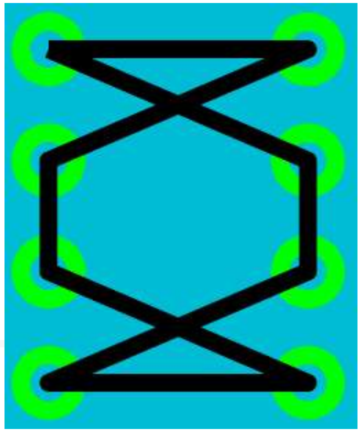
cube 1 cube 2 cube 3 cube 4

- Toto je zadanie, vidíte 4 plášte kocky
- BTW, koľko je rôznych plášťov kocky ?
- A čo vlastne je plášť kocky ? Dá niekto definíciu ?
- A čo sú dva rovnaké plášte kocky ?
- Trúfne si to niekto vysloviť, ako definíciu, trochu hodnú matfyzu ?
- Skúsi niekto dať riešenie, bohužial kocky si musíte doma ofarbiť sami, z niečoho...

- Instant Insanity je hlavolam bol síce patentovaný 1900, ale v 1967 sa ho predalo 12mil.
- Na porovnanie, Rubikovej kocky sa predalo >350 mil. kusov

Šnúrkológia



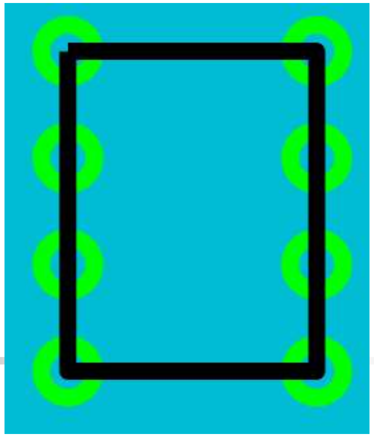


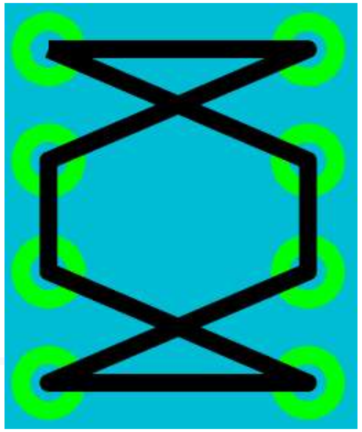
očíslujme si dierky [1..n] a [-1..-n], teda [-n..-1], alebo [-1,-2..(-n)]
hľadáme zoznam dĺžky 2n

```
lubovolne n = map(1:) $ permutations ([2..n]++[-1,-2..(-n)])

pocet_lubovolne n = ((product [1..2*n])`div` n) `div` 2
```

```
length $ lancing 3 = 120
length $ lancing 4 = 5040
length $ lancing 6 = 39916800
```

