resitev

January 28, 2024

Tokratna naloga bo zelo kratka. V vsaki funkciji bo potrebno napisati samo eno vrstico. Točneje, tokratna naloga bo krajša, ko bi si želeli: v vsaki funkciji boste *smeli* napisati samo eno vrstico.

Spet se vračamo k isti nalogi, ki smo jo reševali že dvakrat: prvič smo jo imenovali Šikaniranje, drugič pa Funkcijske ovire.

Tokrat jih bomo reševali s pomočjo izpeljanih seznamov. Če vam kaj pomaga (najbrž vam vsaj malo) imate poleg testov že napisane funkcije, ki rešijo naloge … le predolge so. :)

0.1 Obvezna naloga

Napišite funkcije

- `stevilo_ovir(ovire)`,
- `dolzina_ovir(ovire)`,
- `sirina(ovire)`,
- `dodaj_vrstico(bloki, y)`
- `globina(ovire, x)`
- `senca(ovire)`

tako, da bodo vsebovale samo stavek `return` in … kar je pač potrebno, da izračunate, kar morajo izračunati.

0.1.1 Rešitev

Število ovir je že napisano v vrstici. Daljše skoraj ne gre. :)

[1]:
```
def stevilo_ovir(ovire):
    return len(ovire)
```

`dolzina_ovir(ovire)` mora vrniti vsoto dolžin vseh ovir. Dolžina ovire `(x0, x1, y)` je `x1 - x0 + 1`, vsota tega je `sum(x1 - x0 + 1 ... za vsako oviro`.

[2]:
```
def dolzina_ovir(ovire):
    return sum(x1 - x0 + 1 for x0, x1, _ in ovire)
```

Ker `y` ne potrebujemo, tretjo vrednost iz trojke poimenujemo kar `_`.

`sirina(ovire)` je enaka največjemu (po angleško: maksimalnemu) `x1`.

[3]:
```
def sirina(ovire):
    return max(x1 for _, x1, _ in ovire)
```

`dodaj_vrstico(ovire, y)` gre čez seznam parov in jih zlaga v seznam trojk, ki poleg `x1` in `x0` vsebujeta še `y`.

[5]:
```
def dodaj_vrstico(bloki, y):
    return [(x0, x1, y) for x0, x1 in bloki]
```

Če hočemo poudariti, da je `y` dodan k tistemu, kar smo imeli prej, pa

[6]:
```
def dodaj_vrstico(bloki, y):
    return [blok + (y, ) for blok in bloki]
```

Ne spreglejte: ne `blok + (y)`, temveč `blok + (y, )`. `(y)` bi bila samo številka, ne terka.

`globina(ovire, x)` mora vrniti najmanjši `y` med vsemi `x0, x1, y`, za katere velja, da je `x` med `x0` in `x1`, torej `x0 <= x <= x1`.

To bi lahko bilo tole:

[7]:
```
def globina(ovire, x):
    return min(y for x0, x1, y in ovire if x0 <= x <= x1)
```

vendar ne deluje v primeru, da v stolpcu ni nobene ovire. Tedaj `(y for x0, x1, y in ovire if x0 <= x <= x1)` ne zgenerira ničesar in `max` ne ve, kaj vrniti. Pogledši dokumentacijo funkcije `min` izvemo, da ji lahko podamo še argument `default`, s katero predpišemo vrednost, ki naj jo vrne v takšnem primeru. V naši nalogi želimo, da tedaj vrne `None`, torej

[8]:
```
def globina(ovire, x):
    return min((y for x0, x1, y in ovire if x0 <= x <= x1), default=None)
```

`senca(ovire)` mora sestaviti seznam s toliko elementi, kolikor je stolpcev in vsak vsebuje `True`, če `globina` za ta stolpec vrne `None` in `False`, če ne.

[9]:
```
def senca(ovire):
    return [globina(ovire, x) is None for x in range(1, sirina(ovire) + 1)]
```

0.2 Dodatna naloga

Napišite funkcijo `indeksi(s, subs)`, ki prejme niza `s` in `subs` ter vrne seznam indeksov znotraj `s`, na katerih se pojavi `subs`. Klic `indeksi("pepelka peče prepelice", "pe")` vrne `[0, 2, 8, 16]`, saj se `pe` pojavi na indeksih 0, 2, 8 in 16. Tudi ta funkcija sme seveda obsegati samo `return`.

Potem napišite v eni vrstici funkcijo

- `pretvori_vrstico(vrstica)`.

0.2.1 Rešitev

Funkcijo `indeksi` so nekateri študenti zelo zapletli, ker so si poskušali pomagati z zanko, v kateri bi klicali `index`. To je zelo težko. Veliko preprosteje je pomisliti drugače: kaj hoče funkcija? Kaj moramo vrniti? Vse tiste indekse, za katere velja, da se na njih začenja podniz `subs`. Kako torej preverimo, ali se na `i`-tem mestu `s`-ja začenja `subs`? `s[i:i + len(subs)] ==`

`subs`. Lahko pa uporabimo tudi metodo `startswith`, ki pove, ali se niz začne s podanim nizom: `s[i:].startswith(subs)`.

