

## 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 indeksi
```

```
def skupne_povezave(pot1, pot2):  
    # Tole ni najlepša rešitev, le najpreprostejša - za začetnike :)  
    skupnih = 0  
    for i in range(min(len(pot1), len(pot2)) - 1):  
        if pot1[i:i + 1] == pot2[i:i + 1]:  
            skupnih += 1  
    return skupnih
```

```
# 2. Slovarji in množice
```

```
def najzahtevnejse(zemljevid):  
    potrebne = defaultdict(set)  
    for (a, _), vescine in zemljevid.items():  
        potrebne[a] |= vescine  
  
    naj_krizisce = None  
    for a, vescine in dict(potrebne).items():  
        if len(vescine) > len(potrebne[naj_krizisce]):  
            naj_krizisce = a  
    return naj
```

```
# 3. Datoteke in nizi
```

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

#### *# 4. Rekurzivne funkcije*

```
def stevilo_poti(odkod, kam, povezave):
    if odkod == kam:
        return 1
    nacinov = 0
    for a, b in povezave:
        if a == odkod and b > a:
            nacinov += stevilo_poti(b, kam, povezave)
    return nacinov
```

#### *# 5. Objektno programiranje*

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

    def premik(self, kam):
        if (self._lokacija, kam) in zemljevid and self.zemljevid[(self._lokacija, kam)] <= self.vescine:
            self._lokacija = kam
        else:
            self._lokacija = None

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

### 1. Skupne povezave

Dva kolesarja sta šla istočasno na pot. Za vsako povezavo sta potrebovala enako časa. Napiši funkcijo `skupne_povezave(pot1, pot2)`, ki vrne število povezav, ki sta jih prevozila skupaj. Poti sta podani z nizom zaporednih črk, ki označujejo križišča.

Klic

```
skupne_povezave("ASAIMWGVIEHTEUTUMVIVHIV",
                 "OIMIMAWAREMPBTUMGIBTIOWE")
```

vrne 4, saj sta kolesarja na poteh v okvirčku sta skupaj prevozila štiri povezave: IM, EM, TU in UM.

### Rešitev

Tole je naloga, v kateri je bilo potrebno pokazati, da znate pisati zanke.

Manj lepa (a popolnoma pravilna) je rešitev z indeksi.

```
def skupne_povezave(pot1, pot2):
    skupnih = 0
    for i in range(min(len(pot1), len(pot2)) - 1):
        if pot1[i:i + 1] == pot2[i:i + 1]:
            skupnih += 1
    return skupnih
```

Z `i` stopicamo od 0 do toliko, kolikor dolga je krajša izmed poti. Če sta `i`-ti in `i+1`-ti znak obeh poti enaka, povečamo števec skupnih povezav.

Spretnější in Pythona večšejši naredijo takole. Sestavijo seznam zaporednih znakov za prvo pot (`zip(pot1, pot1[1:])`) in za drugo pot (`zip(pot2, pot2[1:])`). Na to grejo vzporedno prek obeh seznamov (še en `zip`!). `par1` bo par iz prve poti in `par2` bo par iz druge poti. Če sta enaka, povečamo števec skupnih poti.

```
def skupne_povezave(pot1, pot2):
    skupnih = 0
    for a, b in zip(zip(pot1, pot1[1:]), zip(pot2, pot2[1:])):
        if a == b:
            skupnih += 1
    return skupnih
```

Še spretnejši pa prepustijo seštevanje Pythonu.

```
def skupne_povezave(pot1, pot2):
    return sum(par1 == par2 for par1, par2 in zip(zip(pot1, pot1[1:]), zip(pot2, pot2[1:])))
```

**Kasnejši dodatek, ob popravljanju izpitov:** kudos študentki, ki je pomislila malo drugače in napisala tole:

```
def skupne_povezave(pot1, pot2):
    skupnih = 0
    for p1, p2, r1, r2 in zip(pot1, pot1[1:], pot2, pot2[1:]):
        if p1 == r1 and p2 == r2:
            skupnih += 1
    return skupnih
```

Sicer pa sem velikokrat videl ta vzorec: greste vzporedno prek obeh poti - večina z indeksom, redki z `zip`-om. Če sta istoležni križišči enaki, si to zapomnite. In če sta bili enaki tudi prejšnji, to pomeni, da je enaka zadnja povezava.

V nekoliko okornejši različici je to videti tako.