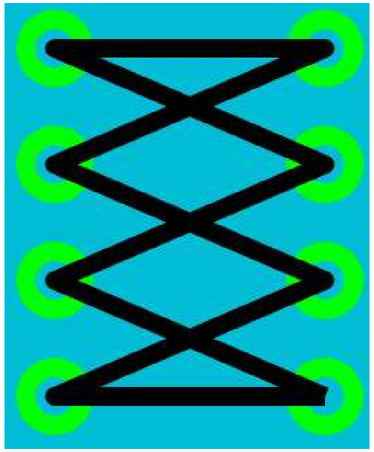
Šnúrkológia

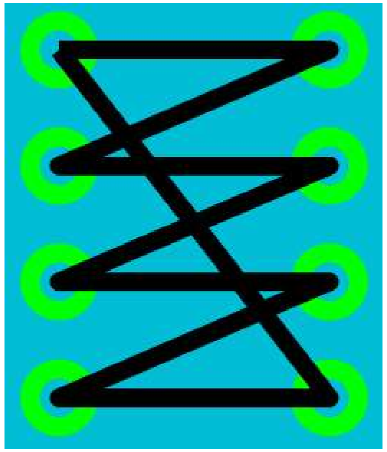




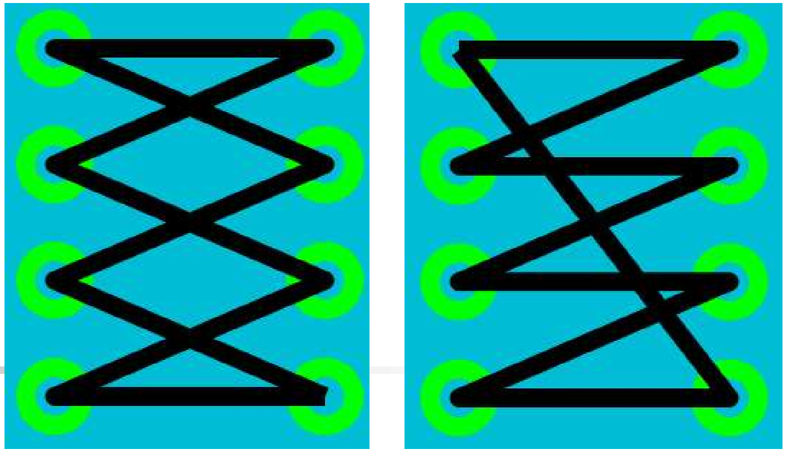
Dierku volajme

- **0-cross**, ak z nej ani jeden koniec šnúrky nejde do opačného radu dierok.
  - na prvom obrázku sú 4 dierky 0-cross.
- **1-cross** volajme takú, že práve jedna z dvoch šnúrok smeruje z dierky do opačného radu, a druhá nie.
  - na druhom obrázku sú 4 dierky 1-cross.
- **2-cross** je dierka, z ktorej oba konce smerujú do opačného radu dierok.
  - na obrázkoch sú len dierky 2-cross.





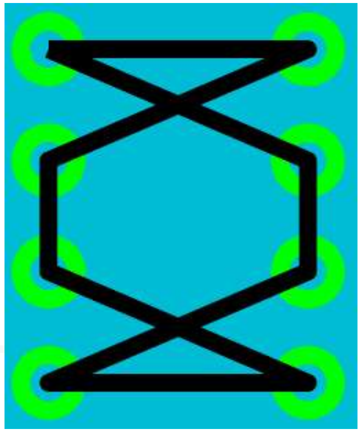
Šnúrkológia



```
isTwoCrossed   :: [Int] -> Bool
isTwoCrossed s = undefined…?????????????????????

twoCrossed n = filter isTwoCrossed (lancing n)
distinctTwoCrossed n = nubBy same (twoCrossed n)  where
                                       same s1 s2 = reverse (tail s1) == tail s2

count_TwoCrossed 1 = 1
count_TwoCrossed n = (fact (n-1) * fact n) `div`2
```

```
length $ distinctTwoCrossed 1 = 1
length $ distinctTwoCrossed 2 = 1
length $ distinctTwoCrossed 3 = 6
length $ distinctTwoCrossed 4 = 72
length $ distinctTwoCrossed 5 = 1440
```

Šnúrkológia

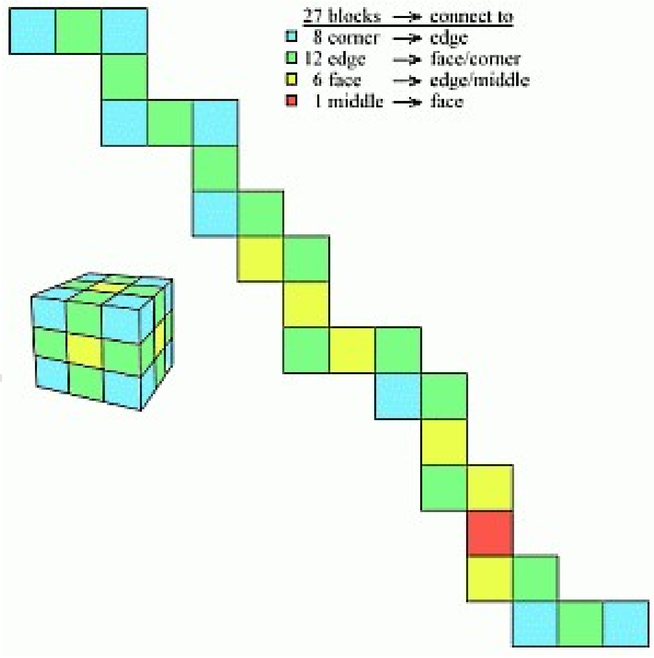


- Dierku volajme
  - **1-cross** volajme takú, že práve jedna z dvoch šnúrok smeruje z dierky do opačného radu, a druhá nie.

