resitev

January 28, 2024

1 Angelčin zapis

Lani so na Oddelku za gospodarstvo in motorni promet Mestne občine Ljubljana izvedel, da so študenti FRI razvili programsko opremo za branje njihovih zemljevidov pik in lojtr, zato so brž pripravili nov format zapisa. Tudi ta, novi zapis je že zastarel in ga lahko izbrskate le v lanski domači nalogi. Letos je gospa Angelca (tista z dolgimi nohti) prepričala (= z neumornim najedanjem od juter do večerov spravila v obup) ostale zaposlene na oddelku, namreč da je format grd in da bi ona ovire raje zapisovala tako:

```
 (4) 5--
(13) 90----------   5---- 19---
 (5) 9---           19--   30-----
(4)           9---
(13)         22---- 17---
```

In da če jim ni prav, bo šla do župana, ker ta je edini pameten in se vedno strinja z njo. (Že čim jo zagleda na vratih pisarne. Župan ima pač početi kaj pametnejšega kot poslušati Angelco. Zadnje čase začne nezavedno kimati že, ko na hodniku zasliši zvok njenih visokih pet.)

Jasni Angelčin Opis Ovir (prepoznamo ga po končnici datoteke, .jaoo) je sestavljen tako:

- Prva številka v vrstici pove številko vrstice na kolesarski stezi. Ta je zaprta v oklepaje, saj je tako najbolj logično.

  Gornji primer torej podaja nekaj ovir v vrstici 4, nato nekaj v vrstici 13, pa ovire v vrstici 5, nato še nekaj ovir v vrstici 4 in še nekaj v 13. (Na ta način lahko dopisujejo naknadno dodane ovire preprosto na konec datoteke.)

- Številki v oklepaju sledijo opisi ovir v podani vrstici. Ovira je opisana z začetno koordinato stolpca, sledi pa toliko minusov, kolikor je dolga ovira.

  V četrti vrstici je le ena ovira: začne se v petem stolpcu in je dolga 2. V našem zapisu bi to bilo (5, 6, 4). V 13. vrstici so tri ovire; prva se začne v 90 in je dolga 11, se pravi (90, 100, 13). Druga se začne v 5 in je dolga 4, torej (5, 9, 8). Tretja je (19, 21, 13).

Da je zapis bolj razgiban (Angelca po osnovni izobrazbi pač ni računalnikarka, temveč je končala likovno akademijo, smer Industrijsko in unikatno oblikovanje), lahko vsebuje poljubno število presledkov na začetku ali koncu vrstice ter med navedbami posameznih ovir - tako kot jasno kaže gornji primer.

Disclaimer Čeprav naloge pri tem predmetu temeljijo na resničnih dogodkih, avtor naloge ne ve, ali na MOL res obstaja kakšna Angelca, ki ustreza opisu, vendar te možnosti ne izključuje. Angelca

v tej nalogi je torej izmišljena, podobnost z resničnimi osebami s takšnim ali drugačnim imenom pa povsem verjetna.

1.1 Obvezna naloga

Napiši naslednje funkcije.

- `koordinate(s)` prejme opis ene ovire in vrne terko z njenimi koordinatami.

  Klic `koordinate("5---")` vrne `(5, 7)`, saj gre za oviro, ki se začne v stolpcu 5 in konča v stolpcu 7 - dolga je namreč 3. Klic `koordinate("123-")` vrne `(123, 123)`.

- `vrstica(s)` prejme niz z eno vrstico in vrne seznam trojk `(x0, x1, y)`, ki predstavljajo ovire v tej vrstici.

  Klic `vrstica("  (4) 1---  5------- 15-")` vrne seznam `[(1, 3, 4), (5, 11, 4), (15, 15, 4)]`.

- `preberi(s)` celoten, večvrstični niz z ovirami in vrne seznam ovir. Ovire naj bodo shranjene v takšnem vrstnem redu, v kakršnem se pojavljajo.

  Če jo pokličemo z gornjim nizom, vrne

  ```
  [(5, 6, 4),
   (90, 100, 13), (5, 8, 13), (9, 11, 13),
   (9, 11, 5), (19, 20, 5), (30, 34, 5),
   (9, 11, 4),
   (22, 25, 13), (17, 19, 13)]
  ```

- `intervali(xs)` prejme seznam parov `(x0, x1)` in vrne seznam nizov, ki opisujejo te intervale.

  Klic `intervali([(6, 10), (12, 12), (20, 22), (98, 102)])` vrne `["6-----", "12-", "20---", "98-----"]`.

- `zapisi_vrstico(y, xs)` prejme številko vrstice in seznam parov `(x0, x1)`. Vrniti mora opis ene vrstice.

