

Day 14: Docking Data

(Povezava na nalogo)

Naloga ni posebej težka: gre za žongliranje z biti. Z vidika Pythona se naučimo pretvarjanja iz dvojiškega zapisa ter operatorjev `&` in `|`, ki računata bitni *and* in *or*.

Prvi del

Podatki so videti tako (takšni blokci se potem nadaljujejo):

```
mask = XXXXXXXXXXXXXXXXXXXXXXXXXXXX1XXXX0X
mem[8] = 11
mem[7] = 101
mem[8] = 0
```

Imamo računalnik in z ukazi `mem[a] = b` na pomnilniško lokacijo `a` zapišemo podatek `b` (ki je podan desetiško, čeprav so v gornjem primeru slučajno same ničle in enice). Trik pa je v tem, da je potrebno upoštevati še bitno masko. Bite, ki so v maski označeni z `X` pustimo pri miru, bite, označene z `1`, postavimo na `1`, bite označene z `0`, postavimo na `0`.

Pri reševanju si bomo na debelo pomagali s tem, da lahko Pythonova funkcija `int` prejme dva argumenta: prvi je niz, ki ga je potrebno pretvoriti v število, drugi je številska osnova, ki bo v našem primeru `2`.

Iz maske bomo pripravili dva podatka: prvi bo povedal, katere bite je potrebno postaviti na `0`. Če imamo

```
mask = "XXXXXXXXXXXXXXXXXXXXXXXXX1XXXX0X"
```

bomo zamenjali vse `"X"` z ničlami, tako da bodo imele vrednost `1` samo še enice.

```
mask.replace("X", "0")
```

To potem pretvorimo v število

```
ones = int(mask.replace("X", "0"), 2)
```

Podobno, le ravno obratno, naredimo z ničlami: želimo, da so vsi biti, ki jih je potrebno postaviti na `0`, enaki `0`, vsi ostali naj bodo `1`.

```
zeros = int(mask.replace("X", "1"), 2)
```

Program je potem takšen:

```
mask = 0
mem = {}
for line in open("input.txt"):
    instr, data = line.split(" = ")
    if instr == "mask":
        zeros = int(data.replace("X", "1"), 2)
```

```

        ones = int(data.replace("X", "0"), 2)
    else:
        addr = int(instr[instr.index("[") + 1: instr.index(")"]])
        mem[addr] = int(data) & zeros | ones

print(sum(mem.values()))

```

Vrstico razdelimo glede na " = "; levo je ukaz, desno podatki. Če je ukaz `mask`, izračunamo novo masko. Če gre za nastavljanje, pa iz levega dela razberemo pomnilniški naslov. Podatke pretvorimo v `int`, nato pa z `& zeros` postavimo vse, kar mora biti 0, na 0, in z `| ones` vse, kar mora biti 1, na 1.

Naloga je vrniti vsoto vseh podatkov v pomnilniku, zato jo v zadnji vrstici izračunamo in izpišemo.

Drugi del

Drugi del pravi, da

- maska ne vpliva ne spreminja podatka temveč pomnilniški naslov
- pustimo pri miru bite, ki so označeni z 0, pač pa so lahko tisti, ki so označeni z X, karkoli. Torej, za X moramo vzeti vse možne kombinacije vrednosti. Če je v vrstici 5 X-ov, je možnih kombinacij 2^5 , torej bomo spreminjali 2^5 pomnilniških naslovov.

Tem bitom v nalogi rečejo, da so *floating*.

Reševanje zahteva nekoliko več telovadba z maskami.

```

mem = {}
for line in open("input.txt"):
    instr, data = line.split("=")
    if instr.strip() == "mask":
        data = data.strip()
        ones = int(data.replace("X", "0"), 2)

        xpos = [2 ** i for i, c in enumerate(reversed(data)) if c == "X"]
        floating = [
            sum(o for c, o in zip(f"{i:0>{len(xpos)}b}", xpos) if c == "1")
            for i in range(2 ** len(xpos))]

        zeros = 2 ** 36 - 1 - sum(xpos)
    else:
        data = int(data)

        addr = int(instr[instr.index("[") + 1: instr.index(")"]])
        addr = (addr & zeros) | ones
        for comb in floating:

```

```

mem[addr | comb] = data

print(sum(mem.values()))
2667858637669

```

Z enicami je tako kot prej.

Z `xpos` naračunamo števila, se pravi potence 2, ki ustrezajo posameznim bitom označeni z `X`. Podatke obrnemo, da bo najnižji bit na vrsti prvi, nato jih oštevilčimo in, če je vrednost bita enaka `X`, izračunamo ustrezno potenco.

Nato v `floating` naračunamo vse kombinacije teh bitov. Trik, ki ga uporabimo, razložimo posebej. Imejmo seznam imen.

```

imena = ["Ana", "Berta", "Cilka", "Dani"]

```

Izpisati želimo vse kombinacije teh imen (vključno s praznim seznamom in seznamom, ki vsebuje vsa imena. Pomagali si bomo s temi nizi: izpišimo vsa štiribitna števila po dvojiško.

```

for i in range(2** len(imena)):
    print(f"{i:0>4b}")

```

```

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

```

To skombiniramo s seznamom imen: ime vzamemo, če je ustrezni bit enak 1.

```

for i in range(2** len(imena)):
    print([ime for ime, bit in zip(imena, f"{i:0>4b}") if bit == "1"])

```

```

[]
['Dani']
['Cilka']
['Cilka', 'Dani']
['Berta']

```

```

['Berta', 'Dani']
['Berta', 'Cilka']
['Berta', 'Cilka', 'Dani']
['Ana']
['Ana', 'Dani']
['Ana', 'Cilka']
['Ana', 'Cilka', 'Dani']
['Ana', 'Berta']
['Ana', 'Berta', 'Dani']
['Ana', 'Berta', 'Cilka']
['Ana', 'Berta', 'Cilka', 'Dani']

```

Zgoraj nimamo ime, temveč števila. In zanimajo nas njihove vsote. Na primer

```

xpos = [1024, 32, 8, 1]
for i in range(2 ** len(xpos)):
    print(sum(val for val, bit in zip(xpos, f"{i:0>4b}") if bit == "1"))
0
1
8
9
32
33
40
41
1024
1025
1032
1033
1056
1057
1064
1065

```

Skoraj točno tako naračunamo vsote v gornjem seznamu **floating**. Dodati je potrebno le še to, da ne potrebujemo nujno štirih bitov, temveč toliko bitov, kolikor je dolg **xpos**. Zato namesto **f"{i:0>4b}"** pišemo **f"{i:0>{len(xpos)}b}"**. Ja, števila je možno vstaviti tudi znotraj niza za formatiranje. Gnezdeni zaviti oklepaji.

Poleg tega pripravimo **zeros**, ki bo imel ničle tam, kjer so "floating biti".

Ideja je torej, da v začetku vstavimo vse te bite na 0, potem jim prištevamo različne kombinacije, ki smo si jih naračunali v **floating**.

Tako torej dobimo masko -- **ones**, **zeros** in **floating**.

V drugem delu, po **else**, naračunamo naslov, postavimo enice, kakor je treba, vse **floating** bite pa na 0. Nato gremo čez vse kombinacije floating bitov,

izračunamo naslov in nastavimo.