Programiranje I: 3. pisni izpit

31. velikega srpana leta Gospodovega 2015

Naloge v Pythonu rešujte na strežniku Tomo, naloge v Mathematici pa v ustreznem Mathematica zvezku (ki ga najdete na spletni učilnici). Čas reševanja je 150 minut. Doseženih 100 točk šteje za maksimalno oceno. Veliko uspeha!

1. naloga (Polžek, 20 + 10 točk)

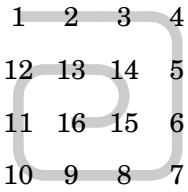
Matriko v Mathematici predstavimo kot seznam seznamov, kjer vsak podseznam predstavlja eno vrstico matrike. Na primer, matriko

$$\begin{pmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \\ 0 & 4 & 1 \end{pmatrix}$$

predstavimo s seznamom

```
{{1, 2, 3}, {5, 6, 7}, {0, 4, 1}}
```

a) (20 točk) V *Mathematici* sestavite funkcijo `polzek[l_]`, ki kot argument dobi seznam `l` ter sestavi in vrne *kvadratno* matriko, ki vsebuje elemente tega seznama. Prvi element seznama `l` naj bo v zgornjem levem vogalu matrike. Elementi seznama `l` naj se v matriki "zvijejo v polžka", kot je prikazano na skici:



Matrika naj bo tako velika, kot je nujno potrebno, da lahko vanjo pospravimo celega polžka. Morebitna "prazna polja" zapolnimo z ničlami. Zgled:

```
In[1]:= polzek[{1, 4, -1, -3, 2}]
Out[1]= {{1, 4, -1}, {0, 0, -3}, {0, 0, 2}}
In[2]:= polzek[{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 2, 7}]
Out[2]= {{2, 3, 5, 7}, {7, 0, 0, 11}, {2, 0, 0, 13}, {29, 23, 19, 17}}
```

b) (10 točk) V *Mathematici* sestavite še "inverzno" funkcijo `odvij[m_]`, ki kot argument dobi kvadratno matriko `m` ter vrne seznam elementov te matrike, tako da bo veljajo

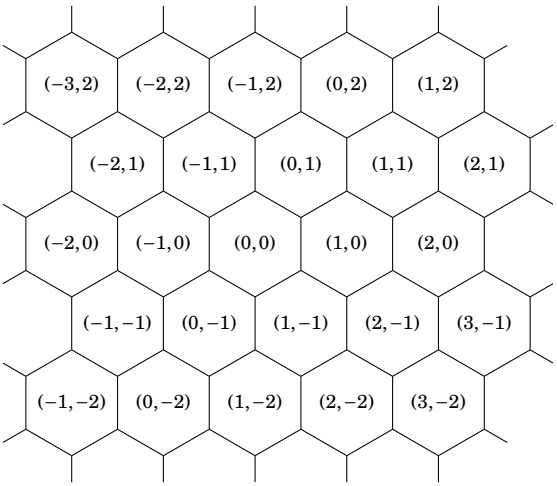
`polzek[odvij[m]] == m`.

Zgled:

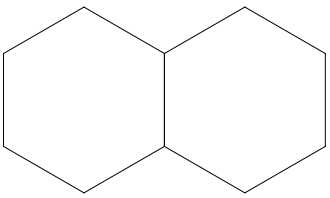
```
In[3]:= odvij[{{1, 4, -1}, {0, 0, -3}, {0, 0, 2}}]
Out[3]= {1, 4, -1, -3, 2, 0, 0, 0, 0}
In[4]:= odvij[{{2, 3, 5, 7}, {7, 0, 0, 11}, {2, 0, 0, 13}, {29, 23, 19, 17}}]
Out[4]= {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 2, 7, 0, 0, 0, 0}
```

2. naloga (Šestkotniki, 30 točk)

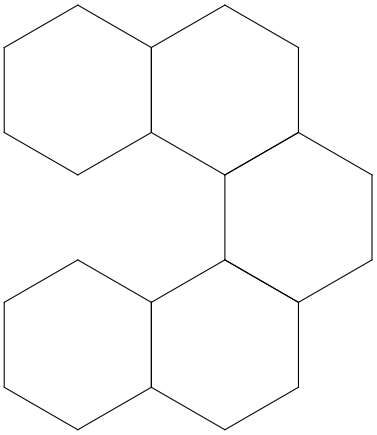
Znano je, da lahko ravnino tlakujemo s pravilnimi šestkotniki. Na tako dobljeni šestkotniški mreži vpeljemo koordinatni sistem, tako da vsak element iz $\mathbb{Z} \times \mathbb{Z}$ označuje natanko določen šestkotnik, kot je prikazano na skici:



V *Mathematici* sestavite funkcijo `benz[l_]`, ki kot argument dobi seznam `l` in nariše tiste šestkotnike, ki so podani v seznamu `l`. Šestkotniki so podani z zgoraj opisanimi koordinatami, tj. kot pari celih števil (glejte zgornjo skico). Zgledi:



`benz[{{-1, 0}, {0, 0}}]`



`benz[{{0, 0}, {1, 0}, {1, 1}, {0, 2}, {-1, 2}}]`

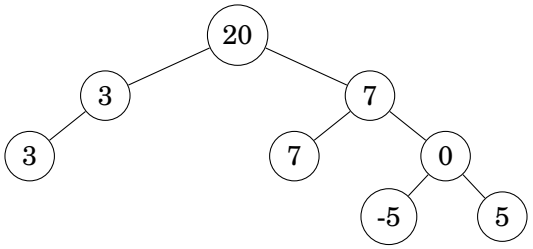
Namig: Pravokotni koordinatni sistem postavimo tako, da je izhodišče koordinatnega sistema v središču šestkotnika z oznako $(0,0)$. Katera točka je središče šestkotnika z oznako (u,v)?

3. naloga (Dvojiško drevo, 10 + 10 + 10 točk)

Podatkovna struktura `Drevo` predstavlja dvojiško drevo, ki ima v vsakem vozlišču shranjeno eno celo število. Naslednji izraz v Pythonu ustvari drevo, ki je prikazano na spodnji sliki:

```
>>> s = Drevo(20, levo=Drevo(3, levo=Drevo(3)), desno=Drevo(7, levo=Drevo(7),
        desno=Drevo(0, levo=Drevo(-5), desno=Drevo(5))))
```

Pripadajoča slika:



Rekli bomo, da je drevo *sumarno*, če za vsako vozlišče, ki ni list, velja, da je njegova vrednost enaka vsoti vseh števil, ki so pod njim (tj. v levem in v desnem poddrevesu). Zgornje drevo je primer sumarnega drevesa.

Razred `Drevo` je že delno implementiran. Vsako vozlišče ima atribut `prazno`. Če je njegova vrednost `True`, predstavlja prazno poddrevo in nima drugih atributov. Če pa drevo ni prazno, ima še atribute `vsebina`, `levo` in `desno`. Dodali bomo še nekaj novih metod.

a) (10 točk) Razredu `Drevo` dodajte metodo `je_sumarno(self)`, ki vrne `True`, če je drevo sumarno, in `False` sicer. Za prazno drevo naj metoda tudi vrne `True`. Zgledi:

```
>>> s.je_sumarno()
True
>>> Drevo(3, levo=Drevo(1), desno=Drevo(1)).je_sumarno()
False
```

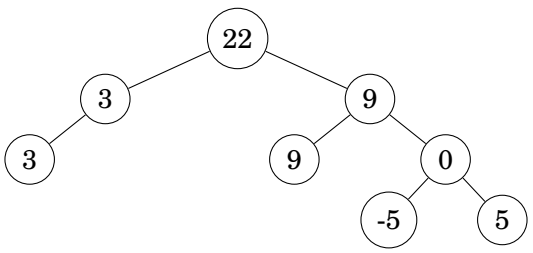
b) (10 točk) Izkaže se, da so vse vrednosti v sumarnem drevesu enolično določene takoj, ko podamo vrednosti v listih. Razredu `Drevo` dodajte metodo `naredi_sumarno(self)`, ki drevo popravi, tako da le-to postane sumarno. Vrednosti v listih metoda ne sme spreminjati. Zgled:

```
>>> d = Drevo(3, levo=Drevo(1), desno=Drevo(1))
>>> d.naredi_sumarno()
>>> d
Drevo(2, levo=Drevo(1), desno=Drevo(1))
```

c) (10 točk) Razredu `Drevo` dodajte metodo `pristej_po_poti(self, x, p)`, ki kot argument dobi število `x` in seznam `p`. Začnemo v korenu drevesa in gremo po poti, tako da se vsakič spustimo v eno od poddreves. Elementi seznama `p` nam povedo, v katero poddrevo se spustimo: niz `'l'` pomeni, da gremo v levo poddrevo; niz `'d'` pomeni, da gremo v desno poddrevo. Ko pridemo do praznega poddrevesa, se ustavimo. Če v seznamu `p` zmanja elementov, začnemo spet pri začetku (kakor da bi bil seznam `p` cikličen). Drevo pri tem spremenimo tako, da vsem elementom na tej poti prištejemo število `x`. Primer:

```
>>> s.pristej_po_poti(2, ['d', 'l', 'd'])
>>> s
Drevo(22, levo=Drevo(3, levo=Drevo(3)), desno=Drevo(9, levo=Drevo(9),
desno=Drevo(0, levo=Drevo(-5), desno=Drevo(5))))
```

Pripadajoča slika:



4. naloga (Igra urejanja, 10 + 10 + 10 točk)

Mojca in Gorazd se igrata zanimivo igro, ki ji pravita *Igra urejanja*. Začneta z nekim seznamom ℓ dolžine n, ki vsebuje vsa števila med 0 in $n-1$, na primer

$$\ell = [0,3,6,7,8,4,5,1,2].$$

Začne Gorazd, ki lahko v eni potezi med seboj zamenja poljubna dva elementa seznama. Poteza $(1,5)$ pomeni, da med seboj zamenja 1. in 5. element. Iz gornjega seznama bi tako dobil:

$$[0,4,6,7,8,3,5,1,2].$$

Mojca po vsaki Gorazdovi potezi lahko reče "Dalje!" ali pa "Stop!". Če reče "Dalje!", Gorazd nadaljuje s svojo naslednjo potezo. Če Mojca reče "Stop!", Gorazd prekine z izvajanjem svojih potez. Mojca pa mora seznam ℓ urediti naraščajoče, pri čemer sme porabiti največ toliko potez, kot jih je do tega trenutka naredil Gorazd. Če Mojci ne uspe urediti seznama, izgubi partijo.

a) (10 točk) Mojca je opazila, da seznam ℓ pravzaprav predstavlja permutacijo $i \mapsto \ell[i]$ na množici {0, 1, ..., n-1}. Sestavite funkcijo `ciklicni_zapis(l)`, ki vrne ciklični zapis permutacije, ki je podana s seznamom `l`. Cikli naj bodo urejeni leksikografsko, na prvem mestu v vsakem ciklu pa naj bo najmanjši element. Zgled:

```
>>> ciklicni_zapis([0, 3, 6, 7, 8, 4, 5, 1, 2])
[[0], [1, 3, 7], [2, 6, 5, 4, 8]]
```

b) (10 točk) Mojca je ugotovila, da je vsaka Gorazdova poteza v bistvu množenje z leve z ustrezno transpozicijo. V gornjem primeru to pomeni

$$(1\ 5) * (0)\ (1\ 3\ 7)\ (2\ 6\ 5\ 4\ 8) = (0)\ (1\ 4\ 8\ 2\ 6\ 5\ 3\ 7).$$

Mojca je ugotovila še, da lahko z eno transpozicijo:

- združimo dva cikla v enega ali pa
- en cikel razbijemo na dva manjša.

Ker urejen seznam ustreza identični permutaciji, moramo vse cikle razbiti do ciklov dolžine 1. Minimalno število potez, ki jih za to potrebujemo, lahko preberemo iz cikličnega tipa permutacije (za razbijanje cikla dolžine d potrebujemo $d-1$ transpozicij).

Sestavite funkcijo `razbij_cikle(p)`, ki kot argument dobi permutacijo `p`, ki je podana s cikličnim zapisom. Funkcija naj vrne seznam transpozicij, s katerimi moramo (z leve) pomnožiti permutacijo `p`, da dobimo identiteto. Število teh transpozicij mora biti najmanjše možno. Zgled:

```
>>> razbij_cikle([[0, 4, 3], [1, 5], [2], [7, 8, 9, 6]])
[(0, 4), (0, 3), (1, 5), (7, 8), (7, 9), (7, 6)]
```

Opomba: Tomo bo sprejel vse pravilne rešitve, tudi če vaša funkcija vrne rezultat, ki se razlikuje od "uradnega".

c) (10 točk) Prej ko Mojca reče "Stop!", več točk bo dobila. Če lahko Mojca seznam uredi po k Gorazdovih potezah, ga lahko uredi tudi po $k+1$ potezah (npr. tako, da razveljavi zadnjo njegovo potezo in postopa enako, kot bi tudi sicer po k-ti njegovi potezi). To pomeni, da lahko minimalno število potez, ki jih potrebuje za zmago, poišče z bisekcijo.

Napotek: Po tem, ko Gorazd na seznamu izvede k zamenjav, Mojca poišče ciklični zapis ter pogleda, če za razbijanje ciklov res porabi kvečjemu k transpozicij.

Napišite funkcijo `min_potez(l, g)`, ki kot argumenta dobi začetni seznam `l` in seznam transpozicij `g`, ki jih bo izvedel Gorazd. Funkcija naj vrne najmanjše število potez, ki jih izvede Gorazd, preden ima Mojca prvo priložnost za zmago. Gorazd poteze dela dovolj dolgo časa, da ima Mojca možnost za zmago. Če je seznam `l` že na začetku urejen, lahko Mojca zavpije "Stop!", še preden Gorazd naredi prvo potezo. Zgled:

```
>>> min_potez([0, 3, 6, 7, 8, 4, 5, 1, 2],
              [(1, 5), (1, 2), (4, 5), (7, 8), (2, 3), (1, 6), (3, 7)])
5
```