

Henrik Lončar

Za reševanje te domače naloge uporabljajte numpy. Vsaka funkcija naj bi imela samo eno vrstico z `return`-om, ki vrne ustrezno filtrirano, indeksirano, prešteto ali karkoli že tabelo. Samo pri `naj_dogajanje` bo morda praktično, če boste dodali še eno vrstico.

Da ste rešili, kot je mišljeno, boste prepoznali po tem, da ne bo v programu nobenih zank, generatorjev, izpeljanih seznamov. Da ne zapletam testov, ti tega ne bodo preverjali in če bo kdo rešil kako drugače, bo tudi ta rešitev sprejeta kot pravilna. Vseeno pa zelo priporočam, da se potrudite: gre za uvodne naloge iz numpyja in dobro je, da to obvladate, preden pridejo na vrsto zahtevnejše.

Nalogi sta priloženi dve datoteki.

- "chapters.txt" vsebuje imena poglavij neke otroške knjige.
- "occurences.txt" ima tri stolpce: številka poglavja, ime osebe in število omemb te osebe v tem poglavju. V datoteki je, recimo, vrstica `25,Trelawney,21`, ki pomeni, da se v 25. poglavju 21-krat omeni oseba s priimkom Trelawney.

Za nalogo nepomembne opombe. ker sem podatke nabral avtomatsko vsebujejo napake - kadar, na primer, osebo pokličejo po priimku, je težko avtomatsko razbrati, za koga gre. Če bom te podatke uporabljal tudi v prihodnih nalogah, jih bom morda izboljševal.

Nekatere osebe so zapisane z imeni, druge s priimki, saj je bralcem včasih bolj znano eno, včasih drugo (najbrž nihče ne ve, komu je bilo ime Gerrick...). Osebe, ki so predstavljene z imeni in ne priimki, so:

```
["Harry", "Hermione", "Ron", "Ginny", "Fred", "George", "Arthur", "Molly", "Bill", "Fleur",  
 "Cedric", "Cho", "Dean", "Dudley", "Draco", "Charlie", "Katie", "Kendra", "Kingsley",  
 "Luna", "Neville", "Oliver", "Sirius", "Xenophilius", "James", "Lily", "Petunia", "Vernon",  
 "Lucius", "Louis", "Isla", "Roxanne", "Burke", "Cygnus", "Beadle", "Sirius", "Phineas",  
 "Alecto", "Scorpius", "Gwenog", "Beatrice", "Lucius", "Ariana", "Narcissa", "Angelina",  
 "Ignotus", "Percy"]
```

Priprava podatkov

Napiši program (ne funkcije!), ki prebere datoteki in pripravi numpyjeve tabele poglavja, osebe in omembe. Tabele vsebujejo podatke iz stolpcev datoteke "occurences.txt", pri čemer sta poglavje in omembe pretvorjene v `int`. Od poglavje odštejte 1, da bo imelo prvo poglavje številko 0.

Poleg tega preberi vsebino datoteke "chapters.txt" v tabelo `naslovi`.

Testi bodo preverjali, ali obstajajo spremenljivke s temi imeni in vsebujejo (del tega), kar morajo.

```
import numpy as np

naslovi = np.genfromtxt("chapters.txt", delimiter="XXX", dtype=str)
podatki = np.genfromtxt("mentions.txt", delimiter=",", dtype=str)

poglavja, osebe, omembe = podatki.T
poglavja = poglavja.astype(int) - 1
omembe = omembe.astype(int)
```

Obvezna naloga

Napiši naslednje funkcije:

- `naj_oseba()`: vrne ime osebe, ki se omeni največkrat v kateremkoli poglavju. (Funkcija vrne Harry, ker se le-ta 150-krat omeni v 26. poglavju, kar je rekord.)
- `naj_dogajanje()`: podobno kot prejšnja funkcija, le da vrne ime poglavja in ime osebe.
- `prva_omemba(oseba)`: vrne ime poglavja, v katerem se podana oseba prvič omeni.
- `koliko_poglavij(oseba)`: vrne število poglavij, v katerih se omeni podana oseba.
- `v_poglavjih_st(oseba)`: vrne tabelo s številkami poglavij, v katerih se omeni podana oseba.
- `v_poglavjih(oseba)`: vrne tabelo naslovov poglavij, v katerih se omeni podana oseba.
- `vseh_omemb(oseba)`: vrne število omemb podane osebe v celotni knjigi.

