

Day 11: Seating System

(Povezava na nalogo)

Kot skoraj vsako leto tudi letos ena naloga v slogu Conwayeve igre življenja.

Tokrat imamo stole (L) vmes pa so prazni prostori.

```
L.LL.LL.LL
LLLLLLL.LL
L.L.L..L..
LLLL.LL.LL
L.LL.LL.LL
L.LLLL.LL
..L.L....
LLLLLLLLL
L.LLLLLL.L
L.LLLL.LL
```

V prvem delu se ljudje posedajo po stolih tako, da

- prazen stol zasedejo, če med osmimi polji okrog njega ni nobenega zasedenega stola;
- poln stol zapustijo, če so zasedeni vsaj štirje stoli okrog njega.

V drugem delu se pravila spremenijo tako, da ne opazujejo osmih polj okrog sebe, temveč pogledajo v vseh osem smeri in preštejejo, v koliko smereh je prvi stol, ki ga vidijo, zaseden.

- prazen stol zasedejo, če noben od teh osmih stolov ni zaseden;
- poln stol zapustijo, če je zasedenih vsaj pet od teh stolov.

"Igra" teče tako, da v vsakem koraku naredimo "statistiko" za vse stole in jih izpraznimo ali napolnimo ali pustimo. Po določenem številu korakov pridemo v situacijo, ko se nič več ne spremeni. Naloga je ugotoviti, koliko stolov je takrat zasedenih.

Reševanje naloge bo imelo tri dele:

- branje podatkov,
- funkciji, ki za dani stol povesta število sosedov - v smislu prvega ali drugega dela naloge
- zanka, ki poganja simulacijo, dokler se stanje ne neha spreminjati.

Naloga ni posebej težka, zato se bomo ob njej predvsem učili lepo programirati. :)

Branje in hramba podatkov

Podatke bomo shranili v seznamu seznamov, poleg tega pa si bomo shranili še širino in višino polja, `w` in `h`.

```
seats = [list(x.strip()) for x in open("example.txt")]
w, h = len(seats[0]), len(seats)
```

```
seats
```

Štetje sosedov za prvi del

Da preverimo vseh osem sosedov polja (x, y) , bomo potrebovali dve zanki, takole

```
for x0 in (x - 1, x, x + 1)
for y0 in (y - 1, y, y + 1)
if (x0, y0) != (x, y) and 0 <= x0 < w and 0 <= y0 < h
```

S pogojem izločimo srednje polje, ki seveda na šteje, poleg tega pa se moramo znebiti koordinat izven pravokotnika.

Za vsako takšno koordinato moramo preveriti, ali velja `seats[y0][x0] == "#`. To je `True` ali `False`, se pravi 1 ali 0.

Pa smo končali.

```
def count_occupied(state, x, y):
    return sum(state[y0][x0] == "#"
               for x0 in (x - 1, x, x + 1)
               for y0 in (y - 1, y, y + 1)
               if (x0, y0) != (x, y) and 0 <= x0 < w and 0 <= y0 < h)
```

Štetje sosedov za drugi del

Za drugi del je potrebno napisati zanki, ki določita smeri. Potem se pomikamo v tej smeri, dokler ne pridemo do stola ali do roba.

Napisati takšno funkcijo ni posebna umetnost, umetnost pa jo je napisati tako, da je kratka in se koda na ponavlja. Kakorkoli obračam, se izkaže, da je še najbolj praktična različica `while True`, v zanki pa `break`, kadar pridemo do stola ali roba.

```
def count_occupied_line(state, x, y):
    occupied = 0
    for dx in (-1, 0, 1):
        for dy in (-1, 0, 1):
            if dx == dy == 0:
                continue
            x0, y0 = x, y
            while True:
                x0 += dx
                y0 += dy
                if 0 <= x0 < w and 0 <= y0 < h:
```

```

        if state[y0][x0] != ".":
            occupied += state[y0][x0] == "#"
            break
    else:
        break
return occupied

```

Simulacija

Najprej napišimo funkcijo, ki za vsak stol (ali prazno mesto) izračuna njegovo naslednje stanje. Funkcija kot argument dobi trenutno stanje, koordinate, funkcijo za izračun števila sosedov (`count_occupied` ali `count_occupied_line`) in število sosedov, ob katerem sedeči zapusti svoj stol.

Funkcija je precej očitna.

```

def next_state(state, x, y, occ_counter, occ_limit):
    curr = state[y][x]
    if curr != ".":
        f = occ_counter(state, x, y)
        if f == 0:
            return "#"
        if f >= occ_limit:
            return "L"
    return curr

```

Zdaj, ko je vse tako lepo pripravljeno, pa gremo z zanko čez oba dela naloge: enkrat štejemo z `count_occupied` in zapustimo sedež pri 4 sosedih, enkrat z `count_occupied_line` in 5 sosedi.

```

seats = [list(x.strip()) for x in open("input.txt")]
w, h = len(seats[0]), len(seats)

for func, limit in ((count_occupied, 4), (count_occupied_line, 5)):
    prev, state = None, seats
    while state != prev:
        prev = state
        state = [[next_state(state, x, y, func, limit)
                  for x in range(w)] for y in range(h)]

    print(sum(sum(c == "#" for c in v) for v in state))

```

2448

2234

Notranja zanka primerja trenutno in prejšnje stanje. Dokler nista enaka, si trenutno stanje zapomni kot prejšnje, novo pa sestavi tako, da izračuna naslednje stanje za vse koordinate stolov.

Program ni najhitrejši. Pythonove zanke pač niso najhitrejša stvar na svetu, kakih dobrih bližnjic, s katerimi bi se jih izognili, pa ni. Z `numpy`-jem se da še kar lepo in hitro poskrbeti za prvi del, z drugim pa se mi ni dalo hecati.