Rekurze

IB111 Úvod do programování skrze Python

2012

Rekurze

- použití funkce při její vlastní definici
- volání sebe sama (s jinými parametry)

Rekurze a sebe-reference

klíčové myšlenky v informatice

některé souvislosti:

- matematická indukce
- funkcionální programování
- rekurzivní datové struktury
- gramatiky
- logika, neúplnost
- nerozhodnutelnost, diagonalizace

Faktoriál

$$n! = 1 \cdot 2 \cdots (n-1) \cdot n$$

$$f(n) = \begin{cases} 1 & \text{if } n = 1 \\ n \cdot f(n-1) & \text{if } n > 1 \end{cases}$$

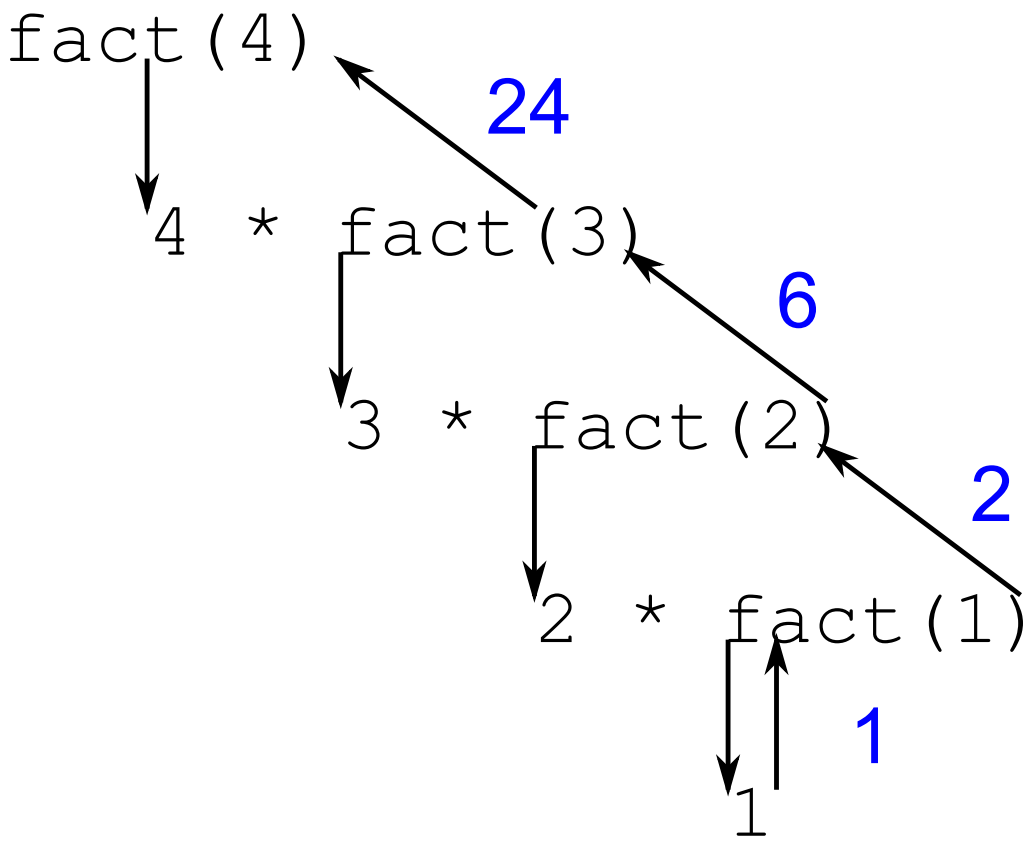
Faktoriál iterativně (pomocí cyklu)

```
def fact(n):
    f = 1
    for i in range(1,n+1):
        f = f * i
    return f
```

Faktoriál rekurzivně

```
def fact(n):
    if n == 1: return 1
    else: return n * fact(n-1)
```

Faktoriál rekurzivně – ilustrace výpočtu



Příklad: výpis čísel

Vymyslete funkci, která:

- vypíše čísla od 1 do N
- pomocí rekurze – bez použití cyklů `for`, `while`

Příklad: výpis čísel

Vymyslete funkci, která:

- vypíše čísla od 1 do N
- pomocí rekurze – bez použití cyklů `for`, `while`

```
def vypis(n):
    if n > 1:
        vypis(n-1)
    print n
```

Co udělá tento program?

```
def test(n):
    print n
    if n > 1:
        test(n-1)
    print -n

test(5)
```

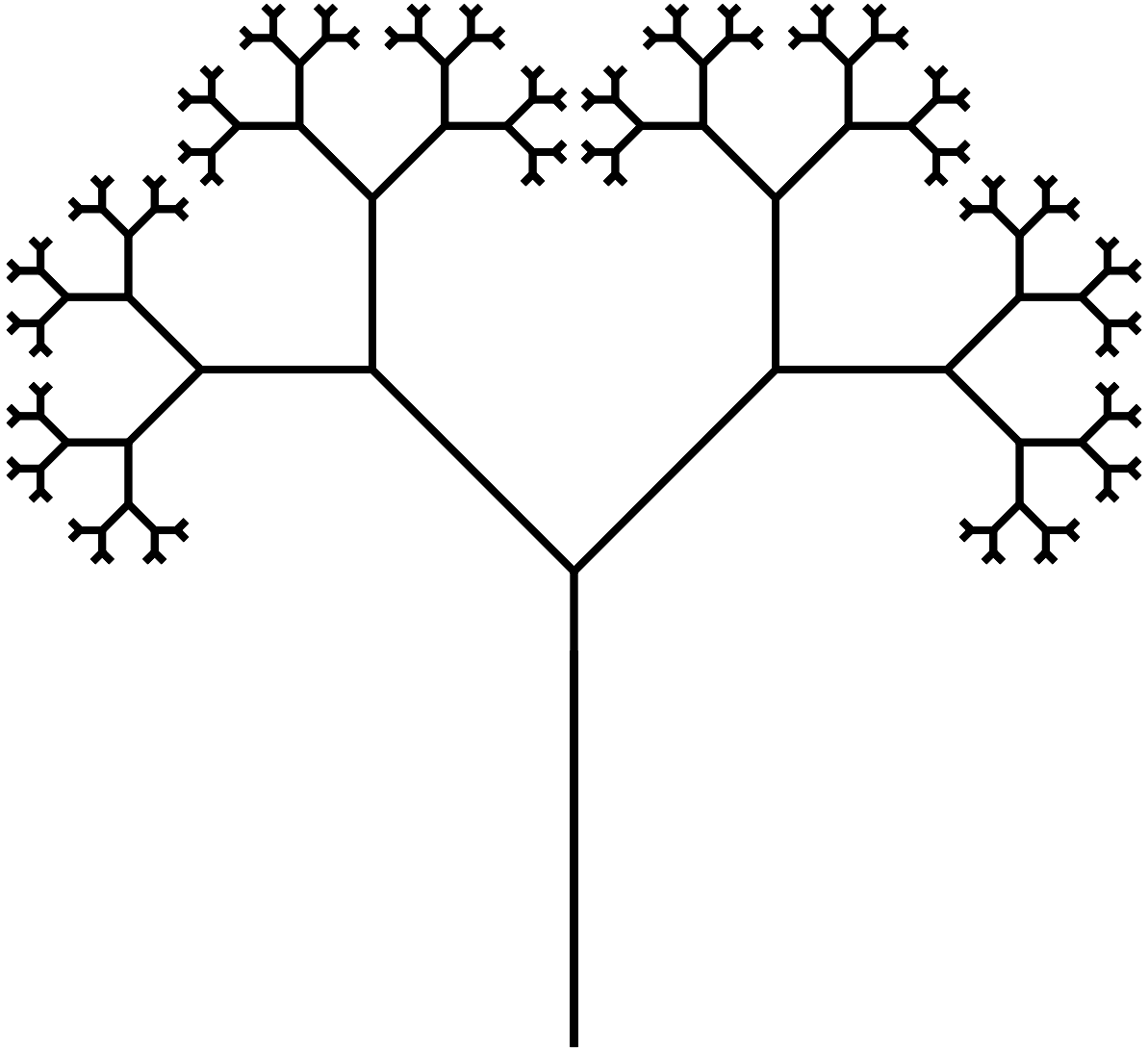
Nepřímá rekurze

```
def suda(n):
    print "suda", n
    licha(n-1)

def licha(n):
    print "licha", n
    if n > 1:
        suda(n-1)

suda(10)
```

