Využití datových struktur, práce s textem

IB111 Úvod do programování

2016

Výhled

- dnes: logické úlohy a hry, textová data
- příště: obrázky

- nástroje pro práci s textem/obrázky
- ilustrační příklady
- využití datových struktur
  - co kdy použít
  - výhody a nevýhody různých reprezentací
- uvažování o problémech, volba přístupu, kladení otázek

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

Logické úlohy, hry

- Hanojské věže
- Problém N dam
- Číselné bludiště
- Šachy
- Člověče, nezlob se!
- Pexeso
- Piškvorky

- algoritmus
- reprezentace stavu

Reprezentace stavu

přednáška o rekurzi:

- Hanojské věže
  - seznam seznamů (různé délky): velikosti disků
- Problém N dam
  - 2D mřížka – seznam seznamů (fixní délky) True/False
  - seznam souřadnic
  - optimalizovaný seznam souřadnic

Číselné bludiště

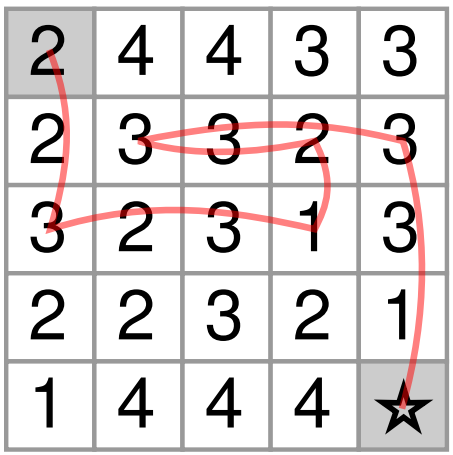
jednoduchá logická úloha

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



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: `tutor.fi.muni.cz`

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?

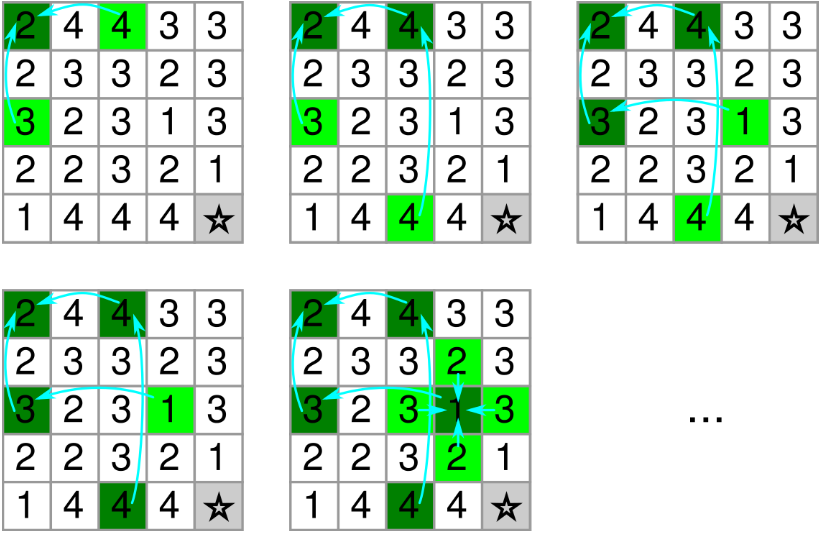
2	4	4	3	3
2	3	3	2	3
3	2	3	1	3
2	2	3	2	1
1	4	4	4	☆

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



...

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 $(x, y) \rightarrow$ 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

