

Rešitev

Branje podatkov je trivialno.

```
import numpy as np
```

```
grid = np.array([[int(c) for c in v.strip()] for v in open("example.txt")])
h, w = grid.shape
```

Rešitev - no, korak rešitve - tokrat izdajmo v enem zamahu in tokrat bo v funkciji.

```
def step(grid):
    grid += 1
    flashed = new = grid > 9
    while np.any(new):
        for gdx0, gdx1, ndx0, ndx1 in ((0, -1, 1, w), (0, w, 0, w), (1, w, 0, -1)):
            for gdy0, gdy1, ndy0, ndy1 in ((0, -1, 1, h), (0, h, 0, h), (1, h, 0, -1)):
                grid[gdy0:gdy1, gdx0:gdx1] += new[ndy0:ndy1, ndx0:ndx1]
        new = (grid > 9) & ~flashed
        flashed |= new
    grid[flashed] = 0
    return np.sum(flashed)
```

`flashed` in `new` bosta bool-ovi matriki enake velikosti kot `grid`. `flashed` bo povedala, katere hobotnice so se zasvetile v tem koraku, `new` pa katere so se zasvetile znotraj koraka, zaradi sosedov.

V začetku povečamo vse vrednosti za 1 in si zapomnimo, kaj se je zastevalo, `grid > 9`.

Dokler se posveti kaj novega, `while np.any(new)`, k vsakemu polju `grid` prištejemo 1 za vse sosede, za katere `new` pravi, da so se pravkar zasvetili. To bi lahko napisali z zaporedjem osmih prištevanj v slogu

```
grid[0:-1, 0:-1] += new[1:h, 1:w]
grid[0:-1] += new[1:h]
grid[0:-1, 1:w] += new[1:h, 0:-1]
```

in tako naprej. Vendar se mi zdi to varneje zapisati z zanko. Stvar okusa. Glavno je, da povečamo vse elemente `grid`, ki jih je potrebno povečati. (Gornja koda poveča števec pri srednji hobotnici, kot da bi imeli `grid += new`. To ni čisto prav, vendar ne bo škodilo; berite naprej.)

Nato pride pomembna logika: na novo so se zasvetila tiste hobotnice, pri katerih je števec večji od 9 in se niso svetile že od prej,

```
new = (grid > 9) & ~flashed
```

Te tudi dodamo med svetleče se hobotnice,

```
flashed |= new
```

To torej ponavljamo, dokler se ne zgodi, da se nobena hobotnica ne zasveti več na novo. Nato nastavimo števec svetlečih se hobotnic na 0,

```
grid[flashed] = 0
```

(Zato tisti `grid += new` ne škodi: števec teh hobotnic bomo tako ali tako nastavili na 0.)

Funkcija na koncu vrne število vseh hobotnic, ki so se v tem koraku zasvetile.

Funkcijo uporabimo za oba dela naloge. Prvi zahteva, povemo, koliko hobotnic se zasveti v 100 korakih.

```
print(sum(step(grid) for _ in range(100)))
```

```
1656
```

V drugem delu nas zanima, po koliko korakih se prvič hkrati zasvetijo vse hobotnice. Za to naredimo zanko prek funkcije `itertools.count(n)`, ki šteje od `n` (privzeta vrednost je 0) do neskončno. Šteli bomo od 101, saj je prvi korak, ki ga bomo naredili, že sto prvi.

```
from itertools import count
```

```
for steps in count(101):  
    if step(grid) == grid.size:  
        break
```

```
print(steps)
```

```
195
```