Izpit iz predmeta Programiranje 1, 13. februar 2024 **Čas pisanja: ~~60 minut~~ → 90 minut**

Dovoljena je uporaba literature (Učilnica in vse, kar prinesete s seboj), prepovedana pa je vsakršna komunikacija.

Kot vemo, so ljubljanski kolesarji sami divjaki in ne prilagodijo svoje hitrosti, ko se znajdejo na snežnih poljanah ljubljanskih pločnikov. Zato jih po sledeh ne nosi le tri korake, kot v domači nalogi, temveč se zgodi, kar kaže slika: kolesar v neskončnost vozi po poti, ki jo kažejo puščice. Na sliki se bo prej ko slej znašel v enem od dveh ciklov. Predpostaviti smemo, da nobena puščica ne kaže ven iz polja.

Razpored puščic bo v teh nalogah podan v obliki seznama nizov. Sliko na desni predstavimo z `[">>v>>v", "vvv^<<", ">>>>v^", "^^<>>v", ">^<<<<"]`.

1. Unzip

Napiši funkcijo `unzip(s, sirina)`, ki sprejme zemljevid v stisnjeni obliki in njegovo širino. Funkcija mora vrniti zemljevid v obliki seznama nizov. Tako klic `unzip(">1v>1v3^<1>3v^2<>>v>^<3", 6)` vrne natančno gornji zemljevid.

Števka za znakom pove, koliko ponovitev posamičnega znaka je potrebno še dodati. Tako, recimo, `>2` pomeni `>`, ki mu sledita še dva `>`. Torej `>2` pomeni `>>>`. Številko vedno beremo po števkah: `>56` ne pomeni `>` in še 56 `>`-jev, temveč `>`, in še 5 `>`-jev in še 6 `>`-jev (torej, skupno 12 `>`-jev). Gornji niz `>1v>1v3^<1>3v^2<>>v>^<3` bo funkcija torej najprej raztegnila v '>>v>>vvvv^<<>>>>v^^^<>>v>^<<<<', potem pa ga razbila po vrsticah, tako da bo v vsaki 6 (drugi argument!) znakov.

2. Pred ciklom

Napiši funkcijo `pred_ciklom(zemljevid, x, y)`, ki sprejme zemljevid in neko začetno koordinato ter vrne število korakov, po katerih se kolesar znajde v ciklu. Klic `pred_ciklom(sneg, 1, 0)` vrne `3`, saj bo kolesar, ki začne na drugem polju prve vrstice do cikla (spodnjega) potreboval tri korake.

Nasvet: napiši si pomožno funkcijo, ki prejme zemljevid in polje ter vrne *seznam* z vsemi polji, ki jih prevozi kolesar do trenutka, ko se prvo polje ponovi. V seznam naj bo vključeno tudi to, ponovljeno polje. Ta funkcija ti bo prišla še bolj prav pri naslednji nalogi. Seveda pa lahko nalogi rešuješ kako drugače, brez te funkcije, če želiš.

3. Polja do

Napiši funkcijo `polja_do(zemljevid, x, y)`, ki vrne množico vseh polj, ki kolesarja pripeljejo na podano polje. Množica naj vključuje tudi podano polje. Klic `polja_do(zemljevid, 2, 1)` vrne `{(0, 1), (0, 2), (0, 3), (1, 2)}`, saj so to polja, ki ga pripeljejo na 2, 1.

4. Rekurzivna polja do

Napiši **rekurzivno** funkcijo `polja_do_rek(zemljevid, x, y, prepovedana)`, ki počne isto kot prejšnja, le da ima dodatni argument - množico polj, ki so prepovedana (na njih je namreč kup snega).

Nasveta: do tega polja pripeljejo vsa sosedna polja, ki vodijo v to polje, poleg njih pa tudi vsa polja, ki pripeljejo do teh polj. S pomočjo prepovedanih polj pa lahko preprečimo, da bi se rekurzija zaciklala.

5. Plug

Sneg bo počasi skopnel, zato MOL ne vidi več razloga, da ne bi splužil pločnikov in kolesarskih stez.

Napiši razred `Plug`.

- Konstruktor sprejme zemljevid in svoje začetne koordinate.
- Metoda `lokacija()` vrne trenutne koordinate pluga.
- Metoda `premik(znak)` premakne plug za en znak v podano smer, *če ga to ne pripelje ven iz zemljevida*; plug pri tem očisti polje, ki ga je zapustil, tako da spremeni znak, ki pripada temu polju v `"."`. *Če bi ga premik pripeljal ven in zemljevida*, pa ne očisti trenutnega polja, poleg tega pa zahteva servis, kar pomeni, da ignorira naslednje tri ukaze; po tem ponovno deluje.
- Metoda `ociscenih()` vrne število polj, ki jih je očistil ta plug.

Pazi: več plugov pluži isti zemljevid! (Dva pluga sta lahko tudi na istem polju, ker to ni problem.)

Nizi v Pythonu so nespremenljivi. Da spremenimo `c`-ti znak niza `s` v piko, uporabimo `s = s[:c] + "." + s[c + 1:]`.