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

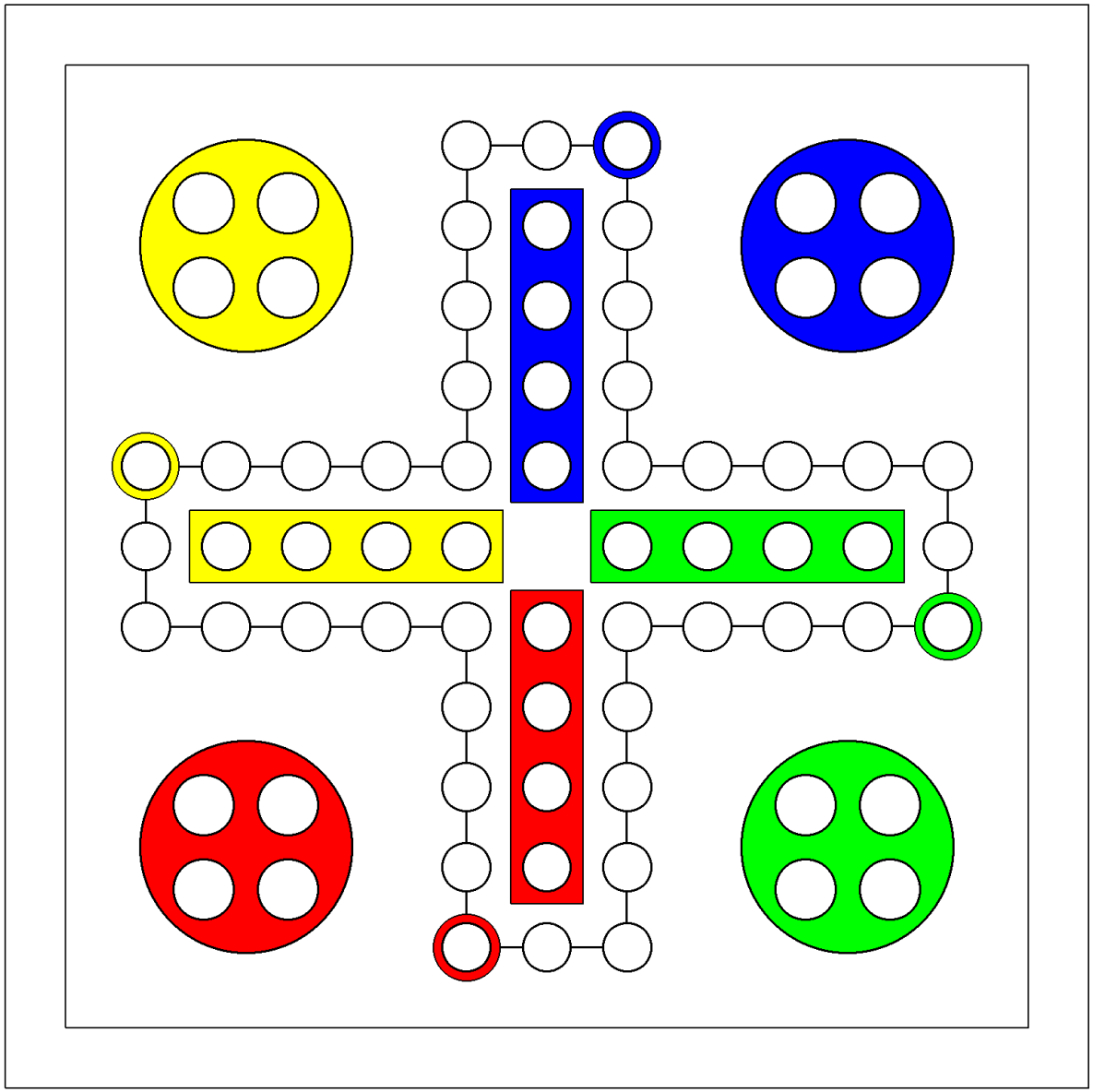
```
class Maze:

  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
```

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?

Práce s textem

- více o operacích s textem
- regulární výrazy
- ukázky

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
- `lower`, `upper` – převod na malá/velká písmena
- `ljust`, `rjust`, `center` – zarovnání textu
- `lstrip`, `rstrip` – ořezání bílých znaků na začátku/konci

Pozn. objektová notace, metody

Regulární výrazy

nástroj pro hledání „vzorů“ v textu

- programování
- textové editory
- příkazová řádka: např. `grep`
- teorie: formální jazyky, konečné automaty

http://xkcd.com/208/

Regulární výrazy

- obecně používaný nástroj
- syntax velmi podobná ve většině jazyků, prostředí
- ukázky základního využití v Pythonu