Rekurzivní stromeček



Rekurzivní stromeček

nakreslit stromeček znamená:

- udělat stonek
- nakreslit dva menší stromečky

Stromeček želví grafikou

```
def stromecek(delka):
    forward(delka)
    if delka > 10:
        left(45)
        stromecek(0.6 * delka)
        right(90)
        stromecek(0.6 * delka)
        left(45)
    back(delka)
```

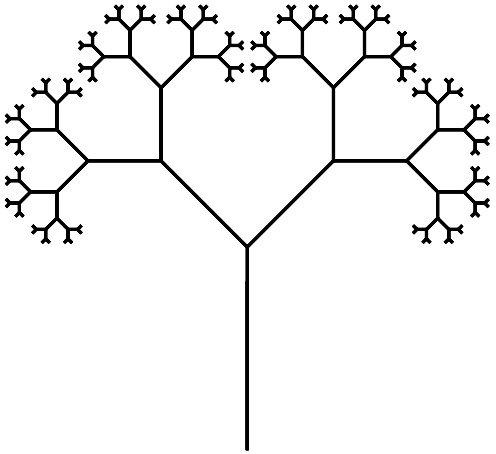
Sierpińského fraktál

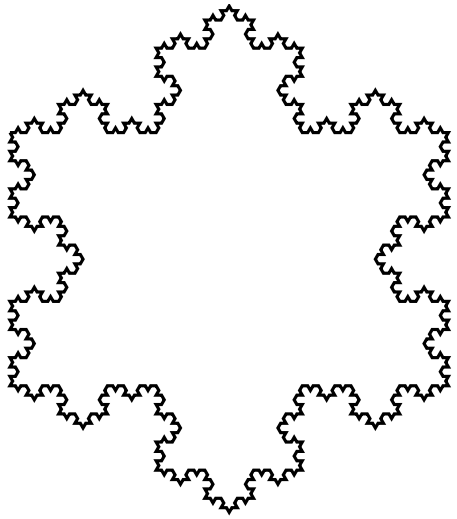
rekurzivně definovaný geometrický útvar

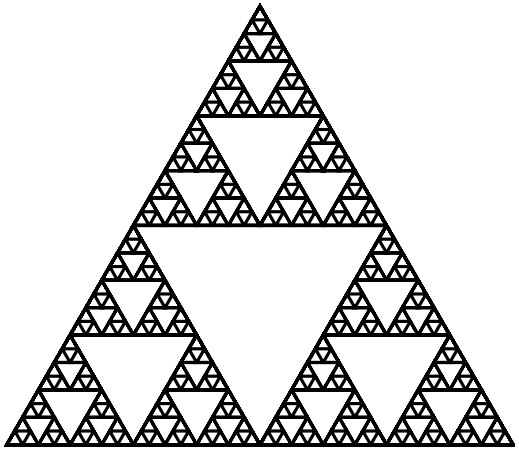
Sierpińského fraktál: kód

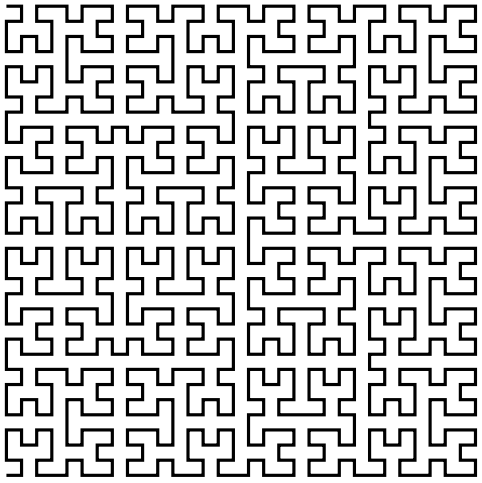
```
def sierpinski(n, delka):
    if n == 1:
        trojuhelnik(delka)
    else:
        for i in range(3):
            sierpinski(n - 1, delka)
            forward((2 ** (n - 1)) * delka)
            right(120)
```

