FAKULTA
INFORMATIKY
Masarykova univerzita

IB111 Základy programovania

Cvičenie 8 – dátové štruktúry

Matej Troják

(Poďakovanie: Valdemar Švábenský)

Dátový typ

- Dátový typ
  - Množina hodnôt s preddefinovanými operáciami na týchto hodnotách
  - Príklad: `bool =({True, False}; and, or, not)`
- Abstraktný datový typ (ADT)
  - Rozhranie a popis operácií, ktoré chceme používať (matematická špecifikácia)
  - Príklad: ADT `zoznam`
    operácie: `create(), add(x), remove(x), is_empty(), ...`
- Konkrétna dátová štruktúra
  - Implementácia, popis uloženia dát v pamäti, definícia funkcií
  - Príklad: `zoznam` v Pythone:
    operácie: `[ ], append(x), remove(x), len(lst)==0, ...`

Príklady ADT a ich implementácií

ADT

- Zoznam
- N-tica
- Zásobník
- Fronta
- Množina
- Slovník (mapa, asoc. pole)

⋮

Dátová štruktúra (Python)

- Zoznam (list)
- N-tica (tuple)
- Zoznam (list)
- Zoznam (list)
- Množina (set)
- Slovník (dict)

⋮

1. Zoznam (list)
2. N-tica (tuple)
3. Zásobník (stack)
4. Fronta (queue)
5. Množina (set)
6. Slovník (dict)

Zoznam – operácie

```
s = [4, 1, 3]      # zoznam
s.append(x)        # prida prvok x na konec
s.extend(t)        # prida vsetky prvky t na konec
s.insert(i, x)     # prida prvok x pred prvok na pozicii i
s.remove(x)        # odstrani prvy prvok rovny x
s.pop(i)           # odstrani (a vrati) prvok na pozicii i
s.pop()            # odstrani (a vrati) posledny prvok
s.index(x)         # vrati index prveho prvku rovneho x
s.count(x)         # vrati pocet vyskytov prvkov rovnych x
s.sort()           # zoradi zoznam
s.reverse()        # obrati zoznam
x in s             # test, ci zoznam obsahuje x
                   # (linearny prechod zoznamom!)
```

Zoznam – generátory

- *list comprehension*
- Špecialita Pythonu

```
>>> s = [x for x in range(1, 7)]
>>> print(s)
[1, 2, 3, 4, 5, 6]
```

```
>>> s = [2 * x for x in range(1, 7)]
>>> print(s)
[2, 4, 6, 8, 10, 12]
```

```
>>> s = [(a, b) for a in range(1, 4) for b in ["A", "B"]]
>>> print(s)
[(1, 'A'), (1, 'B'), (2, 'A'), (2, 'B'), (3, 'A'), (3, 'B')]
```

Vnorené zoznamy

- Prvky zoznamu môžu byť opäť zoznamy!
- Využitie: viacrozmerné dáta (napr. matice)

```
>>> mat = [[1, 2, 3],
           [4, 5, 6],
           [7, 8, 9]]
>>> print(mat[1][2])            # dvojite indexovanie
6
```

```
def null_matrix(m, n):
    return [[0 for col in range(n)] for row in range(m)]
```

Úloha 1 – vnorené zoznamy

- Napíšte funkciu `multiply_vector(matrix, vector)`, ktorá vynásobí maticu `matrix` s vektorom `vektor` a vráti vzniknutý vektor.

```
>>> matrix = [[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]]
>>> vector = [1, 10, 100]
>>> print(multiply_vector(matrix, vector))
[321, 654, 987]
```

- *Lineárna algebra* – násobenie matice s vektorom:

$$\mathrm{Ax} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \end{bmatrix}$$

- **Matematická poznámka**: riadky v matici nie sú vektory!
- *Bonus*: overte, či je násobenie možné.

1. Zoznam (list)
2. N-tica (tuple)
3. Zásobník (stack)
4. Fronta (queue)
5. Množina (set)
6. Slovník (dict)

N-tica (tuple)