```
def skupne_povezave(pot1, pot2):
    skupnih = 0
    prej_skupna = False
    for a, b in zip(pot1, pot2):
        if a == b:
```

```

        if prej_skupna:
            skupnih += 1
            prej_skupna = True
        else:
            prej_skupna = False
    return skupnih

```

Lepše pa je tako.

```

def skupne_povezave(pot1, pot2):
    skupnih = 0
    prej_skupna = False
    for a, b in zip(pot1, pot2):
        if a == b and prej_skupna:
            skupnih += 1
        prej_skupna = a == b
    return skupnih

```

Ali pa kar

```

def skupne_povezave(pot1, pot2):
    skupnih = 0
    prej_skupna = False
    for a, b in zip(pot1, pot2):
        skupnih += (a == b and prej_skupna)
        prej_skupna = a == b
    return skupnih

```

## 2. Najzahtevnejše križišče

Napiši funkcijo `najzahtevnejse(zemljevid)`, ki vrne tisto križišče, ki ga obkrožajo povezave z največ različnimi veščinami. Če je možnih odgovorov več, naj vrne enega od njih. Zemljevid je podan kot slovar, katerega ključi so pari (tuple) križišč, pripadajoče vrednosti pa množica veščin, ki jih je potrebno uporabiti na povezavi.

Za primer na zemljevidu funkcija vrne I ali M, saj ju obkrožajo povezave s šestimi različnimi veščinami (I, na primer, obkrožajo gravel, trava, stopnice, robnik, lonci, avtocesta).

### Rešitev

Tole je naloga iz slovarjev in množic. Za vsako križišče moramo dobiti množico vseh veščin, ki ga obkrožajo. Torej potrebujemo slovar, katerega ključi so križišča, vrednosti pa množice veščin. Da se ne zafrkavamo s praznimi množicami, bomo uporabili `defaultdict(set)`.

V prvem delu funkcije slovar sestavljamo. Gremo čez zemljevid in za vsako povezavo pod ustrezen ključ dodamo večšine, ki jih zahteva ta povezava.

V drugem delu funkcije poiščemo ključ, ki mu pripada največja vrednost.

```
from collections import defaultdict

def najzahtevnejse(zemljevid):
    potrebne = defaultdict(set)
    for (krizisce, _), vescine in zemljevid.items():
        potrebne[krizisce] |= vescine

    naj_krizisce = None
    naj = 0
    for krizisce, vescine in dict(potrebne).items():
        if len(vescine) > naj:
            naj = len(vescine)
            naj_krizisce = krizisce
    return naj_krizisce
```

Ali, malo krajše, tako:

```
def najzahtevnejse(zemljevid):
    potrebne = defaultdict(set)
    for (a, _), vescine in zemljevid.items():
        potrebne[a] |= vescine

    naj_krizisce = None
    for a, vescine in dict(potrebne).items():
        if len(vescine) > len(potrebne[naj_krizisce]):
            naj_krizisce = a
    return naj
```

Precej manj učinkovita, a vseeno pravilna, je rešitev brez slovarja. Ta gre čez vsa križišča (in, uh, čez nekatera celo večkrat) ter za vsakega prešteje vse potrebne večšine.

