Práce s daty

IB111 Základy programování
Radek Pelánek

2017

Práce s daty

připomenutí

- Jaká data budu zpracovávat?
- Jaká data budu potřebovat k řešení problému?
- Jaké operace s daty budu chtít provádět?
- Jak rychle budu chtít, aby operace s daty fungovaly?

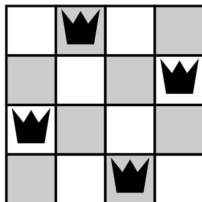
Dnešní přednáška

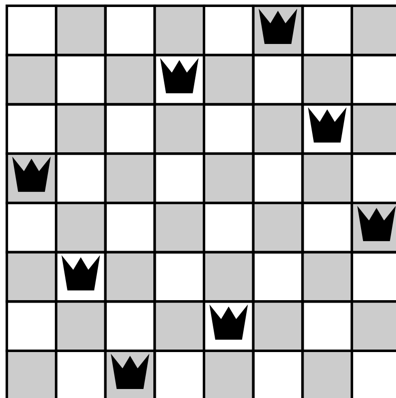
- použití datových struktur na praktických příkladech
- interakce volby dat a algoritmu
- vnořené datové struktury: seznam seznamů, slovník indexovaný dvojicí, slovník slovníků. . .
- práce s textem, soubory

Problém N dam

- šachovnice $N \times N$
- rozestavit N dam tak, aby se vzájemně neohrožovaly
- zkuste pro $N = 4$ a $N = 5$
- jak řešit algoritmicky?

Problém N dam – řešení

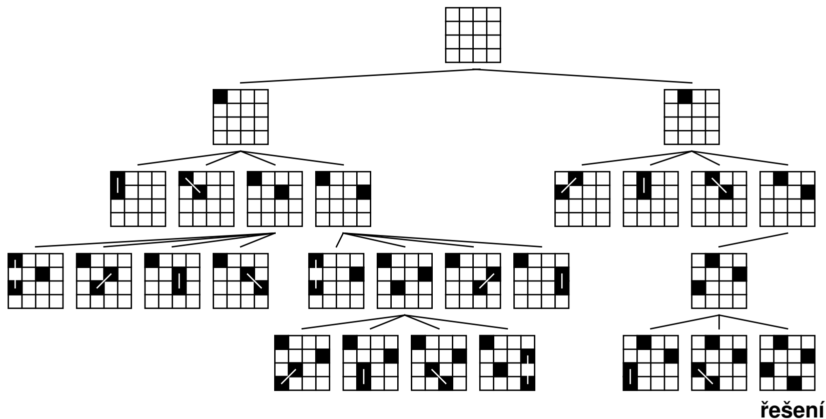




Problém N dam – algoritmus

- ilustrace algoritmu backtracking
- hrubá síla, ale „chytře“
- 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

Volba reprezentace

V jaké podobě a v jaké datové struktuře si budeme pamatovat rozmístění dam na šachovnici?

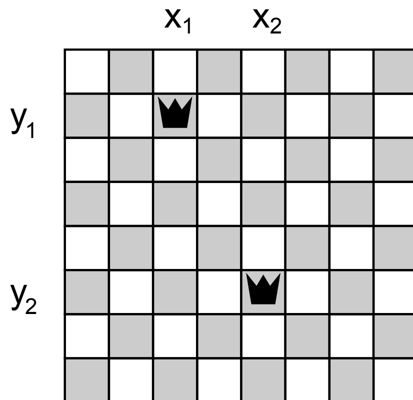
Problém N dam – reprezentace stavu

- pro každé pole si pamatujeme, zda na něm je/není dáma
  - dvourozměrný seznam True/False
- pro každou dámu si pamatujeme její souřadnice
  - seznam dvojic x_i, y_i
- pro každý řádek si pamatujeme, v kterém sloupci je dáma
  - seznam čísel x_i
  - nejvýhodnější reprezentace

