

## Ocena 6

Napiši razred `Kolesar`.

- Konstruktor `__init__` kot argument prejme `zemljevid`, s kakršnimi smo delali v prejšnji nalogi. `Zemljevid` je podan kot seznam nizov, ki predstavljajo vrstice `zemljevida`. `Kolesar` je v začetku na koordinatah 0, 0.
- `pojdi(smer)` prejme smer "<", ">", "^" ali "v". Metoda premakne kolesarja za en kvadratale v podano smer, če ta kvadratale ni izven `zemljevida`. Če bi to pripeljalo kolesarja ven iz `zemljevida`, pa ne spremeni koordinat, temveč obleži mrtev in se poslej ne odziva več na nadaljnje ukaze `premik` ali `pojdi`.
- `prevozi(pot)` prejme niz s potjo, recimo, ">vv4>105v>>>^^3^" in kolesarja pelje po tej poti -- v tem primeru desno, dol, dol, štirikrat desno, 105-krat dol, desno, desno, desno, gor, gor, trikrat gor. Metoda `prevozi` mora klicati metodo `pojdi` za vsak kvadratale posebej. Tako poskrbi, da bi kolesar, ki bi ga nek korak pripeljal ven iz `zemljevida`, umrl na ustreznem mestu.
- `lokacija()` vrne trenutne koordinate kolesarja v obliki terke (x, y).
- `razdalja()` vrne razdaljo (v številu) kvadratkov, ki jih je doslej (oz. do smrti) prevozil kolesar.

## Rešitev

Razred bo imel naslednje atribute:

- `zemljevid` (`zemljevid`),
- koordinati (x, y),
- dolžina prevožene poti (`prevozeno`)
- indikator (`False` ali `True`), ki bo povdal, ali je kolesar še živ.

Konstruktor tako imamo. Metodo `pojdi` prepišemo iz prejšnje domače naloge (le s tem dodatkom, da kolesarja izven `zemljevida` označimo za neživo naravo in ga ne premikamo več), poleg tega pa še štejemo korake. Metodo `prevozi` pa samo prepišemo. Dodamo še metodi `lokacija` in `razdalja`, pa je.

```
class Kolesar:
    def __init__(self, zemljevid):
        self.zemljevid = zemljevid
        self.x = self.y = 0
        self.prevozeno = 0
        self.ziv = True

    def pojdi(self, smer):
        if self.ziv:
            nx, ny = {"<": (self.x - 1, self.y), ">": (self.x + 1, self.y),
```

```

        "^": (self.x, self.y - 1), "v": (self.x, self.y + 1)}[smer]
self.ziv = 0 <= nx < len(self.zemljevid[0]) and 0 <= ny < len(self.zemljevid)
if self.ziv:
    self.prevozeno += 1
    self.x, self.y = nx, ny

def prevozi(self, pot):
    stevilka = ""
    for c in pot:
        if c in "<>^v":
            for _ in range(int(stevilka or 1)):
                self.pojdi(c)
            stevilka = ""
        else:
            stevilka += c

def lokacija(self):
    return self.x, self.y

def razdalja(self):
    return self.prevozeno

```

## Ocena 7

Iz razreda `Kolesar` izpelji razred `Zbiralec`. Razlika v primerjavi s `Kolesar` bo v tem, da zemljevida ne bo uporabljal le zato, da preverja, ali je še živ, temveč bo nabiral "značke", to je, mestne kolesarske znamenitosti (kakor jih poznamo iz prejšnje naloge).

Če se značka nahaja na polju, na katerem kolesar začne svojo pot, je ne pobere (razen, seveda, če kasneje pride spet na to polje).

Metoda `prevozi` pobira tudi vse značke na kvadratih, ki so na poti. `prevozi("5>")`, na primer, pobere vse morebitne značke na petih kvadratih desno od trenutne pozicije kolesarja.

- Konstruktor `__init__` prejme argumente `x`, `y`, `zemljevid`. `zemljevid` ima enak pomen kot prej, `x` in `y` pa sta začetni koordinati kolesarja.
- `pojdi(smer)` dopolni tako, da bo pobirala značke in na primeren način klicala podedovano metodo.
- `znacke()` vrne množico vseh črk, na kakršne je naletel kolesar. (Pike so nezanimive, zato jih v množici ni.)
- `naj_znacke()` vrne množico s črkami, ki jih je največkrat obiskal. Če je kolesar na svoji poti naletel na dva a-ja, pet r-jev, tri b-je, tri d-je in pet c-jev, metoda `naj_znacke()` vrne `{"r", "c"}`.

