

Programování 1 pro matematiky

5. cvičení, 4-11-2021

tags: Programování 1 2021, čtvrtek

Obsah:

- 0. Farní oznamy
- 1. Domácí úkoly
- 2. Opakování
- 3. Pokračování: Třídění a binární vyhledávání

Farní oznamy

1. **Materiály k přednáškám** najdete v GitHub repozitáři <https://github.com/PKvasnick/Programovani-1>. Najdete tam také kód ke cvičením.
2. **Domácí úkoly** Dostali jste zatím 15 úkolů k prvním pěti cvičením. Dostali jste úkoly i minulý týden, i když cvičení nebylo.
 - Nominální počet bodů (bez "bonusových" úloh) - 100%: 95
 - Minimální počet bodů: 66
 - Těší mě, že několik z vás přešlo z kritického do bezpečného pásma a mnozí píšete hezký kód.
 - Přecházíme do pravidelnějšího rytmu - na poslední chvíli, protože průběžná kontrola odevzdaných příkladů mi začíná přerůstat přes hlavu

Kde se nacházíme V posledních cvičeních se mírně odchylujeme od oficiálního sylabu

- Některé věci jsme probrali dřív
- Chci co nejvíc času věnovat psaní kódu.

Domácí úkoly

Srozumitelnost kódu: Medián

Úloha o mediánu si vyžaduje setřídít seznam a vrátit buď prostřední hodnotu nebo průměr dvou prostředních hodnot, podle toho, je-li počet prvků v seznamu lichý anebo sudý:

$$\begin{aligned} Med(1, 2, \underline{3}, 4, 5) &= 3 \\ Med(1, \underline{2}, \underline{3}, 4) &= \frac{2 + 3}{2} = 2.5 \end{aligned}$$

Z některých odevzdaných řešení by ale člověk mohl nabýt dojmu, že vypočítat něco takového je docela náročný úkol:

```

1 ...
2 nums = sorted(nums)
3 if len(nums)%2==0:
4     print((nums[int((len(nums))/2)]+nums[int(((len(nums))/2)-1)))/2)
5 else:
6     print(nums[int(len(nums)/2)])

```

nebo dokonce

```

1 ...
2 list.sort()
3 if len(list)%2!=0:
4     print(list[int((len(list)-1)/2)])
5 else:
6     print(float((list[int(len(list)/2)]+list[int((len(list)-2)/2)])/2))

```

Takovýto kód je nepřehledný. Nešlo by to napsat nějak srozumitelněji, aby bylo jasné na první pohled, že to je správně?

```

1 (5) 0 1 2 3 4 -> 2
2 (7) 0 1 2 3 4 5 6 -> 3

```

tedy pro lichý počet prvků máme `len(list)//2`

```

1 (6) 0 1 2 3 4 5 -> 2,3
2 (8) 0 1 2 3 4 5 6 7 -> 3,4

```

a pro sudý počet máme `len(list) // 2 - 1` a `len(list) // 2`, takže kód může vypadat nějak takto:

```

1 midpoint = len(list) // 2
2 if len(list) % 2 == 1:
3     median = list[midpoint]
4 else:
5     median = 0.5*(list[midpoint-1] + list[midpoint])

```

Srozumitelnost kódu: čtverce v trojúhelníku

Nebojte se při programování vzít do ruky tužku a papír. Dobrý kus kódu si často vyžaduje kus plánování a někdy i kus matematiky.

```

1         1
2       2 3
3     4 5 6
4   7 8 9 10
5 11 12 13 14 15
6      ....

```

Toto samozřejmě jde vyplňovat, například nějak takto (snažíme se, aby jsme pokud možno měli hezké indexy)

```

1 # Trojúhelník
2

```

```

3  n = int(input())
4
5  triangle = []
6  triangle.append([]) # řádek 0, kde nic není
7
8  row = 1 # délka řádku "row" je row.
9  number = 1 # první číslo jde do řádku 1
10
11 for row in range(1,n+1): # 1..n
12     triangle.append([]) # zakládám nový řádek trojúhelníku
13     for _ in range(1,row+1): # 1..row
14         triangle[row].append(number)
15         number += 1
16
17 triangle.remove([]) # pro pořádek odstraním nultý řádek
18 print(triangle)
19
20 5:
21 [[1], [2, 3], [4, 5, 6], [7, 8, 9, 10], [11, 12, 13, 14, 15]]

```

Teď už není problém spočítat kvadráty čísel v některém řádku. Ale někteří z vás si také všimli čísla na pravém okraji trojúhelníku:

```

1      1 = 1x2/2
2      2   3 = 2x3/2
3      4   5   6 = 3x4/2
4      7   8   9  10 = 4x5/2
5     11  12  13  14  15 = 5x6/2
6      ....

