resitev

January 28, 2024

1 Day 16: Ticket Translation

(Povezava na nalogo)

Naloga med študenti ni požela prevelikega navdušenja, zdela se jim je eno samo pisanje, eno samo dolgočasno preobračanje podatkov.

Zanje imam slabo novico: dobrodošli v resničnem življenju.

Naloga bo vseeno zanimiva: učili se bomo lepo po kosih vrteti te podatke. In še nevemkaterič videli, zakaj so izpeljani seznami in množice tako praktični.

Imamo datoteko takšne oblike:

```
class: 1-3 or 5-7
row: 6-11 or 33-44
seat: 13-40 or 45-50

your ticket:
7,1,14

nearby tickets:
7,3,47
40,4,50
55,2,20
38,6,12
```

Prvi blok opisuje podatke in možne interval vrednosti. (Vedno sta dva, vendar to v navodilih ni posebej omenjeno in se na to ne bomo zanašali, saj je splošna rešitev preprostejša.)

Tretji del vsebuje vrstice. Vsaka vrstica ima vse te podatke, vendar ne vemo, v kakšen vrstnem redu. Vemo pa, da isti stolpec vedno ustreza istemu podatku.

Vmesni del vsebuje naše podatke.

Prvi del naloge je preprost: nekatera izmed števil v tretjem bloku se ne pojavijo v prav nobenem intervalu. V gornjem primeru so to 4, 55 in 12. Zanima nas njihova vsota.

Drugi del je zabavnejši. Za vsak stolpec je potrebno ugotoviti, kateri del predstavlja. Za to najprej izločimo vse vrstice, ki vsebujejo kakšno številko, ki se ne pojavi v nobenem intervalu. Nato pa opazujemo, kateri stolpec bi, glede na podane številke, lahko predstavljal kateri podatek. Nekateri lahko ustrezajo več podatkom, a izkazalo se bo, da je rešitev, v kateri vsakemu stolpcu ustreza natančno eno pravilo, enolična.

1.1 Branje podatkov

Preberemo datoteko in jo s `split("\n\n")` razkosamo v tri bloke.

[1]:
```
rules_part, my_ticket, tickets = open("input.txt").read().split("\n\n")
```

1.1.1 Pravila

Prvi del, pravila bomo razstavili v slovar. Ne, ker bi v resnici potrebovali slovar, temveč ker bomo včasih potrebovali samo intervale, včasih pa tudi ime polja. Ključ bo polje, vrednosti pa bodo kar množica, ki vsebuje vsa dovoljena števila za to polje.

S `splitlines()` razstavimo blok na vrstice. Vsako vrstico s `split(":")` razbijemo na ime polja in niz s seznam intervalov, `intervals`. (Kaj se zgodi na koncu, pa povem spodaj.)

(Še tolažba: te vrstice so najbolj zapleteni del programa. :)

[2]:
```
rules = {}
for rule in rules_part.splitlines():
    field, intervals = rule.split(":")
    rules[field] = set.union(*(set(range(int(f), int(t) + 1))
                               for f, t in (p.split("-") for p in intervals.
 ↪split(" or "))))
```

Zdaj recimo, da imamo

[3]:
```
intervals = "1-3 or 5-7"
```

Razbijemo jih glede na `" or "`.

[4]:
```
intervals.split(" or ")
```

[4]:
```
['1-3', '5-7']
```

Vsakega razbijemo glede na `"-"`.

[5]:
```
[p.split("-") for p in intervals.split(" or ")]
```

[5]:
```
[['1', '3'], ['5', '7']]
```

Zdaj gremo z zanko čez to in sestavimo intervale.

[6]:
```
[range(int(f), int(t) + 1)
 for f, t in (p.split("-") for p in intervals.split(" or "))]
```

[6]:
```
[range(1, 4), range(5, 8)]
```

Pravzaprav nas ne zanimajo `range`-i, temveč množice.

[7]:
```
[set(range(int(f), int(t) + 1))
 for f, t in (p.split("-") for p in intervals.split(" or "))]
```

[7]:
```
[{1, 2, 3}, {5, 6, 7}]
```

Vse množice zložimo skupaj.

[8]:
```
set.union(*(set(range(int(f), int(t) + 1))
            for f, t in (p.split("-") for p in intervals.split(" or "))))
```

[8]:
```
{1, 2, 3, 5, 6, 7}
```

No, to smo spravili v slovar.

1.1.2 Vozovnica in ostale vozovnice

To je trivialno.