- Koľko ich je a aké sú ?
  - existuje explicitná formula ?
  - Resp. ako prebrať všetky možnosti
  - backtracking...

```
countOneCrossed 1 = 0
countOneCrossed 2 = 3
countOneCrossed 3 = 42
countOneCrossed 4 = 1080
countOneCrossed 5 = 51840
countOneCrossed 6 = 3758400
countOneCrossed 7 = 382838400
countOneCrossed 8 = 52733721600
countOneCrossed 9 = 9400624128000
countOneCrossed 10 = 2105593491456000
countOneCrossed 11 = 579255485276160000
```

Drevený had



- ako na to ?
- popíšeme hada dĺžkami jednotlivých „rebier"

```
type Had  = [Int]
had1      :: Had
had1      = [3,3,3,3,2,2,2,3,3,2,2,3,2,3,2,2,3]
```

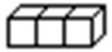
- počet „kĺbov" = počet čiarok v zozname = `length(mysnake)-1 = 16`
- každý kĺb je započítaný 2x, preto

  `(sum mysnake) - ((length mysnake)-1) == 27`
- každý kĺb má 4 možné polohy, takže naivne 4^{16} = 4.294.967.296 možností
- ak si kocku predstavíme v 3-rozmernom priestore, každé "rebro" má smer niektorého z vektorov [±1,0,0], [0,±1,0], [0,0,±1]
- v kĺbe sa had môže=musí zohnúť o 90°, ako vyzerá smer ďalšieho rebra ?
- kolmé vektory k vektoru [±1,0,0] sú [0,±1,0] [0,0,±1,]

  k vektoru [0,±1,0] sú [±1,0,0] [0,0,±1,]

  k vektoru [0,0,±1] sú [±1,0,0] [0,±1,0]

Let’s code

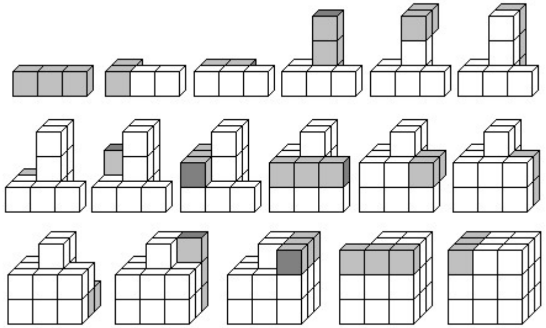


- kolmé vektory k vektoru

```
type SVektor  = (Int,Int,Int)        -- dx, dy, dz
kolme          :: SVektor -> [SVektor]
kolme (_,0,0)  = [(0,1,0),(0,-1,0),(0,0,1),(0,0,-1)]
kolme (0,_,0)  = [(1,0,0),(-1,0,0),(0,0,1),(0,0,-1)]
kolme (0,0,_)  = [(0,1,0),(0,-1,0),(1,0,0),(-1,0,0)]
```

- test, či kocička je v kocke 3x3x3 (ľahko parametrizovateľné pre 4x4x4):

```
type Pozicia          = (Int,Int,Int)       -- x, y, z
vKocke3               ::  Pozicia -> Bool
vKocke3 (x,y,z)       = x `elem` [1..3] &&
                        y `elem` [1..3] &&
                        z `elem` [1..3]
```



Let's code

- ďalšia kocička v danom smere má súradnice

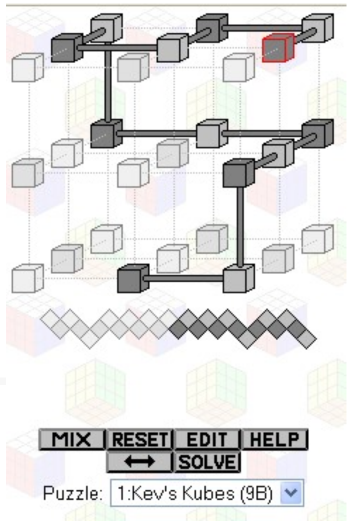
```
nPoz          :: Pozicia -> SVektor -> Pozicia
nPoz (x,y,z) (dx,dy,dz) = (x+dx,y+dy,z+dz)
```

