Programiranje I: 4. izpit

4. kimavec 2014

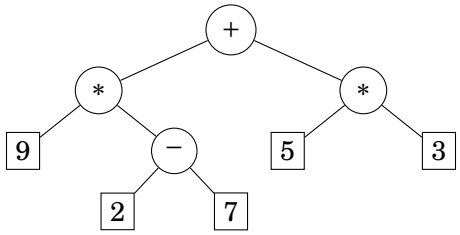
Čas reševanja je 150 minut. Doseženih 100 točk šteje za maksimalno oceno. Veliko uspeha!

1. naloga (Aritmetični izrazi, 10 + 10 + 10 točk)

Aritmetične izraze, kot je na primer

$$(9 * (2 - 7)) + (5 * 3),$$

lahko zapišemo v obliki dvojiškega drevesa (glejte spodnjo sliko). V vsakem vozlišču je zapisano bodisi neko celo število bodisi nek aritmetični operator. (Da ne bi imeli problemov z deljenjem s številom 0, se bomo omejili na operatorje $+$, $-$ in $*$.) Če vozlišče predstavlja operator, potem ima nujno levega in desnega sina, ki predstavljata ustrezna podizraza. Če vozlišče predstavlja število, je nujno list drevesa. (Ste opazili, da je v korenu drevesa na spodnji sliki operator $+$, ki ga v tem izrazu izračunamo kot zadnjega?)



Podatkovna struktura `Izraz` je že implementirana. Vsako vozlišče ima atribut `operator`, ki je lahko `'+'`, `'-'`, `'*'` ali `None`. Če je njegova vrednost `None`, predstavlja število in vozlišče ima še atribut `stevilo` (celo število). Če vrednost atributa `operator` ni `None`, ima vozlišče še atributa `levo` in `desno`, ki predstavljata levi in desni podizraz.

a) (10 točk) Sestavite metodo `naloga1a(self)`, ki izračuna in vrne vrednost tega aritmetičnega izraza. Primer (če `d` ustreza zgornji sliki):

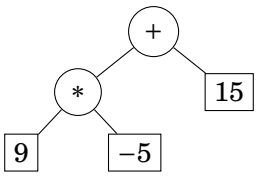
```
>>> d.naloga1a()
-30
```

b) (10 točk) Napišite metodo `naloga1b(self)`, ki vrne niz z običajnim zapisom tega izraza (glejte primer spodaj). Pred in za vsakim operatorjem naj bo po en presledek. Podizrazi naj bodo v oklepajih, razen kadar so le-ti števila. Sicer oklepajev ne smete opuščati (pa čeprav nam asociativnostni zakon to omogoča). Primer (če `d` ustreza zgornji sliki):

```
>>> d.naloga1b()
'(9 * (2 - 7)) + (5 * 3)'
```

c) (10 točk) Sestavite metodo `naloga1c(self)`, ki izraz poenostavi, tako da odpravi "najbolj notranje" operatorje, tj. tiste operatorje, kjer sta oba podizraza števili. Primer (če `d` ustreza zgornji sliki):

```
>>> d.naloga1c()
>>> d
Izraz('+', levo=Izraz('*', levo=Izraz(9), desno=Izraz(-5)), desno=Izraz(15))
```



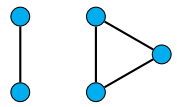
2. naloga (Risanje grafov s paketom TikZ, 15 + 20 točk)

Pri tej nalogi bomo spoznali, kako narišemo graf v LaTeXu z uporabo paketka TikZ. Risali bomo samo takšne slike, kjer bodo povezave *ravne črte*. Graf podamo z:

- slovarjem `vozlisca`, kjer so ključi oznake vozlišč, vrednosti pa njihove koordinate;
- seznamom povezav `povezave`.

Graf na spodnji sliki bi torej podali takole:

```
vozlisca_1 = {0: (0, 0), 1: (0, 1), 2: (1, 0), 3: (1, 1), 4: (1.87, 0.5)}
povezave_1 = [(0, 1), (2, 3), (2, 4), (3, 4)]
```



Ta slika je narisana v TikZ in sicer z naslednjimi ukazi:

```
\begin{tikzpicture}
\tikzstyle{every node} = [draw, thin, fill=cyan, circle, inner sep=2.5pt]
\tikzstyle{every path} = [draw, thick]
\node (v_0) at (0, 0) {};
\node (v_1) at (0, 1) {};
\node (v_2) at (1, 0) {};
\node (v_3) at (1, 1) {};
\node (v_4) at (1.87, 0.5) {};
\path (v_0) -- (v_1);
\path (v_2) -- (v_3);
\path (v_3) -- (v_4);
\path (v_2) -- (v_4);
\end{tikzpicture}
```

a) (15 točk) Sestavite funkcijo `tikz(vozlisca, povezave)`, ki kot argumenta dobi slovar in seznam, kot sta opisana zgoraj. Funkcija naj vrne niz, ki vsebuje TikZ kodo, ki nariše to sliko.