[9]:
```
my_ticket = [int(x) for x in my_ticket.splitlines()[1].split(",")]

tickets = [[int(x) for x in v.split(",")]
           for v in tickets.splitlines()[1:]]
```

1.2 Prvi del: vsota neveljavnih števil

Tudi to je trivialno. Sestavimo množico vseh dovoljenih števil. Gremo čez vse vozovnice; za vsako izračunamo vsoto neveljavnih števil v njej in potem poračunamo vsoto te vsote.

[10]:
```
all_valid = set.union(*rules.values())
print(sum(sum(x
              for x in ticket
              if x not in all_valid)
          for ticket in tickets))
```

```
19060
```

1.3 Drugi del

Ta je pa bolj zanimiv. Najprej sestavimo seznam, ki vsebuje le veljavne vozovnice, torej vozovnice, v kateri se ne pojavi nobena številka, ki je ni v nobenem intervalu. Ali so vse številke veljavne, preverimo tako, da ugotovimo, ali je množica teh številk podmnožica množice vseh veljavnih številk.

[11]:
```
valid_tickets = [ticket for ticket in tickets if set(ticket) <= all_valid]
```

Od tod naprej pa bomo preverjali, kateri stolpec bi lahko, glede na števila, ki jih vsebuje, ustrezal kateremu pravilu. Torej bomo delali s stolpci. Sestavimo seznam množic števil v vsakem stolpcu. Stolpcev je pa toliko kot pravil.

[12]:
```
columns = ({ticket[i] for ticket in valid_tickets}
           for i in range(len(rules)))
```

Stolpce bomo potrebovali samo enkrat, torej je vseeno, če jih ne tlačimo v seznam, temveč naredimo kar generator. Kot zanimivost povejmo še, da bi jih lahko naredili tudi z `columns = map(set, zip(*valid_tickets))` in v tem primeru bi lahko bili tudi `valid_tickets` kar generator.

[13]:
```
valid_tickets = (ticket for ticket in tickets if set(ticket) <= all_valid)
columns = map(set, zip(*valid_tickets))
```

Zdaj pa sestavimo seznam, ki bo vseboval indeks stolpca in imena vseh pravil, ki bi jim stolpec lahko ustrezal. Stolpce torej oštevilčimo (`enumerate(columns)`) in za vsak par `i, column`, poiščemo vsa polja, za katera velja, da so števila v tem stolpcu podmnožica veljavnih števil tega polja.

[14]:
```
maybes = [(i, {field for field, valids in rules.items() if column <= valids})
          for i, column in enumerate(columns)]
```

Izvemo, da bi 15. stolpec lahko ustrezal trem različnim poljem.

[15]:
```
maybes[15]
```

[15]:
```
(15, {'arrival platform', 'class', 'row'})
```

Nekateri ustrezajo tudi več poljem, nekateri pa manj. Preštejmo, koliko poljem ustreza kateri.

[16]:
```
[len(maybe[1]) for maybe in maybes]
```

[16]:
```
[16, 6, 18, 15, 19, 9, 4, 1, 20, 17, 2, 7, 5, 11, 10, 3, 14, 8, 13, 12]
```

Če to reč uredimo, izvemo nekaj zanimivega.

[17]:
```
sorted(len(maybe[1]) for maybe in maybes)
```

[17]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

In zdaj pošpekuliratmo. Mogoče je stvar preprosta. Uredimo seznam `maybe` po številu polj. Za prvi stolpec vemo, kateremu pravilu ustreza. Temu pravilu ne more ustrezati noben drug stolpec. In če imamo srečo (izkazalo se bo, da jo res imamo) prav temu pravilu ustrezajo tudi vsi drugi stolpci. Potemtakem za drugi stolpec ostane le še eno možno pravilo. In tako naprej.

[18]:
```
maybes.sort(key=lambda x: len(x[1]), reverse=True)
assignments = {}
while maybes:
    i, fields = maybes.pop()
    field = (fields - set(assignments)).pop()
    assignments[field] = i
```

Seznam `maybes` smo uredili glede na število pravil. Zaradi lepšega ga uredimo padajoče. Potem vedno vzamemo zadnji stolpec, tisti, ki ustreza enemu samemu pravilu. Iz množice pravil, ki jim je ustrezal v začetku, odštejemo vsa pravila, ki so že zasedena. In nato zabeležimo, da to pravilo ustreza temu stolpcu.

Naloga hoče, da izračunamo zmnožek tistih števil v naši vozovnici, ki ustrezajo podatkom, katerih ime se začne z *departure*.