- `trofeje()`: predstavljajmo si, da značke uredimo v lestvico po pogostosti. Zanima nas, katere značke zasedajo prva tri mesta. `trofeje()` torej vrne seznam parov (crka, pogostost) za tri najbolj pogoste značke. Seznam mora biti urejen po padajoči pogostosti, znotraj enako pogostih pa po abecedi. Če si več značk deli isto mesto, je seznam ustrezno daljši. Konkretno, v gornjem primeru, ko je kolesar na svoji poti naletel na dva a-ja, pet r-jev, tri b-je, tri d-je in pet c-jev, funkcija `trofeje()` vrne `[("c", 5), ("r", 5), ("b", 3), ("d", 3)]`: seznam ima štiri elemente, ker si tretje mesto delita dve črki. Če bi kolesar, recimo, obiskal vsakega od gornjih znakov petkrat, poleg tega pa še enkrat "l", bi funkcija vrnila `[("b", 5), ("c", 5), ("d", 5), ("r", 5)]`, saj si štirje delijo prvo mesto. (Razmišljajte o tem, kdo dobi medalje, če si več kot trije tekmovalci delijo prva tri mesta.)

Zbiralec **naj ne definira** svoje metode `prevozi`, ker za to nima prav nobenega razloga.

## Rešitev

Konstruktor dopolnimo tako, da po klicu podedovanega konstruktorja nastavi še želeni koordinati `x` in `y`. Poleg tega dodamo atribut `nabor`, ki bo slovar katerega ključi bodo imena značk, pripadajoče vrednosti pa bo povedale, kolikokrat je naletel na dotično značko.

Metoda `pojdi` pokliče podedovano metodo nato pa še pobere značko na trenutnem polju - če je kolesar še živ in če je na tem polju kakšna značka.

Ostalo je le programerska spretnost.

Metoda `znacke` vrne množico ključev, torej `set(self.nabor)`. (Nobenih nepotrebnih zank!)

`naj_znacke` ugotovi, kakšna je največja vrednost v slovarju in potem vrne množico vseh ključev, ki jim pripada ta vrednost. Ko pokličemo `max`, podamo še argument `default=0`, da ga smemo klicati tudi s praznim slovarjem. V tem primeru bo `naj` enak 0 - lahko pa bi podali tudi poljubno drugo privzeto vrednost, saj se zanka `for k, v in self.nabor.items()` v primeru, da je slovar prazen, tako ali tako sploh ne bo izvedla in vrnjena množica bo prazna.

Zadnja, `trofeje`, pa je zabavna. Ena preprostejših rešitev je tale: pogledamo vse številke, ki se pojavijo kot vrednosti v slovarju `nabor`, torej `set(self.nabor.values())`. Če je kolesar neko značko osvojil 3x, pa eno 5x, pa eno 2x, pa eno 3x, bo ta množica vsebovala `{3, 5, 2}`. Uredimo jo padajoče, tako da dobimo seznam `[5, 3, 2]`. Zdaj gremo po elementih seznama: v množico trofej najprej dodamo vse značke, ki jih je dobil petkra. Če jih je manj kot 3, dodamo vse, ki jih je dobil trikrat. Če jih je manj kot tri, dodamo vse, ki jih je dobil dvakrat...

```
from collections import defaultdict
```

```

class Zbiralec(Kolesar):
    def __init__(self, x, y, zemljevid):
        super().__init__(zemljevid)
        self.x, self.y = x, y
        self.nabor = defaultdict(int)

    def pojdi(self, smer):
        super().pojdi(smer)
        if self.ziv:
            znacka = self.zemljevid[self.y][self.x]
            if znacka != ".":
                self.nabor[znacka] += 1

    def znacke(self):
        return set(self.nabor)

    def naj_znacke(self):
        naj = max(self.nabor.values(), default=0)
        return {k for k, v in self.nabor.items() if v == naj}

    def trofeje(self):
        trof = []
        for x in sorted(set(self.nabor.values()), reverse=True):
            if len(trof) >= 3:
                break
            trof += sorted((k, v) for k, v in self.nabor.items() if v == x)
        return trof

```

## Ocena 8

Opis te naloge je strašljivo dolg, vendar je tako le zaradi primerov. V resnici zahtevata razred `Drsalec` enajst vrstic kode (brez trikov; težko ga napišete *daljše*), "razred" Sledi pa eno samo, če boste spretni.

Na snegu se dogaja tole: če je sneg na kvadratu svež, kolesar normalno pelje. Če je kakšen kolesar že prevozil določeno polje, pa naslednjega odnese v isto smer kot zadnjega. Isto se zgodi na naslednjem in še na naslednjem polju. Šele po treh poljih sledenja prejšnjemu kolesarju (oziroma kolesarjem) se kolesar ustavi in potem nadaljuje pot, kot si jo je zamislil sam. (Seveda se lahko zgodi, da sledenje traja manj kot tri polja, če že prej pride na polje, ki še ni zvoženo.)

Recimo, da je Ana začela na polju (1, 2), Berta na polju (3, 0) in Cilka na (7, 3).

Ana se opogumi in prevozi pot  $7 > 2v7 <$ . Za seboj pusti takšne sledi. (A označuje končno Anino polje. Sledi tam ni.)