  Klic `zapisi_vrstico(8, [(6, 10), (12, 12), (20, 22), (98, 102)])` vrne niz `"(8) 6----- 12- 20--- 98-----"`. Pazi: ne dodajaj odvečnih presledkov. Angelca ni rekla, da so obvezni.

1.1.1 Rešitev

`koordinate` Da pridemo do začetka ovire (same številke), se moramo minusov znebiti (`strip("-")`), da dobimo dolžino ovire, pa jih moramo prešteti (`count("-")`).

[1]:
```
def koordinate(s):
    x0 = int(s.strip("-"))
    return x0, x0 + s.count("-") - 1
```

[2]:
```
koordinate("5---")
```

[2]:
```
(5, 7)
```

[3]:
```
koordinate("123-")
```

[3]:
```
(123, 123)
```

vrstica V tej nalogi je naš prijatelj `split()`. Poglejte, kaj naredi z nizom iz primera!

[4]:
```
"  (4) 1---  5------- 15-".split()
```

[4]:
```
['(4)', '1---', '5-------', '15-']
```

Prvi element je številka vrstice (ko se bomo znebili prvega in zadnjega znaka, trapastih Angelčinih oklepajev), ostale elemente pa pomečemo funkciji `koordinate`, da dobimo začetke in konce ovir.

[5]:
```
def vrstica(s):
    ovire = []
    s = s.split()
    y = int(s[0][1:-1])
    for ovira in s[1:]:
        ovire.append(koordinate(ovira) + (y, ))
    return ovire
```

[6]:
```
vrstica("  (4) 1---  5------- 15-")
```

[6]:
```
[(1, 3, 4), (5, 11, 4), (15, 15, 4)]
```

Napisati `s = s.split()` ni prav zgledno. Spremenljivka `s` je s tem spremenila svoj pomen - prej je bila niz, zdaj je seznam kosov tega niza. To storimo, če ravno nimamo domišljije, kako poimenovati novo spremenljivko in če menimo, da s tem ne povzročamo prehudih nejasnosti.

V resnici bi pravi programer v Pythonu to napisal drugače. Tole je sicer tečaj programiranja in ne Pythona, vendar obstaja velika verjetnost, da boste kdaj v življenju programirali tudi v Pythonu, zato ne bo škode, če vidite tudi kakšen lep zgled praktičnega dela v tem jeziku.

[7]:
```
def vrstica(s):
    ovire = []
    y, *xs = s.split()
    y = int(y[1:-1])
    for ovira in xs:
        ovire.append(koordinate(ovira) + (y, ))
    return ovire
```

Razlika je minimalna, gre le za to, kaj smo naredili z rezultatom `split`. Prej smo vse skupaj vrgli v en sam seznam, potem pa prebrali prvi element v `y`, čez ostale pa spustili zanko. Zdaj to razkosamo takoj po klicu - že iz prirejanja je očitno, da je rezultat `split`-a `y`, ostalo pa so x-i. Zvezdica pred `xs` pomeni, da v to spremenljivko shrani vse odvečne elemente, ki niso šli v druge spremenljivke (v našem primeru so "druge spremenljivke" pač le `y`).

Pravzaprav lažem. Pravi programer v Pythonu zamahne s čarobno palico in napiše

[8]:
```
def vrstica(s):
    ovire = []
    y, *xs = s.split()
    y = int(y[1:-1])
    return [(*koordinate(ovira), y) for ovira in xs]
```

Naslednji teden bomo mi tudi.

1.1.2 `preberi`

Podani niz razkosamo na vrstice (`splitlines()`), jih podajamo funkciji vrstica in vse, kar vrne, seštevamo v nov seznam.

[9]:
```
def preberi(s):
    ovire = []
    for vrsta in s.splitlines():
        ovire += vrstica(vrsta)
    return ovire
```

Tule je pomembno, da uporabimo `+=`, ki v seznam `ovire` doda *vse elemente* seznama, ki ga vrne `vrstica` in ne morda `append`, ki bi v seznam ovire dodal *seznam*, ki ga vrne `vrstica`.

Kaj na to poreče zaresen programer?

[10]:
```
def preberi(s):
    return sum(map(vrstica, s.splitlines()), start=[])
```

`map(vrstica, s.splitlines())` pokliče funkcijo `vrstica` za vsak element `s.splitlines()`, `sum` pa vse to lepo sešteje, če mu povemo, da mora prištevati v v začetku prazen seznam.

1.1.3 `intervali`

Ta je preprosta.

[11]:
```
def intervali(ovire):
    intv = []
    for x0, x1 in ovire:
        intv.append(str(x0) + "-" * (x1 - x0 + 1))
    return intv
```

Zares pa tako

[12]:
```
def intervali(ovire):
    return [f"{x0}{'-' * (x1 - x0 + 1)}" for x0, x1 in ovire]
```

`zapisi_vrstico` Tu pa niti rešitev z našim znanjem ni daljša od vrstice.

