

Izpit 13. februar 2024

1. Skupna cena

Napišite funkcijo `cena(zasedenost, cene)`, ki prejme numpyjevo tabelo `zasedenost` tipa `bool`, ki ima šest stolpcev in toliko vrstic, kolikor vrst ima letalo ter vsebuje `True` na mestih, ki ustrezajo zasedenim sedežem in `False` na mestih, ki ustrezajo prostim. Poleg tega dobi Pythonov seznam `cene`, ki vsebuje cene vozovnic za sedeže v prvih treh stolpcih, na primer `[200, 175, 193]`, če sedeži ob oknu (stolpec 0) stanejo 200, sedeži na sredini (stolpec 1) 175 in sedeži ob hodniku (stolpec 2) 193 evrov. Cene sedežev v drugih treh stolpcih so zrcalno enake tem (na primer 193, 175 in 200).

Funkcija mora vrniti vsoto cen vseh prodanih vozovnic. Funkcija mora biti napisana v numpyju. Točkovanje bo upoštevalo eleganco rešitve.

Rešitev

Najprej rešitev, potem sledi razlaga.

```
def cena(zasedenost, cene):  
    return np.sum(zasedenost * (cene + cene[::-1]))
```

Najprej razširimo cene na vseh šest stolpcev. Če imamo *Pythonov seznam* `cene`, na primer

```
cene = [200, 175, 193]
```

lahko k njemu pripnemo še prezrcaljen seznam,

```
cene + cene[::-1]
```

```
[200, 175, 193, 193, 175, 200]
```

pa dobimo cene za vse stolpce. Zdaj pa imamo dve poti. Očitnejša (a, kot se bo izkazalo, malenkost daljša, vendar vam pri ocenjevanju tega nisem štel v slabo), je, da najprej preštejemo zasedenost vseh stolpcev.

```
import numpy as np
```

```
zasedenost = np.array([  
    [True, False, False, False, False, False],  
    [False, True, True, False, False, False],  
    [False, False, False, True, True, True],  
    [False, False, False, True, True, True],  
    [False, True, False, True, True, True],  
    [False, False, False, True, True, False],  
    [False, False, False, False, True, False]  
])
```

Zasedenost po stolpcih dobimo tako, da tabelo seštejemo po osi 0,

```
np.sum(zasedenost, axis=0)
```

```
array([1, 2, 1, 4, 5, 3])
```

To tabelo pomnožimo s cenami.

```
np.sum(zasedenost, axis=0) * (cene + cene[::-1])
```

```
array([200, 350, 193, 772, 875, 600])
```

pa dobimo vsote cen po stolpcih. To, končno, seštejemo v skupno ceno.

```
np.sum(np.sum(zasedenost, axis=0) * (cene + cene[::-1]))
```

```
2990
```

Druga rešitev je, da celotno tabelo pomnožimo s cenami.

```
zasedenost * (cene + cene[::-1])
```

```
array([[200,  0,  0,  0,  0,  0],
       [ 0, 175, 193,  0,  0,  0],
       [ 0,  0,  0, 193, 175, 200],
       [ 0,  0,  0, 193, 175, 200],
       [ 0, 175,  0, 193, 175, 200],
       [ 0,  0,  0, 193, 175,  0],
       [ 0,  0,  0,  0, 175,  0]])
```

In to seštejemo.

```
np.sum(zasedenost * (cene + cene[::-1]))
```

```
2990
```

Druga rešitev je krajša, prva (tista, ki smo jo napisali tudi na vrhu) pa najbrž hitrejša. Z vidika tega predmeta sta obe enako dobri.

2. Število letov

Imamo tabelo, v kateri so pari poletov in pristankov, na primer [(360, 420), (400, 580), (485, 1300), (490, 600), (600, 800), (650, 720), (700, 800), (780, 800), (930, 1380), (950, 1380), (950, 1000), (1100, 1200)]. Časi so v minutah od polnoči. Tako prvi par, (360, 420) pomeni da neko letalo vzleti ob 6. uri zjutraj in pristane ob 7. uri. Vsa letala vzletajo in pristajajo na istem letališču, saj gre za turistične prelete.