[19]:
```
from functools import reduce
from operator import mul

print(reduce(mul, (my_ticket[i]
                   for field, i in assignments.items()
                   if field.startswith("departure"))))
```

```
953713095011
```

1.4 Celoten program

Ker je bilo tole razreseno po celi beležki, zberimo celoten program še na enem mestu.

[20]:
```
from functools import reduce
from operator import mul


# Branje podatkov

rules_part, my_ticket, tickets = open("input.txt").read().split("\n\n")

rules = {}
for rule in rules_part.splitlines():
    field, intervals = rule.split(":")
    rules[field] = set.union(*(set(range(int(f), int(t) + 1))
                               for f, t in (p.split("-") for p in intervals.
 ↪split(" or "))))

my_ticket = [int(x) for x in my_ticket.splitlines()[1].split(",")]

tickets = [[int(x) for x in v.split(",")]
           for v in tickets.splitlines()[1:]]


# Part 1

all_valid = set.union(*rules.values())
print(sum(sum(x
              for x in ticket
              if x not in all_valid)
          for ticket in tickets))


# Part 2
```

```
valid_tickets = [ticket for ticket in tickets if set(ticket) <= all_valid]

columns = ({ticket[i] for ticket in valid_tickets}
           for i in range(len(rules)))

maybes = [(i, {field for field, valids in rules.items() if column <= valids})
          for i, column in enumerate(columns)]

maybes.sort(key=lambda x: len(x[1]), reverse=True)
assignments = {}
while maybes:
    i, fields = maybes.pop()
    field = (fields - set(assignments)).pop()
    assignments[field] = i

print(reduce(mul, (my_ticket[i]
                   for field, i in assignments.items()
                   if field.startswith("departure"))))
```

```
19060
953713095011
```

1.5 Drugi del: dve simpatični bližnjici

`columns` je pravzaprav transponiran `valid_tickets`. Za transponiranje obstaja trik z `zip`-om.

[21]:
```
s = [(1, 2, 3), (4, 5, 6), (7, 8, 9), (10, 11, 12)]
list(zip(*s))
```

[21]:
```
[(1, 4, 7, 10), (2, 5, 8, 11), (3, 6, 9, 12)]
```

Kako to deluje? Preprosto. Vse štiri terke postanejo argumenti `zip`-a, kot da bi napisali `zip((1, 2, 3), (4, 5, 6), (7, 8, 9), (10, 11, 12))`. `zip` gre vzporedno čez vse terke; najprej pobira prve elemente, nato druge, nato tretje.

V našem primeru torej naredimo `zip(*valid_tickets)`. Iz vsake od dobljenih terk želimo narediti množico, torej jih premapiramo čez `map(set, ...)`.

Na ta način `valid_tickets` potrebujemo le enkrat. Tako so `valid_tickets` lahko kar generator.

Še bolj kul pa je nadaljevanje. Vemo, da bomo polje najprej dodelili tistemu stolpcu, ki mu ustreza le eno pravilo, drugi stolpec tistemu, ki mu ustrezata dve in tako naprej. Po drugi strani pa vemo tudi, da bo najprej dodeljeno tisto pravilo, ki se pojavi le pri enem stolpcu, nato tisto, ki se pri dveh…

Potemtakem lahko preprosto uredimo stolpce po številu ustrezajočih polj in polja po številu ustrezajočih stolpcev, pa se bodo natančno ujemali.

[23]:
```
# Part 2

```

```
from collections import Counter
from itertools import chain
from functools import reduce
from operator import mul

valid_tickets = (ticket for ticket in tickets if set(ticket) <= all_valid)
columns = map(set, zip(*valid_tickets))

maybes = [(i, {field for field, valids in rules.items() if column <= valids})
          for i, column in enumerate(columns)]

counts = Counter(chain(*(x[1] for x in maybes)))

indices = (maybe[0] for maybe in sorted(maybes, key=lambda x: len(x[1])))
fields = sorted(counts, key=counts.get, reverse=True)

print(reduce(mul, (my_ticket[i]
                   for i, field in zip(indices, fields)
                   if field.startswith("departure"))))
```

```
953713095011
```

1.6 Pa če ne bi imeli takšne sreče?

Kot vse podobne naloge je tudi ta zanimala vašega asistenta Tomaža Hočevarja. Hitro je našel članek, izrek, dokaz … da bomo pri vsaki rešitvi, ki je enolična, gotovo imeli srečo. Sestavljalec naloge nam tu preprosto ni mogel otežiti življenja … razen če bi bilo možnih različnih odgovorov več.