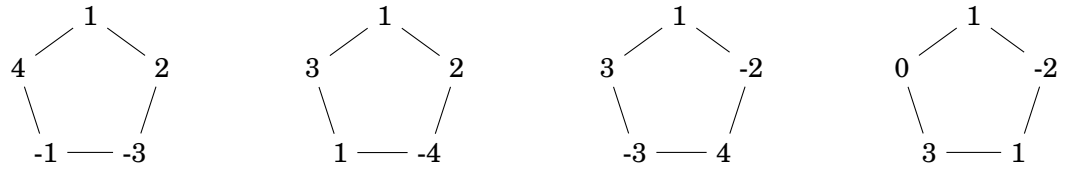
Programiranje I: 2. izpit

29. rožnika leta Gospodovega 2015

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

1. naloga (Igra v petkotniku, 20 + 10 točk)

Stanko se igra nenavadno igro. V oglišča petkotnika zapiše cela števila, katerih vsota je strogo pozitivna. Vsako potezo izbere neko negativno število t in ga spremeni v $-t$, številoma v obeh sosednjih ogliščih pa prišteje t. Pri tej operaciji se skupna vsota števil ohranja. Slika 1 prikazuje eno od možnih zaporedij treh potez.



(a) Začetni petkotnik (b) Stanje po 1. potezi (c) Stanje po 2. potezi (d) Stanje po 3. potezi

Slika 1: Potek igre v petkotniku

Stanko domneva, da se ta igra vselej konča. Nekajkrat je poskusil in vedno se je zgodilo, da je petkotnik po končno korakih ostal brez negativnih števil. Svojo hipotezo bi rad preveril še na nekaterih drugih začetnih podatkih. Ker je to zamudno opravilo, potrebuje računalniško podporo.

a) (20 točk) V *Mathematici* sestavite funkcijo `igra[l_]`, ki kot argument dobi začetno stanje igre, tj. seznam s petimi elementi. Elementi seznama so števila v ogliščih petkotnika, kot si sledijo v nasprotni smeri urinega kazalca. Prvi in zadnji element seznama sta tudi sosedna. (Predpostavite, da bodo elementi seznama vedno cela števila, njihova vsota pa bo strogo pozitivna.) Funkcija naj vrne seznam, ki vsebuje vsa stanja igre, od začetnega do končnega. Končno stanje je tisto, pri katerem ni več nobenega negativnega števila. *Funkcija naj operacijo vedno izvede na tistem negativnem številu, ki ima najmanjši indeks (tj. najbolj levo negativno število v seznamu).* Primer:

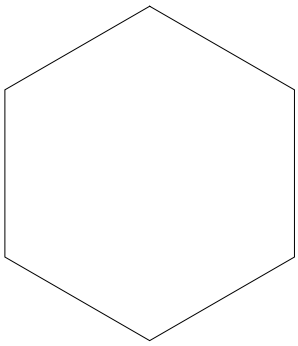
```
In[1]:= igra[{1, 4, -1, -3, 2}]
Out[1]= {{1, 4, -1, -3, 2}, {1, 3, 1, -4, 2}, {1, 3, -3, 4, -2},
         {1, 0, 3, 1, -2}, {-1, 0, 3, -1, 2}, {1, -1, 3, -1, 1},
         {0, 1, 2, -1, 1}, {0, 1, 1, 1, 0}}
```

b) (10 točk) Mojca trdi, da se v splošnem igra ne ustavi. Meni, da je imel Stanko pač srečo, ker funkcija `igra` operacijo vselej izvede na najbolj levem negativnem številu v seznamu. Stanko bi rad Mojco prepričal, da se moti. Strategijo izbiranja števil bo spremenil, tako da bo vedno izbral najmanjše število. Sestavite še funkcijo `igra2[l_]`, ki naj se od prejšnje funkcije razlikuje le v izbiri števila, na katerem izvede operacijo. Vedno naj izbere najmanjše število. Če je takih števil več, naj naključno izbere enega od njih. Primer:

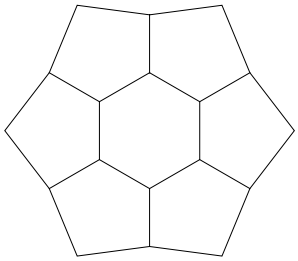
```
In[2]:= igra2[{1, 4, -1, -3, 2}]
Out[2]= {{1, 4, -1, -3, 2}, {1, 4, -4, 3, -1}, {1, 0, 4, -1, -1},
         {1, 0, 3, 1, -2}, {-1, 0, 3, -1, 2}, {-1, 0, 2, 1, 1},
         {1, -1, 2, 1, 0}, {0, 1, 1, 1, 0}}

In[3]:= igra2[{1, 4, -1, -3, 2}]
Out[3]= {{1, 4, -1, -3, 2}, {1, 4, -4, 3, -1}, {1, 0, 4, -1, -1},
         {0, 0, 4, -2, 1}, {0, 0, 2, 2, -1}, {-1, 0, 2, 1, 1},
         {1, -1, 2, 1, 0}, {0, 1, 1, 1, 0}}
```

2. naloga (Nanocevke, 30 točk)

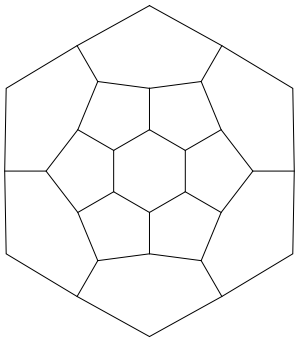
V *Mathematici* sestavite funkcijo `nanocevka[n_]`, ki nariše "nanocevko" dolžine n, kot vidite na primerih:



`nanocevka[0]`



`nanocevka[1]`



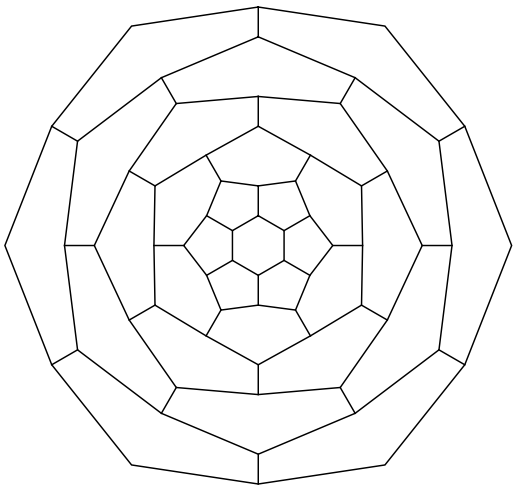
`nanocevka[2]`

Nanocevka dolžine 0 je pravilni šestkotnik (glejte sliko na levi), katerega oglišča ležijo na krožnici s polmerom 2. Najvišje ležeče oglišče ima koordinati $(0,2)$.

Nanocevko dolžine 1 dobimo tako, da nanocevki dolžine 0 dodamo dvanajstkotnik. Njegova oglišča naj ležijo izmenično na krožnicah s polmeroma 4 in 5. Tista oglišča dvanajstkotnika, ki ležijo na manjši od obeh krožnic, povežemo s pripadajočimi oglišči šestkotnika (kot kaže slika na sredini).

Nanocevko dolžine 2 dobimo tako, da nanocevki dolžine 1 dodamo dvanajstkotnik. Njegova oglišča naj ležijo izmenično na krožnicah s polmeroma 7 in 8. Oglišča novonastalega dvanajstkotnika, ki ležijo na manjši od obeh krožnic, povežemo s tistimi oglišči prejšnjega dvanajstkotnika, ki ležijo na večji od obeh krožnic (kot kaže slika na desni).

