resitev

January 28, 2024

0.1 Obvezna naloga

Mednarodni standard ISO-1234 za zapis pozicij ovir na kolesarskih poteh zapiše ovire takole:

```
004:   5-6      9-11
005:   9-11    19-20    30-34
013:   5-8      9-11    17-19    22-25    90-100
```

- Vsaka vrstica datoteke se nanaša na eno "vrstico" kolesarske poti.
- Prva številka v vrstici je koordinata y, zapisana na 3 mesta, z vodilnimi ničlami. Koordinata 42 je torej zapisana kot `042`.
- Sledi dvopičje.
- Sledijo pari; začetni in končni x sta ločena z `-`. Zapisana sta na 4 mesta, pri čemer je začetna koordinata poravnana desno, končna pa levo.

Napiši naslednje funkcije:

- `zapis_meje(x0, x1)` prejme koordinato levega in desnega konca ovire ter vrne niz s tema koordinatama, zapisanama na 4 mesta, pri čemer je prva poravnana desno, druga levo, vmes je `-`.

  Klic zapis_meje(94, 152) vrne `"  94-152 "`. Ne spreglej presledka na koncu. Če testi napišejo, da `94-152 != 94-152`, najbrž manjka presledek na koncu. Ali pa je preveč.

- `zapis_vrstice(y, xs)` prejme številko vrstice in seznam parov začetkov in koncev ovir v njej. Vrne niz z vrstico v gornji obliki. Klic `zapis_vrstice(5, [(9, 11), (19, 20), (30, 34)])` vrne `"005:   9-11    19-20    30-34  "`

- `zapisi_ovire(ime_datoteke, ovire)` prejme ovire v obliki slovarja: ključi so številke vrstic, pripadajoče vrednosti pa seznami parov začetkov in koncev ovir. Prvi argument funkcije pa je ime datoteke: vanjo mora zapisati ovire skladno standardom.

  Klic

  ```
  zapisi_ovire("ovire.txt",
               {4: [(5, 6), (9, 11)],
                5: [(9, 11), (19, 20), (30, 34)],
                13: [(5, 8), (9, 11), (17, 19), (22, 25), (90, 100)]}
  ```

  zapiše v ovire.txt ovire v zgoraj zapisani obliki.

  Testi bodo funkciji vsakič podali drugačno ime datoteke (konkretno: trenutni čas). Če se test izvede uspešno, bodo datoteko pobrisali, sicer jo bodo pustili (in se bodo začele nabirati), tako da lahko vidite, kaj je funkcija (napačno) zapisala vanjo.

0.1.1 Rešitev

zapis_meje

[ ]:
```
def zapis_meje(x0, x1):
    return f"{x0:>4}-{x1:<4}"
```

To je vse. Ne lomite ga z lastnoročnimi poravnavanji v slogu

[1]:
```
def zapis_meje(x0, x1):
    x0s = str(x0)
    x1s = str(x1)
    return " " * (4 - len(x0s)) + x0s + "-" + x1s + " " * (4 - len(x1s))
```

To sicer deluje, vendar s tem ne boste prišli daleč. Že za tako preprost primer - dve številki na štiri mesta, ena poravnana levo, druga desno - je kar zapleteno. Pri čem bolj zapletenem pa na ta način nimamo šans.

zapis_vrstice Najprej `y` na tri mesta (`{y:03}`, nato z `"".join(...)` zlepimo skupaj vse, kar nam za meje iz seznama vrne prejšnja funkcija.

[2]:
```
def zapis_vrstice(y, xs):
    return f"{y:03}:" + "".join(zapis_meje(x0, x1) for x0, x1 in xs)
```

zapisi_ovire Odpremo datoteko in vanjo pišemo vrstice, ki jih vrača prejšnja funkcija. Na konec vsake prilepimo še znak za novo vrsto, `\n`.

[3]:
```
def zapisi_ovire(ime_dat, ovire):
    f = open(ime_dat, "wt")
    for y, xs in ovire.items():
        f.write(zapis_vrstice(y, xs) + "\n")
```

0.2 Dodatna naloga

Oddelek za gospodarstvo in motoriziran promet je ponosen na svoje inovacije. (Temu na primer, da so kolesarjem na Slovenski cesti omogočili voziti slalom med avtobusi, z vso resnostjo pravijo "inovacija v svetovnem merilu". Kar dejansko je.)

Tokrat so največji prometni strokovnjaki na svetu, torej prometni strokovnjaki MOL, staknili glave in sestavili nov standard za zapis ovir, tandard MOL-666-B. Ta zapiše gornje ovire takole:

```
4 2
5
6
9
11
5 3
9
11
19
```

```
20
30
34
13 5
5
8
9
11
17
19
22
25
90
100
```

V prvi vrstici je številka vrstice z ovirami (4) in število ovir v tej vrstici (2). Naslednje štiri vrstice datoteke vsebujejo začetke in konce teh ovir (5 in 6 ter 9 in 11). Nato sledi nova številka vrstice z ovirami (13) in število ovir v njej (5). Sledi 10 vrstic datoteke z začetki in konci teh ovir…

Kot navaja MOL, je novi standard namenjen še večjemu spodbujanju kolesarjenja in trajnostne mobilnosti, hkrati pa tudi hitrejšemu zelenemu prehodu v brezogljično družbo.

Napiši funkcijo `preberi_ovire(ime_datoteke)`, ki prebere ovire iz takšne datoteke v slovar.

Testno datoteko bodo sestavili testi ob prvem zagonu. Njeno ime bo "ovire.txt", vendar naj bo funkcija napisana tako, da zna prebrati datoteko s poljubnim podanim imenom.

0.2.1 Rešitev

No, ta je bolj zanimiva.

[4]:
```
def preberi_ovire(ime_dat):
    ovire = {}
    f = open(ime_dat)
    for ys in f:
        y, nx = ys.split()
        xs = []
        for _ in range(int(nx)):
            xs.append((int(f.readline()), int(f.readline())))
        ovire[int(y)] = xs
    return ovire
```

Odpremo datoteko; predstavljala jo bo spremenljivka `f`. `for ys in open(...)` ne bo delovalo, ker bomo datoteko brali na dveh mestih v funkciji.

Zanka `for ys in f` bo vedno prebrala vrstico z dvema številkama: število vrstice ovir in število ovir. Podatka razbijemo s `split` in shranimo v `y` in `nx`.

`xs` bo seznam ovir v vrstici `y`. Notranjo zanko izvedemo tolikokrat, kolikor je ovir (`for _ in range(int(nx))`). Z `f.readline()` preberemo številko in jo z `int` pretvorimo v število. To

naredimo dvakrat in iz tega sestavimo terko, torej `(int(f.readline()), int(f.readline()))`. Z `append` dodamo terko v seznam.

Notranja zanka bo prebrala začetke in konce vseh ovir. Ko se konča, bo zunanja zanka prebrala naslednjo številko vrstice in število ovir v njej…

Pa še krajša različica:

[5]:
```
def preberi_ovire(ime_dat):
    ovire = {}
    f = open(ime_dat)
    for ys in f:
        y, nx = ys.split()
        ovire[int(y)] = [(int(f.readline()), int(f.readline())) for _ in␣
 ↪range(int(nx))]
    return ovire
```