IB111

Základy programování

František Lachman lachmanfrantisek@mail.muni.cz

cvičení 6

23. říjen 2018

Osnova

- kontrolní otázky
- uživatelský vstup
- HI-LOW
- binární vyhledávání
- druhý domácí úkol

Kontrolní otázky

Docházka

Jaká je základní myšlenka algoritmu pro binární vyhledávání?

Co znamená pojem „složitost algoritmu“? Jaká je složitost algoritmů pro vyhledávání?

Jak získáme vstup od uživatele?

```
name = input("Zadej jmeno: ")
print(name)
```

Jak získáme vstup daného typu?

```
number = int(input("Zadej cislo: "))
print(number)
```

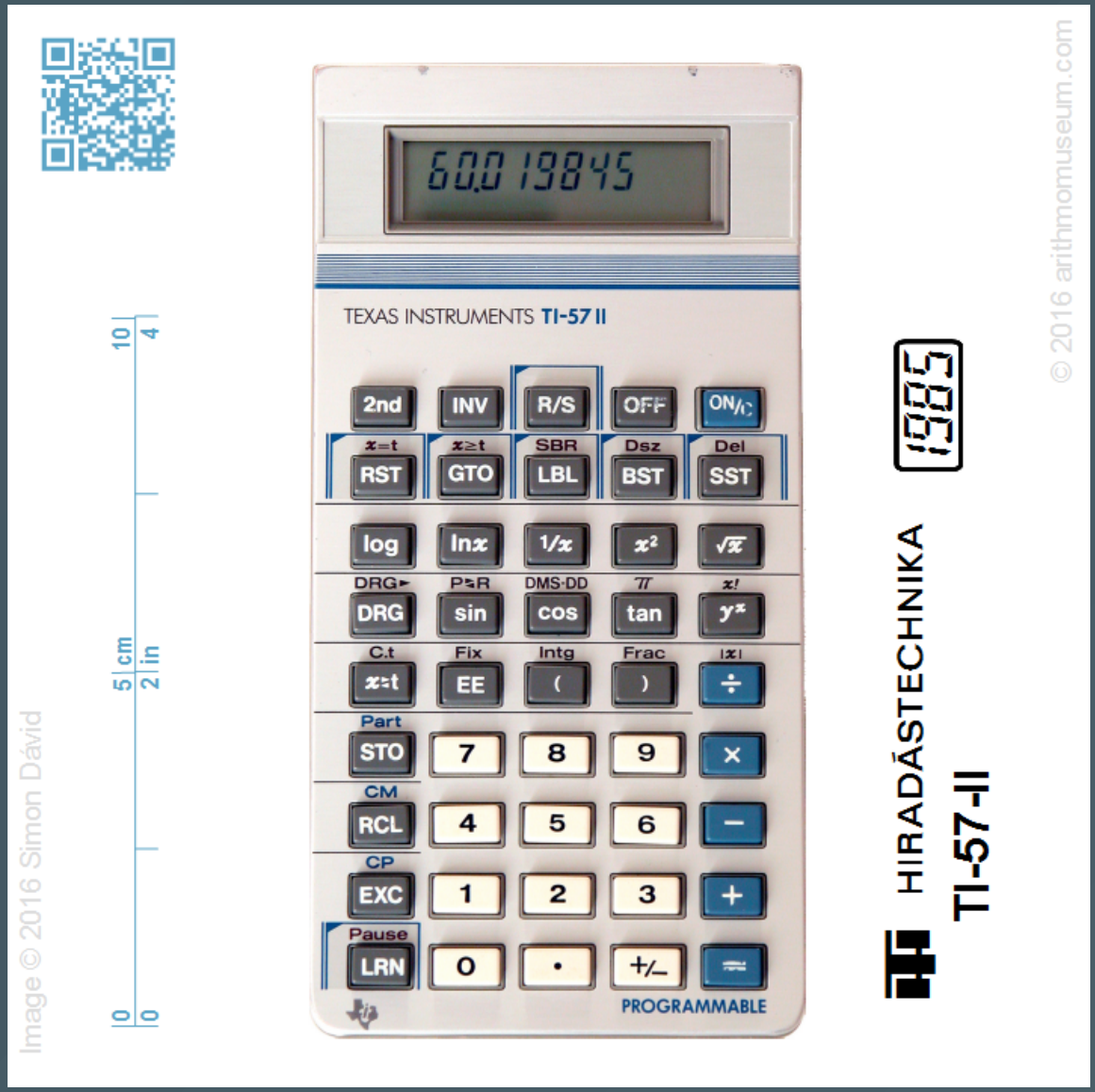
HI-LOW - hádá člověk 👨

6.1.1. Hádání čísla člověkem

Napište funkci `guess_number_human(upper_bound)`, která umožňuje hrát s počítačem hru na hádání čísla:

1. Počítač si myslí číslo (celé číslo v intervalu `[1, upper_bound]`).
2. Hráč se ho snaží uhodnout.
3. Po každém pokusu dostane hráč od počítače informaci, zda je hledané číslo menší nebo větší než to, které si tipnul.