- kocičky celého rebra danej dĺžky (2, 3, event. 4)

```
type Rebro          = [Pozicia]
rebro               :: Pozicia -> SVektor -> Int -> Rebro
rebro start smer 2  = [nPoz start smer]
rebro start smer 3  = [nPoz (nPoz start smer) smer,
                       nPoz start smer]
-- pre 4x4x4 ešte pridáme:
rebro start smer 4  = [nPoz(nPoz(nPoz start smer) smer) smer,
                       nPoz (nPoz start smer) smer,
                       nPoz start smer]
```

Zlož hada

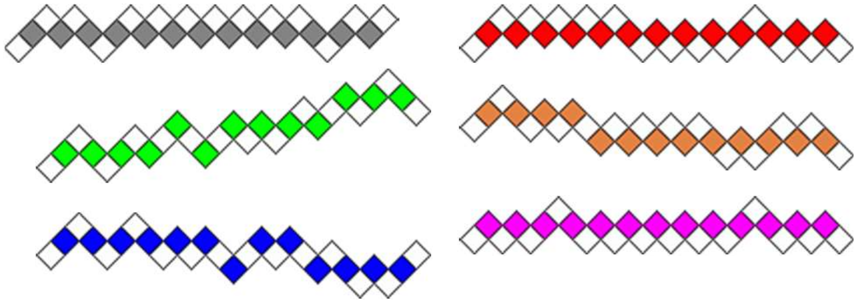
http://www.jaapsch.net/puzzles/javascript/snakecubej.htm



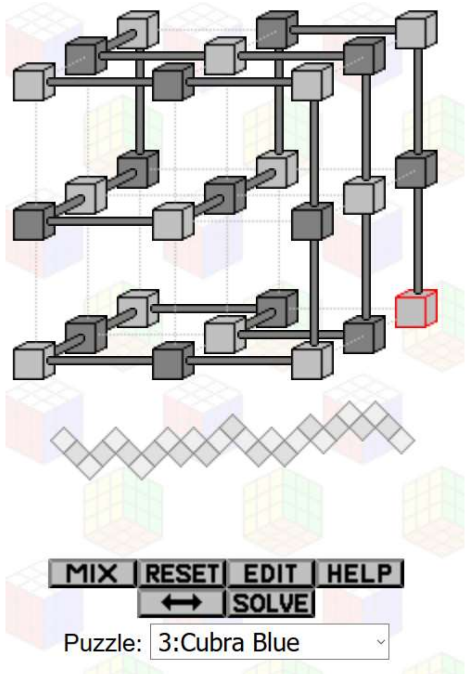

```
zloz [[(1,1,1)]] (0,0,1) had1
```

```
zloz :: Solution -> SVektor -> Had -> [Solution]
zloz ciastocne smer []  = [ciastocne]
zloz ciastocne smer (len:rebra) =
  [ riesenie |
      kolmySmer <- kolmo smer,
      koniecHada <- [head (head ciastocne)],
      noveRebro <- [rebro koniecHada kolmySmer len],
      all vKocke3 noveRebro,
      all (`notElem` concat ciastocne) noveRebro,
      riesenie <-zloz (noveRebro:ciastocne) kolmySmer rebra
  ]
```

Výsledky 3x3x3

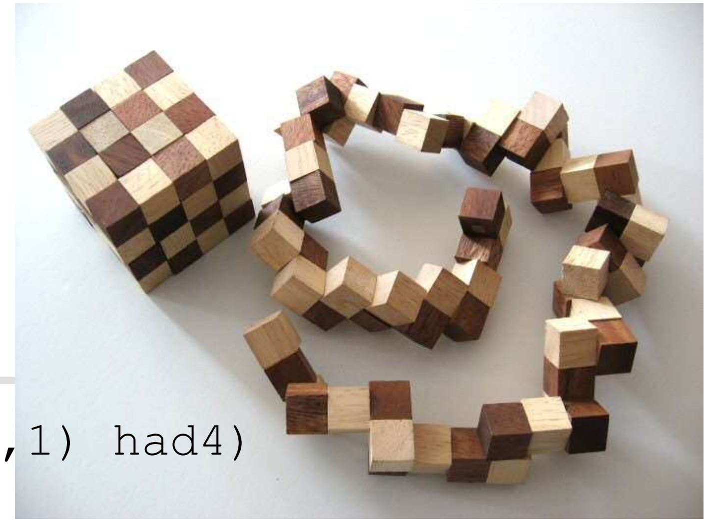


```
Main> head(zloz3 [[(1,1,1)]] (0,0,1) had1)
[[(3,3,3),(2,3,3)],[(1,3,3)],[(1,2,3)],[(2,2,3),(2,2,2)],[(2,2,1)
  ],[(2,1,1),(2,1,2)],[(2,1,3)],[(1,1,3)],[(1,1,2),(1,2,2)],[(1,3
  ,2),(2,3,2)],[(3,3,2)],[(3,2,2)],[(3,2,3)],[(3,1,3),(3,1,2)],[(
  3,1,1),(3,2,1)],[(3,3,1),(2,3,1)],[(1,3,1),(1,2,1)],[(1,1,1)]]
=== 4x4x4
```