```

To jsou trojúhelníková čísla

$$T_n = \frac{n(n+1)}{2}$$

a pak součet čísel v posledním řádku můžeme napsat jako:

$$\sum_{T_{n-1}+1}^{T_n} k^2 = \sum_{k=1}^{T_n} k^2 - \sum_{k=1}^{T_{n-1}} k^2$$

a protože víme, že

$$\sum_{k=1}^N k^2 = \frac{N(N+1)(2N+1)}{6}$$

umíme napsat řešení bez toho, abychom generovali trojúhelník. *Dokonce bez toho, abychom použili Python.*

Nechci po vás geniální řešení, ale důkladný kód - i když autorům důvtipných řešení občas přihodím bonusový bod.

Podobný příklad

Máme matici $n \times n$. Úloha je vyplnit ji spirálově směrem od levého horního rohu do středu čísly $1, 2, \dots, n^2$.

```
1 | 1  2  3  4
2 | 12 13 14 5
3 | 11 16 15 6
4 | 10 9  8  7
```

Plánujeme a kreslíme

Implementace: Matici umíme implementovat jako seznam seznamů. Například matici vyplněnou jedničkami můžeme implementovat takto:

```
1 | >>> n = 5
2 | >>> matrix = [[i+n*j for i in range(n)] for j in range(n)]
3 | >>> matrix
4 | [[0, 1, 2, 3, 4], [5, 6, 7, 8, 9], [10, 11, 12, 13, 14], [15, 16, 17, 18,
   | 19], [20, 21, 22, 23, 24]]
```

Maticí procházíme střídavě po řádcích, tedy po všech prvcích jediného podseznamu, a po sloupcích, tedy po stejném prvku všech podseznamů:

```
1 | >>> for col in range(n):
2 |     print(matrix[0][col], end = ' ')
3 |
4 | 0 1 2 3 4
5 | >>> for row in range(n):
6 |     print(matrix[row][n-1], end = ' ')
7 |
8 | 4 9 14 19 24
9 | >>> for col in range(n-1,-1,-1):
10 |    print(matrix[n-1][col], end = ' ')
11 |
12 | 24 23 22 21 20
13 | >>> for row in range(n-1,-1,-1):
14 |    print(matrix[row][0], end = ' ')
15 |
16 | 20 15 10 5 0
```

Poslední, co potřebujeme, je spirálovité procházení. Teď jsme procházeli řádky a sloupce 0 a $n-1$. Pro spirálovité vyplňování potřebujeme posouvat horní, pravý, spodní a levý okraj tak, že po přejetí ho vždy posuneme o 1. Zavedeme proměnné `top`, `right`, `bot`, `left`, a pak to už jen všechno sestavíme:

```
1 | # vyplňujeme matici nxn spirálovitě z levého horního rohu doprostředka čísly
2 | # 1,2,...n*n.
3 |
4 | n = int(input())
5 |
6 | matrix = [[1 for _ in range(n)] for _ in range(n)] # _ protože proměnné
7 | nepotřebujeme
8 | left, right, top, bot = 0, len(matrix[0])-1, 0, len(matrix)-1
```

```

9  i = 1 # číslo, které vyplňujeme
10
11 while left <= right and top <= bot: # pokračujeme, dokud je kam vyplňovat
12     # Horní řádek zleva doprava
13     for col in range(left, right+1):
14         matrix[top][col] = i # vyplníme a inkrementujeme
15         i += 1
16     top += 1 # číslování je shora zleva
17
18     for row in range(top, bot +1):
19         matrix[row][right] = i
20         i += 1
21     right -= 1
22
23     for col in range(right, left-1, -1):
24         matrix[bot][col] = i
25         i += 1
26     bot -= 1
27
28     for row in range(bot, top-1, -1):
29         matrix[row][left] = i
30         i += 1
31     left += 1
32
33 for row in range(n):
34     for col in range(n):
35         print(f'{matrix[row][col]:3}', end = ' ')
36     print('\n')

```

Opakování

Seznamy, řezy, metody

Index vrací položku, řez (slice) vrací seznam

```

1  P y t h o n
2  0 1 2 3 4 5 6
3  0 1 2 3 4 5

```

```

1  >>> s = [i for i in range(10)]
2  >>> s
3  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
4  >>> s[3]
5  3
6  >>> s[3:4]
7  [3]
8  >>> s[::3]
9  [0, 3, 6, 9]
10 >>> s[10::-1]
11 [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
12 >>> s[::-1]
13 [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