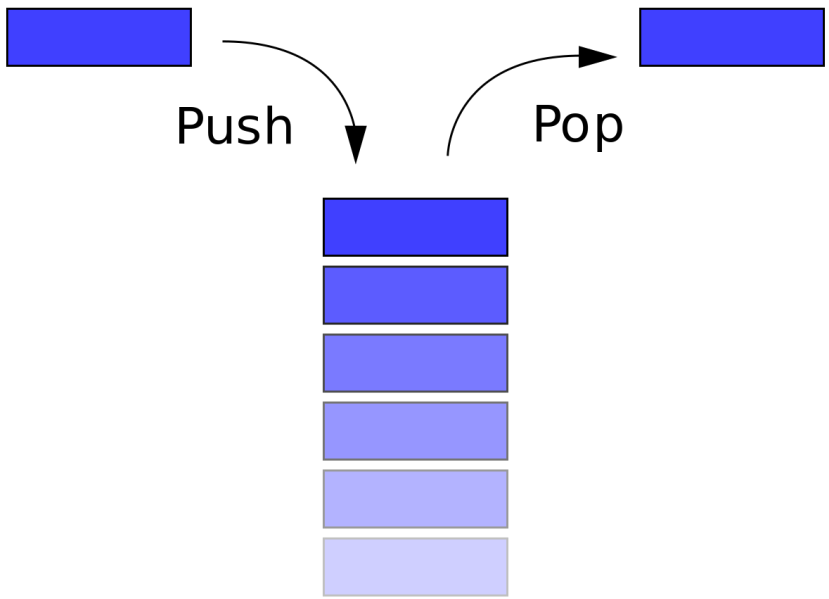
- Definícia: `(item1, item2, ... )`
  - Zátvorky sú nepovinná konvencia (konštruktor je čiarka)
- Presne ako zoznam, ale nemeniteľná
  - Nesmieme pridávať, modfikovať ani mazať prvky
  - Rýchlejšie, bezpečnejšie proti náhodným chybám

```
>>> my_tuple = (1, 2, 3)
>>> my_tuple[1]
2
>>> my_tuple[1] = 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

1. Zoznam (list)
2. N-tica (tuple)
3. Zásobník (stack)
4. Fronta (queue)
5. Množina (set)
6. Slovník (dict)

Zásobník (stack)

- Implementovaný pomocou zoznamu
- Pridávať a odoberať prvky môžeme len na jednom jeho konci (tzv. vrchol zásobníka)
- Nesmieme pridávať ani odoberať prvky na druhom jeho konci (tzv. dno zásobníka) ani nikde uprostred
- LIFO (Last In – First Out)



Operácie zásobníku

- Zásobník podporuje tieto (abstraktné) operácie:
  - `init()` – vytvorenie prázdneho zásobníka
  - `is_empty()` – kontrola na prázdnosť zásobníka
  - `push(x)` – pridanie prvku x na vrchol zásobníka
  - `pop()` – vrátenie a odstránenie prvku na vrchole zásobníka
  - `top()` – náhľad na prvok na vrchole zásobníka
- v Pythone sú implementované nasledovne:

```
>>> stack = []              # init()
>>> len(stack) == 0         # is_empty()
>>> stack.append(x)         # push(x)
>>> stack.pop()             # pop()
>>> stack[-1]               # top()
```

Úloha 2 – zásobník

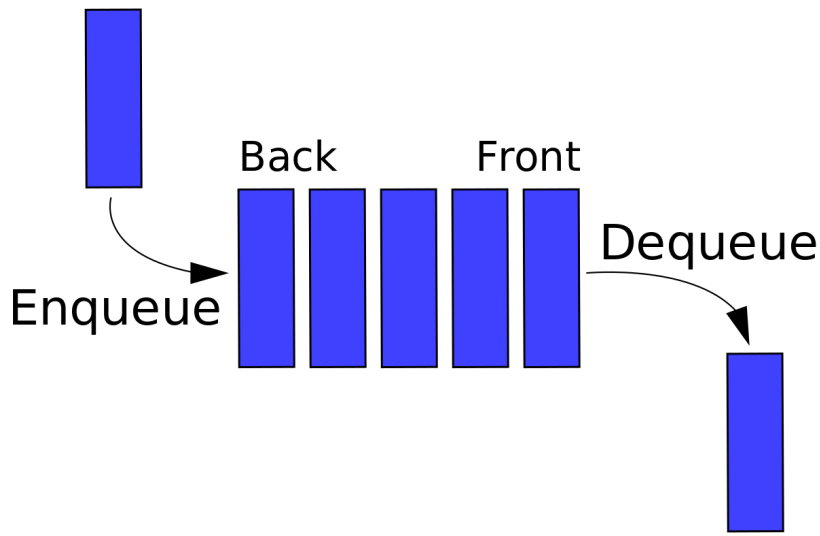
- Napíšte funkciu `parenthesis_check(value)`, ktorá s využitím zásobníka skontroluje, či je daný výraz správne uzátvorkovaný. Pre jednoduchosť môžete predpokladať, že výraz je utvorený iba z ľubovoľnej kombinácie znakov `(`, `[`, `{` (a opačných symbolov).