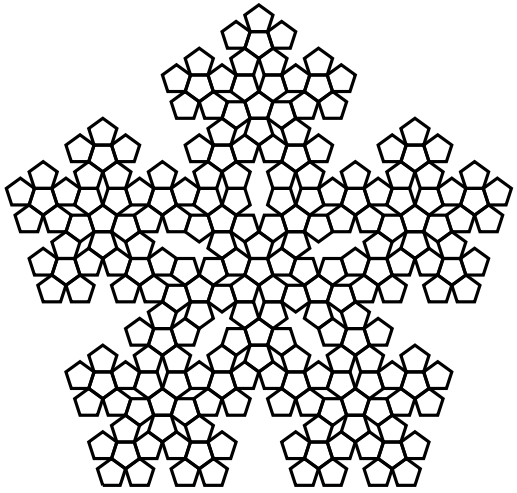
Další podobné fraktály

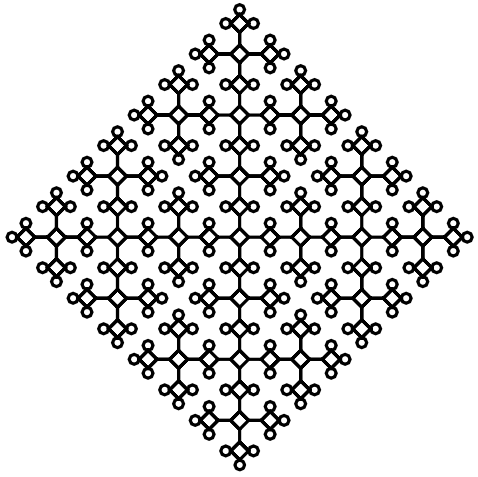








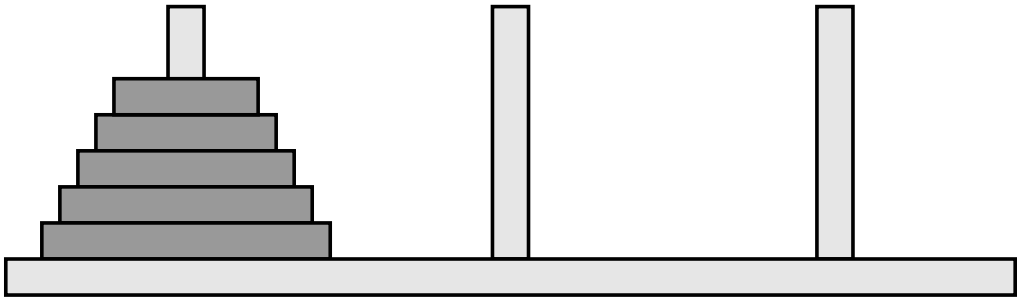




Hanojské věže aneb O konci světa

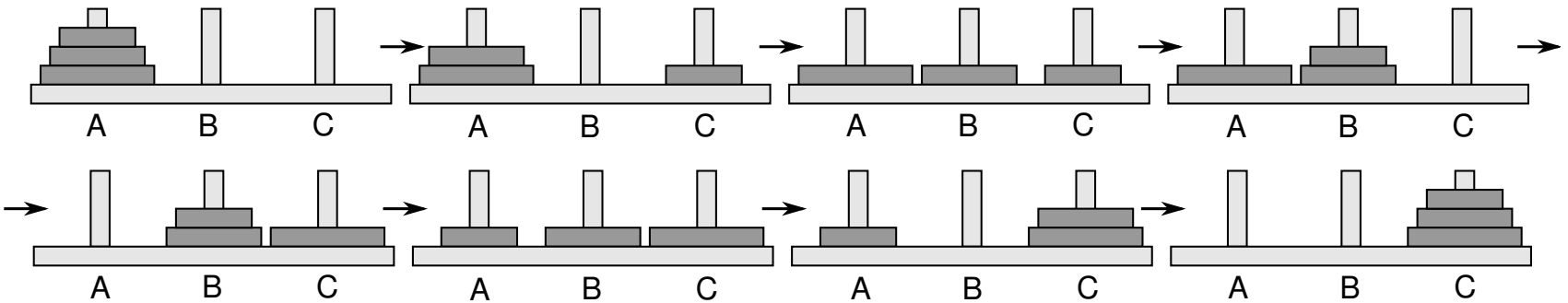
- video:
  `http://www.fi.muni.cz/~xpelanek/IB111/hanojske_veze/`
- klášter kdesi vysoko v horách u města Hanoj
- velká místnost se třemi vyznačenými místy
- 64 různě velkých zlatých disků
- podle věštby mají mniši přesouvat disky z prvního na třetí místo
- a až to dokončí ...

Hanojské věže: pravidla

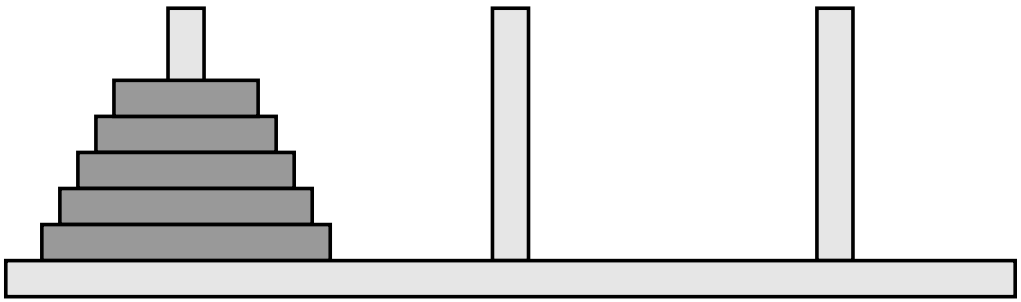


- N disků různých velikostí naskládaných na sobě
- vždy může být jen menší disk položen na větším
- možnost přesunout jeden horní disk na jiný kolíček
- cíl: přesunout vše z prvního na třetí

Hanojské věže: řešení



Hanojské věže: rekurzivní řešení



```
def presun(n, odkud, kam, kudy):
    if n == 1:
        print odkud, "->", kam
    else:
        presun(n-1, odkud, kudy, kam)
        presun(1, odkud, kam, kudy)
        presun(n-1, kudy, kam, odkud)
```

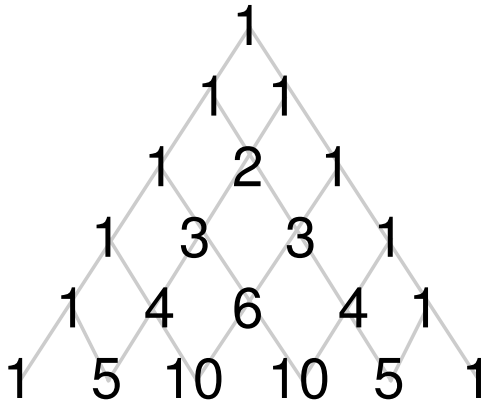
Hanojské věže: použití programu

```
>>> presun(3, "A", "B", "C")
A -> B
A -> C
B -> C
A -> B
C -> A
C -> B
A -> B
```

Odbočka: nečekané souvislosti

- Hanojské věže
- fraktál: Sierpińského košík
- Pascalův trojúhelník

Pascalův trojúhelník



$$\binom{0}{0}$$
$$\binom{1}{0}\ \binom{1}{1}$$
$$\binom{2}{0}\ \binom{2}{1}\ \binom{2}{2}$$
$$\binom{3}{0}\ \binom{3}{1}\ \binom{3}{2}\ \binom{3}{3}$$

