Práce s daty, příklady

IB113
Radek Pelánek

2022

Připomenutí: Největší přesmyčkové skupiny

Vstup: seznam českých slov, např.
`https://wiki.korpus.cz/doku.php/seznamy:srovnavaci_seznamy`
(přes 300 tisíc slovních tvarů)

Výstup: největší množina slov, které jsou vzájemnou přesmyčkou

vlasti, slavit, vstali, stavil, svitla, svalit
vynikal, vanilky, vynikla, navykli, vnikaly, viklany
kotle, loket, kotel, lokte, teklo, otekl
(8 řádků kódu v Pythonu, výpočet pod 1 sekundu)

Dnešní přednáška

- práce se soubory – rychlý úvod
- použití datových struktur na praktických příkladech
- interakce volby dat a algoritmu
- programy „na jeden slide“, ale již netriviální
- vnořené datové struktury: seznam seznamů, slovník indexovaný dvojicí, slovník slovníků…

Práce se soubory

Proč?

- vstupní data
- uložení výstupu programu
- zachování „stavu“ programu mezi jednotlivými běhy

větší projekty: databáze

Práce se soubory

základní postup:

- otevření souboru
- práce se souborem (čtení / zápis)
- zavření souboru

Práce se soubory: otevření, uzavření

- `f = open(filename, mode)`
- jméno souboru: řetězec
- způsob otevření:
  - čtení (`"r"`)
  - zápis (`"w"`) – přepíše soubor, pokud není, vytvoří jej
  - přidání na konec (`"a"`)
  - další možnosti: čtení i zápis, binární režim
- uzavření: `f.close()`

Práce se soubory: čtení, zápis

- `f.write(text)` – zapíše řetězec do souboru
  - neukončuje řádky, je třeba explicitně použít `'\n'`
- `f.readline()` – přečte jeden řádek
- `f.readlines()` – přečte všechny řádky, vrací seznam řádků
- `f.read(count)` – přečte daný počet znaků
- `f.read()` – přečte celý soubor, vrací řetězec
- `f.tell()` – aktuální pozice v souboru
- `f.seek(position)` – přesun pozice v souboru

Práce se soubory: iterování po řádcích

```
for line in f.readlines():
    print(line)
```

Alternativní způsob:

```
for line in my_file:
    print(line)
```

Frekvence slov v souboru

```
def word_freq_file(filename):
    f = open(filename)
    text = ""
    for line in f.readlines():
        text += line
    output_word_freq(text)     # -> předchozí přednáška
    f.close()


word_freq_file("devatero_pohadek.txt")
```

Frekvence slov v souboru: úkol

Upravte výpis, aby vypisoval nejčastější slova zadané minimání délky.

a	2426	jsem	287
se	1231	jako	284
na	792	když	281
to	781	řekl	251
v	468	nebo	120
že	467	ještě	110
je	446	pane	109
si	401	toho	91
tak	384	povídá	83
do	365	král	80

Práce se soubory: with

Speciální blok `with`

- není třeba soubor zavírat (uzavře se automaticky po ukončení bloku)
- souvislost s výjimkami

```
with open("/tmp/my_file", "r") as my_file:
    lines = my_file.readlines()

print(lines)
```

Zápis do souboru: Cenzura dlouhých slov

```
report.txt:
Climate models project robust 7 differences in regional climate
characteristics between present-day and global warming of 1.5 degree C,
and between 1.5 degree C and 2 degree C. These differences include
increases in: mean temperature in most land and ocean regions (high
confidence), hot extremes in most inhabited regions (high confidence),
heavy precipitation in several regions (medium confidence), and the
probability of drought and precipitation deficits in some regions (medium
confidence).

censor_report.txt:
XXX XXX XXX XXX 7 XXX in XXX XXX
XXX XXX XXX and XXX XXX of 1.5 XXX C,
and XXX 1.5 XXX C and 2 XXX C. XXX XXX XXX
XXX in: mean XXX in most land and XXX XXX XXX
XXX hot XXX in most XXX XXX XXX XXX
XXX XXX in XXX XXX XXX XXX and the
XXX of XXX and XXX XXX in some XXX XXX
XXX
```

Námět na cvičení: zachování interpunkce (tečky, čárky, závorky).

Zápis do souboru

Přímočará realizace cenzury dlouhých slov:

```
def censorship(filename, limit=5):
    f_in = open(filename)
    f_out = open("censor_"+filename, "w")
    for line in f_in:
        out_line = ""
        for word in line.split():
            if len(word) < limit:
                out_line += word + " "
            else:
                out_line += "XXX "
        f_out.write(out_line+"\n")
    f_in.close()
    f_out.close()
```

Osmisměrka

vstup (v souboru): mřížka písmen + seznam slov k vyhledání

```
avelu
elsok
skapr
teriz
unped
----
pes
vlk
lev
prase
kos
kapr
```

Osmisměrka: reprezentace dat

Jakou datovou strukturu použijeme?