```
>>> parenthesis_check('([]({()}))[]{[()]}')
True
>>> parenthesis_check('([)]')
False
>>> parenthesis_check('()]')
False
```

1. Zoznam (list)
2. N-tica (tuple)
3. Zásobník (stack)
4. Fronta (queue)
5. Množina (set)
6. Slovník (dict)

Fronta (queue)

- Implementovaná pomocou zoznamu
- Pridávať prvky smieme len na jednom konci
- Odoberať prvky smieme len na druhom konci
- FIFO (First In – First Out)



Operácie fronty

- Fronta podporuje tieto (abstraktné) operácie:
  - `init()` – vytvorenie prázdnej fronty
  - `is_empty()` – kontrola na prázdnosť fronty
  - `enqueue(x)` – pridanie prvku x na koniec
  - `dequeue()` – odstránenie prvku, ktorý „je práve na rade"
  - `first()` – náhľad na prvý prvok
- v Pythone sú implementované nasledovne:

```
>>> queue = []                  # init()
>>> len(queue) == 0             # is_empty()
>>> stack.append(x)             # enqueue(x)
>>> stack.pop(0)                # dequeue()
>>> stack[0]                    # first()
```

1. Zoznam (list)
2. N-tica (tuple)
3. Zásobník (stack)
4. Fronta (queue)
5. Množina (set)
6. Slovník (dict)

Množina (set)

- Definícia: `set(item1, item2, ... )` alebo `{item1, item2, ... }`
- Podobná ako zoznam, ale:
  - Bez duplicitných položiek
  - Nezachováva poradie prvkov!

```
>>> my_list = ['aa', 'bb', 'aa', 'cc', 'bb']
>>> my_set = set(my_list)
>>> my_set
{'bb', 'cc', 'aa'}
```

- Množina je meniteľná
  - Metódy `add()`, `remove()`, `pop()`, `discard()`
- Je možné iterovať cez prvky množiny
  - žiadne garantované poradie!

Operácie na množinách

- Klasické množinové operácie: prienik, zjednotenie, rozdiel, symetrický rozdiel
- Test na podmnožinu/nadmnožinu

```
a.intersection(b)                 # a & b (Prienik)
a.union(b)                        # a | b (Zjednotenie)
a.difference(b)                   # a - b (Rozdiel)
a.symmetric_difference(b)         # a ^ b (Sym. rozdiel)
a.issubset(b)                     # Podmnozina
a.issuperset(b)                   # Nadmnozina
```

Úloha 3 – frekvenčná analýza s množinou

- Z Učebných materiálov si stiahnite súbor frequent.py (Skupiny 14 a 15 (Troják) / Cvičenie 08 / skripty /), v ktorom je definovaná funkcia `most_frequent_set(lst)`.
- Táto funkcia nájde najčastejší prvok v zozname `lst` s využitím množín. Funkcia vracia dvojprvkový zoznam `[item, number]`, kde `number` je počet výskytov prvku `item`.
- V implementácii je chyba! Funkcia sa nechová tak, ako by sme očakávali:

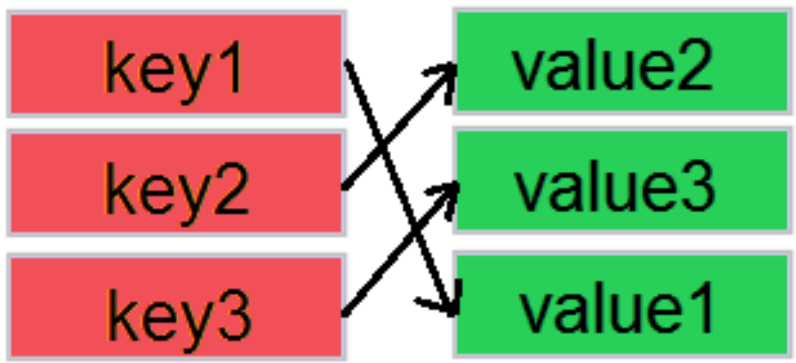
```
>>> most_frequent_set([1, 1, 5, 7, 8, 3, 8, 1, 8, 9, 1])
[1, 0]                                    # chceme [1, 4]
>>> most_frequent_set([9,0,4,1,7,0,2,0,4,9,1,5,7,4,9,9,7,9,2,5])
[0, 3]                                    # chceme [9, 5]
```

- Vašou úlohou je túto chybu **identifikovať a opraviť** tak (môžete upraviť maximálne jeden riadok!), aby funkcia vracala správne výsledky.

1. Zoznam (list)
2. N-tica (tuple)
3. Zásobník (stack)
4. Fronta (queue)
5. Množina (set)
6. Slovník (dict)

Slovník (dict)

- Kolekcia (neusporiadaných) párov (kľúč – hodnota)
- Každý **kľúč** je jedinečný (vyskytuje sa v slovníku len raz)
  - Kľúč je tak trochu ako index v zozname
  - Sprístupňujeme cez neho danú hodnotu
- **Hodnoty** sa v slovníku môžu opakovať
- v Pythone vstavaný dátový typ



Vytvorenie slovníka

- Vytvorenie slovíka priamo