Ukázka

```
f = open("slovnik.txt")
for line in f.readlines():
    line = line.rstrip()
    if re.search(r'st.*st', line):
        print(line)
f.close()
```

Znaky a speciální znaky

- základní znak „vyhoví“ právě sám sobě
- speciální znaky: `.` `^` `$` `*` `+` `?` `{` `}` `[` `]` `\` `|` `(` `)`
  - umožňují konstrukci složitějších výrazů
  - chceme, aby odpovídaly příslušnému symbolu $\Rightarrow$ prefix \

Skupiny znaků

`[abc]` – jeden ze znaků `a, b, c`
`[^abc]` cokoliv jiného než `a, b, c`

`\d`	Čísla: `[0-9]`
`\D`	Cokoliv kromě čísel: `[^0-9]`
`\s`	Bílé znaky: `[ \t\n\r\f\v]`
`\S`	Cokoliv kromě bílých znaků: `[^ \t\n\r\f\v]`
`\w`	Alfanumerické znaky: `[a-zA-Z0-9_]`
`\W`	Nealfanumerické znaky: `[^a-zA-Z0-9_]`

Speciální symboly

`.`	libovolný znak
`^`	začátek řetězce
`$`	konec řetězce
`|`	alternativa – výběr jedné ze dvou možností

Opakování

`*`	nula a více opakování
`+`	jedno a více opakování
`?`	nula nebo jeden výskyt
`{m,n}`	m až n opakování

Pozn. `*`, `+` jsou „hladové“, pro co nejmenší počet opakování `*?`, `+?`

Příklady

Jaký je význam následujících výrazů?

- `\d[A-Z]\d \d\d\d\d`
- `\d{3}\s?\d{3}\s?\d{3}`
- `[a-z]+@[a-z]+\.cz`
- `^To:\s*(fi|kit)(-int)?@fi\.muni\.cz`

Regulární výrazy v Pythonu

- knihovna `re` (`import re`)
- `re.match` – hledá shodu na začátku řetězce
- `re.search` – hledá shodu kdekoliv v řetězci
- (`re.compile` – pro větší efektivitu)
- „raw string“ – r'vyraz' – nedochází k interpretaci speciálních znaků jako u běžných řetězců v Pythonu

Regulární výrazy v Pythonu: práce s výsledkem

- match/search vrací „MatchObject“ pomocí kterého můžeme s výsledkem pracovat
- pomocí kulatých závorek () označíme, co nás zajímá

Regulární výrazy v Pythonu: práce s výsledkem

```
>>> m = re.match(r"(\w+) (\w+)", \
                 "Isaac Newton, fyzik")
>>> m.group(0)
'Isaac Newton'
>>> m.group(1)
'Isaac'
>>> m.group(2)
'Newton'
```

Regulární výrazy: xkcd

http://xkcd.com/1313/
http://www.explainxkcd.com/wiki/index.php/1313:_Regex_Golf
https://regex.alf.nu/

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

Analýza textu – postup

1. text $\rightarrow$ seznam délek slov (vět)
2. seznam délek $\rightarrow$ statistiky

přímočaré řešení 1. kroku:

- procházet vstup po znacích
- pamatovat si délku aktuálního slova, věty
- speciální znak (mezera, tečka a podobně) $\rightarrow$ aktualizace seznamu

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 slovník
  písmeno $\Rightarrow$ frekvence
- rozšířená analýza – slovník slovníků
  písmeno $\Rightarrow$ $\{$ písmeno $\Rightarrow$ frekvence $\}$

