FAKULTA
INFORMATIKY
Masarykova univerzita

IB111 Základy programovania

Cvičenie 6 – vstup, binárne vyhľadávanie

Matej Troják

(Poďakovanie: Valdemar Švábenský)

Typové anotácie

- Jednoduchý program bez typových anotácií

```
def greeting(name):
    return 'Hello ' + name
```

- Ten istý program s typovými anotáciami

```
def greeting(name: str) -> str:
    return 'Hello ' + name
```

- Dokumentácia k typovým anotáciám:
  https://mypy.readthedocs.io/en/latest/getting_started.html()

Typové anotácie

- Majme program `program.py` s týmto obsahom

```
def greeting(name: str) -> str:
    return 'Hello ' + name

greeting(3)
```

- Môžeme zavolať typový checker `mypy` na tomto súbore

```
>>> python3.6 -m mypy program.py
program.py:4: error: Argument 1 to "greeting" has incompatible type
                     "int"; expected "str"
Found 1 error in 1 file (checked 1 source file)
```

Typové anotácie

- Pre zložitejšie typové anotácie, môžeme využiť modul `typing`

```
from typing import List

def greet_all(names: List[str]) -> str:
    return "Hello " + " and ".join(names)

names = ["Alice", "Bob", "Charlie"]
ages = [10, 20, 30]

print(greet_all(names))   # Ok!
print(greet_all(ages))    # Error due to incompatible types
```

- Výstup z `mypy`:

```
program.py:10: error: Argument 1 to "greet_all" has incompatible type
                    "List[int]"; expected "List[str]"
Found 1 error in 1 file (checked 1 source file)
```

Testovanie

- Jedna z možností ako napr. overiť, že funkcia dostala správnu hodnotu daného typu je pomocou príkazu **`assert`**

```
assert <condition>

assert <condition>, <error message>
```

- Ak je podmienka `<condition>` vyhodnotená na `True`, tak program normálne pokračuje
- Inak program vyhodí chybu `AssertionError`

Testovanie

- Jednoduchý program na výpočet priemeru známok

```
def avg(marks):
    assert len(marks) != 0, "No marks given!"
    return sum(marks) / len(marks)

marks_1 = []
print("Average of marks_1:", avg(marks_1))
```

- Príkazom **assert** si overím, že nebudem deliť nulou

```
>>> python3.6 program.py
Traceback (most recent call last):
  File "program.py", line 6, in <module>
    print("Average of marks_1:", avg(marks_1))
  File "program.py", line 2, in avg
    assert len(marks) != 0, "No marks given!"
AssertionError: No marks given!
```

Úloha 0

- Na príklade `1a` `find_max(lst)` z minulého cvičenia urobte:
  - kompletnú typovú anotáciu (predpokladajte, že vstupný zoznam obsahuje iba celé čísla)
  - overte pomocou príkazu **`assert`**, že zadaný zoznam nie je prázdny

1. Testovanie a typy

2. Vstup

3. Vyhľadávanie

Funkcia input()

```
number = input('Choose a number: ')
```

- Funkcia `input(string)`
  - Vypíše do shellu výzvu `string`
  - Načíta vstup z klávesnice potvrdený klávesou Enter
  - Vráti vstup ako reťazec (`str`)
- Vstup je vhodné testovať (neprázdny, korektné znaky, . . .)

```
if len(number) > 0 and number.isdigit():
    print('Number', number, 'is a good choice.')
else:
    print('Sorry, what?')
```

Hra na hádanie čísel

Pravidlá:

- Máme dvoch hráčov, A a B
- A si myslí prirodzené číslo od 1 do n
- B tipuje čísla
- A odpovedá: moje číslo je menšie/rovné/väčšie ako tvoj tip

Otázka:

- Akú stratégiu má B použiť, aby čo najrýchlejšie uhádol?