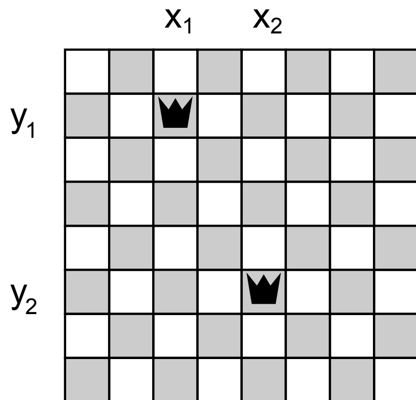
Problém N dam – rekurzivní řešení

```
def solve_queens(n, state):
    if len(state) == n:
        output(state)
        return True
    else:
        for i in range(n):
            state.append(i)
            if check_queens(state):
                if solve_queens(n, state): return True
            state.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 output(state):
    for y in range(len(state)):
        for x in range(len(state)):
            if state[y]==x: print("X", end=" ")
            else: print(".", end=" ")
        print()
    print()

def check_queens(state):
    for y1 in range(len(state)):
        x1 = state[y1]
        for y2 in range(y1+1, len(state)):
            x2 = state[y2]
            if x1 == x2 or x1-y1 == x2-y2 or\
               x1+y1 == x2+y2:
                return False
```

Zajímavost: využití randomizace

- uvedený kód pomalý pro $N \geq 20$
- výrazné urychlení při využití randomizace – procházíme všechny kandidáty na umístění dámy, ale v náhodném pořadí
- proč pomáhá?

Backtracking – další příklady použití

- mnoho logických úloh:
    - Sudoku a podobné úlohy
    - algebrogramy (SEND + MORE = MONEY)
- optimalizační problémy (např. rozvrhování, plánování)
- obecný „problém splnění podmínek" (constraint satisfaction problem)

Práce se soubory: připomenutí

Otevírání a zavírání:

- `f = open("myfile.txt")` – otevření pro čtení
- `f = open("myfile.txt","w")` – otevření pro zápis
- `f.close()` – uzavření souboru
- zápis pomocí `with` – lepší praxe (ale pokročilejší, souvisí s výjimkami)

Čtení a zápis:

- `f.readline()` – vrátí další řádek ze souboru
- `f.readlines()` – vrátí seznam všech zbývajících řádků
- `f.write(text)` – zapíše do souboru

Případová studie: Číselné bludiště

jednoduchá logická úloha

Ilustrace pojmů a postupů:

- návrh algoritmu
- volba datových struktur
- seznam seznamů
- slovník
- fronta
- načítání ze souboru
- objekty

Číselné bludiště

levý horní roh $\rightarrow$ pravý dolní roh

skoky vertikálně a horizontálně, číslo = délka skoku

2	4	4	3	3
2	3	3	2	3
3	2	3	1	3
2	2	3	2	1
1	4	4	4	☆

3	3	1	3	2
1	1	2	1	3
1	3	1	2	4
4	3	4	1	4
4	1	4	4	☆

4	2	1	2	3
1	2	2	2	3
1	4	4	3	3
3	4	3	1	4
3	2	3	4	☆

K vyzkoušení: `www.umimematiku.cz/skakacka`

Variace

- nejkratší cesta
  - vzhledem k počtu skoků
  - vzhledem k celkové délce cesty
- kontrola jednoznačnosti nejkratšího řešení
- generování „co nejtěžšího“ zadání

Reprezentace bludiště

Jak budeme v programu reprezentovat bludiště?

2	4	4	3	3
2	3	3	2	3
3	2	3	1	3
2	2	3	2	1
1	4	4	4	☆

```
zadani.txt:
5
24433
23323
32313
22321
14440
```

Seznam seznamů

„klasická“ reprezentace – dvojrozměrné pole (seznam seznamů v Pythonu)

```
maze[x][y] = number

[[2, 2, 3, 2, 1], [4, 3, 2, 2, 4],
 [4, 3, 3, 3, 4], [3, 2, 1, 2, 4],
 [3, 3, 3, 1, 0]]
```

2	4	4	3	3
2	3	3	2	3
3	2	3	1	3
2	2	3	2	1
1	4	4	4	☆

Poznámka: `maze[x][y]` nebo `maze[y][x]`?
(častý zdroj chyb: nekonzistence)

Načítání ze souboru

```
def read_maze(filename = "input.txt"):
  f = open(filename)
  n = int(f.readline())
  maze = [[0 for y in range(n)]
          for x in range(n)]
  for y in range(n):
    line = f.readline()
    for x in range(n):
      maze[x][y] = int(line[x])
  f.close()
  return maze
```

Algoritmus

Jak najít řešení?

