Programování 2

3. cvičení, 3-3-2022

tags: `Programovani 2`, `čtvrtek 1`, `čtvrtek 2`

Farní oznamy

1. Tento text a kódy ke cvičení najdete v repozitáří cvičení na https://github.com/PKvasnick/Programovani-2.
2. **ReCodEx**: Jsou všichni zaregistrováni?
3. **Domácí úkoly**: Zadáno včera,
    - 2 povinné a 1 volitelná úloha - jenom kvůli počítání bodového limitu, můžete si vybrat libovolné 2 nebo všechny 3.
    - 2 termíny, 10. 3. pro 10 bodů, 17. 3. pro 5

Dnešní program:

- Kvíz
- Domácí úkoly
- Opakování
- Načítání a zpracování posloupností , matice

Na zahřátí

> *"First, solve the problem. Then, write the code." – John Johnson*

Neboli **mějte po ruce tužku a papír**.

Co dělá tento kód

```
1 dict(zip(range(1,4), ["Pascal", "Python", "C++", "Javascript"]))
```

Pojmenované n-tice

```
>>> from collections import namedtuple

>>> # Create a namedtuple type, Point
>>> Point = namedtuple("Point", "x y")
>>> issubclass(Point, tuple)
True

>>> # Instantiate the new type
>>> point = Point(2, 4)
>>> point
Point(x=2, y=4)
```

Toto je jedna z možností, jak implementovat kontejner s různorodými daty, která spolu souvisí. Další možnosti?

Domácí úkoly

Fibonacciho algoritmus pro rozklad zlomků

Mějme dvě celá čísla x, y, a rozložme zlomek $\frac{x}{y}$ na součet zlomků s čitatelem 1,

$$\frac{x}{y} = \frac{1}{c_1} + \frac{1}{c_2} + \cdots + \frac{1}{c_n}$$

Na rozklad použijeme Fibonacciho algoritmus ze 13. století, založený na rovnosti

$$\frac{x}{y} = \frac{1}{\lceil \frac{y}{x} \rceil} + \frac{-y \mod (x)}{y \lceil \frac{y}{x} \rceil}$$

kde $\lceil x \rceil$ je funkce "strop", tedy nejmenší celé číslo větší nebo rovné x; v Pythonu tuto funkci najdete jako `math.ceil()`. Druhou část vztahu budeme rozkládat podle stejného vztahu, dokud nedostaneme čitatel 1 i u druhého zlomku.

Nechť je například $x = 17, y = 36$. Pak

$$\frac{17}{36} = \frac{1}{\lceil \frac{36}{17} \rceil} + \frac{-36 \mod (17)}{36 \lceil \frac{36}{17} \rceil} = \frac{1}{3} + \frac{15}{36 \cdot 3} = \frac{1}{3} + \frac{5}{36}$$

A když použijeme stejný postup na zlomek vpravo:

$$\frac{17}{36} = \frac{1}{3} + \frac{1}{\lceil \frac{36}{5} \rceil} + \frac{-36 \mod (5)}{36 \lceil \frac{36}{5} \rceil} = \frac{1}{3} + \frac{1}{8} + \frac{4}{36 \cdot 8} = \frac{1}{3} + \frac{1}{8} + \frac{1}{72}$$

Implementujte tento algoritmus v Pythonu. Na vstupu dostanete dvě celá čísla, oddělené mezerou, například `17 36`. Na výstup napíšete seznam jmenovatelů rozkladu, oddělených mezerou a seznam ukončíte znakem nového řádku, tedy například `3 8 72\n`.

Příklad 1

Vstup:

1 2

Výstup:

2

Příklad 2

Vstup:

2 3

Výstup:

2 6

Příklad 3

Vstup:

21 32

Výstup:

2 7 75 16800

Poznámky

1. Testy k této úloze jsou všechny příklady, uvedené v zadání úlohy.
2. Svůj kód lehko otestujete převodem zlomků na desetinná čísla a porovnáním původního zlomku s jeho rozkladem.

Dělící bod posloupnosti

Pro danou posloupnost reálných čísel $a_0, a_1, \ldots a_{n-1}$ najděte index j takový, že pro všechny indexy i $a_i < a_j$ pokud $i < j$, a $a_i > a_j$, pokud $i > j$. Jestliže takový index neexistuje, vrátíte -1. Pokud má posloupnost více dělících bodů, vraťte první zleva (nejmenší i. Jako řešení budou akceptovány pouze algoritmy s lineární náročností.

Vstupní posloupnost načtete z konzoly číslo po čísel, každé číslo na novém řádku. Posloupnost je ukončená řádkou s -1, která nepatří do posloupnosti. Výsledek zapíšete na standardní výstup jako celé číslo následované (jedním) znakem nového řádku.

Příklad 1

Vstup:

-1 (prázdná posloupnost)

Výstup:

-1 (nenalezeno)

Příklad 2

Vstup:

1
-1

Výstup:

0

Příklad 3

Vstup:

1
1
1
2
3
3
3
-1

Výstup:

3

Příklad 4

Vstup:

3
3
2
1
1
-1

Výstup:

-1

Testy

Pro tuto úlohu je 7 testů: posloupnosti délky 0 a 1, malá posloupnost s dělícím bodem, posloupnosti s dělícím bodem na levém a pravém konci, posloupnost bez dělícího bodu, a *dlouhá* posloupnost s dělícím bodem.