https://www.jaapsch.net/puzzles/javascript/snakecubej.htm

Výsledky 4x4x4



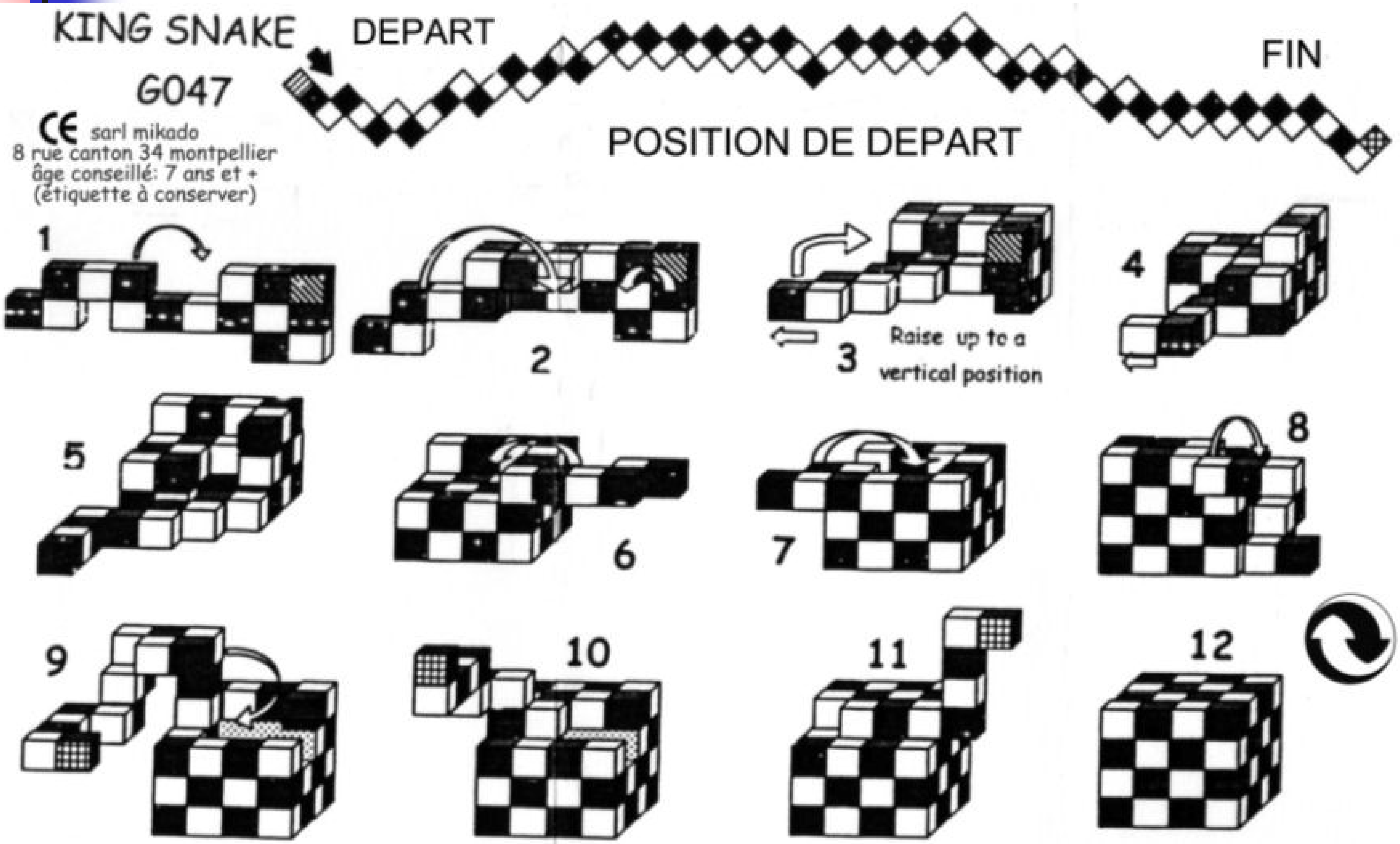
```
*Main> head (zloz4 [[(2,1,1)]] (0,0,1) had4)
```

```
*Main> head (zloz4 [[(2,1,1)]] (0,0,1) had4)
[[(1,4,2)],[(1,4,1),(1,3,1),(1,2,1)],[(1,1,1)],[(1,1,2)],[(2,1,2)],[(2,2,2)],[(1
,2,2)],[(1,3,2)],[(2,3,2)],[(2,3,3)],[(2,2,3)],[(1,2,3),(1,3,3)],[(1,4,3)],[(1,4
,4),(1,3,4),(1,2,4)],[(1,1,4)],[(1,1,3)],[(2,1,3)],[(2,1,4),(2,2,4),(2,3,4)],[(2
,4,4),(3,4,4)],[(4,4,4)],[(4,3,4)],[(3,3,4)],[(3,3,3)],[(4,3,3)],[(4,4,3),(3,4,3
)],[(2,4,3),(2,4,2)],[(2,4,1)],[(3,4,1)],[(3,4,2)],[(4,4,2)],[(4,4,1)],[(4,3,1)]
,[(4,3,2)],[(3,3,2)],[(3,2,2),(3,2,3)],[(3,2,4)],[(3,1,4),(3,1,3)],[(3,1,2)],[(4
,1,2),(4,1,3)],[(4,1,4)],[(4,2,4),(4,2,3),(4,2,2)],[(4,2,1)],[(4,1,1)],[(3,1,1),
(3,2,1)],[(3,3,1)],[(2,3,1),(2,2,1)],[(2,1,1)]]
(130.85 secs, 41,684,701,792 bytes)
```

```
*Main> head (zloz4 [[(1,1,1)]] (0,0,1) had4)
[[(4,4,4)],[(4,3,4),(3,3,4),(2,3,4)],[(1,3,4)],[(1,4,4)],[(1,4,3)],[(1,3,3)],[(2