Rešitev

Prvi dve funkciji sta očitno podobni. Obe zahtevata, da se spomnimo, kako dobiti indeks največjega elementa `argmax`. Tako dobimo indeks z največjim številom omemb, potem pa poiščemo pripadajoče ime osebe oziroma pripadajoče ime poglavja. Ime poglavja dobimo tako, da najprej izvemo njegov indeks, nato pa gremo s tem indeksom pogledati v seznam naslovov poglavij.

```
def naj_oseba():
    return osebe[np.argmax(omembe)]

def naj_dogajanje():
    i = np.argmax(omembe)
    return naslovi[poglavja[i]], osebe[i]

def prva_omemba(oseba):
    return naslovi[np.min(poglavja[osebe == oseba])]
```

Preostale štiri funkcije zahtevajo masko, s katero dobimo vse elemente, ki se nanašajo na podano osebo, `osebe == oseba`. Ker vsako osebo v vsakem poglavju štejemo le enkrat, nam bo `np.sum(osebe == oseba)` povedal število poglavij

s to osebo, `poglavje[osebe == oseba]` indekse teh poglavij, iz indeksov pa dobimo njihove naslove, če gremo pogledat v `naslovi`. Četrta funkcija v `omembe` pogleda vse števec omemb te osebe in jih z `np.sum` sešteje.

```
def koliko_poglavij(oseba):
    return np.sum(osebe == oseba)

def v_poglavjih_st(oseba):
    return poglavja[osebe == oseba]

def v_poglavjih(oseba):
    return naslovi[poglavja[osebe == oseba]]

def vseh_omemb(oseba):
    return np.sum(omembe[osebe == oseba])
```

Dodatna naloga

- `maska_poglavij(oseba)`: vrne tabelo vrste `bool`, ki je dolga toliko, kolikor je poglavij. `i`-ti element je `True`, če se oseba pojavi v `i`-tem poglavju (pri čemer so poglavja oštevilčena od 0).
- `podobnost(oseba1, oseba2)`: vrne Jaccardovo podobnost med dvema osebama. To izračunamo kot število poglavij, v katerih se pojavita obe osebe, deljena s številom poglavij, v katerih se pojavi vsaj ena od njiju.

Naj bosta `a` in `b` dve tabeli vrste `bool`. Potem `a & b` vrne tabelo, ki ima `True` na mestih, kjer sta obe, `a` in `b` enaki `True`, tabela `a | b` pa na mestih, kjer je vsaj ena enaka `True`.

Rešitev

Ta je bolj zafrknjena. Ni kar tako, za hec, dodatna.

Če mora funkcija vrniti masko poglavij, bo potrebno pripraviti tabelo s toliko elementi, kolikor je poglavij. Vsi elementi bodo `False`. Kot smo se naučili v obvezni nalogi, s `poglavja[osebe == oseba]` izvemo indekse poglavij, v katerih se pojavlja podana oseba. Te elemente maske postavimo na `True`. Pa jo imamo, masko.

```
def maska_poglavij(oseba):
    a = np.zeros(np.max(poglavja) + 1, dtype=bool)
    a[poglavja[osebe == oseba]] = True
    return a
```

Podobnost je potem preprosta. Sestavimo obe maski. Z `&` sestavimo masko, ki vsebuje `True` na mestih, na katerih sta obe maski enaki `True`; to so poglavja, v katerih se pojavljata obe osebi. Z `|` dobimo masko, ki ima `True`-je tam, kjer je `True` vsaj ena od mask. To so poglavja, kjer se pojavi vsaj ena od oseb. Preštejemo `True`-je v obeh sestavljenih maskah in ju delimo.

```
def podobnost(oseba1, oseba2):
    maska1 = maska_poglavij(oseba1)
    maska2 = maska_poglavij(oseba2)
    return np.sum(maska1 & maska2) / np.sum(maska1 | maska2)
```

Nekateri študenti so odkrili, da gre tudi brez mask - z `numpy`-jevimi funkcijami `np.intersect1d` in `np.union1d`. Če imamo

```
poglavja_crabbe = v_poglavjih_st("Crabbe")
poglavja_ron = v_poglavjih_st("Ron")
```

lahko z `np.intersect1d` in `np.union1d` dobimo številke poglavjih, v katerih se omenjata oba oz. vsaj eden.

```
np.intersect1d(poglavja_crabbe, poglavja_ron)
array([ 6,  8, 10, 11, 14, 17, 20, 23, 29])

np.union1d(poglavja_crabbe, poglavja_ron)
array([ 3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
        20, 21, 23, 24, 25, 26, 27, 28, 29])
```

Deliti je potrebno velikosti teh dveh tabel.

```
def podobnost(oseba1, oseba2):
    poglavja1 = v_poglavjih_st(oseba1)
    poglavja2 = v_poglavjih_st(oseba2)
    return len(np.intersect1d(poglavja1, poglavja2)) / len(np.union1d(poglavja1, poglavja2))
```

V Pythonu bodo izkušeni programerji za isto (dovolj kratko) reč tipično napisali praktično enake funkcije. V problemih, ki jih rešujemo z `numpy`-jem ni tako. Malo zaradi njihove narave, malo zato, ker je `numpy` ogromen. Pri sestavljanju te naloge sploh nisem pomislil na rešitev s tema funkcijama, pa tudi za funkciji sem komaj vedel (= tako kot za besedo v tujem jeziku, katere pomen poznaš, nikoli pa ti ne bi priplavala v zavest, da bi jo sam uporabil).

Sam bi vseeno uporabil rešitev z maskami. Tivde funkciji zahtevata urejanje elementov tabele; čeprav to storita interno, se urejanju poskušam izogibati, ker je časovno zahtevnejše od sestavljanja mask. Po drugi strani pa maske vzamejo več pomnilnika. Vsaka različica ima svoje prednosti.