resitev-2-del

January 28, 2024

0.1 Dan 3: Binary Diagnostic, drugi del

Drugi del zahteva, da najprej vzamemo vse podatke.

```
00100
11110
10110
10111
10101
01111
00111
11100
10000
11001
00010
01010
```

Nato jih prefiltriramo; obdržimo samo tiste, katerih prvi bit je enak večinskemu. Večinska vrednost je 1, torej bomo obdržali samo vrstice, pri katerih je prvi bit enak 1.

```
11110
10110
10111
10101
11100
10000
11001
```

Vajo ponovimo z drugim bitom. Tu so v večini enice, torej vzamemo vrstice, ki imajo na drugem mestu enico.

```
10110
10111
10101
10000
```

In tako naprej, dokler ne ostane ena sama vrstica.

Preberimo podatke.

[2]:
```
import numpy as np
```

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

Pri reševanju bomo morali znati prebrati posamični stolpec. Ničtega, recimo, preberemo tako:

[3]:
```
column = data[:, 0]

column
```

[3]:
```
array([0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0])
```

Poleg tega bomo morali znati izbrati vrstice, pri katerih ima bit določeno vrednost. Za vajo recimo, da nas zanimajo vrstice, pri kateri je v tem stolpcu ničla. Katere so, izvemo tako:

[4]:
```
column == 0
```

[4]:
```
array([ True, False, False, False, False,  True,  True, False, False,
       False,  True,  True])
```

Ta `column == 0` lahko uporabimo za indeksiranje. Takole izberemo vse vrstice, pri katerih je prvi bit enak 0:

[5]:
```
data[column == 0]
```

[5]:
```
array([[0, 0, 1, 0, 0],
       [0, 1, 1, 1, 1],
       [0, 0, 1, 1, 1],
       [0, 0, 0, 1, 0],
       [0, 1, 0, 1, 0]])
```

Zdaj poznamo vse delčke. Le še v program jih moramo zložiti.

[8]:
```
data = np.array([[int(x) for x in v.strip()] for v in open("example.txt")])

bit = 0
while len(data) > 1:
    column = data[:, bit]
    majority = np.sum(column) >= data.shape[0] / 2
    data = data[column == majority]
    bit += 1

data
```

[8]:
```
array([[1, 0, 1, 1, 1]])
```

Naprej pa tudi znamo.

[9]:
```
powers = 2 ** np.arange(data.shape[1])[::-1]

np.sum(powers * data)
```

[9]: 23

Če smo čisto natančni: `data` je še vedno dvodimenzionalna tabela, čeprav z eno samo vrstico. `powers * data` bo pomnožil *vsako* vrstico z s potencami dvojke. Vemo pa, da bo `np.sum`, če mu ne podamo osi, vrnil vsoto vseh elementov tabele, se pravi, prek vseh vrstic. Ker je vrstica ena sama, pa je to itak eno in isto.

Drugi del naloge zahteva, da na podoben način iščemo vrstice, ki vsebujejo manjšinski bit. Tega ne moramo najti kar kot komplement `data`, temveč moramo ponoviti celoten program tako, da `>=` zamenjamo z `<` ali pa se znajdemo kako drugače.