resitev

January 28, 2024

0.1 Prenašalci

Ana je šla na kavo. Berta tudi. Ana je bila pri tem kot oni genij, ki sem ga zadnjič videl v Hofru: načelno je imel masko (čez usta, nos se mu je zdel prelep, da bi ga skril), le kadar se je pogovarjal z ženo, jo je potegnil dol (mislim, masko), da je lažje govoril. No, resnici na ljubo jo je Ana morala sneti tudi zato, da je lahko pila kavo. Potem je šla Ana še k zdravniku. Dani tudi. Cilka je bila pa tudi na kavi. Pa še na telovadbi. Tam je bila tudi Ema.

V nekoliko drugačnem vrstnem redu, tako:

```
obiski = [("Ana", "kava"), ("Berta", "kava"), ("Cilka", "telovadba"),
          ("Dani", "zdravnik"), ("Ana", "zdravnik"), ("Cilka", "kava"),
          ("Ema", "telovadba")]
```

Napiši naslednje funkcije.

- `osebe(obiski)` prejme seznam, kot je gornji in vrne množico imen vseh akterjev (v gornjem primeru `{"Ana", "Berta", "Cilka", "Dani", "Ema"}`);
- `dejavnosti(obiski)` prejme takšen seznam in vrne množico vseh aktov (npr. `{"kava", "telovadba", "zdravnik"}`);
- `dejavnosti_osebe(oseba, obiski)` vrne množico dejavnosti, ki se jih je udeležila podana oseba;
- `udelezenci(dejavnost, obiski)` vrne množico oseb, ki so se udeležile podane dejavnosti;
- `v_stiku(oseba1, oseba2, obiski)` vrne `True`, če sta se podani osebi udeležila kake iste dejavnosti;
- `v_karanteno(oseba, obiski)` vrne množico oseb, ki jih je podana `oseba` morda okužila;
- `zlatko(obiski)` vrne ime osebe, ki so bile v stiku z največ drugimi osebami. Če je enako zlatih prinašalcev več, lahko funkcija vrne kateregakoli od njih.
- `osumljenci(zbolel, zdravi, obiski)` prejme ime nekoga, ki je zbolel in množico imen ljudi, ki so preverjeno zdravi. Vrniti mora množico oseb, za katere sumimo, da so morda okužili tega, ki je zbolel. Recimo, da pokličemo `osumljenci("Ana", {"Berta"}, obiski)`. Se pravi: Ana je zbolela. Kdo jo je okužil? Ana se je družila z Berto, Cilko in Dani. Vendar vemo, da je Berta zdrava. Prav tako vemo, da Ane gotovo ni okužila Cilka: če bi virus stresala Cilka, bi zbolela tudi Berta. Kandidat za okužbo Ane je torej samo Dani. V splošnem: kandidati za okužbo zbolelega so tiste osebe, ki so se družile z zbolelim, niso pa se družile z nikomer od zdravih (in, seveda, tudi sami niso med preverjeno zdravimi).

0.1.1 Rešitev

Prvi štiri funkcije bi morale biti trivialne.

[1]:
```
def osebe(obiski):
    vse_osebe = set()
    for oseba, _ in obiski:
        vse_osebe.add(oseba)
    return vse_osebe

def dejavnosti(obiski):
    vse_dejavnosti = set()
    for _, dejavnost in obiski:
        vse_dejavnosti.add(dejavnost)
    return vse_dejavnosti

def dejavnosti_osebe(oseba, obiski):
    dejavnosti = set()
    for oseba1, dejavnost in obiski:
        if oseba1 == oseba:
            dejavnosti.add(dejavnost)
    return dejavnosti

def udelezenci(dejavnost, obiski):
    osebe = set()
    for oseba, dejavnost1 in obiski:
        if dejavnost1 == dejavnost:
            osebe.add(oseba)
    return osebe
```

Omembe vredna so le imena spremenljivk. - Prvi dve imata spremenljivko, katere vrednost nas ne zanima, tu je le zaradi zanke. Zato ji ne damo posebnega imena, temveč jo poimenujemo le podčrtaj. - Množico, ki jo moramo vrniti, bi v prvi funkciji najraje poimenovali `osebe` in v drugi `dejavnosti`. Tega nismo storili, ker vobče ni dobra ideja, če se spremenljivka znotraj funkcije imenuje enako kot funkcija (ali, načelno, enako kot katera druga funkcija). V tem programu s tem ne bi bilo težav, prišli pa bodo programi, kjer bomo morali na to resneje paziti. - V drugih dveh funkcijah bi nas mikalo napisati zanko `for oseba, dejavnost in obiski`, vendar bi s tem `oseba` ali `dejavnost` (v eni funkciji eno, v drugi drugo) imela enako ime kot argument funkcije in to zagotovo ne bi bilo dobro.

Zdaj pa ostale štiri funkcije.

Dve osebi sta bili v stiku, če sta se obe udeležili kake iste dejavnosti. Z drugimi besedami, če je presek njunih dejavnosti neprazen. To sprogramiramo, recimo, tako:

[3]:
```
def v_stiku(oseba1, oseba2, obiski):
    return dejavnosti_osebe(oseba1, obiski) & dejavnosti_osebe(oseba2, obiski) !
 ↪= set()
```

Naloga ne pove jasno, ali je vsaka oseba v stiku s seboj. V gornji rešitvi smo predpostavili, da je.

Naslednja funkcija je potem spet podobna prvim štirim: sestavljamo množico vseh oseb, ki so bile v stiku s podano. Gremo čez vse osebe (ki nam jih da funkcija `osebe`) in tiste, ki so v stiku s

podano, damo med potencialno okužene.

[4]:
```
def v_karanteno(oseba, obiski):
    okuzeni = set()
    for oseba1 in osebe(obiski):
        if oseba1 != oseba and v_stiku(oseba1, oseba, obiski):
            okuzeni.add(oseba1)
    return okuzeni
```

Zlatka poiščemo tako, da gremo prek vseh oseb in za vsako preverimo, ali je spravila v karanteno več ljudi, kot najbolj zlata oseba doslej. Začetni zlatko je lahko kar oseba z izmišljenim imenom, recimo "".

[5]:
```
def zlatko(obiski):
    naj_zlatejsi = ""
    for oseba in osebe(obiski):
        if len(v_karanteno(oseba, obiski)) > len(v_karanteno(naj_zlatejsi,␣
 ↪obiski)):
            naj_zlatejsi = oseba
    return naj_zlatejsi
```

Zadnja naloga je bila ena od dveh (poleg `v_stiku`), kjer je bilo res potrebno delati z množicami. No, kaj hudega nam tudi tu ni bilo. V začetku so osumljeni vsi, ki so bili v stiku z zbolelo osebo. Tu spet uporabimo kar funkcijo `v_karanteno`, čeprav na nek način v rikverc - ko smo jo pisali, smo si predstavljali, da bo povedala, koga podana oseba okuži, vendar opazuje le stike, torej deluje tudi v drugo smer. Nato iz množice osumljenih takoj odstranimo vse, ki so zdravi. Potem pa gremo še prek seznama vseh, ki so bili v stiku z zdravimi, saj ti, naivno gledano, tudi ne morejo biti bolni in torej niso mogli okužiti podane osebe.

[6]:
```
def osumljenci(zbolel, zdravi, obiski):
    stiki = v_karanteno(zbolel, obiski) - zdravi
    for osebe in zdravi:
        stiki -= v_karanteno(osebe, obiski)
    return stiki
```