Úloha 1

- Napíšte funkciu `guess_number(n)` na hru Hádanie čísel.
  - Hráč A je počítač, "vymyslí" si číslo od 1 do `n`.
  - Hráč B je človek, tipuje čísla pomocou funkcie `input`.
- Počítač odpovedá, či je tipnuté číslo väčšie/menšie/rovnaké ako hádané číslo.

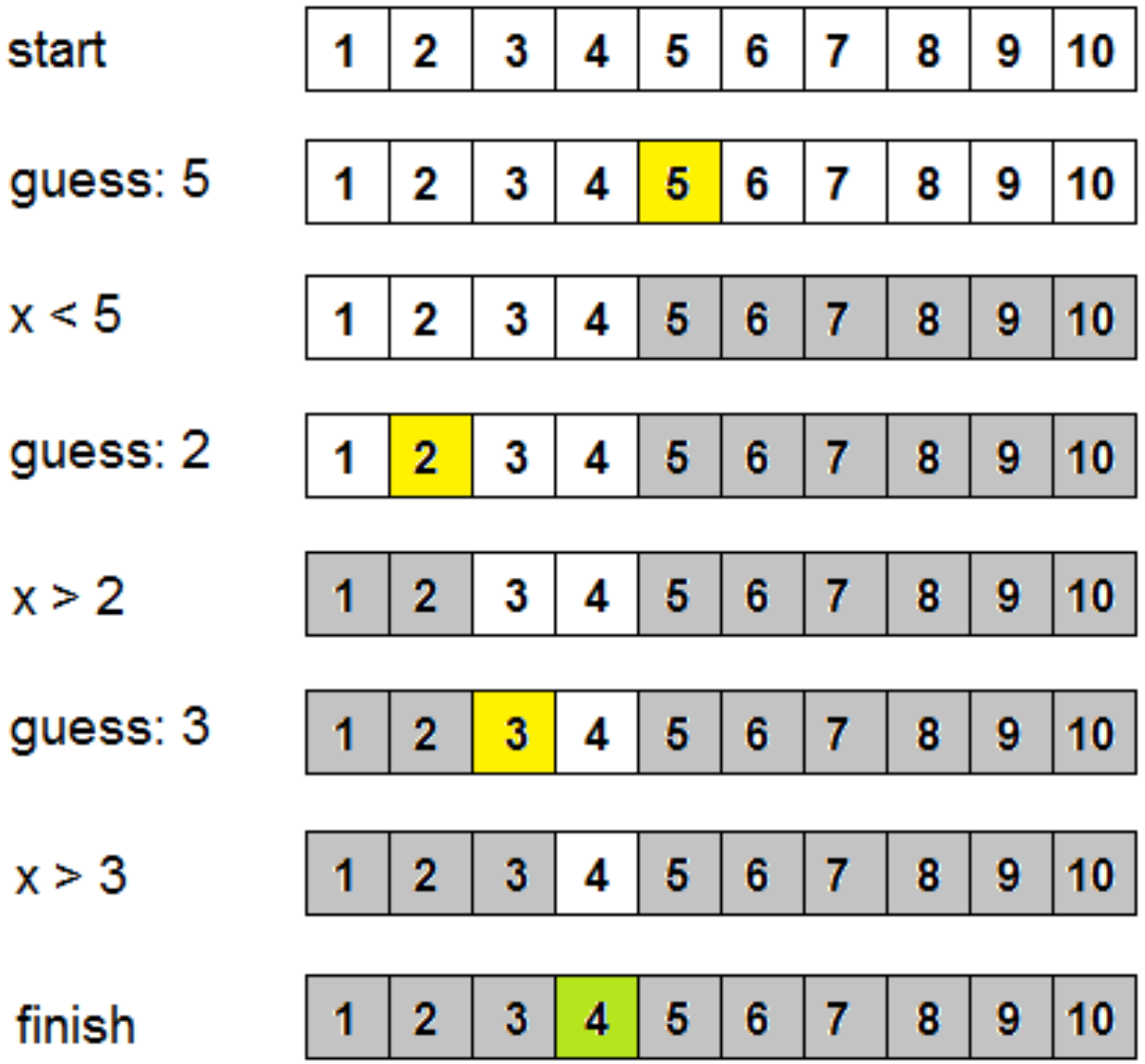
```
>>> guess_number(10)                    # From 1 to 10
Attempt 1
Type your guess: 4
My number is greater.
Attempt 2
Type your guess: 8
My number is lower.
Attempt 3
Type your guess: 6
Correct!
```