```
# Osoba a jej oblubene ovocie
fruits = {
    "Peter" : "apple",
    "Jane" : "banana",
    "Mary" : "apple"
}
# alternativne
phonebook = dict(Peter=147258, Jane =789456, Mary=333333)
```

- Vytvorenie slovíka postupne

```
phonebook = {}
phonebook["Peter"] = 147258
phonebook["Jane"] = 789456
phonebook["Mary"] = 333333
```

Operácie na slovníkoch – základné

- `d = {}` – vytvorenie prázdneho slovníka
- `d[k] = v` – pridanie kľúča `k` do slovníka a jeho zviazanie s hodnotou `v`
- `d[k] = v` – zviazanie existujúceho kľúča `k` s inou hodnotou `v`
- `del d[k]`, `d.pop(k)` – odobranie kľúča `k` (a zároveň aj odpovedajúcej hodnoty) zo slovníka
- `d[k]` – nájdenie hodnoty, ktorá je zviazaná s kľúčom `k`
- `len(d)` – počet prvkov v slovníku

Operácie na slovníkoch – pokročilé

- Syntax: `dictionary.function()`
  - `keys()` – všetky kľúče slovníka
  - `values()` – všetky hodnoty slovníka
  - `items()` – všetky dvojice (kľúč, hodnota), t.j. n-tice
  - `get(key)` – hodnota pre daný kľúč `key`
- **Pozor**! Vracajú iterátor (ak chceme `list`, nutnosť pretypovať)

```
>>> fruits = {"Peter" : "apple",
              "Jane" : "banana",
              "Mary" : "apple"}
>>> list(fruits.keys())
['Mary', 'Peter', 'Jane']
>>> list(fruits.values())
['apple', 'apple', 'banana']
>>> list(fruits.items())
[('Mary', 'apple'), ('Peter', 'apple'), ('Jane', 'banana')]
>>> fruits.get("Peter")
"apple"
```

Poznámka – predanie funkcie ako parametru

- **Spoiler alert**: v Pythone je všetko `object`
- Názov funkcie môžeme predať ako parameter inej funkcii a pracovať s ňou pomocou identifikátora parametru

```
def dict_pretty_print(d):
    for key in d.keys():
        print(key, ":", d[key], end=", ")
```

```
def dict_ugly_print(d):
    print(list(d.keys()))
    print(list(d.values()))
```

```
def my_fun(lst, print_strategy):
    my_dict = create_dict(lst)
    print_strategy(my_dict)
```

```
>>> lst = [9, 0, 44, 1, 7, 23, 2, 0, 44, 9, 1]
```

```
>>> my_fun(lst, dict_pretty_print)
0 : 2, 1 : 2, 2 : 1, 7 : 1,
9 : 2, 23 : 1, 44 : 2,
```

```
>>> my_fun(lst, dict_ugly_print)
[0, 1, 2, 7, 9, 23, 44]
[2, 2, 1, 1, 2, 1, 2]
```

Úloha 4 – frekvenčná analýza so slovníkom

- Napíšte funkciu `most_frequent_dict(lst)`, ktorá nájde najčastejší prvok v zozname `lst` s využitím slovníku. Funkcia vracia dvojprvkový zoznam `[item, number]`, kde `number` je počet výskytov prvku `item`.

```
>>> most_frequent_dict([1, 1, 5, 7, 8, 3, 8, 1, 8, 9, 1])
[1, 4]
>>> most_frequent_dict([9,0,4,1,7,0,2,0,4,9,1,5,7,4,9,9,7,9,2,5])
[9, 5]
```

- *Bonus*: využite toho, že funkcii `max` môžte predať **funkciu**, podľa ktorej sa má vyberať maximum. Elementárny príklad:

```
>>> def fun(x):
        return 1/x
>>> max([1, 2, 3, 4, 5], key=fun)
1
```

```
>>> def ID(x):
        return x
>>> max([1, 2, 3, 4, 5], key=ID)
5
```

Úloha 5

- Napíšte funkciu `most_frequent(lst, strategy)`, ktorá naformátovane vypíše najčastejší prvok v zozname `lst` s využitím danej stratégie `strategy`. Funkcia vracia dvojprvkový zoznam `[item, number]`, kde `number` je počet výskytov prvku `item`.

```
>>> most_frequent([1,1,5,7,8,3,8,1,8,9,1], most_frequent_dict)
Item 1 is the most frequent - 4 times.
>>> most_frequent([9,0,4,1,7,0,2,0,4,9,1], most_frequent_set)
Item 0 is the most frequent - 3 times.
```