`Programování` 1 pro matematiky

6. cvičení, 10-11-2022

tags: `Programovani 1 2022`, `čtvrtek`

Obsah:

Farní oznamy

1. **Materiály k přednáškám** najdete v GitHub repozitáři https://github.com/PKvasnick/Programovani-1. Najdete tam také kód ke cvičením.
2. **Organizace cvičení** Minulý týden se nám cvičení přirozeně rozdělilo - na dřívější termín přišli také všichni zájemci z pozdějšího termínu a cvičení v 17:20 jsem tak mohl věnovat doučování.

   Pokud by vám toto schéma vyhovovalo, byl bych rád, kdybychom u něj mohli zůstat.
3. **Domácí úkoly** Jsem spokojený s průběhem, jenom mi docházejí dobré příklady.

> "Without requirements or design , programming is the art of adding bugs to an empty text file." - Louis Srygley

K tomuto mě inspirovala některá řešení úlohy o rostoucí posloupnosti. Mějte na paměti specifikaci a řiďte se jí.

Kvíz

Co se vypíše? (inspirováno velkou náklonností některých z vás k příkazu `pass`)

```
>>> x = ...
>>> if x is ...:
        print(f"{x=}")

???
```

Ještě si můžeme připomenout jeden objekt, a to podtržítko `_`. To používáme tam, kde má být proměnná, ale chceme naznačit, že tato proměnná nás nezajímá :

```python
from random import randint
print([randint(1,10) for _ in range(10)])

_, x = 20 // 7, 20 % 7
```

Ještě medián

- Vaše řesení obsahovala v nějakém kroku setřídění načtené posloupnosti, něco jako

  `seznam.sort`

  anebo

  `seznam = sorted(seznam)`

- Je ale potřebné seznam setřrídit? O hodnotě mediánu přece nerozhoduje pořadí největších a nejmenších hodnot, takže bychom potenicálně mohli ušetřit nějakou práci.

Příbuzná úloha: Najít v seznamu k-tou největší hodnotu. Pro posloupnost s lichým počtem `n` členů je medián `n // 2`-á největší (nebo nejmenší) hodnota, pro sudé `n` potřebujeme členy `n//2 - 1` a `n//2`, takže pokud vyřešíme tuto obecnější úlohu, bude její řešení použitelné i pro medián.

Rozděl a panuj Budeme postupovat tak, že vytvoříme sérii částečných uspořádání tak, abychom po každém kroku mohli eliminovat část posloupnosti, ve které se medián určitě nenachází.

1. Zvolíme si hodnotu - *pivot* - a rozdělíme posloupnost na dva podseznamy, *mensí* a "větší", s hodnotami menšími, resp. většími nežli pivot.
2. Podle toho, do kterého seznamu připadne hledaný index k, pokračujeme jenom s jedním z těchto podseznamů a vracíme se do kroku 1.
3. Pokračujeme, dokud nedospějeme k podseznamu délky 1. Hodnota, kterou obsahuje, je to, co hledáme.

Otázka je, jak pro daný seznam zvolit pivot. My použijeme náhodný pivot - prostě náhodně zvolíme za pivot některou hodnotu ze seznamu. To není optimálné, ale funguje to docela dobře.

```python
seznam = [randint(1,100) for _ in range(10)]
# while (hodnota := int(input())) != -1:
#     seznam.append(hodnota)

k = int(input())
print(f"{k=}")

assert(0 <= k < len(seznam))

low = 0
high = len(seznam)
while high - low > 1:
    pivot = seznam[randint(low, high - 1)]
    print(f"{low=} {high=} {pivot=} {seznam=}")
    low_numbers = [x for x in seznam[low:high] if x <= pivot]
    high_numbers = [x for x in seznam[low:high] if x > pivot]
    seznam = seznam[:low] + low_numbers + high_numbers + seznam[high:]
    mid = low + len(low_numbers)
    if k - 1 < mid:
```

```
        high = mid
    else:
        low = mid


print(seznam[low])
--------- Výstup ------------
k=7
low=0 high=10 pivot=27 seznam=[98, 84, 47, 27, 46, 48, 21, 65, 32, 59]
low=2 high=10 pivot=59 seznam=[27, 21, 98, 84, 47, 46, 48, 65, 32, 59]
low=2 high=7 pivot=59 seznam=[27, 21, 47, 46, 48, 32, 59, 98, 84, 65]
low=2 high=7 pivot=48 seznam=[27, 21, 47, 46, 48, 32, 59, 98, 84, 65]
59

```

Toto nám moc nechodilo, takže se ještě jednou k tomuto kódu vrátíme. Máte ho v repozitáři, `code/Ex5` nebo `Ex6`. Problém byl v ukončení.

```

```

Proč má toto náročnost O(n)?

$$T(n) \approx n + \frac{n}{2} + \frac{n}{4} + \cdots = 2n$$

Při náhodném výběru pivotu má algoritmus tuto náročnost pouze v průměru. Pro deterministický algoritmus potřebujeme nějak inteligentněji zvolit pivot: medián mediánů.

Binární vyhledáváni a třídění

Vyhledávání v setříděném seznamu

To je to, co potřebují dělat funkce `index` a `count` - najít hodnotu v setříděném seznamu, nebo zjistit, jestli se tam nachází, nebo v kolikrát.

Algoritmus: Půlení intervalu (proto *binární*).