2	4	4	3	3
2	3	3	2	3
3	2	3	1	3
2	2	3	2	1
1	4	4	4	☆

Algoritmus

speciální případ obecného „prohledávání do šířky“:

- systematicky zkoušíme všechny následníky
- pamatujeme si, kde už jsme byli
- pro každé pole si pamatujeme předchůdce (rekonstrukce cesty)

Algoritmus

2	4	4	3	3
2	3	3	2	3
3	2	3	1	3
2	2	3	2	1
1	4	4	4	☆

2	4	4	3	3
2	3	3	2	3
3	2	3	1	3
2	2	3	2	1
1	4	4	4	☆

2	4	4	3	3
2	3	3	2	3
3	2	3	1	3
2	2	3	2	1
1	4	4	4	☆

2	4	4	3	3
2	3	3	2	3
3	2	3	1	3
2	2	3	2	1
1	4	4	4	☆

2	4	4	3	3
2	3	3	2	3
3	2	3	1	3
2	2	3	2	1
1	4	4	4	☆

...

Datové struktury

co si potřebujeme pamatovat:

- *fronta* polí, které musíme prozkoumat (světle zelená pole)
- *množina* polí, která už jsme navštívili (tmavě zelená pole)
- informace o předchůdcích (světle modré šipky)

optimalizace: podle informace o předchůdcích poznáme, kde už jsme byli

Zápis v Pythonu – přímočaré řešení

```
def solve(n, maze):
    start = (0, 0)
    queue = deque([start])
    pred = [[(-1, -1) for x in range(n)]
             for y in range(n)]

    while len(queue) > 0:
        (x, y) = queue.popleft()
        if maze[x][y] == 0:
            print_solution((x, y), pred)
            break
        k = maze[x][y]
        if x+k < n and pred[x+k][y] == (-1, -1):
            queue.append((x+k, y))
            pred[x+k][y] = (x, y)
        if x-k >= 0 and pred[x-k][y] == (-1, -1):
            queue.append((x-k, y))
            pred[x-k][y] = (x, y)
        if y+k < n and pred[x][y+k] == (-1, -1):
            queue.append((x, y+k))
            pred[x][y+k] = (x, y)
        if y-k >= 0 and pred[x][y-k] == (-1, -1):
            queue.append((x, y-k))
            pred[x][y-k] = (x, y)
```

Poznámky

- příkaz `break` – opuštění cyklu
- využití zkráceného vyhodnocování

Alternativní reprezentace: slovník

slovník indexovaný dvojicí
souřadnice $(x, y) \to$ číslo na dané souřadnici

```
maze[(x, y)] = number
```

```
{(1, 2): 2, (3, 2): 1, (0, 0): 2,
 (3, 0): 3, (2, 2): 3, (2, 1): 3,
 (1, 3): 2, (2, 3): 3, (1, 4): 4,
 (2, 4): 4, (4, 2): 3, (0, 3): 2,
 ... }
```

2	4	4	3	3
2	3	3	2	3
3	2	3	1	3
2	2	3	2	1
1	4	4	4	☆

N-tice a závorky

U n-tic většinou nemusíme psát závorky:

- `a, b = b, a`
  místo
  `(a, b) = (b, a)`
- `maze[x, y]`
  místo
  `maze[(x, y)]`

Slovník vs seznam seznamů

Pozor na rozdíl!

- `maze[x][y]` – seznam seznamů
- `maze[x, y]` – slovník indexovaný dvojicí

Další vylepšení

Zobecnění repetitivního kódu:

„copy&paste" kód pro 4 směry

$\Downarrow$

`for` cyklus přes 4 směry

Předchůdci

- pamatujeme si rovnou celou cestu (*)
- také slovník

(*) to je sice „plýtvání pamětí", ale vzhledem k velikosti zadání nás to vůbec nemusí trápit

Upravený kód

```
def solve(maze):
  start = (0, 0)
  queue = deque([start])
  pred = {}
  pred[start] = [start]
  while len(queue) > 0:
    (x, y) = queue.popleft()
    if maze[x, y] == 0:
      print(pred[x,y])
    for (dx, dy) in [(-1,0), (1,0), (0,-1), (0,1)]:
      newx = x + dx * maze[x, y]
      newy = y + dy * maze[x, y]
      if (newx, newy) in maze and \
         not (newx, newy) in pred:
        queue.append((newx, newy))
        pred[newx, newy] = pred[x, y] + [(newx, newy)]
```

Poznámky

Někdy jsou závorky důležité:

- `s = [1, 2]` – seznam obsahující dva prvky
  `s = [(1, 2)]` – seznam obsahující jeden prvek (dvojici)
- `s.append((1, 2))` – přidávám dvojici
  `s.append(1, 2)` – volám append se dvěma argumenty (chyba)

Objektová reprezentace

```python
class Maze:

  def __init__(self):
    # initialize

  def read(self, filename):
    # read maze from file

  def solve(self):
    # find solution

  def print_solution(self):
    # text output

  def save_image(self, filename, with_solution = True):
    # save maze as an image
```

Objektová reprezentace

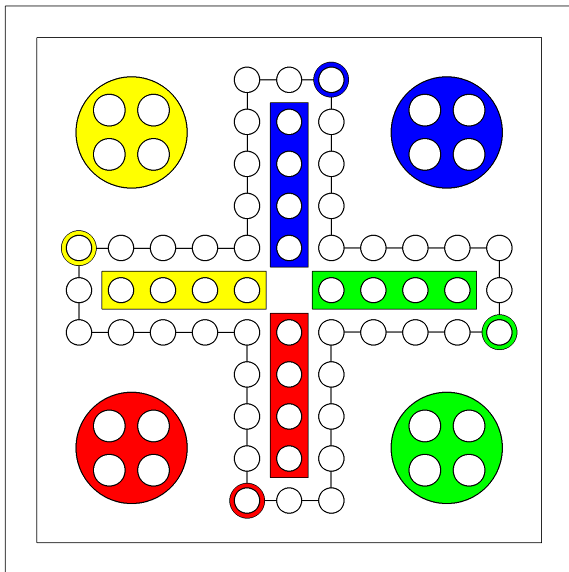
- pro základní úlohu není nezbytné
- velmi vhodné pro rozšíření, např. „hledání těžkých zadání“ (potřeba pracovat s více zadáními současně)
- *doporučené cvičení*

Volba datové struktury

Jaká datová struktura pro reprezentaci stavu?

- Piškvorky
  - plán omezené velikosti
  - neomezený plán
- Člověče, nezlob se!
- Želví závody
- Pexeso

Člověče, nezlob se!



Želví závody

Pexeso – simulace paměti

- simulace hry pexeso
- hráči s různě dokonalou pamětí:
  - prázdná paměť (náhodná hra)
  - dokonalá paměť
  - omezená paměť – „buffer“ velikosti k
- pravděpodobnost výhry pro jednotlivé hráče

- reprezentace stavu hry (kartiček)?
- reprezentace paměti?

Kahoot otázky: 1, 2

```
a = [1, 2, 3]
b = [(1, 2, 3)]
c = [(1, 2), 3]
```

Kahoot otázky: 3, 4

```
data = {"a": [1, 3, 7, 12],
        "c": [2, 5],
        "x": [19, 2, 4]}
data["d"] = [1, 2]
data["c"] = [6, 7]
```

Kahoot otázky: 5, 6, 7

```
d1 = {"a": 5, "c": "R2D2"}
d2 = {"b": 8, "a": 7, "B": "C3PO"}
s = [d1, d2]
t = "BB8"
```

Kahoot otázky: 8

```
x = []
nums = []
for i in range(3):
    nums.append(x)
    x.append(i)
```

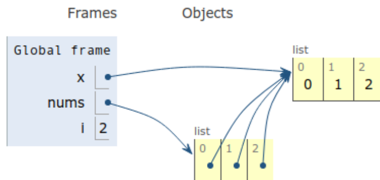
Vizualizace Python Tutor

Python 3.6

```
1  x = []
2  nums = []
3  for i in range(3):
4      nums.append(x)
5      x.append(i)
```

Edit code | Live programming

uted



Práce s textem

- operace s textem – připomenutí a rozšíření
- ukázka zpracování

Rozdělení řetězce

- split – rozdělí řetězec podle zadaného podřetězce, vrací seznam částí
- join – spojení seznamu řetězců do jednoho

```
>>> retezec = "Holka modrooka   nesedavej u potoka"
>>> retezec.split()
['Holka', 'modrooka', 'nesedavej', 'u', 'potoka']
>>> retezec.split('o')
['H', 'lka m', 'dr', '', 'ka   nesedavej u p', 't', 'ka
>>> retezec.split('ka')
['Hol', ' modroo', '   nesedavej u poto', '']
```