generování

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

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/

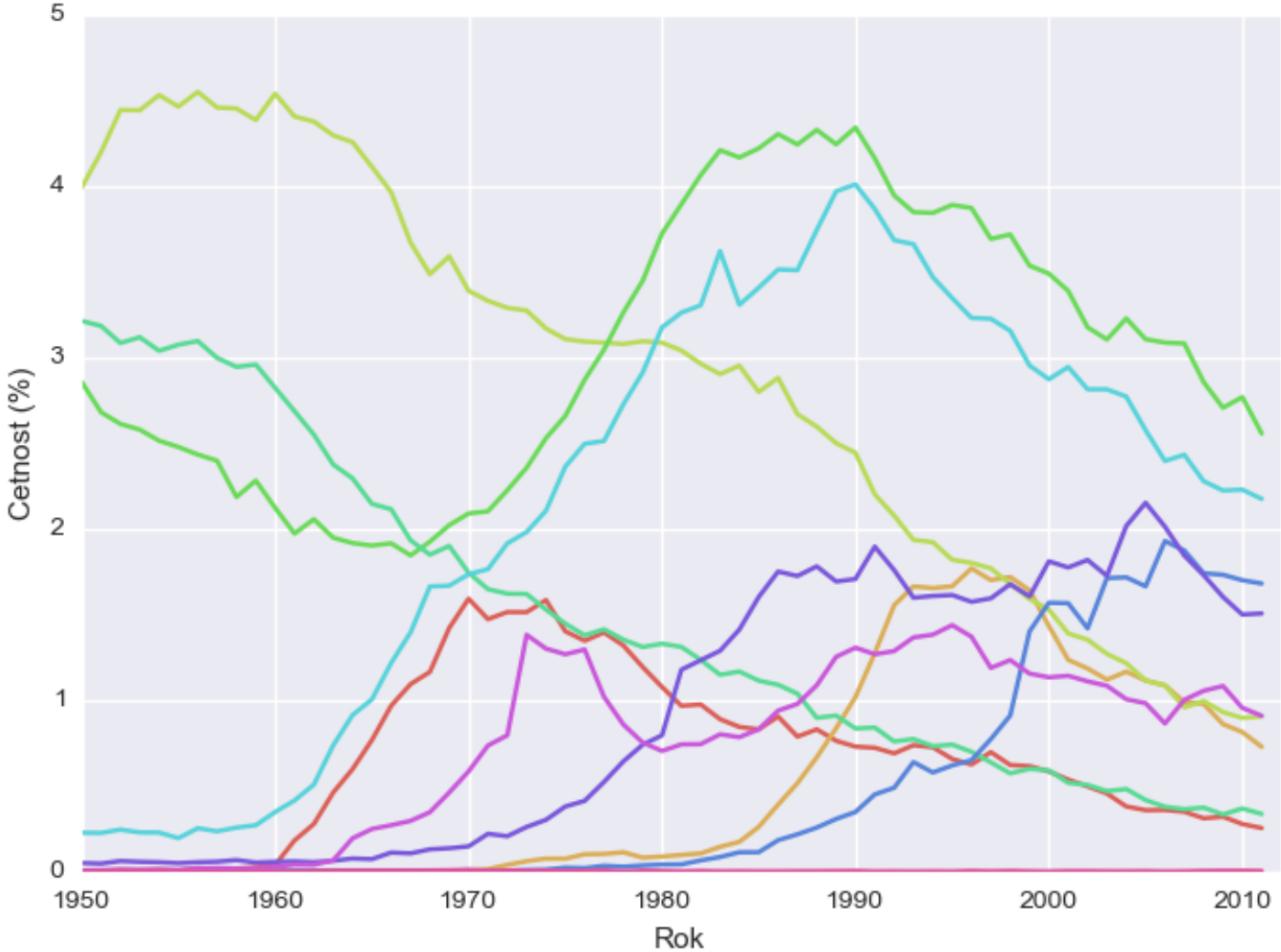
```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

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

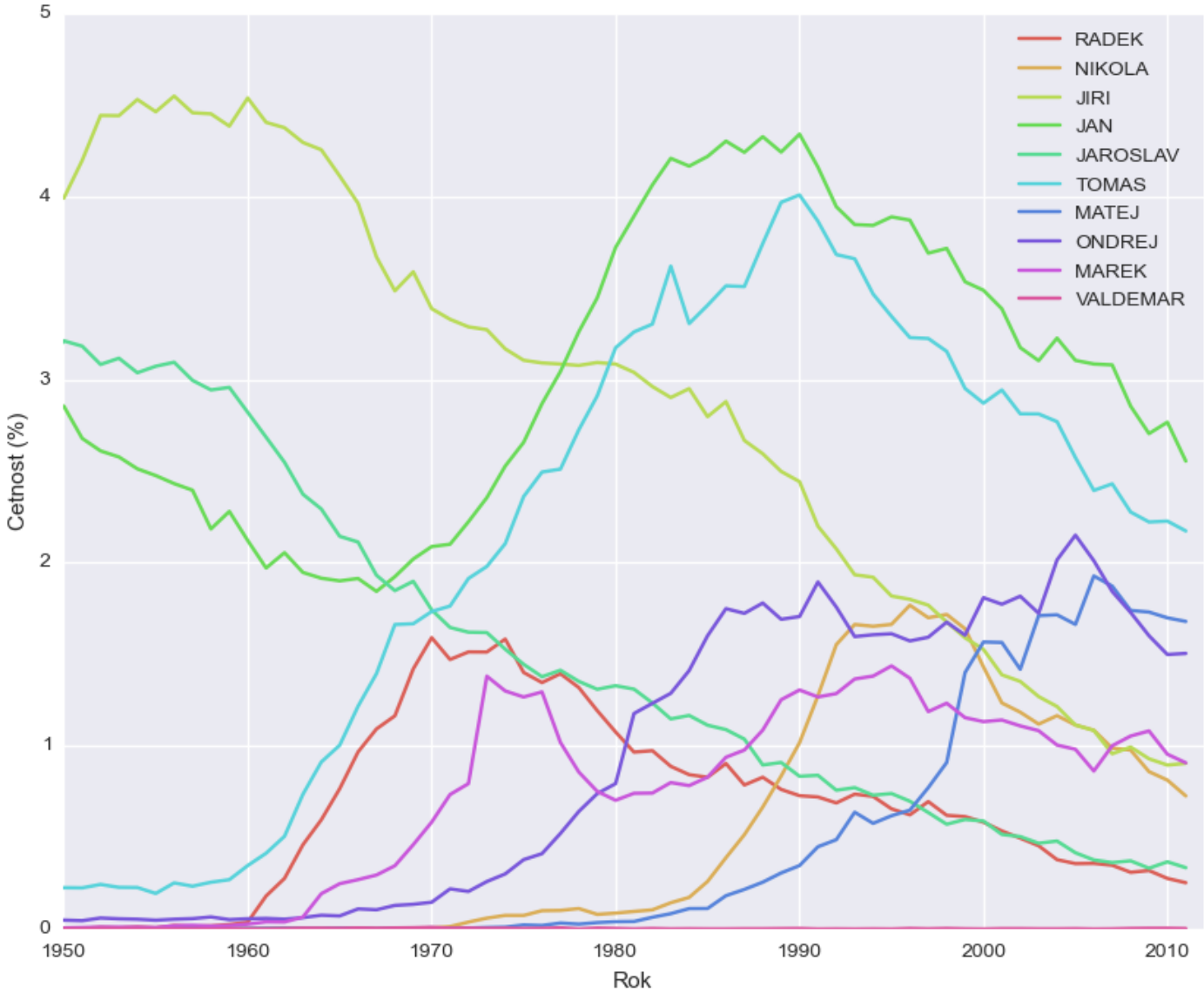
- data: četnosti jmen, příjmení podle roků, krajů, ...