Náročnost: log(n)

```
#!/usr/bin/env python3
# Binární vyhledávání v setříděném seznamu

kde = [11, 22, 33, 44, 55, 66, 77, 88]
co = int(input())

# Hledané číslo se nachazí v intervalu [l, p]
l = 0
p = len(kde) - 1

while l <= p:
    stred = (l+p) // 2
    if kde[stred] == co:   # Našli jsme
        print("Hodnota ", co, " nalezena na pozici", stred)
        break

```

```
    elif kde[stred] < co:
        l = stred + 1      # Jdeme doprava
    else:
        p = stred - 1      # Jdeme doleva
else:
    print("Hledaná hodnota nenalezena.")
```

Aplikace: Řešení algebraických rovnic, minimalizace

Celočíselná druhá odmocnina

```
# Emulate math.isqrt

n = int(input())

l = 0
p = n # Velkorysé počáteční meze

while l < p:
    m = int(0.5 * (l+p))
    # print(l, m, p)
    if m*m == n: # konec
        print(f"{n} is a perfect square of {m}") # format string
        break
    elif m*m < n:
        l = m
    else:
        p = m
    if p-l <= 1:
        print(f"{n} is not pefect square, isqrt is {l}")
        break
```

Úloha: Odmocnina reálného čísla

Řešení rovnice cos(x) = x

```
# solve x = cos(x) by bisection
from math import pi, cos

l = 0.0
p = pi/2.0

while p - l > 1.0e-6: # Tolerance
    m = 0.5*(l + p)
    print(l, m, p)
    if m - cos(m) == 0:
        print(f"Found exact solution x = {m}")
        break
    elif m - cos(m) < 0:
        l = m
    else:
        p = m
else:
    e = 0.5 * (p-l)
    print(f"Converged to solution x = {m}+/-{e}")
```

"Bisection" je bezpečná, ale nikoli rychlá metoda hledání kořenů rovnice a minimalizace.

Třídění

Opakovaný výběr minima

Opakovaně vybíráme minimum a příslušnou hodnotu umísťujeme na začátek seznamu.

```
# Třídění opakovaným výběrem minima

x = [31, 41, 59, 26, 53, 58, 97]

n = len(x)
for i in range(n):
    pmin = i
    for j in range(i+1, n):
        if x[j] < x[pmin]:
            pmin = j
    x[i], x[pmin] = x[pmin], x[i]

print(x)

```

Bublinové třídění

Postupně "probubláváme" hodnoty směrem nahoru opakovaným srovnávaním se sousedy

```
# Třídění probubláváním

x = [31, 41, 59, 26, 53, 58, 97]

n = len(x)
for i in range(n-1):
    nswaps = 0
    for j in range(n-i-1):
        if x[j] > x[j+1]:
            x[j], x[j+1] = x[j+1], x[j]
        nswaps += 1
    if nswaps == 0:
        break

print(x)

```

Funkce

Pokud chceme izolovat určitou část kódu, například proto, že dělá dobře definovaný generalizovatelný úkol anebo úkol často používaný, používáme funkce. Funkce je jeden ze základních nástrojů pro organizaci a vytváření opakovaně použitelného kódu (Dalším jsou třídy).

```
def hafni/():
    print("Haf!")

hafni()
hafni()
```

Funkce má jméno, pro které platí běžná pravidla pro vytváření identifikátorů. Kde to je vhodné, doporučuji používat rozkazovací způsob.

```
def hafni(n):
    for i in range(n):
        print("haf!")
```

`n` je tady parametr funkce. Do hodnoty `n` se při spuštění funkce překopíruje hodnota z volání funkce a platí tady všechny varování ohledně kopírování - o tom budeme vícekrát mluvit později.

Máme Python 3.9, takže modernější verze funkce bude vypadat takto:

```
def hafni(n:int): # Uvádíme očekávaný typ parametru
    for _ in range(n): # Používáme nepojmenovanou proměnnou
        print("haf!")
```

Uvedením typu parametru zabezpečíme, že interpret nás bude varovat, pokud použijeme parametr nesprávného typu. To někdy pomáhá, a jindy nám to zabraňuje psát generický kód.

Návratová hodnota a příkaz return

```
def plus(x,y):
    return x+y

print(plus(1,2))
print(plus("Ne","hafnu!"))
```

Příkaz `return výraz` ukončí vykonávání funkce a vrací jako hodnotu funkce `výraz`.

Nepovinné parametry

```
def hafni(krat:int = 1, zvuk:str = "Haf"):
    for _ in range(krat):
        print(zvuk)

hafni()
hafni(5)
hafni(zvuk = "Miau!")
hafni(krat = 5, zvuk = "Kokrh!")
```

Viditelnost proměnných: lokální a globální jmenný prostor

```
zvuk = "Kuku!"
kolik hodin = 0

def zakukej():
    print(zvuk)
    kolik hodin += 1
```

Proměnné `kolik.hodin` přiřazujeme, a Python ji musí uvnitř funkce zřídit. Implicitní předpoklad je, že chcete zřídit novou proměnnou. Pokud chcete použít proměnnou z globálního prostoru jmen, musíte to Pythonu říci.

```
zvuk = "Kuku!"
kolik hodin = 0

def zakukej():
    global kolik.hodin
    print(zvuk)
    kolik hodin += 1
```

Mimochodem, tato funkce dělá něco, čemu se typicky chceme vyhnout: ovlivňuje proměnnou, která není jejím parametrem. Toto nazývá vedlejší efekt a je to nejčastěji symptom špatného programování.

Správná funkce by měla být čistá, tedy by měla vypočíst a odevzdat svou návratovou hodnotu bez toho, aby měnila hodnoty nějakých proměnných, včetně svých parametrů.

Příklady funkcí, které určitě nejsou čisté, jsme viděli: jsou to metody seznamu, které nějak přetvářejí seznam na místě: `sort`, `reverse`. Tyto funkce mění seznam, který je volá a nevracejí hodnotu. Je to proto, že jde spíše o metody třídy `list`, tedy funkce, které patří do nějaké vyšší datové struktury a operují nad ni.