Zvládne to i tohle!



HI-LOW - hádá člověk 👱

```
>>> guess_number_human(10)
--- pokus c. 1 ---
Zadej svuj tip: 5
Moje cislo je mensi.
--- pokus c. 2 ---
Zadej svuj tip: 3
Moje cislo je vetsi.
--- pokus c. 3 ---
Zadej svuj tip: 4
Jo, to je ono!
```

```
from random import randint

def guess_number_human(upper_bound):

    number = randint(1, upper_bound)

    user = None
    while user != number:
            user = int(input("Give me a number"))
            if user < number:
                print("too small")
            elif user > number:
                print("too big")

    print("Good!")

guess_number_human(20)
```

HI-LOW - hádá počítač 💻

6.1.2. Hádání čísla počítačem

Napište funkci `guess_number_pc(upper_bound)`, která umožňuje hrát s počítačem hru na hádání čísla, tentokrát si však číslo myslí uživatel a počítač hádá. Po každém pokusu si počítač vyžádá od uživatele informaci, zda je myšlené číslo větší nebo menší než to, které si počítač tipnul.

HI-LOW - hádá počítač 💻

```
>>> guess_number_pc(10)
Mysli si cislo od 1 do 10.
Je cislo 5 mensi (0), rovno (1), nebo vetsi (2) nez tvoje
2
Je cislo 2 mensi (0), rovno (1), nebo vetsi (2) nez tvoje
2
Tvoje cislo je 1.
```

```
def guess_number_pc(upper_bound):
    print("Mysli si cislo od 1 do", upper_bound)
    lower_bound = 1
    middle = int((lower_bound + upper_bound) / 2)
    while lower_bound != upper_bound:
        print("Je cislo", middle, "mensi (0), rovno (1), n
        answer = int(input(''))
        if answer == 0:
            lower_bound = middle + 1
        elif answer == 1:
            upper_bound = lower_bound = middle
        elif answer == 2:
            upper_bound = middle - 1
        middle = int((lower_bound + upper_bound) / 2)
    print("Tvoje cislo je", middle)

guess_number_pc(10)
```

HI-LOW - počítač 💻 ❌ 💻 počítač

6.1.3. Hádání čísla: počítač vs. počítač

Napište funkci `guess_number_pc_pc(upper_bound)`, která vykoná hru v hádání myšleného čísla počítače proti počítači. Na závěr vypíše počet pokusů potřebných k uhádnutí čísla.

HI-LOW - počítač 💻 ❌ 💻 počítač

```
>>> guess_number_pc_pc(10)
A: Myslim si cislo od 1 do 10
--- pokus c. 1 ---
B: Tipuji 5
A: Moje cislo je mensi.
--- pokus c. 2 ---
B: Tipuji 2
A: Moje cislo je vetsi.
--- pokus c. 3 ---
B: Tipuji 3
A: Jo, to je ono!
Uhadnuto na 3 pokusu
```

```
from random import randint
def guess_number_pc_pc(upper_bound):
    number = randint(1, upper_bound)
    print("A: Myslim si cislo od", 1, "do", upper_bound)
    attempts = 0
    lower_bound = 1
    done = 0
    while True:
        attempts += 1
        middle = int((lower_bound + upper_bound) / 2)
        print("B: Tipuji ", middle)
        if middle < number:
            print("A: Moje cislo je vetsi.")
            lower_bound = middle + 1
        elif middle > number:
            print("A: Moje cislo je mensi.")
            upper_bound = middle - 1
        else:
            print("A: Spravne!")
            done = 1
```

Binární vyhledávání

6.2.1. Binární vyhledávání

Napište funkci `binary_search(needle, haystack)`, která zjistí, zda se hodnota `needle` nachází ve vzestupně uspořádaném seznamu `haystack`. Funkce musí mít logaritmickou časovou složitost.

```
def binary_search(needle, haystack):
    lower_bound = 0
    upper_bound = len(haystack) - 1
    while lower_bound <= upper_bound:
        middle = int((lower_bound + upper_bound) / 2)
        if haystack[middle] == needle:
            return True
        elif haystack[middle] > needle:
            upper_bound = middle - 1
        else:
            lower_bound = middle + 1
    return False

print(binary_search(5, [1, 2, 5, 8]))
print(binary_search(4, [1, 2, 5, 8]))
```