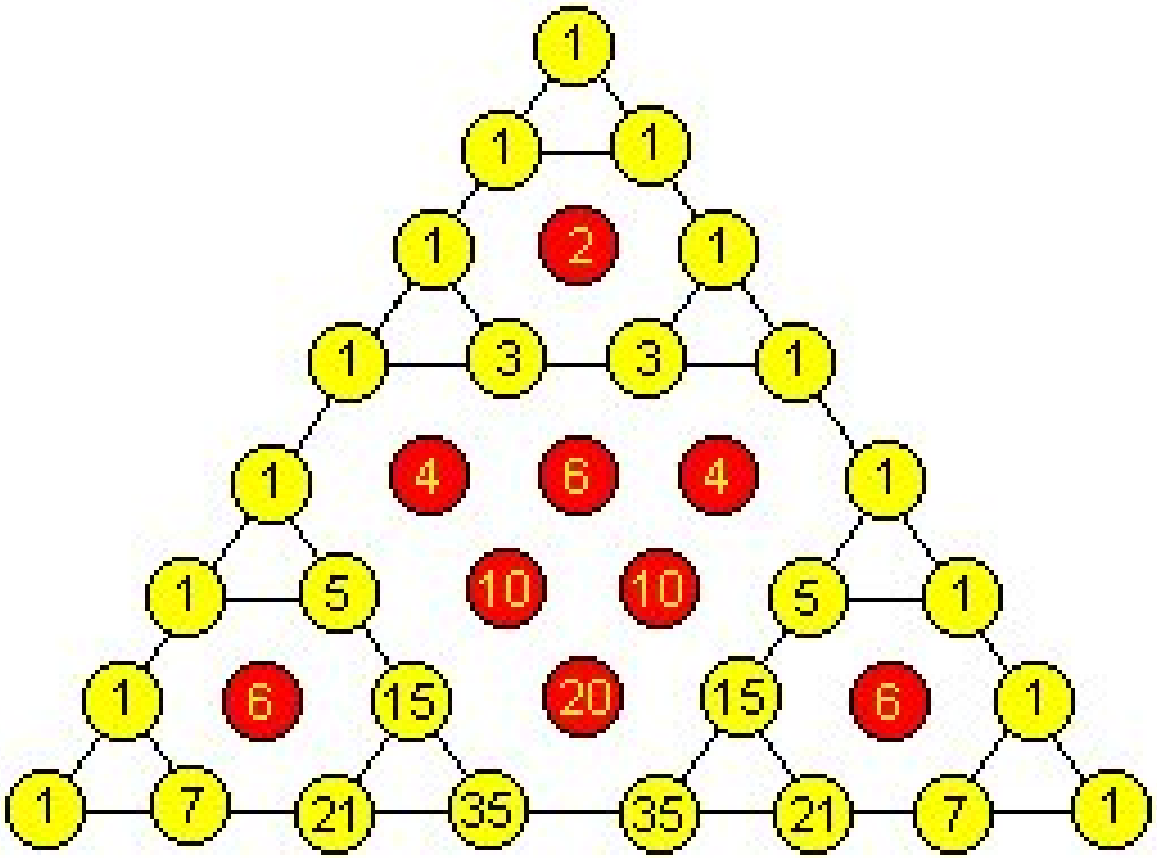
Explicitní vzorec

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

Rekurzivní vztah

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

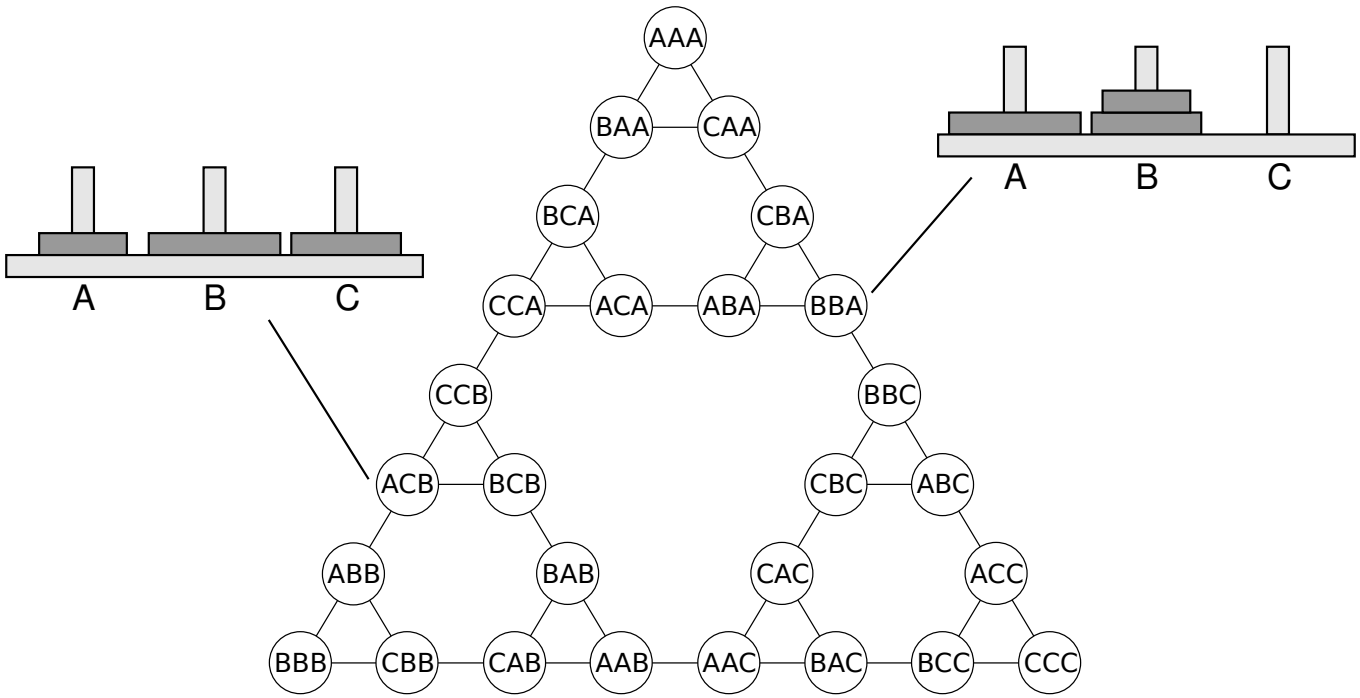
Pascalův trojúhelník a Sierpińského košík



Co to má společného s Hanojskými věžemi?

- jak vypadá stavový prostor úlohy?
- připomenutí, stavový prostor:
  - graf (uzly + hrany)
  - uzly = konfigurace hry
  - hrany = povolené tahy
- jak vypadá stavový prostor pro 1 disk? pro 2 disky?