Nek turist bi rad v istem dnevu naredil čim več poletov. Pri tem pa mora med pristankom nekega leta in odletom naslednjega miniti vsaj 60 minut, da gre vmes po sendvič. Napišite funkcijo `stevilo_letov(prvi, leti)`, ki prejme `prvi` let, na katerega bo šel in seznam vseh letov, ter vrne največje število poletov, ki jih lahko opravi v enem dnevu.

- Za gornje podatke klic `stevilo_letov((360, 420), leti)` vrne 5, saj lahko gre na lete (360, 420), (490, 600), (700, 800), (950, 1000), (1100, 1200).
- Klic `stevilo_letov((600, 880), leti)` vrne 3, saj bo ujel samo (600,800), (950, 1000), (1100, 1200).

Rešitev

Tole je v bistvu globina rodbine. :) Let B je "potomec" nekega leta A, če mu lahko sledi, torej, če let B odleti vsaj 60 minut kasneje kot let A.

```
def stevilo_letov(prvi, leti):
    naj_letov = 0
    for let in leti:
        if prvi[1] + 60 <= let[0]:
            letov = stevilo_letov(let, leti)
            if letov > naj_letov:
                naj_letov = letov
    return naj_letov + 1
```

Ko sestavljaš nalogo iz rekurzije zlahka spregledaš, da je naloga rešljiva tudi brez. Kolega iz laboratorija me je opozoril na preprosto rešitev: v vsakem koraku vzamemo letalo, ki se bo čimprej vrnilo.

```
def stevilo_letov(naslednji, leti):
    letov = 0
    while naslednji != None:
        cas = naslednji[1] + 60
        naslednji = None
        for let in leti:
            if let[0] >= cas and (naslednji is None or let[1] < naslednji[1]):
                naslednji = let
        letov += 1
    return letov
```

Argument `prvi` smo zaradi higiene preimenovali v `naslednji` - predstavljal bo naslednji let. Imeli bomo zanko, ki bo tekla, dokler imamo še kakšen naslednji let. Znotraj zanke shranimo čas pristanka naslednjega leta in z zanko (znotraj zanke) med vsemi leti iščemo tistega, ki ima najzgodnejši pristanek. To bo naš naslednji let...

To rešitev bi morali razumeti vsi, saj ne vsebuje ničesar preveč zapletenega (in predvsem nobene rekurzije :).

Kdor bolj obvlada Python, lahko napiše še krajše.

```
def stevilo_letov(naslednji, leti):
    letov = 0
    while naslednji != None:
```

```

        naslednji = min((let for let in leti if let[0] >= naslednji[1] + 60),
                        key=lambda x: x[1],
                        default=None)

    letov += 1
    return letov

```

Še imenitnejša (pravilnejša, hitrejša) rešitev je urediti lete po časih pristankov. Predstavljajmo si potnika, ki začne z določenim letom. Vsakič ko pristane, pogleda seznam in izbere naslednji let z najzgodnejšim pristankom...

```

def stevilo_letov(naslednji, leti):
    leti = sorted(leti, key=lambda x: x[1])
    letov = 1
    for let in leti[leti.index(naslednji):]:
        if let[0] >= naslednji[1] + 60:
            naslednji = let
            letov += 1
    return letov

```

"Skrivnost" je le `sort` - tisti `key=lambda x: x[1]` poskrbi, da bo elemente (pare, lete) primerjal po elementu z indeksom 1, torej po pristanku.

3. Razpored

Neka letalska družba pa sedežev ne dodeljuje vnaprej, temveč lahko potniki zgolj izrazijo željo. Potnike nato vkrcavajo po seznamu. Vsak potnik se usede na želeni sedež, če je ta že zaseden, pa se pomika nazaj po vrstah, dokler ne naleti na vrsto, v kateri je sedež v želenem stolpcu prazen, ter ga zasede. Če so, na primer, že zasedeni sedeži 12C, 13C in 14C, se bo potnik, ki bi želel sedeti na 12C, usedel na 15C. Letalska družba uporablja dolga letala, ki imajo 130 vrst, obenem pa ji ne gre prav dobro (čudno, čudno), zato smete predpostaviti, da se bo po tem postopku našel sedež za vsakogar.

