resitev

January 28, 2024

0.1 Seznami

1. Napiši funkcijo `unikati(s)`, ki prejme seznam in vrne seznam, ki vsebuje iste elemente kot `s`, v enakem vrstnem redu, vendar brez ponovitev. Funkcija *ne sme spreminjati* podanega seznama, temveč mora vrniti novega.

   Klic `unikati(["Ana", "Ana", "Berta", "Cilka", "Ana", "Berta", "Berta", "Berta", "Ema", "Dani", "Cilka"])` vrne `["Ana", "Berta", "Cilka", "Ema", "Dani"]`.

2. Napiši funkcijo `skupnih(s, t)`, ki prejme dva seznama in vrne število skupnih elementov. Morebitne ponovljene elemente mora ignorirati.

   Klic `skupnih(["Ana", "Berta", "Ana", "Ana", "Cilka"], ["Cilka", "Dani", "Ana", "Ana"])` vrne 2, ker imate seznama dva skupna elementa (Ano in Cilko). To, da se Ana ponovi večkrat, ga ne zmede.

3. Napiši funkcijo `vseh(s, t)`, ki vrne število vseh elementov, ki se pojavijo v `s` ali `t` (ali v obeh).

   Klic `skupnih(["Ana", "Berta", "Ana", "Ana", "Cilka"], ["Cilka", "Dani", "Ana", "Ana"])` vrne 4, ker se v podanih seznamih pojavljajo 4 različna imena.

0.1.1 Rešitev

Za `unikate` bomo najprej sestavili prazen seznam in vanj prepisovali elemente iz podanega seznama, vendar za vsakega poprej preverili, da ni morda že tam.

[1]:
```
def unikati(s):
    t = []
    for x in s:
        if x not in t:
            t.append(x)
    return t
```

Ker gre za čisto generične zadeve, bosta `s` in `t` primerni imeni za seznama.

Število `skupnih` elementov ugotovimo tako, da enega od seznamov pustimo čez unikati. Nato za vsakega od teh, unikatnih elementov, preverimo, ali se pojavi tudi v drugem seznamu.

[2]:
```
def skupnih(s, t):
    sk = 0
```

```
    for x in unikati(s):
        if x in t:
            sk += 1
    return sk
```

Drugega seznama ni treba "unificirati". Enega pa moramo, sicer bi ponovljene elemente šteli večkrat.

Vseh je toliko, kolikor je unikatnih v obeh skupaj.

[3]:
```
def vseh(s, t):
    return len(unikati(s + t))
```

Če že poznamo množice, sta zadnji dve funkciji (še) veliko preprostejši.

[1]:
```
def vseh(s, t):
    return len(unikati(s + t))

def vseh(s, t):
    return len(set(s) | set(t))
```

Tudi prvo bi nas lahko zamikalo rešiti z množicami: seznam pretvorimo v množico, da se znebimo duplikatov

0.2 Procesiranje seznamov

1. Napiši funkcijo `preberi_datoteko(ime_dat, locilo)`, ki prejme ime datoteke, podobne tem, s kakršnimi smo delali doslej. Razlika je le v tem, da ločilo med stolpci ni nujno vejica, zato je ločilo podano z drugim argumentom. Funkcija mora vrniti seznam seznamov. Vsak element vrnjenega seznama je vrstica, ki vsebuje podatke iz vrstice.

   Recimo, da imamo datoteko `kolesa.txt` s takšno vsebino:

   ```
   Cube-5031-159-Janez-2017
   Stevens-3819-1284-Ana-2012
   Focus-3823-1921-Benjamin-2019
   ```

   Klic `preberi_datoteko("kolesa.txt", "-")` mora vrniti seznam

   ```
   [['Cube', '5031', '159', 'Janez', '2017\n'],
    ['Stevens', '3819', '1284', 'Ana', '2012\n'],
    ['Focus', '3823', '1921', 'Benjamin', '2019\n']]
   ```

   (Ne vznemirjaj se zaradi `\n` v zadnjih elementih.)

   Nasvet: funkcija je zelo preprosta. V seznam trpaj, kar ti vrača `split`.

2. Napiši funkcijo `filtriran(s, stolpec, vrednost)`, ki prejme seznam, kakršnega vrača prejšnja funkcija in vrne nov seznam, ki vsebuje samo tiste elemente, ki imajo v podanem "stolpcu" `stolpec` (se pravi: na podanem indeksu) vrstici podano `vrednost`.