Úloha 2

- Napíšte funkciu `guess_number_computer(n)` na hru Hádanie čísel.
  - Hráč A je človek, myslí si číslo od 1 do `n`.
  - Hráč B je počítač a pýta sa, či je hľadané číslo menšie, rovné alebo väčšie ako jeho tip, človek odpovedá pomocou funkcie `input`.

```
>>> guess_number_computer(10)
Think of a number x from 1 to 10.
Is x < 5 (a), x == 5 (b), or x > 5 (c)?
a
Is x < 2 (a), x == 2 (b), or x > 2 (c)?
c
Is x < 3 (a), x == 3 (b), or x > 3 (c)?
c
Your number is 4
```

Úloha 2 – priebeh



1. Testovanie a typy

2. Vstup

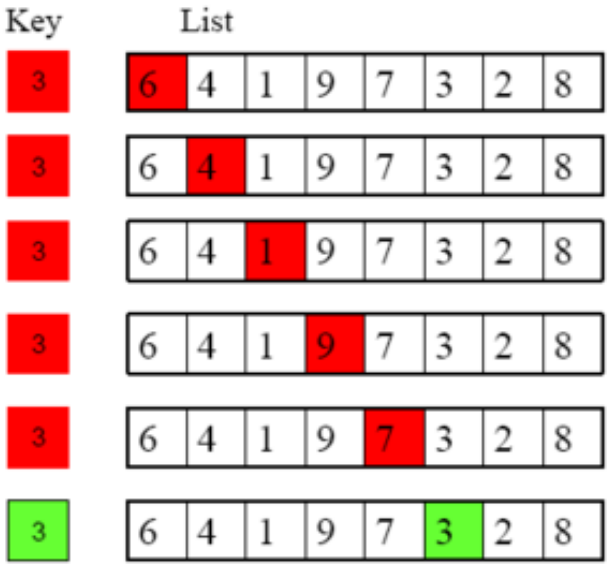
3. Vyhľadávanie

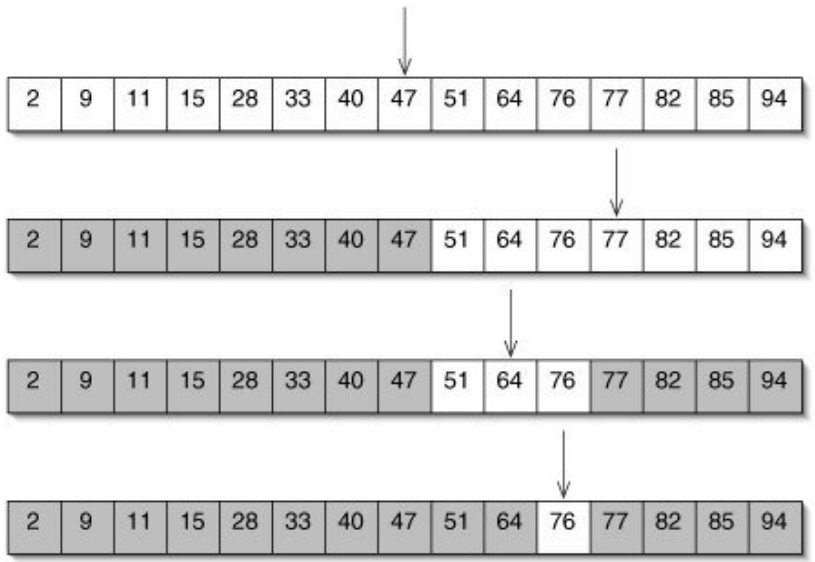
Problém vyhľadávania

Hra je podobná problému vyhľadávania:

- Nájdenie jednej hodnoty v zozname hodnôt
- Odpoveď: je/nie je tam (prípadne index výskytu)
- Príklad: hodnota 42 je v zozname [0, 14, 27, 42, 55] (na indexe 3)
- Dva typy vyhľadávania:
  - *Lineárne*: postupné prechádzanie všetkých prvkov zoznamu po jednom (operátor `in`)
  - *Binárne*: postupné delenie usporiadaného zoznamu na polovice

Lineárne vs. binárne vyhľadávanie

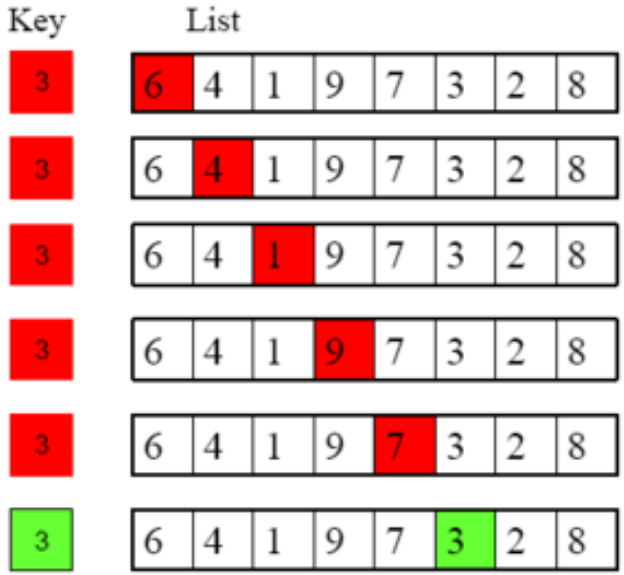




- Lineárne vyhľadávanie
- Nezoradený zoznam
- Časová zložitosť *O(n)* (lineárna)

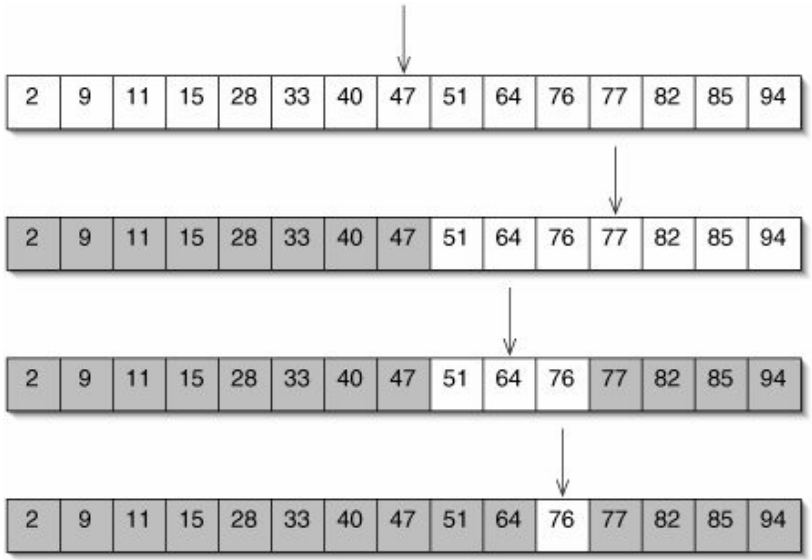
- Binárne vyhľadávanie
- Zoradený zoznam
- Časová zložitosť *O(log(n))* (logaritmická)