Řetězce: další funkce

- `find`, `count` – vyhledávání a počítání podřetězců
- `replace` – nahrazení podřetězce
- `lower`, `upper` – převod na malá/velká písmena
- `ljust`, `rjust`, `center` – zarovnání textu
- `lstrip`, `rstrip` – ořezání bílých znaků na začátku/konci

Pozn. objektová notace, metody

Analýza textu

- vstup: text (v textovém souboru)
- výstup: zajímavé statistiky
  - délka vět, slov
  - nejčastější slova (určité minimální délky)
  - frekvence písmen, digramy, trigramy

$\Rightarrow$ *cvičení*

Analýza textu

statistiky délky slov a vět:

- $\bar{x}$ – průměr
- s – směrodatná odchylka (míra variability)

	slova		**věty**	
	$\bar{x}$	s	$\bar{x}$	s
Starý zákon	4.3	2.3	14.9	7.8
Čapek	4.5	2.5	14.9	13.3
Pelánek	5.9	3.6	13.5	6.9
Wikipedie	5.6	3.0	14.8	8.3

Imitace textu

- vstup: rozsáhlý text
- výstup: náhodně generovaný text, který má „podobné charakteristiky“ jako vstupní text
- imitace na úrovni písmen nebo slov

Náhodnostní imitace vstupního textu

I špiské to pole kavodali pamas ne nebo kdy v Dejný Odm sem uvalini se zabijí s Pan stěží ře, a silobe lo v ne řečekovících blova v nadrá těly jakvěmutelaji rohnutkohonebout anej Fravinci V A pěk finé houty. zal Jírakočítencej ské žil, kdDo jak a to Lorskříže si tomůžu schno mí, kto.

Kterak král kočku kupoval V zemi Taškářů panoval král a zapřisáhl se velikou přísahou že bude pochválena První pán si jí ani nevšimnul zato druhý se rychle shýbl a Jůru pohladil Aha řekl sultán a bohatě obdaroval pana Lustiga koupil od něho telegram z Bombaje v Indii není o nic horší člověk nežli někdo z mých hraček Kdepak mávl Vašek rukou

Základní přístup

1. vstupní text $\Rightarrow$ statistiky textu
2. statistiky $\Rightarrow$ generování náhodného textu

Co jsou vhodné statistiky?

Statistiky textu

- základ: frekvence písmen (slov)
- rozšíření: korelace mezi písmeny (slovy)

příklad: pokud poslední písmeno bylo `a`:

- `e` velmi nepravděpodobné (méně než obvykle)
- `l`, `k` hodně pravděpodobná (více než obvykle)

Implementace

- základní frekvenční analýza – datová struktura seznam nebo slovník
  písmeno $\Rightarrow$ frekvence
- rozšířená analýza – seznam seznamů nebo slovník slovníků
  písmeno $\Rightarrow$ { písmeno $\Rightarrow$ frekvence }

generování

- podle aktuálního písmene získám frekvence
- vyberu náhodné písmeno podle těchto frekvencí – „vážená ruleta“

Reprezentace dat

- seznam seznamů – tabulka 26×26, pouze pro malá písmena anglické abecedy
- slovník slovníků – pro libovolné symboly

Korelace písmen: seznamy

```
def letter_ord(char):
    return ord(char) - ord('a')


def letter_correlations(text):
    counts = [[0 for a in range(26)]
                for b in range(26)]
    for i in range(len(text)-1):
        a = letter_ord(text[i])
        b = letter_ord(text[i+1])
        if 0 <= a <= 26 and 0 <= b <= 26:
            counts[a][b] += 1
    return counts
```

Výpis nejčastějších následujících písmen

```
def most_common_after(letter, counts, top_n=5):
    i = letter_ord(letter)
    letter_counts = [(counts[i][j], chr(ord('a')+j))
                     for j in range(26)]
    letter_counts.sort(reverse=True)
    for count, other_letter in letter_counts[:top_n]:
        print(other_letter, count)
```

Korelace písmen: slovníky

```
def symbol_correlation(text):
  counts = {}
  last = " "
  for symbol in text:
    if last not in counts:
      counts[last] = {}
    counts[last][symbol] = counts[last].get(symbol, 0)-
    last = symbol
  return counts
```

Tip pro kratší kód: `defaultdict`

Výpis nejčastějších následujících písmen

```
def most_common_after_symbol(symbol, counts, top_n=5):
    print(sorted(counts[symbol].keys(),
                 key=lambda s:
                 -counts[symbol][s])[:top_n])
```