[13]:
```
def zapisi_vrstico(y, ovire):
    return "(" + str(y) + ") " + " ".join(intervali(ovire))
```

Profesionalec bi naredil skoraj enako, recimo tako

[14]:
```
def zapisi_vrstico(y, ovire):
    return f"({y}) " + " ".join(intervali(ovire))
```

Ali, odvisno od osebnega sloga

[15]:
```
def zapisi_vrstico(y, ovire):
    return f"({y}) {' '.join(intervali(ovire))}"
```

1.2 Dodatna naloga

Napiši funkcijo `zapisi(ovire)`, ki prejme seznam ovir in vrne niz, ki vsebuje opis ovir v novi obliki. Za razliko od kaosa, ki ga dobimo od MOL, pa mora biti zapis urejen: vrstice se pojavljajo le enkrat in v pravem vrstnem redu, pa tudi ovire morajo biti urejene od leve proti desni.

Klic

```
zapisi([(5, 6, 4),
        (90, 100, 13), (5, 8, 13), (9, 11, 13),
        (9, 11, 5), (19, 20, 5), (30, 34, 5),
        (9, 11, 4),
        (22, 25, 13), (17, 19, 13)])
```

vrne niz

```
(4) 5-- 9---
(5) 9--- 19-- 30-----
(13) 5---- 9--- 17--- 22---- 90-----------
```

Spet: niz mora biti v točno takšni obliki, brez odvečnih ali manjkajočih presledkov.

1.2.1 Rešitev

Glede na to, da še ne poznamo slovarjev in da `y` ne more biti zelo velik, bi bila najbolj prikladna - in pravzaprav nič slabša od rešitve s slovarji, takšna rešitev:

[16]:
```
def zapisi(ovire):
    vrstice = []
    for x0, x1, y in ovire:
        while len(vrstice) <= y:
            vrstice.append([])
        vrstice[y].append((x0, x1))

    zemljevid = ""
    for y, xs in enumerate(vrstice):
        if xs:
            zemljevid += zapisi_vrstico(y, sorted(xs)) + "\n"
    return zemljevid
```

`vrstice` so seznam seznamov: v `vrstice[3]`, recimo, so začetki in konci vseh ovir v tretji vrstici.

V prvi zanki gremo čez vse ovire. V zanki `while` poskrbimo, da ima `vrstice` dovolj elementov: če se ovira nahaja v vrstici, katere številka je večja od števila doslej vzpostavljenih vrstic, dodaja nove prazne sezname. Nato dodamo vsako oviro v ustrezno vrstico.

V drugi zanki pišemo ovire v `zemljevid`, ki ga bomo vrnili kot rezultat. Gremo čez elemente `vrstice` in če je v vrstici kaj ovir, pokličemo `zapisi_vrstico`, da jih zapiše. Seznam ovir, `xs`, predtem uredimo.

Rešitev s slovarji, bi bila malenkost krajša, ideja pa je podobna.

[17]:
```
def zapisi(ovire):
    kupcki = defaultdict(list)
    for x0, x1, y in ovire:
        kupcki[y].append((x0, x1))

    zemljevid = ""
    for y, kupcek in sorted(kupcki.items()):
        zemljevid += zapisi_vrstico(y, sorted(kupcek)) + "\n"
    return zemljevid
```

Gre krajše? Gre v eni vrstici?

Vedno. Vendar tule nima smisla. Prvi del je najbolj jasen takšen, kot je. Drugega pa bi dejansko skrajšal, če bi šlo zares.

[18]:
```
def zapisi(ovire):
    kupcki = defaultdict(list)
    for x0, x1, y in ovire:
        kupcki[y].append((x0, x1))

    return "\n".join(zapisi_vrstico(y, sorted(kupcek))
                     for y, kupcek in sorted(kupcki.items()))
```

V resnici pa gre za problem urejanja in grupiranja teh, urejenih stvari. Zato bi kdo kak ekstremist funkcijskega programiranja v Pythonu nemara napisal

[19]:
```
from operator import itemgetter
from itertools import groupby

def zapisi(ovire):
    return "\n".join(zapisi_vrstico(y, sorted(x[:2] for x in group))
                     for y, group in groupby(sorted(ovire, key=itemgetter(2)),␣
 ↪itemgetter(2)))
```

Deluje, ni pa berljivo. Python je lep jezik, ni pa vsak jezik lep za vsak slog programiranja. Če se kdo potrudi to prebrati, bo videl, da mora brati nazaj - najprej se zgodi (drugi) `sorted`, nato `groupby`, potem `zapisi_vrstico`, katere rezultati se združijo z `join`. To bi se bralo veliko lepše v jezikih, v katerih se to zapiše naprej, recimo v Kotlinu ali Javascriptu.