...B.....

```

.....
.>>>>>>>v...
.....Cv...
.A<<<<<<<...
.....
.....

```

Zdaj se Berta odloči prevoziti 3v. Odnoslo jo bo takole.

```

...B.....
...B.....
.>>BBBB>v...
.....BCv...
.A<<<<<<<...
.....
.....

```

Ko je naletela na Bertine sledi, jo je tri kvadratke nosilo po njeni sledi, potem pa je nadaljevala po svoji poti (dol). Po Bertini vožnji so sledi v snegu takšne:

```

...v.....
...v.....
.>>>>>>v>v...
.....BCv...
.A<<<<<<<...
.....
.....

```

Recimo, da gre Berta zdaj še «. Sledi so

```

...v.....
...v.....
.>>>>>>v>v...
...B<<Cv...
.A<<<<<<<...
.....
.....

```

Potem pride Cilka na (7, 1) in hoče iti 2^>. Kruta usoda, ki se bo poigrala z njo, ji bo namenila takšno pot.

```

...v.....
...v.....
.>>>>>>vCC...
...B<<CC...
.A<<<<<<CC..
.....
.....

```

- Po prvem  $\wedge$  jo odnese po Anini sledi  $>vv;$ ;znajde se v spodnjem desnem kvadratu Anine poti.
- Potem gre  $\wedge$ , kot je načrtovala, in, ojoj, vpade ravno na polje, na katerem je ravnokar bila. Zato jo odnese nazaj  $v$ , na polje, ki ga je pravkar zapustila v smeri  $\wedge$ . Odnese jo torej gor in nato spet  $v$ . S tem se to drsanje ustavi.
- Potem gre, po načrtu,  $>$ . Za sabo pusti tako stanje.

```
...v.....
...v.....
.>>>>>v>>...
....B<<^v...
.A<<<<<<>C..
.....
.....
```

Potem se pojavi Dani in sicer na polju (9, 3) (to je polje nad C) in hoče iti  $<v$ .

- Naredi korak  $<$ . Odnese jo  $v>$  (torej le dvakrat, saj polja (9, 4) Cilka še ni zapustila, torej tam ni sledov).
- Potem gre  $v$ .

```
...v.....
...v.....
.>>>>>v>>...
....B<<^v<...
.A<<<<<<>v..
.....D..
.....
```

Potem se Cilka, ki je na gornjem zemljevidu sicer nismo več narisali (da se vidi puščica, ki jo je naredila Dani; Cilka je torej takoj nad Dani), odloči iti  $\wedge$ . Plan se izjalovi, saj se znajde na sledovih Dani, gre  $<\wedge>$  in konča natančno tam, kjer je začela.

Napiši razreda **Sledi** in **Drsalec**.

Razred **Drsalec** je izpeljan iz razreda **Zbiralec** in ima metodo **pojdi** spremenjeno tako, da po vsakem normalnem premiku izvede še tri korake drsenja (ali manj, če ni sledi oz. če jih zmanjka prej kot v treh korakih). Poleg tega ima konstruktor dodaten argument, ki je pojasnjen spodaj.

Razred **Sledi** beleži sledi. Kako to počne, je tvoja stvar. Testi se bodo začeli takole:

```
zemljevid = [". " * 12] * 7
sledi = Sledi()
ana = Drsalec(1, 2, zemljevid, sledi)
berta = Drsalec(3, 0, zemljevid, sledi)
cilka = Drsalec(7, 3, zemljevid, sledi)
```

Test najprej pripravi nek `zemljevid` (v tem primeru brez znamenitosti) in objekt `sledi`; kaj vsebuje, je tvoja stvar. Vsi drsalci dobijo isti `zemljevid` in, predvsem, iste `sledi`. Tvoja funkcija `pojdi` mora v objekt `sledi` shranjevati `sledi` in jih iz njega razbirati. To počni, kakor želiš. (Namig: razred `Sledi` je lahko res trivialen. Pri meni je njegova definicija dolga 12 znakov.)

### Rešitev

Rešitev je res bistveno bistveno krajša od opisa.

```
Sledi = dict
```

```
class Drsalec(Zbiralec):
    def __init__(self, x, y, zemljevid, sledi):
        super().__init__(x, y, zemljevid)
        self.sledi = sledi

    def pojdi(self, smer):
        self.sledi[self.x, self.y] = smer
        super().pojdi(smer)
        for _ in range(3):
            if (self.x, self.y) not in self.sledi:
                break
            super().pojdi(self.sledi[self.x, self.y])
```

`Sledi` bomo shranjevali kar v slovar. Torej bo "razred" `Sledi` pravzaprav isto kot `dict`.

Metoda `pojdi` na trenutno polje zapiše, v katero smer se pelje kolesar na tem polju. Nato pokliče podedovani `pojdi`, da dejansko premakne kolesarja. Potem pa le še trikrat sledi sledem - če te obstajajo. Če jih zmanjka, pa pač prekine zanko.