Funkcija `indeksi` je torej:

[10]:
```
def indeksi(s, subs):
    return [i for i in range(len(s)) if s[i:].startswith(subs)]
```

Zdaj pa pretvorimo vrstico. Zanima nas, kje so začetki ovir. Ovire za začnejo tam, kjer najdemo `.#`. Da bomo našli tudi ovire, ki bi bile čisto na začetku, na začetek prištejemo piko. Ovire se torej začenjajo na `indeksi("." + s, ".#")`. Vendar bodo ti indeksi žal za 1 premajhni: če imamo `s = "#"`, se ovira začne na `1` (ker v teh nalogah pač štejemo od 1). Klic `indeksi("." + s, ".#")`, ali, konkretneje, `indeksi(".#", ".#")` bo vrnil `[0]`. Nič hudega: indeksom bi sicer lahko prištevali 1 kdaj kasneje, a preprosteje, če nizu `s` namesto ene same pike prištejemo dve, pa se bodo vsi indeksi povečali za `1`. Stolpci, v katerih se začenjajo ovire, so torej `indeksi(".." + s, ".#")`.

Pa konci? Zdaj iščemo `"#."`. Da pravilno zaznamo tudi oviro, ki bi bila čisto na koncu niza, nizu prištejemo še `"."`. Da povečamo indekse za `1`, dodamo piko na začetku (namesto pike bi lahko na začetek dodali tudi karkoli drugega). Konci ovir so torej na `indeksi("." + s  + ".", subs)`.

Funkcija mora vrniti pare začetkov in koncev.

[11]:
```
def pretvori_vrstico(vrstica):
    return list(zip(indeksi(".." + vrstica, ".#"), indeksi("." + vrstica + ".",␣
 ↪"#.")))
```

0.3 Čisto dodatna naloga (ki ne šteje nikamor)

Še preostali dve:

- `pretvori_zemljevid(zemljevid)` (tule boste verjetno potrebovali `reduce` in `add`),
- `naj_stolpec(ovire)`.

0.3.1 Rešitev

Ni tako težko.

Generator `(dodaj_vrstico(pretvori_vrstico(vrstica), y) for y, vrstica in enumerate(zemljevid, start=1))` gre čez vrstice; za vsako poišče začetke in konce vrstic ter v pare doda številke vrstic, `y`. Problem je, da tako dobimo sezname seznamov ovir v vrsticah. Vse "podsezname" je potrebno sešteti v en sam seznam. To lahko storimo s funkcijo `sum`, če ji podamo začetni element. Funkcija `sum` privzeto začne prištevati k `0`, mi želimo, da prišteva k praznemu seznamu.

[12]:
```
def pretvori_zemljevid(zemljevid):
    return sum((dodaj_vrstico(pretvori_vrstico(vrstica), y)
                for y, vrstica in enumerate(zemljevid, start=1)), start=[])
```

Funkcija `naj_stolpec` je pa bolj zoprna. Iščemo nekakšen `max(((x, globina(ovire, x)) for x in range(1, sirina(ovire) + 1)`, pri čemer pa moramo pare primerjati po drugem elementu.

Ena možnost je, da uporabimo `key`, ki mu podamo lambda-funkcijo. Ta prejema elemente, katerih maksimum iščemo in za vsakega vrne ključ, glede na katerega ga primerjamo. V našem primeru imamo pare, ki jih primerjamo po drugem elementu, torej

[16]:
```
def naj_stolpec(ovire):
    return max(((x, globina(ovire, x)) for x in range(1, sirina(ovire) + 1)),
               key=lambda x: x[1])
```

Če lambd ne poznamo (vsaj letos smo se o njih učili le na dodatnih predavanjih), se znajdemo tako, da zamenjamo vrstni red elementov v paru: namesto `(x, globina(ovire, x))` sestavljamo pare `(globina(ovire, x), x)`. Funkcija `max` jih bo primerjala po prvem elementu in tako izbrala pravi največji element – a funkcija mora vrniti par, v katerem je na prvem mestu `x`, ne globina. Nič hudega, ga pač obrnemo pred vračanjem, tako da na konec dodamo `[::-1]`.

[17]:
```
def naj_stolpec(ovire):
    return max((globina(ovire, x), x)
               for x in range(sirina(ovire) + 1) if globina(ovire, x) is not␣
 ↪None)[::-1]
```

Žal nas zafrknejo stolpci brez ovir. Tam `globina` vrne `None`. Funkcija mora v tem primeru vrniti ta stolpec … a namesto tega vrne napako, da poskušamo primerjati `None` in število.

Če delamo z lambdami, to rešimo tako, da funkcija, ki računa ključ, v primeru, da je drugi element enak `None` vrne 1000. Recimo, da tako daleč ne bo nobene ovire. Če nas to moti, pa lahko vrača tudi neskončno, `float("inf")`.

[18]:
```
def naj_stolpec(ovire):
    return max(((x, globina(ovire, x)) for x in range(1, sirina(ovire) + 1)),
               key=lambda x: float("inf") if x[1] is None else x[1])
```

Rešitev brez lambde pa je zabavna. Takole storimo.

[15]:
```
def naj_stolpec(ovire):
    return ([(x, None)
             for x in range(1, sirina(ovire) + 1) if globina(ovire, x) is None]
            + [max((globina(ovire, x), x)
                   for x in range(sirina(ovire) + 1) if globina(ovire, x) is␣
 ↪not None)[::-1]]
           )[0]
```

Seštevamo dva seznama. Prvi seznam vsebuje pare `(x, None)` za vse tiste `x`, kjer ni ovir. Drugi seznam pa vsebuje samo en element, namreč maksimalni element, ki ga dobimo natančno tako, kot smo opisali zgoraj, vendar upoštevamo le tiste `x`, pri katerih ovire obstajajo.

Oba seznama torej seštejemo in nato vrnemo prvi element seznama, ki ga dobimo kot vsoto. Če obstaja kak stolpec brez ovir, bodo te v prvem seznamu in vrnili bomo prvo med njimi. Če stolpcev brez ovir ni, pa bo prvi seznam prazen in vrnili bomo prvi (in edini) element drugega. :)