Hanojské věže: stavový prostor



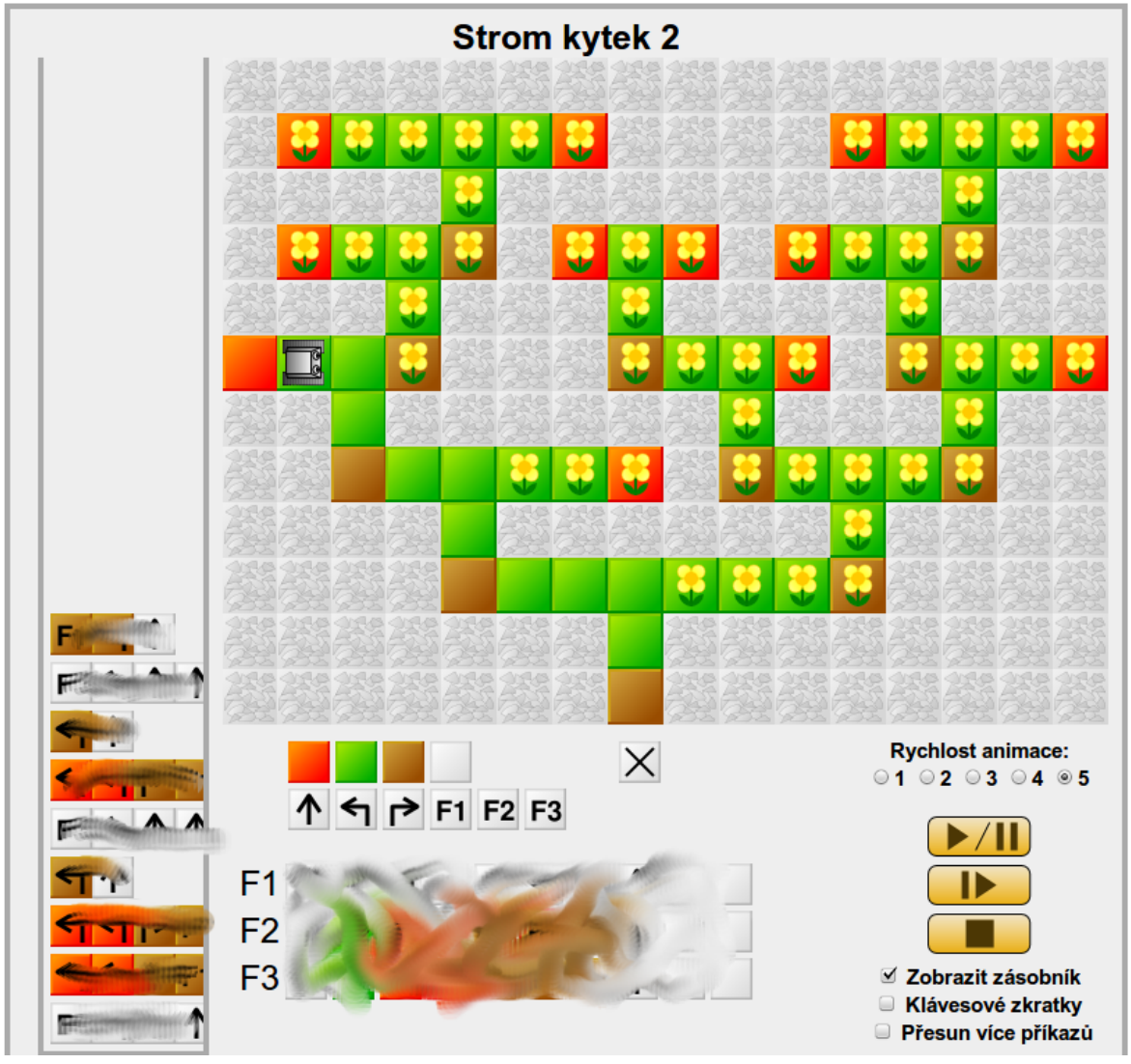
Hanojské věže: variace

- přesun z obecné konfigurace do obecné konfigurace
- více než 3 kolíky

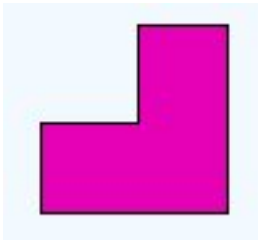
Robotanik

- `tutor.fi.muni.cz`, úloha Robotanik
- jednoduché „grafické“ programování robota
- těžší příklady založeny na rekurzi
- vizualizace průběhu „výpočtu“, zanořování a vynořování z rekurze

Robotanik

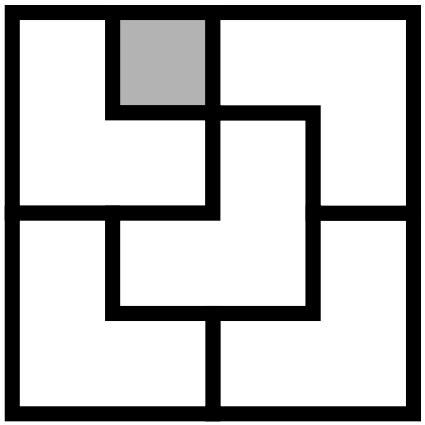

Strom kytek 2
F1
F2
F3
Rychlost animace:
1 2 3 4 5
Zobrazit zásobník
Klávesové zkratky
Přesun více příkazů