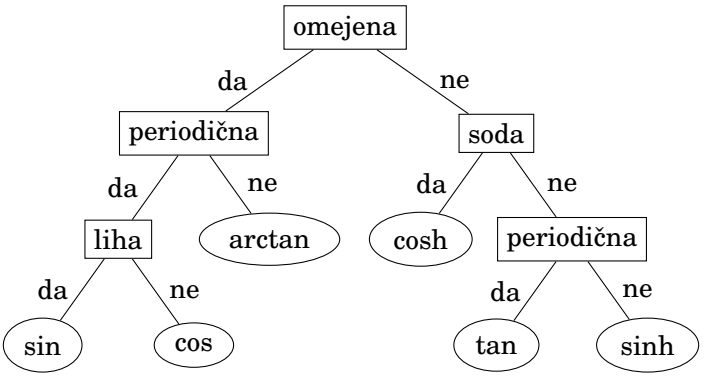
Nanocevko dolžine $n \geq 2$ dobimo tako, da nanocevki dolžine $n-1$ dodamo dvanajstkotnik. Njegova oglišča naj ležijo izmenično na krožnicah s polmeroma $3n+1$ in $3n+2$. Oglišča novonastalega dvanajstkotnika, ki ležijo na manjši od obeh krožnic, povežemo s tistimi oglišči prejšnjega dvanajstkotnika, ki ležijo na večji od obeh krožnic.



`nanocevka[5]`

3. naloga (Odločitvena drevesa, 10 + 10 + 10 točk)

Dan je razred `Drevo`, ki predstavlja *odločitveno drevo*. Primer takšnega drevesa je na spodnji sliki:



V našem primeru smo klasificirali funkcije sin, cos, tan, arctan, sinh in cosh glede na njihove lastnosti (omejenost, periodičnost ipd.). Tem lastnostim bomo rekli *značilnosti*. Omejili se bomo na takšne značilnosti, ki imajo le dve možni vrednosti: `True` in `False`. Vozlišča drevesa so lahko *notranja* ali pa *listi* (atribut `list`).

Če je vozlišče notranje (`list` = `False`), ima vozlišče še atribut `znacilnost`, tj. niz z oznako značilnosti, ter atributa `da` in `ne`, ki predstavljata usrezni poddrevesi. Če je vozlišče list, ima še atribut `odgovor`, ki vsebuje ime objekta z ustreznimi značilnostmi (v našem primeru ime funkcije), lahko pa ima tudi vrednost `None`.

Konstruktor je že implementiran. Zgornje drevo sestavimo takole:

```
d = Drevo('omejena', da=Drevo('periodična', da=Drevo('liha', da=Drevo('sin'),
ne=Drevo('cos')), ne=Drevo('arctan')), ne=Drevo('soda', da=Drevo('cosh'),
ne=Drevo('periodična', da=Drevo('tan'), ne=Drevo('sinh'))))
```

a) (10 točk) Sestavite metodo `najdi(self, slovar)`, ki v drevesu poišče in vrne objekt z danimi značilnostmi, ki so shranjene v slovarju `slovar`. Če objekta ne moremo določiti (ker v slovarju manjka ustrezna značilnost), naj metoda vrne `None`. Primer (če `d` ustreza zgornjemu drevesu):

```
>>> d.najdi({'omejena': True, 'periodična': True, 'liha': True, 'lepa': True})
'sin'
>>> d.najdi({'omejena': False, 'periodična': True, 'soda': True})
'cosh'
>>> print(d.najdi({'periodična': True, 'soda': False, 'liha': True}))
None
```

b) (10 točk) Sestavite metodo `objekti_znacilnosti(self)`, ki vrne urejeni par dveh množic. Prva naj vsebuje vse objekte, ki se nahajajo v drevesu, druga pa oznake vseh uporabljenih značilnosti. Primer:

```
>>> d.objekti_znacilnosti()
({'sin', 'sinh', 'arctan', 'cosh', 'cos', 'tan'},
 {'periodična', 'soda', 'omejena', 'liha'})
```

c) (10 točk) Sestavite metodo `znacilnosti(self, objekt)`, ki v drevesu poišče podani objekt in vrne slovar njegovih značilnosti. Če objekta ni v drevesu, naj vrne `None`. Drevo bo vedno takšno, da se noben objekt ne bo pojavil več kot enkrat. Primer:

```
>>> d.znacilnosti('cosh')
{'soda': True, 'omejena': False}
>>> print(d.znacilnosti('log'))
None
```

4. naloga (Nginx, 15 + 15 točk)

