resitev

January 28, 2024

0.1 Day 10: Adapter Array

(Povezava na nalogo)

Spet ena lepa naloga, predvsem drugi del. Lepa je, ker ni potrebno programirati, temveč razmišljati. Programiranje je potem trivialno.

0.2 Prvi del

Spet bomo izpustili zgodbico: imamo seznam pozitivnih celih števil, ki jim dodamo še 0 in še število, ki je za 3 večje od največjega. Števila so sicer v naključnem vrstnem redu, a če bi opazovali, katera števila so v seznamu, katera ne, bi odkrili, da nikoli ne manjka le eno; luknje med njimi so vedno dolge dve števili. V bistvu nas zanima, koliko je zaporednih števil in koliko lukenj, v katerih manjkata dve števili. To dvoje moramo zmnožiti. (Ta opis ni preveč dober. Če se ne spomnite naloge in ga ne razumete, poglejte originalni opiz, ki pa je precej dolg.)

Očitno bo potrebno števila urediti, zato jih preberimo kar v `sorted`, dodamo pa še 0 in številko, ki je za 3 večja od največje; tako hoče naloga. Potem z `zip` sestavimo razlike, jih preštejemo in zmnožimo, kar moramo.

[1]:
```
from collections import Counter

s = sorted(int(x) for x in open("input.txt"))
s = [0] + s + [s[-1] + 3]

jumps = Counter((y - x for x, y in zip(s, s[1:])))
print(jumps[1] * jumps[3])
```

```
1904
```

0.3 Drugi del

Ta je pa zanimiv. Števila moramo zložiti v zaporedje od 0 do največjega, pri čemer smemo kakšno število tudi izpustiti, vendar razlika med zaporednima številoma nikoli ne sme biti večja od 3. Koliko takšnih zaporedij obstaja?

0.3.1 Rešitev naprej

Poglejmo zaporedje

```
0, 1, 2, 3, 4, 7, 8, 9, 10, 11, 14, 17, 18, 19, 20,
```

- Do 0 lahko pridemo na 1 način (0).
- Do 1 lahko pridemo na 1 način (0, 1).
- Do 2 lahko pridemo na 2 načina - iz 0 ali iz 1, torej (0, 2) ali (0, 1, 2).
- Do 3 lahko pridemo na 4 načine: iz 0, to je (0, 1); iz 1, to je (0, 1, 2) ali iz 2, pri čemer lahko do 2 pridemo na dva načina, torej (0, 2, 3) ali (0, 1, 2, 3).
- Do 4 lahko pridemo na 7 načinov: prek 1 torej (0, 1, 4); prek 2, do katere pa se da na dva načina, kar nam torej da (0, 2, 4) in (0, 1, 2, 4); ali prek 3, kar je šlo na 4 načine in nam da (0, 1, 4), (0, 1, 2, 4), (0, 2, 3, 4) in (0, 1, 2, 3, 4).
- Do 7 lahko pridemo na 7 načinov; najprej gremo do 4 in potem na 7. Vse poti so torej take kot prej, le še 7 dodamo na koncu.
- Do 8 lahko pridemo na 7 načinov; toliko pač, na kolikor načinov lahko pridemo do 7 (saj števili 5 in 6 ne obstajata)
- Do 9 pridemo na 14 načinov - na 7 načinov prek 7 in na 7 načinov prek 8.
- Do 10 pridemo na 28 načinov - na 7 načinov prek 7, na 7 načinov prek 8 in na 14 načinov prek 9.
- Do 11 pridemo na $7 + 14 + 28 = 59$ načinov.

Na kratko: **do 11 pridemo tako, da k vsem možnim potem do 8 do 9 in do 10 pripišemo 11. Poti do 11 je torej toliko, kot poti do 8, 9 in 10 skupaj.** Takšno velja za vsa števila, od 0 do konca. Razen, seveda, za tista, ki manjkajo.

In tako naprej.

Zdaj, ko smo to razmislili, je programiranje trivialno.

[2]:
```
from collections import defaultdict

ways = defaultdict(int)
ways[0] += 1
for jolt in s:
    for i in range(1, 4):
        ways[jolt + i] += ways[jolt]

print(ways[s[-1]])
```

```
10578455953408
```

0.4 Rešitev nazaj

Do števila `n` lahko pridemo na toliko načinov, na kolikor lahko pridemo do `n - 1`, `n - 2` in `n - 3`. Do števila 0 pa pridemo na 1 način. To nam da naslednjo preprosto funkcijo.

```
def ways(n):
    return n == 0 or sum(ways(x) for x in range(n - 3, n) if n in s)
```

ki pa je ne bomo pognali, ker bi tekla v nedogled, saj bi zelo pogosto spraševala za eno in isto število. Da se temu izognemo, bomo dodali dekorator `@lru_cache(3)`, s katerim bomo dosegli, da si bo funkcija zapomnila zadnja dva rezultata in jih ne računala vedno znova. (Seveda bi lahko vzeli več kot dva, vendar to že zadošča!)

Mimogrede ugotovimo, da ni nobene potrebe, da bi bil seznam `s` urejen. Iz estetskih razlogov pa

je lepo, če je množica, saj bo napogostejša operacija, ki jo bomo potrebovali, `in`. (V resnici pa je število števil tako majhno, da je čisto vseeno, ali so shranjena v množici ali seznamu.)

[3]:
```
from functools import lru_cache

s = {0} | {int(x) for x in open("input.txt")}

@lru_cache(2)
def ways(n):
    return n == 0 or sum(ways(x) for x in range(n - 3, n) if x in s)

ways(max(s))
```

[3]:
```
10578455953408
```