Poznámky

Abyste ušetřili porovnání, uvědomte si, že posloupnost s dělícím bodem musí být částečně setříděná.

Součet s nejmenšími společnými násobky

Pro celé číslo n označíme

$$S_n = \sum_{i=1}^{n} \frac{1}{nsn(1 \dots n)}$$

kde $nsn(a, b, c \dots)$ označuje nejmenší společný násobek čísel $a, b, c, \dots$, tedy nejmenší celé číslo, kterého děliteli jsou všechna čísla $a, b, c, \dots$.

Načtěte n ze standardního vstupu. Na standardní výstup napište dvě čísla, oddělená mezerou,

1. $nsn(1, 2, \dots n)$ jako celé číslo
2. S_n jako desetinné číslo s přesností na 5 desetinných míst.

Příklad 1

Vstup:

1

Výstup:

1 1.00000

Příklad 2

Vstup:

5

Výstup:

60 1.76667

Příklad 3

Vstup:

10

Výstup:

2520 1.78770

Příklad 4

Vstup:

20

Výstup:

232792560 1.78778

Poznámka

Testy pro tuto úlohu tvoří všechny příklady zde uvedené.

Opakování a pokračování:

Maximum a jiné vlastnosti posloupností

Tady si procvičíme úplně jednoduché věci, zčásti také proto, abychom si zopakovali některé postupy, které využijete pro domácí úkoly.

Načtení posloupnosti z konzole a ze souboru

Typické zadání úlohy v ReCodExu:

> Načtěte ze standardního vstupu posloupnost desetinných čísel oddělených znakem nového řádku, ukončenou řádkou s číslem -1 (toto číslo do posloupnosti nepatří)

Načítání pomocí generátoru

Pokud chceme hledat například maximum posloupnosti a nechceme ji celou načítat, musíme kód pro načítání a hledání promíchat. To je nešťastné, pokud chceme pro zpracování posloupnosti použít stejný kód pro načítání ze standardního vstupu nebo souboru.

```
def read_from_console():
    while "-1" not in (line := input()):
        yield float(line)
    return

m = float("-inf")
for number in read_from_console():
    if number > m:
        m = number
print(m)
```

Napište kód, který takto nalezne maximum při načítání posloupnosti ze souboru.

Reduce

Uměli bychom uzavřít otevřený cyklus `while`, resp. `for`, který máme v tomto kódu?

Můžeme použít funkci `functools.reduce`, která dělá přibližně toto:

```
def reduce(function, iterable, initializer=None):
    it = iter(iterable)
    if initializer is None:
        value = next(it)
    else:
        value = initializer
    for element in it:
        value = function(value, element)
    return value
```

Tedy funkce `reduce` propaguje a aktualizuje nějaký stav přes posloupnost.

```
from functools import reduce

def read_from_console():
    while "-1" not in (line := input()):
        yield float(line)
    return

maximum = reduce(max, read_from_console, float("-inf"))
print(maximum)
```

Takovýto kód bude rychlý, protože cyklus se vykonává uvnitř funkce, a tedy běží v C a ne v Pythonu.

Podobné úlohy

- rozhodnout, zda je posloupnost čísel monotonní a jak (konstantní, rostoucí, neklesající, klesající, nerostoucí)
- v posloupnosti čísel nalézt druhou největší hodnotu a počet jejích výskytů
- v posloupnosti čísel určit délku nejdelšího souvislého rostoucího úseku
- v posloupnosti čísel určit počet různých hodnot
- v posloupnosti čísel nalézt souvislý úsek se součtem K (pro zadanou hodnotu K)
- v posloupnosti kladných čísel nalézt souvislý úsek se součtem K (pro zadanou hodnotu K)
- v posloupnosti čísel nalézt souvislý úsek s maximálním součtem.

Řešeni

- Pro část úloh stačí implementovat funkci pro `reduce`.
- Pro další musíme vytvořite složitější vyhledávání.

Pomůcka

Python má generátor `pairwise` v modulu `itertools` (Python 3.10) nebo v modulu `more_itertools`. Pokud nechceme instalovat `more_itertools`, vytvoříme si jej sami. Generátor ze seznamu `1, 2, 3` generuje seznam dvojic `(1,2), (2,3)`.

```
from itertools import tee

def pairwise(lst):
    a, b = tee(lst)
    next(b, None)
    return zip(a, b)

```

2D úlohy

Máme matici $r \times s$, třeba s celými čísly, implementovaný jako seznam seznamô.

Chceme

- v matici nalézt maximální podmatici tvořenou kladnými čísly (podmaticí rozumíme souvislý obdélníkový výřez, maximální podmatice znamená podmatice tvořená co nejvíce prvky)
- v matici nalézt podmatici s maximálním součtem prvků.

Pomůcka

Odkud vezmeme posloupnost?

Můžeme si třeba definovat náhodnou posloupnost:

```
from random import randint

low = 0
high = 10
n = 10

print([randint(low, high) for i in range(10)])
```

Vyhledávání v setříděném seznamu

To je to, co dělá funkce `list.index` - najít hodnotu v setříděném seznamu, nebo zjistit, jestli se tam nachází, nebo kolikrát.

Podobně ale můžeme prohledávat interval na reálné ose, pokud definujeme "rozlišovací schopnost", tedy nejmenší interval, který ještě chceme prohledávat unvitř.

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.")

```

- Co kdybychom chtěli nalézt všechny stejné hodnoty, nejen jednu?
- Řešení rovnic