```
def najzahtevnejse(zemljevid):
    naj = 0
    naj_krizisce = None
    for (krizisce, _) in zemljevid:
        potrebne = set()
        for (b, _), vescine in zemljevid.items():
            if b == krizisce:
                potrebne |= vescine
        if len(potrebne) > naj:
            naj = len(potrebne)
            naj_krizisce = krizisce
    return naj_krizisce
```

To različico bi se dalo še malo pospešiti, vendar se s tem ne bomo ukvarjali. Prva je bistveno boljša in hitrejša.

### 3. Branje zemljevidov

Zemljevid je shranjen v datoteki v takšni obliki.

BF-FRI: trava, gravel, pesek

BF-FDV: pesek

BF-EF:

FRI-EF: trava

Napiši funkcijo `preberi_zemljevid_povezav(ime_datoteke)`, ki prejme ime datoteke z zemljevidom in vrne slovar, katerega ključi so veščine, pripadajoče vrednosti pa množica povezav (povezava je podana s parom križišč), ki zahtevajo to veščino. Za primer na sliki mora vrniti `{"trava": {("BF", "FRI"), ("FRI", "EF")}, "gravel": {("BF", "FRI")}, "pesek": {("BF", "FRI"), ("BF", "FDV")}`.

#### Rešitev

Očitno je to naloga iz branja datotek in dela z nizi.

Naš slovar bo torej imel veščine za ključe in množice za vrednosti. Spet bomo uporabili `defaultdict`, da se bodo elementi slovarja pojavljali kar sami od sebe, čim bomo poskušali kaj dodajati v množice.

Zdaj pa h glavnemu. Datoteko pač odpremo in beremo. Vsak niz razbijemo po `:`. Vemo, da bomo vedno dobili dve stvari, torej smemo pisati kar `povezava`, `vescine = vrstica.split(":")`.

Prvi del, ki opisuje povezavo, razbijemo glede na `-`. Spet vemo, da bomo dobili dve stvari, zato pišemo kar `a, b = povezava.split("-")`.

Drugi del, veščine pa razbijemo glede na `,`. Čez dobljene kose niza gremo z zanko in v množico, ki pripada ključu `vescina.strip()` dodamo povezavo (`a, b`).

Še zadnji detajl: če neka povezava ne zahteva nobene veščine, bodo `vescine` enaka `\n` (znak za novo vrstico) in `vescine.split(",")` bo vrnil `["\n"]`, zato se bo v slovarju pojavila tudi ta veščina. Tega se znebimo z `if vescine != "\n"` ali `if vescine.strip()` ali s čim podobnim.

```
def preberi_zemljevid_povezav(ime_datoteke):
    po_vescinah = defaultdict(set)
    for vrstica in open(ime_datoteke):
        povezava, vescine = vrstica.split(":")
        if vescine.strip():
            a, b = povezava.split("-")
            for vescina in vescine.split(","):

```

```

        po_vescinah[vescina.strip()].add((a, b))
    return po_vescinah

```

#### 4. Izbire, izbire, 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, usajajo, zato bi MOL rad pokazal, da v ničemer ne omejuje svobode kolesarjev. MOL prosi, da sestaviš funkcijo `stevilo_poti(odkod, kam, zemljevid)`, ki vrne število možnih načinov, na katere lahko pridemo od odkod do kam.

Klic `stevilo_poti("G", "N", zemljevid)` vrne 3 (možne poti so GIMN, GHJKMN, GJLMN).

#### 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}

```

In zdaj je stvar praktično enaka eni od domačih nalog (in ta je enaka funkciji, ki smo jo imeli na predavanjih, to je, velikosti rodbine).

```

def stevilo_poti(odkod, kam, povezave):
    if odkod == kam:
        return 1
    nacinov = 0
    for naprej in povezani(odkod, povezave):
        nacinov += stevilo_poti(naprej, kam, povezave)
    return nacinov

```

Seveda bi šlo tudi brez pomožne funkcije.

```

def stevilo_poti(odkod, kam, povezave):
    if odkod == kam:
        return 1
    nacinov = 0
    for a, b in povezave:
        if a == odkod and b > a:
            nacinov += stevilo_poti(b, kam, povezave)
    return nacinov

```

Bolj duhoviti pa z nalogo opravijo v enem zamahu.

```
def stevilo_poti(odkod, kam, povezave):
    return odkod == kam or sum(stevilo_poti(b, kam, povezave) for a, b in povezave if a == odkod)
```

## 5. Razredni kolesar

Napiši razred `Kolesar`.

- Konstruktor prejme začetno točko, 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 je kolesar obležal pod cesto.

### Rešitev

Konstruktor shrani vse, kar mu je bilo podano. Da se atribut `lokacija` ne bi stepel z istoimensko metodo, mu pred ime dodamo podčrtaj.

Metoda `premik` preveri, ali povezava iz trenutne lokacije obstaja in ali ima kolesar vse potrebne veščine zanjo. Če je tako, ga premakne tja, sicer pa v `None`. To mu bo tudi onemogočilo nadaljnje premike, saj iz `None` ni nobene povezave.

In metoda `lokacija` pač vrne vrednost atributa `_lokacija`.

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

    def premik(self, kam):
        if (self._lokacija, kam) in zemljevid and self.zemljevid[(self._lokacija, kam)] <= self.vescine:
            self._lokacija = kam
        else:
            self._lokacija = None

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