   Recimo, da imamo seznam

```
s = [["Ana", 5, 9, "Berta"],
     ["Cilka", 5, 12, "Berta"],
     ["Ana", 5, 9, "Cilka"],
     ["Berta", 5, 1, "Ana"]]
```

Klic `filtriran(s, 0, "Ana")` vrne seznam, ki vsebuje le tiste elemente seznama `s`, ki imajo v ničtem elementu niz `"Ana"`. Rezultat klica je torej

```
[["Ana", 5, 9, "Berta"],
 ["Ana", 5, 9, "Cilka"]]
```

Klic `filtriran(s, 3, "Berta")` vrne

```
[["Ana", 5, 9, "Berta"],
 ["Cilka", 5, 12, "Berta"]]
```

Klic `filtriran(s, 1, 5)` vrne kar celoten seznam, saj imajo na prvem mestu slučajno vsi ravno 5.

3. Napiši funkcijo `izlusci(s, stolpec)`, ki vrne vse *različne* vrednosti, ki se pojavijo v podanem "stolpcu". Vrednosti morajo nastopati v takem vrstnem redu, v kakršnem se prvič pojavijo.

   Če je `s` seznam iz prejšnjega primera, potem klic `izlusci(s, 0)` vrne `["Ana", "Cilka", "Berta"]`. Klic `izlusci(s, 1)` vrne `[5]`. Klic `izlusci(s, 3)` vrne `["Berta", "Cilka", "Ana"]`.

0.2.1 Rešitev

Datoteke smo brali že tolikokrat, da sploh ni več česa komentirati.

[4]:
```
def preberi_datoteko(ime_dat, locilo):
    zapisnik = []
    for vrstica in open(ime_dat):
        zapisnik.append(vrstica.split(locilo))
    return zapisnik
```

Tudi s filtriranjem se nimamo kaj mujati: beremo vrstice in v seznam dajemo le tiste, ki imajo v pravem stolpcu pravo vrednost.

[5]:
```
def filtriran(s, stolpec, vrednost):
    t = []
    for vrstica in s:
        if vrstica[stolpec] == vrednost:
            t.append(vrstica)
    return t
```

Tudi `izlusci` je preprosta: v seznam zlagamo vse, kar najdemo v drugem stolpcu.

[6]:
```
def izlusci(s, stolpec):
    t = []
    for vrstica in s:
```

```
        t.append(vrstica[stolpec])
    return t
```

Dodajmo le še, da so vse tri funkcije klasični primeri nečesa, kar se bomo naučili narediti v eni sami vrstici.

[7]:
```
def preberi_datoteko(ime_dat, locilo):
    return [vrstica.split(locilo) for vrstica in open(ime_dat)]

def filtriran(s, stolpec, vrednost):
    return [vrstica for vrstica in s if vrstica[stolpec] == vrednost]

def izlusci(s, stolpec):
    return [vrstica[stolpec] for vrstica in s]
```

0.3 Dražba

Čestitam, prišli ste do sem, programiranja je zdaj skoraj konec. Če boste pametni, boste poslej samo klicali funkcije, ki ste jih napisali doslej.

1. Napiši funkcijo `predmeti(ime_dat, oseba)`, ki prejme ime datoteke z zapisnikom dražbe in ime neke osebe. Vrniti mora seznam vseh predmetov, za katere se je zanimala ta oseba.

2. Napiši funkcijo `osebe(ime_dat, predmet)`, ki prejme ime datoteke z zapisnikom in ime predmeta. Vrniti mora osebe, ki so se zanimale za ta predmet.

3. Napiši funkcijo `podobnost_oseb(ime_dat, oseba1, oseba2)`. Ta prejme ime datoteke z zapisnikom in imeni dveh oseb. Vrniti mora podobnost oseb. Podobnost oseb je definirana kot število predmetov, za katere sta se zanimali obe osebi, deljenemu s številom predmetov, za katere se je zanimala ena ali druga ali obe. (Glej Jaccardov index).