Oznake vozlišč bodo celoštevilske. Za vsako vozlišče mora biti v kodi po eno vrstica oblike

```
\node (v_Q) at (X, Y) {};
```

kjer je namesto `Q` oznaka vozlišča, namesto `X` in `Y` pa koordinati vozlišča. Vrstice naj bodo urejene glede na oznake vozlišč. Za vsako povezavo iz seznama `povezave` mora biti v kodi po ena vrstica oblike

```
\path (v_Q) -- (v_W);
```

kjer sta namesto `Q` in `W` oznaki vozlišč. Glava in noga sta fiksni (in sta tudi že definirani v preambuli). Primer (spremenljivki `vozlisca_1` in `povezave_1` sta podani zgoraj):

```
>>> tikz(vozlisca_1, povezave_1)
'\\begin{tikzpicture}\n\\tikzstyle{every node} = [draw, thin, fill=cyan, circle,
inner sep=2.5pt]\n\\tikzstyle{every path} = [draw, thick]\n\\node (v_0) at (0, 0)
 {};\n\\node (v_1) at (0, 1) {};\n\\node (v_2) at (1, 0) {};\n\\node (v_3) at (1,
 1) {};\n\\node (v_4) at (1.87, 0.5) {};\n\\path (v_0) -- (v_1);\n\\path (v_2) --
 (v_3);\n\\path (v_2) -- (v_4);\n\\path (v_3) -- (v_4);\n\\end{tikzpicture}'
```

b) (20 točk) Sestavite še funkcijo `preberi_graf(koda)`, ki kot argument dobi niz s TikZ kodo. Funkcija naj vrne slovar in seznam, kot sta definirana zgoraj. Primer (v spremenljivki `tikz_koda` naj bo niz iz primera zgoraj):

```
>>> preberi_graf(tikz_koda)
({0: (0.0, 0.0), 1: (0.0, 1.0), 2: (1.0, 0.0), 3: (1.0, 1.0), 4: (1.87, 0.5)},
 [(0, 1), (2, 3), (2, 4), (3, 4)])
```

Predpostavite lahko, da bo `koda` vedno lepo oblikovana, tako kot je na primeru zgoraj.

3. naloga (Igra življenja, 20 + 5 točk)

Pri igri življenja svet predstavimo s pravokotno matriko, katere elementi so števila iz množice $\{0,1\}$. Število 0 predstavlja *mrtvo* celico, število 1 pa predstavlja *živo* celico. Osem celic, ki obdajajo izbrano celico matrike, imenujemo *sosedje*. (Robne celice imajo manj kot 8 sesedov, če smo povsem natančni.) Igra življenja poteka v diskretnih časovnih korakih. Pravila so naslednja:

- Živa celica, ki ima manj kot 2 živa soseda, umre (osamljenost).
- Živa celica, ki ima več kot 3 žive sosede, umre (prenaseljenost).
- Živa celica, ki ima 2 ali 3 žive sosede, preživi.
- Mrtva celica, ki ima natanko 3 žive sosede, oživi (reprodukcija).

a) (20 točk) V *Mathematici* sestavite funkcijo `zivljenje[svet_]`, ki kot argument dobi matriko, ki predstavlja svet, in vrne novo matriko, ki predstavlja novi svet po enem koraku igre življenja. Primer:

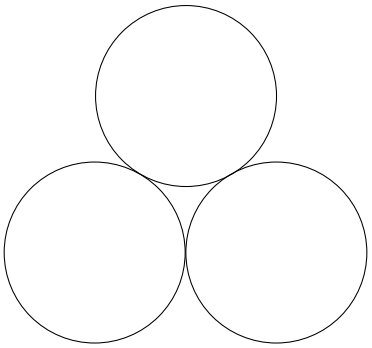
```
In[1]:= svet = {{0, 1, 0, 1}, {1, 0, 1, 0}, {1, 0, 0, 0}, {0, 0, 0, 1}};
In[2]:= zivljenje[svet]
Out[2]= {{0, 1, 1, 0}, {1, 0, 1, 0}, {0, 1, 0, 0}, {0, 0, 0, 0}}
```

b) (5 točk) Ugotovite, koliko korakov je potrebno narediti v igri življenja, da svet `dieHard` izumre. Svet `dieHard` je že definiran v zvezku. Lahko si pomagate tudi s funkcijo `Manipulate`, ki jo prav tako najdete v zvezku.

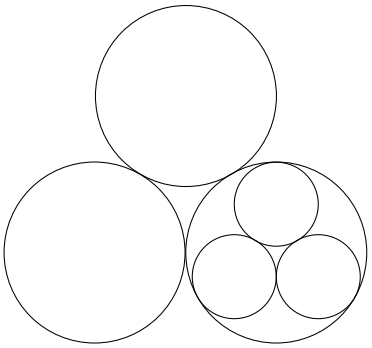
4. naloga (Krogi, 30 točk)

V *Mathematici* sestavite funkcijo `krogi[l_]`, ki nariše kroge, kakor je prikazano na slikah spodaj. Kot argument funkcija dobi poljubno dolg seznam celih števil `l`, ki sme vsebovati le cela števila 0, 1 in 2.

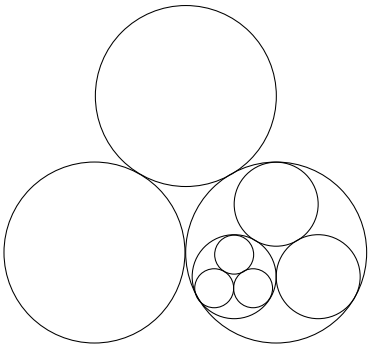
Osnovni vzorec, ki ga bomo imenovali *grozd*, je sestavljen iz treh enako velikih krogov, ki se medsebojno dotikajo (spodnja dva sta horizontalno poravnana, tretji pa je postavljen nad njima). Nato si izberemo enega od krogov in mu včrtamo grozd. V malem grozdu spet izberemo enega od krogov in mu včrtamo grozd itd. Kateri grozd je potrebno izbrati na vsakem koraku, nam povedo števila v seznamu `l`. Število 0 pomeni zgornjega, 1 pomeni spodnjega levega, 2 pa spodnjega desnega.



`krogi[{}]`



`krogi[{2}]`



`krogi[{2, 1}]`