

Rešitev

Najprej samo "povzetek": vse zbrane rešitve nalog (in njihove teme, da boste videli, za kaj je šlo na izpitu). Besedila nalog, alternativne rešitve in podrobnejše razlage pa so spodaj.

```
from collections import defaultdict
```

1. Zanke in pogoji

```
def skupni_odsek(pot1, pot2):
    naj = 0
    dolzina = 0
    for a, b in zip(pot1, pot2):
        if a == b:
            dolzina += 1
            if dolzina > naj:
                naj = dolzina
        else:
            dolzina = 0
    return naj
```

2. Slovarji in množice

```
from collections import defaultdict
```

```
def poraba(pot, zemljevid):
    por = defaultdict(int)
    for a, b in zip(pot, pot[1:]):
        for vescina in zemljevid[(a, b)]:
            por[vescina] += 1
    return por

def vecji_frajer(pot1, pot2, zemljevid):
    poraba1 = poraba(pot1, zemljevid)
    poraba2 = poraba(pot2, zemljevid)
    for vescina, uporab in poraba2.items():
        if poraba1[vescina] < uporab:
            return False
    return poraba1 != poraba2
```

3. Datoteke in nizi

```

def preberi_zemljevid_vescin(ime_datoteke):
    zemljevid = defaultdict(set)
    for vrstica in open(ime_datoteke):
        vescina, povezave = vrstica.split(":")
        if povezave.strip():
            for povezava in povezave.split(","):
                a, b = povezava.strip().split("-")
                zemljevid[a, b].add(vescina)
    return zemljevid

```

4. Rekurzivne funkcije

```

def koristne_vescine(odkod, zemljevid):
    koristne = set()
    for (krizisce, naprej), vescine in zemljevid.items():
        if naprej > krizisce == odkod:
            koristne |= vescine | koristne_vescine(naprej, zemljevid)
    return koristne

```

5. Objektno programiranje

```

class Frajer(Kolesar):
    def __init__(self, lokacija, zemljevid, vescine):
        super().__init__(lokacija, zemljevid, vescine)
        self._zivljenja = 3
        self._uporabljene = set()

    def premik(self, kam):
        zacetek = self.lokacija()
        super().premik(kam)
        if self.lokacija() is not None:
            self._uporabljene |= self.zemljevid[zacetek, kam]
        else:
            self._zivljenja = max(0, self._zivljenja - 1)
            if self._zivljenja > 0:
                self._lokacija = zacetek

    def zivljenja(self):
        return self._zivljenja

    def uporabljene(self):
        return self._uporabljene

```

1. Skupni odsek

Dva kolesarja sta šla istočasno na pot. Za vsako povezavo sta potrebovala enako časa. Napiši funkcijo `skupni_odsek(pot1, pot2)`, ki vrne dolžino najdaljšega zaporedja, na katerih sta bila skupaj. Poti sta podani z nizom zaporednih črk, ki označujejo križišča.

Klic

```
skupni_odsek("ASAIMWGVIEHUTUMVIVHIV",  
             "OIMIMAWAREMPBTUMVIBTIOWE")
```

vrne 5, saj je najdaljše skupno zaporedje TUMVI.

Rešitev

Tole je naloga iz pogojev in zank.

Vzporedno gremo čez obe poti. Če sta znaka enaka, povečamo dolžino in če je dolžina daljša od najdaljše doslej, si jo zapomnimo. Če sta različna, pa postavimo dolžino na 0, da bomo šteli spet od začetka.

```
def skupni_odsek(pot1, pot2):  
    naj = 0  
    dolzina = 0  
    for a, b in zip(pot1, pot2):  
        if a == b:  
            dolzina += 1  
            if dolzina > naj:  
                naj = dolzina  
        else:  
            dolzina = 0  
    return naj
```

2. Večji frajer

Kolesarja sta na svoji poteh po Ljubljani razkazovala veščine. Napiši funkcijo `vecji_frajer(pot1, pot2, zemljevid)`, ki vrne `True`, če je kolesar, ki je prevozil `pot1`, večji frajer od kolesarja, ki je prevozil `pot2`. Sicer naj vrne `False`.

Prvi je večji frajer, če je vsako veščino, ki jo je uporabil drugi, uporabil vsaj tolikokrat kot drugi, vsaj eno veščino pa večkrat kot drugi (pri čemer je to lahko tudi neka veščina, ki je drugi sploh ni pokazal).

Ključni slovarja `zemljevid` so pari (tuple) križišč, pripadajoče vrednosti pa množica veščin, ki jih je potrebno uporabiti na povezavi. Pot je vedno možna.