   Klic `podobnost_oseb("zapisnik.txt", "Cilka", "Ema")` vrne 0.5. Cilka je hotela `['pozlačen dežnik', 'kip', 'srebrn jedilni servis']`, Ema pa `['Meldrumove vaze', 'kip', 'srebrn jedilni servis']`. Skupna predmeta sta 2 (kip in servis), vseh predmetov, za katere sta se zanimali (ena ali druga ali obe), pa 4. Funkcija vrne 2 / 4, torej 0.5.

4. Napiši funkcijo `podobnost_predmetov(ime_dat, predmet1, predmet2)`, ki prejme ime datoteke in imeni dveh predmetov. Vrniti mora podobnost predmetov. Ta je enaka številu oseb, ki so se zanimale za oba predmeta, deljenemu s številom oseb, ki so se zanimale za vsaj enega.

0.3.1 Rešitev

Za `predmeti` preberemo datoteko, izfiltriramo ven tiste, ki imajo v prvem stolpcu ustrezno ime osebe, nato izluščimo vrednosti iz ničtega stolpca in poberemo ven unikate.

[8]:
```
def predmeti(ime_dat, oseba):
    zapisnik = preberi_datoteko(ime_dat, ",")
    z_osebo = filtriran(zapisnik, 1, oseba)
    predm = izlusci(z_osebo, 0)
    return unikati(predm)
```

Gre tudi v eni vrstici,

[9]:
```
def predmeti(ime_dat, oseba):
    return unikati(izlusci(filtriran(preberi_datoteko(ime_dat, ","), 1, oseba),␣
 ↪0))
```

Vendar se pri tem izgubimo v klicih, oklepajih in argumentih. Preglednejše je lepo sproti zlagati v spremenljivke.

Funkcija `osebe` je analogna.

[10]:
```
def osebe(ime_dat, predmet):
    zapisnik = preberi_datoteko(ime_dat, ",")
    s_predmetom = filtriran(zapisnik, 0, predmet)
    oseb = izlusci(s_predmetom, 1)
    return unikati(oseb)
```

Preden se lotimo funkcij `podobnost_oseb` in `podobnost_predmetov` si lahko samoiniciativno napišemo funkcijo `podobnost`, ki meri podobnost seznamov `s` in `t`.

[11]:
```
def podobnost(s, t):
    return skupnih(s, t) / vseh(s, t)
```

Zdaj sta funkciji `podobnost_oseb` in `podobnost_predmetov` le še

[12]:
```
def podobnost_oseb(ime_dat, oseba1, oseba2):
    return podobnost(predmeti(ime_dat, oseba1), predmeti(ime_dat, oseba2))

def podobnost_predmetov(ime_dat, predmet1, predmet2):
    return podobnost(osebe(ime_dat, predmet1), osebe(ime_dat, predmet2))
```

0.4 Priporočilni sistem (Dodatna, neobvezna naloga)

Neobvezno, vendar preprosto, sploh, če uporabite funkcijo `argmax`, ki smo jo napisali na predavanju. Kar skopirajte jo v datoteko s svojim programom.

1. Napiši funkcijo `priporoci_predmet(ime_dat, predmet)`, ki prejme ime datoteke in ime nekega predmeta. Vrne naj predmet, ki je najbolj podoben temu predmetu. (Seveda je vsak predmet najbolj podoben samemu sebi; vrniti mora naslednji najpodobnejši predmet.)
2. Napiši funkcijo `priporoci_prijatelja(ime_dat, oseba)`, ki prejme ime datoteke in ime osebe ter vrne najbolj podobno osebo.

[13]:
```
def argmax(s):
    max_k = max_v = None
    for k, v in s:
        if max_v is None or v > max_v:
            max_k = k
            max_v = v
    return max_k
```

```
def priporoci_predmet(ime_dat, predmet):
    podobnosti = []
    for predmet2 in unikati(izlusci(preberi_datoteko(ime_dat, ","), 0)):
        if predmet2 != predmet:
            podobnosti.append((predmet2, podobnost_predmetov(ime_dat, predmet,␣
 ↪predmet2)))
    return argmax(podobnosti)

def priporoci_prijatelja(ime_dat, oseba):
    podobnosti = []
    for oseba2 in unikati(izlusci(preberi_datoteko(ime_dat, ","), 1)):
        if oseba2 != oseba:
            podobnosti.append((oseba2, podobnost_oseb(ime_dat, oseba, oseba2)))
    return argmax(podobnosti)
```