Napišite funkcijo `razpored(seznam)`, ki dobi seznam s pari (`ime_potnika`, `zeleni_sedež`) in vrne nov seznam, ki je enak temu, le da so sedeži zamenjani z sedeži, ki jih bodo potniki dejansko dobili.

Pomoč: če je c črka A, B, C, D, E ali F, je `ord(c) - 65` enak 0, 1, 2, 3, 4 oz. 5. Mogoče vam pride prav.

Rešitev

Tole je naloga iz množic in iz zanke `while`. Lahko pa tudi ni iz množic. Vsekakor pa je iz zanke `while`. :)

Nekam bo potrebno shranjevati zasedena mesta. Lahko, torej, pripravimo množico `zasedeni` z zasedenimi mesti, poleg tega pa seznam `dejanski`, v katerega bomo postavljali potnike in ga na koncu vrnili.

```
def razpored(zelje):
    zasedeni = set()
    dejanski = []
    for potnik, sedez in zelje:
        vrsta, stolpec = int(sedež[:-1]), sedez[-1]
        while (vrsta, stolpec) in zasedeni:
            vrsta += 1
        zasedeni.add((vrsta, stolpec))
        dejanski.append((potnik, f"{vrsta}{stolpec}"))
    return dejanski
```

Ker smo toliko delali z numpy-jem -- pa tudi zaradi tega, kako so bili shranjeni podatki v prvi nalogi -- je gotovo prišlo komu na misel namesto množice uporabiti numpy-jevo tabelo. (Priznanje: tudi sam sem nalogo najprej rešil tako. Vendar je to predvsem posledica tega, kaj sem razmišljal, ko sem jo sestavljal.)

```
def razpored(zelje):
    zaseden = np.zeros((130, 6), dtype=bool)
    razpored = []
    for potnik, sedez in zelje:
        vrsta, stolpec = int(sedež[:-1]), ord(sedež[-1]) - 65
        while zaseden[vrsta, stolpec]:
            vrsta += 1
        zaseden[vrsta, stolpec] = True
        razpored.append((potnik, f"{vrsta}{sedež[-1]}"))
    return razpored
```

Takšna rešitev je malenkost bolj zapletena, ker potrebujemo še indeks stolpca. Tega dobimo z `ord`, kot je svetoval opis naloge.

Morda bi koga zmotilo, dvojno shranjevanje podatkov. To, kaj je zasedeno, je razvidno že iz `razpored`, torej je `zaseden` v bistvu nepotreben. To je res, vendar je nerodno, da je `razpored` seznam, v katerem je zoprno preverjati, ali je določen sedež zaseden. Če že hočemo, pa lahko uporabimo slovar, ki za vsak sedež pove, kdo sedi na njem.

```
def razpored(zelje):
    zasedeni = {}
    for potnik, sedez in zelje:
        vrsta, stolpec = int(sedež[:-1]), sedez[-1]
        while (vrsta, stolpec) in zasedeni:
            vrsta += 1
        zasedeni[vrsta, stolpec] = potnik

    dejanski = []
    for (vrsta, stolpec), potnik in zasedeni.items():
        dejanski.append((potnik, f"{vrsta}{stolpec}"))
    return dejanski
```

Prvi del se je malenkost poenostavil. V `return` pa moramo iz slovarja sestaviti pare potnikov in sedežev.

Drugi del lahko poenostavimo, če znamo sestavljati izpeljane sezname:

```
return [(potnik, f"{vrsta}{stolpec}")
        for (vrsta, stolpec), potnik in zasedeni.items())
```

4. Ravnotežje

Napišite funkcijo `ravnotezje(ime_datoteke)`, ki prejme ime datoteke s podatki, ločenimi z vejico. V prvi vrstici so zapisana imena stolpcev. Eden od njih se imenuje sedež; sedeži so zapisani v običajni obliki, npr. 12F ali 1C ali 128E. Funkcija mora vrniti razliko med številom potnikov, ki sedijo na desni (stolpci D, E, F) in levi (stolpci A, B, C) strani letala. Če, recimo, na desni sedi 7 potnikov več, vrne 7; če je 7 potnikov več na levi, vrne -7.!