- pro reprezentaci mřížky
- pro uložení hledaných slov

Načtení ze souboru

```
def read_data(filename="osmismerka.txt"):
    f = open(filename)
    grid = []
    words = []
    reading_grid = True
    for line in f.readlines():
        line = line.strip()
        if reading_grid:
            if line[0] == "-":
                reading_grid = False
            else:
                grid.append(line)
        else:
            words.append(line)
    f.close()
    return grid, words
```

Testování přítomnosti slova

- pro každou pozici a směr vyzkoušíme, zda se slovo vyskytuje
- jak zapsat elegantně?

Testování přítomnosti slova: pokus 1

Základní logika ilustrovaná pro fixní slovo, souřadnici a směr:

```
def test_word_naive1(grid):
    return grid[4][2] == "p" and \
           grid[3][1] == "e" and \
           grid[2][0] == "s"
```

Testování přítomnosti slova: pokus 2

Fixní směr (vlevo nahoru):

```
def test_word_naive2(grid, word, x, y):
    for i in range(len(word)):
        if grid[x-i][y-i] != word[i]:
            return False
    return True
```

Chyba: Nekontrolujeme kraje herního plánu, přetýkání do záporných čísel.

Testování přítomnosti slova

```
DIRS = [(0, 1), (1, 0), (0, -1), (-1, 0),
        (1, 1), (1, -1), (-1, 1), (-1, -1)]

def is_inside_grid(grid, x, y):
    return x >= 0 and y >= 0 and\
        y < len(grid) and x < len(grid[y])

def test_word(grid, word, x, y, direction):
    for i in range(len(word)):
        x2 = x + i*DIRS[direction][0]
        y2 = y + i*DIRS[direction][1]
        if not is_inside_grid(grid, x2, y2) or \
           grid[y2][x2] != word[i]:
            return False
    return True
```

Prohledání všech pozic a směrů

```
def search_word(grid, word):
  for y in range(len(grid)):
    for x in range(len(grid[y])):
      for d in range(len(DIRS)):
        if test_word(grid, word, x, y, d):
          highlight_word(grid, word, x, y, d)
          return True
  return False
```

Zvýraznění slova

```
def highlight_word(grid, word, x, y, direction):
    output_grid = [list(line) for line in grid]
    for i in range(len(word)):
        x2 = x + i*DIRS[direction][0]
        y2 = y + i*DIRS[direction][1]
        output_grid[y2][x2] = word[i].upper()
    for y in range(len(output_grid)):
        print("".join(output_grid[y]))
```

Pozn. Převod ze seznamu řetězců na seznam seznamů (důvod: (ne)měnitelnost).

Zpracování dat: hodnocení filmů

```
Matrix: 2
Pupendo: 3
Titanic: 2
Matrix: 3
Titanic: 4
Kolja: 4
Rocky: 1
Titanic: 3
...
```

$\Rightarrow$ výpis filmů: počet hodnocení, průměrné hodnocení

Pozn. Netflix prize, predikce hodnocení, 1 milion dolarů

Filmy: Reprezentace dat

slovník:

- klíč: název filmu
- hodnota: seznam hodnocení

```
{'Matrix': [2, 3, 1, 3],
 'Pupendo': [3],
 'Titanic': [2, 4, 3, 5],
 'Kolja': [4, 4],
 'Rocky': [1, 2, 2]}
```

Filmy: Načtení dat

```
def read_ratings(filename="filmy.txt"):
    f = open(filename)
    ratings = {}
    for line in f.readlines():
        movie, rating = line.strip().split(":")
        if movie not in ratings:
            ratings[movie] = []
        ratings[movie].append(int(rating))
    f.close()
    return ratings
```

Filmy: přímočarý výpis

```
def print_ratings(ratings):
    for movie in ratings:
        count = len(ratings[movie])
        avg = sum(ratings[movie]) / count
        print(movie, count, avg, sep="\t")
```

Filmy: řazený výpis

```
def print_ratings(ratings):
    table_data = []
    for movie in ratings:
        count = len(ratings[movie])
        avg = sum(ratings[movie]) / count
        table_data.append((avg, count, movie))
    table_data.sort()
    for avg, count, movie in table_data:
        print(movie, count, round(avg, 2), sep="\t")
```

Číselné bludiště

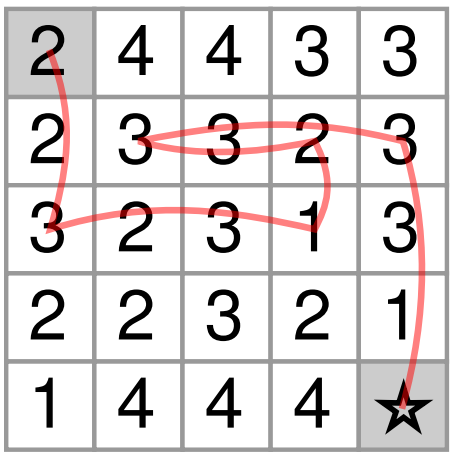
jednoduchá logická úloha

Ilustrace pojmů a postupů:

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

Číselné bludiště

levý horní roh → 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.umimeinformatiku.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?