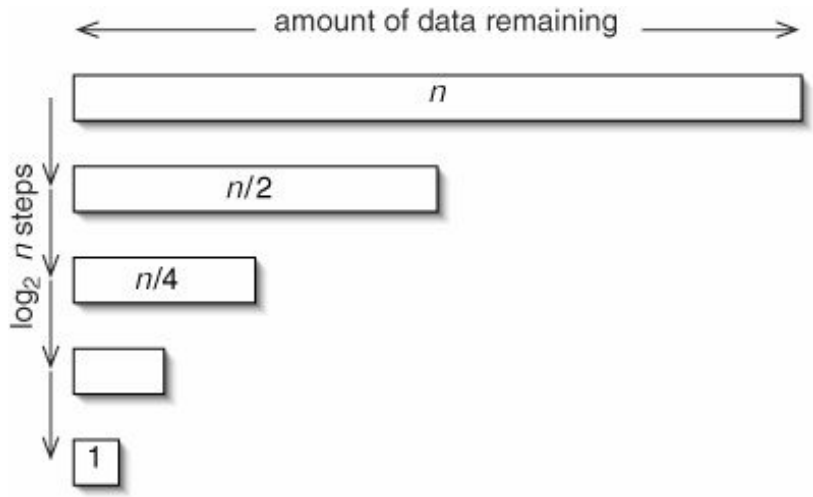
Lineárne vyhľadávanie – analýza času



- Ak n je počet prvkov zoznamu, algoritmus spraví *najviac* n porovnaní (teda skontroluje najviac n prvkov)
- Časová zložitosť *O(n)* (lineárna vzhľadom k počtu prvkov)
  - S rastúcim n predpokladáme lineárny nárast počtu krokov algoritmu
  - Pre 1000000 prvkov maximálne 1000000 krokov