Rešitev

Ta, ki je normalno in pridno delal domače naloge, se ob tej nalogi lahko zahvali za poceni točke. Potrebno je le uporabiti `DictReader`, brati vrednosti v ustreznem stolpcu in glede na zadnji znak spreminjati ravnotežje letala. Lahko ločeno štejemo levo- in desnosedee ter na koncu vrnemo razliko ...

```
import csv
```

```
def ravnotezje(ime_datoteke):
    levo = desno = 0
    for vrstica in csv.DictReader(open(ime_datoteke, encoding="utf-8"), delimiter=","):
        if vrstica["sedež"][-1] <= "C":
            levo += 1
        else:
            desno += 1
    return desno - levo
```

... lahko pa razliko računamo že kar sproti.

```
def ravnotezje(ime_datoteke):
    ravno = 0
    for vrstica in csv.DictReader(open(ime_datoteke, encoding="utf-8"), delimiter=","):
        if vrstica["sedež"][-1] <= "C":
            ravno -= 1
        else:
            ravno += 1
    return ravno
```

Kakor vam drago, je rekel Vili.

5. Leti

Napišite funkcijo `vozni_redi(potniki, ime_datoteke)`. Argument `potniki` je seznam z imeni potnikov, števkami letov in pari (ura, minuta) za odhod in prihod. Primer podatkov je, recimo `[("Ana Argon", "LH2832", (12, 10), (13, 20)), ("Berta Bor", "U0391", (15, 5), (20, 30)), ("Cilka Cankar", "LH192", (7, 0), (12, 30))]`. Funkcija mora v datoteko s podanim imenom zapisati tabelico z imeni, leti in časi, v naslednji obliki.

Ana Argon	LH2832	12:10-13:20
Berta Bor	U0391	15:05-20:30
Cilka Cankar	LH192	7:00-12:30

Točno obliko - število presledkov - razberite iz testov.

Rešitev

Tudi tale je bila kar poceni.

```
def vozni_redi(potniki, ime_datoteke):
    f = open(ime_datoteke, "w", encoding="utf-8")
    for potnik, let, (od_h, od_m), (pri_h, pri_m) in potniki:
        f.write(f"{potnik:20} {let:>6} {od_h:2}:{od_m:02}-{pri_h:2}:{pri_m:02}\n")
```

Celotna rešitev

Ker smo tule napisali veliko različnih rešitev, združimo vse skupaj še v "pričakovano" rešitev izpita. Tudi za vtis o dolžini celotnega izpita.

```
import re
import csv
```

```
import numpy as np
```

```
def cena(zasedenost, cene):
    return np.sum(zasedenost * (cene + cene[::-1]))
```

```
def stevilo_letov(prvi, leti):
    naj_letov = 0
    for let in leti:
        if prvi[1] + 60 <= let[0]:
            letov = stevilo_letov(let, leti)
            if letov > naj_letov:
                naj_letov = letov
    return naj_letov + 1
```

```

def razpored(zelje):
    zasedeni = set()
    dejanski = []
    for potnik, sedez in zelje:
        vrsta, stolpec = int(sedezi[:-1]), sedez[-1]
        while (vrsta, stolpec) in zasedeni:
            vrsta += 1
        zasedeni.add((vrsta, stolpec))
        dejanski.append((potnik, f"{vrsta}{stolpec}"))
    return dejanski

def ravnotezje(ime_datoteke):
    ravno = 0
    for vrstica in csv.DictReader(open(ime_datoteke, encoding="utf-8"), delimiter=","):
        if vrstica["sedez"][-1] <= "C":
            ravno -= 1
        else:
            ravno += 1
    return ravno

def vozni_redi(potniki, ime_datoteke):
    f = open(ime_datoteke, "w", encoding="utf-8")
    for potnik, let, (od_h, od_m), (pri_h, pri_m) in potniki:
        f.write(f"{potnik:20} {let:>6} {od_h:2}:{od_m:02}-{pri_h:2}:{pri_m:02}\n")

```