Binární vyhledávání

6.2.2. Binární vyhledávání pozice

Napište funkci `binary_search_position(needle, haystack)`, která vylepší předchozí funkci `binary_search(needle, haystack)` tak, aby vracela index pozice, kde se hledaný prvek nachází. Pokud prvek v seznamu není, vraťte `-1`.

```
def binary_search_position(needle, haystack):
    lower_bound = 0
    upper_bound = len(haystack) - 1
    while lower_bound <= upper_bound:
        middle = int((lower_bound + upper_bound) / 2)
        if haystack[middle] == needle:
            return middle
        elif haystack[middle] > needle:
            upper_bound = middle - 1
        else:
            lower_bound = middle + 1
    return -1

print(binary_search_position(5, [1, 2, 5, 8]))
print(binary_search_position(4, [1, 2, 5, 8]))
```

Druhý domácí úkol - řešení

```
from random import randint


def dice():
    """
    Simulates roll of a dice

    :return: random int in range 1 - 6
    """
    return randint(1, 6)
```

Druhý domácí úkol - řešení

```
def throw_round():
    """
    Get the list of dice throws. (The six means another th

    :return: list of dice throws
    """
    throw_list = []
    throw = None
    while len(throw_list) == 0 or throw == 6:
        throw = dice()
        throw_list.append(throw)
    return throw_list
```

Druhý domácí úkol - řešení

```
def get_position_change(throw_list):
    """
    Get the change of position (use board rules).

    :param throw_list: list of dice throws
    :return: change of position for given dice throws
    """
    if throw_list.count(6) % 2 == 1:
        return 0
    return sum(throw_list)
```

Druhý domácí úkol - řešení

```
def draw_board(legnth, position, throw):
    """
    Draws a board of length 'length' with a player
    at specified location
    and the numbers the player threw.
    """
    for i in range(legnth):
        if i == (position - 1):
            print("(#)", end="")
        else:
            print("( )", end="")
    print(" ", end="")
    for i in throw:
        print("[" + str(i) + "]", end="")
    print()
```

Druhý domácí úkol - řešení

```
def board_game(length, output=True):
    """
    Simulates the board game.

    :param length: Length of the board.
    :param output: If True, prints the output.
    :return: Number of rounds.
    """
    if length < 1:
        if output:
            print("Length too small.")
        return -1
```

```
    position = 1
    rounds = 0
    while position != length:
        dice_roll = throw_round()
        position_change = get_position_change(dice_roll)

        if position_change + position <= length:
            position += position_change
        rounds += 1

        if output:
            draw_board(length, position, dice_roll)

    if output:
        print('The game ended after {} rounds'
                .format(rounds))
    return rounds
```

Druhý domácí úkol - řešení

```
def game_analysis(length, count):
    """
    Counts average length of a game.

    :param length: game plan length
    :param count:  number of games to make average from
    :return: average number of rounds it takes to finish t
    """
    counter = 0
    for i in range(count):
        counter += board_game(length, output=False)
    return counter / count
```

Druhý domácí úkol - řešení

```
def game_average_length(count):
    """
    Counts average game lengths for given number of games
    with board length from 1 to 50.

    :param count: number of games to count average from
    """
    for i in range(1, 51):
        print('Average game length for plan with length={}
                .format(i, game_analysis(i, count)))
```

Druhý domácí úkol - poznámky

Funkce nad seznamy

- Pokud není úkolem funkce naprogramovat, nebojte se je využívat.
- `len`, `count`, `sum`, ...

Druhý domácí úkol - poznámky

- Neiniciovat proměnné před cyklem, pokud to není potřeba.

```
counter = 0
for i in range(5):
    counter += i
return counter
```

```
throw = 0 # Není potřeba!
for i in range(5):
    throw = dice()
    print("throw:", throw)
```

Druhý domácí úkol - poznámky

Funkce `enumerate`

```
for i, element in enumerate("ahoj"):
    print("pořadí:", i, "prvek:", element)
```

```
pořadí: 0 prvek: a
pořadí: 1 prvek: h
pořadí: 2 prvek: o
pořadí: 3 prvek: j
```

Osnova

- kontrolní otázky
- uživatelský vstup
- HI-LOW
- binární vyhledávání
- druhý domácí úkol