resitev

January 28, 2024

0.1 Dan 1: Sensor Sweep

Če nalogo "oluščimo" zgodbice, je takšna: imamo seznam števil in zanima nas koliko je števil, ki jim sledi večje število. Števila so podana v datoteki, po eno število na vrstico. Preberimo jih v tabelo.

[1]:
```
import numpy as np

globine = np.array([int(x) for x in open("example.txt")])
```

`numpy` ima tudi nekaj funkcij za branje datotek. Ena od njih je `loadtxt`, ki zna prebrati podatke v točno takšni obliki (in še kakšni malo bolj zapleteni, tudi).

[2]:
```
globine = np.loadtxt("example.txt", dtype=int)
```

Tako ali drugače dobimo tabelo. Videti je tako.

[3]:
```
globine
```

[3]:
```
array([199, 200, 208, 210, 200, 207, 240, 269, 260, 263])
```

0.1.1 Prvi del

Če bi programirali brez `numpy`-ja, bi naredili tole:

[4]:
```
stej = 0
for x, y in zip(globine[1:], globine):
    if x > y:
        stej += 1
print(stej)
```

```
7
```

Ali pa kar (ker je `True` praktično enak `1` in `False` enak `0`):

[5]:
```
stej = 0
for x, y in zip(globine[1:], globine):
    stej += x > y
print(stej)
```

```
7
```

Ali pa celo

[6]:
```
sum(x > y for x, y in zip(globine[1:], globine))
```

[6]:
```
7
```

Praktično isto bomo naredili v `numpy`-ju. Bistvo `numpy`-ja pa je v tem, da se trudimo, da za nobeno ceno ne bi napisali zanke. Zanka se mora zgoditi v `numpy`-ju. Namesto seštevanja v Pythonovi zanki, moramo seštevati dve tabeli v numpyju. Namesto primerjanja v zanki, moramo primerjati v numpyju. V bistvu nas zanima tole:

[7]:
```
globine[1:] > globine
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/var/folders/f9/lgq5ysmn24j3lkss6zkgmtr00000gn/T/ipykernel_20647/372271466.py i
 ↪<module>
----> 1 globine[1:] > globine

ValueError: operands could not be broadcast together with shapes (9,) (10,)
```

Numpy se pritoži, da tabeli nimata enako elementov. Numpy ni zip, popariti mora vse elemente. Zipa ni motilo, da je drugi seznam za en element daljši (zadnjega elementa pač ne primerjamo z nobenih elementom za njim) in ga je preprosto ignoriral. Numpy tega ne more: tabeli, ki ju sešteva, odšteva, množi, primerja … morata biti enako dolgi. Zadnji element moramo odbiti sami.

[8]:
```
globine[1:] > globine[:-1]
```

[8]:
```
array([ True,  True,  True, False,  True,  True,  True, False,  True])
```

To, odbijanje zadnjega ali prvega elementa je nekaj, kar bomo morali stalno početi (in se bomo pri tem pogosto tudi ušteli za kak element ali dva).

Tako kot prej, v čistem Pythonu, tudi tu vsak `True` predstavlja par elementov, pri katerem je bil drugi večji od prvega. In tako kot prej jih moramo tudi zdaj sešteti, le da ne bomo poklicali `sum`, temveč `np.sum`. Sicer bi delal tudi `sum`, a `np.sum` je za numpyjeve tabele veliko hitrejši.

[9]:
```
np.sum(globine[1:] > globine[:-1])
```

[9]:
```
7
```

0.1.2 Drugi del

Drugi del pravi, da moramo namesto posamičnih globin opazovati vsote po treh zaporednih globin.

Naivna rešitev Naloge se najprej lotimo naivno, sestavimo seznam vsot trojk. Čisto tako, za vajo.

Sešteti moramo naslednje tri tabele.

[10]:
```
globine[:-2]
```

[10]:
```
array([199, 200, 208, 210, 200, 207, 240, 269])
```

[11]:
```
globine[1:-1]
```

[11]:
```
array([200, 208, 210, 200, 207, 240, 269, 260])
```

[12]:
```
globine[2:]
```

[12]:
```
array([208, 210, 200, 207, 240, 269, 260, 263])
```

Enkrat smo odbili zadnja dva elementa, enkrat prvega in zadnjega, enkrat prva dva.

Te tri torej seštejemo.

[13]:
```
vsote = globine[:-2] + globine[1:-1] + globine[2:]
```

[14]:
```
vsote
```

[14]:
```
array([607, 618, 618, 617, 647, 716, 769, 792])
```

Naprej pa gre tako kot prej.

[15]:
```
np.sum(vsote[1:] > vsote[:-1])
```

[15]:
```
5
```

Premišljena rešitev Kakšna je pravzaprav razlika med dvema zaporednima trojkama? Kdaj je vsota, recimo, petega, šestega in sedmega elementa večja od vsote četrtega, petega in šestega? Glede na to, da peti in šesti element nastopata v obeh trojkah, bo prva trojka večja od druge, kadar je sedmi element večji od četrtega. Drugi del naloge je torej v bistvu enak prvemu, le da namesto parov zaporednih elementov primerjamo pare, katerih indeksi se razlikujejo za 3.

[16]:
```
np.sum(globine[3:] > globine[:-3])
```

[16]:
```
5
```

Ta rešitev je seveda boljša od naivne, le o numpyju nas nauči manj. No, nekaj pa le: malo smo utrdili odbijanje elementov. Če odbijemo tri na začetku, moramo v drugem tri na koncu, da bosta spet enako velika.