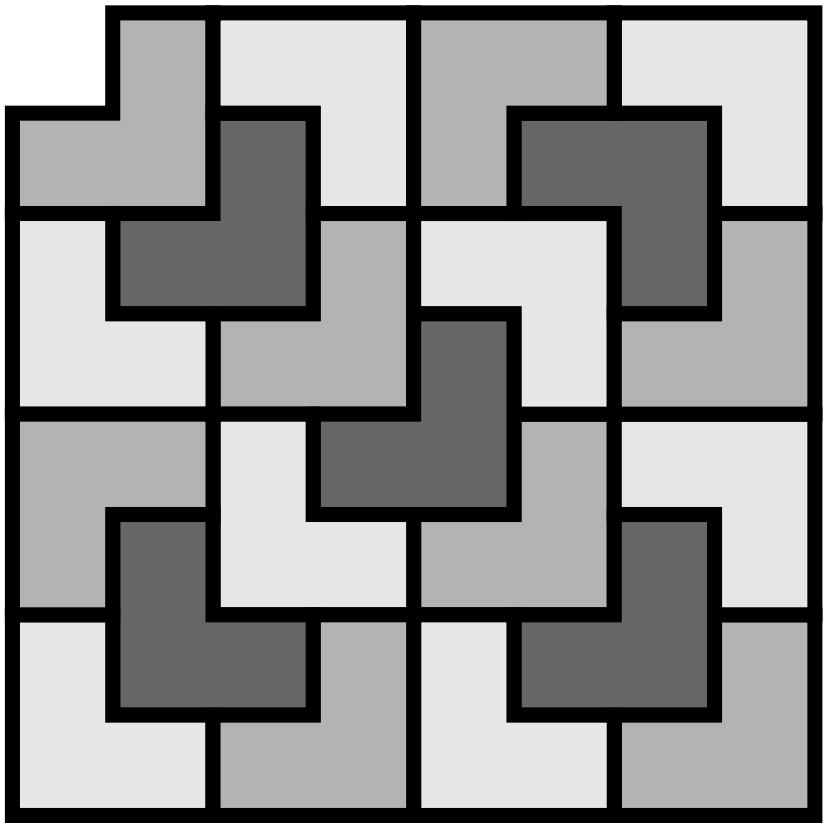
Pokrývání plochy L kostičkami



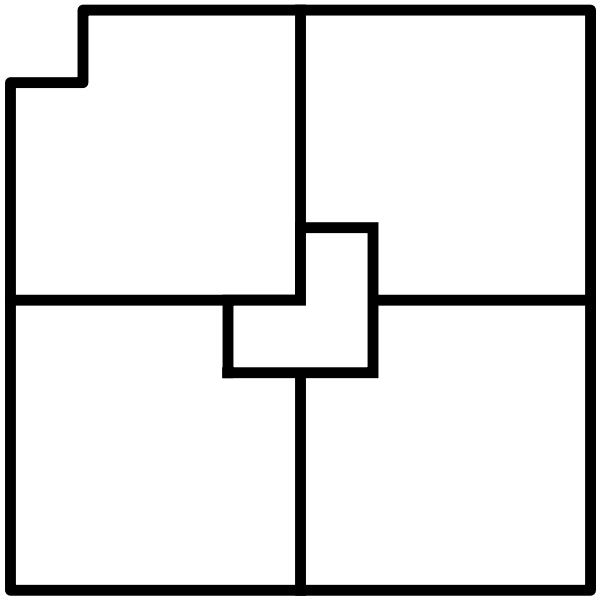
- mřížka 8×8 s chybějícím levým horním polem
- úkol: pokrýt zbývající políčka pomocí L kostiček
- rozšíření:
  - rozměr $2^n \times 2^n$
  - chybějící libovolné pole
  - obarvení 3 barvami, aby sousedi byli různí

Ukázky řešení





Řešení



- rozdělit na čtvrtiny
- umístit jednu kostku
- rekurzivně aplikovat řešení na jednotlivé části

Příklady použití rekurze v informatice

- Euclidův algoritmus – NSD
- vyhledávání opakovaným půlením
- řadicí algoritmy (quicksort, mergesort)
- generování permutací, kombinací
- fraktály
- prohledávání grafu do hloubky
- gramatiky

Euklidův algoritmus rekurzivně

```
def nsd(a,b):
    if b == 0:
        return a
    else:
        return nsd(b, a % b)
```

Vyhledávání opakovaným půlením

- hra na 20 otázek
- hledání v seznamu
- hledání v binárním stromu

Vyhledávání: rekurzivní varianta

```
def binarni_vyhledavani(hodnota, seznam,
                        spodni_mez, horni_mez):
    if spodni_mez > horni_mez:
        return False
    stred = (spodni_mez + horni_mez)/2
    if seznam[stred] < hodnota:
        return binarni_vyhledavani(hodnota, seznam,
                                   stred+1, horni_mez)
    elif seznam[stred] > hodnota:
        return binarni_vyhledavani(hodnota, seznam,
                                   podni_mez, stred-1)
    else:
        return True
```

Řadicí algoritmy

- quicksort
  - vyber pivota
  - rozděl na menší a větší
  - zavolej quicksort na podčásti
- mergesort
  - rozděl na polovinu
  - každou polovinu seřaď pomocí mergesort
  - spoj obě poloviny

Generování permutací, kombinací

- permutace množiny = všechna možná pořadí
  - příklad: permutace množiny $\{1, 2, 3, 4\}$
  - jak je vypsat systematicky?
  - jak využít rekurzi?
- k-prvkové kombinace n-prvkové množiny = všechny možné výběry k prvků
  - příklad: 3-prvkové kombinace množiny $\{A, B, C, D, E\}$
  - jak je vypsat systematicky?
  - jak využít rekurzi?

Nevhodné použití rekurze

- ne každé použití rekurze je efektivní
- Fibonacciho posloupnost (králíci):

$$f_1 = 1$$

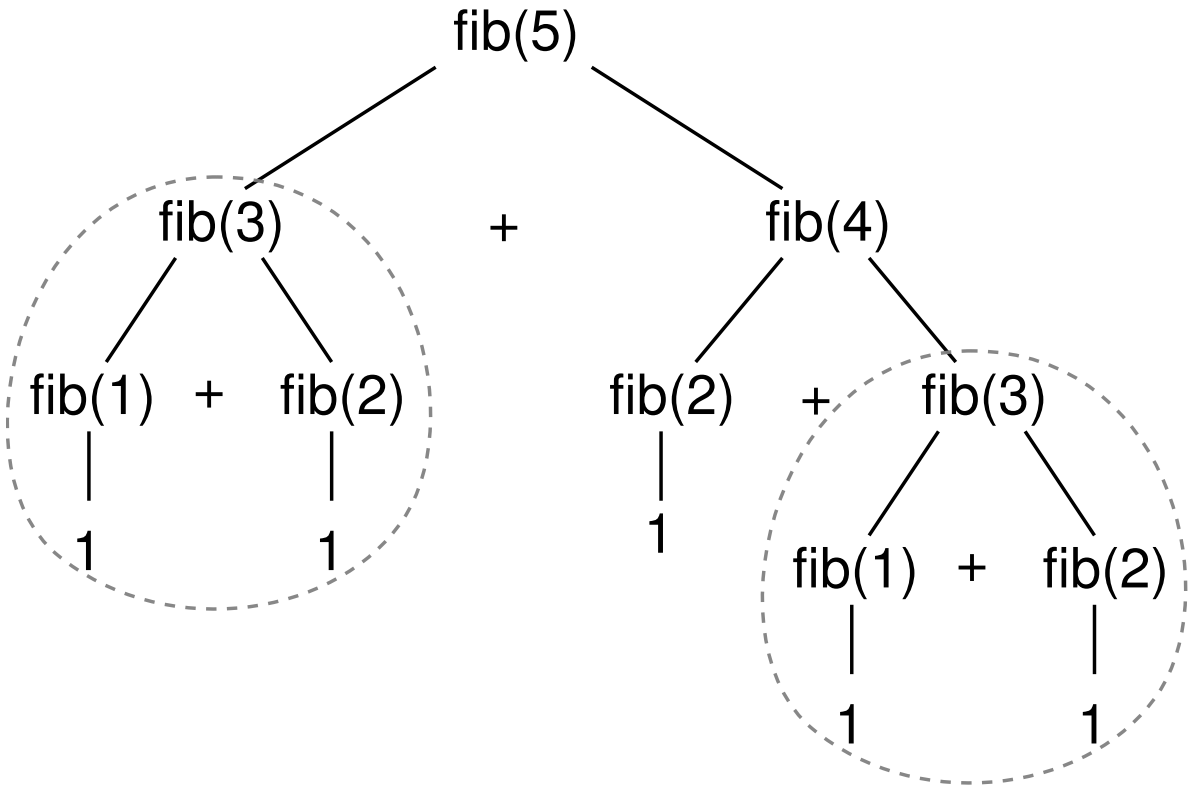
$$f_2 = 1$$

$$f_n = f_{n-1} + f_{n-2}$$

Fibonacciho posloupnost: rekurzivně

```
def fib(n):
    if n <= 2: return 1
    else: return fib(n-1) + fib(n-2)
```