- zdroj: Ministerstvo vnitra ČR
  `http://www.mvcr.cz/clanek/cetnost-jmen-a-prijmeni-722752.aspx`
- XLS – pro zpracování v Pythonu uložit jako CSV (comma-separated values)
- doporučené cvičení
  - snadno zpracovatelné
  - zajímavá data
  - cvičení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

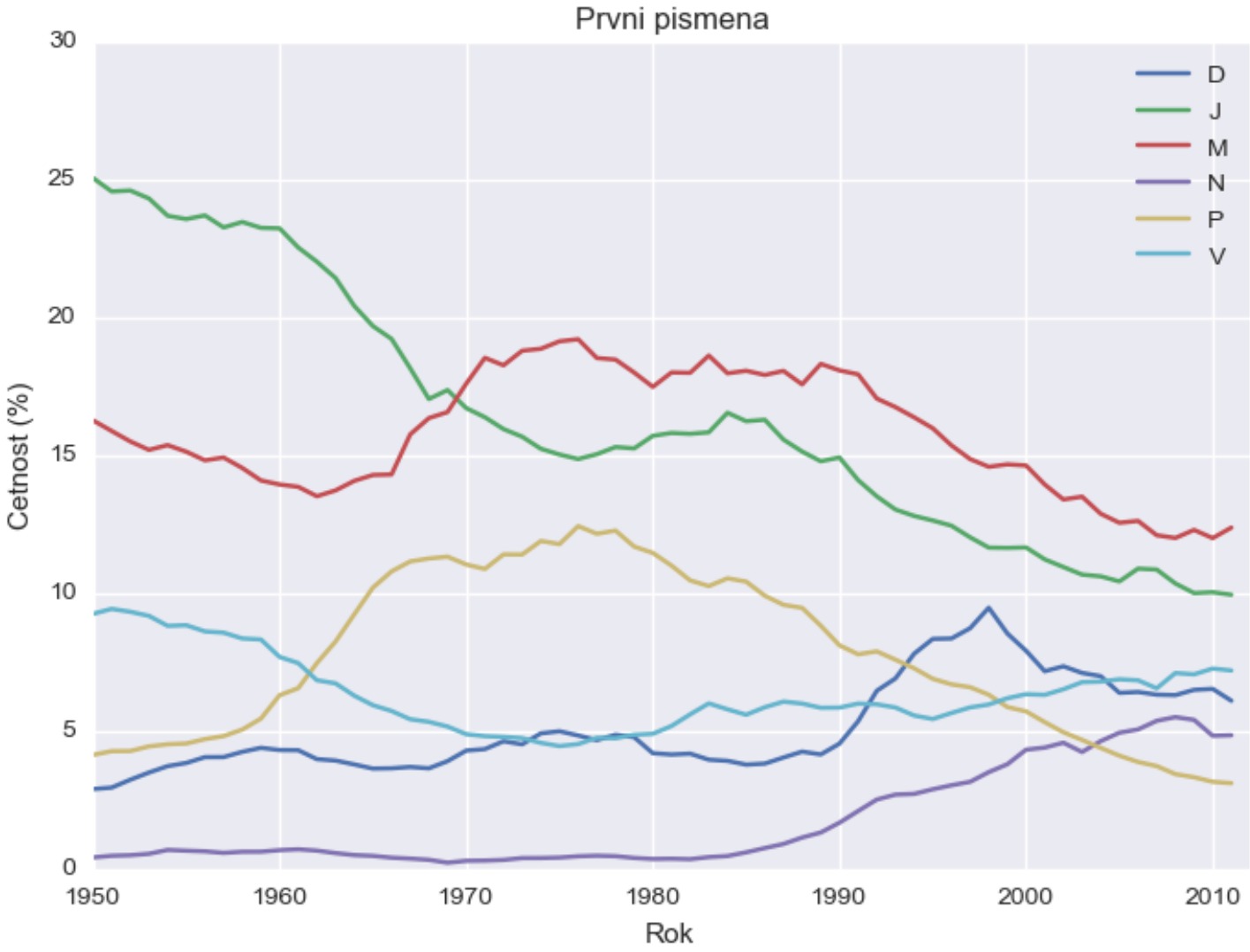
