1. Vožnje

Vožnje so shranjene v datotekah v obliki, ki jo kaže okvirček na desni: vsaka vrstica vsebuje ime voznika in razdaljo. Napišite dve funkciji.

```
Ana,15
Berta,42
Ana,85
Berta,83
Cilka,15
Berta,29
Berta,82
Dani,81
Cilka,192
Dani,92
Ana,115
Ema,200
```

- **`kilometri(ime_datoteke)`** vrne slovar, katerega ključi so imena voznikov, pripadajoče vrednosti pa skupna dolžina voženj, ki jih je opravil ta voznik. Za primer iz okvirčka vrne `{'Ana': 215, 'Berta': 236, 'Cilka': 207, 'Dani': 173, 'Ema': 200}`.
- **`prekrskarji(ime_datoteke, meja)`** `vrne` seznam imen voznikov, ki so prevozili več kot meja kilometrov. Klic `prekrskarji("voznice.txt", 200)` vrne `["Ana", "Berta", "Cilka"]`.

2. Potovanje

Testi vsebujejo funkcijo `razdalja(odkod, kam)`, ki vrne razdaljo med dvema krajema. Klic `razdalja("Kranj", "Ljubljana")` vrne `26`. Klic `razdalja("Kranj", "Kranj")` vrne `0`. **Te funkcije ne pišite, ta že obstaja!** Pač pa napišite:

- **`dolzina_vozenj(poti)`** prejme seznam relacij (v obliki parov krajev), ki jih mora ta voznik prevoziti in vrne skupno dolžino teh relacij. Uporabite zgoraj opisano, podano funkcijo `razdalja`. Klic dolzina_**`vozenj`**`([("Ljubljana", "Kranj"), ("Kranj", "Novo mesto"), ("Lendava", "Ormož")])` vrne 176, kolikor je vsota razdalj med Ljubljano in Kranjem, Kranjem in Novim mestom ter Lendavo in Ormožem.
- **`prevozeno(zacetek, poti)`** poleg tega kot prvi argument prejme ime kraja, v katerem se voznik nahaja v začetku. Pri računu dolžine poti upošteva, da **`mora`** priti voznik do začetnega kraja, poleg tega doda dolžine relacij med koncem ene in začetkom naslednje. Če pokličemo `prevozeno("Postojna", [("Ljubljana", "Kranj"), ("Kranj", "Novo mesto"), ("Lendava", "Ormož")])` bo poleg prej naštetih relacij upošteval še razdaljo med Postojno in Ljubljano ter med Novim mestom in Lendavo.

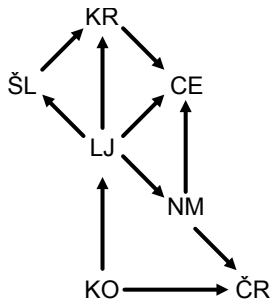
3. Déjà vu

Naslednje funkcije prejmejo tri argumente: ime voznika in tabeli v numpyju, ki vsebujeta stolpca, kakršna imajo datoteke iz prve naloge, torej `["Ana", "Berta", "Ana", "Berta", "Cilka", ...]` in `[15, 42, 85, 83, 15, 29 ...]`. Uporabite, kar potrebujete. Naloge rešujte z numpyjem; drugačne rešitve bodo prejele polovično število točk.

- **`stevilo_vozenj(voznik, vozniki, razdalje)`** vrne število voženj, ki jih je opravil podani voznik.
- **`prevozena_razdalja(voznik, vozniki, razdalja)`** vrne skupno razdaljo, ki jo je prevozil podani voznik
- **`povprecje_treh(voznik, vozniki, razdalja)`** vrne povprečno razdaljo najdaljših treh voženj. Če je voznik opravil le dve vožnji, vrne njuno povprečno `dolžino`; če le eno, vrne dolžino te.

4. Kombiniranje voženj

Imamo nek začetni kraj, recimo LJ. Imamo seznam relacij, ki jih je potrebno prevoziti, na primer `[("LJ", "ŠL"), ("LJ", "KR"), ("ŠL", "KR"), ("KR", "CE"), ("LJ", "CE"), ("LJ", "NM"), ("NM", "ČR"), ("NM", "CE"), ("KO", "ČR"), ("KO", "LJ")]`. (Te relacije so prikazane na sliki.) Relacije ne vsebujejo ciklov – če vozimo po njih, se ne bomo nikoli vrnili v kraj, v katerem smo že bili. (To poenostavi funkcijo, ki jo boste pisali!)



Če začnemo v LJ, lahko gremo v CE in obtičimo tam. Lahko pa gremo LJ – KR – CE. Še daljša možnost je LJ – ŠL – KR – CE. To je hkrati najdaljša pot, ki jo lahko sestavimo iz teh relacij, če začnemo v LJ.

Napišite funkcijo **`kombiniraj(zacetek, relacije)`**, ki prejme začetni kraj in seznam relacij ter vrne največje število relacij, ki jih lahko prevozimo. V gornjem primeru vrne 3.

Če bi namesto tega začeli v KO, bi vrnila 4. (Iz slike je očitno: toliko kot vrne za LJ in še 1 več, ker gremo prej še KO – LJ.)

Kdor je večji frajer, frajerka, pa bo namesto tega napisal funkcijo tako, da bo vrnila zaporedje krajev. Za gornji primer vrne `["LJ", "ŠL", "KR", "CE"]` oziroma `["KO", "LJ", "ŠL", "KR", "CE"]`. (Ta različica pravzaprav ni nič težja od prejšnje, vendar se vam bo prejšnja morda zdela lažja.)