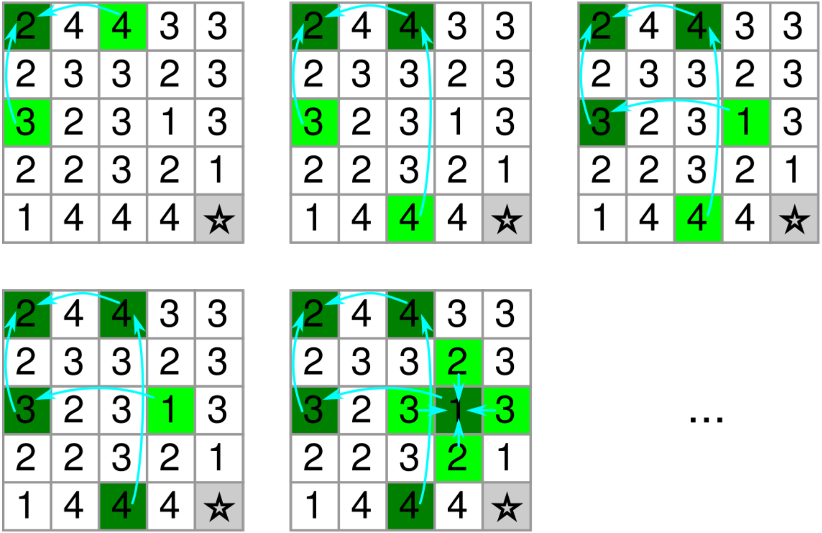
2	4	4	3	3
2	3	3	2	3
3	2	3	1	3
2	2	3	2	1
1	4	4	4	☆

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 $(x, y) \rightarrow$ 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

(*) 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, new
```

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)

Největší přesmyčkové skupiny

Vstup: seznam českých slov, např.
`https://wiki.korpus.cz/doku.php/seznamy:srovnavaci_seznamy`
(přes 300 tisíc slovních tvarů)

Výstup: největší množina slov, které jsou vzájemnou přesmyčkou

vlasti, slavit, vstali, stavil, svitla, svalit
vynikal, vanilky, vynikla, navykli, vnikaly, viklany
kotle, loket, kotel, lokte, teklo, otekl
(8 řádků kódu v Pythonu, výpočet pod 1 sekundu)

Největší přesmyčkové skupiny

- kontrola přesmyček – stejný kanonický tvar (= seřazená písmena)
- přesmyčková skupina – seznam slov se stejným kanonickým tvarem
- reprezentace: slovník seznamů
- `"ailstv"` $\rightarrow$ `["vlasti", "slavit", "vstali", "stavil", "svitla", "svalit"]`

Největší přesmyčkové skupiny

Kompaktní řešení (avšak ne extra čitelné):

```
from collections import defaultdict

f = open("syn2015_word_utf8.tsv")
words = [line.split("\t")[1] for line in f]
word_groups = defaultdict(list)
for word in words:
    word_groups["".join(sorted(word))].append(word)
for w in sorted(word_groups,
                key=lambda w: -len(word_groups[w]))[:10
    print(word_groups[w])
```

Piškvorky

Jak reprezentovat herní plán?

- plán omezené velikosti
- neomezený plán

„chytrý“ algoritmus?

Práce s plánem

- globální proměnné
  - `draw_board()`
  - `make_move(x, y)`
- předávání funkcím
  - `draw_board(board, size)`
  - `make_move(board, size, x, y, player)`
- objektová reprezentace (více později)
  - `game.draw_board()`
  - `game.make_move(x, y)`

Implementace strategií: nevhodný styl