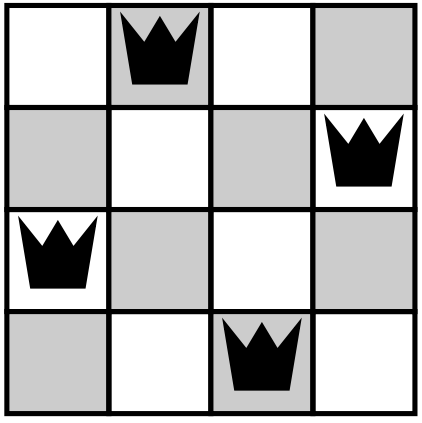
Fibonacciho posloupnost: rekurzivní výpočet

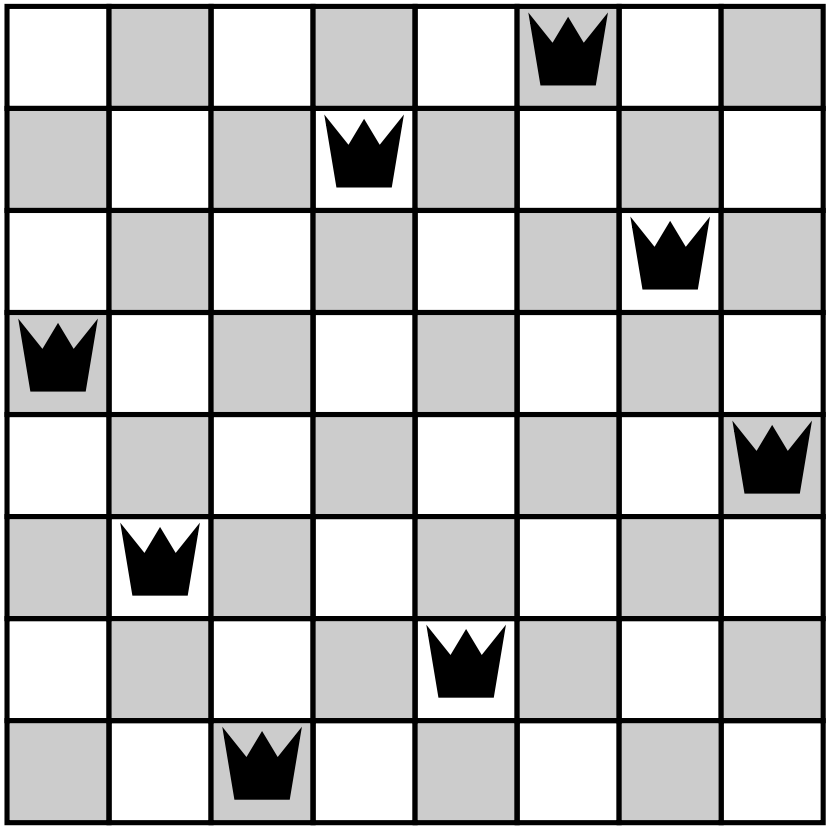


Problém N dam

- šachovnice $N \times N$
- rozestavit N dam tak, aby se vzájemně neohrožovaly
- zkuste pro $N = 4$

Problém N dam – řešení

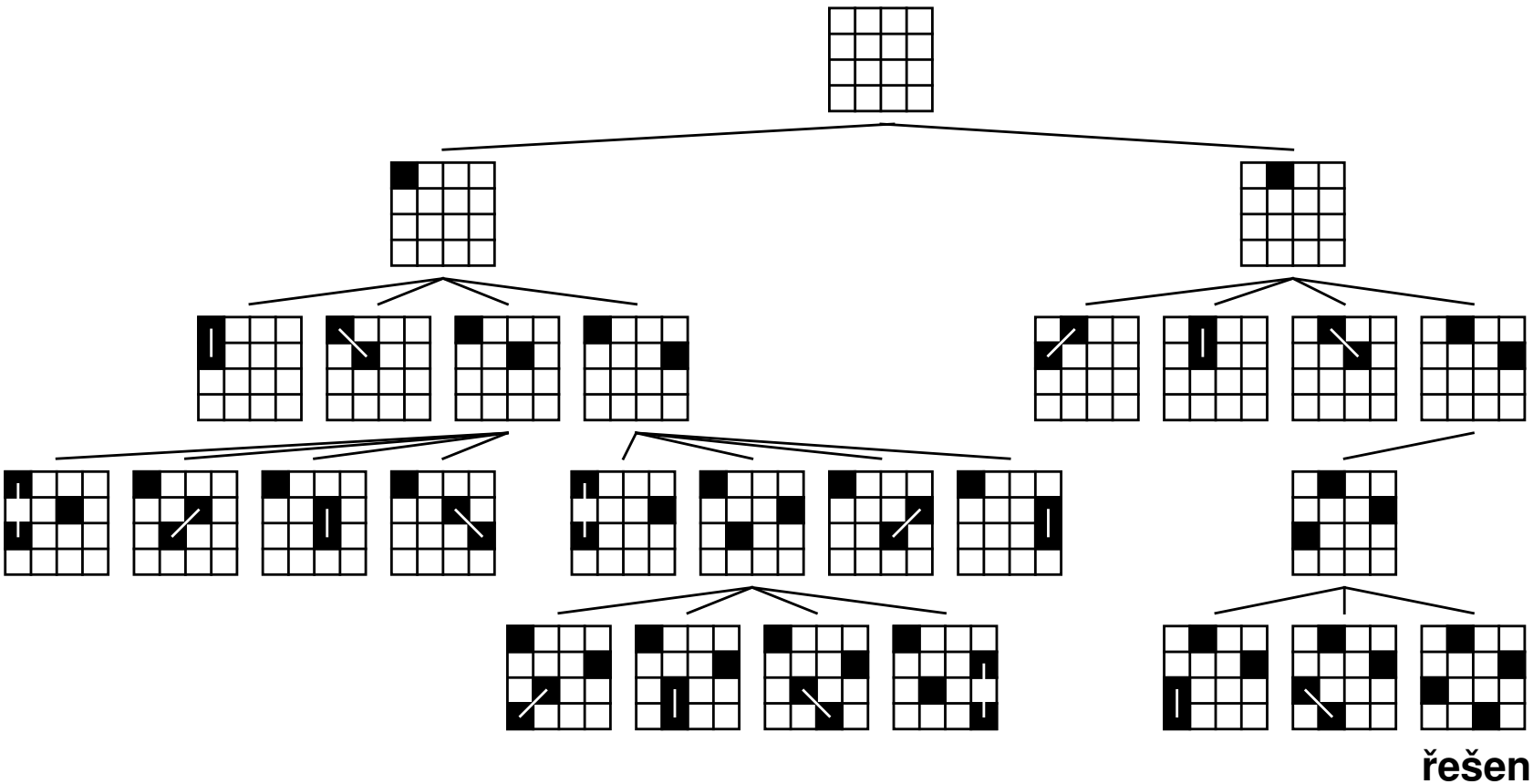




Problém N dam – algoritmus

- ilustrace algoritmu backtracking
- speciální případ obecného typu problémů („problém splnění podmínek“) a algoritmu
- začneme s prázdným plánem, systematicky zkoušíme umisťovat dámy
- pokud najdeme kolizi, vracíme se a zkoušíme jinou možnost
- přirozený rekurzivní zápis

