3_resitev

January 28, 2024

0.1 1. Točni vlaki Slovenskih železnic (hipotetični, miselni eksperiment)

Napiši funkcijo `tocni(redi, dejanski)`, ki prejma dva seznama seznamov. Vsak element se nanaša na en vlak; istoležni elementi se nanašajo na iste vlake. Prvi seznam vsebuje predvidene čase prihoda na postaje, drugi pa dejanske prihode. Časi so podani v minutah od polnoči. Funkcija naj vrne število vlakov, ki niso na nobeni postaji zamujali več kot dvajset minut (kar uporabniki Slovenskih železnic štejemo za praktično točen prihod). Klic

```
tocni([[570, 590, 616, 620], [1200, 1500], [800, 900, 1000], [700, 800]],
      [[570, 611, 622, 630], [1200, 1510], [810, 910, 1000], [800, 900]]))
```

vrne 2: "točna" sta bila drugi in tretji vlak. Prvi in četrti sta vsaj enkrat zamudila več kot 20 minut.

Namig: če želiš, si lahko napišeš tudi pomožno funkcijo tocen(red, dejansko), ki prejme seznam za en sam vlak in vrne True, če je bil točen in False, če je kje zamudil za več kot 20 minut. (Če je ne napišeš, ignoriraj teste zanjo.)

0.1.1 Rešitev

Ubogajmo, napišimo funkcijo `tocen`. Gremo čez vse pare predvidenih in dejanskih časov; če naletimo na zamudo, vrnemo `False`, sicer, na koncu, po zanki, vrnemo `True`.

[1]:
```
def tocen(red, dejansko):
    for cas_red, cas_dejansko in zip(red, dejansko):
        if cas_dejansko - cas_red > 20:
            return False
    return True
```

Naprej je preprosto: gremo čez vse vlake in za vsakega, ki je bil točen, prištejemo 1.

[2]:
```
def tocni(red, dejansko):
    tocnih = 0
    for r, d in zip(red, dejansko):
        if tocen(r, d):
            tocnih += 1
    return tocnih
```

Če znamo uporabljati izpeljane sezname (in če vemo, da je `False` isto kot `0` in `True` isto kot `1`), sta funkciji lahko tudi takšni:

[3]:
```
def tocen(red, dejansko):
    return all(d - r <= 20 for r, d in zip(red, dejansko))

def tocni(redi, dejanski):
    return sum(tocen(red, dejansko) for red, dejansko in zip(redi, dejanski))
```

Brez dodatne funkcije je malo bolj zoprno. Ena od za silo elegantnih rešitev je `else` po `for`:

[4]:
```
def tocni(red, dejansko):
    tocnih = 0
    for red1, dejansko1 in zip(red, dejansko):
        for predviden, prisel in zip(red1, dejansko1):
            if prisel > predviden + 20:
                break
        else:
            tocnih += 1
    return tocnih
```

Obstajajo pa seveda tudi druge.

Kdor je prelen za pisanje dodatne funkcije, zna pa uporabljati izpeljane sezname, pa zbaše vse skupaj v

[5]:
```
def tocni(red, dejansko):
    return sum(
        all(prisel <= predviden + 20 for prisel, predviden in zip(pri, pre))
        for pri, pre in zip(red, dejansko)
    )
```

0.2 2. Enake linije

Dve liniji sta "enaki", če vsebujeta iste postaje **ali** pa imata isto začetno in končno postajo ter ena od njiju vsebuje vse postaje, ki jih vsebuje druga; v tem primeru je druga hitrejša različica prve. Noben vlak ne ustavi večkrat na isti postaji.

Napiši funkcijo `enaki(linija1, linija2)`, ki prejme seznama imen postaj in vrne `True`, če sta liniji enaki in `False`, če ne. Funkcija naj ignorira vrstni red (razen prve in zadnje postaje, ki morata biti enaki): liniji Ljubljana - Sevnica - Trebnje - Novo mesto in Ljubljana - Trebnje - Sevnica - Novo mesto sta enaki.

- Liniji Ljubljana - Kresnice - Jevnica - Litija - Zagorje - Zidani most in Ljubljana - Jevnica - Zidani most sta enaki.
- Liniji Ljubljana - Kresnice - Zidani Most in Ljubljana - Zagorje - Zidani most nista enaki, ker vsaka vsebuje kakšno postajo, ki je druga nima.
- Liniji Kresnice - Zagorje in Ljubljana - Kresnice - Zagorje nimata enake začetne postaje.

0.2.1 Rešitev

Študent naj bi razmislil, kaj pomeni, da ima ena linija vse postaje, ki jih ima druga: gre za podmnožice. Seznama torej pretvorimo v množice in preverimo ali je ena podmnožica druge oziroma

obratno. Poleg tega pa preverimo še prvo in zadnjo postajo, torej:

[6]:
```
def enaki(linija1, linija2):
    return linija1[0] == linija2[0] \
        and linija1[-1] == linija2[-1] \
        and (set(linija1) <= set(linija2)
             or set(linija2) <= set(linija1))
```

Ne spreglejte oklepajev. Imamo tri stvari: prvi postaji sta enaki IN zadnji sta enaki IN (prva je podmnožica druge ali pa je druga podmnožica prve). Brez teh oklepajev rešitev ni pravilna, ker ima `and` prednost pred `or`.