Nginx je odprtokodni spletni strežnik, ki je na drugem mestu glede na priljubljenost (na prvem mestu je Apache). Tako kot vsak drug spletni strežnik tudi Nginx vodi dnevnik dostopa (angl. access log). Ko uporabnik obišče neko spletno stran, brskalnik pošlje strežniku zahtevek. Strežnik se na zahtevek ustrezno odzove (po navadi tako, da posreduje vsebino spletne strani). Vsak tak dogodek strežnik zabeleži in sicer zapiše v dnevnik eno vrstico besedila oblike:

```
$remote_addr - $remote_user [$time_local] "$request" $status $body_bytes_sent
"$http_referer" "$http_user_agent"
```

Zgled (iz dnevnika strani `http://upm.putka.si`):

```
84.20.241.170 - - [15/Nov/2014:10:31:15 +0000] "GET /tasks/2014/ HTTP/1.1" 200
1599 "http://putka.upm.si/tasks/" "Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.111 Safari/537.36"
```

Iz zapisa se da razbrati, da je uporabnik z IP naslovom 84.20.241.170 dne 15. 11. 2014 ob 10:31:15 zahteval vsebino strani `/tasks/2014/` (oz. `http://putka.upm.si/tasks/2014/`). Strežnik je posredoval 1599 bajtov dolg odgovor s statusno kodo 200 (vse OK). Da se še razbrati, da je do te strani prišel preko povezave na `http://putka.upm.si/tasks/` in da je uporabljal brskalnik Chrome na operacijskem sistemu Windows 7.

Še en zgled:

```
74.208.16.114 - - [15/Nov/2014:10:44:45 +0000] "POST /wp-login.php HTTP/1.1" 404
1650 "http://www.google.com/" "Opera/9.80 (Windows NT 6.1; WOW64)
Presto/2.12.388 Version/12.14"
```

Iz zapisa se da razbrati, da se je nepridiprav iz ZDA poskušal prijaviti v spletno stran, pri čemer je predpostavil, da gre za WordPress blog. Ker pa v tem primeru ne gre za WordPress blog (datoteka `wp-login.php` na strežniku ne obstaja), se je strežnik odzval s statusno kodo 404 (strani ni mogoče najti).

a) (15 točk) Napišite funkcijo `beri_log(ime_datoteke)`, ki kot argument dobi niz z imenom datoteke in vrne seznam naborov, kjer vsak nabor ustreza eni vrstici dnevnika. Vsak nabor naj vsebuje 8 elementov, pri čemer naj bo:

- IP naslov tipa `IPv4Address` (iz modula `ipaddress`);
- časovna oznaka naj bo tipa `datetime` (iz modula `datetime`);
- statusna koda in velikost naj bosta celi števili (tip `int`);
- ostali podatki pa naj bodo nizi.

Primer (če datoteka access.log vsebuje zgornji dve vrstici):

```
>>> beri_log('access.log')
[(IPv4Address('84.20.241.170'), '-', datetime.datetime(2014, 11, 15, 10, 31, 15,
tzinfo=datetime.timezone.utc), 'GET /tasks/2014/ HTTP/1.1', 200, 1599,
'http://putka.upm.si/tasks/', 'Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.111 Safari/537.36'),
(IPv4Address('74.208.16.114'), '-', datetime.datetime(2014, 11, 15, 10, 44, 45,
tzinfo=datetime.timezone.utc), 'POST /wp-login.php HTTP/1.1', 404, 1650,
'http://www.google.com/', 'Opera/9.80 (Windows NT 6.1; WOW64) Presto/2.12.388
Version/12.14')]
```

Nasvet: `datetime.datetime.strptime(casovna_oznaka, '%d/%b/%Y:%H:%M:%S %z')`

b) (15 točk) Napišite funkcijo `statistika(seznam)`, ki kot argument dobi seznam naborov, kot ga vrne prejšnja funkcija. Prešteje naj število zahtevkov glede na IP in statusno kodo. Vrne naj slovar, kjer so ključi IP naslovi, vrednosti pa slovarji, ki imajo za ključe statusne kode, pripadajoče vrednosti pa so frekvence vrstic s to kombinacijo IP naslova in statusne kode. Primer:

```
>>> statistika(beri_log('access.log'))
{ipaddress.IPv4Address('74.208.16.114'): {404: 1},
 ipaddress.IPv4Address('84.20.241.170'): {200: 1}}
```