Problém N dam – backtracking



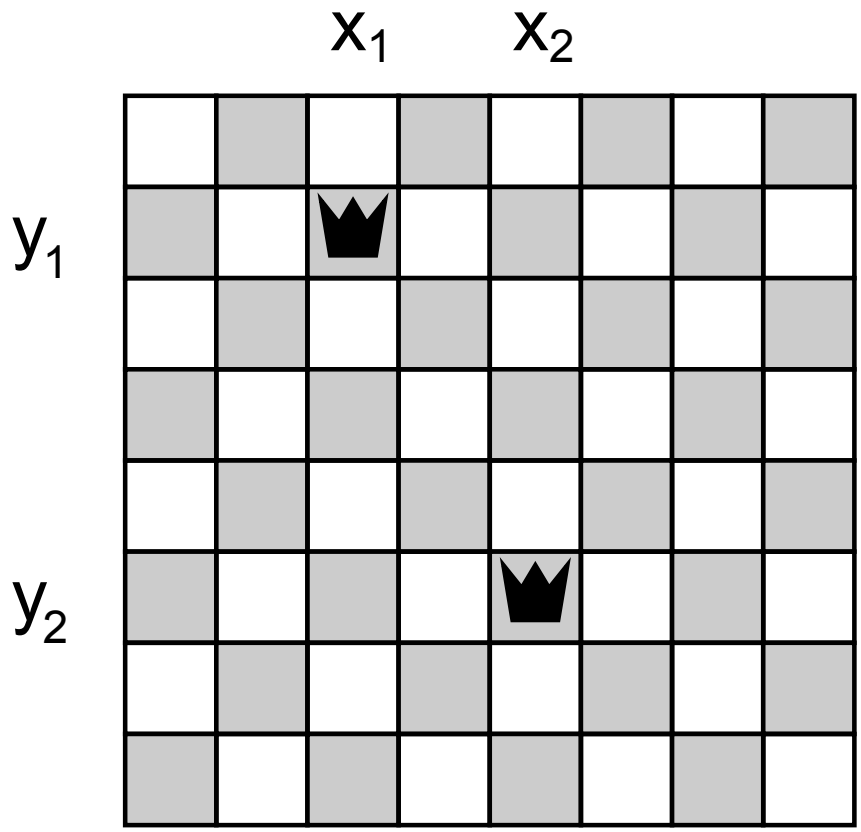
Problém N dam – reprezentace stavu

- pro každé pole si pamatujeme, zda na něm je/není dáma
  - dvourozměrný seznam True/False
- pro každý řádek si pamatujeme, v kterém sloupci je dáma
  - seznam čísel
  - snadnější manipulace

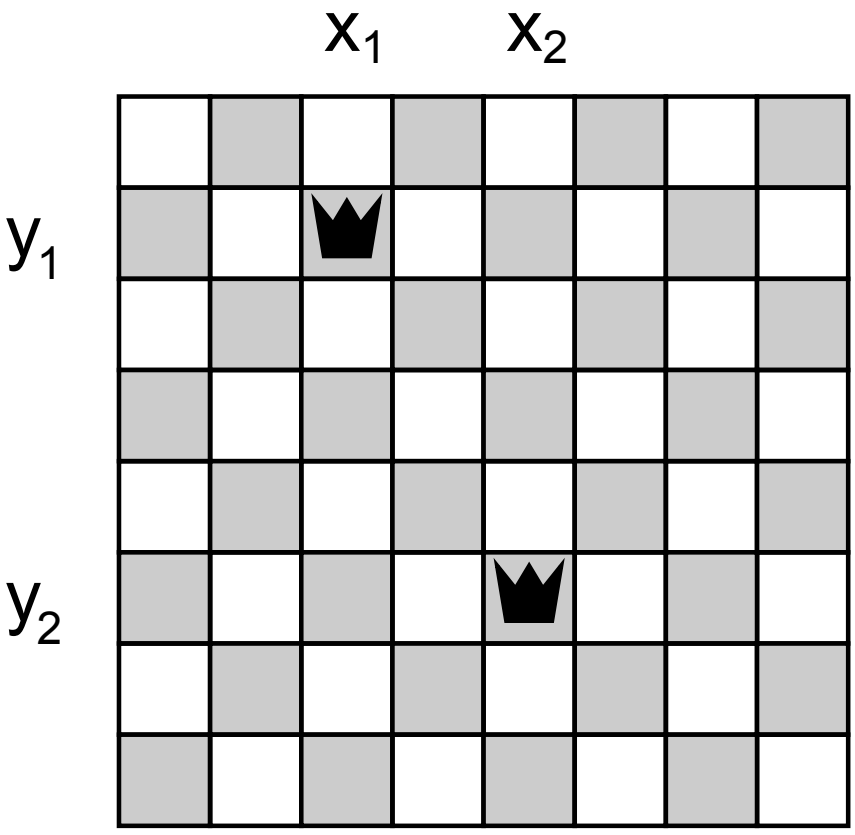
Problém N dam – řešení

```
def vyres_damy(n, stav):
    if len(stav) == n:
        vypis(stav)
        return True
    else:
        for i in range(n):
            stav.append(i)
            if zkontroluj(stav):
                if vyres_damy(n, stav): return True
            stav.pop()
    return False
```

Kdy se ohrožují dvě dámy?



Kdy se ohrožují dvě dámy?



$$x_1 = x_2$$
$$y_1 = y_2$$
$$x_1 + y_1 = x_2 + y_2$$
$$x_1 - y_1 = x_2 - y_2$$

Problém N dam – řešení

```
def vypis(stav):
    for y in range(len(stav)):
        for x in range(len(stav)):
            if stav[y]==x: print "X",
            else: print ".",
        print
    print

def zkontroluj(stav):
    for y1 in range(len(stav)):
        x1 = stav[y1]
        for y2 in range(y1+1, len(stav)):
            x2 = stav[y2]
            if x1 == x2 or x1-y1 == x2-y2 or x1+y1 == x
                return False
    return True
```

Shrnutí

- rekurze: využití rekurze pro definici sebe sama
- logické úlohy: Hanojské věže, L kostičky, dámy na šachovnici
- fraktály
- aplikace v programování: vyhledávání, řazení, prohledávání grafu
- klíčová myšlenka v informatice