Imitace textu

```
def get_next_letter(current, counts):
    total = sum(counts[current].values())
    r = random.randint(0, total-1)
    for symbol in counts[current].keys():
        r -= counts[current][symbol]
        if r < 0:
            return symbol
    return " "

def imitate(counts, length=100):
    current = " "
    for _ in range(length):
        print(current, end="")
        current = get_next_letter(current, counts)
```

Imitace textu: rozšíření

- nebrat v potaz pouze předcházející písmeno, ale k předcházejících písmen
- *doporučené cvičení*

Imitace sofistikovaněji

- Recurrent Neural Networks – dokáží postihnout i složitější aspekty jazyka
- básně, recepty, Wikipedia články, zdrojové kódy, ...

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

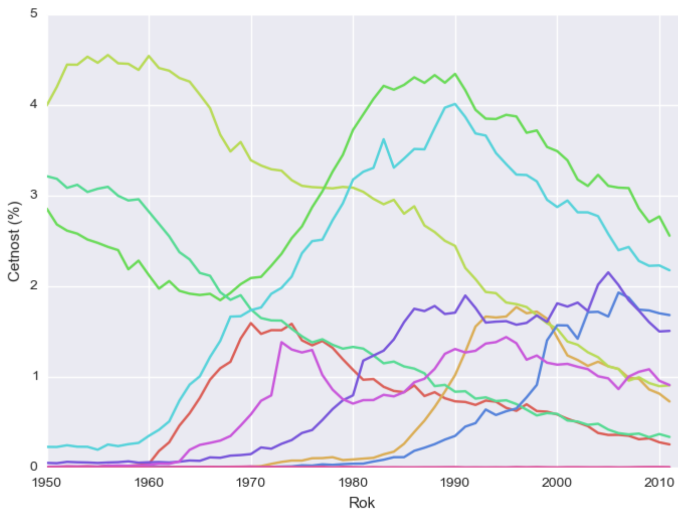
```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.
```

Statistiky jmen

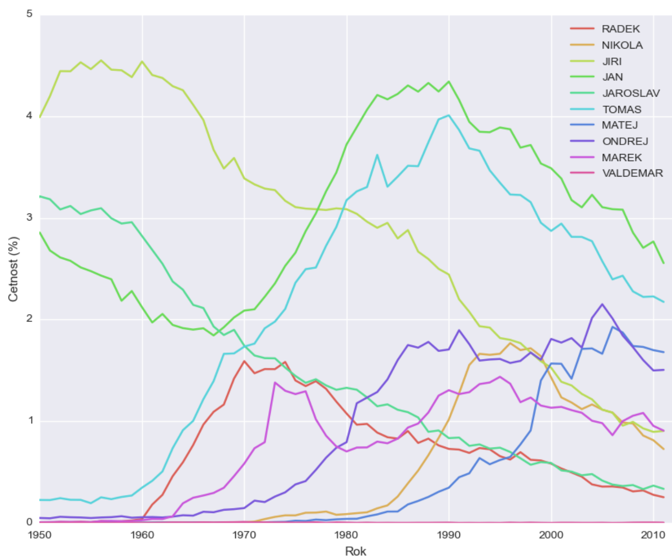
- data: četnosti jmen, příjmení podle roků, krajů, ...
- zdroj: Ministerstvo vnitra ČR
  `http://www.mvcr.cz/clanek/cetnost-jmen-a-prijmeni-722752.aspx`
- XLS – pro zpracování v Pythonu uložit jako CSV (comma-separated values)
- **doporučené cvičení**
  - snadno zpracovatelné
  - zajímavá data
  - cvičení na vymýšlení otázek
- následuje několik ukázek pro inspiraci ...