Rešitev

Tole je naloga iz slovarjev (in malo tudi) množic.

Za oba kolesarja je najprej potrebno prešteti, katere veščine sta uporabila kolikokrat. To smo se naučili storiti v domači nalogi Nagradne točke, v nalogi za oceno 7. Lahko prepišemo od tam, lahko naredimo na novo, saj ni težko.

Najlepše bo, da to damo kar v ločeno funkcijo. Uporabili bomo `defaultdict`, ker bo tako najbolj praktično. Uporaba običajnih slovarjev bi zahtevala dodaten pogoj v tej funkciji - in še enega v naslednji.

```
from collections import defaultdict

def poraba(pot, zemljevid):
    por = defaultdict(int)
    for a, b in zip(pot, pot[1:]):
        for vescina in zemljevid[(a, b)]:
            por[vescina] += 1
    return por
```

Zdaj jo pokličemo, da dobimo uporabljene veščine za oba kolesarja in potem gremo, kot pravi naloga, čez vse, kar je počel drugi kolesar. Za vsako veščino preverimo, ali jo je prvi slučajno pokazal manjkrat: če je tako, vrnemo `False`.

Če pride funkcija do konca, je prvi storil vse, kar je storil drugi (in to vsaj tolikokrat). Možno je torej, da je storil kakšno dodatno reč (ali pa je kaj storil večkrat) ali pa sta slovarja enaka. Če sta enaka, vrnemo `False`, sicer `True`. Se pravi: vrnemo `poraba1 != poraba2`.

```
def vecji_frajer(pot1, pot2, zemljevid):
    poraba1 = poraba(pot1, zemljevid)
    poraba2 = poraba(pot2, zemljevid)
    for vescina, uporab in poraba2.items():
        if poraba1[vescina] < uporab:
            return False
    return poraba1 != poraba2
```

3. Branje zemljevidov

Zemljevid je shranjen v datoteki v takšni obliki.

```
trava: BF-FRI, FRI-EF
gravel: BF-FRI
pešci:
pesek: BF-FRI, BF-FDV
```

Napiši funkcijo `preberi_zemljevid_vescin(ime_datoteke)`, ki prejme ime datoteke z zemljevidom in vrne slovar, katerega ključi so povezave (par križišč), pripadajoče vrednosti pa veščine, zahtevane za to povezavo. Za primer na sliki

mora vrniti `{("BF", "FRI"): {"trava", "gravel", "pesek"}, {"("BF", "FDV"): {"pesek"}}, {"("FRI", "EF"): {"trava"}}}`.

Rešitev

To je očitno naloga iz dela z datotekami in nizi (in spet tudi slovarji in množicami).

Uporabili bomo `defaultdict` z množico kot privzeto vrednostjo. Ker bo tako preprosteje.

Vsako vrstico datoteke razbijemo glede na `:`; levo je `vescina`, desno `povezave`. Če desni del ni prazen, ga razbijemo glede na `,` in gremo z zanko čez dobljene `povezave`. Vsako `povezava` razbijemo glede na `-`. Dobljeni par bo ključ slovarja, in v množico, ki pripada temu ključu, dodamo veččino.

```
def preberi_zemljevid_vescin(ime_datoteke):
    zemljevid = defaultdict(set)
    for vrstica in open(ime_datoteke):
        vescina, povezave = vrstica.split(":")
        if povezave.strip():
            for povezava in povezave.split(","):
                a, b = povezava.strip().split("-")
                zemljevid[a, b].add(vescina)
    return zemljevid
```

4. Izbire

Ker je prva prioriteta Mestne občine Ljubljana (MOL) varnost kolesarjev, so sprejeli predpis, po katerem smemo iz vsakega križišča voziti le v križišča, katerih ime je po abecedi kasnejše od trenutnega: iz D smemo v R, obratno pa ne.

Kolesarji se, kot vedno, pritožujejo, zato bi MOL rad pokazal, da bodo kljub tej omejitvi še vedno lahko pokazali vse, kar znajo. Sestavi funkcijo `koristne_vescine(tocka, zemljevid)`, ki vrne množico vseh veččin, ki lahko pridejo prav nekomu, ki začne v podani točki. Klic `koristne_vescine("P", zemljevid)` vrne `{'gravel', 'trava', 'robnik', 'lonci'}`, saj so to vse veččine, ki se pojavijo, če začnemo v P in gremo po poljubni poti, ki pa nikoli ne gre v točko, ki je po abecedi pred trenutno.

Rešitev

Tole je očitno naloga iz rekurzije.