```

Metody seznamů

- `list.append(x)` přidává položku na konec seznamu.
- `list.extend(iterable)` rozšíří seznam připojením všech prvků `iterable` na konec seznamu.
- `list.insert(i, x)` vloží položku `x` na pozici `i`.
- `list.remove(x)` odstraní ze seznamu první položku s hodnotou `x`.
- `list.pop([i])` odstraní položku na zadané pozici v seznamu anebo poslední položku, pokud index není zadán.
- `list.clear()` odstraní všechny položky ze seznamu.
- `list.index(x[, začátek[, konec]])` vrátí index (počítaný od 0) v seznamu, kde se nachází první položka s hodnotou rovnou `x`.
- `list.count(x)` určí, kolikrát se `x` nachází v seznamu.
- `list.sort(, klíč=None, reverse=False)` utřídí položky seznamu *na místě*.
- `list.reverse()` na místě obrátí pořadí prvků v seznamu.
- `list.copy()` vrací plytkou kopii seznamu.

Logický operátor `in`

Zjišťuje, zda se v iterovatelném objektu nachází daná hodnota.

```
1 list = [10, 20, 30, 40, 50 ];
2
3 if 10 in list:
4     print("y se nachází v seznamu")
```

Pomůcka - opakování

Odkud vezmeme posloupnost?

```
1 from random import randint
2
3 print(*[randint(1, 10) for _ in range(10)], sep=" ")
```

- `*[a, b, c, d]` rozbalí seznam na `a, b, c, d`.
- `_` zastupuje proměnnou, kterou jinak nepoužíváme a nepotřebujeme definovat její jméno.
- separátor v příkazu `print: print(1, 2, 3, 4, sep="x")` vytiskne `1x2x3x4\n`. Pro připomenutí, už jsme měli parametr `end: print(1, 2, 3, 4, end = "x")` vytiskne `1 2 3 4x`. Jestli neuvedete `end`, použije se znak konce řádku `\n`, pokud neuvedete `sep`, použije se mezera .

Binární vyhledávání a třídění

S tímto sme začli na konci předchozího dvičení.

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: $\log(n)$

```

1  #!/usr/bin/env python3
2  # Binární vyhledávání v setříděném seznamu
3
4  kde = [11, 22, 33, 44, 55, 66, 77, 88]
5  co = int(input())
6
7  # Hledané číslo se nachází v intervalu [l, p]
8  l = 0
9  p = len(kde) - 1
10
11 while l <= p:
12     stred = (l+p) // 2
13     if kde[stred] == co:    # Našli jsme
14         print("Hodnota ", co, " nalezena na pozici", stred)
15         break
16     elif kde[stred] < co:
17         l = stred + 1      # Jdeme doprava
18     else:
19         p = stred - 1      # Jdeme doleva
20 else:
21     print("Hledaná hodnota nenalezena.")

```

Aplikace: Řešení algebraických rovnic, minimalizace

Celočíselná druhá odmocnina

```

1  # Emulate math.isqrt
2
3  n = int(input())
4
5  l = 0
6  p = n # velkorysé počáteční meze
7
8  while l < p:
9      m = int(0.5 * (l+p))
10     # print(l, m, p)
11     if m*m == n: # konec
12         print(f"{n} is a perfect square of {m}") # format string
13         break
14     elif m*m < n:
15         l = m
16     else:
17         p = m
18     if p-l <= 1:
19         print(f"{n} is not pefect square, isqrt is {l}")
20         break

```

Úloha: Odmocnina reálného čísla

Řešení rovnice $\cos(x) = x$

```

1  # solve x = cos(x) by bisection
2  from math import pi, cos
3
4  l = 0.0
5  p = pi/2.0
6

```

```

7 while p - l > 1.0e-6: # Tolerance
8     m = 0.5*(l + p)
9     print(l, m, p)
10    if m - cos(m) == 0:
11        print(f"Found exact solution x = {m}")
12        break
13    elif m - cos(m) < 0:
14        l = m
15    else:
16        p = m
17 else:
18     e = 0.5 * (p-l)
19     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.

```

1 # Třídění opakovaným výběrem minima
2
3 x = [31, 41, 59, 26, 53, 58, 97]
4
5 n = len(x)
6 for i in range(n):
7     pmin = i
8     for j in range(i+1, n):
9         if x[j] < x[pmin]:
10             pmin = j
11     x[i], x[pmin] = x[pmin], x[i]
12
13 print(x)
14

```

Bublinové třídění

Postupně "probubláváme" hodnoty směrem nahoru opakovaným srovnáváním se sousedy

```

1 # Třídění probubláváním
2
3 x = [31, 41, 59, 26, 53, 58, 97]
4
5 n = len(x)
6 for i in range(n-1):
7     nswaps = 0
8     for j in range(n-i-1):
9         if x[j] > x[j+1]:
10             x[j], x[j+1] = x[j+1], x[j]
11             nswaps += 1
12     if nswaps == 0:
13         break
14
15 print(x)

```