Poznámky ke zpracování

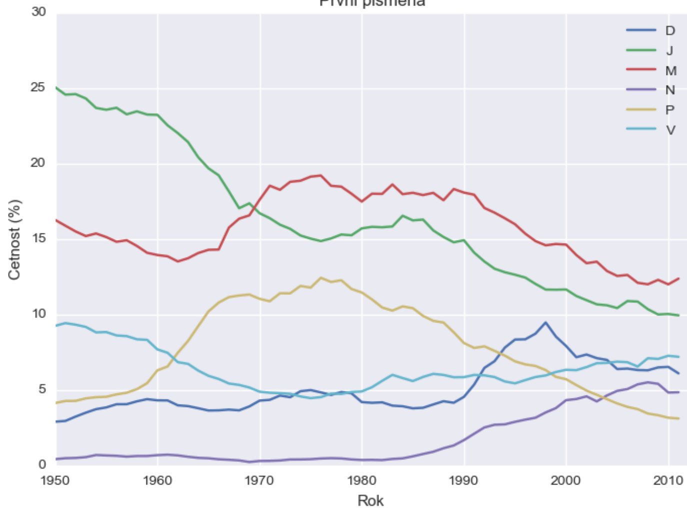
- slovník: jméno → seznam výskytů
- CSV – funkce `split` → seznam
- normalizace (relativní výskyty jmen) – podělit součtem (pro daný rok)
  - různě velké ročníky
  - neúplná data u starých ročníků



JAN, JAROSLAV, JIŘÍ, MAREK, MATĚJ, NIKOLA, ONDŘEJ, RADEK, TOMÁŠ, VALDEMAR



RADEK
NIKOLA
JIRI
JAN
JAROSLAV
TOMAS
MATEJ
ONDREJ
MAREK
VALDEMAR
Cetnost (%)
Rok
0
1
2
3
4
5
1950
1960
1970
1980
1990
2000
2010


Prvni pismena
30
25
20
15
10
5
0
Cetnost (%)
1950
1960
1970
1980
1990
2000
2010
Rok
D
J
M
N
P
V

Otázky

Co zajímavého můžeme z dat zjistit?

Kladení otázek – důležitá dovednost hodná tréninku.

> Computers are useless. They can only give you answers. (Pablo Picasso)

Identifikace trendů

U kterých jmen nejvíce roste/klesá popularita?

- co to vlastně znamená?
- jak formalizovat?

Nejdelší růst/pokles

Kolik let v řadě roste popularita jména:

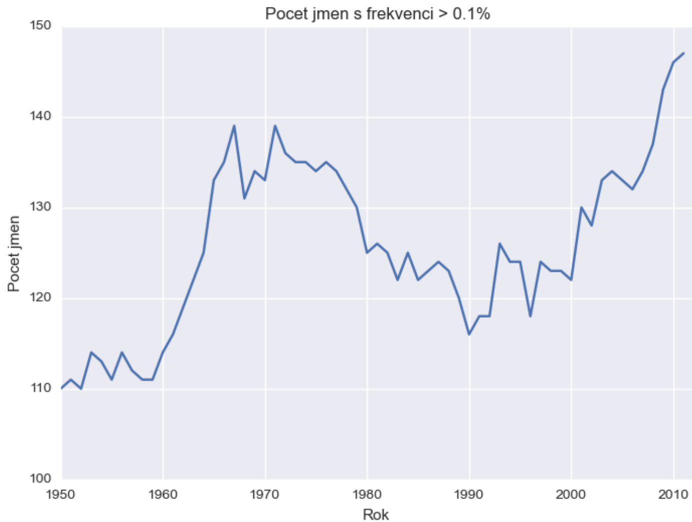
- Tobiáš – 14
- Viktorie, Ella, Sofie – 9
- Elen, Tobias – 8

Kolik let v řadě klesá popularita jména:

- Jana – 26
- Martin – 21
- Petra – 11
- Zdeněk – 9

Největší skok v popularitě za 10 let

- alespoň desetinásobný nárůst popularity:
  Sofie, Elen, Amálie, Ella, Nicol, Nella, Tobias
- pokles alespoň o 60 %:
  Petra, Pavlína, Martina


Pocet jmen s frekvenci > 0.1%
Pocet jmen
150
140
130
120
110
100
1950
1960
1970
1980
1990
2000
2010
Rok

Zdroje zajímavých dat

Otevřená data / Open data

- http://www.opendata.cz
- http://www.otevrenadata.cz
- http://www.data.gov
- http://data.gov.uk

Zpracování dat seriózněji

využití existujících knihoven:

- načítání dat ve standardních formátech: HTML, XML, JSON, CSV, ...
- operace s daty: `numpy`, `pandas`
- vizualizace: `matplotlib`
- interaktivní prozkoumávání dat: `ipython`, `jupyter`

Shrnutí

N dam, číselné bludiště, imitace textu

- jaká data potřebuji reprezentovat?
- jak budu data reprezentovat? (volba datových struktur)
- co se těmi daty budu dělat? (návrh algoritmu)

doporučená cvičení – programujte!