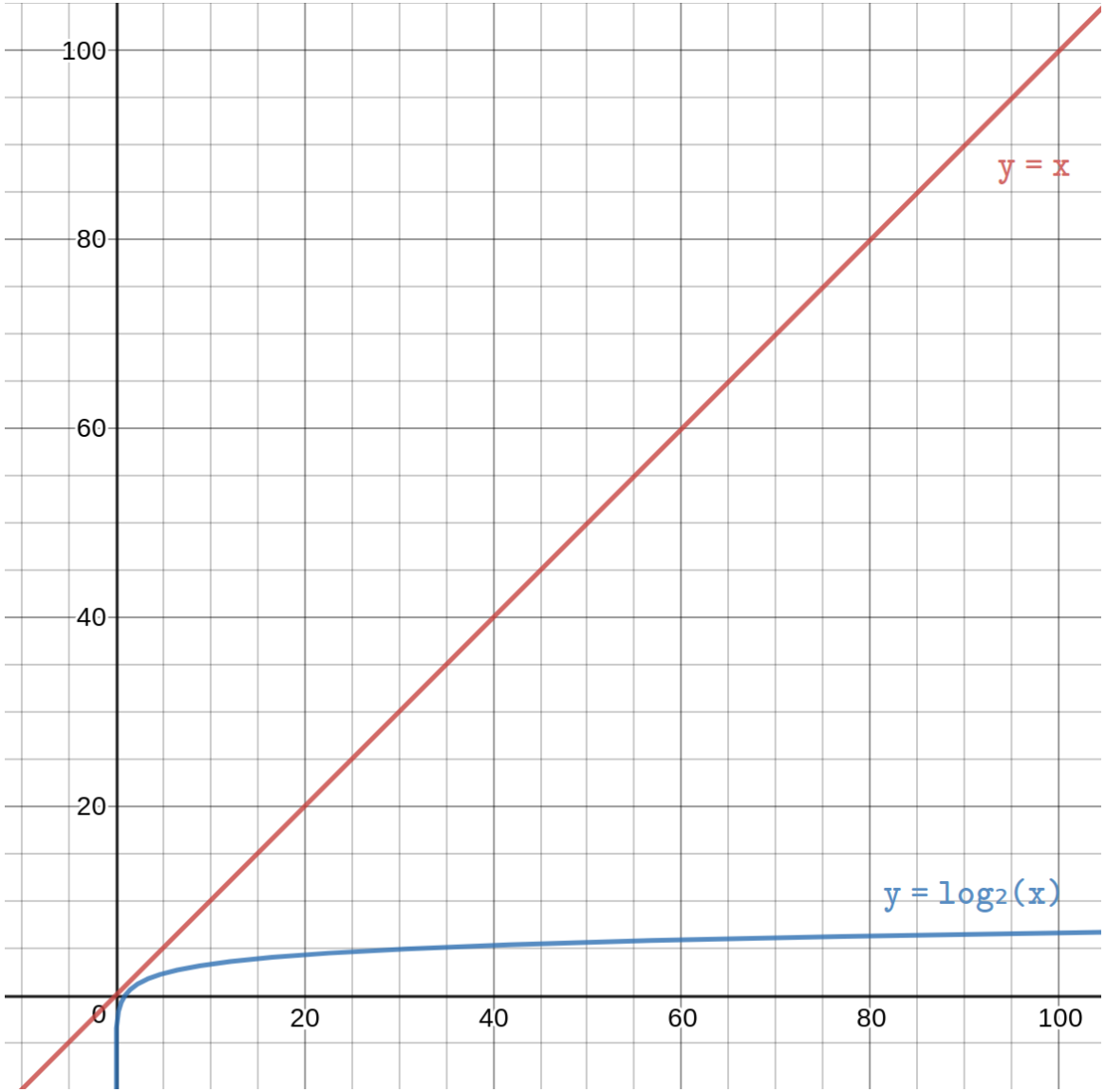
Binárne vyhľadávanie – analýza času





- Ak n je počet prvkov zoznamu, algoritmus spraví najviac $O(log_2(n))$ porovnaní (po každom kroku mu ostane $\frac{1}{2}$ prvkov)
- Časová zložitosť $O(log_2(n))$ (logaritmická vzhľadom k počtu prvkov)
  - S rastúcim n predpokladáme logaritmický nárast počtu krokov algoritmu
  - Pre 1000000 prvkov maximálne 20 krokov

Lineárna vs. logaritmická funkcia



Úloha 3

- Napíšte funkciu `binary_search(needle, haystack)`, ktorá zistí, či sa hodnota `needle` nachádza v usporiadanom zozname `haystack`. Funkcia musí mať logaritmickú časovú zložitosť.

```
>>> binary_search(7, [1, 4, 5, 7, 9])
True
>>> binary_search(7, [])
False
```

- *Bonus (a)*: upravte funkciu tak, aby fungovala aj pre neuspriadaný zoznam (funkcia `sorted`).
- *Bonus (b)*: upravte funkciu tak, aby vracala index, kde sa hľadaný prvok nachádza, inak -1.

Úloha 4

a) Rozšírte funkciu `binary_search(needle, haystack)` z úlohy 3 o počítadlo krokov (iterácií hlavného cyklu).

b) Napíšte funkciu `analyse_search(length, repetitions=1000)`, ktorá pre náhodné zoznamy veľkosti `length` spočíta priemerný počet krokov binárneho vyhľadávania.

   - Vytvorte si pomocnú funkciu, ktorá vygeneruje zoznam náhodných `k` čísel, zoradí ho a vráti.
   - *Hint*: môžete využiť funkciu `random.sample`

```
>>> import random
>>> help(random.sample)
```

Úloha 5

- Zopakujte rovnaký postup pre linerárne vyhľadávanie (z minulého cvičenia) a výsledky porovnajte.

Úloha 6

- Upravte úlohu 2 tak, aby hráča A aj B reprezentoval iba počítač.

```
>>> guess_number_computer_computer(10)
A: I think of a number x from 1 to 10.
B: My guess is 5
A: x > 5
B: My guess is 8
A: x < 8
B: My guess is 6
A: Correct!
A: You got it on 3 attempts.
```