Da bo lepše teklo, si lahko napišemo pomožno funkcijo `povezani(odkod, zemljevid)`, ki vrne vsa križišča, v katera lahko pridemo iz odkod.

```
def povezani(odkod, zemljevid):
    return {b for a, b in povezave if a == odkod and b > a}
```

Zdaj gremo rekurzivno po vseh križiščih, čisto tako, kot smo hodili po rodbini. Za vsako povezavo dodamo koristne veščine na tej povezavi in potem še koristne veščine, ki sledijo tej povezavi.

```
def koristne_vescine(odkod, zemljevid):
    koristne = set()
    for naprej in povezani(odkod, zemljevid):
        koristne |= povezave[odkod, naprej]
        koristne |= koristne_vescine(naprej, zemljevid)
    return koristne
```

Gre tudi brez dodatne funkcije.

```
def koristne_vescine(odkod, zemljevid):
    koristne = set()
    for (krizisce, naprej), vscine in zemljevid.items():
        if naprej > krizisce == odkod:
            koristne |= vscine | koristne_vescine(naprej, zemljevid)
    return koristne
```

5. Kolesar - frajer

Vaši kolegi so ob devetih napisali razred Kolesar z naslednjimi metodami.

- Konstruktor prejme ime začetne točke, zemljevid in množico veščin, ki jih kolesar obvlada.
- Metoda premik(kam) ga premakne s trenutne točke na podano, vendar le, če obstaja povezava s trenutne točke do podane in kolesar obvlada potrebne veščine. Sicer pa zleti s ceste in se ne premakne nikoli več. (Sam si je kriv: pa naj se nauči skakati čez robnike, če hoče voziti po ljubljanskih kolesarskih stezah!)
- Metoda lokacija() vrne trenutno lokacijo ali None, če kolesar leži pod cesto.

```
class Kolesar:
    def __init__(self, lokacija, zemljevid, vscine):
        self._lokacija = lokacija
        self.zemljevid = zemljevid
        self.vscine = vscine

    def premik(self, kam):
        if self.zemljevid.get((self._lokacija, kam), {None}) <= self.vscine:
            self._lokacija = kam
        else:
            self._lokacija = None

    def lokacija(self):
        return self._lokacija
```

Vaša naloga je, da iz njega izpeljete razred Frajer.

- Frajer ima tri življenja. Če poskusi premik, ki ni možen, se ne premakne, vendar se lahko ob naslednjem klicu metode premik vseeno premika naprej. Ampak samo dvakrat. Ko izgubi tretje življenje, dokončno obleži.
- Metoda zivljenja() vrne število preostalih življenj.
- Metoda uporabljene(), ki vrne množico veščin, ki jih je kolesar pokazal na svoji poti.

Razred izpelji tako, da bo imel svojo metodo premik, ki pa bo za samo premikanje poklicala podedovano metodo premik. Prav tako naj morebitni novi konstruktor pokliče podedovani konstruktor.

Rešitev

Število življenj bo zapisano v atributu `_zivljenja` (da se ne tepe z istoimensko metodo, poleg tega pa podrčrtaj na začetku nakazuje, da gre za bolj zaseben atribut razreda), množica uporabljenih veščin pa v `_uporabljene`.

Konstruktor atributoma nastavi ustrezni začetni vrednosti, metodi `zivljenja` in `uporabljene` pa zgolj vračata trenutni vrednosti teh dveh atributov.

Glavna naloga je `premik`. Ta shrani trenutno lokacijo. Nato pokliče podedovani `premik`. Če je po tem trenutna lokacija različna od `None` (pišemo lahko `if self.lokacija() is not None`, `if self.lokacija() != None`, `if self.lokacija()`, `if self._lokacija is not None` ...)) pribeležimo uporabljene veščine. Če se je po (podedovanem) premiku znašel na `None`, pa zmanjšamo število življenj. Če je le-to večje od 0, spremenimo lokacijo nazaj na tisto, ki jo je imel pred premikom.

```
class Frajer(Kolesar):
    def __init__(self, lokacija, zemljevid, vescine):
        super().__init__(lokacija, zemljevid, vescine)
        self._zivljenja = 3
        self._uporabljene = set()

    def premik(self, kam):
        zacetek = self.lokacija()
        super().premik(kam)
        if self.lokacija() is not None:
            self._uporabljene |= self.zemljevid[zacetek, kam]
        else:
            self._zivljenja = max(0, self._zivljenja - 1)
            if self._zivljenja > 0:
                self._lokacija = zacetek

    def zivljenja(self):
        return self._zivljenja

    def uporabljene(self):
```

```
return self._uporabljene
```