,3,3)],[(2,4,3)],[(2,4,4)],[(3,4,4)],[(3,4,3)],[(4,4,3),(4,3,3)],[(4,2,3)],[(4,2
,4),(3,2,4),(2,2,4)],[(1,2,4)],[(1,2,3)],[(1,1,3)],[(1,1,4),(2,1,4),(3,1,4)],[(4
,1,4),(4,1,3)],[(4,1,2)],[(3,1,2)],[(3,1,3)],[(2,1,3)],[(2,2,3)],[(2,2,2),(2,3,2
)],[(2,4,2),(3,4,2)],[(4,4,2)],[(4,4,1)],[(4,3,1)],[(4,3,2)],[(4,2,2)],[(4,2,1)]
,[(4,1,1)],[(3,1,1)],[(3,2,1),(3,2,2)],[(3,2,3)],[(3,3,3),(3,3,2)],[(3,3,1)],[(3
,4,1),(2,4,1)],[(1,4,1)],[(1,4,2),(1,3,2),(1,2,2)],[(1,1,2)],[(2,1,2)],[(2,1,1),
(2,2,1)],[(2,3,1)],[(1,3,1),(1,2,1)],[(1,1,1)]]
(422.73 secs, 158,282,403,568 bytes)
```

http://collection.cassetete.free.fr/1_bois/cube_elastique3x3/cube_elastique_3x3.htm

http://www.jbd-jouetsenbois.com/king-snake-xml-380_385-1238.html



25 N or Y pentacubes in 5x5x5

25 N pentacubes can be packed into a 5x5x5 box in only 4 different ways without considering symmetric solutions.

The process to assemble one of these solutions is displayed in the following pictures.

https://rtomas.web.cern.ch/rtomas/Npentacubes/ http://kociemba.org/themen/125puzzle/solutions.html