Poznámky ke zpracování

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



Vyučující IB111: 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
Cetnost (%)
Rok
0
5
10
15
20
25
30
1950
1960
1970
1980
1990
2000
2010
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:

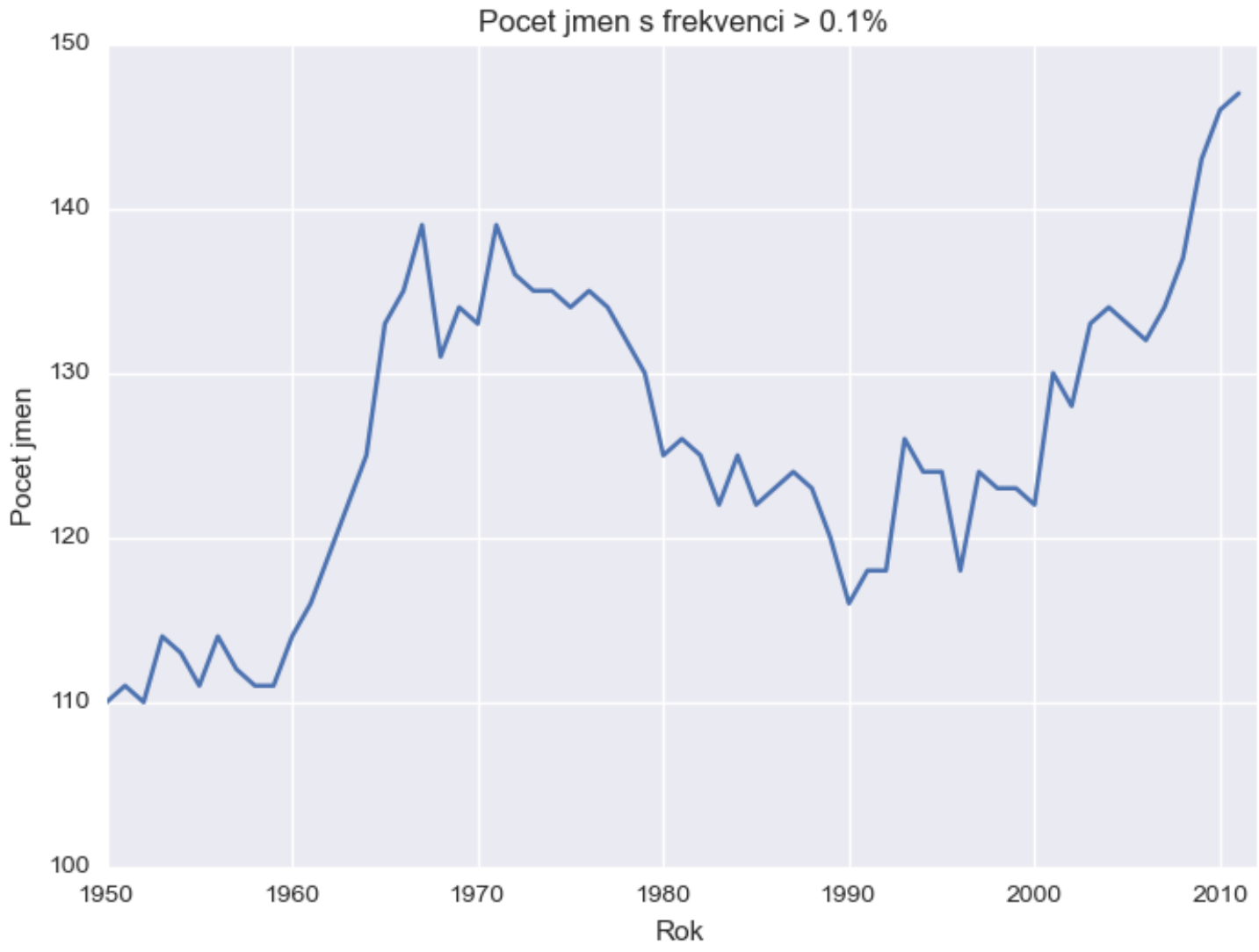
- Tobiáš – 14
- Viktorie, Ella, Sofie – 9
- Elen, Tobias – 8

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%
150
140
130
120
110
100
Pocet jmen
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`

prostředí `ipython` – interaktivní prozkoumávání dat

Shrnutí

- využití datových struktur: seznamy seznamů, slovníky
- práce s textem: split, regulární výrazy
- kladení otázek, přístup k problému