```
def strategy(num):
    if num == 0: # random strategy
        x = randint(1, N)
        y = randint(1, N)
    elif num == 1: # first empty
        for x in range(N):
            for y in range(N):
                s = board[x][y]
                # and so on
    elif num == 2: # clever
        # and so on...
```

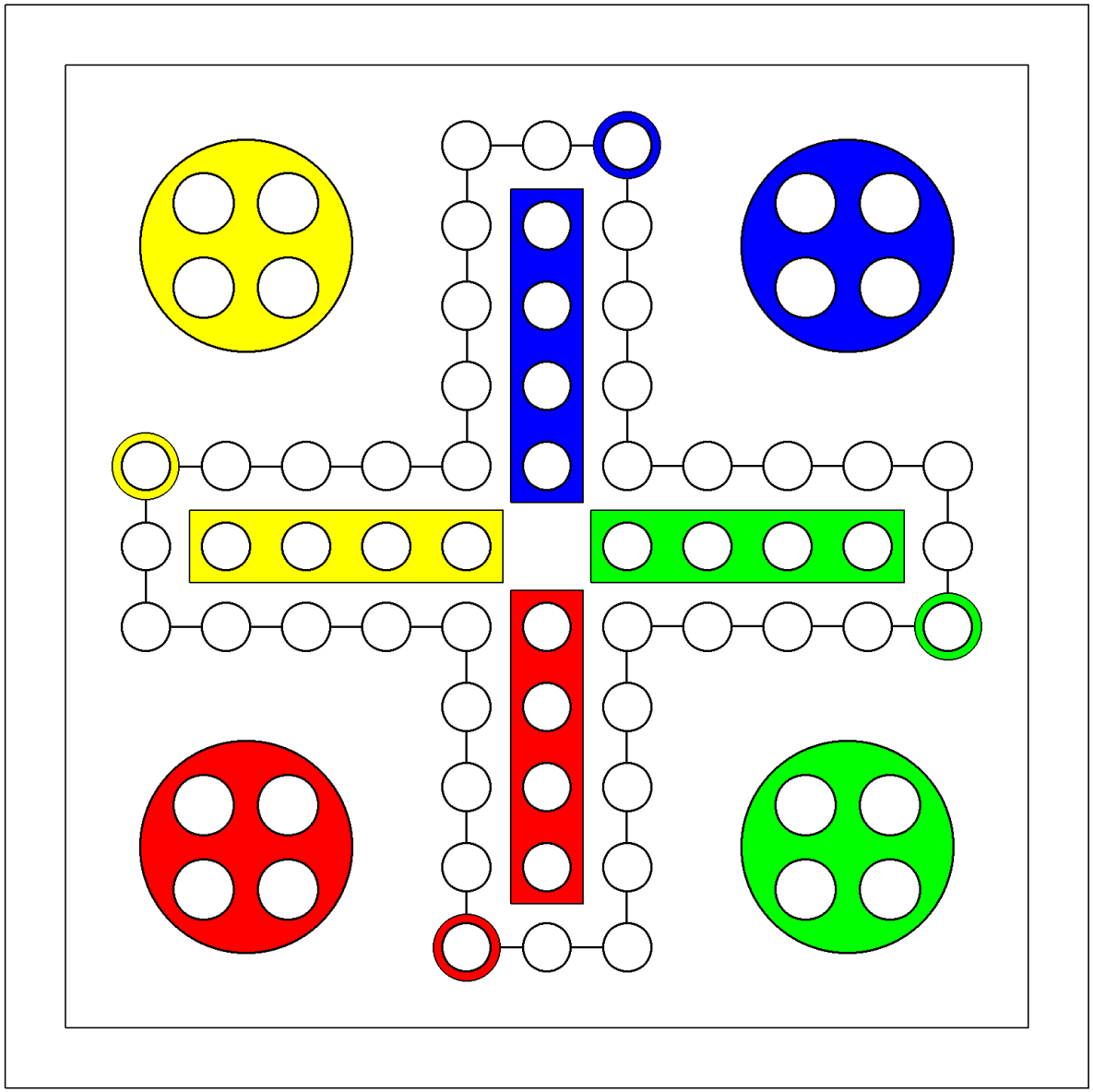
Jak vylepšit?

Volba datové struktury

Jaká datová struktura pro reprezentaci stavu?

- 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?

Kontrolní otázky

- Jakým způsobem pracujeme se soubory (otevření, čtení, zápis, uzavření)?
- Pomocí jakých příkazů projdeme všechny řádky textového souboru? V jakém pořadí je musíme použít?
- Jakými způsoby můžeme v Pythonu reprezentovat dvourozměrnou mřížku?
- Uveďte příklady použití následujících vnořených datových struktur: slovník seznamů, seznam seznamů.
- Ve slovníku `freq` máme napočítané frekvence slov v zadaném textu. Jakým způsobem vypíšeme slova seřazená podle těchto frekvencí?

Doporučené procvičování

`https://www.umimeinformatiku.cz/porozumeni`

⇒ sada „Slovníky“

`https://www.umimeinformatiku.cz/rozhodovacka`

⇒ sada „Přehled datových typů“

Shrnutí

- načtení dat ze souboru
- volba reprezentace dat
- vnořené datové struktury: seznam řetězců, slovník seznamů, seznam seznamů, slovník indexovaný dvojicí
- algoritmy, řazení výstupu

příště: „záludnosti“ $\sim$ proměnné, typy, reprezentace v paměti, rekurze