0.3 3. Kopičenje zamud

Napiši funkcijo `zamujenih(vlak, cakajoci)`, ki prejme niz s kodo vlaka, ki zamuja in slovar, katerega ključi so kode vlakov, vrednosti pa množice vseh vlakov, ki morajo čakati, kadar ta vlak zamuja. Funkcija naj vrne število vseh zamujenih vlakov. Pri tem upoštevaj, da bodo zamujali tudi vlaki, ki čakajo zamujene vlake, ki čakajo zamujene vlake …

V primeru iz testov je tako: če zamuja LP3682, bosta zaradi tega zamujala tudi LP8524 in IC021. Ker zamuja LP8524 bodo morali čakati tudi EN123, IC521 in LP6316, zaradi IC021 pa LP222 in IC204. Zaradi EN123 ne čaka nihče, zaradi IC521 pa … in tako naprej. Skupno zaradi LP3682 zamudi 14 vlakov.

Vlaki se nikoli ne čakajo v krogu: če A čaka B in B čaka C, potem ne B ne C ne čakata na A.

0.3.1 Rešitev

Vrniti morate velikost "rodbine vlakov" - kar je prva naloga iz rekurzije, ki smo jo delali na predavanjih…

[7]:
```
def zamujenih(vlak, cakajoci):
    zamud = 1
    for vlak1 in cakajoci[vlak]:
        zamud += zamujenih(vlak1, cakajoci)
    return zamud
```

Malo hitreje pa je tako:

[8]:
```
def zamujenih(vlak, cakajoci):
    return 1 + sum(zamujenih(vlak1, cakajoci) for vlak1 in cakajoci[vlak])
```

Kdor ve kaj več, je opazil, da ni tako preprosto: lahko bi se zgodil, da bi na A čakala B in C, potem pa bi na oba, torej na B in na C, čakal nek vlak D. Tale program bi D štel dvakrat. To bi lahko rešili tako, da bi namesto preštevanja našteli vsa imena v rodbini in pogledali, koliko jih je.

0.4 4. Raztegnjen vozni red

V želji odpraviti zamude bodo Slovenske železnice preprosto raztegnile vozni red: vsi časi bodo povečani za 10 %. (Vlake, ki bi zaradi tega odpeljali naslednji dan, pa bodo preprosto ukinili.)

SŽ shranjujejo čase v obliki parov (terk) (ure, minute), na primer `(11, 48)`.

Napiši funkcije:

- `v_minute(cas)`, ki prejme čas v obliki terke in vrne čas v minutah od polnoči. Če je `cas = (11, 48)`, klic `v_minute(cas)` vrne `708` (to je, 11 * 60 + 48);
- `v_cas(minute)`, prejme čas v minutah od polnoči in vrne terko z urami in minutami. Klic `v_cas(708)` vrne `(11, 48)`;
- `raztegni(cas)` prejme čas v obliki terke in vrne čas v obliki terke. Čas, ki ga vrne, mora biti enak podanemu času, pomnoženemo z 1.1 in zaokroženemu na najbližje celo število. Za zaokrožanje uporabi funkcijo `round(x)`. Če je cas = `(11, 48)`, klic `raztegni(cas)` vrne `(12, 59)` (708 po polnoči spremeni v 779 minut po polnoči in ga pretvori nazaj v ure in minute).

0.4.1 Rešitev

Tole je le naloga, ki preverja, ali znate napisati par funkcij in v njih malo računati.

[9]:
```
def v_minute(cas):
    return cas[0] * 60 + cas[1]

def v_cas(minute):
    return minute // 60, minute % 60

def raztegni(cas):
    return v_cas(round(1.1 * v_minute(cas)))
```

0.5 Vse skupaj

Rešitev celotnega izpita (brez ekstra trikov) je torej takšna:

[10]:
```
# 1

def tocen(red, dejansko):
    for cas_red, cas_dejansko in zip(red, dejansko):
        if cas_dejansko - cas_red > 20:
            return False
    return True

def tocni(red, dejansko):
    tocnih = 0
    for r, d in zip(red, dejansko):
        if tocen(r, d):
            tocnih += 1
    return tocnih

# 2

def enaki(linija1, linija2):
```

```
    return linija1[0] == linija2[0] \
        and linija1[-1] == linija2[-1] \
        and (set(linija1) <= set(linija2)
             or set(linija2) <= set(linija1))

# 3

def zamujenih(vlak, cakajoci):
    zamud = 1
    for vlak1 in cakajoci[vlak]:
         zamud += zamujenih(vlak1, cakajoci)
    return zamud

# 4

def v_minute(cas):
    return cas[0] * 60 + cas[1]

def v_cas(minute):
    return minute // 60, minute % 60

def raztegni(cas):
    return v_cas(round(1.1 * v_minute(cas)))
```